UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO Y CONSTRUCCION DE UN SISTEMA AUTOMATICO PARA CRONOMETRAR LOS TIEMPOS DE CORREDORES EN COMPETENCIAS DE MARATON

T E S I S

QUE PARA OBTENER EL TITULO DE:

I N G E N I E R O M E C A N I C O

E L E C T R I C I S T A

P R E S E N T A :

ENRIQUE SANCHEZ UGALDE

DIRECTOR DE TESIS: ING. JOSE FRANCISCO ESCAMILLA GUZMAN

COODIRECTOR DE TESIS: ING. JOSE ANTONIO DE JESUS ARREDONDO GARZA

MEXICO, D.F. 2008

A mi padre Enrique Sánchez Torres †

Con el más profundo agradecimiento de tu ejemplo y esfuerzo para mi formación.

Papá:

Siempre te recordaré como el padre cariñoso y dedicado que fuiste, una gran tristeza no compartir contigo éste momento, te quiero y te extraño tanto.....

A mi madre Maria Guadalupe Ugalde de Sánchez

Mamá:

Tu amor, cariño, cuidados y consejos fueron fundamentales en mi vida, tu apoyo y respaldo invaluables. Mil gracias por todo lo que has hecho por mi. Estoy seguro que éste momento cumple con una de tus grandes ilusiones.

A mi hermano Ricardo Sánchez Ugalde

Por esos años maravillosos de la infancia y
la juventud, siempre los tendré presentes.

A mi hermana Diana Gabriela Sánchez Ugalde

Por que siempre has sido una confidente que sabe escuchar y apoyar incondicionalmente. Que las ilusiones que tengas se te cumplan por que te mereces lo mejor.

A mis sobrinos Xena, Ricardo, Emilio y Jesús

Por el profundo cariño que les tengo y la esperanza de que triunfen en la vida.

A mi amigo y director de tesis Ing. José Francisco Escamilla Guzmán

Gracias por todo tu apoyo, conocimiento y esfuerzo, sin duda
fueron fundamentales para concluir mis estudios profesionales

Al Ing. José Antonio de Jesús Arredondo Garza

Por su colaboración y siempre buena disposición
como codirector de mi trabajo para titulación

A mis profesores y sinodales

Por su esfuerzo y dedicación en transmitir sus conocimientos para formar profesionales y en la revisión de éste trabajo de tesis cuyos comentarios fueron relevantes.

A la UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO y sobre todo a la FACULTAD DE INGENIERÍA.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

CAPÍTULO I

INTRODUCCIÓN

El objetivo de este capítulo es el de presentar las razones y los fundamentos que se consideraron para el desarrollo del presente trabajo de tesis

I.1. PROBLEMÁTICA

En la actualidad en nuestro país y a nivel mundial hay una gran conciencia en la población para cuidar la salud por medio de la actividad física. El trote y la carrera es un deporte de preferencia para la gente ya que sólo es necesario para su práctica una pista o parque deportivo, así como el uso de ropa adecuada.

Una condición natural de las personas que practican deporte, es incrementar su rendimiento, lo que los lleva a muchos de ellos a realizar un entrenamiento formal en miras de competencias donde se muestren sus capacidades. En el caso del atletismo la prueba de mayor exigencia es el maratón.

El maratón tiene un poder de convocatoria muy alto, cientos de atletas se inscriben a una carrera lo cual a provocado el interés de empresas para organizar este tipo de competencias por el negocio que representan los patrocinios de compañías fabricante de ropa y artículos deportivos, bebidas energizantes, etcétera; así como las cuotas que a los mismos competidores se les solicita para su inscripción.

El registro, así como la obtención de resultados confiables en tiempo real clasificados por categoría (varonil, femenil, profesionales, amateurs, master, con discapacidad) es información de suma relevancia para el comité organizador de la carrera ya que de ello dependerá la correcta premiación de los ganadores. Cabe señalar que para la mayoría de los atletas su principal rival en la competencia son ellos mismos buscando siempre mejorar sus marcas, por lo que los tiempos de sus registros son de igual importancia que para los que luchan por los primeros lugares. Tradicionalmente, jueces instalados en la meta obtienen los tiempos de los corredores apoyados con un cronómetro, cuando los punteros comienzan a llegar este método es viable, pero, en el momento que se presenta un contingente de competidores es imposible para los jueces tomar todos los tiempos.

En el mercado, en particular en México, existen un par de compañías que cuentan con equipos para cronometrar los tiempo de los corredores por medio de la tecnología de radiofrecuencia, instalando un *transponder* en el zapato tenis del competidor, mismo que transmite su información a una unidad de lectura al inicio y al termino de la carrera, pudiendo con estos datos obtener los tiempos del recorrido. En la investigación de este trabajo de tesis, se encontró que técnicamente el sistema de cronometraje citado no es confiable ya que una gran cantidad de corredores no es registrado por el sistema con la contrayente molestia por parte de los competidores, además, las empresas dueñas del equipo ofrecen el servicio de cronometraje en renta con tarifas muy altas, haciendo prohibitivo para la mayoría de organizadores de competencias el que puedan ofrecer el servicio de cronometraje automático.

La problemática planteada originó la necesidad de encontrar una solución que ofreciera al mercado un equipo de cronometraje confiable y pudiera ser accesible para cualquier organizador de carreras.

I.2. ANTECEDENTES

Desde principio de los años 90 se realizaron esfuerzos, apoyados en la tecnología disponible, para proponer una solución en el cronometraje de maratones que ofreciera eficiencia y confiabilidad en la toma de resultados.

Los primeros intentos se realizaron utilizando computadoras de escritorio instaladas en la meta, en las cuales se ejecutaba un programa que era capaz de clasificar por categoría y asociar un tiempo al momento que se capturaban por medio del teclado los números de los competidores que concluían el recorrido.

Posteriormente, se intentó utilizar la tecnología de código de barras, incluyendo en el número del corredor que colocan en su dorso una etiqueta con las barras que los identificaba, teniendo que presentarse al concluir la competencia ante el lector para la toma de su tiempo.

También se llegaron a utilizar lectoras portátiles, con las cuales los jueces instalados en distintos puntos de la meta, podían recabar la información de los atletas que terminaban la carrera, almacenando la información en la memoria de las lectoras para su posterior transferencia a un sistema central.

Como es evidente, las distintas tecnologías utilizadas ofrecían cierto grado de automatización; sin embargo, ninguna de ellas ofrecía confiabilidad, dado que dependían totalmente de que los jueces pudieran tomar o registrar a todos y cada uno de los corredores, adicionalmente, por la cantidad de competidores aun y cuando fueran registrados, los tiempos no eran exactos, dado que transcurría un tiempo entre que el juez daba a conocer el número de competidor al personal de captura en la computadora o bien, si las lectoras fijas o portátiles se encontraban saturadas al momento de la llegada de un contingente de atletas que los obligaba a esperar su turno para el registro de datos.

En años más recientes y como se mencionó anteriormente, existen equipos de cronometraje automático basados en la tecnología de la radiofrecuencia., sin embargo, técnicamente no ofrecen aun la confiabilidad en la totalidad de los registros y los costos para la contratación del servicio son muy altos.

I.3. ANÁLISIS DEL PROBLEMA

Para poder ofrecer una alternativa tecnológica que supere las dificultades que se han señalado, se establecerán los requisitos técnicos que el sistema de este trabajo de tesis deberá satisfacer:

- Identificación de cada competidor.- Que por medio de la tecnología seleccionada se obtengan de manera automática los datos de los participantes, como son: nombre, edad, No. de participante, categoría, etcétera.

- Toma de resultados.- Altamente eficiente al paso de una gran cantidad de deportistas, registrando automáticamente la hora en que inicio el evento, así como la hora en que cada deportista concluyó la competencia, generando reportes inmediatos con los resultados clasificados por categoría.

- El equipo deberá considerar características que permitan su transportación a los sitios donde se efectúen las competencias, así como autonomía eléctrica para el caso de no contar con alimentación en 127 VCA

- Conforme a los reglamentos de la Federación de Atletismo el sistema deberá cumplir con lo siguiente:

    - No se permite ningún tipo de interferencias o acciones que afecten el desarrollo natural de la competencia.
    - Los resultados deberán obtenerse hasta décimas de segundo

- Que el análisis de costos, permita una recuperación de la inversión en un tiempo razonable, con tarifas de renta del equipo accesibles a cualquier organizador

I.4. PROPUESTA DE SOLUCIÓN

Del análisis efectuado a los requisitos técnicos, el establecido por la Federación de Atletismo en lo referente a que no se permite ningún tipo de interferencias o acciones que afecten el desarrollo natural de la competencia, mandó para determinar la tecnología a utilizar, la base de estudio se presenta a continuación (tabla 1.4.1.):

	REQUISITOS TÉCNICOS					
TECNOLOGÍA	**Interferencia o acción que afecte el desarrollo de la competencia**	**Identificación del competidor**	**Toma de resultados**	**Autonomía eléctrica**	**Registro hasta décimas de segundo**	**Conclusión**
Código de barras	Para la lectura del código de barras, el competidor deberá estar a una distancia muy cercana de la unidad lectora para su registro y prácticamente detenido. El corredor deberá disminuir su velocidad para tal fin.	El código de barras puede identificar al competidor	El código de barras lo permite con apoyo de un software	Es posible la alimentación eléctrica de las lectoras es con baterías	Es posible el software deberá estar programado para tal fin	La tecnología satisface prácticamente todos los requisitos, sin embargo, para el registro del competidor implica una acción que afecta el desarrollo natural de la competencia al tener que disminuir su velocidad. **NO CUMPLE CON LO SOLICITADO**
Censores de presencia	El censor de presencia no implica interferencia o acción para el desarrollo de la competencia	No lo permite, sólo indica la presencia o no presencia del competidor	No es capaz de identificar puntualmente a un competidor, por lo que no se le puede asociar un tiempo	Es posible la alimentación eléctrica podría ser una fuente de CD con una batería	No es capaz de identificar puntualmente a un competidor, por lo que no se le puede asociar un tiempo	El censor de presencia **NO CUMPLE CON LO SOLICITADO**

Tabla 1.4.1. Comparativo de tecnologías contra requisitos técnicos (continua).

	REQUISITOS TÉCNICOS					
TECNOLOGÍA	**Interferencia o acción que afecte el desarrollo de la competencia**	**Identificación del competidor**	**Toma de resultados**	**Autonomía eléctrica**	**Registro hasta décimas de segundo**	**Conclusión**
Identificación por radio frecuencia (*RFID*)	La identificación por radio frecuencia (*RFID*) permite que la información sea escrita, almacenada y leída en un *transponder* por medio de una conexión inalámbrica, haciendo posible la recolección automática sin contacto alguno. Lo anterior cumpliría cabalmente con lo solicitado	Precisamente la *RFID* permite la identificación puntual de quien porta el *transponder*	Es posible, la tecnología del *RFID* proporciona la identificación de los atletas, a los cuales se le puede asociar un tiempo para obtener los resultados	Es posible la alimentación eléctrica podría ser una fuente de CD con una batería	Es posible con el dispositivo que se utilice para asociar el tiempo a la identificación del competidor	La tecnología de la **Identificación por radio frecuencia (RFID), CUMPLE CON LOS REQUISITOS SOLICITADOS**

Tabla 1.4.1. Comparativo de tecnologías contra requisitos técnicos.

Del cuadro anterior, se concluyó que la tecnología a utilizar para este trabajo de tesis es la Identificación por radio frecuencia (*RFID*)

De manera general, la *RFID* funciona por medio de comunicación inalámbrica entre un *transponder*, una antena y una unidad lectora, tal y como se muestra en la figura 1.4.1.

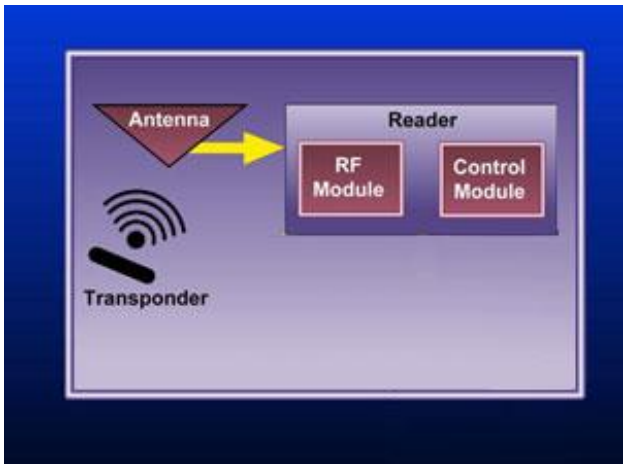


Figura 1.4.1. Esquema general tecnología de identificación por radio frecuencia (*RFID*).

Entonces, con la tecnología *RFID* contamos con la identificación del portador del *transpoder*, sin embargo, para cumplir con todos y cada uno de los requisitos técnicos establecidos, no es suficiente, será necesario desarrollar etapas con un dispositivo de control que permita asociar en tiempo real los tiempos de inicio y terminación de la competencia, una eficiencia y confiabilidad de lecturas ante el paso de un contingente de atletas, así como el resguardo de resultados.

El dispositivo de control que se utilizará en este trabajo de tesis, será un microcontrolador seleccionado con base en los requerimientos de velocidad de procesamiento, memoria y comunicación

que exijan los dispositivos electrónicos que se utilicen para satisfacer los requisitos ya mencionados, así como el sistema de *RFID* que sea seleccionado de entre las empresas que cuentan con tal desarrollo entre sus productos comerciales.

El sistema de resultados deportivos **SDR-1000** es el nombre que recibirá el equipo de cronometraje automático que se desarrollará en este trabajo de tesis.

En los capítulos subsecuentes, se presenta el análisis detallado de los fundamentos teóricos, selección y características del sistema de *RFID*, selección del microcontrolador y en general el diseño del sistema de cronometraje deportivo.

CAPÍTULO II

FUNDAMENTOS TEÓRICOS

El objetivo de este capítulo es el de presentar los conceptos básicos de los elementos que conforman un sistema de Identificación por Radio Frecuencia (*RFID*), tales como amplificadores, filtros, antenas, así como los principales tipos de modulación de señales que existen. También, en el presente capítulo se describirá la organización de un sistema de cómputo, así como los métodos de transmisión y recepción más comunes y los tipos de protocolos necesarios para establecer una comunicación.

II.1. MODULACIÓN

La modulación es el proceso mediante el cual se altera una onda portadora variando uno de los parámetros de la señal: amplitud, frecuencia o fase, de acuerdo a la señal de entrada (señal moduladora); mientras que la demodulación es el proceso inverso, es decir, se obtiene la señal original a partir de la señal modulada.

Muchos tipos de señales no pueden ser enviadas directamente por un canal de transmisión debido a la insuficiente inmunidad al ruido eléctrico, limitaciones de ancho de banda y, principalmente, debido a que las frecuencias de las señales de interés por lo general se encuentran dentro de un intervalo de frecuencias muy bajas, por lo que sus longitudes de onda son bastante largas y en consecuencia la antena transmisora que se tendría que utilizar para convertir las señales en radiación electromagnética sería muy grande. Por estas razones, es necesario utilizar una forma de onda "portadora" con longitud de onda más corta y cuyas propiedades se adapten mejor al medio de transmisión que se esté empleando. Debido a esto, la antena transmisora que se utilice tendrá una longitud moderada.

En general, las señales se clasifican en dos tipos, aquellas que puedan tomar cualquier valor de amplitud dentro de un rango y que son llamadas analógicas y las que sólo pueden tener un número finito de valores de amplitud llamadas digitales. Debido a esto, es posible identificar dos técnicas de modulación, las cuales se discuten a continuación.

II.1.1. MODULACIÓN ANALÓGICA

En la modulación analógica la señal de información es analógica y la onda portadora la constituye una señal senoidal de mayor frecuencia; la cual dependiendo del parámetro que le sea variado, ya sea la amplitud, fase o frecuencia, será el tipo de modulación empleada. Los tipos de modulación para este caso son:

- Amplitud Modulada (AM)
- Frecuencia Modulada (FM)
- Modulación de Fase (MF)

AMPLITUD MODULADA (AM). En la modulación analógica la frecuencia y la fase de la onda portadora se mantiene constante; no así su amplitud, la cual varía de acuerdo a la señal de entrada (moduladora).

La modulación en amplitud es muy sensible al ruido, por lo que su empleo sólo resulta conveniente en aquellos sistemas en los que no afecta que la señal recuperada venga mezclada con ruido.

En la figura 2.1.1 se muestra un ejemplo de la modulación AM.

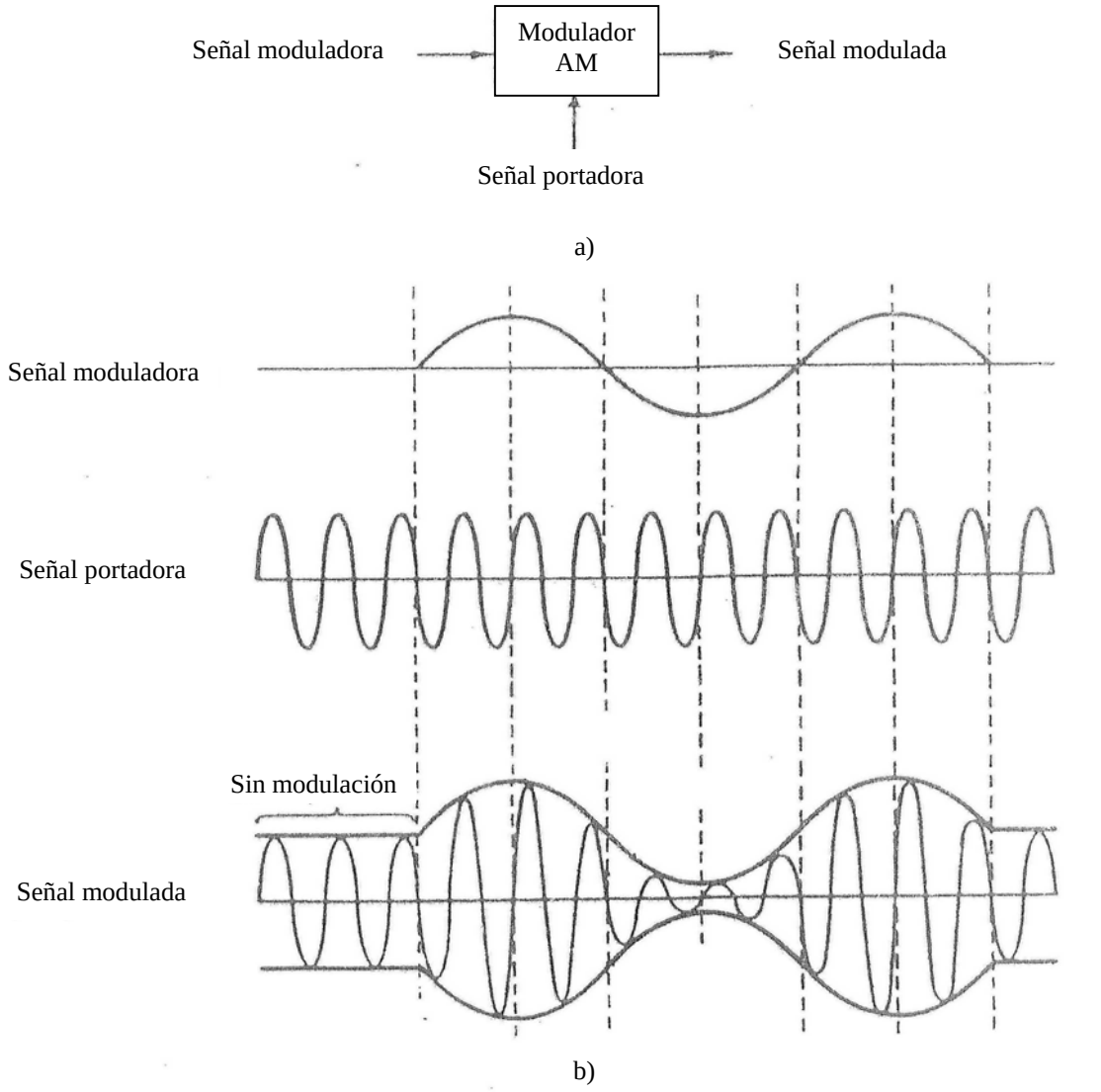


Fig. 2.1.1. Modulación en AM: a) Modulador AM, b) proceso de modulación en AM.

