ESCUELA SUPERIOR POLITÉCNICA DEL EJÉRCITO

ESPE – LATACUNGA

CARRERA DE INGENIERÍA ELECTRÓNICA E INSTRUMENTACIÓN

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA ELECTRÓNICA E INSTRUMENTACIÓN

DISEÑO E IMPLEMENTACION DE UN SISTEMA DE ADQUISICION DE DATOS PARAI NSTRUMENTACION VIRTUAL UTILIZANDO UN MICROCONTRLODADOR PIC

ELABORADO POR:

AMPARO MEYTHALER NARANJO

Latacunga-Agosto

2005

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por la Sra. Ing. Amparo Meythaler Naranjo, bajo nuestra supervisión.

Ing. Eddie Galarza
DIRECTOR DE TESIS

Ing. Marco Singaña
CODIRECTOR DE TESIS

AGRADECIMIENTO

Mi sincero agradecimiento a todos quienes contribuyeron de una u otra forma a la culminación de mi carrera y de este trabajo, en especial a los Sres. Ings. Eddie Galarza y Marco Singaña por su ayuda, colaboración y aporte desinteresado.

Amparo Meythaler N.

DEDICATORIA

Todo mi esfuerzo y en especial este trabajo, está dedicado a mis alumnos, mis tres hermosos hijos y en especial a mi esposo por su cariño, paciencia y ayuda incondicional.

Amparo Meythaler N.

ÍNDICE

PAG.

CAP. II SOFTWARE DE ADQUISICIÓN DE DATOS

CAP. III MICROCONTROLADOR PIC16F877

CAP. IV DISEÑO Y CONSTRUCCIÓN

CAP. V ANÁLISIS DE RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

INTRODUCCIÓN

Uno de los objetivos de todo sistema de Instrumentación es el análisis de los datos adquiridos del proceso y a su vez la respuesta que se emita como consecuencia de dicho análisis; bajo esta perspectiva es imprescindible que la adquisición de datos sea eficiente.

Para cumplir el cometido expuesto, en el mercado existen muchas tarjetas de adquisición de datos, unas genéricas y otras dedicadas tanto en hardware como en software, pero todas ellas a precios elevados; entonces, se vuelve importante brindar una alternativa más barata y que esté a alcance de todos para la adquisición de datos, como es la de utilizar un microcontrolador PIC muy inferior en precio que las tarjetas y que puede realizar esta tarea adecuadamente.

De acuerdo a lo anterior, se realiza el diseño e implementación de una tarjeta de adquisición de datos que pueda recibir datos analógicos y que sea controlada por un software diferente a los tradicionales. Se condiciona a que la adquisición de los datos sea por el puerto serie, situación común en procesos reales, para lo cual se deben configurar parámetros como son: velocidad, longitud y paridad.

Como un objetivo adicional, se plantea el ejemplificar el trabajo que realiza un protocolo de comunicación, para este efecto se utiliza el código ASCII en la estandarización de la comunicación entre el microcontrolador y el computador que servirá de maestro.

El software Matlab es el medio manipulador directo del puerto serie y el generador de la interfase amigable. En la interfase diseñada, a más de escoger los parámetros antes mencionados, se puede seleccionar de entre dos microcontroladores con cuál se desea trabajar, situación que ejemplifica el direccionamiento que debe existir cuando se trabaja entre un maestro y sus esclavos.

El microcontrolador empleado es el PIC16F877 el mismo que cuenta con un ADC capaz de recibir hasta 8 entradas análogas multiplexadas, pero en la tarjeta diseñada se utilizarán sólo 6 ya que por dos de los pines de cada microcontrolador se ingresarán los voltajes de referencia. La adquisición de los datos se realiza de una sola entrada a la vez, ya que a más de estar multipexadas se debe recordar que la comunicación es del tipo serie.

Para lograr el cometido antes expuesto se requiere de la revisión de ciertos lineamientos teóricos que sustenten adecuadamente el diseño; bajo ésta óptica el fundamento teórico se ha dividido en los tres capítulos que se mencionan a continuación.

En el capítulo I se revisan los principios relacionados con la adquisición de datos, los protocolos, la interfase RS232, entre otros; el capítulo II se centra en el trabajo con el puerto serie y las interfaces gráficas usando Matlab, para finalmente en el capítulo III revisar el microcontrolador PIC16F877 poniendo énfasis en el trabajo del conversor análogo – digital y la comunicación serie que posee.

El capítulo IV expone las consideraciones de diseño y detalla el hardware de la tarjeta y el software realizado tanto para los microcontroladores como para el Matlab, en este punto se revisan las interfases y los lineamientos de la comunicación serie.

En el capítulo V se realiza un análisis de los resultados obtenidos, para culminar con las conclusiones y recomendaciones que arrojó el desarrolló del proyecto.

Se incluye también en anexos el software de uno de los microcontroladores y una breve descripción de la forma de emplear la aplicación a manera de manual de uso.

CAPÍTULO I

PRINCIPIOS DE ADQUISICIÓN DE DATOS

1.1 SISTEMA DE ADQUISICIÓN DE DATOS

"Un sistema de adquisición de datos es un equipo que permite tomar señales físicas del entorno y convertirlas en datos que posteriormente se podrán procesar y presentar. A veces el sistema de adquisición es parte de un sistema de control, y por tanto la información recibida se procesa para obtener una serie de señales de control. En el diagrama de la figura 1.1 se puede ver los bloques que componen un típico sistema de adquisición de datos"[1]:

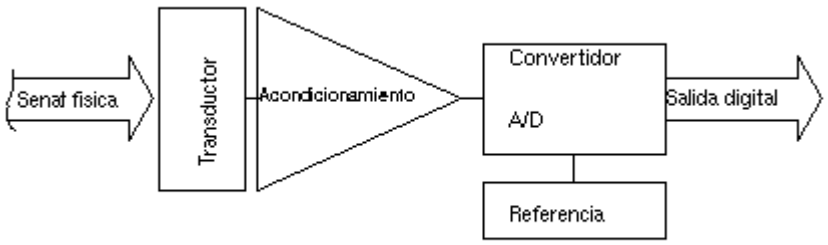


Figura 1.1 Diagrama de Bloques de un sistema de Adquisición de Datos

Como puede verse, los bloques principales son:

- El transductor.
- El acondicionamiento de señal.
- El convertidor analógico-digital.

---

[1]http://www.redeya.com

- La etapa de salida (interfaz con la lógica).

El **transductor** es el elemento que convierte la magnitud física que va a medirse en una señal de salida (normalmente tensión o corriente) que puede ser procesada por el sistema. Salvo que la señal de entrada sea eléctrica, puede decirse que el transductor es un elemento que convierte energía de un tipo en otro. Por tanto, el transductor debe tomar poca energía del sistema bajo observación, para no alterar la medida.

El **acondicionamiento de señal** es la etapa encargada de filtrar y adaptar la señal proveniente del transductor a la entrada del convertidor analógico/digital. Esta adaptación suele ser doble y se encarga de:

- Adaptar el rango de salida del transductor al rango de entrada del convertidor. (Normalmente en tensión).
- Acoplar la impedancia de salida de uno con la impedancia de entrada del otro.

La adaptación entre los rangos de salida del convertidor y el de entrada del convertidor tiene como objetivo el aprovechar el margen dinámico del convertidor, de modo que la máxima señal de entrada debe coincidir con la máxima del convertidor (pero no con la máxima tensión admisible, ya que para ésta entran en funcionamiento las redes de protección que el convertidor lleva integrada).

Por otro lado, la adaptación de impedancias es imprescindible ya que los transductores presentan una salida de alta impedancia, que normalmente no puede excitar la entrada de un convertidor, cuya impedancia típica suele estar entre 1 y 10 kΩ.

El **convertidor Analógico / Digital** es un sistema que presenta en su salida una señal digital a partir de una señal analógica de entrada, (normalmente de tensión) realizando las funciones de cuantificación y codificación.

La cuantificación implica la división del rango continuo de entrada en una serie de pasos; de modo que para infinitos valores de la entrada, la salida sólo puede presentar una serie determinada de valores. Por tanto la cuantificación implica una pérdida de información que no puede olvidarse.

La codificación es el paso por el cual la señal digital se ofrece según un determinado código binario, de modo que las etapas posteriores al convertidor puedan leer estos datos adecuadamente. Este paso hay que tenerlo siempre en cuenta, ya que puede hacer que obtengamos datos erróneos, sobre todo cuando el sistema admite señales positivas y negativas con respecto a masa, momento en el cual la salida binaria del

convertidor presenta tanto la magnitud como el signo de la tensión que ha sido medida.

La **etapa de salida** es el conjunto de elementos que permiten conectar el sistema de adquisición de datos con el resto del equipo, y puede ser desde una serie de buffers digitales incluidos en el circuito convertidor, hasta un interfaz RS 232, RS 485 o Ethernet para conectar a un ordenador o estación de trabajo, en el caso de sistemas de adquisición de datos comerciales.

1.2 ACONDICIONAMIENTO DE LA SEÑAL

"En el sistema de acondicionamiento se puede encontrar estas etapas, aunque no todas están siempre presentes:

- Amplificación
- Excitación
- Filtrado
- Multiplexado
- Aislamiento
- Linealización

1.2.1 AMPLIFICACIÓN

Es el tipo más común de acondicionamiento. Para conseguir la mayor precisión posible la señal de entrada deber ser amplificada de modo que su máximo nivel coincida con la máxima tensión que el convertidor pueda leer.

1.2.2. AISLAMIENTO

Otra aplicación habitual en un acondicionamiento de la señal es el aislamiento eléctrico entre el transductor y el ordenador, para proteger al mismo de transitorios de alta tensión que puedan dañarlo. Un motivo adicional para usar aislamiento es el garantizar que las lecturas del convertidor no sean afectadas por diferencias en el potencial de masa o por tensiones en modo común.

Cuando el sistema de adquisición y la señal a medir están ambas referidas a masa pueden aparecer problemas si hay una diferencia de potencial entre ambas masas, apareciendo un "bucle de masa", que puede devolver resultados erróneos.

1.2.3 MULTIPLEXADO

El multiplexado es la conmutación de las entradas del convertidor, de modo que con un sólo convertidor pueden medirse los datos de diferentes canales de entrada.

Puesto que el mismo convertidor está midiendo diferentes canales, su frecuencia máxima de conversión será la original dividida por el número de canales muestreados.

1.2.4 FILTRADO

El fin del filtro es eliminar las señales no deseadas de la señal que se obtiene. Las señales alternas como la vibración, necesitan un tipo distinto de filtro, conocido como filtro antialiasing, que es un filtro pasabajo pero con un corte muy brusco, que elimina totalmente las señales de mayor frecuencia que la máxima a medir, ya que si no se eliminasen, aparecerían superpuestas a la señal medida, con el consiguiente error.

1.2.5 EXCITACIÓN

La etapa de acondicionamiento de señal a veces genera excitación para algunos transductores, como por ejemplos las galgas extensométricas, termistores o RTD, que necesitan de la misma, bien por su constitución interna, (como el termistor, que es una resistencia variable con la temperatura) o bien por la configuración en que se conectan (como el caso de las galgas, que se suelen montar en un puente de Wheatstone).

1.2.6 LINEALIZACIÓN

Muchos transductores, como los termopares, presentan una respuesta no lineal ante cambios lineales en los parámetros que están siendo medidos. Aunque la linealización puede realizarse mediante métodos numéricos en el sistema de adquisición de datos, suele ser una buena idea el hacer esta corrección mediante circuitería externa"[2].

1.3 CARACTERÍSTICAS BÁSICAS DE UN CONVERTIDOR A/D

Las características esenciales de un convertidor análogo – digital son:

- Impedancia de entrada
- Rango de entrada
- Número de bits
- Resolución
- Tensión de fondo de escala
- Tiempo de conversión
- Error de conversión

El Número de bits se refiere al número de bits que tiene la palabra de salida del convertidor, y por tanto es el número de pasos que admite el convertidor.

[2]http://www.redeya.com

La Resolución es el mínimo valor que puede distinguir el convertidor en su entrada analógica, o dicho de otro modo, la mínima variación, Vi, en el voltaje de entrada que se necesita para cambiar en un bit la salida digital. Se presenta la siguiente relación:

$$Vi = \frac{Vfe}{(2^n - 1)} \qquad \textbf{(Ec. 1.1)}$$

donde:

Vi es el voltaje de entrada.

n es el número de bits del convertidor.

Vfe la tensión de fondo de escala.

La Tensión de Fondo de escala es aquella para la que la salida digital es máxima, este valor depende del tipo de convertidor, pero normalmente la fija el usuario, en forma de una tensión de referencia externa

El Tiempo de conversión, es el tiempo que tarda en realizar una medida el convertidor en concreto, y dependerá de la tecnología de medida empleada. Evidentemente da una cota máxima de la frecuencia de la señal a medir, este tiempo se

mide como el transcurrido desde que el convertidor recibe una señal de inicio de conversión (normalmente llamada SOC, Start of Conversión) hasta que en la salida aparece un dato válido. Para tener constancia de un dato válido se tienen dos caminos:

- Esperar el tiempo de conversión máximo que aparece en la hoja de características.
- Esperar a que el convertidor envíe una señal de fin de conversión.

Si no se respeta el tiempo de conversión, en la salida se tendrá un valor, que dependiendo de la constitución del convertidor será:

- Un valor aleatorio, como consecuencia de la conversión en curso.
- El resultado de la última conversión.

1.4 MUESTREO DE LA SEÑAL

El muestreo de la señal implica pérdida de información respecto a la señal de entrada, ya que de un número infinito de valores posibles para la entrada sólo se tiene un valor finito de valores posibles para la salida. Por tanto es fundamental saber cuántas muestras se deben tomar.

La respuesta a esta pregunta depende del error medio admisible, del método de reconstrucción de la señal (si es que se usa) y del uso final de los datos de la conversión. Independientemente del uso final, el error total de las muestras

será igual al error total del sistema de adquisición y conversión más los errores añadidos por el ordenador o cualquier sistema digital.

Para dispositivos incrementales, tales como motores paso a paso y conmutadores, el error medio de los datos muestreados no es tan importante como para los dispositivos que requieren señales de control continuas. De cualquier modo, la precisión instantánea en cada muestra es igual a la precisión del sistema de adquisición y conversión, y en muchas aplicaciones esto puede ser más que suficiente.

La precisión media de los datos muestreados puede mejorarse con estos métodos:

- Aumentar el número de muestras por ciclo.
- Filtrado previo al multiplexado.
- Filtrar la salida del convertidor digital / analógico.

1.4.1 EL TEOREMA DE NYQUIST O TEOREMA DE MUESTREO

"El objetivo fundamental de la adquisición es el poder reconstruir la señal muestreada de una manera fiel. Este teorema dice que la frecuencia mínima de muestreo para poder reconstruir la señal ha de ser el doble de la frecuencia de la señal a medir. Pero, para que la

reconstrucción sea fiable, se debe tomar muestras a una frecuencia unas 10 veces superior a la de la señal a evaluar.

En la figura 1.2 se observa una señal sinusoidal, que es muestreada con dos medidas por ciclo y su reconstrucción mediante los dos métodos que más se usan (reconstrucción de orden cero y reconstrucción de orden uno)

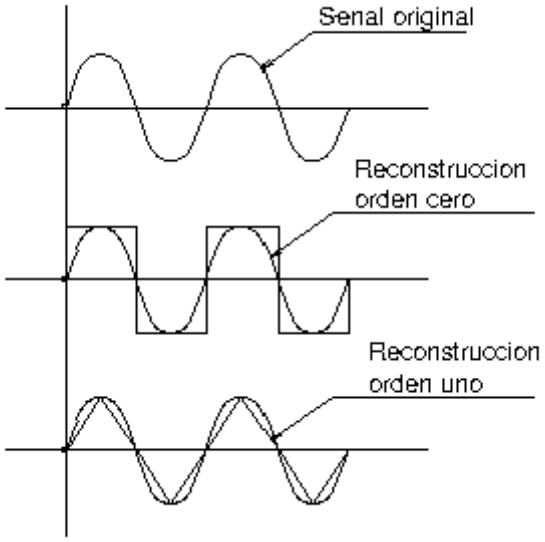


Figura 1.2 Reconstrucción de una Señal Sinusoidal

Como se evidencia, aplicando el teorema de Nyquist se puede saber al menos la frecuencia de la señal medida, aunque no su tipo, ni si el muestreo es eficaz o no. La reconstrucción de orden cero es la salida directa de un convertidor analógico digital, mientras que la de orden uno es la interpolación simple mediante rectas, de modo que la señal se aproxima más a la original.

1.4.2 EFECTO ALIASING

El Aliasing se produce cuando la frecuencia de muestreo es menor que la de la señal que se muestrea, y se refiere al hecho de que se puede interpretar de una manera no exacta la señal, apareciendo un "alias" de la señal (de ahí el término). Este efecto se pone de manifiesto en la figura 1.3:

Figura 1.3 Efecto del Aliasing

Como se aprecia, al tomar varias muestras con un periodo de muestreo superior al de la señal medida, se llega a creer que la señal tiene una frecuencia mucho menor de la que realmente tiene"[3].

En este efecto también influyen los armónicos, señales que interfieran con la señal a medir, de modo que pueden aparecer señales de alta frecuencia

[3] http://www.redeya.com

superpuestas como ruido y otras senoidales, que aparentemente no son ruido, pero que también afectan a la señal bajo medida. Por tanto, cualquier frecuencia de

muestreo excesivamente baja da información falsa sobre la señal.

1.5 OTROS CONCEPTOS NECESARIOS PARA LA ADQUISICIÓN DE SEÑALES

1.5.1 ESTABILIDAD DE LA TENSIÓN DE REFERENCIA

"Los convertidores usan varios métodos para digitalizar la señal, pero siempre respecto a una tensión de referencia. En los casos en los que la señal de referencia sea externa se debe tener en cuenta estas ideas:

- Usar un elemento que de una tensión con poca deriva térmica.
- Adecuar la impedancia de salida de la referencia a la impedancia de entrada del convertidor.
- Filtrar adecuadamente la salida de la referencia, así como la tensión de alimentación que se le aplica.

1.5.2 FILTRADO DE LAS LÍNEAS DE ALIMENTACIÓN

Es imprescindible que las líneas de alimentación estén debidamente desacopladas con el uso de condensadores. Además del típico condensador electrolítico que es adecuado para atenuar las fluctuaciones de la

alimentación debidas al rizado de red, es imprescindible añadir condensadores cerámicos de unos 100 nF próximos al convertidor para evitar los transitorios de alta frecuencia.

1.5.3 TRAZADO ADECUADO Y SEPARADO DE LA ALIMENTACIÓN ANALÓGICA Y DIGITAL

Este aspecto, que muchas veces no se tiene en cuenta, es fundamental y puede llegar a causar muchos problemas, sobre todo cuando se miden tensiones del orden de uno o dos mV. El problema se debe a que los conductores de alimentación tienen una resistencia no nula, y si se dispone de un microcontrolador, por ejemplo, trabajando a 4 Mhz, aparecerán en la alimentación picos de intensidad de la misma frecuencia.

Estos picos generarán caídas de tensión al circular por las pistas de la placa, y estas tensiones harán que el nivel de masa fluctúe, con el consiguiente efecto en la circuitería analógica. En resumen, se puede recomendar la observación de estos puntos:

- Las pistas de masa han de ser anchas y ocupar la mayor extensión posible (planos de masa).
- Debe haber dos planos de masa separados, uno para los circuitos digitales y otro para los analógicos.

- Los planos de masa deben conectarse en un sólo punto, que habitualmente es la masa del conector de alimentación.
- Si es posible, usar dos reguladores separados para cada uno de los bloques (analógico y digital).
- Tanto si se usa un regulador, como si se usan dos es necesario dividir las líneas de alimentación del mismo modo que las de masa, esto es, con una conexión en estrella"[4].

1.6 SOBREMUESTREO

Dentro de las distintas técnicas de conversión de señales, el sobremuestreo (oversampling) se ha hecho popular en los últimos años debido a que evita muchos de los inconvenientes encontrados en los métodos convencionales de conversión digital - analógica (DAC) y analógica - digital (ADC), especialmente en aquellas aplicaciones que requieren alta resolución de representación a baja frecuencia de las señales.

Los convertidores convencionales tienen dificultades a la hora de ser implementados en tecnología VLSI (Very Large Scale Integration), estas dificultades son debidas a que los métodos convencionales precisan componentes analógicos en sus filtros y circuitos de conversión que pueden ser muy vulnerables al ruido y a las interferencias, sin embargo estos métodos precisan una velocidad de muestreo mucho menor, la frecuencia de Nyquist de la señal.

1.7 TÉCNICAS DE CODIFICACIÓN

”La información, para ser transmitida, necesita ser adaptada al medio de transmisión; para ello, generalmente, será preciso codificarla de tal forma que pueda asegurarse una recepción adecuada y segura.

[4]http://www.burr-brown.com

Si se tiene la información en un determinado alfabeto fuente y se desea transformarla en otro alfabeto destino, se puede definir codificación como a la realización de dicha transformación, siendo el código la correspondencia existente entre cada símbolo del alfabeto fuente y cada conjunto de símbolos del alfabeto destino”[5].

Existen muchas formas de codificar la información, a continuación se detallan algunas de ellas.

1.7.1 CÓDIGO BAUDOT

Se trata de un código de 5 bits capaz de representar hasta 32 caracteres distintos, pero tiene además dos de ellos que permiten conmutar entre dos grupos denominados letras y figuras. El grupo de letras contiene el abecedario completo de mayúsculas de la A a la Z, mientras que el grupo de figuras contiene las cifras del 0 al 9, los signos de puntuación y caracteres especiales hasta un total de 26. La tabla 1.1 muestra el código Baudot.

1.7.2 CÓDIGO ASCII

Después del código Baudot se hizo necesario aumentar el conjunto de caracteres, apareciendo códigos de 6 bits, capaces de manejar 64 símbolos distintos (el código FIELDATA es un ejemplo de ellos).

[5]http://www.itlp.edu.mx/publica/tutoriales/telepro/t2_3.htm

Tabla 1.1 Código Baudot

Número de orden	Grupo de bits	Grupo de letras	Grupo de figuras
01	00011	A	-
02	11001	B	?
03	01110	C	:
04	01001	D	$
05	00001	E	3
06	01101	F	!
07	11010	G	&
08	10100	H	#
09	00110	I	8
10	01011	J	‘ o timbre
11	01111	K	(
12	10010	L	)
13	11100	M	.
14	01100	N	,
15	11000	O	9
16	10110	P	0
17	10111	Q	1
18	01010	R	4
19	00101	S	timbre o ’
20	10000	T	5
21	00111	U	7
22	11110	V	; o =
23	10011	W	2
24	11101	X	/
25	10101	Y	6
26	10001	Z	+ o ”
27	01000	CR	retorno de carro
28	00010	LF	avance de línea
29	11111	LTRS	cambio a letras
30	11011	FIGS	cambio a figuras
31	00100	SP	espacio
32	00000	BLK	blanco

Más tarde se pasó a códigos de 7 bits capaces de manejar hasta 128 caracteres, entre los cuales está el código ASCII (*American Standard Code for Information Interchange*), de 7 bits, también denominado código número 5 de CCITT.

A continuación, las tablas 1.2 y 1.3 presentan los valores codificados en ASCII de 7 bits.

Tabla 1.2 Código ASCII de 7 bits

bits	654							
3210	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P		p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Tabla 1.3 Caracteres de Control del Código ASCII de 7 bits

NUL	Null (carácter nulo)	DC1	Device control 1 (control de dispositivo 1)
SOH	Start of heading (comienzo de cabecera)	DC2	Device control 2 (control de dispositivo 2)
STX	Start of text (comienzo de texto)	DC3	Device control 3 (control de dispositivo 3)
ETX	End of text (final de texto)	DC4	Device control 4 (control de dispositivo 4)
EOT	End of transmission (fin de transmisión)	NAK	Negative acknowledge (trans. negativa)
ENQ	Enquiry (petición de transmisión)	SYN	Synchronous idle (espera síncrona)
ACK	Acknowledge (reconocimiento de trans.)	ETB	End of transmission block (fin bloque de transmisión)
BEL	Bell (señal audible, timbre o alarma)		
BS	Backspace (retroceso)	CAN	Cancel (cancelar)
HT	Horizontal tabulation (tab. horizontal)	EM	End of medium (fin del medio)
LF	Line feed (avance de página)	SUB	Substitute (sustitución)
VT	Vertical tabulation (tab. vertical)	ESC	Escape (escape)
FF	Form feed (avance de página)	FS	File separator (separador de archivos)
CR	Carriage return (retorno de carro)	GS	Group separator (separador de grupos)
SO	Shift out (quitar desplazador de bits)	RS	Record separator (separador de registros)
SI	Shift in (poner desplazador de bits)	US	Unit separator (separador de unidades)
DLE	Data link escape (escape enlace de datos)	DEL	Delete (borrar)

En la actualidad se utiliza el código ASCII de 8 bits, en que aparecen los 128 caracteres del código anterior más otros 128 caracteres, donde cada fabricante puede hacer su propia ampliación del conjunto de caracteres a manejar. En las transmisiones entre equipos diversos no es recomendable la utilización de caracteres de estas ampliaciones por los posibles errores de interpretaciones incorrectas que pudieran producirse.

1.7.3 CÓDIGOS DETECTORES DE ERRORES

No existe ningún sistema de comunicación de datos que pueda impedir que ocurran errores durante la transmisión, aunque la mayoría de estos pueden detectarse mediante diseños apropiados que permiten saber si la información recibida es la misma que se transmitió originalmente.

Entre las técnicas para la detección de errores se tiene las siguientes:

a. Técnica del Eco

Es una forma simple de detección de errores usada en situaciones interactivas. Cuando una estación recibe una transmisión, la almacena y retransmite de nuevo a la estación emisora (eco), ésta compara el eco con el mensaje original y de esta forma se puede determinar si se presentó un error y corregirlo.

Esta técnica tiene la desventaja de requerir al menos el doble de transmisiones, y además está la posibilidad de una "corrección" espontánea durante la retransmisión.

b. Técnicas de Detección Automática de Errores

Estas técnicas consisten en la adición al dato por enviar de un marco de verificación de secuencia o FCS (frame check sequence), el cual es obtenido a partir de los datos a transmitir por medio de un algoritmo. Una vez recibido el mensaje, la estación receptora aplica el mismo algoritmo a los datos recibidos y compara el FCS obtenido de esta forma con el que se adicionó a los datos originales. Si son iguales se toma el mensaje, de lo contrario se supone un error.

Estas técnicas están basadas en dos métodos comunes:

- Verificación de paridad en dos coordenadas Cuando se transmiten datos a un dispositivo que cuente con un buffer, es posible extender la verificación de paridad simple añadiendo un bloque de verificación de carácter (Block Check Character BCC) al final del bloque de datos, el cual realizará la segunda verificación de paridad a todo el bloque. A continuación un ejemplo:

Tabla 1.4 Ejemplo de la Técnica de Paridad de Coordenadas

Carácter	Bits del carácter							Bit de paridad
1	1	0	0	1	1	0	0	1
2	0	0	1	1	1	0	1	0
3	0	1	1	0	0	0	0	0
4	1	1	0	1	0	1	1	1
5	1	0	1	0	1	0	1	0
6	0	0	1	1	0	0	0	0
7	1	1	0	0	0	0	1	1
BCC	0	1	0	0	1	1	0	1

Verificación de paridad en dos coordenadas.

En la técnica de verificación de paridad en dos coordenadas se pueden dar los casos indicados en la tabla 1.5 (en rojo están los bits erróneos).

Tabla 1.5 Ejemplo de Errores en la Técnica de Paridad de Coordenadas

1 error detectado (por las 2 paridades)				2 errores detectados (sólo paridad vertical)				Errores ocultos			
0	1	1	0	0	1	1	0	0	1	1	0
1	1	**1**	0	1	1	0	0	1	**0**	1	0
0	0	1	1	0	**1**	**0**	1	0	**1**	**0**	1
1	0	0	1	1	0	0	1	1	0	0	1

- Verificación por redundancia cíclica (CRC) Esta técnica es ampliamente usada debido a que es fácil de implementar en los circuitos integrados a muy gran escala (VLSI) que forman el hardware. Un mensaje puede verse como un simple número binario, el cual puede ser dividido por una cantidad que consideraremos constante, al efectuar la división (a módulo 2) se obtiene un cociente y un residuo, este último es transmitido después del mensaje y es comparado en la estación receptora con el residuo obtenido por la división de los datos recibidos y el mismo valor constante. Si son iguales los residuos se acepta el mensaje, de lo contrario se supone un error de transmisión. En el proceso de datos comercial es ampliamente usada la verificación por redundancia cíclica de 16 bits de longitud, aunque también es posible usar 32 bits lo cual puede ser más efectivo.

1.7.4 CORRECCIÓN DE ERRORES

"Se poseen las siguientes técnicas para la corrección de errores:

a. Por Operador Humano

Si los mensajes transmitidos son únicamente textos, puede resultar más económico y fácil que un operador humano reciba e interprete el mensaje y de ser necesario lo corrija usando su propio criterio.

Algunos sistemas que aplican verificación por paridad cambian automáticamente los caracteres con error de paridad por el símbolo "?" para que el operador humano pueda identificarlos y corregirlos.

b. Código Hamming de Corrección Automática de Errores

Asocia bits de paridad par con combinaciones únicas de bits de datos. Este método permite detectar y corregir con seguridad hasta un bit por cada bloque de información transmitida.

A cada n bits de datos se le añaden k bits de paridad de tal forma que el carácter transmitido tiene n+k bits de longitud. Los bits se numeran de izquierda a derecha (el 1º bit es el más significativo). Todo bit cuyo número sea potencia de 2 es un bit de paridad, los restantes serán bits de datos. Los bits de dato se acomodan en sus posiciones y los bits de paridad se calculan de modo que tengan una paridad par sobre los bits cuyo número de bit formen. De esta forma cada bit está verificado por una combinación única de bits de paridad, de modo que analizando los errores de paridad se puede determinar que bit es el que ha invertido su estado. A continuación se dan algunos ejemplos que muestran cómo se pueden localizar los bits alterados:

Tabla 1.6 Ejemplo del Código Hamming

Paridad incorrecta en los bits	El error está en el bit número

4	4
1 y 4	5
1, 2 y 4	7
1 y 8	9

En el caso que exista más de un error en el bloque de información se llegan a producir varias situaciones que pueden llevar a la "corrección" de un bit no alterado (Ej: si cambian los bits 1 y 2 llevan a la corrección del bit sano 3), entre muchas otras situaciones.

Una variante del código Hamming es adicionarle 1 bit de paridad global. De esta forma es posible tener la seguridad de detección de 2 errores, manteniendo la capacidad de corrección si se produce sólo 1 error.

Las Desventajas del código Hamming son:

La cantidad de bits de paridad empleados en la transmisión de la información le restan eficiencia al proceso. Se define la eficiencia de transmisión con la siguiente fórmula:

$$Eficiencia = \frac{Información / Tiempo\ \ unitario}{Capacidad / Tiempo\ \ unitario} \quad \textbf{(Ec. 1.2)}$$

Suponiendo que se desea transmitir bloques de 8 bits de información, se necesitan 4 bits de paridad para ello, con lo que se tiene un total de 12 bits. La eficiencia será:

$$Eficiencia(8+4) = \frac{8}{12} = 0.6666 \ (x\,100) = 66.66\%$$

La eficiencia de este tipo de transmisión resulta de 66.66% debida solamente al plan de codificación. Además, dependiendo del método de transmisión puede decaer todavía más"[6].

1.8 PROTOCOLOS

"Cuando se dispone de dispositivos de hardware separados geográficamente, existirán procedimientos para control de cada dispositivo implementados por procesos de software. Como los procesos se ejecutan en hardware separado, deben intercambiar mensajes para coordinar la acción y obtener SINCRONIZACIÓN.

Para realizar el intercambio de mensajes se debe diseñar (cuidadosamente) los procedimientos o protocolos. La principal característica, es la habilidad para trabajar en un ambiente donde los periodos (timing) y secuencia de eventos es desconocida y se esperan errores en la transmisión de datos.

[6] http://www.itlp.edu.mx/publica/tutoriales/telepro/t1_3.htm

El término protocolo se usa para describir el intercambio de información entre Procesos (Programas que se ejecuten en un hardware).

Pueden darse Procesos en: Equipos de una red, Sistemas multiprocesador para controlar interacción de procesos paralelos, aplicaciones en tiempo real para el control de dispositivos y en cualquier sistema donde no existe relación fija en el tiempo de ocurrencia de los eventos.

Una definición más formal de Protocolo será entonces: *Especificación de la lógica y de los procedimientos de los mecanismos de comunicación entre procesos.*

La definición lógica constituye la sintaxis y la definición de los procedimientos constituye la semántica. Las Funciones más importantes son:

- Control de Errores
- Control de Flujo
- Control de Congestión
- Estrategias de Encaminamiento

El Control de Errores protege la integridad de los datos del usuario y de los mensajes de control.

El Control de Flujo y Congestión permite a la red compartir sus recursos entre un gran número de usuarios, entregando a cada uno un servicio satisfactorio sin que sus operaciones corran peligro.

Las Estrategias de Encaminamiento permiten optimizar la utilización de los recursos de la red, aumentando la disponibilidad de los servicios al proveer caminos alternativos entre nodos terminales.

Los protocolos son implementados vía procesos, un proceso se ejecuta en un procesador virtual o lógico. Un proceso puede ser Autocontenido, o sea que no se da cuenta (y no le interesa), que un procesador real comparta sus recursos entre varios procesos activos.

La Entrada a los procesos ocurre por las puertas lógicas de software, por donde el proceso recibe mensajes desde procesos residentes en el mismo o en otro procesador. Un conjunto de datos privados definen el estado actual de un proceso y determinan la acción a tomar por el receptor de un mensaje. La figura 1.4 presenta el esquema de un modelo general para un proceso.

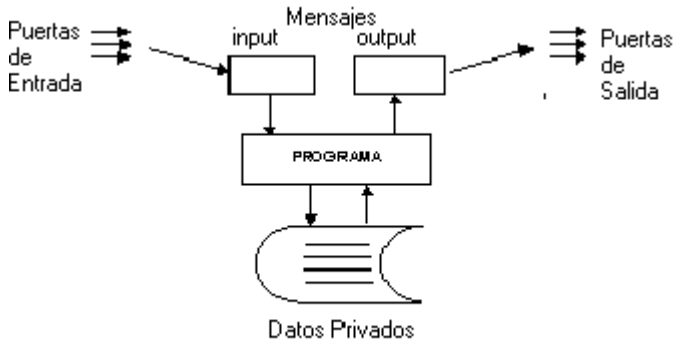


Figura 1.4 Esquema del Modelo de un Proceso

La forma en la que opera un protocolo es la siguiente:

Un proceso recibe un mensaje, lo procesa y envía una respuesta, sin que exista relación entre éste evento y otro anterior o posterior. El proceso origen, conocerá la dirección del proceso destino y la incluirá en el mensaje, esta dirección, identificará únicamente a un procesador, que conocerá al proceso destino. El originador despacha un mensaje, entra a un estado de espera de respuesta en una de sus puertas.

El proceso destino ejecuta la función especificada en el mensaje, construye la respuesta (con resultados y dirección del origen) y envía el mensaje respuesta por una puerta de salida quedando libre para aceptar otro mensaje. La respuesta llega al originador, quien realiza un chequeo para asegurarse que viene del lugar correcto antes de aceptarla, luego, pasa al estado “no espera respuesta” en esa puerta de entrada.

Este es un protocolo muy sencillo, necesita de la sintaxis para definición de formatos de los mensajes y una semántica muy sencilla. Debe considerarse el hecho que, la red introduce retardos causadas por congestión, encaminamiento, etc., e incluso puede ocurrir pérdida del mensaje. Para esto, el proceso que realiza la consulta deberá tener un reloj (timer) el que será activado al enviar el mensaje. El reloj enviará una señal al expirar el tiempo indicado en la activación, indicando que la respuesta no llegó en el tiempo esperado por lo que el mensaje deberá ser retransmitido.

En definitiva, los **protocolos** se utilizan para la comunicación entre **entidades** de diferentes sistemas. En general, una entidad es *algo* capaz de enviar o de recibir información, y un sistema es un objeto que contiene una o más entidades. Para que dos entidades puedan comunicarse deben *hablar el mismo idioma*. Qué se comunica, cómo se comunica y cuándo se comunica, debe cumplir ciertas convenciones entre las entidades involucradas. Este conjunto de convenciones constituye un **protocolo,** que puede definirse como *el conjunto de reglas que gobiernan el intercambio de datos entre dos entidades*.

En la industria se aceptó hace ya bastante tiempo, la necesidad de estándares que gobernaran las acciones y las características físicas y eléctricas de los equipos de comunicación"[7].

Este punto de vista, sin embargo ha tardado en imponerse en la industria de los ordenadores. Entre las organizaciones más importantes que han colaborado en el desarrollo de estándares en el área digital están:

ISO ***(International Organization for Standarization):*** Agrupa a 89 países, se trata de una organización voluntaria, no gubernamental, cuyos miembros han desarrollado estándares para las naciones participantes. Uno de sus comités se ocupa de los sistemas de información. Han desarrollado el modelo de referencia **OSI** (Open Systems Interconnection) y protocolos estándar para varios niveles del modelo.

CCITT ***(Comité Consultatif International de Télégraphique et Téléphonique):*** Organización de la Naciones Unidas constituida, en principio, por las autoridades de Correos, Telégrafos y Teléfonos (PTT) de los países miembros. Estados Unidos está representado por el departamento de Estado. Se encarga de realizar recomendaciones técnicas sobre teléfono, telégrafo e interfaces de comunicación de datos que a menudo se reconocen como estándares. Trabaja en colaboración con **ISO** (que en la actualidad es miembro de CCITT).

EIA ***(Electronic Industries Association):*** Asociación vinculada al ámbito de la electrónica. Es miembro de **ANSI**. Sus estándares se encuadran dentro del nivel 1 del modelo de referencia **OSI.**

[7] http://www.itlp.edu.mx/publica/tutoriales/telepro/t3_3.htm

ANSI ***(American National Standard Institute):*** Asociación con fines no lucrativos, formada por fabricantes, usuarios, compañías que ofrecen servicios públicos de comunicaciones y otras organizaciones interesadas en temas de comunicación. Es el representante estadounidense en **ISO**, que adopta con frecuencia los estándares **ANSI** como estándares internacionales.

La aceptación mayoritaria de los diferentes estándares ha supuesto un crecimiento de la oferta de equipos compatibles de diversos fabricantes, proporcionando a los usuarios una mayor libertad de elección, favoreciendo la competencia entre fabricantes e incrementando la demanda de equipos compatibles. Sin embargo los estándares llevan también aparejados ciertos inconvenientes, como puede ser la introducción de retraso tecnológico que ralentiza nuevos desarrollos y la multiplicidad de estándares no compatibles.

Por todo lo indicado, antes de iniciar cualquier comunicación se debe determinar el protocolo a seguir. Es así que por ejemplo para la comunicación serial, el protocolo debe decidir sobre: el número de bits de los datos, la paridad, el número de bits de parada, la velocidad de transmisión y el control de flujo.