FRECUENCIA MODULADA (FM). En este caso, la amplitud y la fase de la onda portadora se mantiene constante; no así su frecuencia, la cual varía de acuerdo a la señal de entrada.

MODULACIÓN DE FASE (MF). Este proceso, a diferencia de los anteriores, ocasiona que la fase de la onda portadora varíe de acuerdo a la señal de entrada. De tal forma que cada cambio en la fase represente la información deseada. En tales sistemas, el cambio de fase es referenciado a la fase de la señal actual y no a una posición predeterminada.

En la figura 2.1.2 se muestra un ejemplo de la modulación FM Y MF.

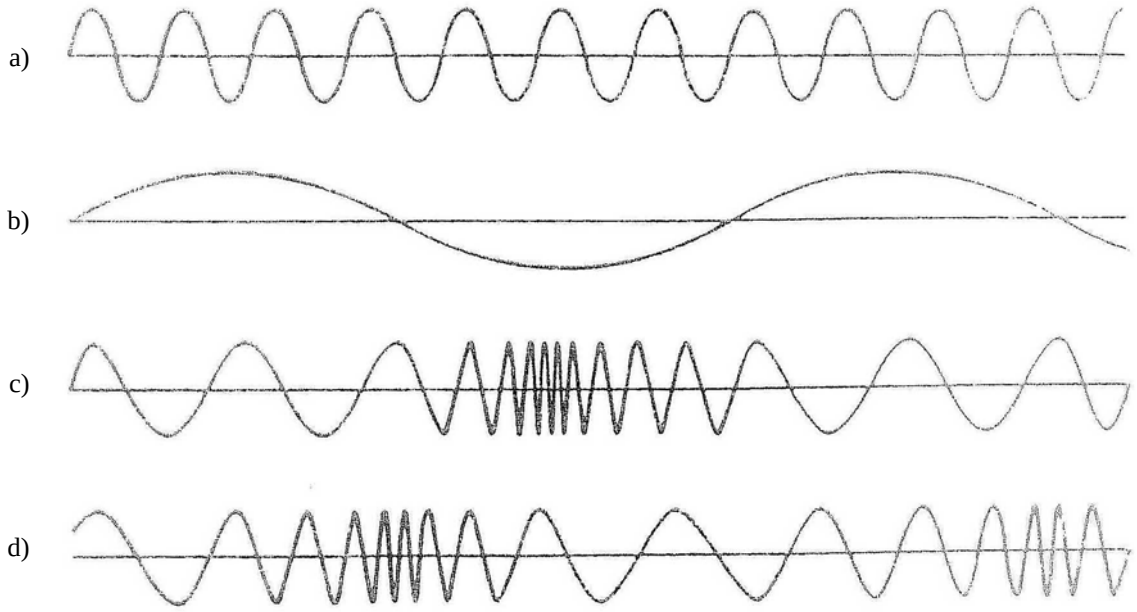


Fig. 2.1.2. Modulación en FM y MF: a) señal portadora, b) señal moduladora, c) señal modulada en FM, d) señal modulada en MF.

II.1.2. MODULACIÓN DE SEÑALES DIGITALES.

En la modulación digital, la señal de entrada es una señal digital y la onda portadora es una onda senoidal de mayor frecuencia, la cual, dependiendo del parámetro variado, ya sea amplitud, fase o frecuencia, será el tipo de modulación empleado:

- Modulación por cambio de Amplitud ASK *(Amplitude Shift Keying)*
- Modulación por cambio de frecuencia FSK *(Frequency Shift Keying)*
- Modulación por cambio de fase PSK *(Phase Shift Keying)*

MODULACIÓN POR CAMBIO DE AMPLITUD *(ASK)*. En este tipo de modulación la amplitud de la señal portadora se alterna entre dos o más valores, por lo general en 0 lógico y 1 lógico, correspondiente a las señales binarias. En la figura 2.1.3b se muestra un ejemplo de la modulación *ASK*, en el cual la onda portadora es modulada por cambio de amplitud de acuerdo al mensaje binario 101101.

MODULACIÓN POR CAMBIO DE FASE *(PSK)*. En la modulación por cambio de fase la modulación ocurre cuando la fase de la onda portadora varía tantas veces como lo hace la señal de entrada (señal moduladora), en donde cada cambio en la fase representa un digito binario. En la figura 2.1.3c se muestra un ejemplo de la modulación *PSK*, en el cual la onda portadora es modulada por cambio de fase de acuerdo al mensaje binario 101101.

MODULACIÓN POR CAMBIO DE FRECUENCIA *(FSK)*. La modulación *FSK* es parecida a la modulación FM, excepto que, como ya se mencionó, la señal moduladora es un tren de pulsos binario que varía entre dos niveles de voltaje en comparación a la forma de onda continua y variante en el tiempo de la señal moduladora para el caso de la modulación en FM. En la figura 2.1.3.d se muestra un ejemplo de modulación *FSK,* en el cual la onda portadora es modulada por cambio de frecuencia de acuerdo al mensaje binario 101101.

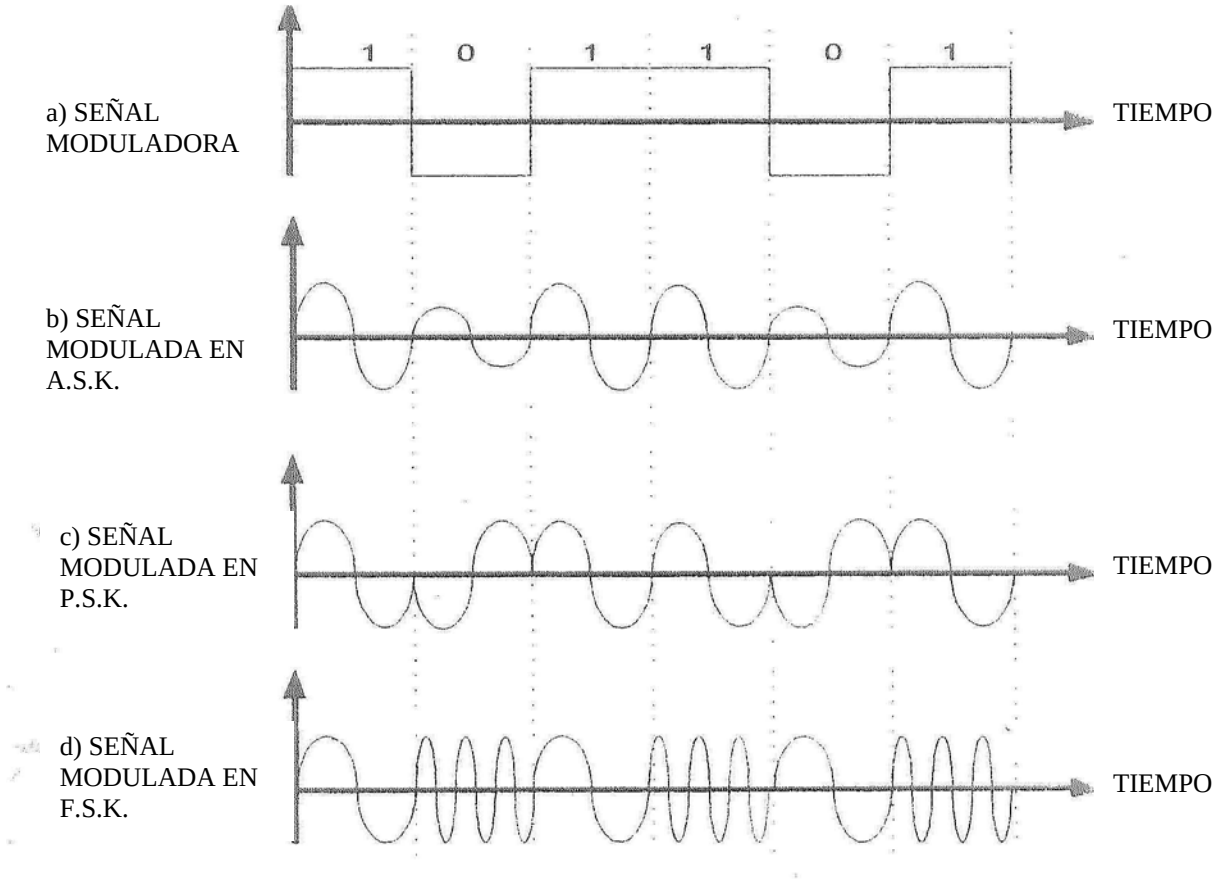


Fig. 2.1.3. Modulación digital a) señal moduladora, b) modulación ASK, c) modulación PSK, d) modulación FSK.

II.2. AMPLIFICADORES

La necesidad de amplificar señales se presenta en virtud de que los transductores proporcionan señales que se les considera débiles o pequeñas, es decir, del orden de microvolts (uV) o milivolts (mV) y con poca energía.

Una red de dos puertos operando como amplificador puede dar a la carga una potencia mayor que la que se obtiene de la fuente de señal. Los amplificadores necesitan fuentes de voltaje de dc (Vdc) para su operación, estas Vdc proporcionan la potencia extra que se entrega a la carga, así como cualquier potencia que pueda disiparse en la red misma de dos puertos.

Cuando una señal se amplifica, debe tenerse cuidado para que la información contenida en la señal no cambie, ni se introduzca nueva información. Así, cuando se alimenta una señal a un

amplificador, es deseable que la señal de salida del amplificador sea una réplica exacta de la entrada, excepto, que tendrá una mayor amplitud. Es decir, las variaciones en la forma de la onda de salida deben ser idénticas a las de la onda de entrada. Cualquier cambio en la forma de onda se considera distorsión.

Un amplificador que preserva los detalles de la forma de onda de la señal se caracteriza por la relación

$$Vo(t) = AVi(t)$$

Donde Vi y Vo son las señales de entrada y salida respectivamente, A es una constante que representa la magnitud de la amplificación y se le conoce como la ganancia del amplificador.

Hasta ahora, se ha supuesto que los amplificadores operan con señales de entrada muy pequeñas. Su propósito es hacer más grande la amplitud de la señal y por tanto se consideran como amplificadores de voltaje o de tensión.

Otro tipo de señal de amplificador es el de potencia. Este tipo de amplificador otorga poca o ninguna ganancia de voltaje, pero si una sustancial ganancia de corriente.

II.2.1. CLASIFICACIÓN DE AMPLIFICADORES

Hay muchas clases y subclases de amplificadores electrónicos. Por lo que es difícil proponer una clasificación completa. Cuando había solamente amplificadores con tubo de vacío, estos se clasificaban como amplificadores de voltaje o de potencia. Aún hoy, los amplificadores se clasifican en dos categorías.

a.1) Una de las categorías incluye amplificadores de voltaje. Estos amplificadores se diseñan de tal forma que pueden levantar el nivel de voltaje o de corriente de señal. Desde luego que los voltajes y corrientes no son en general cantidades independientes, pero conviene usar un modelo de amplificación de voltaje o uno de corriente.

a.2) La otra categoría general la constituyen los amplificadores de potencia, donde el rendimiento es una consideración importante y los niveles de la señal generalmente son altos.

Los amplificadores también se pueden clasificar como sintonizados y no sintonizados.

b.1) Los sintonizados llevan circuitos resonantes sintonizados y generalmente operan en un intervalo de frecuencias, por encima de algunos centenares de KHz. Estos amplificadores son generalmente de banda angosta, se diseñan con el propósito de selección de frecuencias específicas.

b.2) Los amplificadores no sintonizados son en general de banda ancha.

Los amplificadores también se clasifican de acuerdo a la posición del punto de operación sobre sus curvas características.

c.1) El amplificador clase A se polariza de tal manera que la magnitud de la corriente de colector sea mayor que cero, todo el tiempo y que la señal pueda variar igualmente por encima y por debajo del punto de operación. Los amplificadores no sintonizados de salida única (no balanceados con respecto a tierra) generalmente operan como clase A.

c.2) El amplificador clase B se polariza de tal manera que la corriente promedio de colector es casi cero. Las corrientes de colector aumentan en magnitud a medida que la señal de entrada aumenta en magnitud.

c.3) El amplificador clase AB tiene el punto de operación entre los de clase A y la clase B. La magnitud de la corriente de colector debe estar muy por encima del corte.

c.4) El amplificador clase C tiene el punto de operación por debajo del corte. La magnitud de la corriente de colector es mayor que cero, durante menos de la mitad de un ciclo del voltaje de la señal de entrada. Los amplificadores clase C son generalmente amplificadores sintonizados.

d) Los amplificadores también se clasifican, de acuerdo a su aplicación, algunos se diseñan para niveles de señal de entrada muy bajos y por tanto se llaman preamplificadores. El nombre dado a una categoría de amplificadores, generalmente indica el servicio a que se destinan, por ejemplo, amplificador para línea, amplificador de separación, amplificador monitor, amplificador de distribución, amplificador de radio frecuencia (RF), amplificador de frecuencia intermedia (IF), etcétera.

Un amplificador de RF es un amplificador sintonizado de alta ganancia y bajo ruido y es la primera etapa en un sistema de recepción. El propósito principal del amplificador de RF es lograr la selectividad, la amplificación, y la sensibilidad de una señal.

El término de RF simplemente significa que la frecuencia es bastante grande para ser eficientemente radiada por una antena y propagarse a través del espacio en forma de una onda electromagnética. La RF para la banda de emisión en AM se encuentra entre 530 y 1600 kHz, para la banda de FM entre 88 y 108 MHz, y para la radiación de microondas a partir de 1 GHz. Una frecuencia de IF común en receptores para la banda de emisión en FM, es de 10.7 MHz y para los receptores de AM es de 455 KHz.

La RF es simplemente la frecuencia radiada o recibida, y la IF es una frecuencia intermedia dentro de un transmisor o un receptor. Por lo que muchas de las consideraciones para los amplificadores de RF también son aplicados a los amplificadores de IF, tales como: el filtrado y el acoplamiento de impedancias.

II.3. FILTROS

Considerando que un filtro tiene el propósito de dejar pasar señales que tienen un espectro de frecuencias específico y suprime señales cuyo espectro de frecuencias difieren de éste en especial, se puede decir que los filtros son circuitos selectivos de frecuencias. Aunque se usan en casi todos los sistemas electrónicos su uso más extensivo se encuentra en los sistemas de comunicaciones.

De acuerdo a las bandas de frecuencias que se dejan pasar o suprimen, existen cuatro tipo de filtros: paso bajas, paso altas, paso banda, supresor de banda, así como un quinto filtro conocido como pasa todo en el cual la respuesta en magnitud es constante e independiente de la frecuencia

FILTRO PASO BAJAS. El filtro paso bajas ideal, es aquel que proporciona una salida constante a partir de cd hasta una frecuencia de corte fc y después de esta frecuencia no permite pasar ninguna señal.

En la figura 2.3.1a se muestra la respuesta en frecuencia de un filtro paso bajas.

FILTRO PASO ALTAS. El filtro paso altas ideal es aquel que sólo permite el paso de señales arriba de la frecuencia de corte fc y anula todas las señales que estén por debajo de esta frecuencia.

En la figura 2.3.1b se muestra la respuesta en frecuencia de un filtro paso altas.

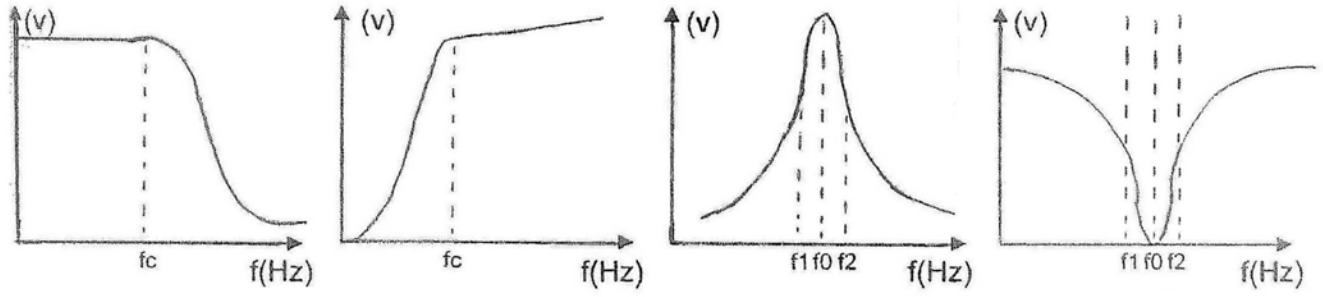
FILTRO PASO BANDAS. Cuando el circuito del filtro pasa señales que se encuentran arriba de una frecuencia de corte f1 y por debajo de una segunda frecuencia de corte f2 es un filtro paso banda.

En la figura 2.3.1 c se muestra la respuesta en frecuencia de un filtro paso bandas.

FILTRO SUPRESOR DE BANDAS. Cuando el circuito del filtro deja pasar señales que se encuentran debajo de una frecuencia de corte f1 y por arriba de una segunda frecuencia de corte f2 en un filtro supresor de banda ideal.

En la figura 2.3.1d se muestra la respuesta en frecuencia de un filtro supresor de banda.

FILTRO PASA TODO. Como ya se mencionó, en este tipo de filtros la respuesta en magnitud es constante e independiente de la frecuencia; por lo tanto resulta el nombre de pasa todo. Estos circuitos se utilizan para modificar la respuesta en fase.



a) b) c) d)

Fig. 2.3.1. Respuesta en frecuencia de un filtro: a) Paso baja, b) Paso altas, c) Paso banda y d) Supresor de banda.

II.3.1 ANÁLISIS DE TIPO DE FILTROS

Los filtros consisten primordialmente de reactancias, las cuales pueden adoptar las formas físicas de bobinas, capacitores, cristales y resonadores mecánicos. Los filtros que utilizan amplificadores en combinaciones con resistores y capacitores se denominan filtros activos.

FILTROS LC. Estos filtros constan de inductores y capacitores. Debido a las capacitancias e inductancias parásitas de sus elementos, su aplicación en bandas de ultra alta frecuencia (UHF) no es práctica. Los filtros de frecuencias bajas (LF) usan valores altos de inductancia y capacitancia que a su vez exigen componentes de tamaño grande. Por lo que se considera que la gama óptima de frecuencias de operación de los filtros LC para evitar los inconvenientes anteriores, es aproximadamente de 100 Hz a 300 MHz

FILTROS ACTIVOS. Como ya se mencionó, este tipo de filtros emplean amplificadores operacionales, resistores y capacitores. Estos filtros pueden ser diseñados para operar a frecuencias muy bajas lográndose resultados satisfactorios, pero también en frecuencias mayores que normalmente no superan los 10 MHz.

Se pueden obtener valores del factor de calidad (Q) de unos cuantos centenares en la gama inferior del espectro de frecuencias de operación, donde los amplificadores tienen una ganancia muy alta de lazo abierto. La reducción de la ganancia de lazo abierto limita los Q obtenibles a las frecuencias más altas. El factor de calidad Q en un circuito es la relación entre la energía reactiva que almacena y la energía que disipa durante un ciclo completo de la señal. Es un parámetro importante para los filtros y otros circuitos sintonizados, pues proporciona una medida de lo aguda que es su resonancia.

El diseño de filtros activos permite establecer una impedancia deseada de entrada y salida, independiente de la frecuencia, además de que se dispone de ganancia de voltaje.

FILTROS DE CRISTAL. Los resonadores de cristal de cuarzo tienen un factor Q de hasta 1 000 000, por lo que se consideran elementos de filtros casi perfectos. Los filtros de cristal tienen una alta estabilidad, ya que los parámetros eléctricos del cuarzo permanecen esencialmente constantes con el tiempo y la temperatura.

Por factores económicos, se considera que los cristales son convenientes como elementos de filtros sólo cuando se requieren valores muy altos de Q y una alta estabilidad en filtros paso banda con ancho de banda muy estrecho.

FILTROS ELECTROMECÁNICOS. Este tipo de filtros reciben una señal eléctrica, misma que convierten en vibraciones mecánicas con un transductor, aplica esas vibraciones a una serie de discos interconectados y vuelve a convertir las vibraciones resultantes en señales eléctricas de salida. Mediante el diseño apropiado de estos discos metálicos se pueden obtener resonancias mecánicas de un factor Q alto, de modo que cada disco sea el equivalente mecánico de un circuito resonante eléctrico en paralelo. Puesto que esos discos se acoplan mecánicamente, la señal de entrada se ve afectada por la respuesta de cada disco al pasar entre transductores de entrada y salida.