1.9 INTERFASE RS - 232

"La comunicación serie es la forma de bajo nivel más utilizada para comunicar dos o más dispositivos. Normalmente, un dispositivo es una computadora, mientras el otro dispositivo puede ser un módem, una copiadora, otra computadora, o un instrumento científico como un osciloscopio o un generador de funciones.

Como el nombre lo sugiere, el puerto de serie envía y recibe bytes de información en una forma serie, los bits de uno en uno en cada tiempo. Estos bytes que se transmiten pueden estar en un formato binario (numérico) o en un formato de texto.

La interfase del puerto serie para conectar dos dispositivos se especifica por el TIA/EIA-232C norma estándar publicada por la Asociación de la Industria de las Telecomunicaciones. La norma original fue dada a través de RS-232, término todavía popular y que define las siguientes características:

- El velocidad máxima de transferencia de los bits y la longitud del cable.
- El nombre, las características eléctricas y la función de las señales.
- Las conexiones mecánicas y la asignación de pines.

Para la comunicación se necesitan tres pines: el pin de Transmisión de Datos, el pin de Recepción de Datos y el pin de Tierra. Otros pines están disponibles para el control del flujo de los datos, pero no se requieren (ver tabla 1.7).Ya que RS-232 involucra la conexión de un Equipo Terminal de Datos (DTE) a un Equipo de Comunicación de Datos (DCE), los pines conectados a través del cable deben conectarse el 1 con el 1, el 2 con el 2, y así sucesivamente. El DTE y el DCE usan para transmitir los datos el pin TD y para recibirlos el pin RD. Como referencia se encuentra la figura 1.5.

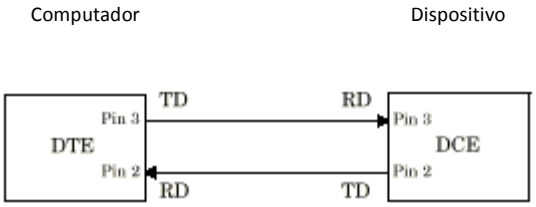


Figura 1.5 Conexión de un DTE a un DCE

Si se conectan dos DTE o dos DCE mediante un cable serial recto, el TD de un dispositivo se conecta con el RD del otro dispositivo. Otra forma es utilizar el cable null modem, como lo indica la figura 1.6, los cables null MODEM cruzan el TD con el RD"[8].

[8]MATLABHelp

Computador Dispositivo

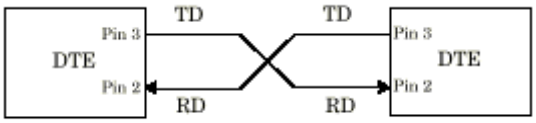


Figura 1.6 Conexión de un DTE a otro DTE

1.9.1 Asignación de Pines y Funciones del Puerto Serial

Los puertos serie poseen dos tipos de señales: señales de datos y señales de control. Los RS-232 normales poseen 25 pines; sin embargo, la mayoría de PCs y las plataformas de UNIX usan un conector de 9 pines. De hecho, se requieren sólo tres pines para el puerto de comunicaciones serie: uno para recibir los datos, uno para transmitir datos y otro para la tierra, como se indicó anteriormente.

El esquema de asignación de pines para un conector de 9 pines macho en un DTE se indica en la figura 1.7.

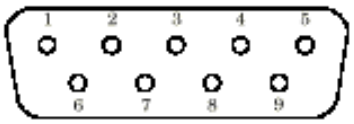


Figura 1.7 Cable de 9 pines macho

Los pines y señales asociadas con el conector de 9 pines se describen en la tabla 1.7.

Tabla 1.7 Función de los pines del conector macho

Pin	Etiqueta	Nombre de la Señal	Tipo de Señal
1	CD	Detector de portadora	Control
2	RD	Dato recibido	Datos
3	TD	Dato transmitido	Datos
4	DTR	Listo el terminal	Control
5	GND	Señal de referencia	Tierra
6	DSR	Listo set de datos	Control
7	RTS	Petición de envío	Control
8	CTS	Listo para envío	Control
9	RI	Indicador de timbrado	Control

1.9.2 Estado de las Señales

Las señales pueden estar en un estado activo o en un estado inactivo. Un estado activo corresponde al valor 1 binario, mientras un estado inactivo corresponde al valor 0 binario; un estado señalado activo se describe a menudo como lógica 1 o verdadero, un estado señalado inactivo se describe a menudo como lógica 0 o falso.

Para las señales de datos, el estado “ON” ocurre cuando el voltaje recibido es más negativo que -3 voltios, mientras que el estado “OFF” ocurre para voltajes más positivo que 3 voltios. Para las señales de control, el estado “ON” ocurre cuando la señal recibida tiene un voltaje más positivo que 3 voltios, mientras el estado “OFF” ocurre para voltajes más negativos que -3 voltios. El voltaje entre -3 voltios y +3 voltios es considerado una región de transición y el estado señalado es indefinido.

Los estados “ON” y “OFF” para una señal de datos y para una señal de control se muestran en la figura 1.8.

Señal de Datos Señal de Control

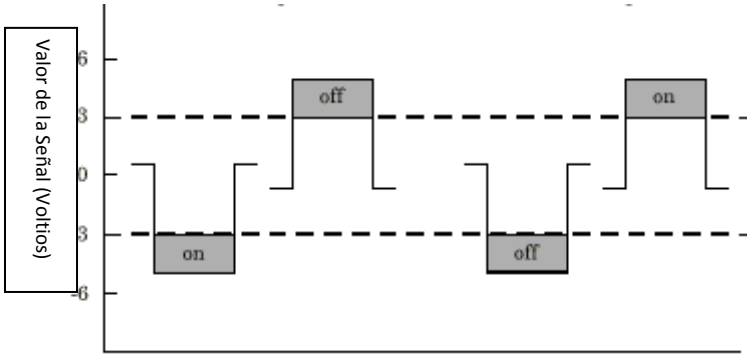


Figura 1.8 Estado de las Señales de Datos y de Control

1.9.3 Los Pines de Datos

La mayoría de los dispositivos que utilizan el puerto de serie emplean comunicación full-duplex, lo que significa que ellos pueda enviar y recibir datos al mismo tiempo, por consiguiente, se usan pines separados para transmitir y recibir los datos; estos dispositivos usan los pines TD, RD y GND.

Sin embargo, algunos tipos de dispositivos del puerto de serie utilizan sólo una vía o la comunicación half-duplex, para estos dispositivos, sólo se utilizan los pines TD y GND.

El pin TD lleva los datos transmitidos por un DTE a un DCE. El pin RD lleva los datos que son recibidos por un DTE de un DCE.

1.9.4 Los Pines de Control

De los 9 pines, algunos son de control, los mismos que tienen las siguientes funciones:

- Detectar la presencia de dispositivos conectados.
- Controlar el flujo de datos.

Los pines de control son: RTS (Petición de envío), CTS (Listo para envío), DTR (Listo el terminal), DSR (Listo set de datos), CD (Detector de portadora) y RI (Indicador de timbrado). Los pines RTS y CTS se utilizan para señalar si los dispositivos están listos para enviar o recibir datos. Este tipo de control del flujo de datos (llamado handshaking del hardware) se usa para prevenir la pérdida de los datos durante la transmisión.

Los pines CD y RI indican la presencia de ciertos signos durante las conexiones de módem-módem. CD en un módem señala que ha hecho una conexión con otro módem, o ha descubierto un tono del portador. CD es habilitado cuando el DCE está recibiendo un signo de una frecuencia conveniente, CD se deshabilita si el DCE no está recibiendo un señal conveniente.

1.9.5 Formato de los Datos Serie

"El formato de los datos de serie incluye un bit de inicio, entre cinco y ocho bits de datos y un bit de parada. Un bit de paridad y un bit de parada adicional podrían ser incluidos en el formato también. El diagrama de la figura 1.9 ilustra el formato de los datos serie.

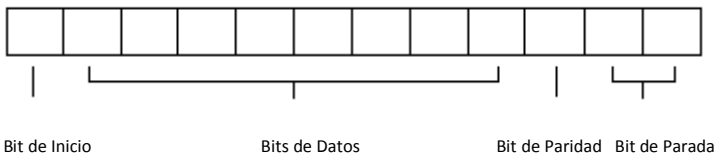


Figura 1.9 Formato de los Datos

El formato para los datos del puerto serie usa la notación siguiente:

Número de bits de los datos - el tipo de paridad - el número de bits de parada.

Por ejemplo, se interpreta 8-N-1 como ocho bits de los datos, ninguna paridad y un bit de parada, mientras se interpreta 7-E-2 como siete bits de los datos, paridad par y dos bits de parada.

Los bits de los datos a menudo se llaman “caracter”, porque normalmente representan un caracter en código ASCII. Los bits restantes se llaman a menudo “framing bits” porque ellos completan la trama de los datos.

a. Bytes vs. Valores

La colección de bits que comprenden el formato de los datos serie se llama un byte. A primera vista, este término podría parecer inexacto porque un byte es 8 bits y el rango de los datos en serie puede ir entre 7 y 12 bits. Sin embargo, cuando los datos serie se almacenan, por ejemplo en una computadora, los bits extras se despojan y sólo se retienen los bits de los datos. Es más, ocho bits de los datos siempre se usan sin tener en cuenta el número de bits que se especificó para la transmisión, es así que a los bits sin usar se les asigna un valor de 0.

Al leer o escribir datos se puede necesitar especificar un valor ya sea de uno o más bytes; por ejemplo, si se lee el valor de un dispositivo usando el formato int 32, entonces ese valor consta de cuatro bytes.

b. Comunicación Síncrona y Asíncrona

La RS-232 normal apoya dos tipos de protocolos de comunicación: síncrono y asíncrono. Usando el protocolo síncrono, todos los bits transmitidos se sincronizan a una señal de reloj común. Los dos dispositivos se sincronizan inicialmente uno con otro y entonces continuamente se envían los caracteres y se termina también sincronizadamente. Incluso cuando el dato real no está enviándose, existe un flujo constante de bits que permite a cada dispositivo saber en qué estado está el otro en cualquier momento; es decir, cada momento se envían datos reales o un caracter inactivo. Las comunicaciones síncronas permiten la transferencia de datos en forma más rápida que con los métodos asíncronos, porque los bits adicionales que marcan el inicio y el fin de cada byte de los datos no son requeridos.

Usando el protocolo asíncrono, cada dispositivo usa su propio reloj interior produciendo bytes que se transfieren en momentos arbitrarios. Así, en lugar de usar el tiempo como una manera de sincronizar los bits, se utiliza el formato de los datos. En particular, la transmisión usa el bit de inicio para sincronizar los datos, mientras uno o más bits de parada indican el fin de la palabra.

El requisito de enviar en las comunicaciones asíncronas los bits adicionales puede hacerla que sea ligeramente más lenta que las comunicaciones síncronas; sin embargo, se tiene la ventaja que el procesador no tiene que tratar con los caracteres inactivos adicionales. La mayoría de los puertos serie operan asincrónicamente.

c. Transmisión de los Bits

Por definición, el datos serie son transmitidos mediante un bit en cada tiempo. El orden en que los bits se transmiten se da a continuación:

- El Bit de inicio se transmite con un valor de 0.
- Se transmiten los bits de los datos. El primer bit de los datos corresponde al bit menos significativo (LSB), mientras que el último bit de los datos corresponde al bit más significativo (MSB).
- A continuación se transmite la paridad (si se definió).
- Se transmiten uno o dos bits de parada, cada uno con un valor de 1.

El número de bits transferidos por segundo está dado por el valor del baudio. La transferencia de bits incluye a todos los mencionados anteriormente.

BITS DE INICIO Y DE PARADA

El bit de inicio indica cuando el byte de los datos está a punto de empezar y el bit (s) de parada indica cuando el byte de los datos se ha

transferido. El proceso de identificar bytes con el formato de los datos serie sigue estos pasos:

- Cuando un pin del puerto de serie está ocioso (no transmitiendo datos), entonces está en un estado “ON”.
- Cuando el dato va a ser transmitido, el pin del puerto serie conmuta a un estado “OFF” debido al bit de inicio.
- El pin del puerto de serie conmuta luego a un “ON” debido al bit (s) de parada. Esto indica el fin del byte.

BITS DE LOS DATOS

Los bits de los datos transferidos a través de un puerto de serie podrían representar órdenes de un dispositivo, lecturas de un sensor, mensajes de error y así sucesivamente. Los datos pueden ser transferidos como datos binarios o datos de ASCII.

La mayoría de los puertos de serie usa entre cinco y ocho bits de datos. El dato binario es típicamente transmitido con ocho bits. El dato - texto se transmite como siete u ocho bits. Si el dato es basado en el carácter ASCII, entonces un mínimo de siete bits son requeridos porque hay 2^7 o 128 caracteres distintos.

Si se usa un puerto serie I/O de ocho bits, el bit más significativo MSB debe tener un valor de 0. Si el dato está basado en el carácter ASCII extendido, entonces deben usarse los ocho bits porque hay 2^8 o 256 caracteres distintos.

BIT DE PARIDAD

El bit de paridad proporciona una información de error simple (paridad) en los datos transmitidos. Los tipos de paridad se dan en la tabla 1.8.

Tabla 1.8 Tipos de Paridad

Tipo de Paridad	Descripción
Even (Par)	La sumatoria de bits 1 del dato resulta un número par
Mark (Marca)	El bit de paridad siempre es 1
Odd (Impar)	La sumatoria de bits 1 del dato resulta un número impar
Space (Espacio)	El bit de paridad siempre es 0

Rara vez se usa Mark como bit de paridad, debido a que ofrece mínima detección del error. También se puede escoger no usar paridad. El proceso que verifica la paridad sigue los pasos siguientes:

- El dispositivo transmisor pone el bit de paridad a 0 o a 1 dependiendo de los datos y del tipo de paridad seleccionada.
- El dispositivo receptor verifica si el bit de paridad es consistente con el de los datos transmitidos. Si es, entonces los bits de los datos se aceptan. Si no es, entonces un indicador de error se devuelve.

Hay que tomar en cuenta que el chequeo de Paridad puede descubrir sólo 1 bit de error. Los errores debido a múltiples bits pueden aparecer como datos válidos. Por ejemplo, suponga que el dato dado por los siguientes bits 01110001 se transmiten. Si la paridad seleccionada es la par, entonces el bit de paridad lo pone a 0 el dispositivo transmisor, para producir un número par de 1's. Si la paridad impar se selecciona, entonces el bit de paridad se pone a 1 para producir un número de 1's impar"[9].

CAPÍTULO II

SOFTWARE DE ADQUISICIÓN DE DATOS

2.1 GENERALIDADES

MATLAB es un lenguaje de programación de alto nivel, con un enfoque directo hacia la computación científica. Más concretamente, es un programa de cálculo numérico con gran número de instrucciones dirigidas a la resolución de problemas científicos. Desde este punto de vista puede ser considerado entonces como una gran calculadora científica programable y muy potente.

El nombre de MATLAB proviene de las palabras inglesas *Matrix Laboratory* (laboratorio de matrices), que evidentemente dan una idea de la utilidad primordial de dicho programa. Algunas de las aplicaciones del programa MATLAB pueden ser:

- Computación y matemáticas.
- Desarrollo de algoritmos.
- Modelado y simulación.
- Exploración, visualización y análisis de datos.
- Creación de gráficas científicas.

Estas características hacen del programa MATLAB una herramienta de trabajo muy extendida tanto entre estudiantes como entre técnicos e investigadores. A continuación se

brinda una revisión general del paquete, para luego hacer hincapié en la comunicación serial y el desarrollo de interfaces.

Las dos instrucciones más importantes en MATLAB son:

- **quit**: que sirve para acabar la sesión de trabajo y salir del programa (también puede usarse **exit**).
- **help**: que proporciona ayuda sobre las distintas funciones e instrucciones que estén disponibles en la versión usada. Hay que resaltar que la ayuda proporcionada por MATLAB ha cambiado con las distintas versiones del programa y, por ello, las respuestas a mismas cuestiones pueden ser distintas.

Cuando no se recuerda el nombre exacto de una función, hay otra instrucción que aparece en algunas versiones recientes del programa y que permite encontrarla partiendo de un patrón de texto determinado, todas las funciones que contienen, en su nombre o en la primera línea de ayuda respectiva, ese patrón. Esta instrucción es **lookfor**, y su formato de escritura es como el de **help**, o sea, se escribe la instrucción y luego el patrón de búsqueda, separados por un espacio (por ejemplo, **lookfor mesh** proporciona algunas funciones relacionadas con el mallado de regiones y dibujos tridimensionales).

Los formatos de escritura tienen, en MATLAB, la misma importancia que en cualquier otro lenguaje de programación. En primer lugar, hay que resaltar que MATLAB distingue entre mayúsculas y minúsculas y, habitualmente, las funciones e instrucciones básicas se escriben con minúsculas. Por ejemplo, no es lo mismo **pi** que **Pi**. Podemos teclear ambas expresiones y comprobar que la primera funciona perfectamente mientras que la segunda causa un error al ser ejecutada.

Los paréntesis "**( )**" y los corchetes "**[ ]**" tienen significados y utilidades distintas. Los primeros se utilizan para evaluar funciones (como en **sin(pi/3)**) y

para cambiar el orden básico de las operaciones (por ejemplo, **2 ∗(2+5 ∗(2+1))**), mientras que los segundos sirven como delimitadores de vectores (por ejemplo **[1,2,3,4]**) o de matrices (por ejemplo **[5,6,7;8,9,10]**). De ningún modo se pueden utilizar las llaves "**{ }**" en MATLAB con la finalidad de cambiar el orden de las operaciones.

"Los componentes más importantes del entorno de trabajo de MATLAB son los siguientes:

- **El Escritorio de Matlab** (Matlab Desktop), que es la ventana o contenedor de máximo nivel en la que se pueden situar los demás componentes.
- Los componentes individuales, orientados a tareas concretas, entre los que se puede citar a:

  - **La ventana de comandos** (Command Window), en la que se ejecutan interactivamente las instrucciones y en donde se muestran los resultados correspondientes, si es el caso. En cierta forma es la ventana más importantes y la única que existía en las primeras versiones de la aplicación.
  - **La ventana histórica de comandos** (Command History), ofrece acceso a las sentencias que se han ejecutado anteriormente en la ventana de comandos.
  - **El espacio de trabajo** (Workspace), es el conjunto de variables y de funciones de usuario que en un determinado momento están definidas en la memoria del programa.
  - **La plataforma de lanzamiento** (Launch Pad), es un componente muy general que da acceso a otros componentes de Matlab, sin tener que recurrir a los menús o a otros comandos.

- **El directorio actual** (Current Directory), el concepto de directorio activo o directorio actual es muy importante en Matlab. Los programas de Matlab se encuentran en ficheros con la extensión *.m, estos ficheros se ejecutan tecleando su nombre en la línea de comandos (sin la extensión), seguido de los argumentos entre paréntesis, si se trata de funciones. Para que un fichero *.m se pueda ejecutar es necesario que se cumpla una de las dos condiciones: que esté en el directorio actual, que esté en uno de los directorios indicados en el Path de Matlab.
- **La ventana de ayuda** (Help), donde se encuentra una amplia ayuda para todo el manejo del paquete.
- **El editor de ficheros y depurador de errores** (Editor&Debugger), el editor permite crear y modificar ficheros *.m y el debugger ejecutarlos paso a paso para ver si contienen errores.
- **El editor de vectores y matrices** (Array Editor), permite ver los valores de los elementos de cualquier matriz o vector definido en el programa donde es posible también modificar estos valores.
- **La ventana que permite revisar cómo se emplea el tiempo de ejecución** (Profiler), es una herramienta útil para determinar los cuellos de botella de un programa"[10].

A continuación, en la tabla 2.1. se encuentran algunos de los comandos de Matab.

Tabla 2.1 Algunos Comandos de Matlab

COMANDO	FUNCIÓN
FORMAT	Controla el formato numérico de los valores.
LIGHTING	Son varios comandos que habilitan y ajustan las características de los

	objetos.
CLEARING	Equivale al CLC, solo borra la pantalla, no la limpia.
HISTORY	Despliega las funciones usadas recientemente.
LINE EDITING	Son varios comandos y permiten operaciones como cortar, copiar, etc.

[10]MATLABTutorial - Javier García de Jalón de la Fuente (jgjalon≅etsii.upm.es)

2.2 COMUNICACIÓN CON EL PUERTO SERIE

∀El MATLAB proporciona acceso directo a los dispositivos periféricos como los módems, copiadoras y los instrumentos científicos a los que se conecta el puerto serie de la computadora. Esta interfase se establece a través de un objeto del puerto serie.

El objeto del puerto serial soporta funciones y propiedades que permiten realizar lo siguiente:

- Configurar las comunicaciones del puerto de serie.
    - Controlar los pines del puerto serie.
    - Leer y escribir los datos.
    - Usar eventos y recursividad (callbacks).

- Almacenar en el disco.

Para comunicar la PC con un hardware de adquisición de datos compatible en multifunciones de I/O, se necesita el Data Acquisition Toolbox. En cambio, para comunicarse con GPIB o los instrumentos VISA compatibles, se necesita el Instrument Control Toolbox.

Durante los últimos años se han desarrollado varias normas de interfase para el manejo del puerto serie; estas normas incluyen RS-232, RS-422, y RS-485 todas ellas pueden apoyarse en objetos del MATLAB.

De éstas, la más utilizada como interfase normal para las computadoras y los dispositivos periféricos serie es la RS-232.

2.2.1 FORMA DE TRABAJO CON LA I/O SERIE

Para trabajar en MATLAB con la interfase del puerto de serie, se deben tomar en cuenta los tres puntos siguientes:

- Inicialización del Puerto I/O Serie.

- Sesión con el Puerto Serie.
- Configurando y Restaurando Propiedades.

a. Inicialización del Puerto I/O Serie

Ilustra la forma básico de manejar el Puerto Serie. Si se tiene un dispositivo conectado al puerto serie COM1 y se configuró para una velocidad de 4800 baudios, se puede ejecutar el ejemplo siguiente de inicialización del puerto:

```
s = serial('COM1');
set(s,'BaudRate',4800);
fopen(s);
fprintf (s,'*IDN?')
out =  fscanf (s);
fclose(s)
delete(s)
clear s
```

En el programa anterior se crea un objeto serie **s,** al cual se le define su velocidad a 4800 baudios. A continuación se abre el objeto, situación imprescindible para que pueda transmitirse o recibirse información. Luego se envía un carácter al dispositivo, para seguidamente recibir en

la variable out la información que devuelve el dispositivo. Finalmente se cierra la comunicación, se limpia el objeto creado y se elimina la variable s definida.

El *IDN? da la información de identificación del dispositivo que está conectado.

b. Sesión con el Puerto Serie

La sesión del puerto de serie comprende todos los pasos probables que se deben tomar en cuenta cuando se conecta un dispositivo al puerto de serie. Estos pasos son:

- Crear un objeto del puerto de serie: Se crea un objeto del puerto de serie para un específico puerto de serie usando la función "serial creation". También se pueden configurar las propiedades durante la creación del objeto. En particular, se pueden configurar propiedades asociadas con las comunicaciones con el puerto de serie como la velocidad en baudios, el número de bits de los datos, y así sucesivamente.

- Conectar el dispositivo: Se conecta el objeto del puerto serie al dispositivo usando la función del "fopen".Después de que el objeto se conecta, se puede alterar la definición del dispositivo

configurando los valores de las propiedades, leer datos y escribir datos.

- Configurar las propiedades: Para establecer la conducta deseada del objeto de puerto de serie, se asigna valores a propiedades que usan la función “set” o “dot notation”. En la práctica, se puede configurar muchas de las propiedades a cualquier tiempo, al inicio, durante, o simplemente después de la creación del objeto. Recíprocamente, dependiendo de su dispositivo y los requisitos de la aplicación del puerto de serie, pueden aceptarse los valores de las propiedades predefinidos y saltarse este paso.

- Escribiendo y leyendo datos: Para escribir los datos en el dispositivo se usan las funciones “fprintf” o “fwrite” y para leer los datos del dispositivo se utilizan las funciones “fgetl” “fgets” “fread” “fscanf” o “readasync”. El objeto del puerto de serie se comporta según los valores previamente configurados o con los valores por defecto.

- Desconectar y limpiar: Cuando ya no se necesita el objeto del puerto de serie debe desconectárselo del dispositivo, para esto se usa la función “fclose”, para liberar la memoria se usa la función “delete” y para quitarlo del espacio de trabajo de MATLAB se utiliza el comando “clear”.

c. Configurando y Restaurando Propiedades

Establece la conducta deseada del objeto del puerto de serie, configurando los valores de las propiedades. Se puede desplegar o

configurar los valores de las propiedad usando las funciones “set”, “get” o “dot notation”.

.

2.2.2 CREACIÓN DE UN OBJETO DEL PUERTO SERIE

Se crea un objeto del puerto serie con la función “serial”. Se requiere el nombre del puerto serie conectado al dispositivo como un argumento de la entrada. Adicionalmente, se pueden configurar los valores de las propiedades durante la creación del objeto. Por ejemplo, crear un objeto del puerto de serie asociado con el puerto COM1, s = de serie (' COM1 '). El objeto creado **s** existe ahora en el workspace de MATLAB. Se puede desplegar la clase de **s** con el comando “whos”.

whos s

Name Size Bytes Class

s 1x1 512 serial object

Grand total is 11 elements using 512 bytes

Una vez creado el objeto del puerto de serie, las propiedades listadas en la tabla 2.2. son valores asignados automáticamente. Estas propiedades de propósito general proporcionan información descriptiva sobre el objeto del puerto serie basado en el tipo del objeto y el puerto serie.

Tabla 2.2 Descripción General del Propósito de la Propiedades

Nombre de la Propiedad	Descripción
Name	Especifa un nombre para el objeto del puerto serie
Port	Especifica el nombre de la plataforma del puerto serie
Type	Indica el tipo de objeto

Se pueden desplegar los valores de estas propiedades con la función “get”.

```
get(s,{'Name','Port','Type'})
ans =
'Serial-COM1'    'COM1'    'serial'
```

a. Configuración de las Propiedades Durante la Creación del Objeto

Se pueden definir las propiedades del puerto serie durante la creación del objeto, mediante las propiedades “name” y “values” dentro del formato, así como también usar la función “set”.

Por ejemplo:

```
s = serial('COM1','BaudRate',4800,' parity','even');
```

Si especifica el nombre de una propiedad inválida, el objeto no se crea. Sin embargo, si se especifica un valor inválido para algunas propiedades (por ejemplo, BaudRate se fija a 50), el objeto podría crearse pero no se informará de la invalidez del valor hasta que se conecte el objeto al dispositivo con la función del “fopen”.

b. Despliegue del Objeto del Puerto Serie

El despliegue del objeto del puerto serie proporciona un resumen importante sobre la configuración y el estado del mismo. Se puede invocar el despliegue de estas tres maneras:

- Digitando el nombre de la variable del objeto del puerto serie desde la línea del comando.
- No incluyendo el punto y coma al crear un objeto del puerto serie.
- No incluyendo el punto y coma al configurar las propiedades que usan la notación punto.

El ejemplo del despliegue para un objeto del puerto serie “s” se da a continuación:

Communication Settings

Port: COM1

BaudRate: 9600

Terminator: 'LF'

Communication State

Status: closed

RecordStatus: off

Read/Write State

TransferStatus: idle

BytesAvailable: 0

ValuesReceived: 0

ValuesSent: 0

c. Creación de un Arreglo para Objetos del Puerto Serie

En MATLAB, se puede crear un arreglo (array) de variables concatenándolas entre sí, esto también sirve para los objetos del puerto serie.

Por ejemplo, si se crearon en el puerto de serie los objetos s1 y s2:

```
s1 = serial('COM1');
s2 = serial('COM2');
```

Se puede crear un arreglo con s1 y s2 utilizando la sintaxis usual de MATLAB. Para crear la serie de la fila x:

```
x = [s1 s2]
```

El arreglo de objetos creado es:

Index:	Type:	Status:	Name:
1	serial	closed	Serial-COM1
2	serial	closed	Serial-COM2

Para crear la columna y, puede digitar:

```
y = [s1;s2];
```

No se puede crear una matriz de objetos del puerto de serie. Por ejemplo:

```
z = [s1 s2;s1 s2];
```

Obtiene:

??? Error using ==> serial/vertcat

Only a row or column vector o instrument objects can be created.

Sólo una fila o vector columna de objetos pueden crearse. Dependiendo de la aplicación, podría querer pasar un arreglo de objetos del puerto de serie a una función. Por ejemplo, para configurar los baudios y la paridad para s1 y s2 puede usar la función "set", así:

set(x, 'BaudRate',19200, 'Parity', 'even')

2.2.3 CONECCIÓN AL DISPOSITIVO

Antes de que el objeto del puerto serie pueda escribir o leer datos, debe conectarse a él el dispositivo vía el puerto serie especificado en la función "serial". Se conecta un objeto del puerto serie al dispositivo con la función "fopen". Ejemplo:

fopen(s)

Algunas propiedades son sólo de lectura mientras el objeto del puerto serie se conecta y deben configurarse antes de usar "fopen". Algunas de ellas son

InputBufferSize y las propiedades de OutputBufferSize. Se puede crear cualquier número de objetos del puerto serie; sin embargo, sólo un objeto está conectado al puerto de serie en un momento.

Se puede examinar la propiedad "status" para verificar que el objeto del puerto serie está conectado al dispositivo. Por ejemplo:

s.Status

ans =

open

La figura 2.1 ilustra la conexión entre el puerto serie y un dispositivo.

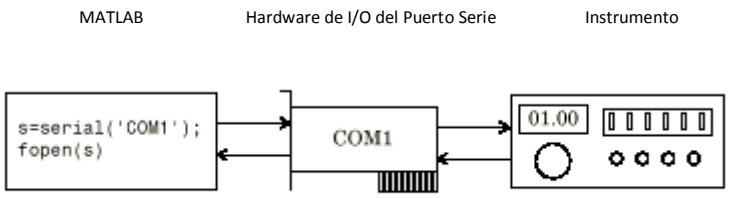


Figura 2.1 Conexión de un Dispositivo al Puerto Serie

2.2.4 DEFINICIONES DE LAS CONFIGURACIONES DE LA COMUNICACIÓN

Antes de que se pueda escribir o leer datos, el objeto del puerto serie y el dispositivo deben poseer los parámetros de comunicación idénticos. Configurar las comunicaciones involucra especificar valores para las propiedades que controlan la velocidad y el formato de los datos serie. Estas propiedades se indican en la tabla 2.3.

Tabla 2.3 Propiedades de la Comunicación

Nombre de la Propiedad	Descripción
BaudRate	Especifica la velocidad a la que serán transmitidos los bits
DataBits	Especifica el número de bits de los datos a transmitir

Parity	Especifica el tipo de chequeo de la paridad
StopBits	Especifica el número de bits utilizados para indicar el inicio y el fin
Terminator	Especifica el caracter final

Si el objeto del puerto serie y el dispositivo no están seteados con los mismos parámetros no se puede leer o escribir datos con éxito.

2.2.5 ESCRITURA Y LECTURA DE DATOS

Para muchas aplicaciones del puerto serie, hay tres importantes puntos que se deben considerar al escribir o leer datos:

- El control de acceso al comando en línea de Matlab.
- El tipo de dato que se va a transmitir, binario (numérico) o texto.
- Bajo qué condiciones se completa la lectura y escritura.

a. Control del acceso al comando en línea

Se controla el acceso al comando en línea especificando si la operación de lectura o escritura es síncrona o asíncrona. Un funcionamiento síncrono bloquea al acceso al comando en línea hasta que la operación de lectura o escritura haya completado su ejecución. Un funcionamiento asíncrono no bloquea el acceso al comando en línea, además pueden emitirse órdenes adicionales mientras la lectura o escritura están ejecutándose.

Entonces, las dos ventajas principales de la escritura – lectura de datos asíncrona son:

- Se pueden emitir otras órdenes mientras las funciones de escritura o lectura están ejecutándose.
- Se pueden usar todas las propiedades de la recursividad (callback).

Ya que los puertos serie tienen pines separados para la lectura y la escritura, se puede simultáneamente leer y escribir datos. Esto se ilustra en la figura 2.2.

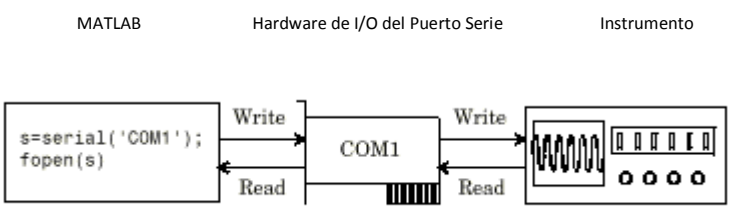


Figura 2.2 Ejemplo de lectura – escritura a un dispositivo

b. Escritura de Datos

Deben analizarse los siguientes puntos:

- **El Buffer de Salida y el Flujo de Datos.** El buffer de salida es la memoria de la computadora asignada para que el objeto del puerto serie almacene los datos que serán escritos en el dispositivo. Al escribir datos al dispositivo, el flujo de los datos sigue estos dos pasos: Los datos especificados por la función "write" se envían al buffer de salida y luego los datos salen de éste hacia el dispositivo.

La propiedad de OutputBufferSize especifica el número máximo de bytes que se pueden guardar en el buffer de salida y la propiedad de BytesToOutput indica el número de bytes actual en el buffer. Los valores predefinidos para estas propiedades son las siguientes:

```
s = serial('COM1');
get(s,{'OutputBufferSize','BytesToOutput'})
ans =
[512]    [0]
```

Si se intenta escribir más datos de los que puede almacenar el buffer de salida, se devuelve un error y ninguno de los datos se escribe. La figura 2.3 recrea una operación de escritura en el puerto.

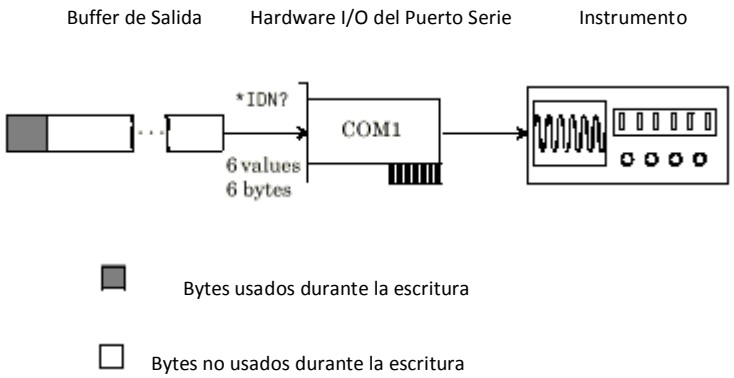


Figura 2.3 Operación de Escritura en el Puerto Serie

- **Escribir Datos tipo Texto**. Se utiliza la función "fprintf" para escribir datos de texto al dispositivo. Para muchos dispositivos, escribir datos de texto significa enviar órdenes de seteo del dispositivo, preparar al dispositivo para devolver datos o información de estado, y así sucesivamente.

Por defecto, "fprintf" escribe datos que usan el formato %s\n porque muchos dispositivos del puerto aceptan órdenes sólo tipo texto. Sin embargo, se pueden especificar muchos otros formatos. Se puede verificar el número de valores enviados al dispositivo con la propiedad "ValuesSent". Como por ejemplo:

s.ValuesSent

ans =

20

También por defecto, el “fprintf” opera síncronamente y Matlab bloqueará las órdenes en línea hasta la ejecución completa. Para escribir datos de texto asincrónicamente al dispositivo, se debe especificar “async” como el último argumento de la entrada al “fprintf”. Por ejemplo:

fprintf(s, 'Display:Contrast 45 ' , 'async')

Adicionalmente, se puede determinar qué operación asíncrona está en marcha con la Propiedad “TransferStatus”. Si ningún funcionamiento asíncrono está en marcha, entonces se obtiene la siguiente información :

s.TransferStatus

ans =

idle

La operación de escritura con “fprintf”, tanto síncrona o asíncrona, está completa cuando:

- Los datos especificados están ya escritos.
- El tiempo especificado por la propiedad “Timeout” ha transcurrido.

También, se puede detener el funcionamiento de escribir asincrónicamente con la función “stopasync”.

- **Escribir Datos Binarios**. Se usa la función “fwrite” para escribir datos binarios al dispositivo. Escribir datos binarios significa escribir valores numéricos. Una aplicación típica por escribir datos binarios es la calibración de un instrumento.

Por defecto, la función “fwrite” traduce valores que usan la precisión uchar. Sin embargo, se pueden especificar muchas otras precisiones. También, por defecto, “fwrite” opera síncronamente.

Para escribir datos binarios asincrónicamente a un dispositivo, se debe especificar “async” como la última entrada en el argumento de “fwrite”.

Entonces, las funciones asociadas con la escritura de datos se resumen en la tabla 2.4 y las propiedades asociadas con las mismas en la tabla 2.5.

Tabla 2.4 Funciones Asociadas con la Escritura de Datos

Nombre de la Función	Descripción
Fprintf	Escritura de texto en el dispositivo
Fwrite	Escritura de datos binarios en el dispositivo
Stopasync	Finalización asincrónica de las operaciones de lectura y escritura

Tabla 2.5 Propiedades Asociadas con la Escritura de Datos

Nombre de la Propiedad	Descripción
BytesToOutput	Indica el número de bytes en el buffer de salida
OutputBufferSize	Especifica el tamaño del buffer de salida en bytes
Timeout	Especifica el tiempo de espera al completar una operación de lectura o escritura
TransferStatus	Indica si un operación de lectura o escritura asincrónica está en progreso
ValuesSent	Indica el número total de valores escritos en el dispositivo

c. Lectura de Datos

Deben analizarse los siguientes puntos:

- **El Buffer de Entrada y el Flujo de Datos.** El buffer de entrada es la memoria de la computadora asignada para que el objeto del puerto serie guarde los datos leídos del dispositivo. Al leer datos de un dispositivo, el flujo de los datos sigue estos dos pasos: El dato leído del dispositivo se guarda en el buffer de entrada y luego el buffer lo devuelve a la variable de especificada por la función "read".

  La propiedad "InputBufferSize" especifica el número máximo de bytes que se puede guardar en el buffer de entrada. La propiedad "BytesAvailable" indica el número de bytes disponibles para ser leídos del buffer de entrada. El valor por defecto para estas propiedades son:

```
s = serial('COM1');
get(s,{'InputBufferSize','BytesAvailable'})
ans =
[512]    [0]
```

Si se intenta leer más datos de los que puede almacenar el buffer de entrada, se devuelve un error y no se leen los datos. La figura 2.4 ilustra la operación de lectura de datos de un instrumento.

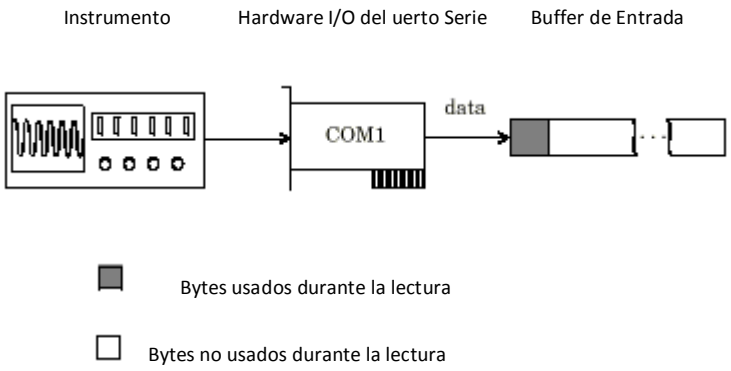


Figura 2.4 Operación de Lectura de un Dispositivo

- **Lectura de Datos de Texto.** Para leer datos con el formato texto se pueden utilizar las funciones “fgetl”, “fgets”, y el “fscanf”.

Por defecto, el “fscanf” lee datos que usan el formato %c. Sin embargo, se pueden especificar muchos otros formatos. Se puede también verificar el número de valores leídos del dispositivo con la propiedad de “ValuesReceived”. A continuación un ejemplo:

```
s.ValuesReceived
ans =
56
```

Se puede especificar si la operación de lectura es sincrónica o asincrónica con la propiedad “ReadAsyncMode”, la cual puede configurarse como continua o manual.

Si “ReadAsyncMode” es continua (valor por defecto), el objeto del puerto serie continuamente pregunta al dispositivo para determinar si los datos están disponibles para ser leídos. Si el dato está disponible, se guarda asincrónicamente en el buffer de entrada. Para transferir el dato del buffer de entrada a Matlab, se puede usar la forma de lectura síncrona usando las funciones como “fgetl” o “fscanf”. Si el dato está disponible en el buffer de entrada, estas funciones retornan rápidamente. Por ejemplo:

```
s.ReadAsyncMode = 'continuous';
fprintf (s,'*IDN?')
s.BytesAvailable
ans =
 56
out =  fscanf (s);
```

Si “ReadAsyncMode” es manual, el objeto del puerto serie no pregunta continuamente al dispositivo para determinar si el dato está disponible para ser leído. Para leer datos asincrónicamente,

se usa la función “readasync”. Se puede emplear también una de las funciones síncronas de lectura para transferir datos al buffer de entrada a Matlab. Por ejemplo:

```
s.ReadAsyncMode = 'manual';
fprintf (s,'*IDN?')
s.BytesAvailable
ans =
   0
readasync(s)
s.BytesAvailable
ans =
   56
out =  fscanf (s);
```

Adicionalmente, mientras la operación de lectura asíncrona está en progreso, se puede realizar una operación de escritura en el puerto y utilizar las propiedades del callback. Se puede determinar qué funcionamientos asíncronos están en progreso con la propiedad “TransferStatus”. Si ningún funcionamiento asíncrono está en marcha, entonces “TransferStatus” no está operando.

- **Lectura de Datos Binarios.** Se utiliza la función “fread” para leer datos binarios desde un dispositivo.

Debido a que el valor predefinido para la propiedad “ReadAsyncMode” es continuo, los datos se devuelven asincrónicamente al buffer de entrada en cuanto esté disponible el dispositivo. Se puede verificar el número de valores leído con la propiedad “BytesAvailable”. Por ejemplo:

```
s.BytesAvailable
ans =
69
```

Se puede devolver los datos a Matlab usando cualquiera de las funciones de lectura sincrónicas; sin embargo, si se emplea “fgetl”, “fgets” o “fscanf”, entonces se debe emitir la función dos veces porque hay dos valores guardados en el buffer de entrada. Si se usa “fread”, se devuelven todos los datos a Matlab llamando a la función. Así:

```
out = fread(s,69);
```

Por defecto, el “fread” devuelve valores numéricos en arreglos de precisión doble; sin embargo, se pueden especificar muchas otras precisiones. Se puede también, convertir los datos numéricos a texto empleando la función “char”. Por ejemplo:

```
val = char(out) '
```

Las funciones y propiedades asociadas a la operación de lectura se encuentran en las tablas 2.6 y 2.7

Tabla 2.6 Funciones Asociadas a la Lectura

Nombre de la Función	Descripción
Fgetl	Lectura de una línea de texto del dispositivo descartando la finalización
Fgets	Lectura de una línea de texto del dispositivo incluyendo la finalización
Fread	Lectura de un dato binario desde el dispositivo
Fscanf	Lectura de datos desde el dispositivo, y el formato como texto
Readasync	Lectura de datos asincrónicamente desde el dispositivo
Stopasync	Finalización asincrónica de las operaciones de lectura y escritura

Tabla 2.7 Propiedades de la Lectura de Datos

Nombre de la Propiedad	Descripción
BytesAvailable	Indica el número de bytes disponibles en el buffer de entrada
InputBufferSize	Especifica el tamaño del buffer de entrada en bytes
ReadAsyncMode	Especifica si la operación de lectura asincrónica es continua o manual
Timeout	Especifica el tiempo de espera al completar una operación de lectura o escritura

TransferStatus	Indica si un operación de lectura o escritura asincrónica está en progreso
ValuesReceived	Indica el número total de valores leídos desde el dispositivo

2.2.6 EVENTOS Y RUTINAS DE SERVICIO (CALLBACKS)

Se puede reforzar el poder y flexibilidad de una aplicación del puerto de serie usando eventos. Un evento ocurre después de una condición y si se repiten muchos eventos se podría producir uno o más callbacks.

Mientras el objeto del puerto de serie se conecta al dispositivo, se pueden usar eventos para desplegar un mensaje, desplegar datos, analizar datos, etc. El callback es controlado a través de las propiedades y las funciones del callback, todos los eventos tienen una propiedad callback asociada. Las funciones de callback son funciones M-file y se ejecuta un callback cuando un evento particular ocurre para lo cual se debe especificar el nombre del archivo tipo M y el valor por la propiedad asociada al callback.

a. Tipos de Eventos y Propiedades Callback

Los tipos de eventos del puerto serie y las propiedades del callback se describen en la tabla 2.8.

Tabla 2.8 Eventos y Propiedades

Tipo de Evento	Propiedad Asociada
Interrupción por ruptura	BreakInterruptFcn,
Bytes disponibles	BytesAvailableFcn BytesAvailableFcnCount BytesAvailableFcnMode
Error	ErrorFcn

Salida vacía	OutputEmptyFcn
Estado de los pines	PinStatusFcn
Cronómetro	TimerFcn TimerPeriod

- **Interrupción.** Un evento de interrupción se genera inmediatamente después de que una interrupción es generada por el puerto serie. El puerto serie genera una interrupción cuando se recibe.

  Este evento ejecuta la función del callback especificada para la propiedad "BreakInterruptFcn". Puede generarse para funcionamiento síncrono y asíncrono, en lectura y escritura.

- **Bytes disponibles.** Un evento byte disponible se genera inmediatamente después de que el número predeterminado de bytes está disponible en el buffer de entrada o cuando ha terminado la lectura. Está determinado por la propiedad "BytesAvailableFcnMode".

  Este evento sólo puede generarse durante un funcionamiento de lectura asíncrona.

- **Error.** Un evento de error se genera inmediatamente después de que ocurre un error. Este evento ejecuta la función del callback especificada para la propiedad de "ErrorFcn". Sólo puede generarse durante un funcionamiento de lectura o escritura asíncrona.

  Un evento del error se genera cuando una interrupción ocurre y un error ocurre si las operaciones de lectura o escritura no se completan con éxito dentro del tiempo especificado por la propiedad de la Interrupción. Un evento del error no se genera para los errores de la configuración como por ejemplo poner un valor de propiedad inválido.

- **Salida Vacía.** Un evento de salida vacía se genera inmediatamente después que el buffer de salida está vacío. Este evento ejecuta la función del callback especificada para la propiedad "OutputEmptyFcn"; sólo puede generarse durante una escritura asíncrona.

- **Estado de los Pines.** Un evento de estado de los pines se genera inmediatamente después de los cambios de estado (valor de los pines) para los pines CD, CTS, DSR, o RI. Este evento ejecuta la función del callback especificada para la propiedad "PinStatusFcn". Puede generarse para la lectura y escritura tanto síncrona como asíncrona.

- **Cronómetro**. Un evento de tipo cronómetro se genera cuando el tiempo especificado por la propiedad ha concluido (TimerPeriod). El tiempo es medido cuando el objeto del puerto se conecta al

dispositivo. Este evento ejecuta la función del callback especificada para la propiedad de “TimerFcn”.

Algunos eventos del cronómetro pueden no procesarse si el sistema está significativamente retardado o si el valor de “TimerPeriod” es demasiado pequeño.

b. Almacenamiento de la Información de los Eventos

Se puede guardar la información de un evento en una función del callback o en un archivo de registro.

La información del evento se guarda en una función del callback usando dos campos: Tipo y Datos. El campo Tipo contiene el tipo de evento, mientras el campo de Datos contiene la información del evento específico. El tipo de evento y los valores se dan en la tabla 2.9.

Tabla 2.9 Información de los Eventos

Tipo de Evento	Campo	Valor del Campo
Interrupción por ruptura	Type Data.AbsTime	BreakInterrupt day-month-year hour:minute:second
Bytes disponibles	Type	BytesAvailable

	Data.AbsTime	day-month-year hour:minute:second
Error	Type Data.AbsTime Data.Message	Error day-month-year hour:minute:second An error string
Salida vacía	Type Data.AbsTime	OutputEmpty day-month-year hour:minute:second
Estado de los pines	Type Data.AbsTime Data.Pin Data.PinValue	PinStatus day-month-year hour:minute:second CarrierDetect,ClearToSend, DataSetReady, or RingIndicator cn or off
Cronómetro	Type Data.AbsTime	Timer day-month-year hour:minute:second

El Campo “AbsTime” se define para todos los eventos e indica el tiempo absoluto en que ocurrió el evento. El tiempo absoluto usa el formato del reloj y devuelve lo siguiente:

day-month-year hour:minute:second

El Campo “Pin” es usado por el evento de estado de los pines para indicar si CD, CTS, DSR, o RI cambiaron de estado.

El Campo “PinValue” es utilizado por el evento de estado de los pines para indicar el estado de los pines CD, CTS, DSR, o RI. Los posibles valores son “ON” y “OFF”.

El Campo “Message” es usado por el evento error para guardar la descripción del mensaje cuando ocurrió un error.

c. Creación y Ejecución de Funciones Callback

Se pueden especificar funciones callback para ser ejecutadas cuando un tipo de evento ocurre, incluyendo el nombre del archivo M-file como un valor asociado en la propiedad del callback. Se puede especificar que los callback funcionen como un controlador de funciones o como un elemento de una cadena o arreglo.

Las funciones callback de archivo M-file requieren dos argumentos de entrada por lo menos. El primer argumento es el objeto del puerto serie y el segundo argumento es una variable que captura la información del evento; ésta información sólo pertenece al evento que causó la función de callback en ejecución.

d. Habilitación de Funciones Callback después que se presentó un Error

Si un error ocurre mientras una función callback se está ejecutando, entonces:

- La función callback es automáticamente inválida.
- Una advertencia se despliega en el comando en línea, indicando que la función callback es inválida.

Si se quiere habilitar para que funcione el mismo callback, se puede poner en la propiedad del callback el mismo valor o también se puede desconectar el objeto con la función “fclose”. Si se quiere usar una función callback diferente, los callback deberán ser habilitados cuando se configura la propiedad del callback al nuevo valor.

2.2.7 UTILIZACIÓN DE LOS PINES DE CONTROL

Los puertos serie incluyen seis pines de control, éstos pines de control permiten:

- Detectar la presencia de dispositivos conectados.
- Controlar el flujo de datos

Las propiedades asociadas con los pines de control del puerto serie se dan en la tabla 2.10.

Tabla 2.10 Propiedades de los Pines de Control

Nombre de la Propiedad	Descripción
DataTerminalReady	Especifica el estado del pin DTR
FlowControl	Especifica el método a utilizar para el control de flujo de datos
PinStatus	Indica el estado de los pines CD, CTS, DSR y RI
RequestToSend	Especifica el estado del pin RTS

a. Señalización por la Presencia de Dispositivos Conectados

A menudo los DTE y DCE usan los pines CD, DSR, RI, y DTR para indicar si una conexión se establece entre los dispositivos del puerto serie. Una vez que la conexión se ha establecido, se puede empezar a escribir o leer datos.

Se puede supervisar el estado de los pines CD, DSR, y RI con la propiedad "PinStatus", en cambio para especificar o supervisar el estado del pin DTR se emplea la propiedad "DataTerminalReady".

b. Control del Flujo de Datos: Handshaking

El control del flujo de datos o handshaking es un método utilizado en la comunicación de un DCE y un DTE para prevenir la pérdida de los datos durante la transmisión. Por ejemplo, si una

computadora puede recibir sólo una cantidad limitada de datos antes de que los procese, este límite al ser alcanzado, puede ser avisado con un signo de handshaking que se transmite al DCE para dejar de enviar datos. Cuando la computadora pueda aceptar más datos, otro signo del handshaking se transmite al DCE para reasumir el envío de datos.
Se puede controlar el flujo de datos usando uno de éstos métodos:

- Handshaking por Hardware.
- Handshaking por Software.

Aunque se podría configurar el dispositivo para ambos tipos de handshaking, MATLAB no puede soportarlo. Se puede especificar la forma de control del flujo de los datos con la propiedad "FlowControl".

- **Control de Flujo de Datos por Hardware.** En este tipo de control, los pines del puerto serie especifican el trabajo del flujo de los datos; en la mayoría de los casos son los pines RTS y CTS, los cuales serán automáticamente manejados por el DTE y DCE. Se puede volver a fijar los valores de CTS y RTS con las propiedades "PinStatus" y "RequestToSend" respectivamente.

  Algunos dispositivos usan el DTR y DSR para fijar el handshaking; sin embargo, estos pines se usan típicamente para indicar que el sistema está listo para comunicación y no para controlar la transmisión de los datos. En MATLAB, el handshaking del hardware siempre usa los pines RTS y CTS.

Si el dispositivo no usa control de flujo de datos por hardware de la manera normal, entonces se podría necesitar configurar la propiedad de "RequestToSend" manualmente, en este caso se debe configurar "FlowControl" a "Ninguno". Si "FlowControl" es por hardware, entonces el valor de "RequestToSend" que se especifica no será ignorado.

- **Control de Flujo de Datos por Software.** Usa caracteres ASCII específicos para controlar el flujo de los datos. Estos caracteres, conocidos como Xon y Xoff (o XON y XOFF), se describe en la tabla 2.11.

Al utilizar control por software, los caracteres de control se envían por la línea de transmisión de la misma forma que los datos regulares. Por consiguiente, se necesitan sólo los pines TD, RD y GND. La desventaja principal de handshaking por software es que no se pueden escribir los caracteres Xon o Xoff mientras los datos numéricos están escribiéndose en el dispositivo; esto es porque los datos numéricos podrían contener un "17H" o un "19H" y sería imposible distinguir entre los caracteres de control de los datos. Sin embargo, sí se puede escribir en ellos mientras el datos está leyéndose asincrónicamente del dispositivo, porque se estarían usando los dos pines TD y RD.

Tabla 2.11.Caracteres para Control de Flujo de Datos por Software

Caracter	Valor Entero	Descripción
Xon	17	Continúa la transmisión de datos
Xoff	19	Pausa en la transmisión de datos

2.2.8 ALMACENAMIENTO DE LA INFORMACIÓN EN EL DISCO

Mientras el objeto del puerto serie se conecta al dispositivo, se puede grabar esta información a un archivo del disco:

- El número de valores escritos en el dispositivo, el número de valores leídos del dispositivo y el tipo de dato de estos valores.
- Los datos escritos en el dispositivo y los datos leídos del dispositivo.
- La información de los eventos.

Al almacenar la información en el disco se proporciona un registro permanente de la sesión con el puerto serie, lo cual es una manera fácil y útil para depurar una aplicación. Se graba información a un archivo de disco con la función "record" . Las propiedades asociadas con ésta función se detallan en la tabla 2.12.

Tabla 2.12 Propiedades del Almacenamiento

Nombre de la Propiedad	Descripción
RecordDetail	Especifica la cantidad de información guardada en un archivo de registro
RecordMode	Especifica que información de datos y eventos está guardada en un archivo o en múltiples archivos
RecordName	Especifica el nombre del archivo
RecordStatus	Indica si la información de datos y eventos serán guardados al archivo de registro

a. Creación de Archivos Múltiples

Cuando se inicia grabando con la función “record”, la propiedad “RecordMode” determina si se crea un nuevo archivo o si la información se añade a un archivo existente.

Se puede configurar “RecordMode” para borrar, añadir o indexar. Si “RecordMode” está en “borrar”, entonces el archivo del registro se borra grabando desde el inicio; si “RecordMode” está en “añadir”, la nueva información se añade al archivo especificado por “RecordName” y si “RecordMode” está en indexar, cada vez se crea un archivo diferente grabando desde el inicio.

b. Especificación del nombre de un Archivo

Se especifica el nombre del archivo con la propiedad “RecordName”, puede también especificar un camino para el directorio. Adicionalmente, si “RecordMode” está en indexar, se siguen estas reglas:

- Los archivos indexados son identificados por un número. Este número precede a la extensión del nombre del archivo y se aumenta en 1 para los archivos de registro sucesivos.
- Si ningún número se especifica como parte del filename inicial, entonces el primer archivo no tiene un número asociado con él.
- “RecordName” se actualiza después de que el archivo se cierra.

- Si el archivo especificado ya existe, entonces se sobrescribe la información.

c. Formato del Archivo

El archivo creado es un archivo de ASCII que contiene la información de una o más sesiones con el puerto serie. Se especifica la cantidad de información guardada en el archivo con la propiedad “RecordDetail”.

“RecordDetail” puede ser compacto o expandido. Un archivo compacto contiene el número de valores escritos en el dispositivo, el número de valores leídos del dispositivo, el tipo de dato y la información de los eventos. Un archivo del registro expandido en cambio contiene la información precedente, así como los datos que se transfirieron desde y hasta el dispositivo.

Datos binarios con precisión uchar, schar, (u)int8, (u)int16 o (u)int32 se graban usando formato hexadecimal; por ejemplo, si el valor entero 255 se lee del instrumento como un entero de 16-bits, el dato almacenado es 00FF. Los datos en punto flotante con simple y doble precisión se graban como valores decimales que usan el formato %g y como valores hexadecimal usando el formato especificado por la IEEE Standard 754-1985 para Aritmética Binaria
de punto flotante.

El formato para punto flotante de la IEEE incluye tres componentes: el bit de signo, el campo del exponente, y el campo del valor. Valores de punto flotante con simple precisión constan de 32 bits; el valor se da por:

$$value = (-1)^{sign}(2^{\exp-127})(1.significand) \quad \textbf{(Ec. 2.1)}$$

Los valores de punto flotante con doble precisión consisten de 64 bits; el valor se da por:

$$value = (-1)^{sign}(2^{\exp-1023})(1.significand) \quad \textbf{(Ec. 2.2)}$$

El componente del formato de punto flotante, para simple y doble precisión se dan en la tabla 2.13.

Tabla 2.13 Formato de datos según la IEEE

Componente	Bits para Precisión Simple	Bits para Precisión Doble
signo	1	1
exponente	2 – 9	2 – 12
significativo	10 - 32	13 –64

El bit 1 (signo) va a la izquierda de los bits que se guardan en el archivo.

2.2.9 ALMACENAMIENTO Y RECUPERACIÓN

Se pueden grabar objetos del puerto serie en un archivo MAT, así como se trabaja con la variables en workspace usando el comando “save”. Por ejemplo, suponga que se crea un objeto del puerto y se lo asocia con el puerto COM1, se configuran los valores para las propiedades y se realizan operaciones de escritura lectura.

```
s = serial('COM1');

s.BaudRate = 19200;
s.Tag = 'My serial object';

fopen(s)

fprintf (s, '*IDN?')

out =  fscanf (s);
```

Para almacenar el objeto del puerto serie y los datos que se leyeron del dispositivo en el archivo MAT, se pondría:

```
save myserial s out
```

Se puede volver a crear a **s** y sacarlo fuera en el workspace usando el comando “load”.

```
load myserial
```

Los valores de la propiedades de sólo lectura se restauran a los predefinidos en el momento de la carga.

2.2.10 DESCONEXIÓN Y BORRADO

Cuando ya no se necesita el objeto del puerto serie, debe desconectarlo del dispositivo y borrarlo del ambiente de MATLAB quitando el objeto de la memoria y del "workspace". Se deben seguir los pasos que se detallan a continuación para cerrar la sesión con el puerto serie.

a. Desconexión del Objeto del Puerto Serie

Cuando ya no necesita comunicarse con el dispositivo, debe desconectar el objeto del puerto serie con la función "fclose".

```
fclose(s)
```

Se puede examinar la propiedad "Status" para verificar que el objeto del puerto serie está desconectado del dispositivo. Por ejemplo:

```
s.Status
ans =
closed
```

Después de que se emite la función "fclose", el puerto serie asociado con **s** está disponible. Se puede conectar otro objeto del puerto serie usando "fopen".

b. Borrado del Ambiente de Matlab

Cuando ya no necesita el objeto del puerto serie, debe quitarlo de la
memoria con la función “delete”, así:

delete(s)

Antes de usar la función, se debe desconectar el objeto del puerto serie del dispositivo con la función del “fclose”.

Un objeto del puerto serie borrado se invalida, por lo que no puede conectárselo al dispositivo. En este caso, se debe quitar el objeto del workspace de Matlab. Para quitar objetos del puerto serie y otras variables del workspace de MATLAB, use el comando “clear”.

Si se utiliza el comando “clear” con un objeto del puerto serie que todavía está conectado a un dispositivo, el objeto es removido del “workspace” pero mantiene conexión con el dispositivo. Se pueden restaurar objetos borrados de Matlab con la función “instrfind”∀[11].

2.3 DESARROLLO DE INTERFASES GRÁFICAS

∀Una interfase gráfica del usuario (GUI) es una interfase construida con objetos gráficos — los componentes del GUI— como botones, campos del texto, deslizadores, y menús.

El MATLAB mantiene un juego de herramientas para crear GUIs. Estas herramientas simplifican el proceso de poner y programar interfaces gráficas.

[11] Traducciones de Manuales del MATLAB

Cuando se abre una GUI en GUIDE, se despliega en el "Layout Editor" el tablero de mando con todas las herramientas. La figura 2.5 presenta esta ventana.

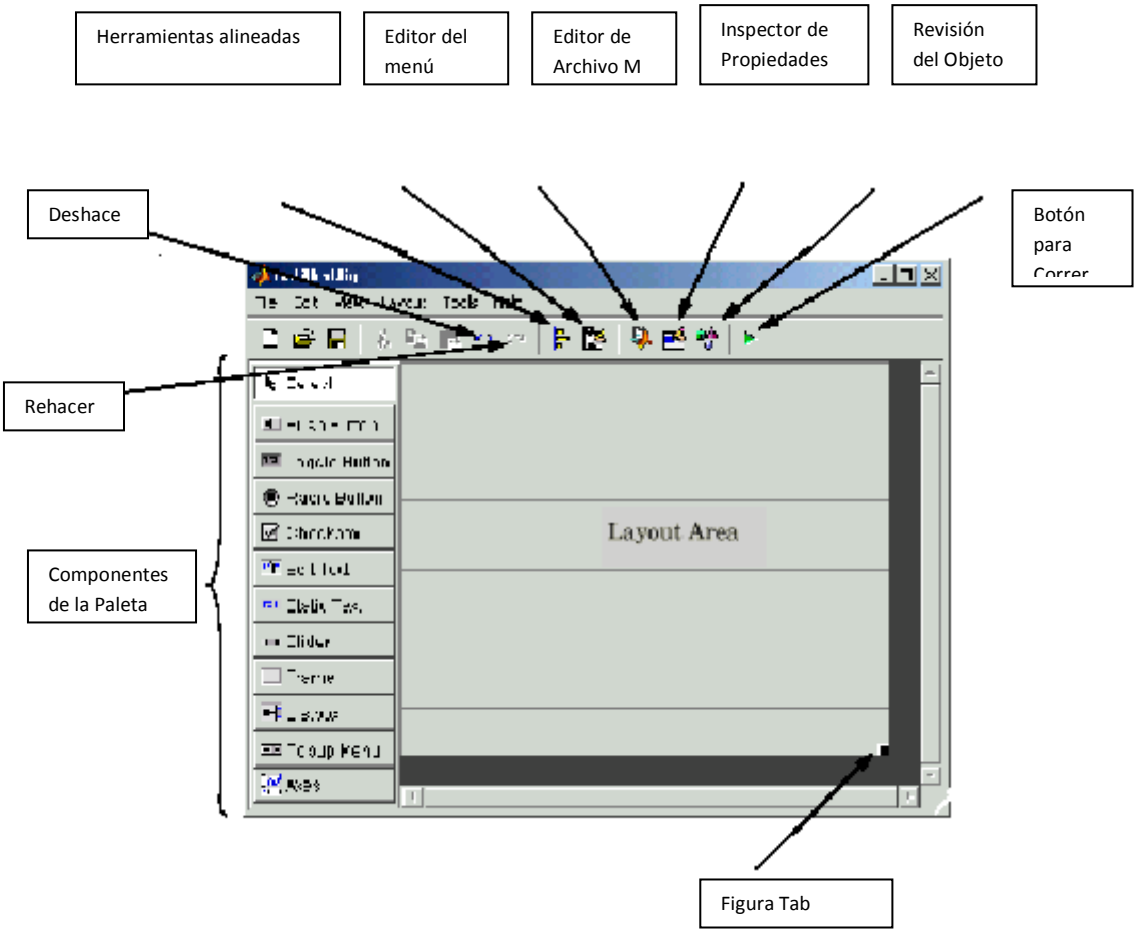


Figura 2.5 Ventana del "Layout Editor".

Crear una GUI involucra dos tareas básicas: Colocar los componentes de GUI y la Programación, que consiste en escribir las órdenes para los componentes de GUI. Las figuras 2.6 y 2.7, presentan ejemplos de estas tareas.

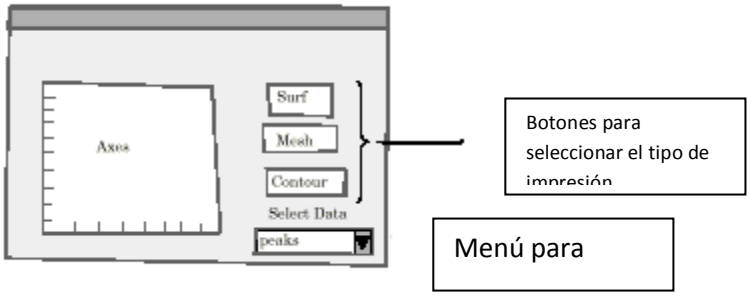


Figura 2.6. Ejemplo del diseño externo de una GUI

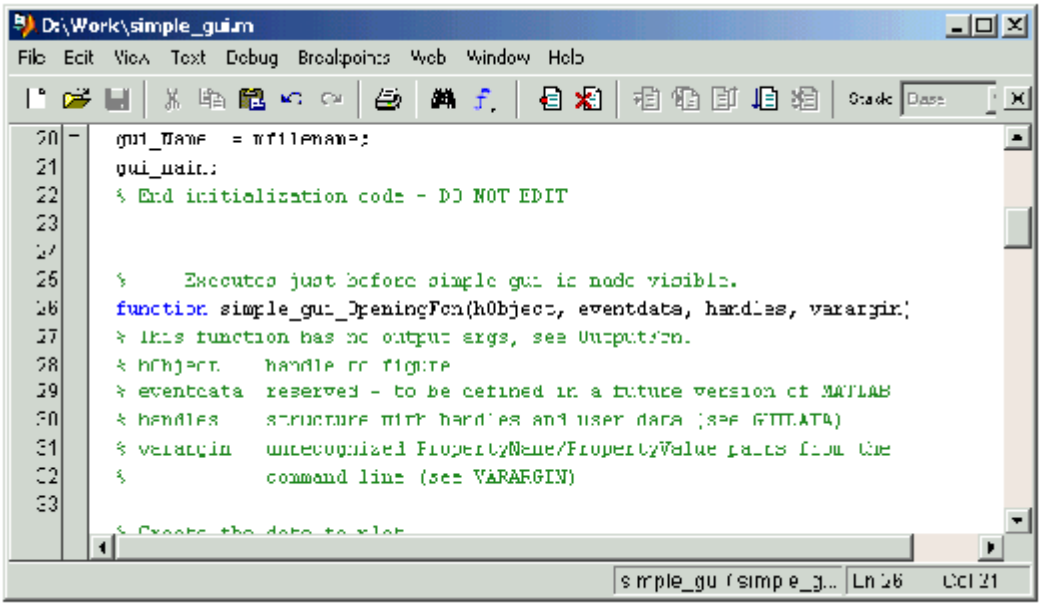


Figura 2.7. Ejemplo de la programación de una GUI

La función GUIDE guarda GUIs en dos archivos que se generan automáticamente antes de ejecutar (run) la aplicación:

- FIG-file: Es un archivo con extensión .fig que contiene una descripción completa de los esquemas del GUI y los componentes del mismo: botones, menús, etc. Cuando se hacen cambios al esquema de la GUI, estos se almacenan en el FIG-file.

- M-file: Es un archivo con extensión .m que contiene el código que controla la GUI, incluso las componentes de los callbacks∀[12].

El trabajo con GUIDE de Matlab, es una tarea similar a la labor que se realiza con cualquier lenguaje de programación gráfico como por ejemplo el Visual Basic.

[12] MATLABHelp

CAPÍTULO III

MICROCONTROLADOR PIC16F877

3.1 GENERALIDADES

El utilizar un microcontrolador de la fábrica Microchip, denominados PIC's no significa que esta familia sea la "mejor", pero considerando el momento actual, comparando los parámetros fundamentales con los modelos comerciales de otros fabricantes y las aplicaciones más habituales a las que se destinan los microcontroladores, se puede indicar que casi en un 90 % de los casos la elección de una versión adecuada de PIC es la mejor solución. Sin embargo, otras familias de microcontroladores son más eficaces en aplicaciones concretas, especialmente si predomina una característica especial.

Pero de todas formas, en la actualidad los PIC tienen "algo" que fascina a los diseñadores. Pueden ser la velocidad, el precio, la facilidad de uso, la información, las herramientas de apoyo, quizás un poco de todo es lo que produce esa imagen de sencillez y utilidad; es muy posible que mañana otra familia de microcontroladores le arrebate ese "algo", es la ley del mercado y la competencia.

3.2 ARQUITECTURA

∀Las características más representativas de los PIC son las que se detallan a continuación:

- La arquitectura del procesador sigue el modelo Harvard

En esta arquitectura, el CPU se conecta de forma independiente y con buses distintos con la memoria de instrucciones y con la de datos. Figura 3.1.

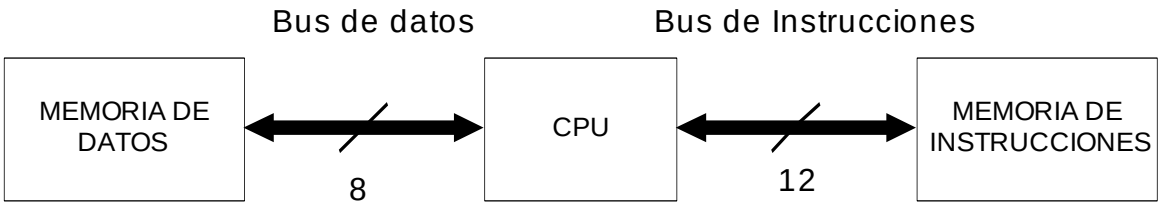


Figura 3.1 Arquitectura Harvard

La arquitectura Harvard permite a el CPU acceder simultáneamente a las dos memorias. Además, propicia numerosas ventajas al funcionamiento del sistema como se irán describiendo.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontrarán físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).

Los microcontroladores PIC 16X5X, 16XXX y 17XXX poseen arquitectura Harvard, con una memoria de datos de 8 bits, y una memoria de programa que, según el modelo, puede ser de 12 bits para los 16X5X, 14 bits para los 16XXX y 16 bits para los 17XXX.

- 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 períodos de reloj). Figura 3.2.

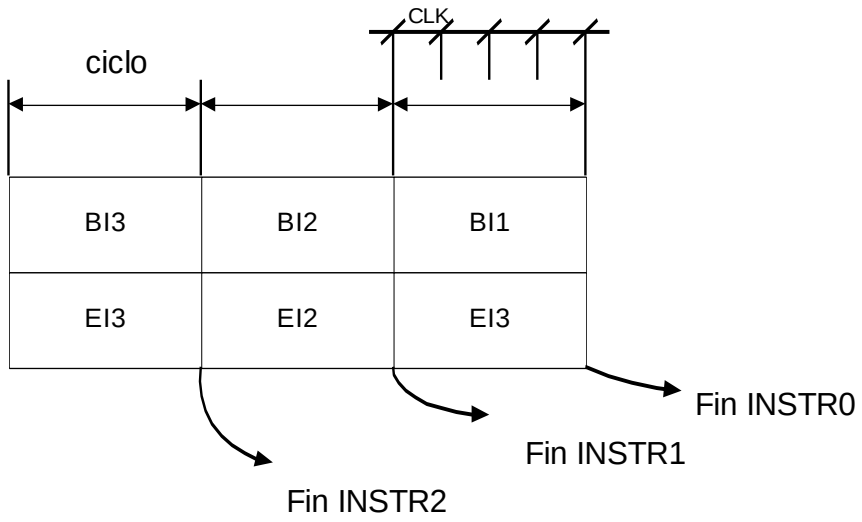


Figura 3.2 Segmentación

La segmentación permite al procesador ejecutar cada instrucción en un ciclo de instrucción equivalente a cuatro períodos de reloj. En cada ciclo se realiza la búsqueda de una instrucción y la ejecución de la anterior.

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

- El formato de todas las instrucciones tiene 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 y más las de la gama alta. Esta característica es muy ventajosa en la optimización de la memoria de instrucciones y facilita enormemente la construcción de ensambladores y compiladores.

- Procesador RISC (Computador de Juego 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.

- Todas las instrucciones son ortogonales.

Cualquier instrucción puede manejar cualquier elemento de la arquitectura como fuente o como destino.

- Arquitectura basada en un banco de registros.

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

- Diversidad de modelos de microcontroladores con prestaciones y recursos diferentes.

La gran variedad de modelos de microcontroladores PIC permite que el usuario pueda seleccionar el más conveniente para su proyecto.

- Herramientas de soporte potentes y económicas

La empresa Microchip y otras que utilizan los PIC ponen a disposición de los usuarios numerosas herramientas para desarrollar hardware y software. Son muy abundantes los programadores, los simuladores software, los emuladores en tiempo real, ensambladores, Compiladores C, Intérpretes y Compiladores BASIC, etc $\forall^{13}$.

Con base a lo anterior, se escoge el microcontrolador PIC16F877 (perteneciente a la gama media) para realizar la aplicación porque cuenta con múltiples recursos, especialmente en cuanto a comunicación serie - paralelo se refiere y la inclusión de conversores A/D.

Estos recursos se detallan a continuación:

RECURSOS FUNDAMENTALES

- Procesador de arquitectura RISC avanzada
- Juego de 35 instrucciones con 14 bits de longitud. Todas ellas se ejecutan en un ciclo de instrucción, menos las de salto que tardan dos.

- Frecuencia de hasta 20 MHz.
- Hasta 8 K palabras de 14 bits para la memoria de Programa, tipo Flash.
- Hasta 368 bytes de memoria de Datos RAM.
- Hasta 256 bytes de memoria de Datos EEPROM.
- Hasta 14 fuentes de interrupción internas y externas.

[13]http://www.Microchip.com

- Pila con 8 niveles.
- Modos de direccionamiento directo, indirecto y relativo.
- Perro Guardián (WDT).
- Código de protección programable.
- Modo SLEEP de bajo consumo.
- Programación serie mediante dos pines.
- Voltaje de alimentación comprendido entre 2 y 5.5 V.
- Bajo consumo (menos de 2 mA a 5 V y 5 MHz).

DISPOSITIVOS PERIFÉRICOS

- Timer 0: temporizador – contador de 8 bits con predivisor de 8 bits.
- Timer 1: temporizador – contador de 16 bits con predivisor.
- Timer 2: temporizador – contador de 8 bits con predivisor y postdivisor.
- Dos módulos de Captura – Comparación – PWM.
- Conversor A/D de 10 bits.
- Puerto Serie Síncrono (SSP) con SPI e I2C.
- USART
- Puerta Paralela Esclava (PSP).

En la figura 3.3, se especifica la distribución de pines PIC16F877.

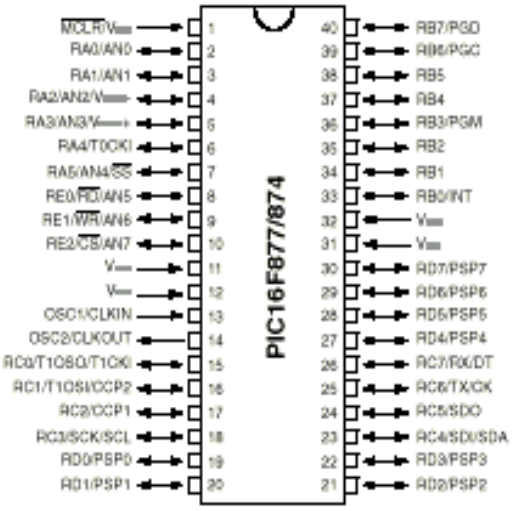


Figura 3.3 Distribución de pines del PIC 16F877

3.3 REGISTROS Y REPERTORIO DE INSTRUCCIONES

Los registros de trabajo y las instrucciones necesarias para el manejo del PIC16F877 se indicarán a continuación, pero en las secciones siguientes se detallarán aquellos más relacionados con la comunicación, objetivo central del proyecto.

"La figura 3.4 presenta, en bloques, el esquema general del microcontrolador PIC16F877, en el cual se evidencian los recursos y sus conexiones internas, información importante para la programación adecuada del integrado.