Los filtros mecánicos son más apropiados para filtros paso banda de ancho de banda estrecho, en la gama de frecuencias de 50 a 500 kHz. Se pueden obtener valores de Q de paso bandas de hasta

1000, con una buena estabilidad de frecuencia. Estos filtros son utilizados en sistemas de comunicación de *GPS* y aplicaciones que utilizan bandas libres.

Uno de los inconvenientes principales de este tipo de filtros es la elevada pérdida por inserción. Esto se debe principalmente a la ineficiencia de los transductores de entrada y salida.

La figura 2.3.2 muestra la gama de frecuencias operacionales para los diferentes tipos de filtros existentes.

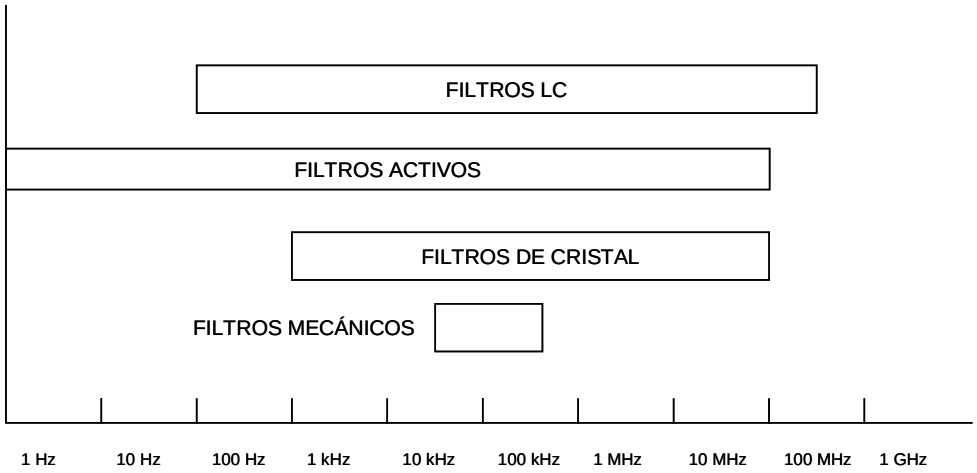


Fig. 2.3.2. Gama de frecuencias de aplicación de los filtros.

II.4. ANTENAS

La definición formal de una antena es un dispositivo que sirve para transmitir y recibir ondas de radio. Convierte la onda guiada por la línea de transmisión (el cable o guía de onda) en ondas electromagnéticas que se pueden transmitir por el espacio libre.

II.4.1. PARÁMETROS GENERALES DE UNA ANTENA

Una antena va a formar parte de un sistema (transmisor – antena – receptor), por lo que se tienen que definir los parámetros que la describan y permitir entonces evaluar el efecto que va a producir sobre dicho sistema.

Los principales parámetros a considerar en la caracterización de una antena son:

IMPEDANCIA. Una antena se tendrá que conectar a un transmisor y deberá radiar el máximo de potencia posible con un mínimo de pérdidas. Se deberá adaptar la antena al transmisor para una máxima transferencia de potencia, que se suele hacer a través de una línea de transmisión. Esta línea también influirá en la adaptación, debiéndose considerar su impedancia característica, atenuación y longitud.

Como el transmisor producirá corrientes y campos, a la entrada de la antena se puede definir la impedancia de entrada mediante la relación tensión-corriente en ese punto. Esta impedancia poseerá una parte real $R_e(w)$ y una parte imaginaria $R_i(w)$, dependientes de la frecuencia.

Si a una frecuencia una antena no presenta parte imaginaria en su impedancia $R_i(w)=0$, se dirá que esa antena está resonando a esa frecuencia.

Normalmente se usará una antena a su frecuencia de resonancia, que es cuando mejor se comporta, luego, a partir de este momento no se hablará de la parte imaginaria de la impedancia de la antena, si no que se hablará de la resistencia de entrada a la antena R_e. Lógicamente esta resistencia también dependerá de la frecuencia.

Esta resistencia de entrada se puede descomponer en dos resistencias, la resistencia de radiación (R_r) y la resistencia de pérdidas (R_L). Se define la resistencia de radiación como una resistencia que disiparía en forma de calor la misma potencia que radiaría la antena. La antena por estar compuesta por conductores tendrá unas pérdidas en ellos. Estas pérdidas son las que definen la resistencia de pérdidas en la antena.

Nos interesa que una antena esté resonando para que la parte imaginaria de la antena sea cero. Esto es necesario para evitar tener que aplicar corrientes excesivas, que lo único que hacen es producir grandes pérdidas.

EFICIENCIA. Relacionado con la impedancia de la antena tenemos la Eficiencia de Radiación y la Eficiencia de Reflexión. Estas dos eficiencias nos indicarán una, cuanto de buena es una antena emitiendo señal, y otra, cuanto de bien está adaptada una antena a una línea de transmisión.

La Eficiencia de Radiación se define como la relación entre la potencia radiada por la antena y la potencia que se entrega a la misma antena. Como la potencia está relacionada con la resistencia de la antena, se puede volver a definir la Eficiencia de Radiación como la relación entre la Resistencia de radiación y la Resistencia de la antena:

La Eficiencia de Adaptación o Eficiencia de Reflexión es la relación entre la potencia que le llega a la antena y la potencia que se le aplica a ella. Esta eficiencia dependerá mucho de la impedancia que presente la línea de transmisión y de la impedancia de entrada a la antena, luego se puede volver a definir la Eficiencia de Reflexión como 1 - Módulo del Coeficiente de reflexión, siendo el coeficiente de reflexión el cociente entre la diferencia de la impedancia de la antena y la impedancia de la línea de transmisión y la suma de las mismas impedancias.

Eficiencia de Reflexión = 1 - (Módulo del Coeficiente de Reflexión)

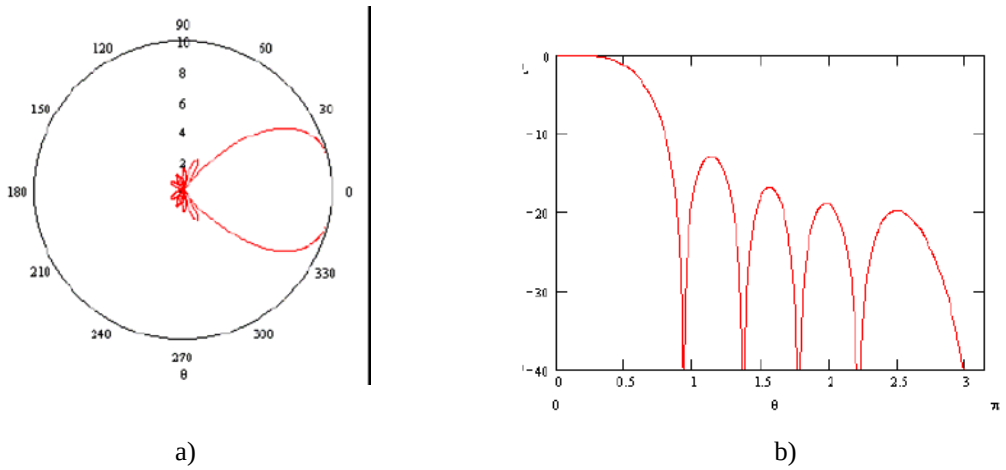
Algunas veces se define la Eficiencia Total, siendo ésta el producto entre la Eficiencia de Radiación y la Eficiencia de Reflexión.

Eficiencia Total = Eficiencia de Radiación x Eficiencia de Reflexión

DIAGRAMA DE FASE O PATRÓN DE RADIACIÓN. Un patrón de radiación es un diagrama polar o gráfico que representa las intensidades de los campos o las densidades de potencia en varias posiciones angulares en relación con una antena. Si el patrón de radiación se traza en términos de la intensidad del campo eléctrico (E) o de la densidad de potencia (P), se llama patrón de radiación absoluto. Si se traza la intensidad del campo o la densidad de potencia en relación al valor en un punto de referencia, se llama patrón de radiación relativo.

La mayoría de las veces no interesa el diagrama de radiación en tres dimensiones, al no poder hacerse mediciones exactas sobre él. Por lo que se suele hacer es un corte en el diagrama de radiación en tres dimensiones para pasarlo a dos dimensiones. Este tipo de diagrama es el más habitual ya que es más fácil de medir y de interpretar.

En la fig. 2.4.1a se muestra un corte bidimensional de un patrón de radiación en coordenadas polares y la fig. 2.4.1b en coordenadas cartesianas y escala logarítmica.



a) b)

Fig. 2.4.1. Corte bidimensional de un patrón de radiación: a) Coordenadas polares y b) Coordenadas cartesianas y escala logarítmica

CAMPOS CERCANOS Y LEJANOS. El campo de radiación que se encuentra cerca de una antena no es igual que el campo de radiación que se encuentra a gran distancia. El término campo cercano se refiere al patrón de campo que está cerca de la antena, y el término campo lejano se refiere al patrón de campo que está a gran distancia. Durante la mitad del ciclo, la potencia se irradia desde una antena, en donde parte de la potencia se guarda temporalmente en el campo cercano. Durante la segunda mitad del ciclo, la potencia que está en el campo cercano regresa a la antena. Esta acción es similar a la forma en que un inductor guarda y suelta energía. Por tanto, el campo cercano se llama a veces campo de inducción. La potencia que alcanza el campo lejano continúa irradiando lejos y nunca regresa a la antena por lo tanto el campo lejano se llama campo de radiación. La potencia de radiación, por lo general es la mas importante de las dos, por consiguiente, los patrones de radiación de la antena, por lo regular se dan para el campo lejano. El campo cercano se define como el área dentro de una distancia D^2/l de la antena, en donde l es la longitud de onda y D el diámetro de la antena en las mismas unidades.

GANANCIA DIRECTIVA Y GANANCIA DE POTENCIA. La ganancia directiva es la relación de la densidad de potencia radiada en una dirección en particular con la densidad de potencia radiada al mismo punto por una antena de referencia, suponiendo que ambas antenas irradian la misma cantidad de potencia. El patrón de radiación para la densidad de potencia relativa de una antena es realmente un patrón de ganancia directiva si la referencia de la densidad de potencia se toma de una antena de referencia estándar, que por lo general es una antena isotrópica. La máxima ganancia

directiva se llama directividad. La ganancia de potencial es igual a la ganancia directiva excepto que se utiliza el total de potencia que alimenta a la antena (o sea, que se toma en cuenta la eficiencia de la antena). Se supone que la antena indicada y la antena de referencia tienen la misma potencia de entrada y que la antena de referencia no tiene perdidas (h = 100%). Matemáticamente, la ganancia de potencia (Ap) es:

$$Ap = D\,h \qquad \text{donde: } D = \text{ganancia directiva (sin unidades)}$$

Si una antena no tiene perdidas, irradia 100% de la potencia de entrada y la ganancia de potencia es igual a la ganancia directa.

POLARIZACIÓN DE LA ANTENA. La polarización de una antena se refiere sólo a la orientación del campo eléctrico radiado desde ésta. Una antena puede polarizarse en forma lineal (por lo general, polarizada horizontal o vertical), en forma elíptica o circular. Si una antena irradia una onda electromagnética polarizada verticalmente, la antena se define como polarizada verticalmente; si la antena irradia una onda electromagnética polarizada horizontalmente, se dice que la antena está polarizada horizontalmente; si el campo eléctrico radiado gira en un patrón elíptico, está polarizada elípticamente; y si el campo eléctrico gira en un patrón circular, está polarizada circularmente.

ANCHO DEL HAZ DE LA ANTENA. El ancho del haz de la antena es solo la separación angular entre los dos puntos de media potencia (-3dB) en el lóbulo principal del patrón de radiación del plano de la antena, por lo general tomando en uno de los planos "principales".

ANCHO DE BANDA DE LA ANTENA. El ancho de banda de la antena se define como el rango de frecuencias sobre las cuales la operación de la antena es "satisfactoria". Se puede definir un ancho de banda de impedancia, de polarización o de ganancia

II.4.2. TIPO DE ANTENAS

Las antenas tienen la siguiente clasificación:

- Antena colectiva: Es una antena receptora que, mediante la conveniente amplificación y el uso de de distribuidores, permite su utilización por diversos usuarios
- Antena de cuadro: Antena de escasa sensibilidad, formada por una bobina de una o varias espiras arrolladas en un cuadro, cuyo funcionamiento bidireccional la hace útil en radiogoniometría.
- Antena de reflector o parabólica: Antena provista de un reflector metálico, de forma parabólica, esférica o de bocina, que limita las radiaciones a un cierto espacio, concentrando la potencia de las ondas; se utiliza especialmente para la transmisión y recepción vía satélite.
- Antena lineal: Es aquella que está constituida por un conductor rectilíneo, generalmente en posición vertical.
- Antena multibanda: Es aquella que permite la recepción de ondas cortas en una amplitud de banda que abarca muy diversas frecuencias.
- Dipolo de Media Onda: El dipolo de media onda lineal o dipolo simple es una de las antenas más ampliamente utilizadas en frecuencias arriba de 2MHz. En frecuencias abajo de 2 MHz, la longitud física de una antena de media longitud de onda es prohibitiva. Al dipolo de media onda se le refiere por lo general como antena de *Hertz* .

- Antena Yagi: Antena constituida por varios elementos paralelos y coplanarios, directores, activos y reflectores, utilizada ampliamente en la recepción de señales televisivas.

II.5. ARQUITECTURA DE UNA COMPUTADORA PERSONAL

Una computadora personal *PC (Personal computer)* es un sistema digital que ejecuta una secuencia de operaciones, sobre datos que se expresan en forma binaria.

Una computadora puede ser un conjunto de elementos individuales que tienen relación entre ellos para lograr un objetivo común. Una *PC* se divide en dos subsistemas llamados *hardware* y *software*.

El *hardware* es la parte eléctrica, electrónica y mecánica del sistema, es decir, involucra todos los componentes físicos de la *PC*.

El *hardware* se divide en: unidad de procesamiento central, memorias y dispositivos de entrada/salida. Fig. 2.5.1.

El *hardware* requiere de otro subsistema para funcionar, es decir, del *software*, que tiene los elementos necesarios para el control del sistema, es decir, programas de aplicación, algoritmo y procedimientos para solucionar problemas. El *software* es el término genérico de todos los programas de un sistema de procesamiento de datos.

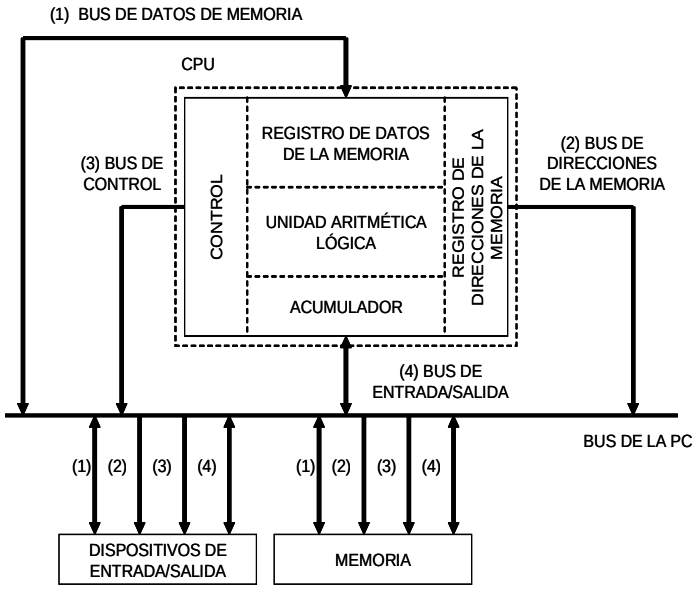


Fig. 2.5.1. Organización básica del hardware de una *PC*.

II.5.1. UNIDAD CENTRAL DE PROCESAMIENTO

La Unidad Central de Procesamiento *CPU (Central Processing Unit)* de la fig. 2.5.1. es el cerebro de la computadora. Ejecuta los programas almacenados en la memoria principal a través de las secuencias de operaciones básicas, con la búsqueda (*fetching)* de sus instrucciones, examinándolas y ejecutándolas.

II.5.2. UNIDAD DE CONTROL

Esta unidad tiene el papel supervisor para toda la máquina, adquiere las instrucciones de la memoria en el orden apropiado, interpreta las instrucciones de la memoria y hace que los componentes apropiados de la máquina efectúen las operaciones especificadas por las instrucciones. Esta unidad es la parte más importante del hardware de la computadora, ya que interpreta los comandos del usuario dirigidos al sistema de programas.

La unidad de control ejecuta dos tareas:

- Controla los eventos en el procesador central, como la transferencia de datos de una localidad de memoria a otra, o de una localidad de memoria a la unidad aritmética – lógica.

- Controla el flujo de datos entre el procesador central y los dispositivos periféricos. Para esto usa las líneas de dirección y control.

La unidad de control coordina todas las unidades funcionales de la *PC,* con una secuencia lógica en un tiempo correcto. Un reloj central controlado genera los pulsos de reloj. El resto de los circuitos están sincronizados con estos pulsos. La unidad de control recibe las instrucciones del programa almacenándolas en la memoria principal, después, las decodifica y dirige al resto de unidades funcionales para ejecutarlas.

II.5.3. UNIDAD ARITMÉTICA – LÓGICA

La Unidad Aritmética Lógica *ALU (Arithmetic – Logical Unit)* ejecuta operaciones para cálculo de datos numéricos, tales como adición, resta, etcétera; enseguida almacena el resultado en un registro de salida y de allí regresa a la memoria si así se desea. La *ALU* también toma decisiones lógicas, es decir, puede comparar un par de datos para determinar si un dato es mayor que, menor que, o igual que otro.

II.5.4. REGISTROS

La *CPU* contiene una memoria pequeña de alta velocidad usada para almacenar resultados temporales y cierta información de control. Esta memoria consiste de un número de registros, cada uno de ellos con una función especial.

Los registros más importantes son:

Acumulador. Es el registro principal de trabajo del microprocesador. En muchos microcontroladores los resultados de las operaciones aritméticas o lógicas realizadas en la *ALU* se transfieren y almacenan en el acumulador. El acumulador se considera un registro de propósito

general que almacena uno de los operandos que usa la *ALU,* al realizar operaciones aritméticas o lógicas. El acumulador puede ser registro fuente o registro destino.

Contador de Programa *PRC* (*Program Counter*). En un programa las instrucciones están almacenadas en posiciones sucesivas de la memoria de programa, que puede consistir de varios circuitos integrados de memoria *ROM* (*Read Only Memory*). A cada posición de memoria se le asigna un número o código único llamado dirección. Para ejecutar el programa en la secuencia correcta, el *CPU* debe conocer en que posición de memoria buscar la próxima instrucción. El *PRC,* es el registro que apunta a la siguiente instrucción que va a ser buscada, y por esa razón el *PRC* se le considera como el registro más importante de la *CPU.*

Registro de direcciones. Es un dispositivo de almacenamiento temporal, que mantiene la dirección de lectura o escritura de datos en la memoria de datos. En algunos microprocesadores más sofisticados, el registro de direcciones es programable y el programa puede modificar su contenido mediante las instrucciones adecuadas.

Registro de instrucciones. La búsqueda de instrucciones de la memoria de programa requiere dos operaciones independientes. Primero, la *CPU* transmite la dirección de la instrucción del contador del programa a la memoria. La memoria entonces transmite el contenido de la posición de memoria direccionada a la *CPU,* donde se almacena temporalmente en un registro especial llamado registro de instrucción.

Memoria. En una computadora el sistema de memoria se usa principalmente para dos propósitos:

- Para disponer de un sistema de almacenamiento de datos u operandos.

- Para disponer de un medio para almacenar el programa (un grupo de comandos e instrucciones).

Existen dos tipos de memorias en la computadora: Memoria principal y memoria secundaria.

Memoria principal. Es la parte de la computadora donde se almacenan programas y datos.

La porción de memoria destinada al almacenamiento de operandos se llama memoria de datos. Esta memoria almacena los datos que la computadora usa durante la ejecución del programa. La memoria de datos es del tipo memoria de acceso aleatorio *RAM* (*Random Access Memory*).