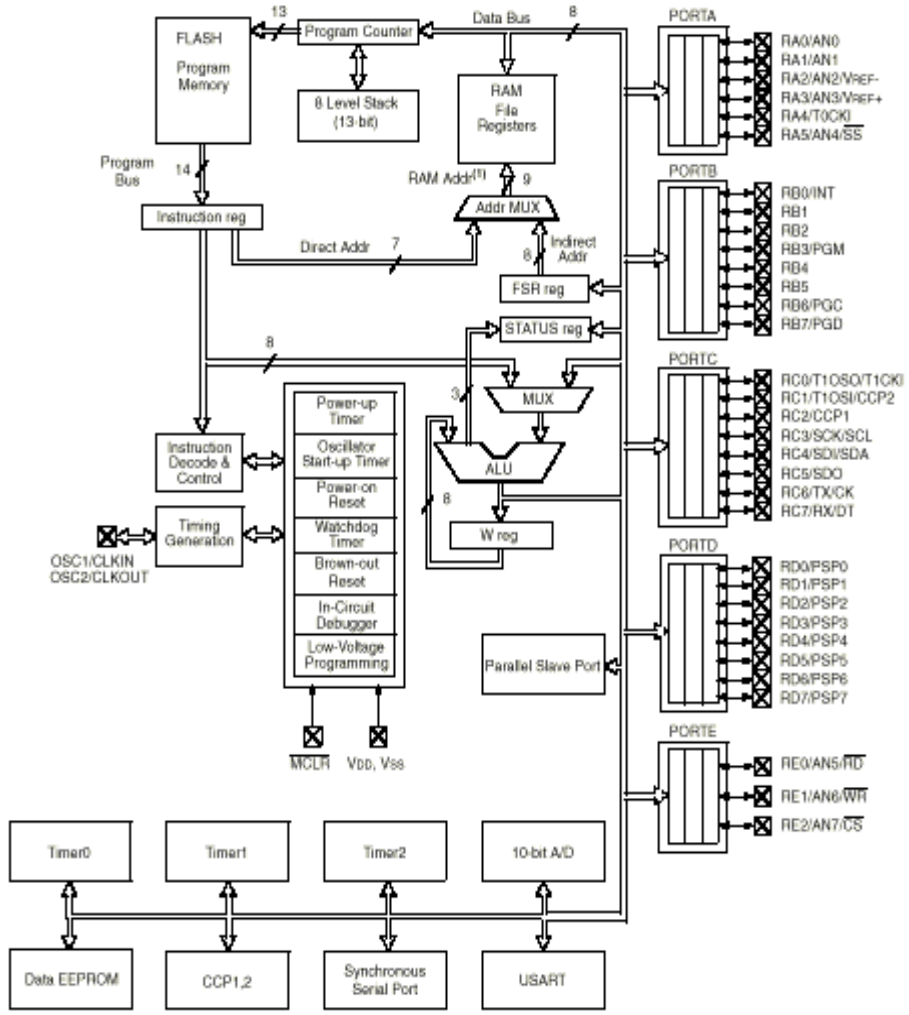


Figura 3.4 Esquema general del PIC 16F877

La figura 3.5 indica la organización de la memoria interna de datos, en la que se encuentran los registros del PIC 16F877, se detalla la localidad y el banco en el que encuentra, esto último regulado por dos bits del registro de Estado.

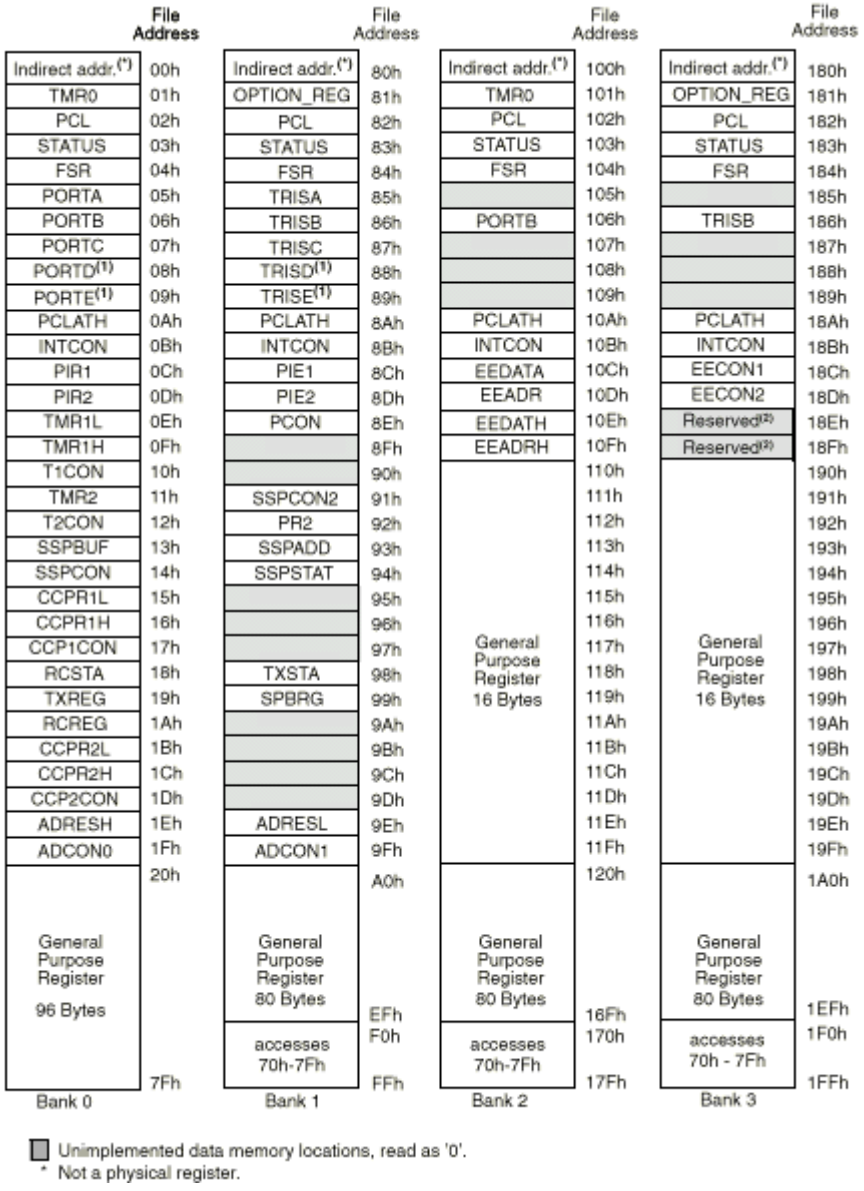


Figura 3.5 Organización de la memoria interna de datos

La tabla 3.1 detalla las funciones de los diferentes bits de cada uno de los registros especificados en la figura 3.5.

Tabla 3.1 Bits de los Registros del 16F877

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

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

Tabla 3.1 Bits de los Registros del 16F877 (Continuación)

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	—

La tabla 3.2 reúne el repertorio de instrucciones, el cual es el mismo de todos los PICs de la gama media$\forall^{14}$.

3.4 COMUNICACIÓN SERIE Y PARALELO

3.4.1. PUERTAS DE ENTRADA SALIDA

Dispone de 5 puertas o puertos (A, B, C, D y E). Todas las líneas de estas puertas son multifuncionales; es decir, realizan diversas funciones según están programadas. Sin embargo, todas ellas tienen la capacidad de trabajar como líneas de entrada/salidas digitales.

[14]http://www.Microchip.com

Tabla 3.2 Instrucciones del PIC 16F877

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

a. Puerta A

∀Dispone de 6 líneas, denominadas RA0, RA1, RA2, RA3, RA4 y RA5. Son bidireccionales y su sentido queda configurado según la programación de los bits del registro TRISA. Si el bit 0 del registro TRISA

se pone a 1, la línea 0 (RA0) de la puerta A funciona como entrada, si se pone en 1 funciona como salida .

Al leer el registro PORTA de la Puerta A, se lee el estado de las patitas, que es el que se halla escrito en la báscula de datos de la figura 3.6. La escritura entraña una operación de "lectura – modificación – escritura", o sea, se leen las patitas, luego se modifica su valor y finalmente se escribe en la báscula de datos.

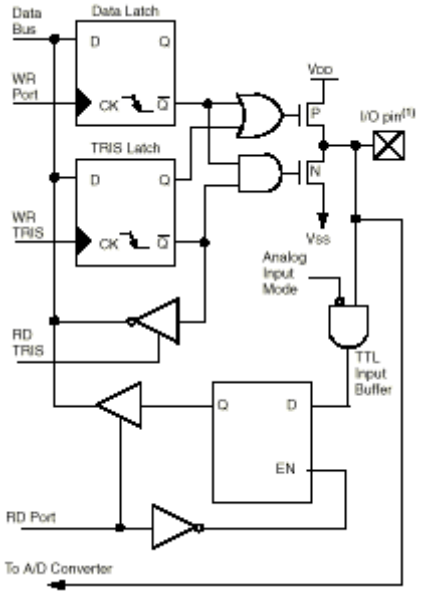


Figura 3.6 Esquema de conexión de las patitas RA0 – RA3 y RA5

Los pines RA0/AN0, RA1/AN1 y RA2/AN2, además de líneas de entrada/salida digitales también pueden actuar como los canales 0,1 y 2 por los que se puede aplicar una señal analógica al conversor A/D. Los pines RA3/AN3/V_{REF+} también puede actuar como entrada de la Tensión de Referencia para los periféricos que la precisan. El pin RA4/TOCKI actúa como E/S digital y como entrada en la señal de reloj para el Timer 0. Por último, el pin RA5/AN4/SS# tiene multiplexadas

tres funciones: E/S digital, canal 4 para el conversor A/D y selección del modo esclavo cuando trabaja con la comunicación serie síncrona.

Para seleccionar si las líneas de la Puerta A van a trabajar como entrada/salida digitales o como canales de entrada para el conversor A/D, hay que escribir el valor adecuado sobre el registro ADCON1. La tabla 3.3 recoge las características y bits más importantes de los registros que manejan la Puerta A.

Tabla 3.3 Registros para la Puerta A

DIREC-CIÓN	NOMBRE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	POR BOR	RESET
05h	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0	-0x000	-0u0000
85h	TRISA	-	-	REGI	STR	0 DE	CON	FIGU	RAR	-111111	-111111
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	-0-0000	-0-0000

NOTAS: POR (Power On Reset), BOR (Brown Out Reset). La x significa desconocido, la u que no cambia y un – que no está implementado y se lee como 0.

El bit ADFM selecciona el formato del resultado de la conversión. Si vale 1, el resultado está justificado en el registro ADRESH, que tiene sus 6 bits de más peso a 0; mientras que si vale 0 la justificación se realiza sobre el registro ADRESL, que tiene sus 6 bits de menos peso a 0. Esto significa que los 16 bits que forman la concatenación de ADRESH:ADRESL unas veces tiene a 0 los 6 bits de más peso y otras los 6 bits de menos peso (alineación a la derecha o a la izquierda).

Los restantes cuatro bits (PCFG3 – 0) de ADCON1 se usan para configurar los pines de los canales de entrada al conversor como analógicas o como entradas/salidas digitales.

b. Puerta B

Dispone de 8 líneas bidireccionales cuya función se elige mediante la programación del TRISB. Todas las patitas de la puerta B disponen de una resistencia interna de pull-up al positivo de la alimentación, que queda conectada cuando el bit RBPU#, que es el bit 7 del registro OPTION, tiene valor 0. La resistencia de pull-up, que es un transistor CMOS tipo P, como se aprecia en la figura 3.7, se conecta automáticamente siempre que la línea esté configurada como salida. Cuando se produce un Reset por conexión de la alimentación (POR) se desconectan todas las resistencias pull – up.

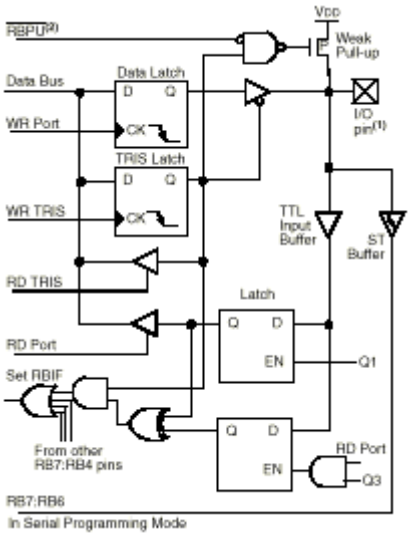


Figura 3.7 Conexión interna de las patitas RB<7-4> de la puerta B

Las líneas RB<7-4> pueden programarse para generar una interrupción cuando una de ellas cambia de estado. Se deben configurar como entradas y el valor que se introduce por ellas se compara con el anterior para, si no coinciden, generar una interrupción, siempre que lo autorice el bit de permiso. En cualquier caso, el señalizador RBIF, que es el bit 0 del registro INTCON, se pondrá a 1.

El pin RB0/INT también puede programarse como petición de interrupción externa, si se autoriza con el correspondiente bit de permiso, que también está ubicado en INTCON. La tabla 3.4 muestra algunas características de los registros que manejan la puerta B.

Tabla 3.4 Registros para la Puerta B

DIREC-CIÓN	NOMBRE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	POR BOR	RESET

06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	Uuuu uuuu
86h 186h	TRISB	-	-	REGI	STR	0 DE	CON	FIGU	RAR	-1111111	-1111111
81 181h	OPTION – REG	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	11111111	11111111

c. Puerta C

Consta de 8 líneas bidireccionales cuyo sentido se configura mediante el registro TRISC. Todos los pines de esta puerta tienen multiplexadas diferentes funciones, las mismas que se indican a continuación.

- RC0/T1OSO/T1CK1: Esta línea puede actuar como entrada/salida digital, como salida del Timer 1 o como entrada de impulsos para el Timer 1.
- RC1/T1OSI/CCP2: Entrada/salida digital, entrada al oscilador del Timer 1, entrada del módulo de captura 2, salida del comparador 2 y salida del PWM 2.
- RC2/CCP1: Entrada/salida digital, entrada captura 1, salida del comparador 1 y salida PWM 1.
- RC3/SCK/SCL: Entrada/salida digital, señal de reloj en modo SPI y señal de reloj en modo I2C.
- RC4/SDI/SDA: Entrada/salida digital, entrada de datos en modo SPI y línea de datos en modo I2C.
- RC5/SDO: Entrada/salida digital y salida de datos en modo SPI.

- RC6/TX/CK: Entrada/salida digital, línea de trasmisión en USART y señal de reloj síncrona en transmisión serie.
- RC7/RX/DT Entrada/salida, línea de recepción del USART y línea de datos en transmisión serie síncrona.

d. Puerta D

Esta puerta de 8 líneas bidireccionales sólo la tienen los PIC16F87X encapsulados de 40 pines. Ocupa la dirección 08h, mientras que su registro de configuración TRISD ocupa la dirección 88h. Todas las patitas disponen en su entrada de un Disparador Schmitt.

Si un circuito tiene entradas TTL normales, las señales de entrada que los impulsan deben observar transiciones relativamente rápidas para lograr una operación confiable, caso contrario puede presentarse oscilaciones en las salidas del mismo. En cambio, un circuito disparador de Schmitt produce transiciones de salida muy rápidas independientes de los tiempos de transición de entrada.

Además de usarse como líneas de entrada/salida digitales normales los pines de la puerta D, implementan una puerta paralela esclava de 8 líneas (PSP) que sirve para permitir la comunicación en paralelo con otros elementos del sistema.

Los pines se denominan RD0/PSP0-RD7/PSP7 y para que funcionen como puerta de comunicación esclava en paralelo es preciso poner el bit PSPMODE = 1. Este bit es el 4 del registro TRISE.

e. Puerta E

Ocupa la dirección 09h y sólo la tienen los PIC16F87X con 40 patitas. Dispone de 3 pines multifunción, que se configuran como entrada o salida, según el valor de los tres bits de menos peso del registro TRISE, que está ubicado en la dirección 89h.

- RE0/RD#/AN5: Entrada/salida digital, señal de lectura en el modo de puerta paralela esclava y canal 5 del conversor A/D.
- RE1/WR#/AN6: Entrada/salida digital, señal de escritura en modo PSP y canal 6 del conversor A/D.
- RE2/CS#/AN7: Entrada/salida digital, selección de chip en el modo PSP y canal 7 del conversor A/D.

La PSP actúa como un puerto de comunicación en paralelo de 8 bits y para su activación hay que poner el bit PSPMODE a 1. Dicho bit es el 4 del registro TRISE. Además de las 8 líneas de transferencia de datos, se precisan 3 señales de control, que determinan si la operación es de lectura, de escritura y de permiso de funcionamiento (RD#, WR# y CS#). Estas tres líneas de control están implementadas en la puerta E$\forall$[15].

[15]Microcontroladores PIC16F87x. Diseño práctico y aplicaciones – Ángulo, Romero y Ángulo, pags: 67 – 72

3.4.2. PUERTO SERIE SÍNCRONO (MSSP)

∀Se trata de un periférico diseñado para soportar una interfaz serie síncrono que resulta muy eficiente para la comunicación del microcontrolador con dispositivos tales como displays, EEPROM, ADC, etc. Tiene dos modos de trabajo:

- Interfaz Serie de Periféricos (SPI)
- Interfaz Inter-Circuitos (I^2C)

a. Modo SPI

Sirve para conectar varios microcontroladores de la misma o diferentes familias, bajo el formato “maestro-esclavo”, siempre que dispongan de un interfaz compatible.

En este modo se pueden emplear 3 o 4 señales de control: Salida de Datos (SDO), Entrada de Datos (SDI), Reloj (SCK) y Selección de Esclavo (SS).

Dichas señales corresponden a los pines RC5, RC4, RC3 y RA5 respectivamente. Cada una de las señales debe programarse como entrada o salida según su condición, utilizando los bits de los registros TRIS. Cualquier función del modo SPI queda anulada poniendo con el valor opuesto a su condición el bit correspondiente de TRIS. Por ejemplo, si solo se quiere recibir datos, se programa el pin que soporta a SDO como entrada y así se anula su función. Figura 3.8.

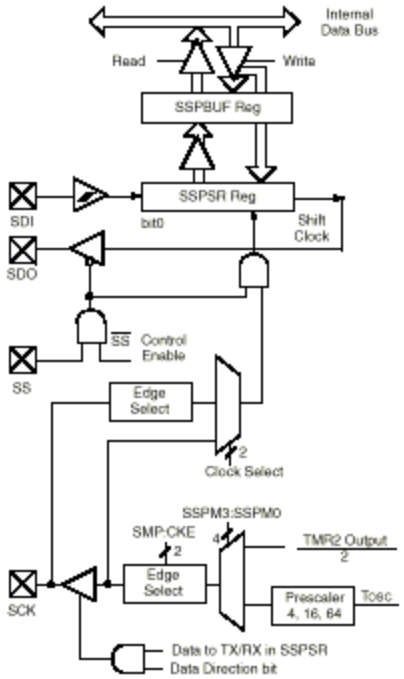


Figura 3.8 Modo SPI

Con el registro de control SSPCON se eligen las diferentes opciones de trabajo:

Modo Master (SCK es salida), Modo Esclavo (SCK es entrada), tipo de flanco de reloj, velocidad de SCK en Modo Master, etc.

Cuando se recibe un dato útil, este se introduce en serie en SSPSR y pasa a SSPBUF en paralelo. El dato a transmitirse se deposita en SSPBUF y de aquí pasa a SSPSR. Se puede recibir y transmitir datos simultáneamente. SSPSR es un registro de desplazamiento que funciona serie/paralelo/serie.

Cuando se acaba de transmitir o recibir un dato completo se activa el bit BF (Buffer Lleno) del registro SSPSTAT. También lo hace el señalizador SSPIF y si el bit de permiso esta activado se genera una interrupción.

Cuando se recibe un dato durante una transmisión se ignora y se activa el bit WCOL que indica que se ha producido una “colisión”.

En el caso que se reciba un nuevo dato en SSPSBUF sin haber leído el anterior se genera un error de desbordamiento.

b. Modo I^2C

Este tipo de interfaz serie ha sido desarrollado por Phillips y utiliza solo dos hilos trenzados y una masa común para la interconexión de los diversos dispositivos, que han tenido que ser diseñados para soportar este protocolo, asegurando una gran fiabilidad en la comunicación que llega a tolerar una velocidad máxima de 400 Kbps. Es capaz de interconectar hasta 128 dispositivos situados a gran distancia, por lo que resulta muy usado en edificios inteligentes, control de distribuciones de electricidad, agua y gas, piscifactorías, etc.

El master es el que inicia y termina la transferencia general y provee de la señal de reloj. El esclavo ("slave") es el dispositivo direccionado por el master, mediante 7 bits, lo que limita el número de componentes a 128.

El inicio de la transmisión se determina con el bit de inicio (S) y el final con otro bit de stop (P). El bus serie de 2 hilos utiliza uno de ellos para transferir datos (SDA) y el otro para la señal de reloj (SCL).

En el protocolo I^2C cada dispositivo tiene asignada una dirección de 7 o de 10 bits que envía el master cuando comienza la transferencia con

uno de ellos. Tras la dirección se añade el bit de recepción/transmisión o lectura/escritura (R/W). Los datos se transmiten con longitud byte y al finalizar cada uno se inserta un bit de reconocimiento ACK. Debe existir un modulo de arbitraje que gestione que solo hay un maestro en cada instante sobre el bus compartido.

La figura 3.9 muestra un esquema interno de funcionamiento del interfaz I^2C.

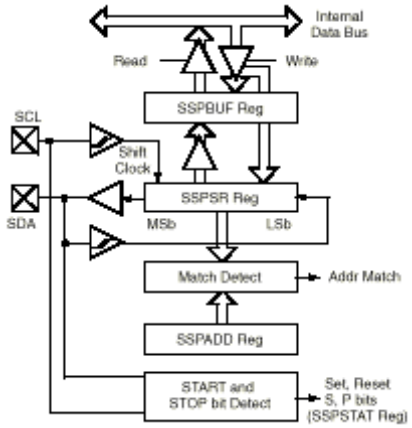
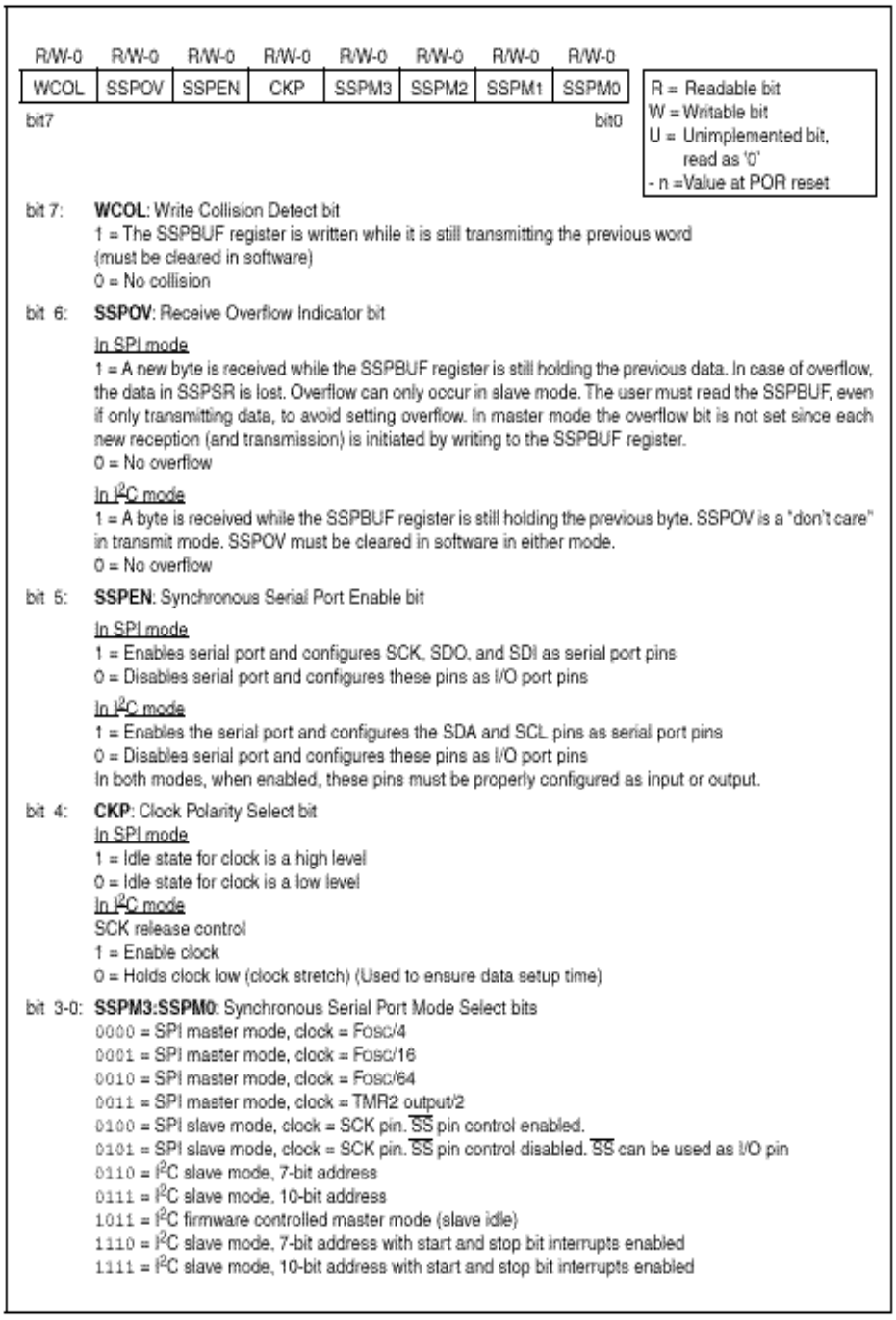


Figura 3.9 Modo I^2C

SSPBUF es el registro donde se almacena el byte a transmitir o el que se recibe. SSPSR es el registro desplazamiento serie de la línea E/S. SSPADD es el registro de direcciones que identifica el dispositivo (modo esclavo) o que lo direcciona (modo master). El registro de control SSPCON selecciona las diversas funciones del modo I^2C. Figura 3.10.



R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n =Value at POR reset

bit 7: **WCOL**: Write Collision Detect bit
1 = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6: **SSPOV**: Receive Overflow Indicator bit

In SPI mode
1 = A new byte is received while the SSPBUF register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in slave mode. The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow. In master mode the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.
0 = No overflow

In I^2C mode
1 = A byte is received while the SSPBUF register is still holding the previous byte. SSPOV is a "don't care" in transmit mode. SSPOV must be cleared in software in either mode.
0 = No overflow

bit 5: **SSPEN**: Synchronous Serial Port Enable bit

In SPI mode
1 = Enables serial port and configures SCK, SDO, and SDI as serial port pins
0 = Disables serial port and configures these pins as I/O port pins

In I^2C mode
1 = Enables the serial port and configures the SDA and SCL pins as serial port pins
0 = Disables serial port and configures these pins as I/O port pins
In both modes, when enabled, these pins must be properly configured as input or output.

bit 4: **CKP**: Clock Polarity Select bit
In SPI mode
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level
In I^2C mode
SCK release control
1 = Enable clock
0 = Holds clock low (clock stretch) (Used to ensure data setup time)

bit 3-0: **SSPM3:SSPM0**: Synchronous Serial Port Mode Select bits
0000 = SPI master mode, clock = FOSC/4
0001 = SPI master mode, clock = FOSC/16
0010 = SPI master mode, clock = FOSC/64
0011 = SPI master mode, clock = TMR2 output/2
0100 = SPI slave mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0101 = SPI slave mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin
0110 = I^2C slave mode, 7-bit address
0111 = I^2C slave mode, 10-bit address
1011 = I^2C firmware controlled master mode (slave idle)
1110 = I^2C slave mode, 7-bit address with start and stop bit interrupts enabled
1111 = I^2C slave mode, 10-bit address with start and stop bit interrupts enabled

Figura 3.10 Registro SSPCON

Cada vez que se detecta un bit de inicio o un bit de stop es posible que se active el señalizador SSPIF y en el caso de estar también activado el bit de permiso correspondiente generar una interrupción.

3.4.3 EL USART: TRANSMISOR/RECEPTOR, SÍNCRONO/ ASÍNCRONO SERIE

El USART programable es llamado también SCI. Se puede configurar de dos modo diferentes:

Asíncrono (full-duplex)

La comunicación es bidireccional. La patita RC6/Tx/CK actúa como línea de transmisión y la RC7/Rx/DT como línea de recepción. Cada dato lleva un bit de inicio y otro de stop.

Síncrono (semiduplex)

Comunicación unidireccional. Una sola línea para los datos que se implementan sobre el pin RC7/Rx/DT. En el modo master la señal de reloj sale por la patita RC6/Tx/CK. En el modo esclavo (“slave”) entra por ella.

En ambos modo los datos pueden ser de 8 o 9 bits, pudiendo emplear el noveno como bit de paridad, transmitiéndose o recibiéndose por el bit <0> de RXSTA y/o RCSTA.

El registro especifico TXSTA actúa como registro de estado y control del transmisor y el RCSTA hace lo mismo para el receptor.

Los baudios se establecen por el valor cargado en el registro SPBRG y el bit BRGH del registro TXSTA, con el que se puede elegir la velocidad alta (1) o baja (0) en el modo asíncrono.

$$BAUDIOS = \frac{Fosc}{n(x+1)} \qquad \textbf{(Ec. 3.1)}$$

donde:

n = 4 en el modo síncrono

n = 16 en el modo asíncrono de alta velocidad

n = 64 en el modo asíncrono de baja velocidad

x = valor cargado en el registro SPBRG

siendo:

$$x = \frac{Fosc / Baudios - n}{n} \qquad \textbf{(Ec. 3.2)}$$

Mediante la programación de los bits del registro TXSTA y RCSTA se configura el modo de trabajo. Así, SPEN configura RC7/Rx y RC6/Tx como líneas de comunicación serie. El transmisor se activa con el bit TxEN. El dato a transmitir se carga en TxREG y luego pasa al registro transmisor TSR, cuando se haya transmitido el bit de stop del dato anterior. Entonces se activa el señalizador TxIF y si el bit de permiso esta activado se produce una interrupción.

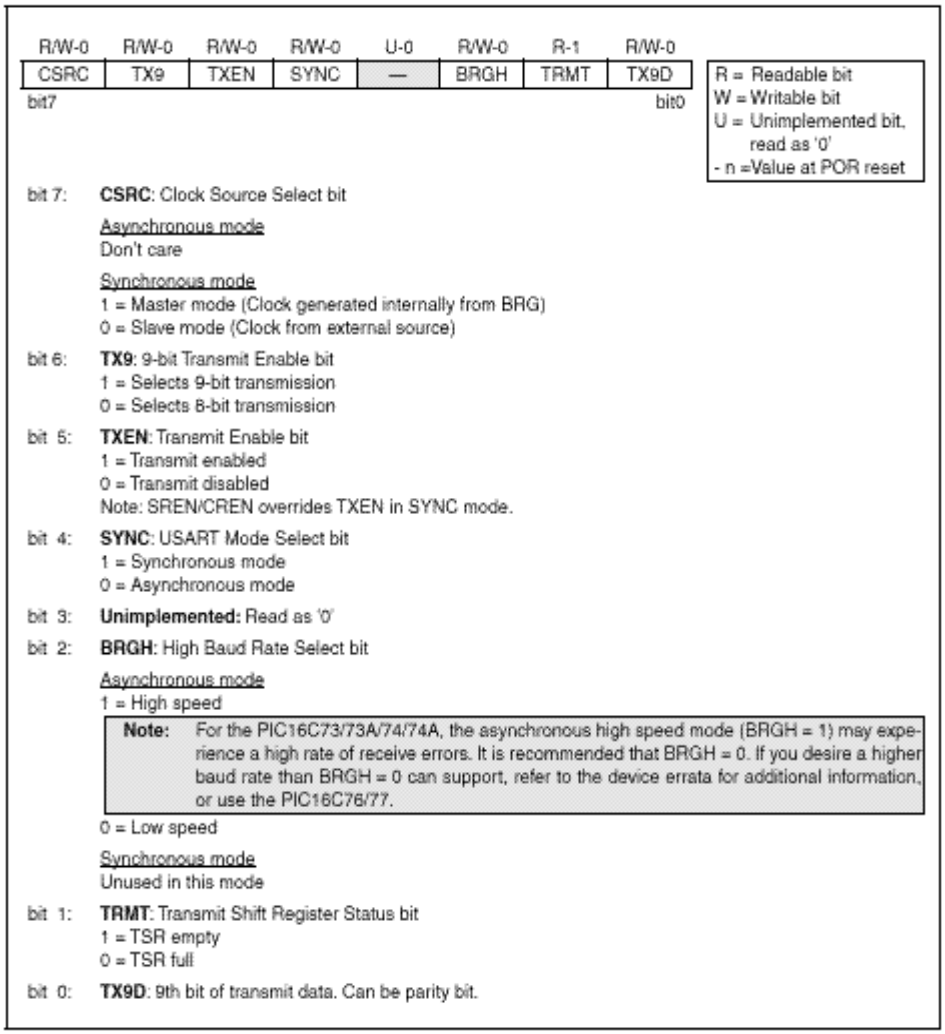
Activando Tx8/9 se inserta el noveno bit almacenado en el bit <0> (TxD8) de TXSTA. El bit TRMT indica si el transmisor esta vacío o no. El dato se recibe por RSR y cuando se completa se pasa al registro RCREG para su posterior lectura, activándose el señalizador RCIF y si acaso la interrupción.

Si se activa el bit RC8/9del RCSTA el noveno bit se deposita en el bit <0> (RCD8) del RCSTA. Los bits OERR y FERR indican error de desbordamiento y de trama, respectivamente.

En las figuras 3.11 y 3.12 se ofrece la asignación de funciones de los bits de los registros TXSTA y RCSTA que gobiernan al receptor y transmisor asíncronos, respectivamente.

En modo síncrono el SCI trabaja en half duplex, no pudiendo emitir y transmitir a la vez. La señal de reloj la envía el transmisor (maestro) conjuntamente con los datos. Los principios y el funcionamiento de la emisión y la recepción síncronas son similares al modo asíncrono y únicamente hay que seleccionar esta forma de trabajo cargando adecuadamente los registros TXSTA y RCSTA. La función de los pines de estos registros se detallan en las figuras 3.11 y 3.12″[16].

[16]Traducciones de los MANUALES del PIC16F877



R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n =Value at POR reset

bit 7: **CSRC**: Clock Source Select bit

Asynchronous mode
Don't care

Synchronous mode
1 = Master mode (Clock generated internally from BRG)
0 = Slave mode (Clock from external source)

bit 6: **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5: **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled
Note: SREN/CREN overrides TXEN in SYNC mode.

bit 4: **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3: **Unimplemented:** Read as '0'

bit 2: **BRGH**: High Baud Rate Select bit

Asynchronous mode
1 = High speed

Note: For the PIC16C73/73A/74/74A, the asynchronous high speed mode (BRGH = 1) may experience a high rate of receive errors. It is recommended that BRGH = 0. If you desire a higher baud rate than BRGH = 0 can support, refer to the device errata for additional information, or use the PIC16C76/77.

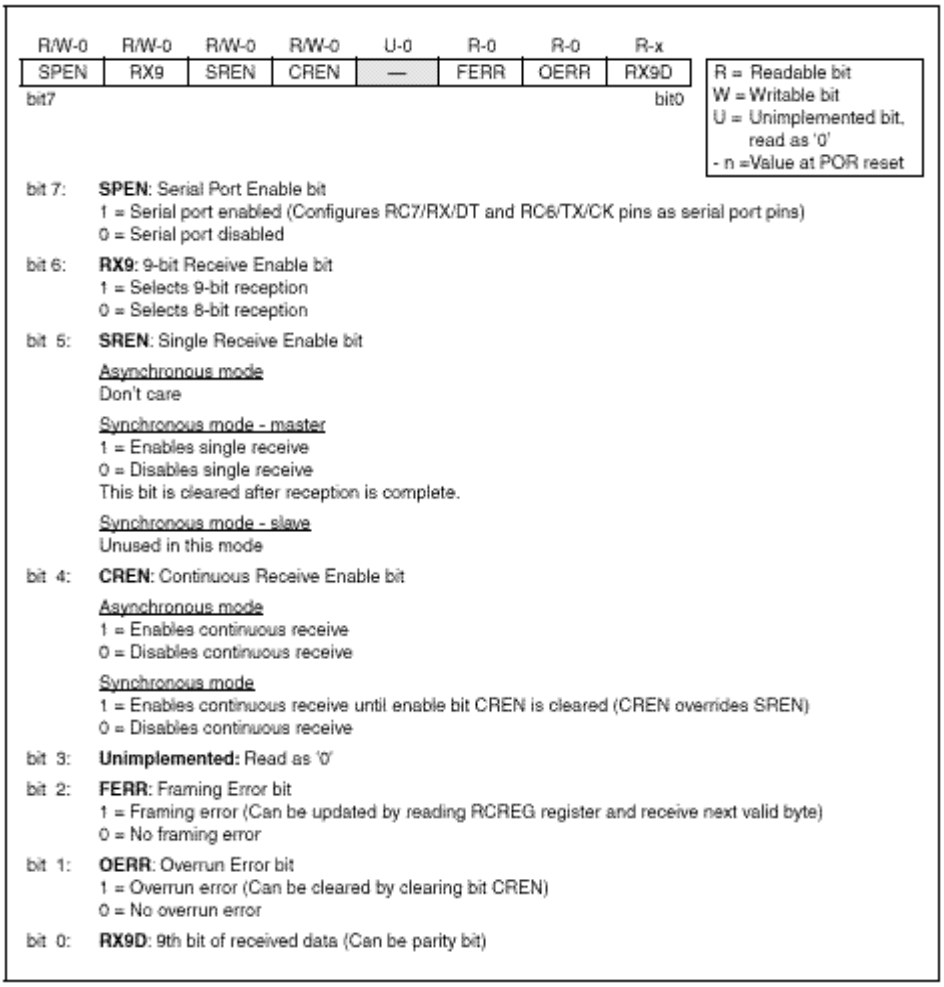
0 = Low speed

Synchronous mode
Unused in this mode

bit 1: **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0: **TX9D**: 9th bit of transmit data. Can be parity bit.

Figura 3.11 Registro TXSTA



R/W-0	R/W-0	R/W-0	R/W-0	U-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n =Value at POR reset

bit 7: **SPEN**: Serial Port Enable bit
1 = Serial port enabled (Configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6: **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5: **SREN**: Single Receive Enable bit

Asynchronous mode
Don't care

Synchronous mode - master
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode - slave
Unused in this mode

bit 4: **CREN**: Continuous Receive Enable bit

Asynchronous mode
1 = Enables continuous receive
0 = Disables continuous receive

Synchronous mode
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3: **Unimplemented:** Read as '0'

bit 2: **FERR**: Framing Error bit
1 = Framing error (Can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1: **OERR**: Overrun Error bit
1 = Overrun error (Can be cleared by clearing bit CREN)
0 = No overrun error

bit 0: **RX9D**: 9th bit of received data (Can be parity bit)

Figura 3.12 Registro RCSTA

3.5 ADQUISICIÓN DE DATOS

Cuando de adquisición de datos se trata, debe considerarse el concepto "Tiempo Real", el cual es uno de los términos mas comúnmente usados en la industria, pero su definición es ambigua. La mayoría de los ingenieros están de acuerdo en que Tiempo Real significa "con retrasos aceptables". El término *Tiempo Real duro* comúnmente se

utiliza para definir a un sistema que debe ejecutarse sin falla y cumplir con los requerimientos de tiempo real en todo momento.

El error más común es pensar que Tiempo Real significa en realidad, rápido; cuando de hecho, muchas aplicaciones de adquisición de datos y control tienen ciclos muy lentos. Los controladores de temperatura, por ejemplo, comúnmente muestrean y controlan la temperatura un par de veces por segundo, así que para que el controlador de temperatura sea estable, debe ejecutar los lazos de control en el orden de un par por segundo. Es el grado de inseguridad con cada tiempo de ciclo del lazo de control el que define los requerimientos de Tiempo Real de un sistema.