La parte de la memoria que almacena los comandos o instrucciones se llama memoria de programa. Cada instrucción de esta memoria es suministrada a la computadora en una cierta secuencia preestablecida. La computadora decodifica cada instrucción e inicia el proceso especifico ordenado por ésta. La memoria de programa es el del tipo sólo lectura *(ROM).*

Memoria secundaria. La memoria secundaria se usa para mantener datos en un lugar lejano con respecto a la memoria principal. En seguida se describen algunos tipos de memoria secundaria más utilizados:

Discos magnéticos. Es una pieza de metal circular la cual tiene una cubierta magnetizable puesta directamente en el proceso de manufactura, generalmente en ambos lados. La información se graba en círculos concéntricos llamados pistas.

Cada manejador de disco (*drive*) tiene una cabeza movible que se desplaza del centro hacia fuera. La cabeza tiene la anchura exacta para leer o escribir información solamente en una pista.

Casi todas las computadoras usan discos múltiples para su registro principal de datos. Estos se llaman discos duros (*Hard disk*).

Discos floppy. En la actualidad y ya casi en desuso se usan discos *floppy* de 3.5 pulgadas (conocidos como *disquetes*). Los *disquetes* de 3.5 pulgadas vienen con una cubierta rígida para su protección. Su capacidad de almacenamiento es de 1.38 Mbytes. Este dispositivo fue inventado por IBM (*Internacional Buisness Machines*) para grabar información de mantenimiento para *mainframes* pero fue rápidamente adquirido por los fabricantes de computadoras personales para distribuir la venta de *software* y para transferir información entre usuarios.

Discos ópticos. A diferencia de los magnéticos tienen mayor densidad de grabación que los discos magnéticos convencionales. Estos discos se basan en la misma tecnología de los *compact disk* y se llaman *CD ROM`s* (*Compact disk Read Only Memory*). Los *CD ROM´s* se leen con dispositivos similares a los reproductores de CD de audio. En el *CD ROM* se obtiene un mejor grabado que en los discos *floppy*, estos pueden producirse en masa con una máquina automatizada a bajo costo. La información de un *CD ROM* se escribe como una espiral continua simple, a diferencia de los discos magnéticos, con sus cilindros discretos y sus pistas. Cada *CD ROM* contiene 270 000 bloques de datos, para una capacidad total de 700 Mega bytes.

II.5.5. SISTEMA OPERATIVO

El sistema operativo es un programa que se encarga de administrar los recursos de la computadora.

Dentro de las ventajas del uso de los sistemas operativos se tienen las siguientes:

- Permite aumentar la velocidad de proceso, ejecutando programas independientes en forma consecutiva y sin interrupción.
- Incluye compiladores y ensambladores que facilitan la programación.
- La participación del operador se reduce al mínimo.
- Incluye programas de definición para ayudar al programador a descubrir y corregir errores de programación.
- Permite reducir el tiempo ocioso de la *CPU* al atender por separado la ejecución de trabajos en ésta y la ejecución de entradas y salidas

El sistema operativo tiene el control de todos los programas. Esta es la tarea más importante del sistema.

II.5.6. DISPOSITIVOS DE ENTRADA/SALIDA

Los dispositivos de entrada/salida (*I/O*) son el medio que usa la computadora para transferir información.

No todas las entradas o salidas van hacia el operador, ya que si conectamos la computadora como un operador, la información puede llegar de otros medios, como son los sensores, o llegar hacia ellos.

Los dispositivos de entrada/salida más comunes en las computadoras son: el teclado, el ratón, la impresora y el monitor, aunque hay muchos más dispositivos de *I/O* disponibles hoy en día.

El teclado es un conjunto ordenado de teclas que al pulsarse realizan la comunicación del usuario con la máquina. Existe a la venta una gran variedad de ellos.

El monitor es un tubo de rayos catódicos *CRT* (*Cathode Ray Tube*) y su fuente de poder. El *CRT* tiene un cañón de electrones que son dirigidos hacia una pantalla fosforescente (Para las pantallas de color se tiene 3 cañones de electrones, uno para cada color: rojo, verde y azul)

En la actualidad también se cuentan con monitores de cristal líquido *LCD (Liquid Cristal Display)*, esta tecnología funciona a través de un arreglo de transistores tan delgados como una película de cualquier cámara de fotos, de ahí el *TFT* (*thin-film transistor*), que alimentan de voltaje a celdas llenas de líquido empaquetadas entre dos cristales. Dependiendo del voltaje, el líquido despliega el color, brillo y contraste requerido.

El ratón es una pequeña caja de plástico que se coloca sobre la mesa de la computadora. Cuando éste se mueve sobre la mesa el cursor también se mueve en la pantalla. El ratón tiene uno, dos o tres botones para seleccionar fácilmente las opciones del menú.

Las computadoras personales tienen una estructura física sencilla. Muchas de estas máquinas tienen una caja con una tarjeta de circuito impreso en la parte superior, llamada “Tarjeta Madre” (*Mother Board*). Esta contiene el circuito impreso de la *CPU*, memoria y varios *CI* de soporte. También contienen un *bus* y ranuras en las que se conecta memoria adicional y tarjetas de *I/O*.

La estructura lógica de una *PC* es mostrada en la figura 2.5.2. Muchas *PC* tienen sólo un *bus* para conectar la *CPU*, la memoria y los dispositivos *I/O*. Cada dispositivo de *I/O*, consiste de dos partes, una que contiene la parte de electrónica llamada el “Controlador” y otra, que contiene el dispositivo de *I/O*.

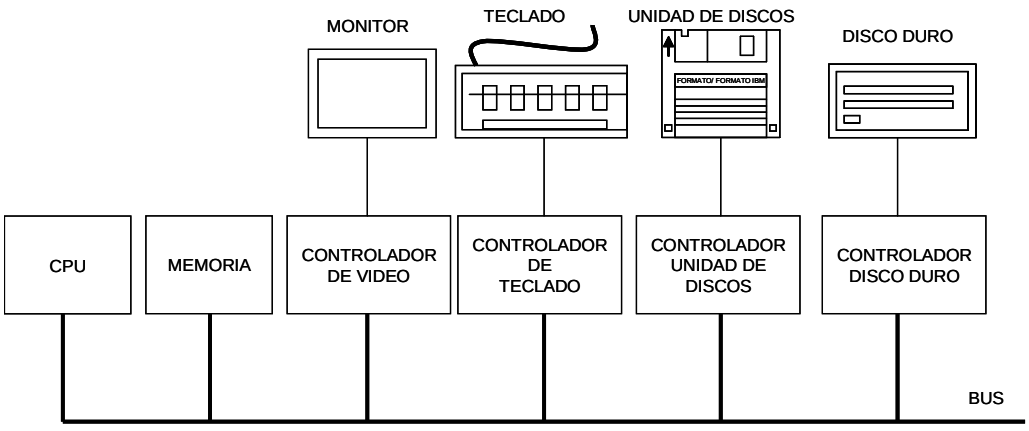


Fig. 2.5.2. Estructura lógica de una computadora personal.

Los controladores están conectados generalmente en el bus de la *PC* excepto para aquellos controladores que no son opcionales (como el teclado y el monitor), los cuales están en la tarjeta madre.

El trabajo de un controlador es manejar su dispositivo de *I/O* y maneja el *bus* de acceso hacia él. Cuando un programa quiere datos del disco, manda un comando al controlador del disco, el cual inicia una búsqueda. Cuando se localizan la pista y el sector, el *drive* comienza a enviar los datos como un flujo de *bytes* en serie. Cuando el controlador lee o escribe un bloque de datos a otra memoria sin la intervención de la *CPU*, se dice que está desempeñando un acceso directo a memoria *DMA (Direct Memory Access*).

Si la *CPU* y algún controlador quieren usar el *bus* al mismo tiempo, existe un circuito que decide quien usa el *bus* primero y quién después. En general, los dispositivos de *I/O* tienen preferencia sobre la *CPU*, ya que los dispositivos de movimiento como los discos no se pueden parar, y forzarlos a esperar podría resultar en una pérdida de datos.

La computadora tiene como funciones principales los siguientes puntos:

- Almacenamiento de los datos.
- Procesamiento para obtener información adicional.
- Despliegue de los datos de la pantalla.

II.6. METODOS DE TRANSMISIÓN DE DATOS.

Los métodos de transmisión más comunes son:

Simplex, Half-duplex y full-duplex:

SIMPLEX. Un enlace en modo *simplex* lleva información en una sola dirección, esto es, que sólo transmite o sólo recibe información. Un enlace en modo *simplex* es la elección apropiada cuando sólo se necesita enviar una información de una terminal a otra y nunca se necesita recibir información.

Una de sus ventajas es su bajo costo de instalación y es simple en el *software*. Sin embargo, no puede enviar información en las dos direcciones, sus aplicaciones son limitadas.

HALF-DUPLEX. Para muchas aplicaciones es necesario comunicar datos en ambas direcciones (transmitir y recibir). La forma menos cara de conseguir esto es con el *Half-duplex*, en la que la línea lleva la información en una dirección a la vez.

En un sistema *Half-duplex*, las dos terminales de la línea pueden comunicar no sólo información si no también sus caracteres de información. Suponiendo que la información viaja en una terminal A a una terminal B de izquierda a derecha. La terminal A es la que envía y la B es la que recibe. Cuando la terminal A termina de enviar sus datos, la terminal B reconoce la recepción de datos y entonces pide la línea. Cuando la terminal A termina de enviar y comienza a recibir, se tiene una conmutación de línea. Cuando la terminal B termina de transmitir, la terminal A puede pedir la línea y conmutará de nuevo.

Un sistema *Half-duplex* requiere un circuito para realizar la conmutación de la línea. El tiempo que el sistema invierte cambiando la línea es tiempo perdido. Sin embargo, *Half-duplex* es menos caro que el sistema *Full-duplex,* por que el primero necesita sólo dos alambres para efectuar la transmisión de datos.

FULL-DUPLEX. En un sistema *Full-duplex* la información viaja en ambas direcciones (transmite y recibe) a la vez.

Puesto que los datos son transmitidos como señales analógicas o digitales, se debe considerar el ancho de banda. Cuando se transmite en una dirección a la vez con *Half-duplex* o *simplex*, se puede usar la banda entera, o el rango de frecuencias que la línea tiene disponible. Si se trata de transmitir en ambas direcciones a la vez, cada dirección puede usar sólo la mitad de la banda. Si se usan cuatro alambres en lugar de dos, que son los usuales, entonces se le puede dar rapidez al sistema *Full-duplex*.

II.6.1. TRANSMISIÓN DE CARACTERES

En la comunicación de datos, los *bits* en un carácter pueden ser transmitidos usando transmisión en paralelo o transmisión serial.

TRANSMISIÓN EN PARALELO. En la transmisión en paralelo un carácter completo (de 8 *bits*) es transmitido a la vez: cada *bit* se envía por cable y se desplaza paralelamente a los otros *bits*. Las posiciones de los *bits* en los cables no varían.

Sin embargo, sólo podemos usar transmisión en paralelo sobre distancias menores a 1.52 mts. Más allá de esta distancia, la interferencia por radiofrecuencia se convierte en un problema y la señal de datos se vuelve muy débil.

Los periféricos de un sistema de cómputo, por ejemplo, una impresora, frecuentemente usan una conexión en paralelo; en general, cuando la distancia entre los dispositivos es pequeña, la transmisión en paralelo es la más acertada.

TRANSMISIÓN SERIAL. Con la transmisión serial, un carácter se mueve en un cable a la rapidez de un *bit* por unidad de tiempo. Se puede usar transmisión serial sobre distancias grandes o cortas, lo que hace el método más común para transmitir datos; la comunicación serial de datos se lleva a cabo de la siguiente manera:

Primeramente, el receptor debe saber cuando iniciar la búsqueda del primer *bit* en el primer carácter. Después de esto, si el reloj de las dos terminales están corriendo a la misma velocidad, el transmisor y el receptor permanecerán sincronizados.

Podemos especificar la velocidad de la transmisión de datos en *bauds o bits* por segundo (*bps*). Estos términos están muy relacionados pero no son idénticos aunque la mayoría de los especialistas en comunicación de datos los utilizan de manera indistinta.

Un *baud* es la unidad de velocidad de modulación, se refiere al lado analógico, y se define como el número de cambios de estado de la señal analógica en un segundo, un cambio de estado puede ser un cambio de amplitud, frecuencia, fase o una combinación de éstas.

Un *bps* es una unidad de información, se utiliza para expresar la velocidad de transmisión de la información, se refiere al lado digital y es la cantidad de bits transmitidos en un segundo.

SINCRONIZACIÓN POR CARÁCTER

Los sistemas seriales tienen dos alternativas de comunicación: Transmisión síncrona y asíncrona.

TRANSMISIÓN SÍNCRONA. En un bloque de transmisión síncrona, los caracteres de datos son organizados en bloques de longitudes estándar (frecuentemente 256 caracteres) y son las unidades de transmisión. Cada bloque está formado por un carácter de inicio de texto (*STX*) y un fin de texto (*ETX*). Fig. 2.6.1.

Fig. 2.6.1. Bloque de transmisión síncrono.

En este método de transmisión tanto el dispositivo emisor como el receptor operan simultáneamente y se resincronizan después de transmitirse algunos miles de *bits* de señal de datos.

No se requiere *bits* de inicio/parada por cada carácter. La sincronización se establece y mantiene cuando no se están transmitiendo señales o justo antes de la transmisión de una señal de datos. Esta sincronización se establece enviando un grupo predeterminado de caracteres de sincronización entre los dispositivos transmisor y receptor.

Durante una sesión de transmisión, el receptor comenzará por ver dos o más caracteres de espacio (*SPC*) de aviso de comienzo de transmisión de un bloque de datos o de separación entre bloque de datos. El carácter *SYN* deja al receptor sincronizado colocando el nivel adecuado para que el receptor interprete el carácter de arranque. Ahora el receptor sabe qué *bit* es el primer *bit* en el primer carácter, y puede leer los caracteres en el bloque que sigue. Después del carácter *SYN*, el receptor verá un carácter *STX*. Todo lo que sigue, hasta que el receptor vea un *ETX*, son datos (Fig. 2.6.2).

Fig. 2.6.2. Secuencia de transmisión de un bloque típico.

La transmisión síncrona se utiliza cuando es necesario un gran volumen y alta velocidad en la transmisión de datos. Debido a que los datos son empaquetados en bloque, esta transmisión de datos puede ser muy eficiente. En el mercado hay disponible software de comunicación síncrona para terminales inteligentes y microcomputadoras.

La transmisión síncrona es más eficiente en el sentido de que hay menos *bits* de control respecto al número total de *bits* transmitidos. La sincronización puede necesitar sólo de 16 a 32 *bits*, en tanto que la secuencia de *bits* para la señal de datos puede tener varios miles.

TRANSMISIÓN ASÍNCRONA. Al modo de transmisión asíncrona se le conoce como transmisión de arranque y parada, por que el dispositivo transmisor puede transmitir un carácter en cualquier momento que sea conveniente y el dispositivo receptor lo acepta, por lo que es posible enviar caracteres en intervalos irregulares.

Para este tipo de transmisión, cada carácter debe permanecer independiente. El receptor debe sincronizarse a cada carácter independiente (fig. 2.6.3).

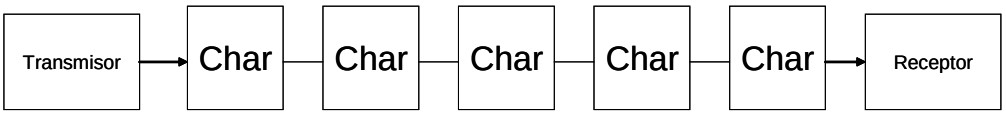


Fig. 2.6.3. Transmisión asíncrona de caracteres.

En una transmisión asíncrona, cada carácter esta formado por *bits* de inicio y *bits* de parada (Fig. 2.6.4). El *bit* de inicio le dice al receptor que un carácter está llegando y el *bit* de parada le dice al receptor que espere otro *bit* de inicio antes de muestrear. El receptor sabe cuantos *bits* deberían ir dentro de un carácter. Este toma ese número de *bits* y lo guarda en un espacio del carácter. La mayoría de los sistemas usan un *bit* de inicio y dos *bits* de parada. En la transmisión asíncrona se usa la verificación del carácter por paridad, el cual detecta errores en la información.

Fig. 2.6.4. Estructura de un carácter asíncrono.

II.6.3. PROTOCOLOS DE COMUNICACIÓN

Para que dos entidades puedan comunicarse, deben “hablar el mismo lenguaje”. Qué, cómo y cuándo se comunican estas entidades deben conformar un arreglo de convenciones mutuas. Un protocolo es un conjunto de reglas y procedimientos que proporcionan una técnica uniforme para gobernar el intercambio de información entre dos o más entidades.

En el caso de la comunicación de datos, estás reglas están diseñadas para resolver los problemas de operación en las siguientes áreas:

- Detección: Es determinar del grupo de 8 *bits* cuáles son caracteres, y lo más importante, que grupo de caracteres constituyen mensajes.
- Control de errores: La detección de errores en la aceptación de mensajes y la solicitud para aceptar mensajes defectuosos, se efectúan mediante varias técnicas, tales como verificación longitudinal, vertical y redundancia cíclica.

- Control de frecuencia: Es la enumeración de los mensajes para eliminar aquellos que son repetidos.
- Transparencia: El usuario debe ser capaz de elegir el código o patrones de *bits* a usar en la transferencia de datos.
- Control de línea: Es la determinación de cuál estación va a transmitir y cuál estación va a recibir la información correspondiente. En caso de línea duplex o multipunto.
- Casos especiales: Resuelve el problema de que envía el transmisor cuando no hay un dato para transmitir.
- Control de tiempo muerto: Resuelve el problema de qué hacer cuando el flujo de transmisión cesa por completo.
- Control de inicio: El proceso de obtención de transmisiones iniciadas en un sistema de comunicación deficiente.

Los protocolos de enlaces de datos se clasifican en dos categorías: Protocolo de *bit* orientado (*BOP*) y protocolos de *bit* controlado (*BCP*). Los protocolos de *bit* controlado usan caracteres de control específicos para informar al receptor cuando está mandando direcciones o datos y donde empiezan y terminan los *bits* de datos. El protocolo de *bit* orientado, depende de la posición de los *bits* dentro de un campo o un bloque.

El desarrollo teórico del trabajo de tesis, estará fundamentado en los temas que conforman este capítulo.

CAPÍTULO III

SELECCIÓN DE LOS DISPOSITIVOS Y SUS CARACTERÍSTICAS GENERALES

Como se mencionó en el capítulo 1 el sistema de cronometraje deportivo SDR-1000 estará conformado principalmente por un sistema de identificación por radio frecuencia *RFID* y un dispositivo de control con base en un microcontrolador que se seleccionará de acuerdo con los requerimientos de velocidad de procesamiento, memoria y comunicación. En este capítulo se presenta la selección de dichos dispositivos, así como sus características generales.

III.1. SISTEMA DE IDENTIFICACIÓN POR RADIOFRECUENCIA

De los múltiples requerimientos técnicos a cumplir por el sistema SDR-1000, para el que corresponde a la identificación de competidores se utilizará la tecnología *RFID* por las ventajas que ofrece la comunicación inalámbrica para dar cumplimiento con lo establecido por la Federación de Atletismo en lo concerniente a que en una carrera no deberá haber interferencias o acciones que afecte el desarrollo natural de la competencia.