De acuerdo a lo anterior, el grado de aceptación de una adquisición de datos, dependerá de la variable medida, la precisión que se requiera de un proceso puntual, etc. Por otro lado, la velocidad y precisión con que un microcontrolador tome datos depende de su arquitectura y en alto grado del software realizado.

Los microcontroladores PICs, gracias a su arquitectura, pueden adquirir datos a velocidades bastante aceptables para las variables industriales, tanto en comunicación serie como paralela; quedando sólo dependientes de la “buena programación” a las que se les someta.

En la mayoría de aplicaciones de adquisición de datos juega un papel muy importante la conversión de análogo a digital, recurso con que cuentan algunos de los microcontroladores PIC como por ejemplo el PIC 16F877. A continuación se revisa el funcionamiento de este recurso adicional.

3.5.3 EL CONVERSOR ANÁLOGO DIGITAL

"Se trata de un potentísimo periférico que se halla integrado en el microcontrolador PIC16F877. Es un conversor analógico a digital de 10 bits de resolución y 5 canales de entrada en los modelos de 28 pines y 8 canales en los que tienen 40 pines.

La resolución que tiene cada bit procedente de la conversión tiene un valor que es función de la tensión de referencia Vref, de acuerdo a la ecuación 3.3.

$$Resolución = \frac{V_{ref+} - V_{ref-}}{1024} = \frac{V_{ref}}{1024} \qquad \textbf{(Ec. 3.3)}$$

A través del canal de entrada seleccionado, se aplica la señal analógica a un condensador de muestreo y retención (sample and hold) y luego se introduce al conversor, el cual proporciona un resultado digital de 10 bits de longitud usando la técnica de "aproximaciones sucesivas".

El conversor A/D es el único dispositivo que puede funcionar en modo reposo (SLEEP), para ello el reloj del conversor deberá conectarse al oscilador RC interno.

La tensión de referencia puede implementarse con la tensión interna (V_{DD}) o externa (entra por la patita AN3/V_{ref+}). En cada momento la conversión solo se realiza con la entrada de uno de sus canales, depositando el resultado de la misma en el registro ADRES y activándose el señalizador ADIF, que provoca una interrupción si el bit de permiso correspondiente esta activado. Además, al terminar la conversión el bit GO/DONE se pone a 0. La tensión de referencia puede provenir de la tensión interna V_{DD} o de la externa que se introduce por la patita AN3/V_{ref}.

Para gobernar el funcionamiento del CAD se utilizan dos registros: ADCON0 y ADCON1. El primero, que se muestra en la figura 3.13, selecciona el canal a convertir con los bits CHS <2:0>, activa al conversor y contiene el señalizador que avisa del fin de la conversión (ADIF) y el bit GO/DONE.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = channel 0, (RA0/AN0)
001 = channel 1, (RA1/AN1)
010 = channel 2, (RA2/AN2)
011 = channel 3, (RA3/AN3)
100 = channel 4, (RA5/AN4)
101 = channel 5, (RE0/AN5)(1)
110 = channel 6, (RE1/AN6)(1)
111 = channel 7, (RE2/AN7)(1)

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
If ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion)
0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is operating
0 = A/D converter module is shut-off and consumes no operating current

Figura 3.13 Registro ADCON0

El registro ADCON1 establece las entradas que son digitales y analógicas, así como el tipo de tensión de referencia (interna o externa). En la figura 3.14 se encuentra la configuración de este registro"[17].

El tiempo que dura la conversión depende de la frecuencia de funcionamiento del PIC mediante la relación dada por la ecuación 3.4 y del valor de los bits ADCS1 y ADCS0, según la tabla 3.5.

$$T_{AD} = n.T_{OSC} \qquad \textbf{(Ec. 3.4)}$$

[17] Traducciones de los MANUALES del PIC16F877

donde:

n es un valor dependiente de la tabla.

T_{AD} es el tiempo de conversión por bit.

T_{OSC} es la frecuencia de trabajo del microcontrolador.

Cuando se trabaja con valores digitales de 10 bits, se requiere un tiempo mínimo de 12. $T_{AD.}$

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+	VREF-	CHAN/ Refs(2)
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	RA3	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1/2

A = Analog input D = Digital I/O

Figura 3.14 Registro ADCON1

Por ejemplo, si el PIC trabaja a 20 MHz y los bits ADCS1:ADCS0 =00, el tiempo de conversión T_{AD} = 100 ns; si la frecuencia es de 5 MHz, T_{AD} = 400 ns; si la frecuencia es de 1,25 MHz, T_{AD} =1,6 μs y si la frecuencia es de 333,33 KHz, T_{AD} = 6 μs.

Tabla 3.5 Valores de T_{AD} según el estado de los bits ADCS1:ADCS0

T_{AD}	ADCS1:ADCS0

$2.T_{OSC}$	00
$8.T_{OSC}$	01
$32.T_{OSC}$	10
RC	11

Finalmente, se describen de forma resumida los pasos para realizar una conversión en el CA/D:

- Se configura correctamente el CA/D programando los bits de los registros de control.
- Se autoriza o prohibe la generación de interrupción al finalizar la conversión, cargando los bits del PIE1.
- Para iniciar la conversión se pone el bit GO/DONE = 1. Hay que tener en cuenta el tiempo que durará la conversión.
- Se detecta el final de la conversión bien porque se genera la interrupción, o bien porque se explora cuando el bit GO/DONE = 0.
- Se lee el resultado de la conversión en los 10 bits válidos de ADRESH y ADRESL y se debe borrar la bandera ADIF.

Debe recordarse que los registros que participan en la programación del conversor A/D son: INTCON, PIR1, PIE1, ADRESH, ADRESL, ADCON0, ADCON1, TRISA, PORTA, TRISE y PORTE.

CAPÍTULO IV

DISEÑO Y CONSTRUCCIÓN

4.1 DESCRIPCIÓN DEL PROYECTO

Para el diseño de la aplicación se debe separar inicialmente el trabajo del microcontrolador del que realiza Matlab, para finalmente acoplarlos y realizar las pruebas respectivas. Lo que si se debe tomar en cuenta son las características generales que poseerá la tarjeta y la comunicación, y son las que se detallan a continuación

- Velocidades de trabajo en baudios: 9600, 4800, 2400 y 1200.
- Longitud de los datos en bits: 8 y 9.
- Paridad (si se escoge 9 bits): par o impar.
- Entradas analógicas para un máximo de $\pm$ 5V: 12 (6 de cada microcontrolador).
- Exposición visual amigable de los datos analógicos adquiridos.
- Protocolo de comunicación entre maestro y esclavos el ASCII versión larga.
- Avisos de fallas en la comunicación.

En la parte teórica ya se detallaron las formas de trabajo y registros necesarios para el microcontrolador, la forma de manipulación del puerto serie en Matlab, las consideraciones a seguir para la adquisición de datos, así como también los conceptos referentes a los protocolos; por lo que, antes de iniciar el trabajo sólo resta por

exponer los lineamientos del código ASCII que se va emplear como protocolo de comunicación.

PROTOCOLO ASCII

″Este protocolo es diseñado para la comunicación con sensores y dispositivos de control de procesos que se comunican con el puerto serie del PC u otro tipo de dispositivo en forma serial.

Utiliza los estándares EIA-232 o el EIA-485. Los dispositivos que se comunican, transmiten y reciben información en forma digital y actúan sobre procesos con señales analógicas utilizando conversores A/D y D/A

La información se almacena en forma ASCII en un buffer en donde los contenidos se guardan 8 veces por segundo. La actualización de la información desde y hacia el computador se la hace mediante el envío de simples comandos en forma ASCII, se debe tomar en cuenta que son máximo 20 caracteres.

La comunicación se basa en un protocolo simple comando/respuesta ASCII, entre el dispositivo HOST y el módulo de comunicación. La estructura del protocolo en su forma larga es la siguiente:

- Comando desde el HOST para escritura.

#		W	R			[CR]
Inicio	Dirección	Escritura		Checksum		Final

- Comando desde el HOST para lectura.

#		R	D			[CR]
Inicio	Dirección	Lectura		Checksum		Final

- Respuesta del módulo transmisor para escritura.

*		W	R						[CR]
Inicio	Dirección	Escritura			Dato			Checksum	Final

- Respuesta del módulo transmisor para lectura.

*		R	D						[CR]
Inicio	Dirección	Lectura			Dato			Checksum	Final

- Error en la comunicación (dos formas)"[18].

?		[SP]	B	A	D	[SP]	C	H	E	C	K	S	U	M	[CR]

Inicio Dirección

Final

?		[SP]	S	Y	N	T	A	X	[SP]	E	R	R	O	R	[CR]

Inicio Dirección

Final

4.2 HARDWARE DE LA TARJETA DE ADQUISICIÓN DE DATOS

La tarjeta de adquisición de datos posee básicamente los siguientes elementos:

- 2 Microcontroladores
- 1 Transceiver MAX232
- 1 Multiplexor de 2 a 1 74157
- 2 Osciladores de 10 MHz
- Condensadores de 1 µF y 15 nF

[18]Copiados de la asignatura COMUNICACIÓN PARA INSTRUMENTACIÓN- Ing. Eddie Galarza Z. MSc. Capítulo IV

La configuración del microcontrolador PIC16F877 está en el capítulo 3 y la del Transceiver se visualiza en la figura 4.1

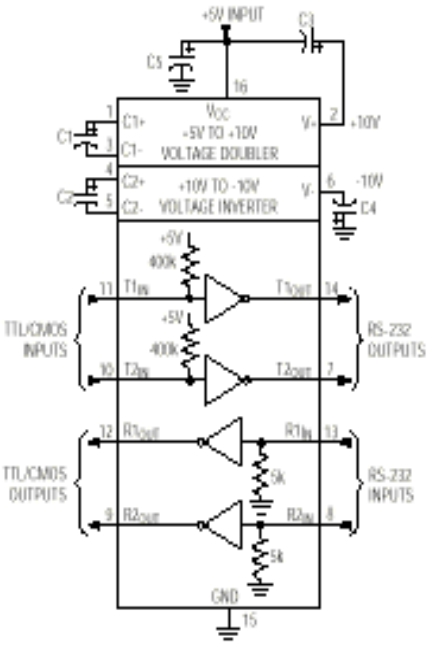


Figura 4.1 Configuración del Transceiver MAX232

"Las reglas para la conversión son muy simples y se indican a continuación:

TTL → **RS-232**	**RS-232** **TTL**→
+ 5 V (alto) - 12 V	- 12 V + 5 V (alto)
0 V (bajo) +12 V	+12 V 0 V (bajo)"[19]

Se utiliza un multiplexor de 2 a 1 para asegurar que el transceiver y por ende la PC reciba datos sólo de un microcontrolador a la vez

[19]IBM PC para Ingenieros – Ricardo Zelenovsky, pag: 10-2

El diagrama esquemático se observa de la figura 4.2. En el anexo A se encuentra el esquema de la placa diseñada.

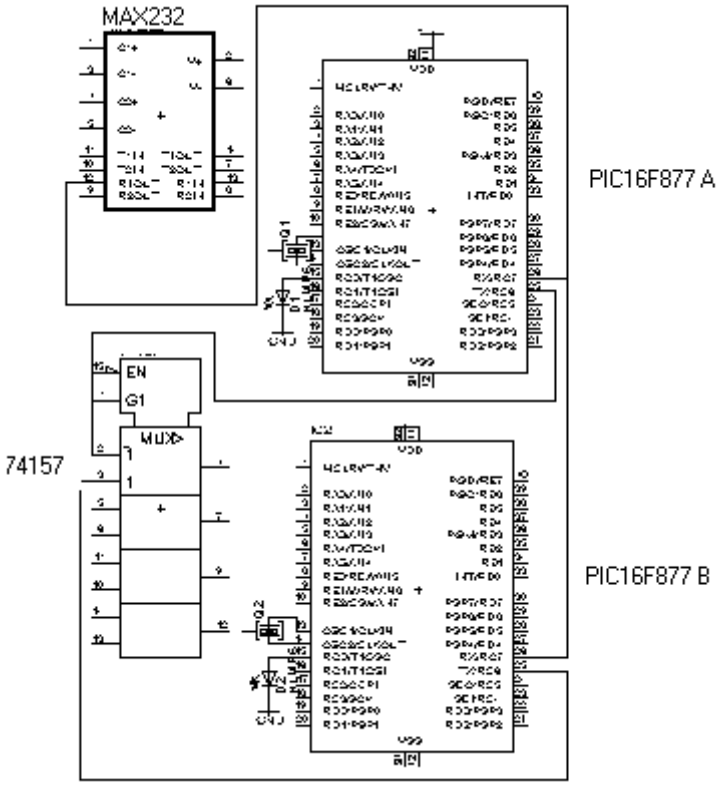


Figura 4.2 Diagrama General de la Tarjeta

4.3 SOFTWARE DE LOS MICROCONTROLADORES

Para el desarrollo del software de los microcontroladores se realizó el programa principal y luego se separó el análisis en los siguientes puntos:

- Recepción y verificación de la dirección del microcontrolador.
- Recepción y verificación de la velocidad de comunicación.
- Recepción y verificación de la longitud de los datos.
- Recepción y verificación del tipo de paridad.
- Recepción y verificación de las entradas analógicas.
- Recepción y verificación de que se desea leer datos del microcontrolador.
- Lectura de datos del ADC y almacenamiento en los bancos.

- Construcción de la trama ASCII y envío de los datos al computador.

- Envío de los códigos de error en el checksum al computador.

- Creación del bit de paridad.

Cada uno de estos puntos se desarrollaron en subrutinas independientes, con el objetivo de brindar una mayor facilidad en la depuración del programa final.

PROGRAMA PRINCIPAL

En este punto se realizan las configuraciones iniciales o por defecto para la comunicación serial y el ADC. También desde aquí se envía a las subrutinas necesarias, siempre que se den las condiciones para ello.

Para pasar de una subrutina a otra se configuran algunos bits de control, los cuales sirven para verificar que una operación es correcta o no, la función específica de cada uno de estos se indica en la tabla 4.1.

Tabla 4.1 Bits de Control

CONTROL.7	CONTROL.6	CONTROL.5	CONTROL.4	CONTROL.3	CONTROL.2	CONTROL.1	CONTROL.0

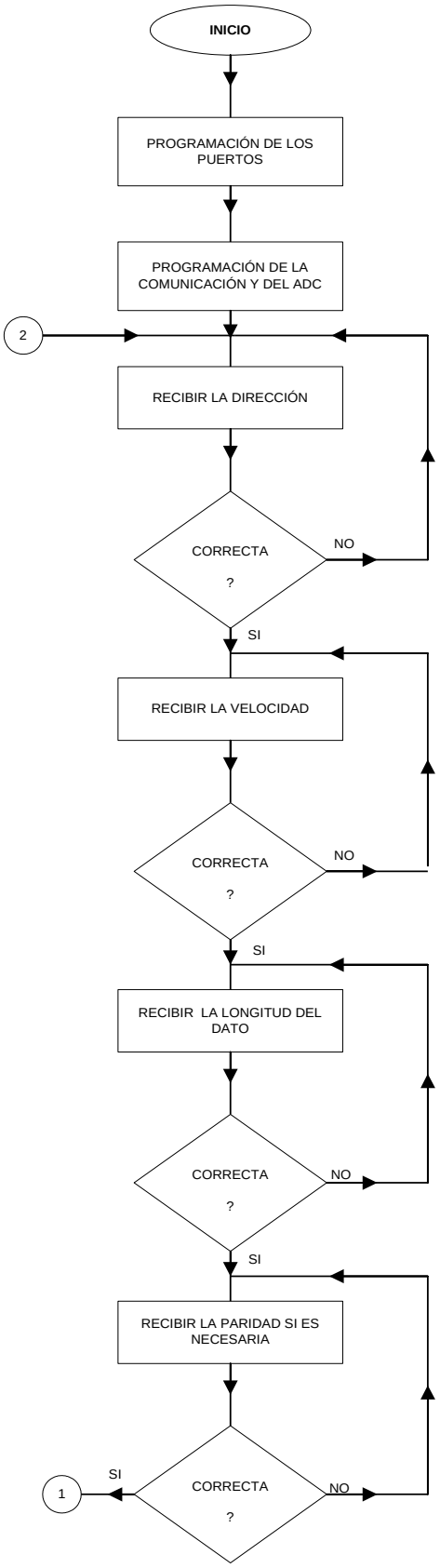
Par-Impar	-	Lectura	Entrada	Paridad	Longitud	Velocidad	Dirección

El diagrama de flujo del programa principal de cada uno de los microcontroladores se presenta en la figura 4.3.

4.3.1. RECEPCIÓN Y VERIFICACIÓN DE LA DIRECCIÓN DEL MICROCONTROLADOR

En este punto se chequea que la comunicación es con el microcontrolador deseado, para lo cual analiza la dirección y luego verifica el estado del checksum.

Si los datos recibidos son los correctos se habilita el bit de control respectivo. Si no es el bit adecuado no se realiza nada, es decir, se deja pasar esa información. En cambio, si en el checksum se detecta error, transmite al Matlab ese mensaje.


INICIO
PROGRAMACIÓN DE LOS PUERTOS
PROGRAMACIÓN DE LA COMUNICACIÓN Y DEL ADC
2
RECIBIR LA DIRECCIÓN
CORRECTA ?
NO
SI
RECIBIR LA VELOCIDAD
CORRECTA ?
NO
SI
RECIBIR LA LONGITUD DEL DATO
CORRECTA ?
NO
SI
RECIBIR LA PARIDAD SI ES NECESARIA
CORRECTA ?
NO
SI
1

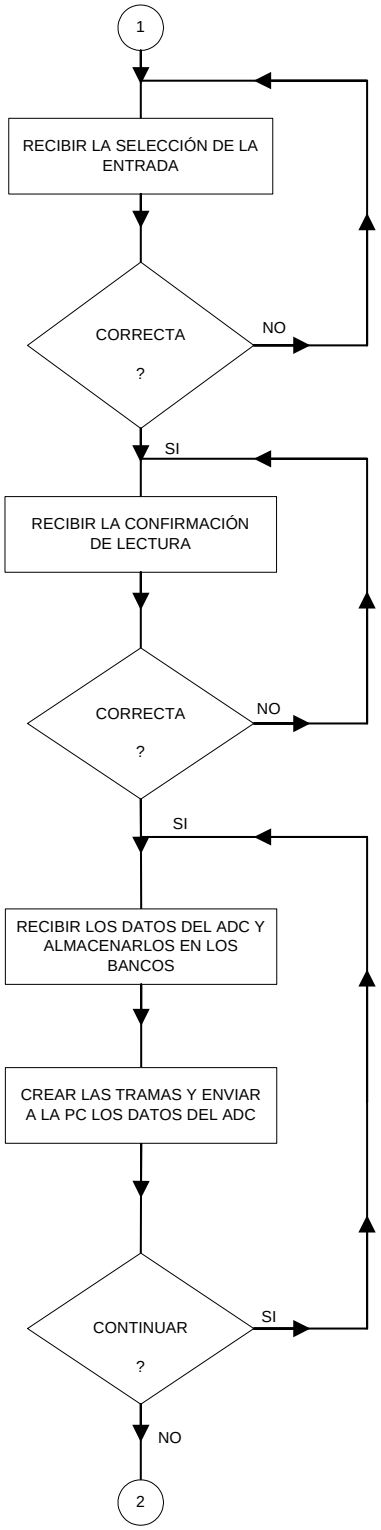


Figura 4.3 Diagrama de Flujo del Programa Principal

En la tabla 4.2 se indican los valores utilizados para las direcciones de los microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

Tabla 4.2 Códigos ASCII para la Dirección

MICRO	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		CHECKSUM		FINAL (CR)
1	23	31	57	52	46	44	OD
2	23	32	57	52	46	45	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.4.

4.3.2. RECEPCIÓN Y VERIFICACIÓN DE LA VELOCIDAD DE COMUNICACIÓN

En este punto se recibe y chequea la velocidad a la que se desea establecer la comunicación entre el microcontrolador y el Matlab.

Si los datos recibidos son los correctos se habilita el bit de control respectivo y especialmente se configura la velocidad deseada en el registro respectivo. En cambio, si el checksum detecta error transmite al Matlab ese mensaje.

Para la regulación de los datos en el registro SPBGR se toma como referencia la ecuación 3.2 y como valores en *n* el 64, o sea modo asincrónico de baja velocidad y en *Fosc* los 10 MHz del oscilador utilizado.

A continuación se indica el ejemplo para el cálculo de la velocidad de 9600 baudios, tomando en cuenta que deberá pasarse el valor obtenido a hexadecimal:

$$x = \frac{10MHz/9600 - 64}{64} = 15{,}2 \text{ decimal} = 0F \text{ hexadecimal}$$

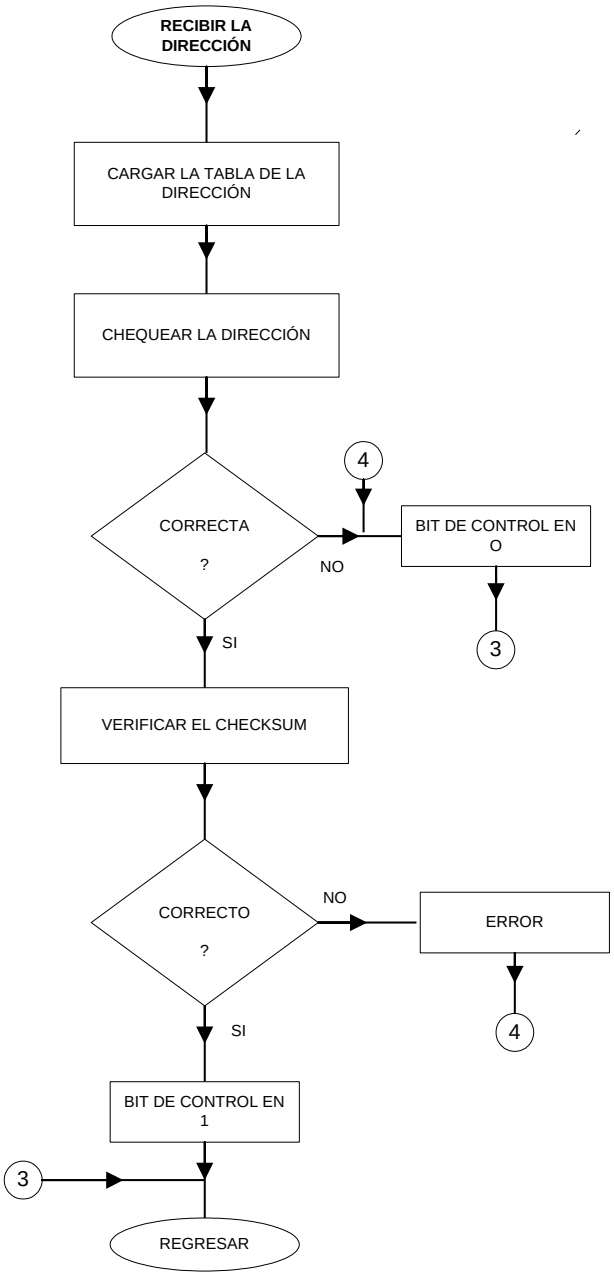


Figura 4.4 Diagrama de Flujo de la Subrutina para la Recepción y Verificación de la Dirección del Microcontrolador

En las tablas 4.3 y 4.4 se indican los valores utilizados para las velocidades en los dos microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

Tabla 4.3 Códigos ASCII para la Velocidad en el Microcontrolador 1

VELOCIDAD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO		CHECKSUM			FINAL (CR)
9600	23	31	57	52	39	36	31	36	43	OD
4800	23	31	57	52	34	38	31	36	39	OD
2400	23	31	57	52	32	34	31	36	33	OD
1200	23	31	57	52	31	32	31	36	30	OD

Tabla 4.4 Códigos ASCII para la Velocidad en el Microcontrolador 2

VELOCIDAD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO		CHECKSUM			FINAL (CR)
9600	23	32	57	52	39	36	31	36	44	OD
4800	23	32	57	52	34	38	31	36	41	OD
2400	23	32	57	52	32	34	31	36	34	OD
1200	23	32	57	52	31	32	31	36	31	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.5.

4.3.3. RECEPCIÓN Y VERIFICACIÓN DE LA LONGITUD DE LOS DATOS

En este punto se recibe y chequea la longitud del dato que se desea para la transmisión y recepción de los datos entre el microcontrolador y el Matlab.

Si los datos recibidos son correctos; se habilita el bit de control respectivo y se configura el registro respectivo. En cambio, si el checksum detecta error transmite al Matlab ese mensaje.

En las tablas 4.5 y 4.6 se indican los valores utilizados para las dos longitudes que pueden regularse en los microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

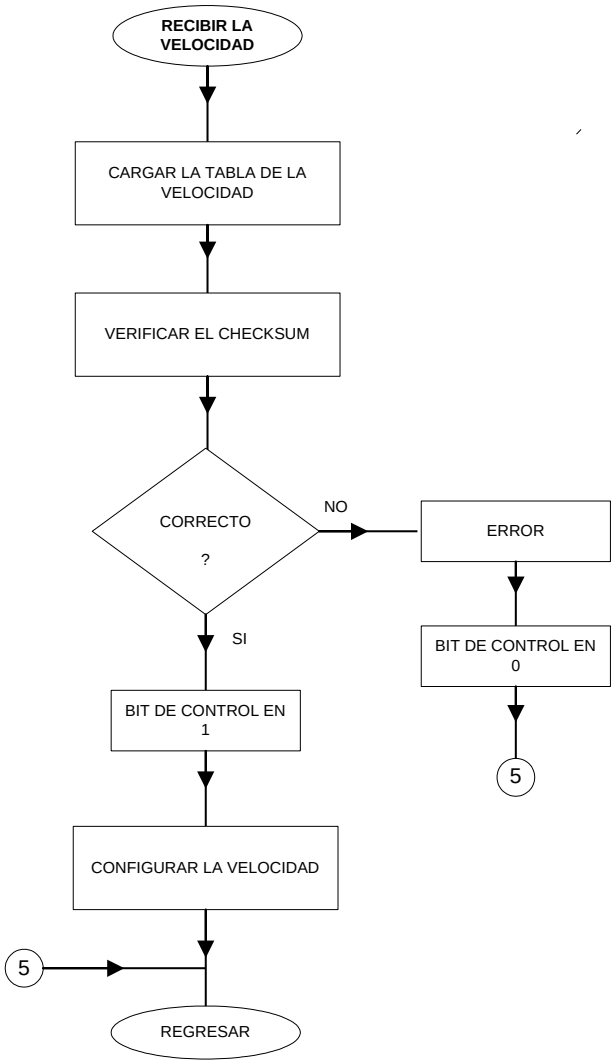


Figura 4.5 Diagrama de Flujo de la Subrutina para la Recepción y Verificación de la Velocidad de Comunicación

Tabla 4.5 Códigos ASCII para la Longitud en el Microcontrolador 1

LONGITUD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
8	23	31	57	52	38	31	33	35	OD
9	23	31	57	52	39	31	33	36	OD

Tabla 4.6 Códigos ASCII para la Longitud en el Microcontrolador 2

LONGITUD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
8	23	32	57	52	38	31	33	36	OD
9	23	32	57	52	39	31	33	37	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.6.

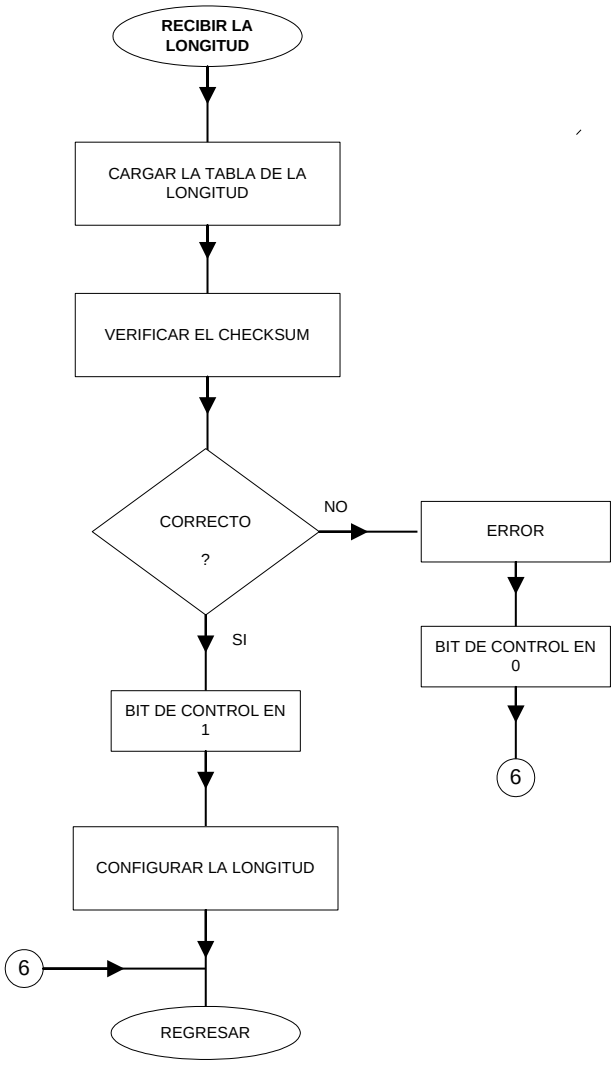


Figura 4.6 Diagrama de Flujo de la Subrutina para la Recepción y Verificación de la Longitud del Dato

4.3.4. RECEPCIÓN Y VERIFICACIÓN DEL TIPO DE PARIDAD

Se llega a esta condición si se escogió comunicación con 9 bits en la longitud del dato, se recibe y chequea el tipo de paridad que se desea.

Si los datos recibidos son los correctos se habilita el bit de control respectivo y se configuran los registros respectivos. En cambio, si el checksum detecta error transmite al Matlab ese mensaje.

En las tablas 4.7 y 4.8 se indican los valores utilizados para los dos tipos de paridad que pueden regularse en los microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

Tabla 4.7 Códigos ASCII para la Paridad en el Microcontrolador 1

PARIDAD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
Par	23	31	57	52	31	31	32	45	OD
Impar	23	31	57	52	32	31	32	46	OD

Tabla 4.8 Códigos ASCII para la Paridad en el Microcontrolador 2

PARIDAD	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
Par	23	32	57	52	31	31	32	46	OD
Impar	23	32	57	52	32	31	33	30	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.7.

4.3.5. RECEPCIÓN Y VERIFICACIÓN DE LA ENTRADA

En este punto se recibe y verifica la entrada con la que se desea trabajar en el ADC. Si los datos recibidos son los correctos, se habilita el bit de control respectivo y se configuran los registros necesarios. En cambio, si el checksum detecta error transmite al Matlab ese mensaje.

En las tablas 4.9 y 4.10 se indican los valores utilizados para las seis entradas que pueden emplearse en los microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

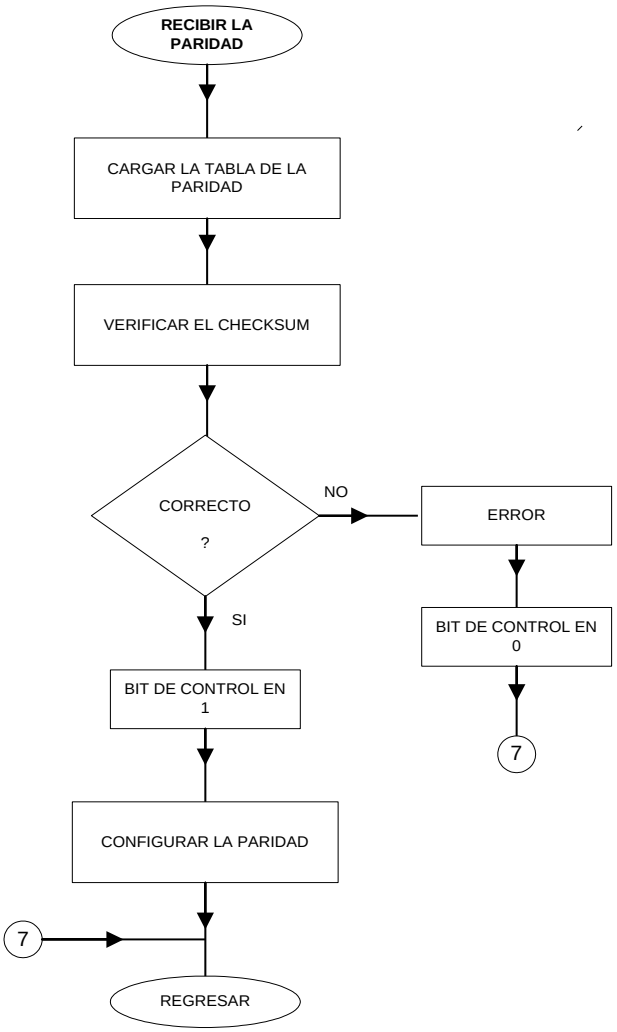


Figura 4.7 Diagrama de Flujo de la Subrutina para la Recepción y Verificación de la Paridad

Tabla 4.9 Códigos ASCII para las Entradas en el Microcontrolador 1

ENTRADA	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
0	23	31	57	52	30	31	32	44	OD
1	23	31	57	52	31	31	32	45	OD
2	23	31	57	52	34	31	33	31	OD
3	23	31	57	52	35	31	33	32	OD
4	23	31	57	52	36	31	33	33	OD
5	23	31	57	52	37	31	33	34	OD

Tabla 4.10 Códigos ASCII para las Entradas en el Microcontrolador 2

ENTRADA	INICIO (#)	DIRECCIÓN	ESCRITURA (WR)		DATO	CHECKSUM			FINAL (CR)
0	23	32	57	52	30	31	32	45	OD
1	23	32	57	52	31	31	32	46	OD
2	23	32	57	52	34	31	33	32	OD
3	23	32	57	52	35	31	33	33	OD
4	23	32	57	52	36	31	33	34	OD
5	23	32	57	52	37	31	33	35	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.8.

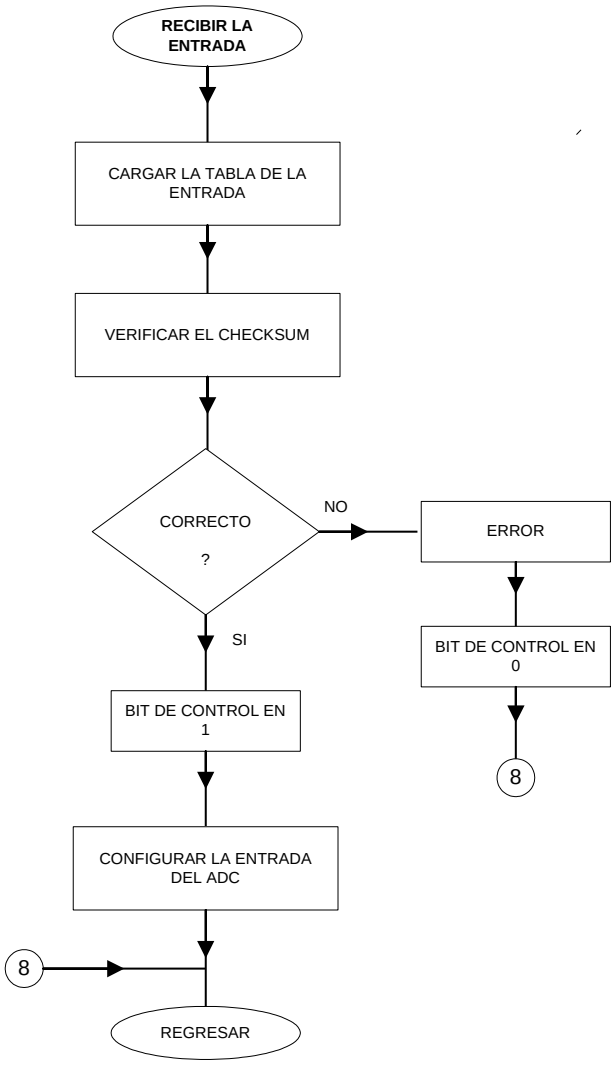


Figura 4.8 Diagrama de Flujo de la Subrutina para la Recepción y Verificación de la Entrada Analógica

4.3.6. RECEPCIÓN Y VERIFICACIÓN DE UNA OPERACIÓN DE LECTURA

Aquí se chequea que la comunicación es con el microcontrolador seleccionado y que se desea realizar una operación de lectura, para lo cual analiza la dirección y luego verifica el checksum.

Si los datos recibidos son los correctos se habilita el bit de control respectivo. Si no es el bit adecuado no se realiza nada, es decir, se deja pasar esa información. En cambio, si el checksum detecta error, transmite al Matlab ese mensaje.

En la tabla 4.11 se indican los valores utilizados en la operación lectura para los microcontroladores en códigos ASCII, los mismos que sirven para la verificación del checksum.

Tabla 4.11 Códigos ASCII para Lectura de Datos

MICRO	INICIO	DIRECCIÓN	LECTURA (WR)		CHECKSUM		FINAL (CR)
1	23	31	52	44	45	41	OD
2	23	32	52	44	45	42	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.9.

4.3.7. LECTURA DE DATOS DEL ADC Y ALMACENAMIENTO EN LOS BANCOS

En este caso se debe tomar en cuenta que las 6 entradas analógicas ingresarán por los pines PA.0, PA.1, PA.5, PE.0, PE.1 y PE.2, según la entrada analógica seleccionada; por lo tanto, se debe setear adecuadamente el valor del registro

ADCON1 y en los pines PA.2 y PA.3 debe ir el voltaje de referencia negativo y positivo respectivamente.

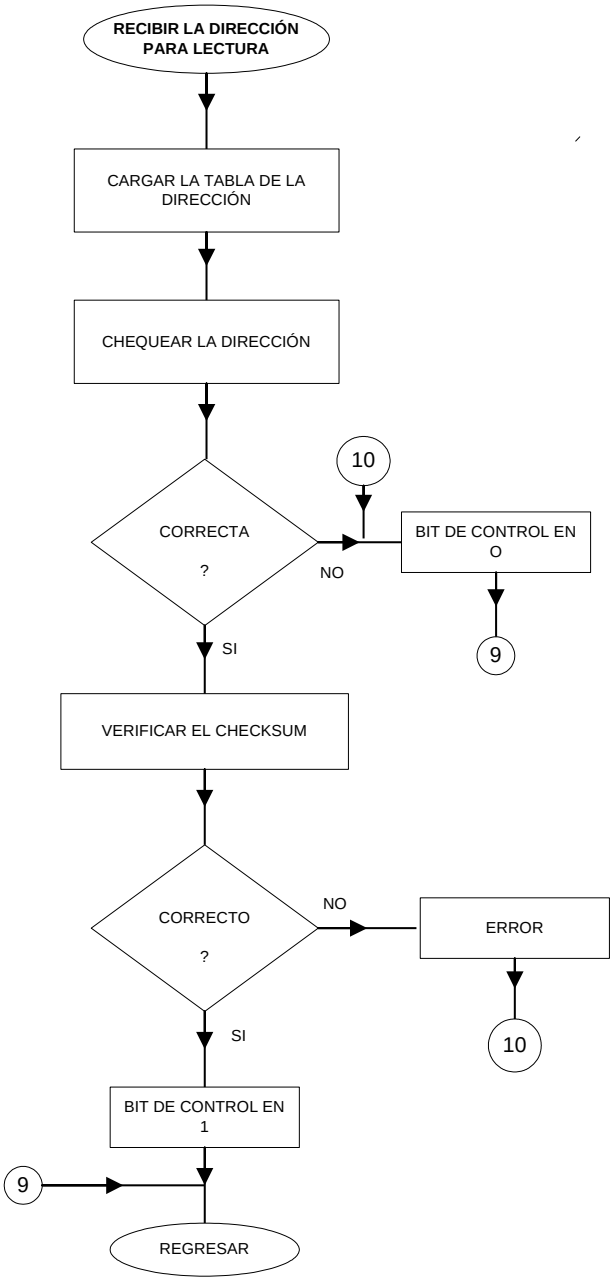


Figura 4.9. Diagrama de Flujo de la Subrutina para la Recepción y Verificación de Lectura de Datos

Se debe tomar en cuenta también, que los 10 bits que conforman el dato binario del ADC se encuentran en ADRESH y ADRESL y estos a su vez están en bancos diferentes.