La *RFID* es una tecnología ampliamente desarrollada, por lo que se realizó una investigación para determinar que firma ofrecía las mejores características técnicas y condiciones económicas para el desarrollo del SDR-1000. El esquema general de dicha tecnología se presenta en la figura 3.1.1

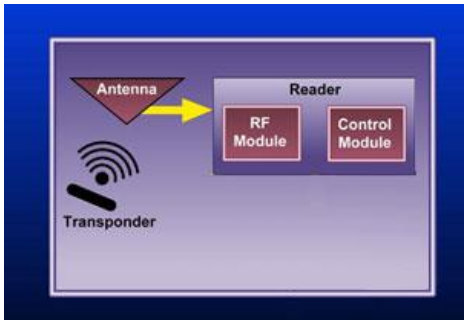


Fig. 3.1.1. Estructura general de la tecnología *RFID.*

En el estudio se determinó que las firmas ofrecen equipos de características muy similares, sin embargo, hubo puntos a considerar tanto en las lectoras y principalmente en los *transponders* que influyeron de manera determinante para tomar la decisión. Las antenas no entraron en el análisis por razones que más adelante se mencionan. Los resultados se muestran en la tabla 3.1.1 y 3.1.2

	TRANSPONDER				
MARCA COMERCIAL	# DE PARTE	DESCRIPCIÓN	PRECIO (USD)	DIMENSIÓN	ALCANCE
Atmel	TK5530HM-232-PP	Glass TAG R/O 125KHZ	$ 1.97	12 mm	10 cm
NXP	HT2DC20S20M	Glass TAG R/O 125KHZ	$ 1.61	12 mm	10 cm
Texas Instruments	**RI-TRP-RE2B-01**	**Glass TAG R/O 134 KHz**	**$ 2.42**	**32 mm**	**100 cm**
STId	H4102	Glass TAG R/O 125KHZ	$ 2.15	25 mm	40 cm
Intersoft	GTL12XRO	Glass TAG R/O 125KHZ	$ 3.98	12 mm	15 cm
AegId	GLASS id162/bde/3	Glass TAG R/O 134KHZ	$ 2.55	13.5 mm	20 cm
UKID	UKID 13mm	Glass TAG R/O 134KHZ	$ 1.98	13 mm	25 cm

Tabla 3.1.1. Comparación técnica del *transponder* diferentes marcas.

Una característica muy particular de los corredores de maratón, es que su zancada es muy grande, por lo que entonces el alcance del *transponder* es muy importante, ya que como más adelante se explicará este dispositivo se sujetará al zapato tenis del competidor y al paso por las antenas será leído, evidentemente entre mayor ser el alcance la posibilidad de ser registrado por el sistema aumenta.

Como se mencionó en el inicio de este trabajo, una de las premisas para su desarrollo fue el alto precio que tiene actualmente la renta de los equipos de cronometraje automático y no se diga su adquisición, por lo que entonces los costos son una razón importante para decidir.

Refiriéndonos a la tabla 3.1.1 podemos observar que el *transpoder* de la compañía *texas instruments* es el dispositivo de mayor alcance, y el de la firma *NXP* el más económico, sin embargo, de las dos premisas, es prioridad obtener un sistema confiable de cronometraje por lo que entonces el *transpoder texas instrument* sería el más adecuado para su uso en el sistema SDR-1000

Ahora bien, respecto a la unidad lectora tenemos lo siguiente:

	UNIDAD LECTORA		
MARCA COMERCIAL	COMUNICACIÓN	ALIMENTACIÓN	PRECIO (USD)
Atmel	RS-232	10 – 24 VDC	755.0
NXP	RS-232	10 – 24 VDC	780.0
Texas Instruments	RS-232 ó RS422/RS485	10 – 24 VDC	820.0
STId	RS-232 ó RS422/RS485	10 – 24 VDC	885.0
Intersoft	RS-232 ó RS422/RS485	10 – 24 VDC	825.0
AegId	RS-232	10 – 24 VDC	821.0
UKID	RS-232 ó RS422/RS485	10 – 24 VDC	770.0

Tabla 3.1.2. Comparación técnica de la unidad lectora diferentes marcas.

Las características técnicas de las lectoras son muy parecidas (tabla 3.1.2), por lo que entonces será mandatorio para el sistema de *RFID* a utilizar en el SDR-1000 los resultados obtenidos en los *transponders.* Será entonces obligado el uso de las lectoras de *texas instruments* que si bien es cierto no son las más económicas tendrán que utilizarse como parte integral del equipo de *RFID* esta firma comercial.

Bien, entonces el equipo de *RFID* seleccionado es el de la compañía *texas instruments*, el cual es denominado *TIRIS* (*Texas Instruments Registration and Identification System*).

TIRIS definido por el mismo *texas instruments* es un medio altamente confiable para controlar, detectar y rastrear una gran variedad de elementos como boletos, animales, paquetes, etcétera. *TIRIS* es un sistema de identificación por radio frecuencia (*RFID*) basado en técnicas de transmisión de FM de baja frecuencia.

A continuación se presentan características generales de los componentes de la tecnología *RFID* del sistema *TIRIS* (*transponder*, unidad lectora y antenas).

III.1.1. SISTEMA *TIRIS* DE *TEXAS INSTRUMENTS*

Transponder

El sistema *TIRIS* tiene como base un pequeño dispositivo denominado *transponder* (transmisor + receptor), los cuales pueden ser adheridos o incorporados en un objeto a identificar. Los *transpoder*s operan bajo un principio en el que por un efecto inductivo, generado por las antenas del sistema, cargan un capacitor que se encuentra dentro del mismo *transponder*, convirtiéndose por unos instantes en una fuente de energía que alimenta al circuito de transmisión de dicho *transponder* (el tiempo aproximado de carga del capacitor es de 50 milisegundos), de esa forma, el *transpoder* no requiere para su funcionamiento una fuente de poder, como podría ser una batería.

El *transponder* cargado, responde transmitiendo los datos almacenados en su memoria mediante una señal de respuesta a través de las antenas que en su momento lo energizaron. En el caso de *transponders* de sólo lectura (*read only*), los datos vienen programados de fábrica con un código de 64 *bits*. En los de lectura-escritura el usuario puede almacenar datos en la memoria del *transponder*. El ciclo total de lectura del *transponder* es menor a 100 milisegundos, permitiendo obtener más de 10 lecturas por segundo. Los datos recibidos por el lector (*reader*) desde el *transponder* pueden ser enviados a una *PC* o un *PLC* mediante interfaces estándar (RS-232 y RS-485) o bien pueden ser almacenados o simplemente leídos por un lector portátil.

Los sistemas de identificación por radiofrecuencia (*RFID*) sobrepasan las limitaciones de otros sistemas de identificación automática, porque no requieren que el lector esté dentro del campo visual del *transponder*. Esto significa que funcionan eficientemente en ambientes con polvo, suciedad, humedad y mala visibilidad. Además, *TIRIS* puede leer a través de la mayoría de materiales no metálicos.

Unidad lectora

La unidad lectora (reader) es una parte esencial del sistema *TIRIS*, dado que es el componente que cuenta con las funciones de Radio Frecuencia y control requeridas para la comunicación con los *transponders*. Dicha unidad también se utiliza para cargar información a los *transponders* del tipo lectura/escritura (*read/write*).

Por medio de los puertos RS-232 ó RS422/RS485 la unidad lectora puede recibir instrucciones de una computadora utilizando cualquiera de los dos protocolos utilizados por el sistema *TIRIS* (*ASCII o TIRIS bus protocol*) y a su vez, la unidad lectora puede comunicarse con cualquier *transponder* que se encuentre dentro del rango de cobertura de las antenas. Las antenas pueden colocarse a un máximo de 5 metros de la posición de la unidad lectora.

La unidad lectora cuenta con las siguientes conexiones/interfaces:

- Interfaz de comunicación: RS232, RS422 ó RS485
- 8 Entrada /salidas (*I/O*) para propósitos generales
- 2 salidas de colector abierto
- *Bus* de sincronización
- Conectores para alimentación eléctrica
- Conectores para indicación de lectura de *transponders*
- Conector para antena

Protocolos de comunicación

Existen dos protocolos que se pueden utilizar con la unidad lectora, éstos son:

PROTOCOLO *ASCII*. Éste es un protocolo simple en el que se utilizan caracteres *ASCII* para dar instrucciones a la unidad lectora, incluso es posible utilizar un emulador para enviar los comandos *ASCII*. El protocolo *ASCII* sólo puede ser utilizado con la interfaz estándar RS-232 o RS-422.

TIRIS BUS PROTOCOL. Éste es un protocolo binario adecuado para la comunicación entre un dispositivo de control (por ejemplo una computadora personal) y hasta 31 unidades lectoras por medio de la interfaz RS422/485.

Antenas

Las antenas del sistema *TIRIS* tienen las siguientes funciones:

A través del pulso de energía transmitido por la unidad lectora, generar un campo magnético que por medio de inducción cargue a los capacitores de los *transponder* que se encuentran dentro del patrón de radiación de las antenas.

Captar la información contenida en los *transponder* y transmitirla a la unidad lectora. Los *transponders* tanto de sólo lectura como de lectura/escritura transmiten sus datos inmediatamente después a que fueron cargados sus capacitores.

Aplicaciones comunes

EL sistema de identificación por radiofrecuencia *TIRIS* es una elección ideal para aplicaciones como:

Inmovilización de automóviles.- Si la unidad lectora instalada en el vehículo no detecta el *transponder* codificado para ese vehículo, el auto no arrancará o se parará en determinado tiempo.

Identificación de animales.- *Transponder* especiales se inyectan en animales como perros, puercos, pájaros, etcétera; de tal forma que unidades lectoras portátiles puedan ser identificados, conteniendo datos de fecha de nacimiento, vacunas que les hayan sido aplicadas, número de camadas, etcétera.

Cronometraje deportivo.- Obtención de tiempos en diversas competencias atléticas, asociando un número de identificación al atleta con el tiempo en que efectuó el recorrido, así como la emisión de reportes con información clasificada por categoría, tiempos, etcétera.

Control de acceso.- Por medio de *transponder* en forma de etiquetas, colocadas en vehículos o en credenciales, permiten el acceso automático de automóviles a estacionamientos levantando la pluma o liberando torniquetes en el caso de personas para el acceso a edificios, instalaciones deportivas, bibliotecas, etcétera.

Identificación y captura de precios.- Por medio de *transponders* en forma de etiquetas colocadas a los diferentes insumos que se venden en centros comerciales, con la simple presentación en la caja del producto se obtienen su identificación, precio y total a pagar.

Diagrama de bloques

En la figura 3.1.2 se presenta un diagrama de bloques, mostrando los tres principales componentes del sistema *TIRIS* conformado por el *transponder*, la unidad lectora y las antenas.

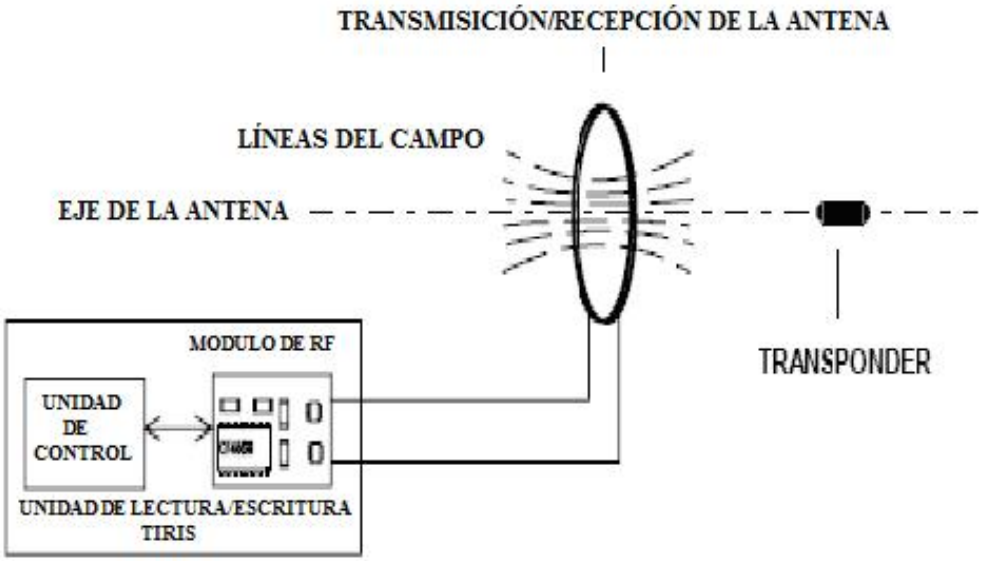


Fig. 3.1.2. Diagrama de bloque sistema *TIRIS.*

El *transponder* se compone por una antena, un capacitor y un circuito integrado. La inductancia de la antena y el capacitor forman un circuito resonante de alta calidad como se muestra en la figura 3.1.3

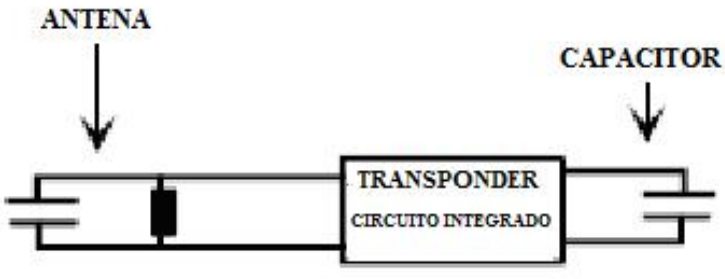


Fig. 3.1.3. Diagrama de bloque del *transponder*.

III.1.2. PROTOCOLO *ASCII*

Como se mencionó anteriormente el sistema *TIRIS* utiliza un protocolo definido por *texas instruments* denominado precisamente protocolo *ASCII* (*American Standard Code for Information Interchange)* por que utilizan caracteres *ASCII* para dar instrucciones a la unidad lectora por medio de su interfaz RS-232. A continuación, se proporciona una breve reseña de los orígenes del código *ASCII*, así como de la información técnica del protocolo *ASCII* establecido por t*exas instruments* (Tabla 3.1.3).

Código *ASCII*

Código Estadounidense Estándar para el Intercambio de Información es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y otras lenguas occidentales. Creado aproximadamente en 1963 por la Agencia Estadounidense de Estándares (*ASA*), como una evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyen las minúsculas y se redefinen algunos códigos de control para formar el código conocido como *ASCII*.

Casi todos los sistemas informáticos de hoy en día utilizan el código *ASCII* o una extensión compatible para representar textos y para el control de dispositivos que manejan texto.

ASCII consta de 128 códigos posibles, dividido en 4 grupos de 32 caracteres (7 *bits* de información por código), aunque utiliza menos de la mitad, para caracteres de control, alfabéticos (no incluye minúsculas), numéricos y signos de puntuación. Su principal ventaja, aparte de constituir un estándar, consiste en la ordenación alfabética de los códigos.

Normalmente el código *ASCII* se extiende a 8 *bits* (1 *byte*) añadiendo un *bit* de control, llamado *bit* de paridad.

Protocolo *ASCII*

Protocolo *ASCII*.- Este es un protocolo simple en el que se utilizan caracteres *ASCII* para dar instrucciones a la unidad lectora. El protocolo *ASCII* sólo puede ser utilizado con interfaz estándar RS-232 ó RS-422.

Para la comunicación entre el microcontrolador/computadora y la unidad lectora, así como la comunicación entre la unidad lectora y el microcontrolador/computadora, t*exas instruments* tiene definido un protocolo con base en comandos con caracteres *ASCII* para utilizarse en la programación del sistema.

A continuación se presenta un resumen del protocolo *ASCII* con los principales comandos a utilizar en el presente trabajo de tesis. Tabla 3.2.1

En la última columna del citado resumen de comandos se hace mención a los modos k0 y k1, mismos que tienen la siguiente correspondencia con los *transponder* que se utilicen para el desarrollo de este trabajo.

K0: *Transponder* de 64 *bit;* K1: *Transponder Multipage*

En el capítulo V se definirá el tipo de *transponder* a utilizar

	PROTOCOLO ASCII SISTEMA TIRIS (RESUMEN DE COMANDOS)				
COMANDO	**CARÁCTER ASCII**	**Lector ----> uC(PC)**	**EJEMPLO**	**DESCRIPCION**	**MODO**
	uC(PC) ---> Lector				
VERSION COMMAND	V	S2500 - REV 1.1x<CR><LF>		TRANSMITE NUMERO DE VERSION DE SOFTWARE DE LECTOR	K0-K1
FORMAT COMMAND	F	F<CR><LF>	L10M 05 ABCFED672889AD38<CR><LF> (MODO K1)	CAMBIA EL FORMATO DE LA INFORMACION ENVIADA DEL TRANSPONDER A LA PC (UNICAMENTE) DE DECIMAL A HEXADECIMAL HASTA QUE SE RECIBE EL COMANDO ESCAPE	K0-K1
CLEAR COMMAND	C	C<CR><LF>		SIN IMPORTAR EL MODO DEL TRANSPONDER BORRA EL BUFFER DE ALMACENAMIENTO EMPLEADO EN MODO NORMAL Y PONE EN 0 EL CONTADOR EMPLEADO EN MODO GATE	K0-K1
READOUT BUFFER COMMAND	B	LECTURA : B10M 01 0000 0000000000000001<CR><LF>		LEE EL CONTENIDO DEL BUFFER	K0-K1
		NO LECTURA : B<CR><LF>			
I/O STATUS COMMAND	J	J2A<CR><LF>	I/O: 0=0, 1=1, 2=0, 3=0; 4=0, 5=1, 6=0, 7=1	ENVIA EL ESTADO DE LOS PINES 0 A 7	K0-K1
SET OUPUTS COMMAND	Y	Y		CAMBIA EL ESTADO DE LOS PINES 0 A 7	K0-K1
	1	<CR><LF>	I/O: 0=0, 1=0, 2=0, 3=0; 4=1, 5=0, 6=0, 7=0		
SET OUTPUT & GET I/O STATUS	H	H			
	I2	03<CR><LF>	I/O LINES OF PORT 0..3 : 0&1=1, 2&3=0	SOLICITA EL ESTADO Y LECT LO REGRESA DE LINEAS 0 A 3	K0-K1
SET CHARGE PERIOD COMMAND	Z	Z			
	20	<CR><LF>	PONE PERIODO DE CARGA EN 32 mS (20 h)	CAMBIA PERIODO DE CARGA EN RANGO DE 15 A 255 mS EN PASOS DE 1	K0-K1
EXECUTE COMMAND	X	XR 4095 4503599627370495<CR><LF>	TIPO RO CORRECTAMENTE LEIDO	FORZA AL LECTOR A REALIZAR UNA LECTURA Y REGRESA	K0
	X	XW 2095 3414678905035996<CR><LF>	TIPO R/W CORRECTAMENTE LEIDO	AL ESTADO DE ESPERA	
		X<CR><LF>	SIN LECTURA		
		XI<CR><LF>	IDENTIFICACION INVALIDA		
GATE MODE	G	G<CR><LF>		COMPARA UNA LECTURA CORRECTA CON TODAS LAS	K0
		GR 4095 4503599627370495<CR><LF>	LECTURA CORRECTA	ALMACENADAS EN LA MEMORIA SI ES DIFERENTE LA	
		* MEMORY FULL<CR><LF>	EL LECTOR ENVIA ESTE MENSAJE CUANDO SE LLENA LA MEMORIA	ALMACENA Y LA ENVIA A LA PC	
STORE COMMAND	S	R 001 4095 4503599627370495<CR><LF>	ENVIA DOS LECTURAS ALMACENADAS	ENVIA TODOS LOS REGISTROS ALMACENADOS EN LA MEM	K0
		W 002 2095 3414678905035996<CR><LF>			
		S<CR><LF>			
READ MEMORY COMMAND	?	?	PIDE REGISTRO 909d=38Dhex, INF. SIEMPRE EN HEX	ENVIA EL CONTENIDO DE UN REGISRO EN ESPECIFICO	K0
	38D	<space>38D 0123456789ABCDEF00<CR><LF>	CARACT 1 A 16 REG, CARACT 17 Y 18 TIPO	ALMACENADO EN LA MEMORIA DEL LECTOR	
			00-R0, 01-R/W, 10 MPT		
RAM FILL COMMAND	R	R	LLENA LA MEMORIA CON EL NUMERO 5Ahex	LLENA LA MEMORIA CON ALGUN NUMERO PARA PROBARLA	K0
	5	5	LOS NUMEROS SE MANDAN SIEMPRE EN HEX	Y PONE EL CONTADOR DE ID EN 909dec=38Dhex	
	A	AR<CR><LF>			
NUMBER COMMAND	N	N 38D<CR><LF>	INFORMA QUE HAY 909dec=38Dhex REGISTROS EN MEM	PREGUNTA POR EL NUM. DE REGISROS ALM EN MEMORIA	K0
LINE FUNCTION	L	LW 2095 3414678905035996<CR><LF>	TRP R/W CORRECTAMENTE LEIDO	PONE AL LECTOR EN UN MODO DE LECTURA CONTINUA	KO
		LI<CR><LF>	IDENTIFICACION INVALIDA		
NORMAL MODE/ESCAPE	<Esc>	E<CR><LF>	TRP RO CORRECTAMENTE LEIDO	MODO DE LECTURA CONTINUA DE ALTA VELOCIDAD, CUANDO	K0
		R 4095 4503599627370495<CR><LF>		LEE CORRECTAMENTE UN TRP LO COMPARA CON EL ALMACENADO	
				EN EL BUFFER SI ES DIFERENTE LO ENVIA A LA PC Y LO ALMACENA	

Tabla 3.1.3. Protocolo *ASCII* del sistema *TIRIS* (continua).