Cabe mencionar que la resolución del ADC está dada por la ecuación 3.3 y por los voltajes de referencia. Si por ejemplo *Vref+* es 5 voltios y *Vref-* también es 5 voltios se tiene como resultado:

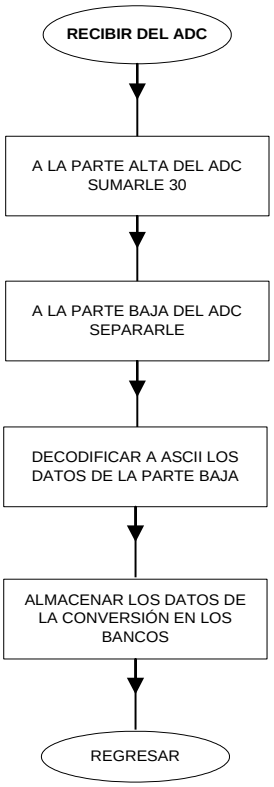
$$Resolución = \frac{5V - (-5V)}{1024} = \frac{10V}{1024}$$

$Resolución = 9{,}764625 \times 10^{-3}\ V$

El tiempo que durará la conversión del ADC depende de la frecuencia de funcionamiento, o sea de los 10 MHz y del valor de los bits ADCS1 y ADCS0, que en este caso están en 1 y 0 respectivamente, porque con ellos se logra un T_{AD} de 3,2 μs, el cual está dentro del rango recomendado por el fabricante. Además, como el conversor es de 10 bits, el tiempo mínimo dado por el fabricante es de 12. $T_{AD,}$ o sea 38,4 μs.

Debido a que se deben muestrar los datos antes de enviarlos al computador, en esta subrutina se almacenan 120 datos, cada uno ocupa dos localidades, por lo tanto se emplean 80 celdas de los bancos 1, 2 y 3 del microcontrolador.

El diagrama de flujo general de esta subrutina se indica en la figura 4.10.



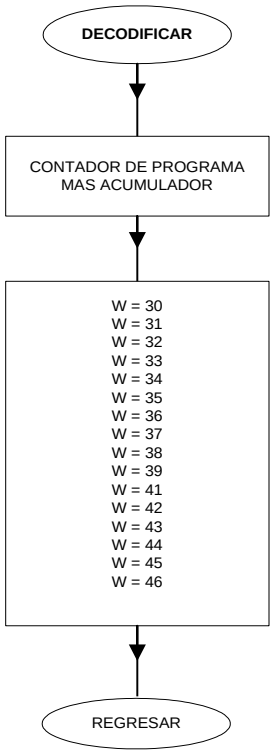


Figura 4.10 Diagrama de Flujo de la Subrutina para la Lectura del ADC y Almacenamiento en los Bancos

4.3.8. CONSTRUCCIÓN DE LA TRAMA ASCII Y ENVÍO DE LOS DATOS AL COMPUTADOR

En esta subrutina se transmiten los datos ya convertidos a códigos ASCII al MATLAB, a la misma velocidad, longitud y paridad que se trabajó en la recepción de datos por parte del PIC.

Como punto central, se realiza la tarea de decodificación a códigos ASCII de las entradas analógicas, aquí debe considerarse que de los diez bits del ADC, uno es de signo y deberá adicionarse el dato que indique este parámetro. En la tabla 4.12 se realizan 4 ejemplos de la decodificación a ASCII de los datos del ADC.

Tabla 4.12 Ejemplos de la Conversión a ASCII del ADC

VOLTIOS	BINARIO	ADRESH	ADRESL	ASCII		
0	0000000000	00	00	30	30	30
$9{,}765 \times 10^{-3}$	0000000001	00	01	30	30	31
$19{,}53 \times 10^{-3}$	0000000010	00	02	30	30	32
-4,9902	1111111110	03	FE	31	46	45
-5	1111111111	03	FF	31	46	46

A más del dato, se deben generar y enviar los valores del checksum que servirán de base para el chequeo en el Matlab. En caso que se escogió incluir la paridad, también se enviará el bit adicional adecuado.

A continuación, en la tabla 4.13 se presentan dos ejemplos de los datos que se enviarán al MATLAB, el primero corresponde al microcontrolador 1 si ingresan a su ADC 0 voltios y el segundo se refiere al microcontrolador 2 si ingresan a su ADC - 5 voltios.

Tabla 4.13 Ejemplos de datos Transmitidos por el Microcontrolador

MICROCONT.	INICIO (*)	DIRECCIÓN	LECTURA (RD)		DATO				CHECKSUM			FINAL (CR)
MC 1	2A	31	52	44	2B	30	30	30	31	41	43	OD
MC 2	2A	32	52	44	2D	31	46	46	31	44	43	OD

El diagrama de flujo general de esta subrutina se indica en la figura 4.11.

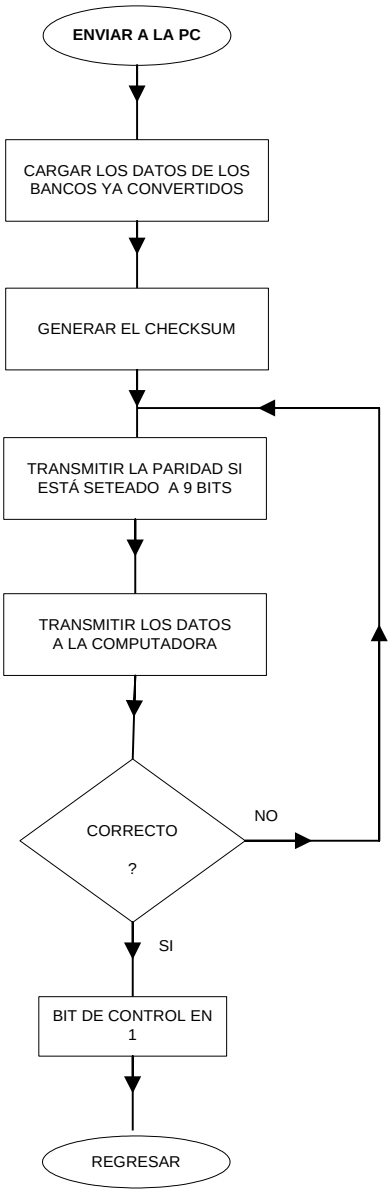


Figura 4.11 Diagrama de Flujo de la Subrutina para la Construcción de la Trama ASCII y Envío de Datos al Matlab

4.3.9. ENVÍO DE LOS CÓDIGOS DE ERROR AL COMPUTADOR

En esta subrutina se envían los códigos ASCII al Matlab que indican que existe un error en el checksum. Estos datos pueden incluir paridad si así está seteada la comunicación serial.

El diagrama de flujo general de esta subrutina se indica en la figura 4.12.

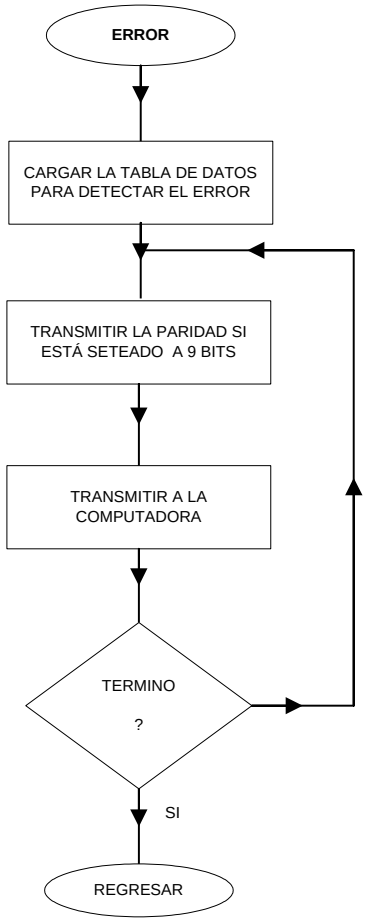


Figura 4.12 Diagrama de Flujo de la Subrutina para el Envío de los Datos de Error en el Checksum

4.3.10. CREACIÓN DEL BIT DE PARIDAD

En esta subrutina se genera el bit de paridad (1) de acuerdo a la selección escogida, par o impar y se pondrá en el bit TX9D del registro TXSTA el mismo que está en el banco 1.

Esta subrutina sirve tanto para envío de los datos del ADC cuanto para los códigos de error en la comunicación.

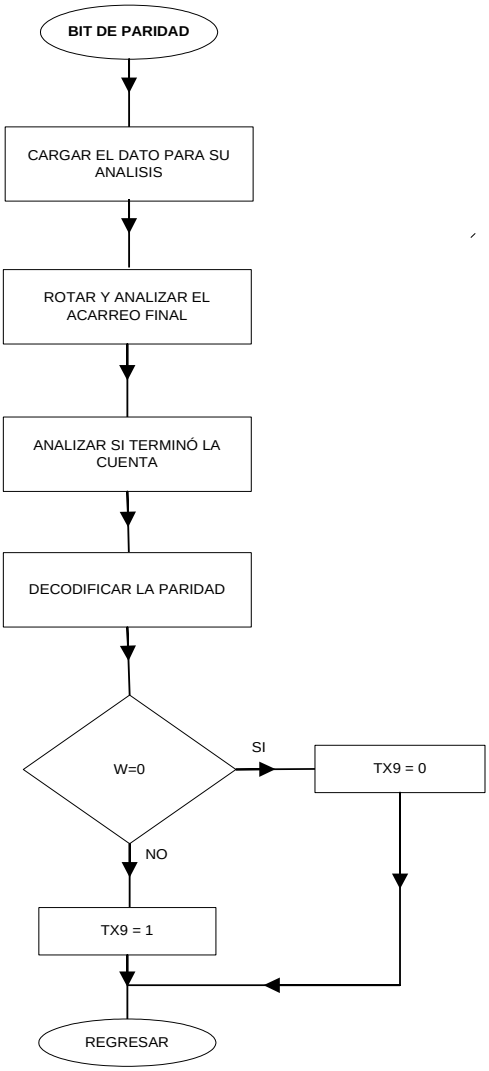
El diagrama de flujo general de esta subrutina se encuentra en la figura 4.13. En el anexo B se presentan las instrucciones para la programación del microcontrolador 1.

4.4 LA INTERFASE GRÁFICA CON MATLAB

Para el desarrollo de la interfase se deben tomar en cuenta los siguientes puntos:

- "Todos los códigos, incluyendo las *callbacks*, están contenidos en la aplicación M-file. Cada callback está implementada como una subfunción en el M-file.
- La estructura *handles* proporciona un fácil acceso a todas las componentes del manejador en el GUI. Además, puede usar esta estructura para almacenar todos los datos globales requeridos por la aplicación M-file.
- Se puede usar la fijación de propiedades específicas para crear un GUI que funcione más consistentemente cuando corra sobre diferentes plataformas.

- El primer mecanismo para implementar un GUI es programar el callback del objeto usado para construir la interfase.



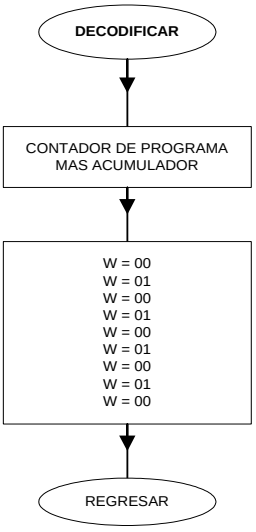


Figura 4.13 Diagrama de Flujo de la Subrutina para la Creación del Bit de Paridad

Se debe tomar en cuenta también, que todos los objetos gráficos tienen tres propiedades que habilitan la definición de la rutinas, las mismas que son las siguientes:

- *ButtonDownFcn:* MATLAB ejecuta este callback cuando el usuario hace clic sobre el objeto.
- *CreateFcn:* MATLAB ejecuta este callback cuando crea el objeto.
- *DeleteFcn:* MATLAB ejecuta este callback justo antes de borrar el objeto.

De igual manera, las figuras tienen propiedades adicionales que ejecutan las subrutinas, estas son las siguientes:

- *CloseRequestFcn:* Tiene un callback definido por defecto. MATLAB ejecuta este callback cuando se cierra la figura.
- *KeyPressFcn:* MATLAB ejecuta este callback cuando el usuario presiona una tecla, si el cursor está en la ventana de la figura.
- *ResizeFcn:* MATLAB ejecuta este callback cuando el usuario minimiza la ventana de la figura.
- *WindowButtonDownFcn:* MATLAB ejecuta este callback cuando se cierra el ususario hace clic sobre la figura, pero no sobre un control habilitado.
- *WindowButtonMotionFcn :* MATLAB ejecuta este callback cuando el usuario mueve el mouse dentro de la ventana de la figura.

- *WindowButtonUpFcn :* MATLAB ejecuta este callback cuando el usuario libera el botón del mouse después de haber presionado el botón en la figura"[20].

[20] MATLABHelp

Bajo estas recomendaciones y de acuerdo a las características indicadas al inicio del capítulo, se diseñan tres ventanas, las mismas que se detallan a continuación:

4.4.1 VENTANA PRINCIPAL

En esta pantalla aparecen datos informativos de la aplicación y fue realizada con textos estáticos y un botón que permite ingresar a la siguiente ventana.

La ventana principal se indica en la figura 4.14.

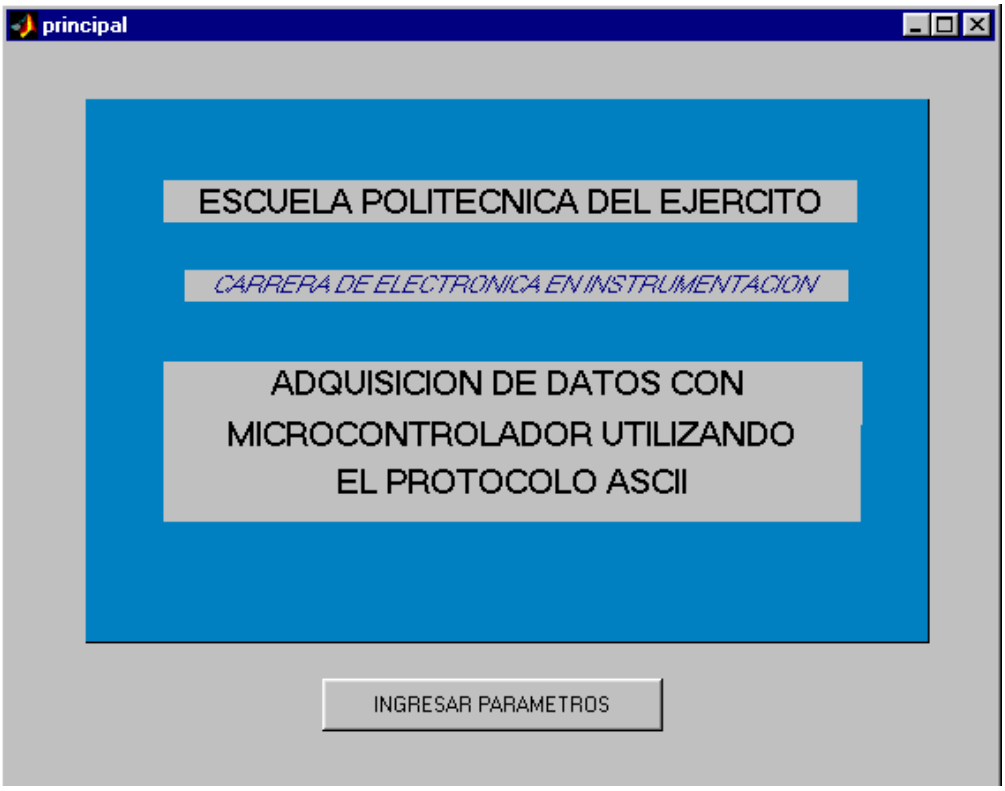


Figura 4.14 Ventana Principal

4.4.2 VENTANA PARA SELECCIONAR LOS PARÁMETROS DE LA COMUNICACIÓN

En esta ventana se definen los parámetros para la adquisición como son:

- Microcontrolador de trabajo: MC A y MC B
- La Velocidad: 9600, 4800, 2400 y 1200 baudios
- La Longitud: 8 y 9 bits
- La Paridad: Par e Impar
- La Entrada: 1, 2 ,3 ,4, 5 o 6

- Indicadores de funcionamiento

Esta pantalla está diseñada entre otros con textos estáticos, botones y controles de texto. El diseño de esta pantalla se indica en la figura 4.15.

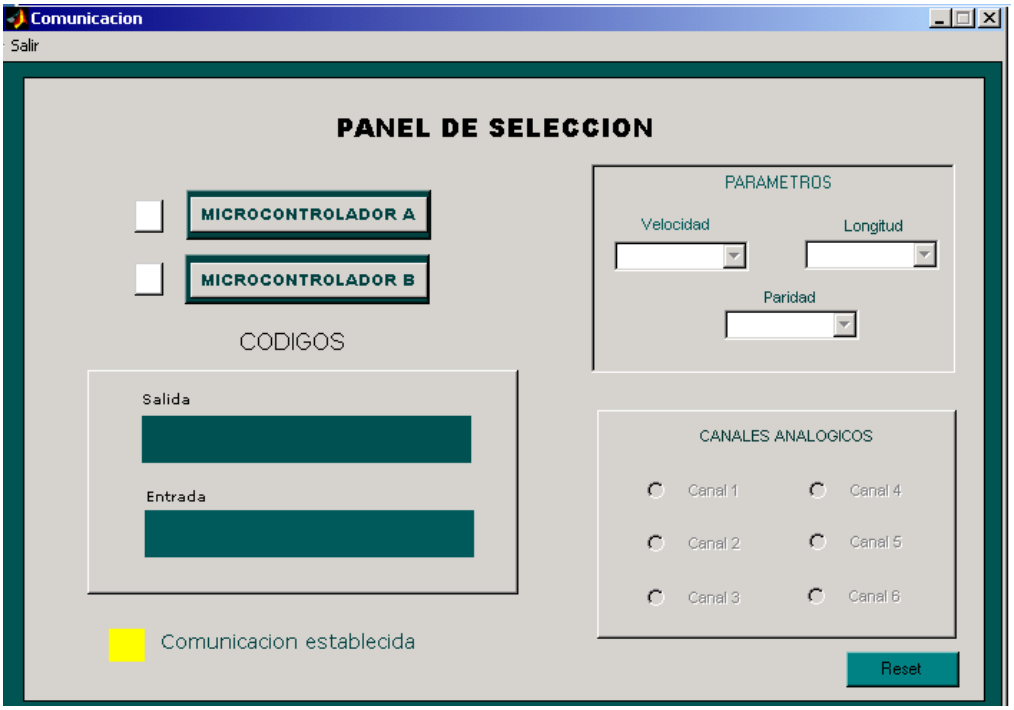


Figura 4.15 Ventana para la Selección de Parámetros

4.4.3 VENTANA PARA LA ADQUISICIÓN DE LOS DATOS

En esta pantalla se presenta la siguiente información:

- Trama recibida en ASCII
- Datos recibidos en decimal
- Gráfico en coordenadas del datos recibidos en voltios
- Controles e indicadores.

La ventana se diseñó utilizando entre otros textos estáticos, un gráfico de coordenadas, cuadros de textos, deslizadores, control de listas y botones.

Esta pantalla se indica en la figura 4.16.

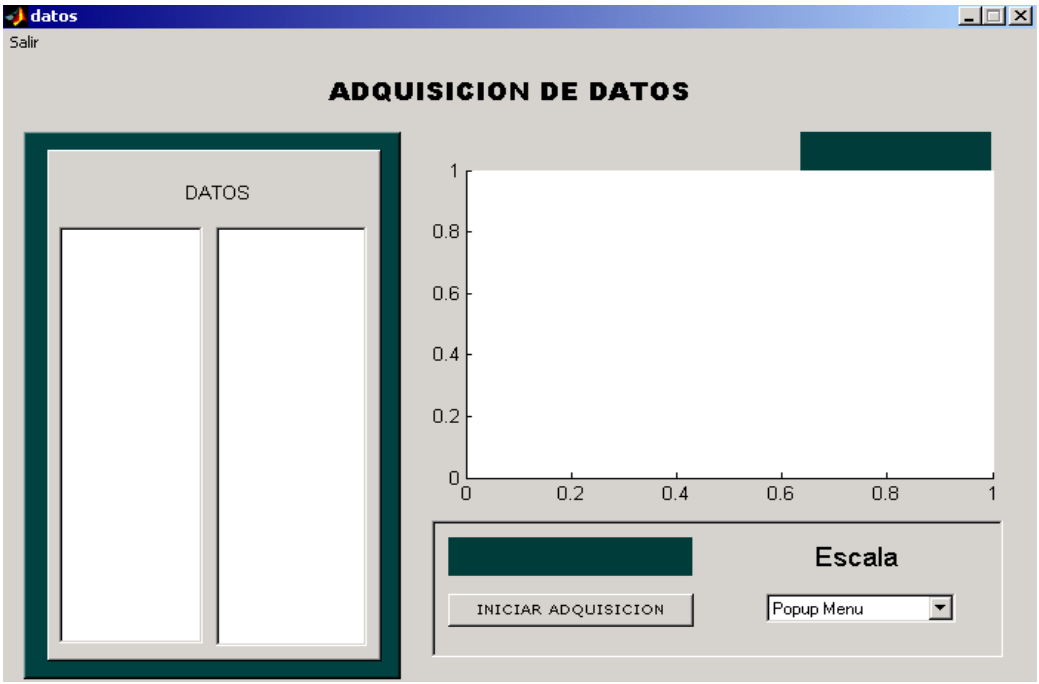


Figura 4.16 Ventana para la Adquisición de Datos

4.5 SOFTWARE DE COMUNICACIÓN CON MATLAB

Para la comunicación entre el microcontrolador y Matlab se requieren de los siguientes pasos básicos:

- Crear un objeto del puerto serie, que para este caso utiliza el COM1.

- Fijar los parámetros de comunicación por defecto, estos son: trabajo asincrónico y continuo, 9600 baudios, 8 bits del dato y sin paridad.

- Transmitir en códigos ASCII al microcontrolador los parámetros de comunicación que desee el usuario. Entre cada uno se espera la respuesta del microcontrolador, también en ASCII, esto con el objetivo de poder detectar fallas en la comunicación.

- Enviar al microcontrolador los códigos ASCII que informan la necesidad de realizar una operación de lectura.

- Recibir las datos ASCII de la tarjeta de adquisición e interpretarlos para graficar en los niveles de tensión adecuados.

Para cumplir con lo anterior, las instrucciones básicas empleadas en la comunicación son las que se indican a continuación, debiendo aclararse que **micro** es el nombre del objeto serial creado:

- micro = serial (COM1)
- fopen(micro)
- fprintf = (micro, 'dato', 'async')
- readasync (micro)

- fscanf(micro, '%c', size)
- stopasync(micro)
- fclose(micro)
- delete(micro)
- clear micro

La función de las instrucciones anteriores, así como los efectos y sus propiedades se detallaron en el capítulo 2; lo que si debe puntualizarse es que la eficiencia con la que trabajan está íntimamente ligada al orden con el que se las emplea.

En la tabla 4.14 se presentan las funciones más importantes utilizadas para la manipulación de los datos y el despliegue de los resultados en forma de onda.

Tabla 4.14 Funciones para la Obtención de los Resultados Gráficos

FUNCIÓN	CARACTERÍSTICA
SIZE	Da la dimensión de un arreglo.
FIX	Redondea y aumenta ceros a los elementos de un arreglo.
CAT	Concatena arreglos.
HEX2DEC	Convierte de hexadecimal a decimal
PLOT	Dibuja en dos dimensiones.
HOLD	Añade o reemplaza objetos en un gráfico.
DRAWNOW	Completa el gráfico durante los eventos.

En el anexo B se encuentra el programa realizado para las pantallas.

CAPÍTULO V

ANÁLISIS DE RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

5.1 ANÁLISIS DE RESULTADOS

Para el funcionamiento de la aplicación, se requiere conectar a la tarjeta diseñada (ANEXO A) las fuentes de corriente continua (polarización), el puerto serie de la computadora mediante el cable construido para el efecto y las señales analógicas a adquirir. En cuanto al software, se debe instalar la aplicación en Matlab 6.5 y correrla siguiendo las instrucciones del manual de usuario (ANEXO C).

Para analizar el funcionamiento del diseño realizado se consideran los siguientes aspectos básicos:

- Parámetros de comunicación.
- Funcionamiento del protocolo ASCII.
- Adquisición de señales de corriente continua.
- Adquisición de señales de corriente alterna.

5.1.1 PARÁMETROS DE LA COMUNICACIÓN

Dentro de este aspecto se toma en cuenta fundamentalmente que Matlab es un paquete poderoso, eficiente y probado a nivel mundial, por lo tanto si hay fallas en la comunicación, en cuanto a la velocidad y la longitud del dato, estas recaen directamente en los valores especificados en el microcontrolador.

Bajo la consideración anterior, se comprobó que si Matlab trabaja a una velocidad, seleccionada ésta con la propiedad BaudRate y el microcontrolador está seteado a un valor diferente, el computador recibe datos diferentes considerados como basuras.

Las pruebas de la velocidad se realizaron con las cuatro posibilidades que presenta la aplicación como son 9600, 4800, 2400 y 1200 baudios.

La veracidad en el cambio de los baudios para la comunicación también se verifica con la ayuda de un osciloscopio, el mismo que arroja diversos valores en el período de la onda según la velocidad seleccionada.

En cuanto a la longitud de los datos y al chequeo de la paridad se procede a realizar la misma prueba de la velocidad, esto es modificar el parámetro respectivo sólo en Matlab y analizar el dato que el microcontrolador envía, obteniéndose basuras si no están seteados con el mismo valor.

5.1.2 FUNCIONAMIENTO DEL PROTOCOLO ASCII

Para la visualización de la forma de trabajo del protocolo ASCII se incluyen indicadores del dato que envía Matlab (HOST) y del que recibe desde el microcontrolador (TERMINAL), tanto en la selección de los parámetros de comunicación cuanto en la adquisición de datos.

Se constató que en condiciones correctas la comunicación fluye eficientemente y se reciben los mensajes esperados. En la figura 5.1 se presenta como ejemplo la pantalla con el dato ASCII que recibe la PC del microcontrolador A para que continúe, luego de escoger la velocidad de 9600 baudios y una longitud de 8 bits, previa a selección de una entrada analógica.

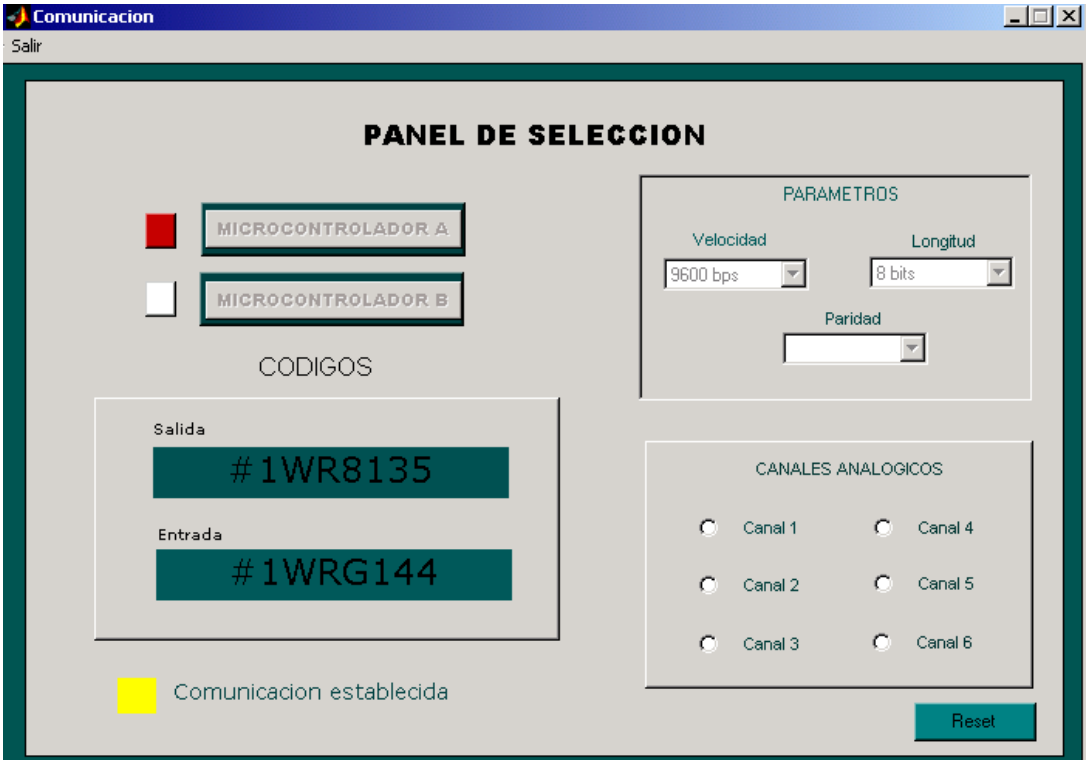


Figura 5.1 Pantalla que presenta el funcionamiento del Protocolo ASCII

Con el objeto de probar la respuesta de la aplicación a fallas en la comunicación se forzó un dato erróneo y se obtiene el mensaje que utiliza el protocolo ASCII para este fin. En la figura 4.2 se indica la pantalla con la respuesta del Microcontrolador B al recibir el dato para escoger la paridad par erróneo.

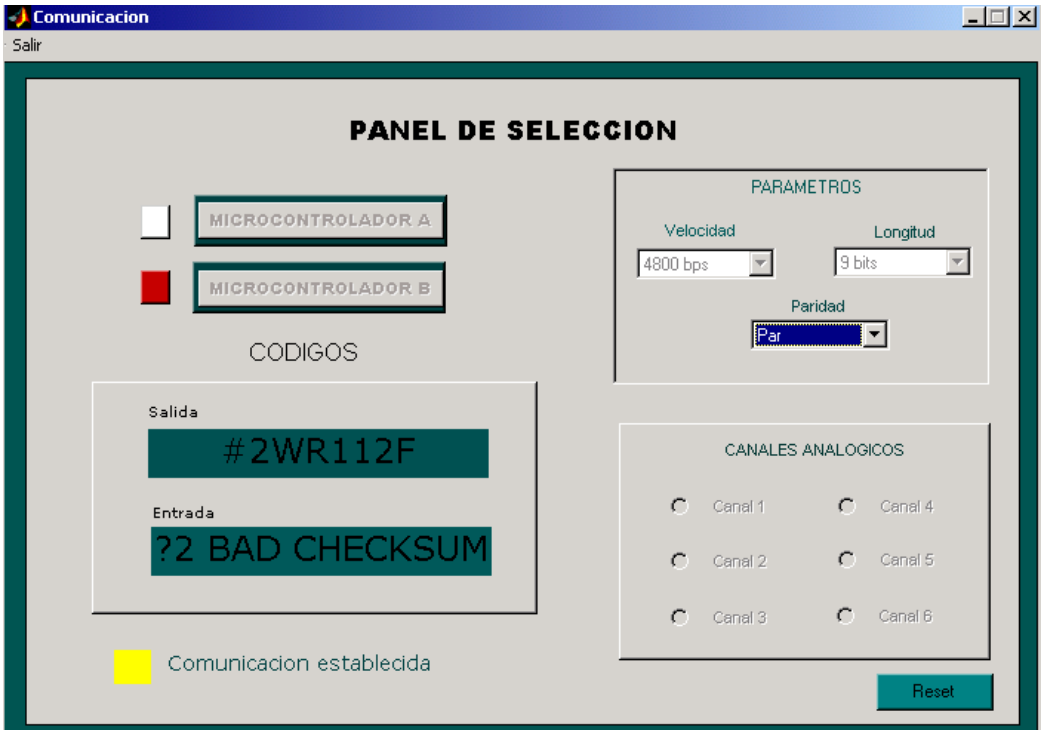


Figura 5.2 Pantalla que indica Fallas en la Comunicación

De acuerdo a lo anterior, se corrobora que el protocolo implementado trabaja conforme a los lineamientos teóricos del mismo.

En cuanto a la eficiencia del protocolo es apenas del 25 % ya que cada dato enviado desde el microcontrolador hacia Matlab consta de 12 bytes de los cuales apenas 3 bytes poseen la información proveniente del ADC.

Por ejemplo, si los voltajes de referencia son 5 y 0 voltios, y en el Matlab se recibe el dato *1RD+19A1C7, quiere decir que se está trabajando con el Microcontrolador A en una operación de lectura y su dirección es la 1. El dato recibido es positivo y en códigos ASCII es el 19A, el valor 1C7 corresponde al checksum y sirve para comprobar si el dato es correcto.

Entonces de los 12 datos recibidos sólo los tres, correspondientes a 19A, son la información proveniente del ADC y el valor en decimal sería el siguiente:

$1 \times 16^2 + 9 \times 16^1 + 10 \times 16^0 = 410$

Este valor expresado como voltaje analógico sería el siguiente:

Voltios = 410x5/1024 = 2,0019

5.1.3 ADQUISICIÓN DE SEÑALES DE CORRIENTE CONTINUA

En el análisis de la adquisición de una señal de corriente continua se fijaron varios valores de tensión y se compararon las medidas de un voltímetro digital, un osciloscopio y la aplicación, obteniendo resultados muy satisfactorios.

En la tabla 5.1 se presentan los valores en ASCII y decimal para dos mediciones realizadas y en la figura 5.3 se indica la forma de onda obtenida para 1,5 voltios.

Tabla 5.1 Ejemplos de Voltajes DC Adquiridos

VOLTÍMETRO	ASCII	DECIMAL
2 V	*1RD+19A1C7	2,00195
4 V	*2RD+3331B8	3,99302

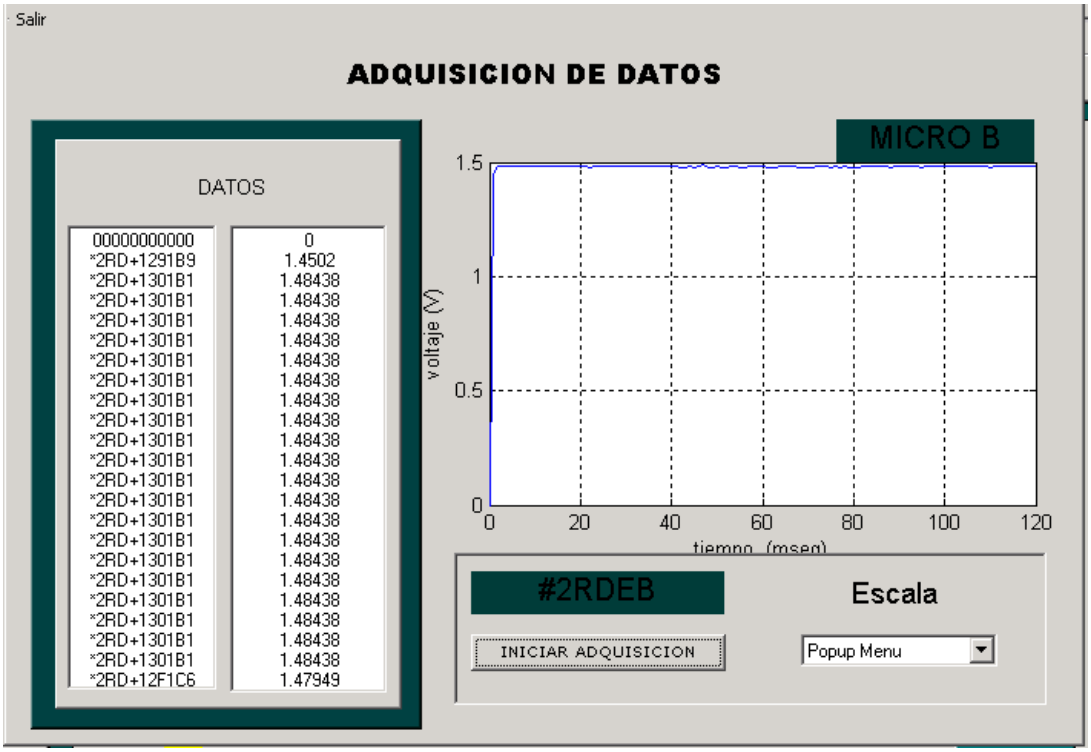


Figura 5.3 Pantalla que indica un Ejemplo en DC

5.1.4 ADQUISICIÓN DE SEÑALES DE CORRIENTE ALTERNA

Para las pruebas de la adquisición de señales alternas se conectaron varios tipos de ondas, tanto en su forma como en amplitud y frecuencia, verificándose su similitud con las formas que presenta un osciloscopio.

En este punto, se resalta el hecho de que es mucho más eficiente, en cuanto a la visualización de períodos completos, la adquisición de señales de frecuencia mayores a 0,5 KHz que de las menores a este valor. Esto se justifica plenamente debido a que si la señal es rápida se obtienen muestras de diferentes valores en el ADC para llenar los bancos; en cambio, si es lenta la señal los valores almacenados en un buen grupo de localidades es el mismo.

En la figura 5.4 se presenta una onda de 50 Hz y 1,5 voltios en la que se evidencia el efecto anterior.

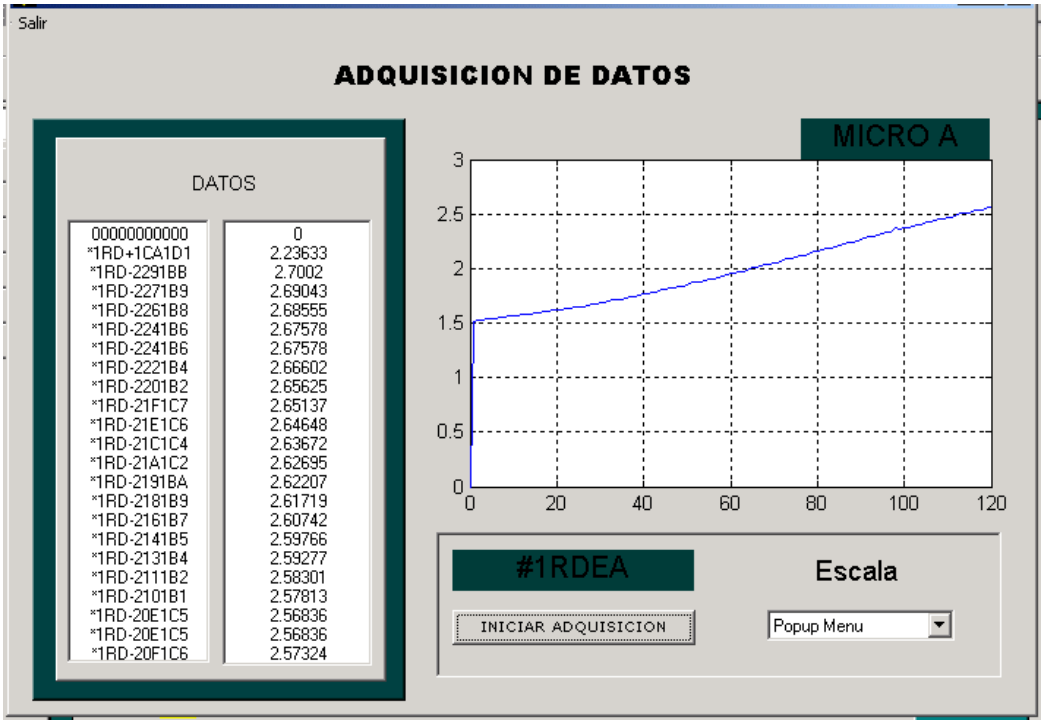


Figura 5.4 Pantalla que indica el Efecto a Baja Frecuencia

Por lo anterior, el tiempo de adquisición para graficar un período completo varía de acuerdo a la frecuencia de la señal recibida, esto debido también a la cantidad de bits que se desperdician al trabajar en el protocolo ASCII.

En las figuras 5.5, 5.6 y 5.7 se presenta la adquisición de una onda senoidal de 1 KHz, una onda triangular a 1,2 KHz y una cuadrada de 1,4 KHz con 1,3 voltios de amplitud pico a pico, respectivamente. En todas ellas se aprecia que el número de muestras es suficiente para la presentación de varios peíodos .

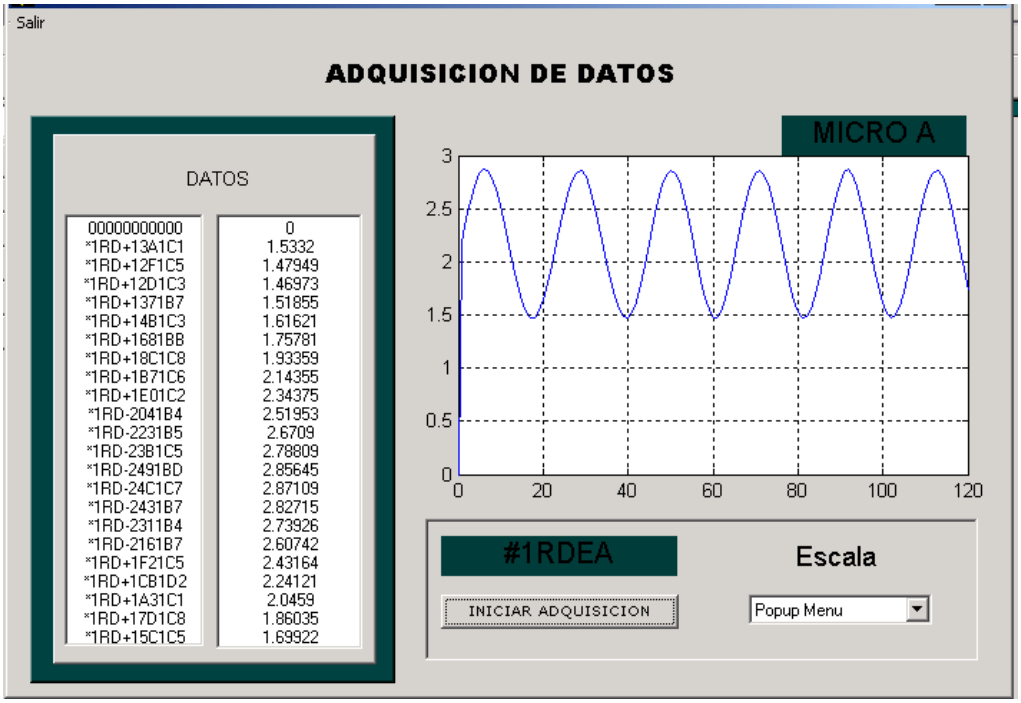


Figura 5.5 Pantalla que indica la Adquisición de una Onda Senoidal

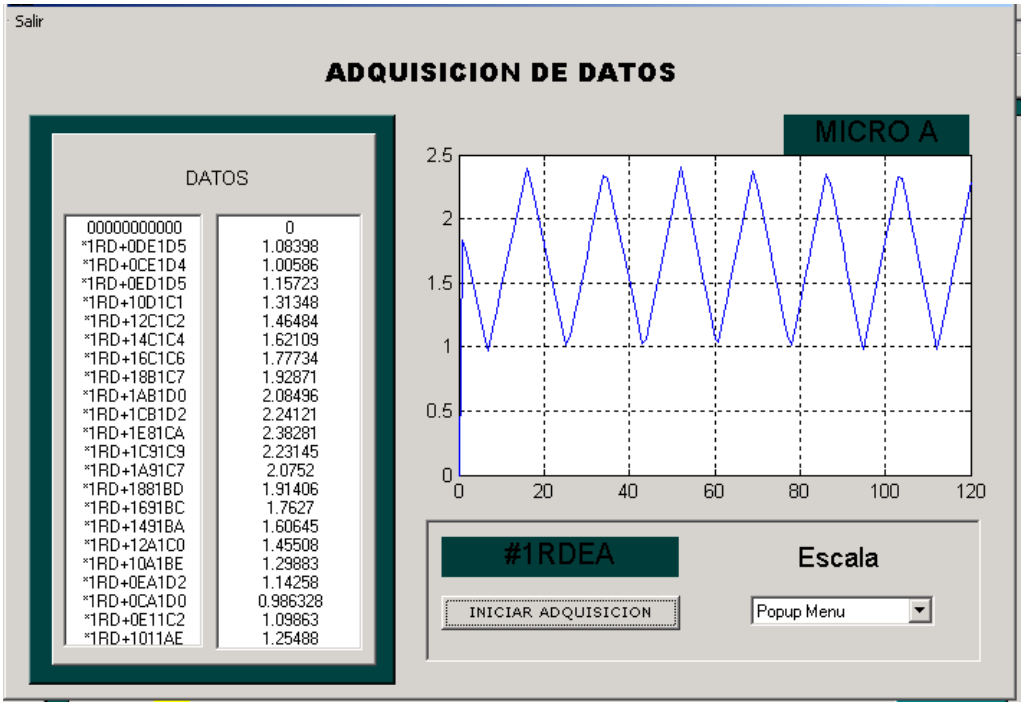


Figura 5.6 Pantalla que indica la Adquisición de una Onda Triangular

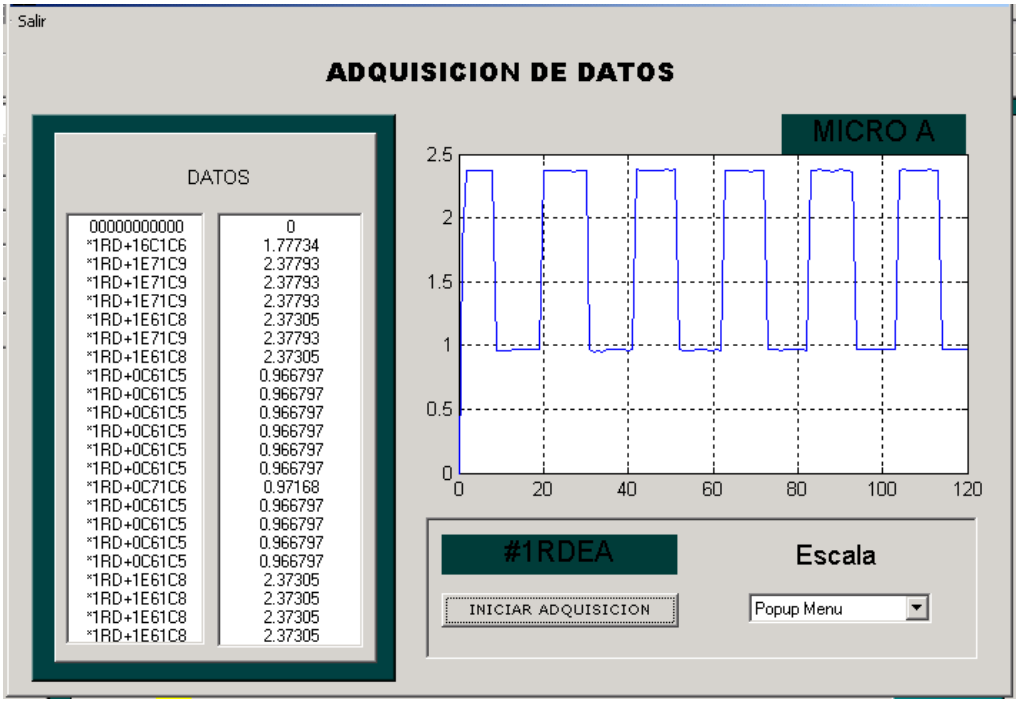


Figura 5.7 Pantalla que indica la Adquisición de una Onda Cuadrada

5.2 CONCLUSIONES

Una vez concluido el trabajo se obtienen las siguientes conclusiones:

- Como una conclusión central puedo indicar que se cumplieron con los objetivos propuestos en el plan, como son el de diseñar e implementar la tarjeta de adquisición de datos y la aplicación en el software Matlab.

- La aplicación diseñada cumple satisfactoriamente el objetivo de servir como una herramienta didáctica en la visualización de la forma de trabajo de los protocolos, el ASCII en particular, y de la comunicación tipo serie.

- El trabajo con el puerto serial es a la par, **interesante y frustrante**, ya que cuando no funciona, produce muchos dolores de cabeza. Esta conclusión se corrobora con el comentario manifestado por Steve Ciarcia en 1983 e incluido en el libro IBM PC PARA INGENIEROS de Ricardo Zelenovsky en la página10-1, ″Entre las experiencias más cansadas en cualquier profesión que utilice computadores, está la conexión de dos dispositivos seriales. No me refiero a la conexión de un terminal y un módem, esta conexión es como un pedazo de postre, mas, cualquier otra conexión puede ser un problema real″

- Los microcontroladores PIC son elementos poderosos para la adquisición de datos, tal es así que en la aplicación diseñada la velocidad de adquisición está en el rango de los megahertzios, razón por la cual debía "esperar" que el Matlab reciba esa información.

- La comunicación serie del microcontrolador PIC16F877 trabaja eficientemente a rangos de velocidad estandarizadas y con longitudes de 8 o 9 bits, pero el programador debe crear el software para incluir el bit de paridad deseado, ya que éste parámetro se encuentra abierto para el usuario.

- El crear el protocolo ASCII versión larga en el microcontrolador para el envío de datos disminuye la eficiencia ya que se desperdician muchos bits, pero incrementa más la seguridad en la comunicación.

- La limitación del microcontrolador, en cuanto a la adquisición de datos, está relacionada con la capacidad de su memoria RAM, ya que la velocidad de adquisición de cada dato es realmente alta, comparable a la de conversores como el ADC 0808.

- Como consecuencia de la velocidad del ADC se optó por almacenar las muestras en las localidades disponibles de la memoria de datos del microcontrolador, esto permite al Matlab recibir los datos a las diferentes velocidades con que se puede trabajar en la aplicación, con mayor tranquilidad y sin temor de saturar su buffer.

- Es necesario implementar un sistema de tal manera que no se presenten problemas en su operación, el diseño realizado permite adquirir una señal (formada por un grupo de muestras adquiridas) a la vez, esto debido a que la comunicación es serie, es decir bit a bit.

- Es necesario acondicionar las señales de comunicación serie, por lo tanto el circuito MAX232 es un elemento indispensable para la comunicación serie entre la PC y los microcontroladores, ya que convierte eficientemente los voltajes, tanto TTL a RS232 como RS232 a TTL.

- La utilización de la comunicación serial asincrónica aunque es más lenta tiene como ventaja la de utilizar menos líneas, entonces para la conexión entre la computadora y la tarjeta es necesario un cable de tres hilos únicamente (TX, RX y GND), ya que no se emplean las señales de MODEM de la interfase RS232.

- Debido a que cuando los microcontroladores están en espera colocan la línea de comunicación en un nivel lógico alto, es necesario asegurar que al MAX232 y por ende a la PC llegue siempre la información de uno solo de ellos, esto con el fin de evitar problemas de realimentación y también de que nunca reciba datos la computadora de los dos a la vez. Esta función cumple eficientemente un multiplexor de 2 a 1 cuyas entradas son los RX de los dos microcontroladores y su salida se conecta a la entrada al transceiver.

- El paquete Matlab posee instrucciones sencillas para el manejo del puerto serie en su operación básica, pero que deben seguir un orden muy definido si trabajan en conjunto, que al no respetarse producen fallos y abandono del paquete.

- Ya que Matlab es una calculadora muy poderosa, la decodificación, interpretación y visualización de los datos ASCII recibidos se realiza correctamente.

- Las interfaces gráficas que presenta el Matlab 6.5 no son tan elaboradas como las de otros lenguajes de programación visual, pero son sencillas de manejar y para aplicaciones que no requieren mucha resolución brindan las garantías necesarias.

- Los indicadores de texto diseñados se constituyen en el recurso didáctico de la aplicación, objetivo importante del trabajo propuesto.

- La comunicación serie requiere del cumplimiento de características físicas especiales en su implementación, para la conexión de la tarjeta a la PC se utiliza un cable de 10 metros de largo, con lo cual se verifica también el parámetro longitud especificado para este tipo de comunicación.

- En el fundamento teórico se incluyen algunos aspectos que no se utilizaron en la aplicación, pero que pueden servir de referencia y apoyo bibliográfico para otros trabajos similares y/o complementarios.

- La realización de este trabajo de grado, me permitió consolidar los conocimientos de temas tales como la comunicación serie de los PIC y el manejo de la GUI de Matlab.

RECOMENDACIONES

De las experiencias obtenidas puedo emitir las siguientes recomendaciones:

- En la realización de cualquier tipo de trabajo se deben analizar y acatar las sugerencias de los fabricantes de elementos y software, esto evitará errores y demoras.
- Debido a que el microcontrolador 16F877 es muy veloz en su adquisición sería mejor utilizarlo en aplicaciones de tipo paralelo, ya que en serie disminuye su eficiencia, peor aún si se emplea el protocolo ASCII.
- Para manejar una mayor cantidad de muestras sería necesario usar un microcontrolador que posea más capacidad en su RAM, ya que el PIC16F877 sólo dispone de 240 localidades libres para este efecto.
- Cuando se trabaja con el puerto serie en Matlab se debe tomar la precaución de abrir y cerrar el objeto cuando se desea entablar y cerrar la comunicación respectivamente, ya que si no se cierra adecuadamente no se permite el trabajo con ningún dispositivo y debe salirse del paquete.
- Respetar las características de corrientes y voltajes de los circuitos integrados, con el fin de evitar daños en los mismos.
- Conectar la aplicación siguiendo los lineamientos del Manual de Usuario.
- Fomentar nuevos trabajos con la comunicación serie de Matlab, ya que tiene muchas posibilidades y ésta es apenas una referencia.
- Se debe continuar con la política de realizar trabajos aplicativos, ya que con estos se afianzan los conocimientos teóricos adquiridos en las aulas y laboratorios.

BIBLIOGRAFÍA

- ÁNGULO, ROMERO y ÁNGULO, "Microcontroladores PIC16F87X. Diseño práctico y aplicaciones"
- ZELENOVSKY Ricardo, "IBM PC para Ingenieros", Escuela Politécnica del Ejército
- PÉREZ L. César, "Matlab y sus Aplicaciones en la Ciencia y la Ingeniería", Madrid 2002
- MANUALES DE MATLAB
- MANUALES DEL PIC16F877 – Microchip
- HELPMatlab
- GALARZA Z. Eddie, "Copiados de la asignatura Comunicación para Instrumentación", Escuela Politécnica del Ejército – Latacunga 2004
- MANUAL ECG

ENLACES

- http://www.redeya.com
- http://www.burr-brown.com
- http://www.itlp.edu.mx/publica/tutoriales/telepro/t2_3.htm
- http://www.itlp.edu.mx/publica/tutoriales/telepro/t1_3.htm
- http://www.itlp.edu.mx/publica/tutoriales/telepro/t3_3.htm
- MATLABTutorial - Javier García de Jalón de la Fuente (jgjalon≅etsii.upm.es)
- http://www.Microchip.com

ANEXOS

ADQUISICIÓN DE DATOS CON MICROCONTROLADOR UTILIZANDO EL PROTOCOLO ASCII

1. CONEXIÓN DE LA TARJETA A LA PC

- Conecte el cable con conector DB9 a la PC y a la tarjeta, en los conectores colocados para el efecto.
- Polarice la tarjeta con 5 voltios y tierra, en los conectores ahí especificados.
- Conecte los voltajes de referencia en los sitios preparados para el efecto (Vref + y Vref-).
- Conecte las señales analógicas en las entradas que desee: I1.1, I1.2, I1.3, I1.4, I1.5, I1.6, I2.1, I2.2, I2.3, I2.4, I2.5 e I2.6.
- Revise las conexiones nuevamente y apague todas las fuentes continuas y alternas.

2. CONEXIÓN DEL SOFTWARE

- Cargue el ejercicio con nombre en COMUNICACIÓN.
- Aparece la pantalla para seleccionar los parámetros de comunicación.
- Corra la aplicación, pero no seleccione nada todavía.
- Energice la tarjeta y las señales análogas.

3. UTILIZACIÓN DE LA APLICACIÓN

- Seleccione el microcontrolador.
- Seleccione la velocidad de trabajo.
- Seleccione la longitud del dato.
- Si escogió 9 bits, seleccione la paridad.
- Seleccione la entrada analógica.
- Aparece la pantalla para la adquisición de datos.
- Pulse Iniciar la Adquisición.
- Debe visualizar la onda y los valores tanto en ASCII como en voltaje.
- Si desea cambiar las escalas de tiempo utilice el campo especificado con ese nombre.
- Para escoger otra entrada analógica, resetee y vuelva a empezar.
- Para terminar la aplicación pulse la tecla Salir

PROGRAMA DE LA PANTALLA PARA LA ADQUISICIÒN DE DATOS

```
function varargout = datos(varargin)
% DATOS M-file for datos.fig
%      DATOS, by itself, creates a new DATOS or raises the existing
%      singleton*.
%
%      H = DATOS returns the handle to a new DATOS or the handle to
%      the existing singleton*.
%
%      DATOS('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DATOS.M with the given input arguments.
%
%      DATOS('Property','Value',...) creates a new DATOS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before datos_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to datos_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help datos


% Last Modified by GUIDE v2.5 31-Jul-2005 22:08:20
```

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @datos_OpeningFcn, ...
                   'gui_OutputFcn',  @datos_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before datos is made visible.
function datos_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to datos (see VARARGIN)

% Choose default command line output for datos
```

```
handles.output = hObject;


% Update handles structure
guidata(hObject, handles);


% UIWAIT makes datos wait for user response (see UIRESUME)
% uiwait(handles.figure1);


posicion = get(0,'DefaultFigurePosition');
unidadv = get(hObject,'Units');
set(hObject,'Units','pixels');
posicion1 = get(hObject,'Position');
ancho = posicion1(3);
largo = posicion1(4);
if isempty(gcbf)                %gcbf
   unidadwin=get(0,'Units');    %ScreenUnits
   set(0,'Units','pixels');
   unidaddim = get(0,'ScreenSize');    %ScreenSize
   set(0,'Units',unidadwin);
   posicion(1) = 1/2*(unidaddim(3)-ancho);
   posicion(2) = 2/3*(unidaddim(4)-largo);
else
   GCBFOldUnits = get(gcbf,'Units');
   set(gcbf,'Units','pixels');
   GCBFPos = get(gcbf,'Position');
   set(gcbf,'Units',GCBFOldUnits);
   posicion(1:2) = [(GCBFPos(1) + GCBFPos(3)/2) - ancho/2,...
         (GCBFPos(2) + GCBFPos(4)/2 ) - largo/2];
end
```

```
posicion(3:4)= [ancho largo];
set(hObject, 'Position',posicion);
set(hObject, 'Units',unidadv);

global micro;
save micro;
global controla;
save controla;
datos3 = 0;
global p0;
p0 = 0;
save p0;
global pics;
save pics;
set(handles.procesador,'string',pics);
global canales;
save canales;
set(handles.analogico,'string',canales);

% --- Outputs from this function are returned to the command line.
function varargout = datos_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
delete(hObject);


% --------------------------------------------------------------------
function salgo_Callback(hObject, eventdata, handles)
% hObject    handle to salgo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
```

```
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pushbutton3 as text
%        str2double(get(hObject,'String')) returns contents of pushbutton3 as a double


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3


% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu4 contents as cell array
```

```
%       contents{get(hObject,'Value')} returns selected item from popupmenu4


% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu5 contents as cell array
%       contents{get(hObject,'Value')} returns selected item from popupmenu5


% --- Executes during object creation, after setting all properties.
```

```
function popupmenu6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu6 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu6



% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2


% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3


% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton4
```

```
% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton8


% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton9


% --- Executes on button press in radiobutton10.
function radiobutton10_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton10


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mic1.
function mic1_Callback(hObject, eventdata, handles)
% hObject    handle to mic1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = [0.778 0 0];
set(handles.color1,'backgroundColor',a);


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in mic2.
function mic2_Callback(hObject, eventdata, handles)
% hObject    handle to mic2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a = [0.778 0 0];
set(handles.color2,'backgroundColor',a);

% --- Executes on button press in color1.
function color1_Callback(hObject, eventdata, handles)
% hObject    handle to color1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in Color2.
function Color2_Callback(hObject, eventdata, handles)
% hObject    handle to Color2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in color2.
function color2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to color2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton19.
function pushbutton19_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton20.
function pushbutton20_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function datoa_CreateFcn(hObject, eventdata, handles)
% hObject    handle to datoa (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function datoa_Callback(hObject, eventdata, handles)
% hObject    handle to datoa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of datoa as text
%        str2double(get(hObject,'String')) returns contents of datoa as a double



% --- Executes during object creation, after setting all properties.
function datod_CreateFcn(hObject, eventdata, handles)
% hObject    handle to datod (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end




function datod_Callback(hObject, eventdata, handles)
% hObject    handle to datod (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of datod as text
%        str2double(get(hObject,'String')) returns contents of datod as a double




function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of togglebutton2 as text
%        str2double(get(hObject,'String')) returns contents of togglebutton2 as a double



% --- Executes on button press in recibir.
```

```
function recibir_Callback(hObject, eventdata, handles)
% hObject    handle to recibir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

   global micro;
   save micro;
   global controla;
   save controla;
   datos3 = 0;
   global p0;
   save p0;
   m = 'qw';
   n = 1;
   na = '00000000000';
   cont = 0;

      if controla == 1
         set(handles.analogico,'string','#1RDEA');
      else
         set(handles.analogico,'string','#2RDEB');
      end

      b = 1;

  if controla == 1

      while m ~= 'G'
         fprintf(micro,'#1RDEA','async');
```

```
%readasync(micro);
%micro.BytesAvailable;
m = fscanf(micro,'%c',1351);

pt = size(m);
pt1 = pt(2)/11;
pt2 = fix(pt1);
pt3 = pt2*11;
for k = 0:11:pt3-1
   for q = 1:11
      w(q) = m(k+q);
   end
   na = cat(1,na,w);
end

dato1 = 0;
dato2 = 0;
j = 2;
ha = size(na);
for c = 2:ha
   dato1 = 0;
   dato2 = 0;
   j = 2;
   w = na(c,:);

   for i = 1:3
      dato = hex2dec(w(i+5));
      dato1 = dato*(16^j);
      j = j-1;
```

```
            dato2 = dato2 + dato1;
          end

          datos = ((dato2*5)/1024);
          datos3 = [datos3 datos];
        end

        p0 = size(datos3);
        p1 = [0:p0(2)-1];
        hold off
        plot(p1,datos3);
        grid on
        drawnow
        xlabel('tiempo  (mseg)');
        ylabel('voltaje (V)');
        cont = 0;
        set(handles.datod,'string',datos3);
        set(handles.datoa,'string',na);


        fclose(micro);
        fopen(micro);
        na = '00000000000';
        datos3 = 0;
        n = n+1;
      end

else
```

```
while m ~= 'G'


   fprintf(micro,'#2RDEB','async');
   %readasync(micro);
   %micro.BytesAvailable;
   m = fscanf(micro,'%c',1351);


   pt = size(m);
   pt1 = pt(2)/11;
   pt2 = fix(pt1);
   pt3 = pt2*11;
   for k = 0:11:pt3-1
      for q = 1:11
         w(q) = m(k+q);
      end
      na = cat(1,na,w);
   end


   dato1 = 0;
   dato2 = 0;
   j = 2;
   ha = size(na);
   for c = 2:ha
      dato1 = 0;
      dato2 = 0;
      j = 2;
      w = na(c,:);


      for i = 1:3
```

```
        dato = hex2dec(w(i+5));
        dato1 = dato*(16^j);
        j = j-1;
        dato2 = dato2 + dato1;
      end

      datos = ((dato2*5)/1024);
      datos3 = [datos3 datos];
    end

    p0 = size(datos3);
    p1 = [0:p0(2)-1];
    hold off
    plot(p1,datos3);
    grid on
    %drawnow
    xlabel('tiempo  (mseg)');
    ylabel('voltaje (V)');
    cont = 0;
    set(handles.datoa,'string',na);
    set(handles.datod,'string',datos3);
    pause(1)
    fclose(micro);
    fopen(micro);
    na = '00000000000';
    datos3 = 0;
    n = n+1;
  end
```

```
    end



% --- Executes during object creation, after setting all properties.
function popupmenu7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
% --- Executes on selection change in popupmenu7.
function popupmenu7_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu7 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu7
function [punt1,punt2] = escala(punt);
global p0;
punt2 = 5.35/2;
punt1 = fix(punt/2);
save p0;
```

PROGRAMA DEL MICROCONTROLADOR A

```
list p=16f877      ;SELECCION DEL MICROCONTROLADOR

;ETIQUETAS PARA LAS VARIABLES INTERNAS

indf equ 00
pcl equ 02
pclath equ 0a
estado equ 03
fsr equ 04
puertaa equ 05
puertac equ 07
puertae equ 09
pir1 equ 0c
rcsta equ 18
txreg equ 19
rcreg equ 1a
adresh_l equ 1e
adcon0 equ 1f

;ETIQUETAS PARA LAS VARIABLES PRINCIPALES

control equ 20
direc_aux equ 21
paridad  equ 22
parid_aux equ 23
cont1 equ 24
cont2 equ 25

;INICIALIZACION DE LOS VECTORES

org 0

goto inicio

;PROGRAMA PRINCIPAL

;INICIALIZACIONES
;TRABAJA POR DEFECTO RECIBIENDO INFORMACION DE LA PC
;A VELOCIDAD DE 9600 BAUDIOS, DATOS DE 8 BITS Y SIN CONTROL DE PARIDAD

        inicio  clrf puertac
                bsf estado,5        ;CAMBIO DE BANCO
                movlw 2f
                movwf puertaa       ;PINES DE LA PUERTA A ENTRADA
                movlw 0be
                movwf puertac       ;PIN PC0 LED INDICADOR Y PIN PC6 HABILITADO COMO
                                    SALIDA-TX, LOS DEMAS COMO ENTRADAS
                movlw 07
                movwf puertae       ;PINES DE LA PUERTA E ENTRADA
                movlw 0f
                movwf txreg         ;FRECUENCIA PARA TX-RX, 9600 BAUDIOS Y 10 MHZ
```

```
        movlw 88
        movwf adcon0    ;6 ENTRADAS ANALOGICAS Y JUSTIFICACION A LA
                        DERECHA ADC
        movlw 20
        movwf rcsta     ;HABILITACION DE LA TRASMISION
        bcf estado,5    ;REGRESO AL BANCO 0
        movlw 90
        movwf rcsta     ;HABILITACION DEL PUERTO SERIE, CON 8 BITS Y
                        RECEPCION CONTINUA ASINCRONA

otra_dir call Sdireccion    ;VA A RECIBIR LA DIRECCION
        btfss control,0     ;CHEQUEO DE LA DIRECCION
        goto otra_dir

        bcf 18,4
        call continuar
otra_vel call Svelocidad    ;VA A RECIBIR LA VELOCIDAD
        btfss control,1     ;CHEQUEO DE LA VELOCIDAD
        goto otra_vel

        bcf 18,4
        call continuar
        movf 5b,0
        bsf estado,5
        movwf txreg
        bcf estado,5
otra_lon call Slongitud     ;VA A RECIBIR LA LONGITUD
        btfss control,2     ;CHEQUEO DE LA LONGITUD
        goto otra_lon

        bcf 18,4
        call continuar
        btfss control,7
        goto sigent
 otra_par call Sparidad     ;VA A RECIBIR LA PARIDAD
                            btfss control,3         ;CHEQUEO DE LA PARIDAD
                            goto otra_par
                            bcf 18,4
                            call continuar
                            bsf rcsta,6                 ;9 BITS
                            bsf estado,5
                            bsf rcsta,6
                            bcf estado,5
                            goto otra_ent

sigent      bcf rcsta,6                         ;8 BITS
            bsf estado,5
            bcf rcsta,6
            bcf estado,5
otra_ent call Sentrada      ;VA A RECIBIR LA ENTRADA
            btfss control,4 ;CHEQUEO DE LA ENTRADA
            goto otra_ent

            bcf 18,4
            call continuar
otro_dat call Slectura      ;CONFIRMACION DE LECTURA
            btfss control,5 ;CHEQUEO DE LECTURA
            goto otro_dat

            bcf 18,4
```

```
otro_env call Stomar            ;TOMA LOS DATOS DEL ADC
         call Senviar           ;ENVIA LOS DATOS DEL ADC
         call Slectura
         btfss control,5
         goto termine

         bcf estado,7
         bcf estado,6
         bcf estado,5
         goto otro_env

termine  bcf estado,7
         bcf estado,6
         bcf estado,5
         bcf 18,4
         bcf adcon0,0
         bsf estado,5           ;Deshabilito la transmisión
         bcf 18,5
         bcf estado,5
         bcf puertac,0          ;led indicador del MC1
         goto inicio

;SUBRUTINAS DE SERVICIO

decodificar  addwf pcl,1
             retlw 30
             retlw 31
             retlw 32
             retlw 33
             retlw 34
             retlw 35
             retlw 36
             retlw 37
             retlw 38
             retlw 39
             retlw 41
             retlw 42
             retlw 43
             retlw 44
             retlw 45
             retlw 46

decod2       addwf pcl,1
             retlw 00
             retlw 01
             retlw 00
             retlw 01
             retlw 00
             retlw 01
             retlw 00
             retlw 01
             retlw 00

;SUBRUTINA PARA RECIBIR LA DIRECCION DEL PIC

Sdireccion   movlw 31        ;Es el micro 1
             movwf direc_aux
             movlw 40
             movwf fsr
```

```
otro    btfss pir1,5
        goto otro
        bcf pir1,5
        movf rcreg,0
        movwf indf
        incf fsr,1
        movlw 42
        xorwf fsr,0
        btfss estado,2
        goto siga
        movf direc_aux,0
        decf fsr,1
        xorwf indf,0
        btfss estado,2
        goto pasa
        incf fsr,1
siga    movlw 47
        xorwf fsr,0
        btfsc estado,2
        goto chequeo
        goto otro

chequeo clrf 52
        movlw 40
        movwf fsr
        movf indf,0
sume    incf fsr,1
        addwf indf,0
        movwf 52
        movlw 43
        xorwf fsr,0
        btfsc estado,2
        goto compare
        movf 52,0
        goto sume
compare movlw 0fd
        xorwf 52,0
        btfss estado,2
        goto Emal
        incf fsr,1          ;CODIGO DE ESCRITURA FD - MC1
        movlw 46
        xorwf indf,0
        btfss estado,2
        goto Emal
                incf fsr,1
        movlw 44
        xorwf indf,0
        btfsc estado,2
        goto contesto
Emal    call mal
pasa    bcf control,0
salga   return

contesto bsf control,0      ;aqui acepto la direccion
        bsf puertac,0       ;led indicador del MC1
        goto salga
```

;SUBRUTINA PARA CONTESTAR QUE CONTINUE

```
continuar
                movlw 23
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg             ;ENVIO LOS CODIGOS PARA CONTINUAR
                bsf estado,5
        seguv1  btfss 18,1
                goto seguv1
                bcf estado,5
                movlw 31                ;MICRO 1
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
        seguv2  btfss 18,1
                goto seguv2
                bcf estado,5
                movlw 57
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
        seguv3  btfss 18,1
                goto seguv3
                bcf estado,5
                movlw 52
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
        seguv4  btfss 18,1
                goto seguv4
                bcf estado,5
                movlw 47
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
        seguv5  btfss 18,1
                goto seguv5
                bcf estado,5
                movlw 31
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
        seguv6  btfss 18,1
                goto seguv6
                bcf estado,5
                movlw 34
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
```

```
        bsf estado,5
seguv7  btfss 18,1
        goto seguv7
        bcf estado,5
        movlw 34
        movwf 56
        call bit_paridad
        movf 56,0
        movwf txreg
        bsf estado,5
seguv8  btfss 18,1
        goto seguv8
        bcf estado,5
        movlw 0d
        movwf 56
        call bit_paridad
        movf 56,0
        movwf txreg
        bsf estado,5
seguv9  btfss 18,1
        goto seguv9
        bcf estado,5

        return

;SUBRUTINA PARA CONTESTAR ERROR DEL CHECKSUM

        mal     bcf 18,4
                movlw 3f        ;CARGA DEL MENSAJE
                movwf 70
                movlw 31        ;MC1
                movwf 71
                movlw 20
                movwf 72
                movlw 42
                movwf 73
                movlw 41
                movwf 74
                movlw 44
                movwf 75
                movlw 20
                movwf 76
                movlw 43
                movwf 77
                movlw 48
                movwf 78
                movlw 45
                movwf 79
                movlw 43
                movwf 7a
                movlw 4b
                movwf 7b
                movlw 53
                movwf 7c
                movlw 55
                movwf 7d
                movlw 4d
                movwf 7e
                movlw 0d
                movwf 7f
```

```
                movlw 70
                movwf fsr
envie           movf indf,0
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
seguerr btfss 18,1
                goto seguerr
                bcf estado,5
                incf fsr,1
                movlw 80
                xorwf fsr,0
                btfss estado,2
                goto envie
                return
```

;SUBRUTINA PARA RECIBIR LA VELOCIDAD

```
Svelocidad      bsf 18,4
                movlw 40
                movwf fsr
otro1           btfss pir1,5
                goto otro1
                bcf pir1,5
                movf rcreg,0
                movwf indf
                incf fsr,1
                movlw 4a
                xorwf fsr,0
                btfsc estado,2
                goto chequeo1
                goto otro1

chequeo1        bcf 18,4
                clrf 53
                clrf 52
                movlw 40
                movwf fsr
                movf indf,0
sume1           incf fsr,1
                addwf indf,0
                btfsc estado,0
                incf 53,1
                movwf 52
                movlw 45
                xorwf fsr,0
                btfsc estado,2
                goto compare1
                movf 52,0
                goto sume1

compare1        btfss estado,2
                goto salga1
                movlw 6c
                xorwf 52,0
                btfss estado,2
```

```
        goto segunda
        movlw 46
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto segunda
        incf fsr,1
        movlw 36
        xorwf indf,0
        btfss estado,2
        goto segunda
        incf fsr,1
        movlw 43
        xorwf indf,0
        btfsc estado,2
        goto velocidad1
segunda movlw 69
        xorwf 52,0
        btfss estado,2
        goto tercera
        movlw 46
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto tercera
        incf fsr,1
        movlw 36
        xorwf indf,0
        btfss estado,2
        goto tercera
        incf fsr,1
        movlw 39
        xorwf indf,0
        btfsc estado,2
        goto velocidad2
tercera movlw 63
        xorwf 52,0
        btfss estado,2
        goto cuarta
        movlw 46
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto cuarta
        incf fsr,1
        movlw 36
        xorwf indf,0
        btfss estado,2
        goto cuarta
        incf fsr,1
        movlw 33
        xorwf indf,0
        btfsc estado,2
        goto velocidad3
cuarta  movlw 60
        xorwf 52,0
        btfss estado,2
```

```
                goto veloerrada
                movlw 46
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto veloerrada
                incf fsr,1
                movlw 36
                xorwf indf,0
                btfss estado,2
                goto veloerrada
                incf fsr,1
                movlw 30
                xorwf indf,0
                btfsc estado,2
                goto velocidad4
veloerrada      call mal
                bcf control,1
        salga1  return

velocidad1      bsf control,1   ;aqui seteo la velocidad de 9600
                movlw 0f
                movwf 5b
                goto salga1

velocidad2      bsf control,1   ;aqui seteo la velocidad de 4800
                movlw 1e
                movwf 5b
                goto salga1

velocidad3      bsf control,1   ;aqui seteo la velocidad de 2400
                movlw 3e
                movwf 5b
                goto salga1

velocidad4      bsf control,1   ;aqui seteo la velocidad de 1200
                movlw 7e
                movwf 5b
                goto salga1


;SUBRUTINA PARA RECIBIR LA LONGITUD

    Slongitud   bsf 18,4
                movlw 40
                movwf fsr
          otro2 btfss pir1,5
                goto otro2
                bcf pir1,5
                movf rcreg,0
                movwf indf
                incf fsr,1
                movlw 49
                xorwf fsr,0
                btfsc estado,2
                goto chequeo2
                goto otro2

chequeo2        bcf 18,4
```

```
                clrf 53
                clrf 52
                movlw 40
                movwf fsr
                movf indf,0
        sume2   incf fsr,1
                addwf indf,0
                btfsc estado,0
                incf 53,1
                movwf 52
                movlw 44
                xorwf fsr,0
                btfsc estado,2
                goto compare2
                movf 52,0
                goto sume2
compare2        btfss estado,2
                goto salga2
                movlw 35
                xorwf 52,0
                btfss estado,2
                goto nueve
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto nueve
                incf fsr,1
                movlw 33
                xorwf indf,0
                btfss estado,2
                goto nueve
                incf fsr,1
                movlw 35
                xorwf indf,0
                btfsc estado,2
                goto longitud1
        nueve   movlw 36
                xorwf 52,0
                btfss estado,2
                goto longerrada
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto longerrada
                incf fsr,1
                movlw 33
                xorwf indf,0
                btfss estado,2
                goto longerrada
                incf fsr,1
                movlw 36
                xorwf indf,0
                btfsc estado,2
                goto longitud2
longerrada      call mal
                bcf control,2
```