PROTOCOLO ASCII SISTEMA TIRIS (RESUMEN DE COMANDOS)					
COMANDO	**CARÁCTER ASCII**	**Lector ----> uC(PC)**	**EJEMPLO**	**DESCRIPCION**	**MODO**
	uC(PC) ---> Lector				
PROGRAM COMMAND	P	P	PROGRAMA DATO EN HEXADECIMAL	PROGRAMA DATO Y LO VERIFICA	K0
	FEDCBA9876543210	0<CR><LF>	CONFIRMACION DE PROG. EXITOSA		
		1<CR><LF>	TRP. GRABO OTRO DATO O NO LO GRABO		
		2<CR><LF>	NO LEYO EL TRP. PARA CONFIRMAR		
K1 MODE	K	K	CAMBIA LECTOR A MODO K1	CAMBIA LECTOR A MODO K1 MULTIPAGE TRANSPONDERS	K1
	1	1<CR><LF>			
EXECUTE COMMAND	X	X	LECTURA DE PAGINA 1 SE ENVIA EN FORMATO HEX	REALIZA UNA SOLA LECTURA DE ALGUNA PAGINA DE UN MPT	K1
	O1	10M 01 4095 4503599627370495<CR><LF>	LECTURA CORRECTA EN FORMATO DECIMAL	EN EL RANGO DE 01 A 11hex O ENVIANDO 00 COMO PAG LEE	
		10M 01 FFFFFFFFFFFFFFFF<CR><LF>	LECTURA CORRECTA EN FORMATO HEXADECIMAL	RAPIDAMENTE LA PRIMERA PAGINA DE UN MPT O TRPs DEL	
		1R 4095 4503599627370495<CR><LF>	TRP RO CORRECTAMENTE LEIDO FORMATO DECIMAL	TIPO RO O R/W	
		10M 01 4095 4503599627370495<CR><LF>	TRP RO CORRECTAMENTE LIDO FORMATO HEXADECIMAL		
		1W FFFFFFFFFFFFFFFF<CR><LF>	TRP R/W CORRECTAMENTE LEIDO FORMATO HEX		
		1<CR><LF>	NO HUBO LECTURA		
		1I<CR><LF>	LECTURA DE IDENTIFICACION INVALIDA		
LINE FUNCTION	L	L	LECTURA DE PAGINA 1 SE ENVIA EN FORMATO HEX	PONE AL LECTOR EN UN MODO DE LECTURA CONTINUA	K1
	O1	<CR><LF>		DE LA PAGINA INDICADA	
		L10M 01 4095 4503599627370495<CR><LF>	LECTURA CORRECTA EN FORMATO DECIMAL		
		L1<CR><LF>	NO HUBO LECTURA		
		L1I<CR><LF>	LECTURA DE IDENTIFICACION INVALIDA		
NORMAL MODE	<Esc>	E	LECTURA DE PAGINA 5 SE ENVIA EN FORMATO HEX	MODO DE LECTURA CONTINUA DE ALTA VELOCIDAD, CUANDO	K1
	O5	<CR><LF>		LEE CORRECTAMENTE UN TRP LO COMPARA CON EL ALMACENADO	
		10M 05 0000 11237906738429913<CR><LF>		EN EL BUFFER SI ES DIFERENTE LO ENVIA A LA PC Y LO ALMACENA	
		1R 1024 1111111100101010<CR><LF>			
PROGRAM COMMAND	P	P	PROGRAMA EN PAGINA 5 HEXADECIMAL	PROGRAMA DATO Y LO VERIFICA	K1
	O5				
	432FFA6B22228FFA	10M 05 1074 4497462691794938<CR><LF>	CONFIRMACION DE PROG. EXITOSA		
		12<CR><LF>	PROG. FALLO		
LOCK PAGE	O	O	BLOQUEA PAGINA 05 HEX	BLOQUEA UNA PAGINA	K1
	O5	11M 05 1074 4497462691794938<CR><LF>	CONFIRMA DATO BLOQUEADO		
		1<CR><LF>	COMANDO NO EJECUTADO		

Tabla 3.1.3. Protocolo *ASCII* del sistema *TIRIS.*

III.2. MICROCONTROLADOR

Como se mencionó anteriormente el SDR-1000 está conformado por un sistema que proporciona la identificación del competidor (*TIRIS*), sin embargo, para los propósitos del sistema de cronometraje deportivo no es suficiente, ya que de acuerdo con los requerimientos técnicos establecidos es necesario que el SDR-1000 registre los tiempos de inicio y de terminación de cada competidor, sea confiable en la toma de datos al paso de un gran contingente de atletas, asegure el resguardo de la información y ofrezca la posibilidad de entregar resultados como lo indica la Federación de Atletismo hasta décimas de segundo. Lo anterior, se realizará utilizando como base un microcontrolador *(μC)* el cual será seleccionado conforme a los requerimientos de velocidad de procesamiento, memoria y comunicación que exijan los dispositivos electrónicos que se utilicen para satisfacer los requisitos ya mencionados, así como el sistema de identificación *TIRIS*

En la tabla 3.2.1 Se muestran los requerimientos generales que el microcontrolador deberá satisfacer de acuerdo con la información que se conoce del sistema *TIRIS* y de los requerimientos establecidos líneas atrás.

ETAPA	Requerido?
Interfaz para ingreso de datos	Si
Desplegado de información	Si
De almacenamiento de información	Si
De comunicación con *PC*	Si
De comunicación entre dispositivo de control	Si
Manejo de *RTC*	Si
Señales analógicas de entrada	Si (1)
Señales de analógicas de salida	No

Tabla 3.2.1. Requerimientos del microcontrolador.

Para la selección del microcontrolador a utilizar en el desarrollo del SDR-1000, se presenta la tabla 3.2.2 con un comparativo de microcontroladores de diferentes marcas.

MARCA	FAMILIA	MEMORIA DE PROGRAMA (máx)	MEMORIA EEPROM (máx)	MEMORIA RAM (máx)	I/O PINS (máx)	VCC (V)	COMUNICACIÓN	PRECIO (USD)
ATMEL	Atmega325	32k	1k	2k	54	1.8-5.5	SPI/UART	9.0
MICROCHIP	PIC18	32k		1.5k	52	2-5.5	SPI/I2C/USART	14.0
MOTOROLA	68HC11	32k	0.6k	1k	51	5	SCI/SPI	17.0
TEXAS INST	TMS370	32k	0.2k	1k	55	5	SCI/SPI	10.0
NEC	Upd78	32k		1k	52	1.8-5.5	2 UART/2 3-WIRE	8.0

Tabla 3.2.2. Comparativo de microcontroladores diferentes marcas.

De la tabla anterior se puede concluir que técnicamente los distintos microcontroladores ofrecen características muy similares, entonces, una base importante para la decisión es que se contaba con las herramientas de desarrollo para el *µC* de la marca *texas instruments*, así como la experiencia suficiente en su manejo, además el precio de este dispositivo como se puede apreciar es muy competitivo.

Bien, entonces se trabajará con el microcontrolador de *texas instruments* de la familia TMS370, de la cual se indican sus principales características:

III.2.1. LA FAMILIA DEL MICROCONTROLADOR TMS370

La familia TMS370 es una serie de microcontroladores *CMOS* de circuitos integrados de muy grande escala (*VLSI: Very Large Scale Integrated*), de 8 *bits* con una memoria *EEPROM* de almacenamiento y funciones de soporte periférico. Estos dispositivos ofrecen un desempeño superior en aplicaciones complejas y de control en tiempo real en ambientes de mucha demanda y cuentan con memorias *ROM* programables y *EPROM*. Además, se cuenta con una gran cantidad de opciones para elegir de acuerdo a las necesidades de diseño y capacidad económica.

Aplicaciones comunes

La familia TMS370 de dispositivos es una elección ideal para aplicaciones como:

Modos automáticos.- En sistemas de control de acondicionamiento de ambiente, control de cruce, sistemas de entrenamiento, instrumentación, sistemas de navegación, control de máquinas, etcétera.

Computadoras.- En teclados, control de interfaces periféricas, controladores de discos, terminales.

Industria.- Controladores de motores, controladores de termómetros, control de procesos, control de mediciones, instrumentación médica y sistemas de seguridad.

Telecomunicaciones.- *Modems*, teléfonos inteligentes, control de tarjetas telefónicas, telecopiadoras, tarjetas de crédito.

Elementos de la familia TMS370

La familia TMS370 tiene las siguientes características:

- *Software* compatible entre los elementos de la familia
- Tecnología *CMOS EPROM*, que contiene *EPROM* para reprogramar, memoria programable de una oportunidad para programar (*OTP: One Time Programmable*) para ser usada como prototipos y para volúmenes pequeños de producción, respectivamente.
- Tecnología *CMOS EEPROM*, que contiene programación de *EEPROM* con una alimentación de 5 volts
- Tecnología *A/D*, que convierte señales analógicas a digitales
- Registros *RAM* estáticos que ofrecen numerosas opciones de memoria
- Características de operación flexible:
  Reducción de potencia en estado de espera y en modos de interrupción

Temperatura de operación de -40º C a 85º C
Frecuencia de reloj de entrada de 2 MHz a 20 MHz
Voltaje de operación de 5V ± 10%

- Manejo de interrupción flexible para la versatilidad en el diseño:
  Dos niveles de interrupción programables
  Programación de detección de límites superiores y/o inferiores.
- Características para la integridad del sistema que incrementa la flexibilidad durante la fase del desarrollo del *software* con:
  Un oscilador de detección de fallas
  Modo de protección privilegiado
  Contador *watchdog*
- Puertos mapeados en memoria para direccionar fácilmente
- Una librería modular para cambiar rápidamente a configuraciones distintos
- Catorce modos de direccionamiento que usan ocho formatos, incluyendo:
  Aritmética de registro a registro
  Direccionamiento indirecto
  Saltos y accesos directos e indexados
- 250 mA de corriente típica estable a 25ºC

Componentes comunes en la arquitectura de la familia TMS370

La familia de microcontroladores TMS370 tienen los siguientes elementos de arquitectura, de acuerdo a la categoría del dispositivo.

CPU. La *CPU* de 8 *bits* de la TMS370 tiene un registro de estado, un registro de contador de programa, y el *(SP: stack pointer*). La *CPU* usa el archivo de registro como registros de trabajo, accesados en el *bus* interno a un ciclo de *bus*. Los dispositivos TMS370Cx5x permiten una expansión de memoria externa a través de los puertos A, B, C y D.

ÁREA DE REGISTROS. Estos registros se ubican al principio del mapa de memoria de la TMS370. Cuando este sector del mapa de memoria es usado como una *RAM* de propósito general. las instrucciones de acceso a registro, permiten el acceso a cualquiera de los primeros 256 registros en un ciclo de *bus*. Esta área de registros también puede ser empleada como *stack*.

MEMORIA *RAM*. Los otros módulos de memoria *RAM* que no están en el archivo de registro se encuentran en otra área de memoria. La familia TMS370 accesa esta *RAM* en dos ciclos de reloj del sistema.

MEMORIA *EEPROM*. Los módulos de *EEPROM* de datos proporcionan programabilidad y retención de datos en el modo apagado. Los módulos contienen 256 o 512 bytes de *EEPROM*. Esta memoria es útil para almacenar datos constantes y variables, sin cambios continuos, requeridos por el programa de aplicación. La *EEPROM* puede ser programada y borrada con programadores de *EEPROM* o con el mismo TMS370 bajo el control del programa.

MEMORIA DE PROGRAMA. La memoria de programa provee de alternativas para respaldar las necesidades de una aplicación. Los módulos de memoria de programa de acuerdo a la categoría del dispositivo contienen 2k, 4k, 8k, 16k ó 32k *bytes* de memoria. La memoria de programa de los, TMS370C6xx y TMS370C7xx es una *EPROM*. La *EPROM* puede ser programada, borrada y

reprogramada para prototipos (con cubierta cerámica). Los dispositivos *EPROM* que no tienen ventana (con cubierta de plástico) son dispositivos para programar sólo una vez (*OTP*). En los TMS370C0xx y TMS370C3xx, el programa de memoria es una *ROM* con máscara programada por el fabricante.

PUERTOS DE ENTRADA/SALIDA. Los TMX370Cx5x tienen cuatro puertos de 8 *bits*: el A, B, C, y (el puerto D de los dispositivos de 64 terminales es de 6 *bits*). Estos puertos pueden ser configurados para trabajar como líneas de datos, de control, y de dirección para memorias externas.

TIMER 1 y *TIMER 2*. Los *timer* 1 y 2 son de 16 *bits* y pueden ser configurados de las siguientes formas:

- Prescalador programable de 8 *bits* que determina las fuentes independientes de reloj para el *timer* de propósito general y el *timer watchdog*.
- *Timer* de evento de 16 *bits*, para llevar el conteo de transiciones.
- Acumulador de pulso de 16 *bits*, para medir el ancho de pulso a la entrada.
- Función de captura a la entrada de 16 *bits*, que guarda el valor del contador al existir una entrada externa.
- Dos registros de 16 *bits* de comparación que se disparan cuando el contador iguala los contenidos de un registro de comparación.
- Función de control de salida del mismo contenido de modulación por ancho de pulso (*PWM: Pulse Width Modulation*).

El resultado de estas operaciones puede ocasionar interrupciones a la *CPU*, encender bits de bandera, resetear el contador *timer*, cambiar una línea entrada/salida, generar salidas *PWM*. Estos *timers* pueden tener una resolución de 200 nseg operando con un cristal de 20 MHz.

TIMER WATCHDOG. El *timer watchdog* ayuda a asegurar la integridad del sistema. Puede ser programado para generar un *reset* de *hardware* al acabar su tiempo. Esta función genera un monitoreo del *software* para evitar pérdidas del programa. Si no se necesita como *watchdog*, éste también puede ser usado como *timer* de propósito general.

TIMER PROGRAMABLE DE ADQUISICIÓN Y CONTROL El módulo (PACT: *Programmable Adquisition and Control*) es un módulo de tiempo programable que utiliza la memoria *RAM* para guardar sus comandos, así como los valores del *timer*. Este módulo ofrece las siguientes opciones:

- Captura de entrada de 6 terminales, de los cuales 4 terminales pueden tener un prescalador
- Una terminal de entrada de captura puede manejar un contador de eventos de 8 *bits*
- 8 salidas para trabajar como *timers* manejables
- Capacidad de *timer* de 20 bits
- Interacción del contador de eventos y la actividad del *timer*
- 18 vectores de interrupción independientes para permitir un mejor servicio de eventos
- Un *watchdog* con un período selectivo de tiempo – fuera
- Una interfaz serial de comunicación (*SCI: Serial Communications Interface*), que trabaja con un Receptor Transmisor Universal Asíncrono (*UART: Universal Asynchronous Reciver Transmitter) full-duplex.*

Una vez configurado, el *PACT* ya no requiere de la supervisión del *CPU,* excepto para servicios de interrupción.

La interfaz *SCI* ofrece las siguientes características:

- Se puede programar para trabajar en modo asíncrono (156 kbits/seg) o modo síncrono (2.5 Mbits/seg).
- Comunicación *full-duplex* con doble *buffer* para Tx y Rx
- Formato programable con capacidad para revisión de errores

El módulo *SCI* programa y controla los factores tiempo, formato y protocolo de datos. La *CPU* no toma parte en la comunicación serial excepto para escribir datos transmitidos en los registros de la *SCI* y para leer los datos recibidos de los registros de la *SCI* cuando está interrumpido.

INTERFAZ PERIFÉRICA SERIAL (*SPI: Serial Peripheral Interface*). El módulo *SPI* facilita la comunicación entre la *CPU* y los dispositivos periféricos externos. Proporciona transmisión síncrona de datos a 2.5 Mbits/seg. Al igual que la *SCI*, es configurada por *software*. Después de eso, la *CPU* no toma parte en el tiempo, formato, y protocolo de los datos. También al igual que la *SCI*, la *CPU* lee y escribe en la memoria mapeada por los registros para recibir y transmitir datos. Una interrupción de la *SPI* avisa a la *CPU* cuando los datos recibidos están listos.

CONVERTIDOR ANALÓGICO DIGITAL *A/D* (*Analog to Digital).* El módulo del convertidor *A/D* de 8 *bits* desarrolla aproximaciones sucesivas. La fuente de referencia y el canal de entrada son selectivos. Las líneas de entrada que no se necesitan para la conversión *A/D* se pueden programar para funcionar como líneas de entradas digitales.

Diagrama a bloques del TMS370Cx5x

La familia TMS370 se basa en una arquitectura de registro a registro, la cual permite el acceso a un archivo de registro (de 256 *bits*) en un sólo ciclo de *bus*. El circuito integrado de memoria incluye la memoria de programa (*ROM* con máscara o *EPROM*), *RAM* estática, y *EEPROM* de datos.

Las funciones periféricas incluyen una interfaz de comunicación serial, una interfaz periférica serial, seis diferentes módulos *timers* y 55 *terminales* de entrada/salida digital. El diagrama a bloques del TMS370Cx5x se muestra en la fig. 3.2.1.

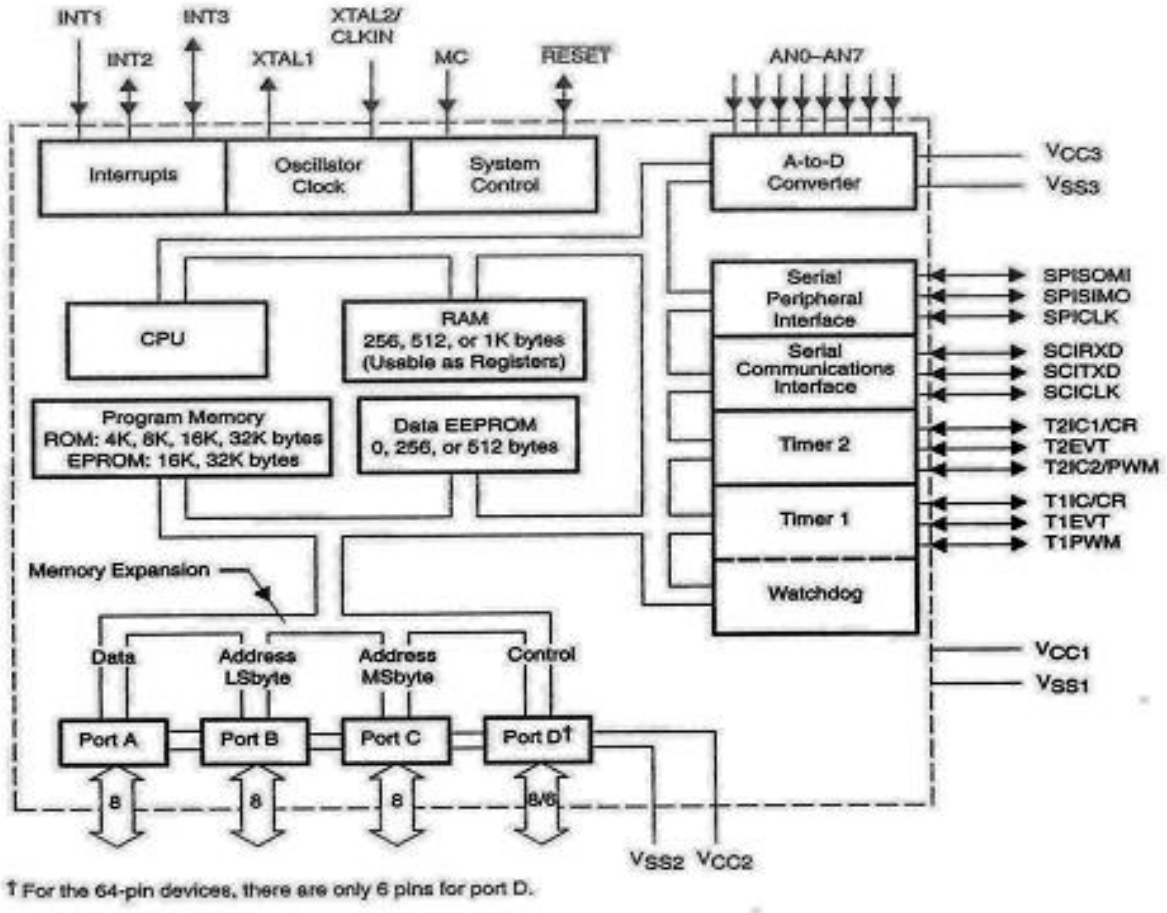


Fig. 3.2.1. Diagrama a bloques del TMS370Cx5x.