```
          salga2    return

longitud1           bsf control,2     ;aqui seteo la longitud a 8 bits
                    bcf control,7
                    goto salga2

longitud2           bsf control,2     ;aqui seteo la longitud a 9 bits
                    bsf control,7
                    goto salga2

;SUBRUTINA PARA RECIBIR LA PARIDAD

Sparidad            bsf 18,4
                    clrf paridad
                    movlw 40
                    movwf fsr
          otro3     btfss pir1,5
                    goto otro3
                    bcf pir1,5
                    movf rcreg,0
                    movwf indf
                    incf fsr,1
                    movlw 49
                    xorwf fsr,0
                    btfsc estado,2
                    goto chequeo3
                    goto otro3

chequeo3            bcf 18,4
                    clrf 53
                    clrf 52
                    movlw 40
                    movwf fsr
                    movf indf,0
          sume3     incf fsr,1
                    addwf indf,0
                    btfsc estado,0
                    incf 53,1
                    movwf 52
                    movlw 44
                    xorwf fsr,0
                    btfsc estado,2
                    goto compare3
                    movf 52,0
                    goto sume3

compare3            btfss estado,2
                    goto salga3
                    movlw 2e
                    xorwf 52,0
                    btfss estado,2
                    goto impar
                    movlw 45
                    movwf fsr
                    movlw 31
                    xorwf indf,0
                    btfss estado,2
                    goto impar
                    incf fsr,1
```

```
                movlw 32
                xorwf indf,0
                btfss estado,2
                goto impar
                incf fsr,1
                movlw 45
                xorwf indf,0
                btfsc estado,2
                goto paridad1
        impar   movlw 2f
                xorwf 52,0
                btfss estado,2
                goto parerrada
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto parerrada
                incf fsr,1
                movlw 32
                xorwf indf,0
                btfss estado,2
                goto parerrada
                incf fsr,1
                movlw 46
                xorwf indf,0
                btfsc estado,2
                goto paridad2
parerrada       call mal
                bcf control,3
        salga3  return

                paridad1bsf control,3   ;aqui seteo la paridad par
                        bcf paridad,0
                        goto salga3

                paridad2bsf control,3   ;aqui seteo la paridad impar
                        bsf paridad,0
                        goto salga3


;SUBRUTINA PARA RECIBIR EL NUMERO DE LA ENTRADA

                Sentradabsf 18,4
                        movlw 40
                        movwf fsr
                otro4   btfss pir1,5
                        goto otro4
                        bcf pir1,5
                        movf rcreg,0
                        movwf indf
                        incf fsr,1
                        movlw 49
                        xorwf fsr,0
                        btfsc estado,2
                        goto chequeo4
                        goto otro4

        chequeo4        bcf 18,4
```

```
                clrf 53
                clrf 52
                movlw 40
                movwf fsr
                movf indf,0
        sume4   incf fsr,1
                addwf indf,0
                btfsc estado,0
                incf 53,1
                movwf 52
                movlw 44
                xorwf fsr,0
                btfsc estado,2
                goto compare4
                movf 52,0
                goto sume4

compare4        btfss estado,2
                goto salga4
                movlw 2d
                xorwf 52,0
                btfss estado,2
                goto ent1
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto ent1
                incf fsr,1
                movlw 32
                xorwf indf,0
                btfss estado,2
                goto ent1
                incf fsr,1
                movlw 44
                xorwf indf,0
                btfsc estado,2
                goto entrada0
        ent1    movlw 2e
                xorwf 52,0
                btfss estado,2
                goto ent4
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto ent4
                incf fsr,1
                movlw 32
                xorwf indf,0
                btfss estado,2
                goto ent4
                incf fsr,1
                movlw 45
                xorwf indf,0
                btfsc estado,2
                goto entrada1
        ent4    movlw 31
```

```
        xorwf 52,0
        btfss estado,2
        goto ent5
        movlw 45
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto ent5
        incf fsr,1
        movlw 33
        xorwf indf,0
        btfss estado,2
        goto ent5
        incf fsr,1
        movlw 31
        xorwf indf,0
        btfsc estado,2
        goto entrada4
ent5    movlw 32
        xorwf 52,0
        btfss estado,2
        goto ent6
        movlw 45
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto ent6
        incf fsr,1
        movlw 33
        xorwf indf,0
        btfss estado,2
        goto ent6
        incf fsr,1
        movlw 32
        xorwf indf,0
        btfsc estado,2
        goto entrada5
ent6    movlw 33
        xorwf 52,0
        btfss estado,2
        goto ent7
        movlw 45
        movwf fsr
        movlw 31
        xorwf indf,0
        btfss estado,2
        goto ent7
        incf fsr,1
        movlw 33
        xorwf indf,0
        btfss estado,2
        goto ent7
        incf fsr,1
        movlw 33
        xorwf indf,0
        btfsc estado,2
        goto entrada6
ent7    movlw 34
```

```
                xorwf 52,0
                btfss estado,2
                goto enterrada
                movlw 45
                movwf fsr
                movlw 31
                xorwf indf,0
                btfss estado,2
                goto enterrada
                incf fsr,1
                movlw 33
                xorwf indf,0
                btfss estado,2
                goto enterrada
                incf fsr,1
                movlw 34
                xorwf indf,0
                btfsc estado,2
                goto entrada7
enterrada       call mal
                bcf control,4
        salga4  return

        entrada0bsf control,4   ;aqui seteo la entrada 0 del ADC
                bcf adcon0,3
                bcf adcon0,4
                bcf adcon0,5
                goto salga4

        entrada1bsf control,4   ;aqui seteo la entrada 1 del ADC
                bsf adcon0,3
                bcf adcon0,4
                bcf adcon0,5
                goto salga4

        entrada4bsf control,4   ;aqui seteo la entrada 4 del ADC
                bcf adcon0,3
                bcf adcon0,4
                bsf adcon0,5
                goto salga4

        entrada5bsf control,4   ;aqui seteo la entrada 5 del ADC
                bsf adcon0,3
                bcf adcon0,4
                bsf adcon0,5
                goto salga4

        entrada6bsf control,4   ;aqui seteo la entrada 6 del ADC
                bcf adcon0,3
                bsf adcon0,4
                bsf adcon0,5
                goto salga4

        entrada7bsf control,4   ;aqui seteo la entrada 7 del ADC
                bsf adcon0,3
                bsf adcon0,4
                bsf adcon0,5
                goto salga4
```

```
;GENERACION DEL BIT DE PARIDAD

bit_paridad     movwf parid_aux
                clrf cont1              ;Cuenta el número de unos del dato
                movlw 08
                movwf cont2             ;Indica las 8 rotaciones
        bor     bcf estado,0
                rrf parid_aux
                btfsc estado,0
                incf cont1,1
                decfsz cont2,1
                goto bor
                movf cont1,0
                call decod2
                btfsc paridad,0
                goto invierta
        filtro  andlw 01
                movwf 55
                movlw 00
                xorwf 55
                btfss estado,2
                goto uno
                goto cero
                uno bsf estado,5
                bsf 18,0
                bcf estado,5
                goto vuelva
        cero    bsf estado,5
                bcf 18,0
                bcf estado,5
         vuelva return
        invierta movwf cont2
                comf cont2,0
                goto filtro


;CONFIRMAR QUE ES LECTURA DE DATOS

        Slectura bsf 18,4
                movlw 31                ;Es el micro 1
                movwf direc_aux
                movlw 40
                movwf fsr
        Lotro   btfss pir1,5
                goto Lotro
                bcf pir1,5
                movf rcreg,0
                movwf indf
                incf fsr,1
                movlw 42
                xorwf fsr,0
                btfss estado,2
                goto Lsiga
                movf direc_aux,0
                decf fsr,1
                xorwf indf,0
                btfss estado,2
                goto Lsalga
                incf fsr,1
```

```
        Lsiga   movlw 47
                xorwf fsr,0
                btfsc estado,2
                goto Lchequeo
                goto Lotro


      Lchequeo  clrf 52   ;CONFIRMAR QUE ES LECTURA DE DATOS DEL MC1
                movlw 40
                movwf fsr
                movf indf,0
        Lsume   incf fsr,1
                addwf indf,0
                movwf 52
                movlw 43
                xorwf fsr,0
                btfsc estado,2
                goto Lcompare
                movf 52,0
                goto Lsume

      Lcompare  movlw 0ea           ;CODIGO DE LECTURA EA - MC1
                xorwf 52,0
                btfss estado,2
                goto Lmal
                incf fsr,1
                movlw 45
                xorwf indf,0
                btfss estado,2
                goto Lmal
                incf fsr,1
                movlw 41
                xorwf indf,0
                btfsc estado,2
                goto Slec
        Lmal    call mal
                bcf control,5
                goto Lsalga
        Slec    bsf control,5
        Lsalga  return


;TOMAR LOS DATOS DEL ADC Y GUARDARLOS EN LOS BANCOS

      Stomar  bcf 18,4

              bsf adcon0,7
              bcf adcon0,6          ;TAD EN 32.TOSC
              bsf adcon0,0

              bcf estado,7          ;BANCO 1
              bcf estado,6          ;BANCO 1
              bsf estado,5          ;BANCO 1
              movlw 0a0
              movwf fsr
    sigab11   bcf estado,5
              bcf pir1,6
              bsf adcon0,2          ;INICO DE CONVERSION DEL ADC
    sigab1    btfss pir1,6
              goto sigab1
```

```
        movf adresh_l,0
        bsf estado,5
        movwf indf
        incf fsr,1
        movf adresh_l,0
        movwf indf
        incf fsr,1
        movlw 0f0
        xorwf fsr,0
        btfss estado,2
        goto sigab11

        bsf estado,7            ;BANCO 2
        bsf estado,6            ;BANCO 2
        bcf estado,5            ;BANCO 2
        movlw 20
        movwf fsr
sigab21 bcf estado,7
        bcf estado,6
        bcf pir1,6
        bsf adcon0,2            ;INICO DE CONVERSION DEL ADC
sigab2  btfss pir1,6
        goto sigab2
        movf adresh_l,0
        bsf estado,7
        bsf estado,6
        movwf indf
        incf fsr,1
        bcf estado,7
        bcf estado,6
        bsf estado,5
        movf adresh_l,0
        bsf estado,7
        bsf estado,6
        bcf estado,5
        movwf indf
        incf fsr,1
        movlw 70
        xorwf fsr,0
        btfss estado,2
        goto sigab21


        bsf estado,7            ;BANCO 3
        bsf estado,6            ;BANCO 3
        bsf estado,5            ;BANCO 3
        movlw 0a0
        movwf fsr
sigab31 bcf estado,7
        bcf estado,6
        bcf estado,5
        bcf pir1,6
        bsf adcon0,2            ;INICO DE CONVERSION DEL ADC
sigab3  btfss pir1,6
        goto sigab3
        movf adresh_l,0
        bsf estado,7
        bsf estado,6
        bsf estado,5
        movwf indf
```

```
incf fsr,1
bcf estado,7
bcf estado,6
movf adresh_l,0
bsf estado,7
bsf estado,6
movwf indf
incf fsr,1
movlw 0f0
xorwf fsr,0
btfss estado,2
goto sigab31

bcf adcon0,0
bcf estado,7
bcf estado,6
bcf estado,5
return
```

;ENVIARLOS LOS DATOS DEL ADC EN ASCII A LA PC

```
Senviar     bcf adcon0,0
            bcf estado,7
            bcf estado,6
            bcf estado,5

            movlw 2a
            movwf 60
            movlw 31            ;Responde el MC1
            movwf 61
            movlw 52
            movwf 62
            movlw 44
            movwf 63


            bcf estado,7        ;DATOS DEL BANCO1
            bcf estado,6
            bsf estado,5

            movlw 0a0
            movwf fsr
      conB1 movf indf,0         ;tomo los bits mas significativos del ADC
            bcf estado,5
            movwf 65            ;almacenar la parte alta de la conversion
            btfss 65,1
            goto positivo
            movlw 2d            ;negativo
            movwf 64
            goto acondicB1
   positivo movlw 2b            ;positivo
            movwf 64
acondicB1   movf 65,0
            andlw 03
            addlw 30
            movwf 65
            bsf estado,5
            incf fsr,1
            movf indf,0     ;tomo los bits menos significativos del ADC
```

```
                bcf estado,5
                movwf 66
                movwf 67
                bsf estado,5
                movf fsr,0
                bcf estado,5
                movwf 5a        ;guardo el FSR del banco1
                swapf 66,1
                movlw 0f
                andwf 66,0
                call decodificar
                movwf 66        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 67,0
                call decodificar
                movwf 67        ;almacenar la parte bajaL de la conversion
                clrf 68         ;generacion del checksum
                clrf 54         ;auxiliar de el llevo
                movlw 60
                movwf fsr
                movf indf,0
                movwf 53        ;auxiliar de la suma
sigasumando     incf fsr,1
                movlw 68
                xorwf fsr,0
                btfsc estado,2
                goto deccheckSum
                movf 53,0
                addwf indf,0
                btfsc estado,0
                incf 54,1
                movwf 53
                goto sigasumando

deccheckSum     movf 54,0
                addlw 30
                movwf 68
                movf 53,0
                movwf 69
                movwf 6a
                swapf 69,1
                movlw 0f
                andwf 69,0
                call decodificar
                movwf 69        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 6a,0
                call decodificar
                movwf 6a
transmita       bsf estado,5    ;Inicio de la transmisión del ADC
                bsf 18,5
                bcf estado,5
                movlw 60
                movwf fsr
        continue movf indf,0
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
```

```
segudat btfss 18,1
        goto segudat
        bcf estado,5
        incf fsr,1
        movlw 6b
        xorwf fsr,0
        btfss estado,2
        goto continue

        movf 5a,0
        bcf estado,7      ;DATOS DEL BANCO1
        bcf estado,6
        bsf estado,5
        movwf fsr
        incf fsr,1
        movlw 0f0
        xorwf fsr,0
        btfss estado,2
        goto conB1


        bcf 18,4
        bsf estado,7      ;DATOS BANCO 2
        bsf estado,6
        bcf estado,5
        movlw 20
        movwf fsr
conB2   movf indf,0       ;tomo los bits mas significativos del ADC
        bcf estado,7
        bcf estado,6
        movwf 65          ;almacenar la parte alta de la conversion
        btfss 65,1
        goto positivo2
        movlw 2d          ;negativo
        movwf 64
        goto acondicB2
positivo2   movlw 2b      ;positivo
        movwf 64
acondicB2   movf 65,0
        andlw 03
        addlw 30
        movwf 65
        bsf estado,7
        bsf estado,6
        incf fsr,1
        movf indf,0       ;tomo los bits menos significativos del ADC
        bcf estado,7
        bcf estado,6
        movwf 66
        movwf 67
        bsf estado,7
        bsf estado,6
        movf fsr,0
        bcf estado,7
        bcf estado,6
        movwf 5a          ;guardo el FSR del banco2
        swapf 66,1
        movlw 0f
        andwf 66,0
        call decodificar
```

```
                movwf 66        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 67,0
                call decodificar
                movwf 67        ;almacenar la parte bajaL de la conversion
                clrf 68         ;generacion del checksum
                clrf 54         ;auxiliar de el llevo
                movlw 60
                movwf fsr
                movf indf,0
                movwf 53        ;auxiliar de la suma
sigasumando2    incf fsr,1
                movlw 68
                xorwf fsr,0
                btfsc estado,2
                goto deccheckSum2
                movf 53,0
                addwf indf,0
                btfsc estado,0
                incf 54,1
                movwf 53
                goto sigasumando2

deccheckSum2    movf 54,0
                addlw 30
                movwf 68
                movf 53,0
                movwf 69
                movwf 6a
                swapf 69,1
                movlw 0f
                andwf 69,0
                call decodificar
                movwf 69        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 6a,0
                call decodificar
                movwf 6a
transmita2      bsf estado,5    ;Inicio de la transmisión del ADC
                bsf 18,5
                bcf estado,5
                movlw 60
                movwf fsr
continue2       movf indf,0
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
segudat2        btfss 18,1
                goto segudat2
                bcf estado,5
                incf fsr,1
                movlw 6b
                xorwf fsr,0
                btfss estado,2
                goto continue2

                movf 5a,0
                bsf estado,7    ;DATOS BANCO 2
```

```
                bsf estado,6
                bcf estado,5
                movwf fsr
                incf fsr,1
                movlw 70
                xorwf fsr,0
                btfss estado,2
                goto conB2

                bsf estado,7    ;DATOS BANCO 3
                bsf estado,6
                bsf estado,5
                movlw 0a0
                movwf fsr
        conB3   movf indf,0     ;tomo los bits mas significativos del ADC
                bcf estado,7
                bcf estado,6
                bcf estado,5
                movwf 65        ;almacenar la parte alta de la conversion
                btfss 65,1
                goto positivo3
                movlw 2d        ;negativo
                movwf 64
                goto acondicB3
positivo3       movlw 2b        ;positivo
                movwf 64
acondicB3       movf 65,0
                andlw 03
                addlw 30
                movwf 65
                bsf estado,7
                bsf estado,6
                bsf estado,5
                incf fsr,1
                movf indf,0     ;tomo los bits menos significativos ADC
                bcf estado,7
                bcf estado,6
                bcf estado,5
                movwf 66
                movwf 67
                bsf estado,7
                bsf estado,6
                bsf estado,5
                movf fsr,0
                bcf estado,7
                bcf estado,6
                bcf estado,5
                movwf 5a        ;guardo el FSR del banco3
                swapf 66,1
                movlw 0f
                andwf 66,0
                call decodificar
                movwf 66        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 67,0
                call decodificar
                movwf 67        ;almacenar la parte bajaL de la conversion
                clrf 68         ;generacion del checksum
                clrf 54         ;auxiliar de el llevo
                movlw 60
```

```
                movwf fsr
                movf indf,0
                movwf 53        ;auxiliar de la suma
sigasumando3    incf fsr,1
                movlw 68
                xorwf fsr,0
                btfsc estado,2
                goto deccheck sum3
                movf 53,0
                addwf indf,0
                btfsc estado,0
                incf 54,1
                movwf 53
                goto sigasumando3

deccheck sum3   movf 54,0
                addlw 30
                movwf 68
                movf 53,0
                movwf 69
                movwf 6a
                swapf 69,1
                movlw 0f
                andwf 69,0
                call decodificar
                movwf 69        ;almacenar la parte bajaH de la conversion
                movlw 0f
                andwf 6a,0
                call decodificar
                movwf 6a
transmita3      bsf estado,5    ;Inicio de la transmisión del ADC
                bsf 18,5
                bcf estado,5
                movlw 60
                movwf fsr
continue3       movf indf,0
                movwf 56
                call bit_paridad
                movf 56,0
                movwf txreg
                bsf estado,5
segudat3        btfss 18,1
                goto segudat3
                bcf estado,5
                incf fsr,1
                movlw 6b
                xorwf fsr,0
                btfss estado,2
                goto continue3

                movf 5a,0
                bsf estado,7    ;DATOS BANCO 3
                bsf estado,6
                bsf estado,5
                movwf fsr
                incf fsr,1
                movlw 0f0
                xorwf fsr,0
                btfss estado,2
                goto conB3
```

```
        bcf estado,7
        bcf estado,6
        bcf estado,5
        movlw 0d
        movwf 56
         call bit_paridad
        movf 56,0
        movwf txreg
        bsf estado,5
seguTX  btfss 18,1
        goto seguTX
        bcf estado,5

        bcf 18,4
        bcf adcon0,0
        bsf estado,5        ;Deshabilito la transmisión
        bcf 18,5
        bcf estado,5
        return

        sleep
        nop
        end
```

PROGRAMA DE LA PANTALLA PARA EL PANEL DE SELECCIÒN

```
function varargout = Comunicacion(varargin)
% COMUNICACION M-file for Comunicacion.fig
%      COMUNICACION, by itself, creates a new COMUNICACION or raises the existing
%      singleton*.
%
%      H = COMUNICACION returns the handle to a new COMUNICACION or the handle to
%      the existing singleton*.
%
%      COMUNICACION('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in COMUNICACION.M with the given input arguments.
%
%      COMUNICACION('Property','Value',...) creates a new COMUNICACION or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Comunicacion_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Comunicacion_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES


% Edit the above text to modify the response to help Comunicacion


% Last Modified by GUIDE v2.5 30-Jul-2005 23:45:11


% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Comunicacion_OpeningFcn, ...
                   'gui_OutputFcn',  @Comunicacion_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end


if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before Comunicacion is made visible.
function Comunicacion_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Comunicacion (see VARARGIN)


posicion = get(0,'DefaultFigurePosition');
unidadv = get(hObject,'Units');
```

```
set(hObject,'Units','pixels');
posicion1 = get(hObject,'Position');
ancho = posicion1(3);
largo = posicion1(4);
if isempty(gcbf)                %gcbf
   unidadwin=get(0,'Units');    %ScreenUnits
   set(0,'Units','pixels');
   unidaddim = get(0,'ScreenSize');    %ScreenSize
   set(0,'Units',unidadwin);
   posicion(1) = 1/2*(unidaddim(3)-ancho);
   posicion(2) = 2/3*(unidaddim(4)-largo);
else
   GCBFOldUnits = get(gcbf,'Units');
   set(gcbf,'Units','pixels');
   GCBFPos = get(gcbf,'Position');
   set(gcbf,'Units',GCBFOldUnits);
   posicion(1:2) = [(GCBFPos(1) + GCBFPos(3)/2) - ancho/2,...
         (GCBFPos(2) + GCBFPos(4)/2 ) - largo/2];
end

posicion(3:4)= [ancho largo];
set(hObject, 'Position',posicion);
set(hObject, 'Units',unidadv);

global canales;
save canales;
global pics;
save pics;
global micro
```

```
save micro
global controla
save controla
controla = 1;
handles.prmicro = controla;

micro = serial('com1');
micro.InputBufferSize = 1521;
micro.BaudRate=9600;
micro.DataBits=8;
micro.Parity='none';
micro.ReadAsyncMode = 'manual';
micro.TimeOut = 10;
micro.Terminator='CR';
fopen(micro);
handles.puerto = micro;

set(handles.mic1,'enable','on');
set(handles.mic2,'enable','on');
set(handles.velocidad,'enable','off');
set(handles.longitud,'enable','off');
set(handles.paridad,'enable','off');
set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
set(handles.canal6,'enable','off')
```

```
% Choose default command line output for Comunicacion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Comunicacion wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = Comunicacion_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
delete(hObject);



% ------------------------------------------------------------------
function salgo_Callback(hObject, eventdata, handles)
% hObject    handle to salgo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double



function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of pushbutton3 as text
```

```
%       str2double(get(hObject,'String')) returns contents of pushbutton3 as a double


% --- Executes during object creation, after setting all properties.
function velocidad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to velocidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in velocidad.
function velocidad_Callback(hObject, eventdata, handles)
% hObject    handle to velocidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns velocidad contents as cell array
%       contents{get(hObject,'Value')} returns selected item from velocidad

val = get(hObject,'Value');
switch val
```

```
case 1


case 2


   controla = handles.prmicro;
   controla;
   micro = handles.puerto;


   if controla == 1
      set(handles.salida,'string','#1WR9616C');
   else
      set(handles.salida,'string','#2WR9616D');
   end


   m = 'qw';


   if controla == 1


      while m ~= 'G'
      fprintf(micro,'#1WR9616C','async');
      readasync(micro);
      micro.BytesAvailable;
      m = fscanf(micro,'%c',16);
      stopasync(micro);
      fclose(micro);
      fopen(micro);
      end
      set(handles.entrada,'string',m);
```

```
    else

        while m ~= 'G'
        fprintf(micro,'#2WR9616D','async');
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end
        set(handles.entrada,'string',m);
    end
    a = [0.778 0 0]
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.mic1,'enable','off');
set(handles.mic2,'enable','off');
set(handles.velocidad,'enable','off');
set(handles.longitud,'enable','on');
micro.BaudRate=9600;
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);
set(handles.entrada,'string','');
set(handles.salida,'string','');



case 3
```

```
controla = handles.prmicro;
controla;
micro = handles.puerto;

if controla == 1
   set(handles.salida,'string','#1WR48169');
else
   set(handles.salida,'string','#2WR4816A');
end

m = 'qw';

if controla == 1

   while m ~= 'G'
   fprintf(micro,'#1WR48169','async');
   readasync(micro);
   micro.BytesAvailable;
   m = fscanf(micro,'%c',16);
   stopasync(micro);
   fclose(micro);
   fopen(micro);
   end

   set(handles.entrada,'string',m);

else

   while m ~= 'G'
```

```
        fprintf(micro,'#2WR4816A','async');
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.mic1,'enable','off');
set(handles.mic2,'enable','off');
set(handles.velocidad,'enable','off');
set(handles.longitud,'enable','on');
micro.BaudRate=4800;
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);

case 4

    controla = handles.prmicro;
    micro = handles.puerto;
```

```
if controla == 1
   set(handles.salida,'string','#1WR24163');
else
   set(handles.salida,'string','#2WR24164');
end

m = 'qw';

if controla == 1

   while m ~= 'G'
   fprintf(micro,'#1WR24163','async');
   readasync(micro);
   micro.BytesAvailable;
   m = fscanf(micro,'%c',16);
   stopasync(micro);
   fclose(micro);
   fopen(micro);

   end

   set(handles.entrada,'string',m);

else

   while m ~= 'G'
   fprintf(micro,'#2WR24164','async');
   readasync(micro);
   micro.BytesAvailable;
```

```
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.mic1,'enable','off');
set(handles.mic2,'enable','off');
set(handles.velocidad,'enable','off');
set(handles.longitud,'enable','on');
micro.BaudRate=2400;
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);

case 5

    controla = handles.prmicro;
    micro = handles.puerto;

    if controla == 1
        set(handles.salida,'string','#1WR12160');
    else
```

```
    set(handles.salida,'string','#2WR12161');
end

m = 'qw';

if controla == 1

    while m ~= 'G'
    fprintf(micro,'#1WR12160','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);

else

    while m ~= 'G'
    fprintf(micro,'#2WR12161','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
```

```
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.mic1,'enable','off');
  set(handles.mic2,'enable','off');
  set(handles.velocidad,'enable','off');
  set(handles.longitud,'enable','on');
  micro.BaudRate=2400;
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);


end

% --- Executes during object creation, after setting all properties.
function longitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in longitud.
function longitud_Callback(hObject, eventdata, handles)
% hObject    handle to longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns longitud contents as cell array
%        contents{get(hObject,'Value')} returns selected item from longitud

val = get(hObject,'Value');
switch val
    case 1


    case 2


        controla = handles.prmicro;
        controla;
        micro = handles.puerto;


        if controla == 1
            set(handles.salida,'string','#1WR8135');
        else
```

```
    set(handles.salida,'string','#2WR8136');
end

m = 'qw';

if controla == 1

    while m ~= 'G'
    fprintf(micro,'#1WR8135','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);

else

    while m ~= 'G'
    fprintf(micro,'#2WR8136','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
```

```
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.paridad,'enable','off');
set(handles.longitud,'enable','off');
set(handles.canal1,'enable','on');
set(handles.canal2,'enable','on');
set(handles.canal3,'enable','on');
set(handles.canal4,'enable','on');
set(handles.canal5,'enable','on');
set(handles.canal6,'enable','on');
micro.DataBits=8;
micro.Parity='none';
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);

case 3

    controla = handles.prmicro;
    controla;
    micro = handles.puerto;
```

```
if controla == 1
   set(handles.salida,'string','#1WR9136');
else
   set(handles.salida,'string','#2WR9137');
end

m = 'qw';

if controla == 1

   while m ~= 'G'
   fprintf(micro,'#1WR9136','async');
   readasync(micro);
   micro.BytesAvailable;
   m = fscanf(micro,'%c',16);
   stopasync(micro);
   fclose(micro);
   fopen(micro);

   end

   set(handles.entrada,'string',m);

else

   while m ~= 'G'
   fprintf(micro,'#2WR9137','async');
   readasync(micro);
   micro.BytesAvailable;
```

```
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.paridad,'enable','on');
  set(handles.longitud,'enable','off');
  micro.DataBits=8;
  micro.Parity='none';
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);


end

% --- Executes during object creation, after setting all properties.
function paridad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to paridad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in paridad.
function paridad_Callback(hObject, eventdata, handles)
% hObject    handle to paridad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns paridad contents as cell array
%        contents{get(hObject,'Value')} returns selected item from paridad

val = get(hObject,'Value');
switch val
   case 1


   case 2


      controla = handles.prmicro;
      controla;
      micro = handles.puerto;
```

```
if controla == 1
   set(handles.salida,'string','#1WR112E');
else
   set(handles.salida,'string','#2WR112F');
end

m = 'qw';

if controla == 1

   while m ~= 'G'
   fprintf(micro,'#1WR112E','async');
   readasync(micro);
   micro.BytesAvailable;
   m = fscanf(micro,'%c',16);
   stopasync(micro);
   fclose(micro);
   fopen(micro);

   end

   set(handles.entrada,'string',m);

else

   while m ~= 'G'
   fprintf(micro,'#2WR1129','async');
   readasync(micro);
   micro.BytesAvailable;
```

```
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);
    set(handles.entrada,'string',m);
    pause (1)
    end

    set(handles.entrada,'string',m);
  end


  a = [0.778 0 0];
  set(handles.activa,'backgroundColor',a);
  pause(1)

set(handles.paridad,'enable','off');
set(handles.canal1,'enable','on')
set(handles.canal2,'enable','on')
set(handles.canal3,'enable','on')
set(handles.canal4,'enable','on')
set(handles.canal5,'enable','on')
set(handles.canal6,'enable','on')
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);

case 3

  controla = handles.prmicro;
```

```
controla;
micro = handles.puerto;

if controla == 1
   set(handles.salida,'string','#1WR212F');
else
   set(handles.salida,'string','#2WR2130');
end

m = 'qw';

if controla == 1

   while m ~= 'G'
   fprintf(micro,'#1WR212F','async');
   readasync(micro);
   micro.BytesAvailable;
   m = fscanf(micro,'%c',16);
   stopasync(micro);
   fclose(micro);
   fopen(micro);

   end

   set(handles.entrada,'string',m);

else

   while m ~= 'G'
```

```
        fprintf(micro,'#2WR2130','async');
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end


    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.paridad,'enable','off');
  set(handles.canal1,'enable','on')
  set(handles.canal2,'enable','on')
  set(handles.canal3,'enable','on')
  set(handles.canal4,'enable','on')
  set(handles.canal5,'enable','on')
  set(handles.canal6,'enable','on')
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);


end
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu6 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu6
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in canal1.
function canal1_Callback(hObject, eventdata, handles)
% hObject    handle to canal1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of canal1

    controla = handles.prmicro;
    controla;
    micro = handles.puerto;

    if controla == 1
       set(handles.salida,'string','#1WR012D');
    else
       set(handles.salida,'string','#2WR012E');
    end

    m = 'qw';

    if controla == 1
```

```
    while m ~= 'G'
    fprintf(micro,'#1WR012D','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);

else

    while m ~= 'G'
    fprintf(micro,'#2WR012E','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);
end
```

```
    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.canal1,'enable','off')
  set(handles.canal2,'enable','off')
  set(handles.canal3,'enable','off')
  set(handles.canal4,'enable','off')
  set(handles.canal5,'enable','off')
  set(handles.canal6,'enable','off')
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);
  pause(1)
  global micro;
  save micro;
  micro = handles.puerto;
  global controla;
  save controla;
  global canales;
  canales = 'Canal 1';
  save canales;
  datos



% --- Executes on button press in canal2.
function canal2_Callback(hObject, eventdata, handles)
% hObject    handle to canal2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of canal2


    controla = handles.prmicro;
    controla;
    micro = handles.puerto;

    if controla == 1
       set(handles.salida,'string','#1WR112E');
    else
       set(handles.salida,'string','#2WR112F');
    end

    m = 'qw';

    if controla == 1

       while m ~= 'G'
       fprintf(micro,'#1WR112E','async');
       readasync(micro);
       micro.BytesAvailable;
       m = fscanf(micro,'%c',16);
       stopasync(micro);
       fclose(micro);
       fopen(micro);

       end
```

```
    set(handles.entrada,'string',m);

  else

    while m ~= 'G'
    fprintf(micro,'#2WR112F','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);
    end

    set(handles.entrada,'string',m);
  end

  a = [0.778 0 0];
  set(handles.activa,'backgroundColor',a);
  pause(1)

set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
set(handles.canal6,'enable','off')
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);
```

```
   pause(1)
   global micro;
   save micro;
   micro = handles.puerto;
   global controla;
   save controla;
   datos
```

```
% --- Executes on button press in canal3.
function canal3_Callback(hObject, eventdata, handles)
% hObject    handle to canal3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of canal3


      controla = handles.prmicro;
      controla;
      micro = handles.puerto;

      if controla == 1
         set(handles.salida,'string','#1WR4131');
      else
         set(handles.salida,'string','#2WR4132');
      end
```

```
m = 'qw';

if controla == 1

    while m ~= 'G'
    fprintf(micro,'#1WR4131','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);

else

    while m ~= 'G'
    fprintf(micro,'#2WR4132','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);
    end
```

```
        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
set(handles.canal6,'enable','off')
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);
pause(1)
global micro;
save micro;
micro = handles.puerto;
global controla;
save controla;
datos



% --- Executes on button press in canal4.
function canal4_Callback(hObject, eventdata, handles)
% hObject    handle to canal4 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of canal4


    controla = handles.prmicro;
    controla;
    micro = handles.puerto;

    if controla == 1
       set(handles.salida,'string','#1WR5132');
    else
       set(handles.salida,'string','#2WR5133');
    end

    m = 'qw';

    if controla == 1

       while m ~= 'G'
       fprintf(micro,'#1WR5132','async');
       readasync(micro);
       micro.BytesAvailable;
       m = fscanf(micro,'%c',16);
       stopasync(micro);
       fclose(micro);
       fopen(micro);
```

```
        end

        set(handles.entrada,'string',m);

    else

        while m ~= 'G'
        fprintf(micro,'#2WR5133','async');
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
set(handles.canal6,'enable','off')
```

```
    a = [0.99 0.99 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)
    global micro;
    save micro;
    micro = handles.puerto;
    global controla;
    save controla;
    datos




% --- Executes on button press in canal5.
function canal5_Callback(hObject, eventdata, handles)
% hObject    handle to canal5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of canal5



    controla = handles.prmicro;
    controla;
    micro = handles.puerto;

    if controla == 1
       set(handles.salida,'string','#1WR6133');
    else
       set(handles.salida,'string','#2WR6134');
```

```
end

m = 'qw';

if controla == 1

    while m ~= 'G'
    fprintf(micro,'#1WR6133','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);

    end

    set(handles.entrada,'string',m);

else

    while m ~= 'G'
    fprintf(micro,'#2WR6134','async');
    readasync(micro);
    micro.BytesAvailable;
    m = fscanf(micro,'%c',16);
    stopasync(micro);
    fclose(micro);
    fopen(micro);
```

```
        end


        set(handles.entrada,'string',m);
    end


    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)


set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
set(handles.canal6,'enable','off')
a = [0.99 0.99 0];
set(handles.activa,'backgroundColor',a);
pause(1)
global micro;
save micro;
micro = handles.puerto;
global controla;
save controla;
datos




% --- Executes on button press in canal6.
function canal6_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to canal6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of canal6


    controla = handles.prmicro;
    controla;
    micro = handles.puerto;

    if controla == 1
       set(handles.salida,'string','#1WR7134');
    else
       set(handles.salida,'string','#2WR7135');
    end

    m = 'qw';

    if controla == 1

       while m ~= 'G'
       fprintf(micro,'#1WR7134','async');
       readasync(micro);
       micro.BytesAvailable;
       m = fscanf(micro,'%c',16);
       stopasync(micro);
       fclose(micro);
       fopen(micro);
```

```
        end

        set(handles.entrada,'string',m);

    else

        while m ~= 'G'
        fprintf(micro,'#2WR7135','async');
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);
        end

        set(handles.entrada,'string',m);
    end

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

set(handles.canal1,'enable','off')
set(handles.canal2,'enable','off')
set(handles.canal3,'enable','off')
set(handles.canal4,'enable','off')
set(handles.canal5,'enable','off')
```

```
   set(handles.canal6,'enable','off')
   a = [0.99 0.99 0];
   set(handles.activa,'backgroundColor',a);
   pause(1)
   global micro;
   save micro;
   micro = handles.puerto;
   global controla;
   save controla;
   datos



% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in mic1.
function mic1_Callback(hObject, eventdata, handles)
% hObject    handle to mic1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
   global pics;
   pics = 'MICRO A';
   save pics;
   a = [0.778 0 0];
   set(handles.color1,'backgroundColor',a);
   pause(0.01)
   global controla;
   controla = 1;
   save controla;
   handles.prmicro = controla;
   guidata(hObject, handles);
   controla = handles.prmicro;
   micro = handles.puerto;

   set(handles.salida,'string','#1WRFD');
   m = 'qw';

      while m ~= 'G'
         fprintf(micro,'#1WRFD','async');
```

```
        readasync(micro);
        micro.BytesAvailable;
        m = fscanf(micro,'%c',16);
        stopasync(micro);
        fclose(micro);
        fopen(micro);

    end
    set(handles.entrada,'string',m);



    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.mic1,'enable','off');
  set(handles.mic2,'enable','off');
  set(handles.velocidad,'enable','on');
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);



% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in mic2.
function mic2_Callback(hObject, eventdata, handles)
% hObject    handle to mic2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


   global pics;
   pics = 'MICRO B';
   save pics;
   a = [0.778 0 0];
   set(handles.color2,'backgroundColor',a);


   pause(0.01)
   global controla;
   controla = 0;
   save controla;
   handles.prmicro = controla;
   guidata(hObject, handles);
   controla = handles.prmicro;
   micro = handles.puerto;


   set(handles.salida,'string','#2WRFE');
   m = 'qw';


      while m ~= 'G'
         fprintf(micro,'#2WRFE','async');
         readasync(micro);
         micro.BytesAvailable;
         m = fscanf(micro,'%c',16);
```

```
        stopasync(micro);
        fclose(micro);
        fopen(micro);

    end
    set(handles.entrada,'string',m);

    a = [0.778 0 0];
    set(handles.activa,'backgroundColor',a);
    pause(1)

  set(handles.mic1,'enable','off');
  set(handles.mic2,'enable','off');
  set(handles.velocidad,'enable','on');
  a = [0.99 0.99 0];
  set(handles.activa,'backgroundColor',a);
  set(handles.entrada,'string','');
  set(handles.salida,'string','');



% --- Executes on button press in color1.
function color1_Callback(hObject, eventdata, handles)
% hObject    handle to color1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in Color2.
function Color2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to Color2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in color2.
function color2_Callback(hObject, eventdata, handles)
% hObject    handle to color2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)
% hObject    handle to reset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global micro
save micro

fclose(micro);
clear micro;

function enlace_Callback(hObject, eventdata, handles)

   a = [0.778 0 0];
   set(handles.activa,'BackgroundColor',a);
   pause on
```

Latacunga, agosto del 2005

Ing. Amparo Meythaler

CI. 050151580-3

Ing. Nancy Guerrón

DIRECTORA DE CARRERA

Ab. Eduardo Vásquez

SECRETARIO ACADÉMICO