III.2.2. LENGUAJE DE PROGRAMACIÓN

Los lenguajes de programación son utilizados para implementar algoritmos, esto es, ejecutar secuencias o pasos para la solución de problemas. Un lenguaje de programación tiene que definir la acción de las trayectorias dinámicas de los algoritmos. No sólo debe dar un significado para especificar las operaciones básicas de una computadora, si no que también debe permitir al usuario conocer la secuencia a seguir de los pasos para resolver un problema en particular.

Lenguajes de alto nivel

Los lenguajes de programación de cómputo pueden ser clasificados en dos grandes grupos: Lenguajes de bajo nivel y lenguajes de alto nivel.

Los lenguajes de bajo nivel son parecidos al lenguaje de máquina y tienen una fuerte correspondencia entre las operaciones implementadas por el lenguaje y las operaciones del *hardware.* Los lenguajes de alto nivel, por otro lado, son parecidos a los lenguajes usados por el humano para

expresar problemas y algoritmos. Cada instrucción en un lenguaje de alto nivel puede ser equivalente a muchas instrucciones en un lenguaje de bajo nivel.

Podemos decir en forma concreta que un lenguaje de programación de alto nivel es, aquel que permite con una sola instrucción hacer una serie de operaciones.

El intervalo de operaciones directamente implementado por el *hardware* de una computadora es usualmente muy limitado, por lo que la codificación de algoritmos complejos usando sólo estas operaciones es un problema grande. La codificación puede ser tediosa y se pueden cometer errores. Ese es el obstáculo a vencer de los lenguajes de programación, especialmente de los lenguajes de alto nivel, dar al programador herramientas para que los problemas de codificación de los algoritmos sean realmente simplificados. El algoritmo, una vez codificado en el lenguaje de programación elegido, es traducido, normalmente sin la intervención humana, a una instrucción básica de máquina. Estas instrucciones son ejecutadas por el *Hardware*, y si todo está bien, el efecto deseado del algoritmo se produce.

En resumen, las ventajas de los lenguajes de alto nivel incluyen:

- Respaldo en las ideas de abstracción, ya que así el programador puede concentrarse en la solución del problema, y no en los detalles de bajo nivel en la representación de datos.
- Facilidad en su aprendizaje
- Rapidez en la solución de problemas
- Portabilidad y depuración simplificado, facilidad de modificación y mantenimiento
- Bajo costo de *software* y mayor confiabilidad

El cómo hacer que una computadora llevé a cabo la tarea de cálculo que deseamos, se logra mediante una serie de instrucciones que obedecen ciertas reglas y métodos utilizando por llamarle de alguna manera, un vocabulario propio.

De acuerdo al problema que se tenga (científico, técnico, administrativo) se empleará el lenguaje más apropiado.

El programa se debe realizar de acuerdo a las reglas del lenguaje utilizado para transmitir a la computadora las instrucciones que debe realizar exactamente, este conjunto de instrucciones alimenta a la computadora a través de la unidad de entrada, las cuales son traducidas por el compilador al lenguaje de máquina. Esto trae como consecuencia que para cada lenguaje debe haber un traductor o compilador que transforme nuestras instrucciones al lenguaje de máquina. El compilador es un programa suministrado generalmente por el fabricante del equipo de cómputo.

Compiladores

Un lenguaje de programación de alto nivel es traducido a un equivalente código de máquina por medio de un programa llamado compilador.

Los compiladores son programas largos y complejos, posiblemente envolviendo decenas o cientos de miles de líneas de código. Un compilador no se puede analizar considerándolo como una estructura monolítica, por lo que es necesario descomponerlo en módulos más simples.

El proceso de compilación se divide en dos subtareas:

- El análisis del programa fuente
- La síntesis del programa objeto

Lenguaje C

El lenguaje C es un lenguaje de programación de propósito general y ha sido utilizado en una gran gama de aplicaciones. Su aplicación más famosa fue en la creación del sistema operativo *UNIX*, donde el lenguaje C ha sido utilizado no solamente para la codificación del sistema operativo si no que también para sus utilidades asociadas.

Una característica poderosa del lenguaje C es el uso de variables con direcciones (apuntadores). Se pueden definir arreglos y estructuras. También, se pueden manejar uniones para que las variables contengan valores de diferente tipo.

El lenguaje C estándar es relativamente portable, a pesar de las operaciones de bajo nivel que maneja. Las funciones del lenguaje C pueden operar en la mayoría de las computadoras y sistemas operativos. El lenguaje C es un lenguaje popular, y debido a que está muy asociado con el sistema operativo *UNIX*, probablemente continuará siendo popular por algún tiempo.

LENGUAJE C++. Un interesante evolución del lenguaje C es lenguaje C++. El lenguaje C++ es una mejora del lenguaje C (con muy pocas excepciones).

El lenguaje C++ fue desarrollado a principios de los 80`s en los laboratorios *AT&T Bell*. El propósito era perfeccionar el lenguaje C, para que generará un lenguaje que, con estructuras incluidas, ayudará en el desarrollo de sistemas de *software* extensos, manteniendo las ventajas del C de control de bajo nivel y formato conciso.

Con el lenguaje C++ los programas en el lenguaje C necesitan muy poca modificación para correr en el ambiente C++. El lenguaje C++ es capaz de efectuar una compilación eficiente.

III.2.3. LENGUAJE ENSAMBLADOR DE LA FAMILIA TMS370

Un lenguaje ensamblador es un lenguaje simbólico que muestra de forma más legible los códigos de máquina binarios, por lo que se puede decir que el lenguaje ensamblador es un lenguaje intermedio entre un lenguaje de alto nivel y el código de máquina. La familia TMS370 consta de 73 instrucciones de ensamblador con una gran variedad de modos de direccionamiento.

El grupo de instrucciones de lenguaje ensamblador provee un método conveniente para programar al microcontrolador. Cada instrucción consiste de los siguientes elementos.

- Un mnemónico de función. El mnemónico especifica el tipo de operación del μC
- De cero a tres operandos. Los operandos indican donde el μC puede encontrar o almacenar datos durante la ejecución de una instrucción. El tipo y combinación de operandos determina el código de operación actual para una instrucción. La instrucción *MOV*, por ejemplo, tiene 27 opciones diferentes, cada una con su propio código de operación.

Una instrucción típica de dos operando se muestra a continuación:

MNEMÓNICO	FUENTE	DESTINO
ADD	#9,	R3

El ejemplo anterior se lee de la siguiente forma: Suma el valor 9 al contenido del registro número 3 y coloca el resultado en el mismo registro número 3. El registro destino sirve como una segunda fuente y como dirección final del resultado; los registros pueden ser manipulados sin utilizar registros intermedios.

El siguiente ejemplo muestra como la instrucción anterior puede aparecer en una línea de programa completa.

ETIQUETA	INSTRUCCIÓN	OPERANDOS	COMENTARIOS
XXXXX	ADD	#9,R3	;comentario

Debe existir al menos un espacio entre cada tipo de entrada. La entrada de la etiqueta y el comentario son opcionales.

Las 73 instrucciones son respaldadas por 246 códigos de operación que proveen un control flexible del flujo del programa del μC. Algunas instrucciones tales como *CLRC* y *TEST* A, comparten el mismo código de operación con el objetivo de ayudar al programador a entender todas las funciones de un código de operación. Algunas instrucciones utilizan 16 códigos de operación de 16 *bits*, dependiendo del tipo de instrucción y modo de direccionamiento empleado. El ensamblador forma varias manipulaciones de instrucciones de *bit* de otras instrucciones para simplificar la escritura y lectura del programa.

Modos de direccionamiento

Cada instrucción del lenguaje ensamblador de la familia TMS370 tiene de 0 a 3 operandos. Cada operando tiene un modo de direccionamiento. El modo de direccionamiento especifica la forma en que el μC calcula la dirección del dato requerido por la instrucción. La potencialidad de la familia TMS370 está basada por el gran número de modos de direccionamiento disponible.

Los 14 modos de direccionamiento se dividen en dos clases:

- General, el cual utiliza un intervalo de direccionamiento de 8 *bits*
- Extendido, que utiliza un intervalo de direccionamiento de 16 *bits*

La división de estas dos clases de direccionamiento se muestra en la tabla 3.2.3

	MODOS DE DIRECCIONAMIENTO
GENERAL	IMPLICADO
	REGISTRO
	PERIFÉRICO
	INMEDIATO
	CONTADOR DE PROGRAMA RELATIVO
	STACK POINTER RELATIVO
EXTENDIDO	DIRECTO ABSOLUTO
	INDEXADO ABSOLUTO
	INDIRECTO ABSOLUTO
	INDIRECTO DE OFFSET ABSOLUTO
	DIRECTO RELATIVO
	INDEXADO RELATIVO
	INDIRECTO RELATIVO
	INDIRECTO DE OFFSET RELATIVO

Tabla 3.2.3. Modos de direccionamiento.

En el apéndice A se incluye la lista del formato de todas las instrucciones, códigos de operación, longitudes de *bits*, ciclos/instrucción, operandos, registros de estados y una descripción de las instrucciones, mismas que pueden ser consultadas en el *Data Manual TMS370 Family* de *Texas instruments.*

CAPÍTULO IV

ORGANIZACIÓN DEL SISTEMA

El presente capítulo tiene como objetivo mostrar en diagramas de bloques la organización y planeación del sistema SDR-1000, en función de la selección de los dispositivos y las características general que se presentaron en el capítulo III.

IV.1 DIAGRAMA A BLOQUES DEL SISTEMA

Como ya se definió anteriormente, el sistema SDR-1000 estará conformado principalmente por un sistema de identificación por radio frecuencias y un dispositivo de control que permita asociar en tiempo real los tiempos de inicio y terminación de la competencia, una eficiencia y confiabilidad de lecturas ante el paso de un contingente de atletas, así como el resguardo de resultados y procesamiento de la información.

El sistema **SDR-1000** se presenta en 3 bloques principales, a continuación se describen las funciones que realizará cada uno.

- Sistema *TIRIS*

El primer bloque corresponde al sistema de identificación por radiofrecuencia *TIRIS*, el cual está conformado por los *transponders*, la unidad lectora y las antenas. En esta etapa la unidad lectora tiene la función de transmitir un pulso de energía, con objeto de generar un campo magnético en las antenas del sistema, que energicen al capacitor por medio de la inducción de la bobina que se encuentran dentro del *transponder*, al término de dicho pulso el *transponder* responderá transmitiendo su información a dicha unidad lectora.

- Controlador lógico dedicado (DLC)

El segundo bloque corresponde a unas unidades que se denominarán Controladores Lógicos Dedicados, que se diseñarán con base en un microcontrolador de la familia TMS370 *texas Instruments*. Los DLCs a través de su unidad lectora recibirán los registros de los *transponder* que hayan leído, éstos tendrán la función de etiquetar, almacenar, y transmitir la información a una unidad concentradora.

- Unidad concentradora (UC)

El tercer bloque estará conformado por la Unidad Concentradora, esta etapa será la responsable de adquirir la información recibida por cada DLC, así como la transmisión de datos al equipo de cómputo de manera ordenada y dosificada. El número máximo de DLCs que manejará cada unidad concentradora será de seis. La unidad concentradora también basará su diseño en un microcontrolador de la familia TMS370. Un equipo de cómputo conectado a la unidad concentradora a través del puerto serial RS-232 procesará la información.

En la fig. 4.1.1 se muestra el diagrama de bloques del sistema SDR-1000

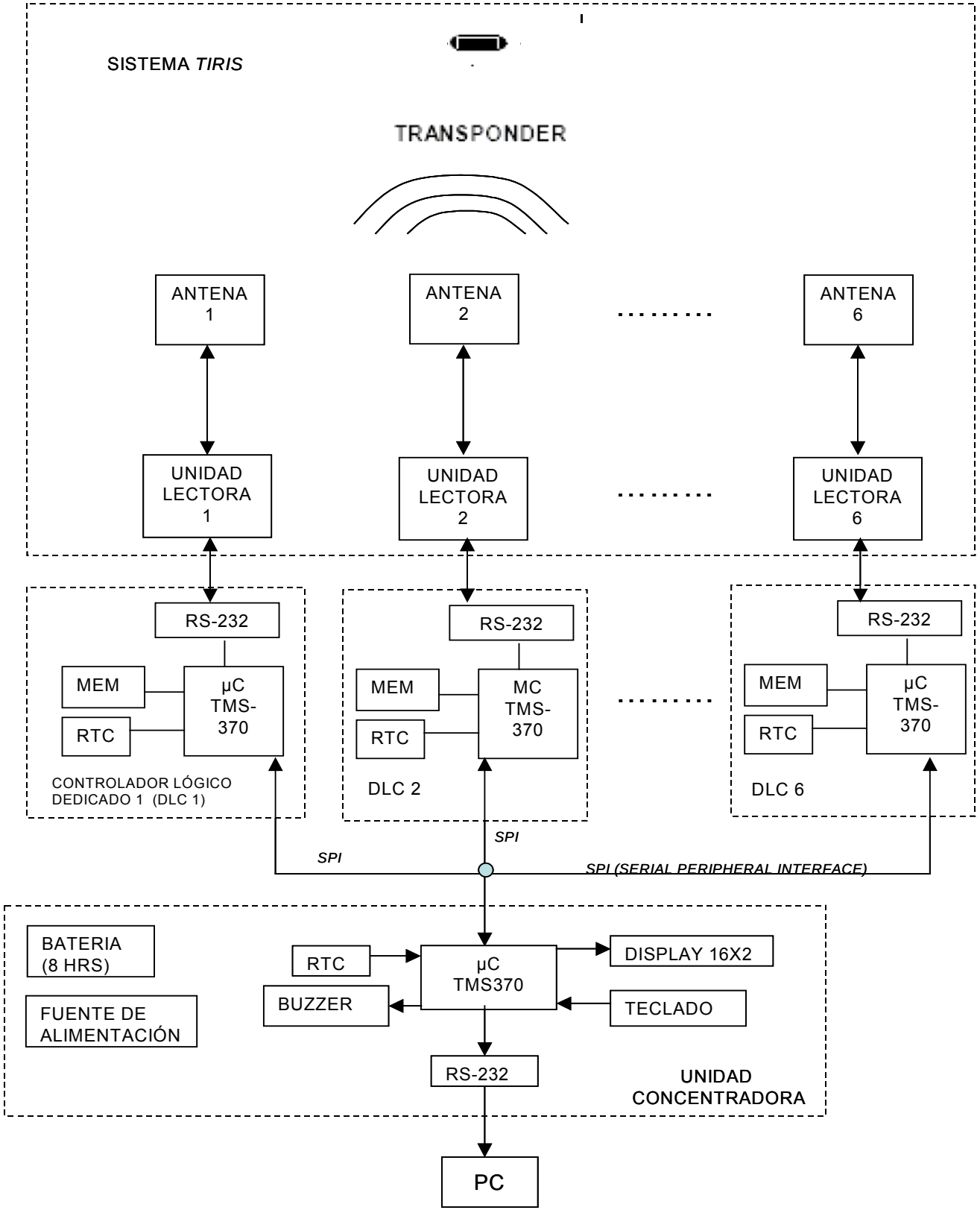


Fig. 4.1.1. Diagrama de bloques del sistema SDR-1000.

IV.2. SISTEMA *TIRIS*

En esta sección se describirán las características de funcionamiento del Sistema *TIRIS* de la firma t*exas instruments*.

En la fig. 4.2.1 se muestra el diagrama de bloques correspondiente al sistema *TIRIS*

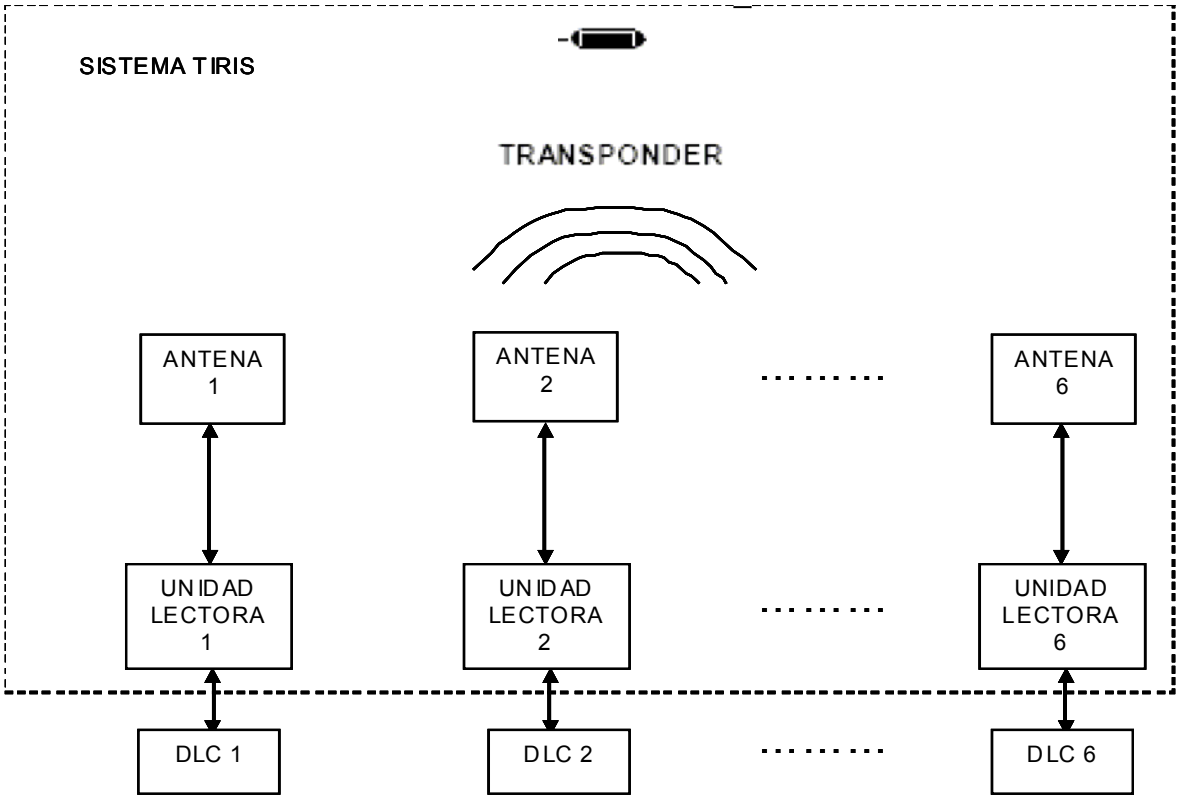


Fig. 4.2.1. Diagrama de bloques del sistema *TIRIS.*

Transponder

El sistema *TIRIS* opera en conjunto con un pequeño dispositivo denominado *transponder* (transmisor + receptor), el cual puede ser adherido o incorporado en un objeto a identificar. Los *transponders* operan bajo un principio en el que por un efecto inductivo generado por las antenas del sistema, cargan un capacitor por medio de la inducción de una bobina que se encuentran dentro del mismo *transponder*, convirtiéndose por unos instantes en una fuente de energía que alimenta a su circuito de transmisión (el tiempo aproximado de carga del capacitor es de 50 milisegundos), de esa forma el *transpoder* no requiere para su funcionamiento una fuente de poder externa, como podría ser una batería. El alcance máximo como ya se definió anteriormente es de 100 cms.

El *transponder* cargado, responde transmitiendo los datos almacenados en su memoria mediante una señal de respuesta a través de las antenas que en su momento lo energizaron. En el caso de *transponders* de sólo lectura (*read only*), los datos vienen programados de fábrica con un código de 64 *bits*. En los de lectura-escritura el usuario puede almacenar datos en la memoria del *transponder*. El ciclo total de lectura del *transponder* es menor a 100 milisegundos permitiendo obtener más de 10 lecturas por segundo.

Unidad lectora

La unidad lectora es una parte esencial del sistema *TIRIS*, dado que es el componente que cuenta con las funciones de Radio Frecuencia, control y sintonización requeridas para la comunicación con los *transponders.* Sus funciones son las siguientes:

Módulo de control (CTL).- Es la interfaz entre el módulo de radiofrecuencia (RFM) y el dispositivo de control (microcontrolador). Controla las funciones de transmisión y recepción del módulo RFM de acuerdo con los comandos enviados por el microcontrolador para transmitir señales y recibir datos de los *transponders*. Decodifica las señales de RF recibidas para obtener el número de identificación de los *transponders*, los valida y maneja los protocolos de comunicación de las interfaces seriales con que cuenta para comunicarse con algún controlador.

Módulo de RF (RFM).- El módulo de radio frecuencia contiene todas las funciones analógicas (oscilador, demodulador, amplificador, etapa PWM) de la unidad lectora necesarias para enviar la señal para energizar vía las antenas los *transpoders* y enviar los datos recibidos al módulo de control.

Sintonizador dinámico (DAT).- El autosintonizador tiene la función de sintonizar la antena a los parámetros requeridos por el módulo de RF y la conserva sintonizada durante la operación.

En la fig. 4.2.2 se muestra un diagrama de bloques, con las etapas internas que conforman la unidad lectora.

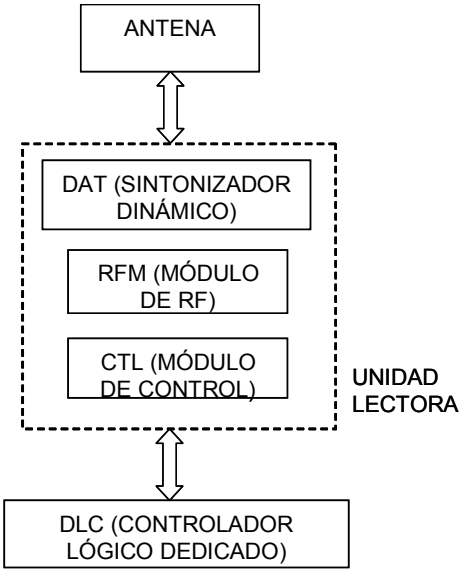


Fig. 4.2.2. Diagrama de bloques Unidad Lectora.

Antena

Las antenas del sistema *TIRIS* tiene doble función, la primera es generar un campo magnético que energice al capacitor por medio de la inducción de la bobina que se encuentran dentro del *transponder*, para luego, al término de dicho pulso cuando el *transponder* envíe su información, la antena funcione como un elemento receptor.

IV.3. CONTROLADOR LÓGICO DEDICADO

A continuación se muestran en la figura 4.3.1 los componentes que integrarán cada DLC para llevar a cabo las funciones de etiquetar, almacenar y transmitir la información recibida por las unidades lectoras.

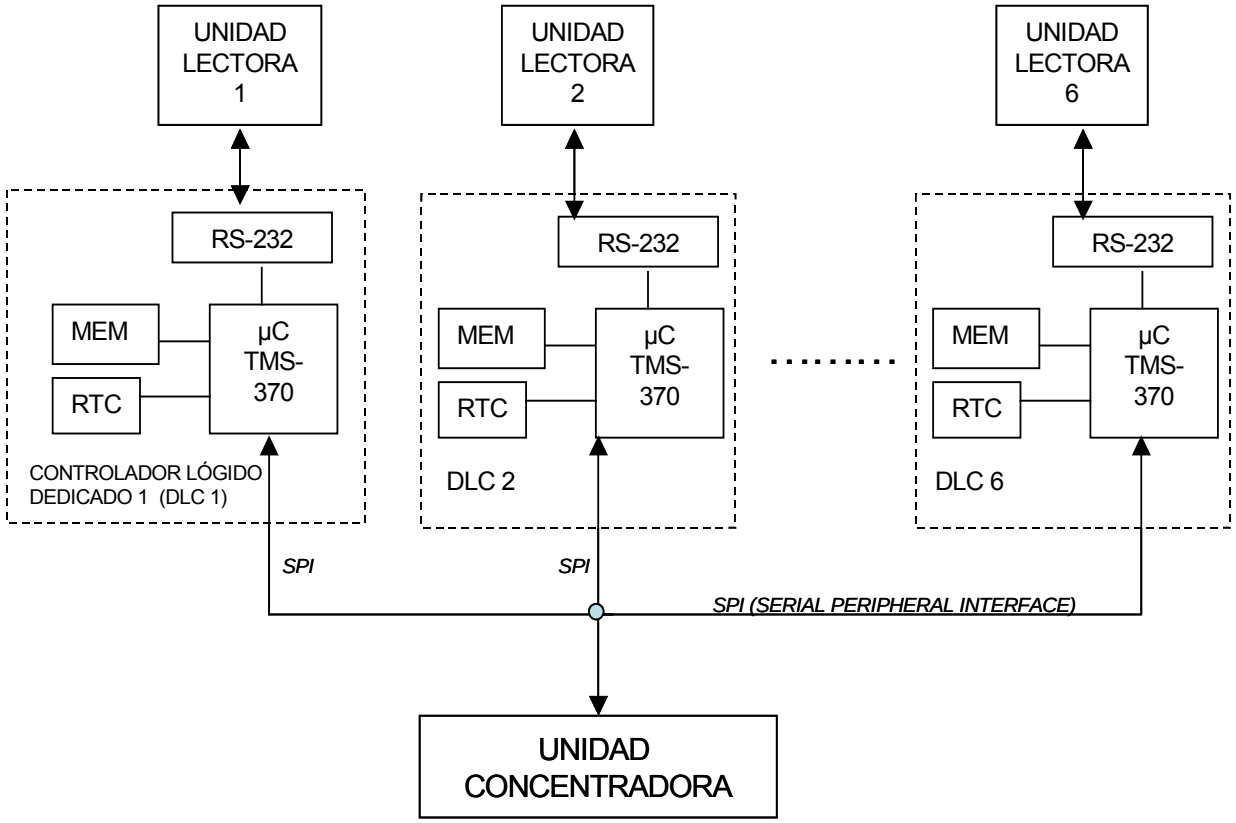


Fig. 4.3.1. Diagrama de bloques
Controlador Lógico Dedicado (DLC).

El diagrama de bloques muestra que el DLC está conformado por un microcontrolador TMS370, memoria externa, un circuito de reloj y un puerto serial para la comunicación con las unidades lectoras. Es importante mencionar que cada DLC maneja una unidad lectora y su correspondiente antena.

Se presenta una breve descripción de las funciones que realiza cada componente.

Microcontrolador (μC)

El microcontrolador de cada DLC será el encargado de procesar, administrar y transmitir la información de los *transponders* que recibirá de su unidad lectora, así como del control de los dispositivos periféricos como la memoria y el puerto serial.

Reloj de tiempo real (*RTC*)

Cada DLC contará con un circuito de reloj de tiempo real, de tal forma, que cada vez que una unidad lectora transmita a su DLC la lectura de un *transponder* que haya sido captado en el radio de cobertura de su antena, se le asociará el tiempo que en ese instante marque el *RTC*, etiquetando la información contenida en el *transponder* con el tiempo asociado. Lo anterior hará que el sistema de cronometraje deportivo SDR-1000 sea altamente confiable en la toma de lecturas. Los *RTC* serán sincronizados a través de otro *RTC* que se encuentra en la Unidad Concentradora.

Memoria externa

En la memoria de cada DLC, se almacenará la información contenida en cada *transponder* con su tiempo asociado, resguardándola hasta que se le requiera. De esta manera, el sistema de cronometraje deportivo será altamente confiable, ya que será capaz de almacenar cada lectura de cada una de las unidades lectoras.

Interfaz RS232

A través de la interfaz RS232 el microcontrolador de cada DLC se comunicará con su unidad lectora de *TIRIS*.

IV.4. UNIDAD CONCENTRADORA

La unidad concentradora será la responsable de concentrar la información recibida por cada DLC, así como la transmisión de datos al equipo de cómputo de manera ordenada y dosificada. El número máximo de DLCs que manejará cada unidad concentradora será de seis. La unidad concentradora también basará su diseño en un microcontrolador de la familia TMS370 de la firma *Texas Instruments*.

En la fig. 4.4.1 se muestra el diagrama de bloques correspondiente a la Unidad Concentradora

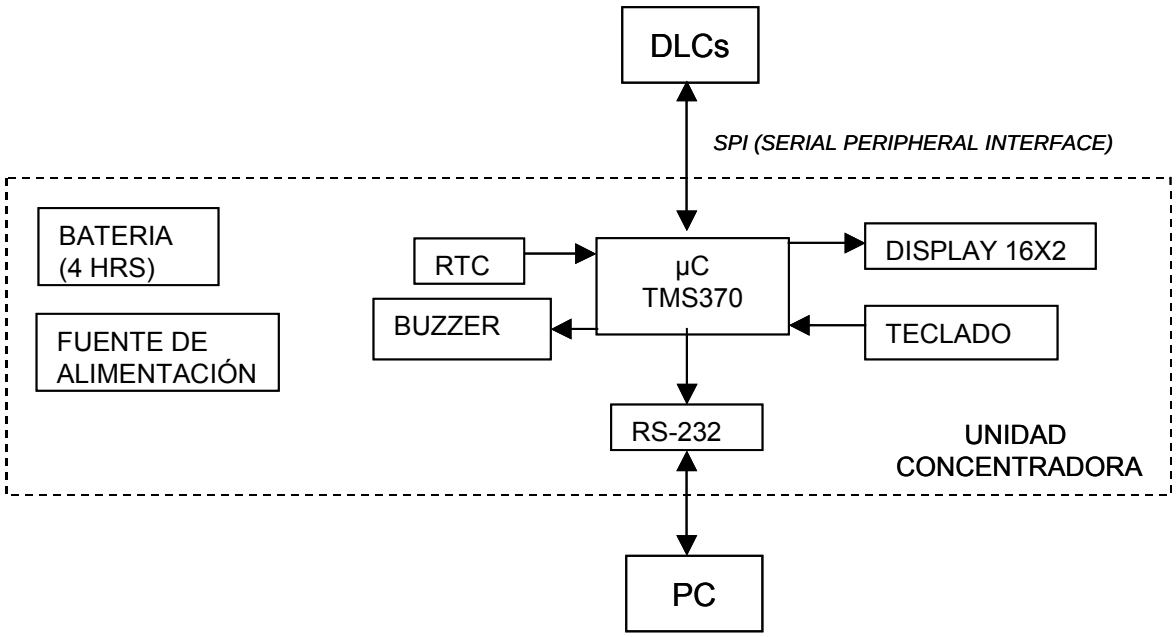


Fig. 4.4.1. Diagrama de bloques
Unidad Concentradora.

En el diagrama de bloques, se muestra que la unidad concentradora está conformada por un microcontrolador, circuito de reloj de tiempo real, *display*, teclado, *buzzer*, batería, fuentes de alimentación y un puerto serial RS-232 para la comunicación con una computadora personal. El módulo SPI será utilizado para comunicar a los microcontroladores de los DLCs y la UC.

Microcontrolador (*µC*)

El microcontrolador de la unidad concentradora, será el encargado de procesar, administrar y transmitir toda la información recibida de los DLCs, así como del control de los dispositivos periféricos como el puerto serial, *display*, *buzzer*, teclado, etcétera

Reloj de tiempo real (*RTC*)

Se contará con un circuito de reloj de tiempo real, que será utilizado para inicializar el sistema al arranque de una competencia, todos y cada uno de los circuitos de reloj de los DLC`s serán sincronizados con el reloj de la unidad concentradora. El *RTC* de la UC se puede sincronizar a través de la *PC*

Teclado

El teclado, será el medio por cual se introduce la información a la Unidad Concentradora, servirá para inicializar y configurar el sistema.

Display

El *display* será el dispositivo con el cual se tenga acceso visual a los diferentes comandos para inicializar y configurar el sistema.

Buzzer

Es un dispositivo que permite generar señales audibles en el caso de presentarse alguna condición que debe ser atendida por el usuario.

Fuente de alimentación

El sistema contará con la posibilidad de alimentarse en operación normal de una toma de corriente eléctrica en 127 VCA, cuando el equipo está alimentado con dicho voltaje de corriente alterna, un circuito recarga la batería.

Batería

El sistema contará con una batería para una autonomía de 4 hrs, para los casos donde no se cuente con una toma eléctrica en 127 VCA. La batería será del tipo recargable.

Interfaz RS232

Un equipo de cómputo conectado a la unidad concentradora, a través del puerto serial RS-232, procesará la información para generar un archivo en formato txt, mismo que a través de hojas de cálculo o *software* en aplicaciones deportivas, emitirá los reportes correspondientes con los datos de los atletas que concluyeron la competencia y clasificado por categorías.

CAPÍTULO V

DISEÑO DEL SISTEMA

En el presente capítulo, se describe el proceso de selección y diseño de los componentes del sistema *TIRIS*, así como del microcontrolador familia TMS370 y los circuitos periféricos para la unidad concentradora y los DLCs.

V.1. SELECCIÓN Y DISEÑO DE COMPONENTES DEL SISTEMA *TIRIS*

El sistema *TIRIS* de la firma *texas instruments* como ya se mencionó está conformado por 3 componentes: El *transponder*, la unidad lectora y las antenas. A continuación con base en la información técnica de ese sistema de *RFID* se hará la selección especifica del *transponder* y la unidad lectora, así como del diseño de la antena.

Transponder

Requisitos para su selección:

- Tamaño: El menor tamaño posible de tal forma que no represente para el atleta ningún tipo de interferencia o peso adicional.
- Alcance: El mayor alcance posible para garantizar su registro por la unidad lectora.

La tabla 5.1.1 se utiliza como guía para seleccionar el *transponder*.

Transponder / Applications Selection Guide

	134.2 kHz Transponder											13.56 MHz Transponder							LUHF
Transponder Types / Applications	Glass 50 mm	Glass 32 mm	Glass 23 mm	Keyring Tag	Wedge	Disk	Card	Slimline Disk	Cylinder	Mount on Metal	3D Analog Front End		Tag-it Square Inlay	Tag-it Rectangular Inlay	Tag-it Rectangle-mini Inlay	Tag-it Strip Inlay	Keyfinh Tag	Vicinity Badge		Vehicle Tag
Access Control				X			X	X	X	X	X			X			X	X		
Airline Baggage ID														X						
Automotive Security		X	X	X							X									
Document Tracking													X	X	X	X				
Door Locks			X	X	X		X										X	X		
Express Parcel ID													X	X	X	X				
Livestock ID		X	X			X														
Logistics Supply Chain																				
Pallets/Crates		X	X			X			X	X										
Containers/Totes		X	X			X		X	X	X										
Forklifts/Trucks									X	X										
Library Materials													X	X	X	X				
Gas Bottles/Kegs		X	X		X															
Payment & Loyalty				X			X										X	X		X
Product Authentication													X	X	X	X				
Sports Timing		X	X																	
Ticketing													X	X	X	X				
Waste Management		X	X																	

Tabla 5.1.1. Guía para la selección del *transponder*.

Por nuestra aplicación, de la guía de selección nos referiremos a los que se recomiendan para el registro de tiempos deportivos (*sports timing*) los cuales trabajan en una frecuencia de 134 kHz. Se muestran dos tipos de *transponder* que son diferentes en sus dimensiones, para seleccionar el más adecuado se consultó la hoja técnica de cada uno, mismas que se adjuntan en el apéndice A. Sus características principales se muestran en la tabla 5.1.2

Longitud	**Frecuencia**	**Alcance**
32 mm	134 kHz	100 cms
23 mm	134 kHz	60 cms

Tabla 5.1.2. Características del *transponder* para aplicaciones deportivas.

Bien, las consideraciones son la longitud del dispositivo y su alcance. Aunque lo recomendable para el atleta es que el *transponder* sea lo más pequeño posible, en términos prácticos los 9 mm que hay de diferencia entre uno y otro son despreciables, no así el alcance, ya que como se estableció en los requisitos técnicos de selección garantizar la detección del *transpoder* por parte de la unidad lectora es de consideración muy importante.

Transponder: Longitud: 32 mm, frecuencia: 134 kHz, alcance: 100 cms

Ahora, existen dos tipos de *transponders*: Sólo lectura y lectura-escritura. En la tabla 5.1.3 se muestran sus características.

Modo	**longitud**	**Alcance**	**Costo**
			Usd
Sólo lectura	32 mm	100 cms	3.5
Lectura-escritura	32 mm	100 cms	4.5

Tabla 5.1.3. *Transponder* modo de lectura

Como se ha explicado anteriormente, el sistema de cronometraje SDR-1000 en términos generales registra y asocia un tiempo cuando detecta un *transponder*. Por tanto, se requiere para una plena identificación del atleta que cada *transponder* cuente con un código propio. *Texas instrumens* garantiza que no hay la posibilidad de repetición en el número de identificación precargado de los *transponders* de sólo lectura, siendo entonces útiles para nuestra aplicación, adicionalmente y no menos importante como se aprecia en la tabla 5.1.3, la diferencia en precio es de consideración, siendo como se recordará que una de las premisas para el desarrollo del SDR-1000 precisamente fue que se obtenga un equipo con los costos más bajos posible.

Selección del *transponder*: **Longitud: 32 mm, frecuencia: 134 kHz, alcance: 100 cms y modo: sólo lectura** (tabla 5.1.1)

Fig. 5.1.1. Imagen del *transponder* de 32 mm..

Como protección para el *transponder*, se encapsulará en un gabinete de material ABS, el cuál presenta una altísima resistencia, requerida en esta aplicación. El gabinete fue diseñado con características ergonómicas que permitan sujetarlo de forma cómoda al zapato tenis del corredor o con una cinta elástica a su tobillo. En la Fig. 5.1.2 se muestra el gabinete de plástico

.

Fig. 5.1.2 Gabinete de plástico.

Unidad lectora

Criterios para su selección:

- Compatible para la lectura de *transponder* que trabajen en una frecuencia de 134.2 kHz.
- Deberá contar con Interfaz RS-232 para realizar la comunicación con los DLCs.
- Permitir la sincronización para configuraciones donde se utilice más de una unidad lectora.
- De fácil sintonía en la etapa de RF, para calibrar esta etapa en los diferentes tipos de superficie en que se instalará el equipo como son: concreto armado, tierra, pasto, etcétera.

La tabla 5.1.3 muestra una guía para la selección de la compatibilidad entre la unidad lectora y el *transponder.*

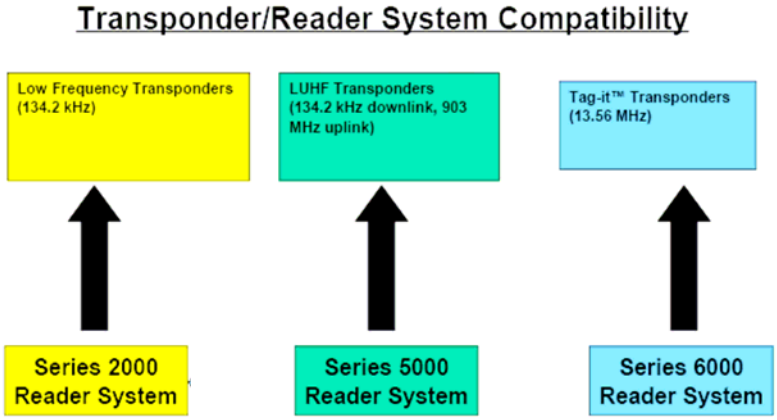


Fig. 5.1.3. Guía para la selección de la compatibilidad lectora/*transponder.*

De acuerdo con la información anterior, el lector compatible con los *transponder* que trabajan en la frecuencia de 134.2 kHz son los que corresponden a la serie 2000.

La tabla 5.1.4. muestra el sistema de configuración para la serie 2000.

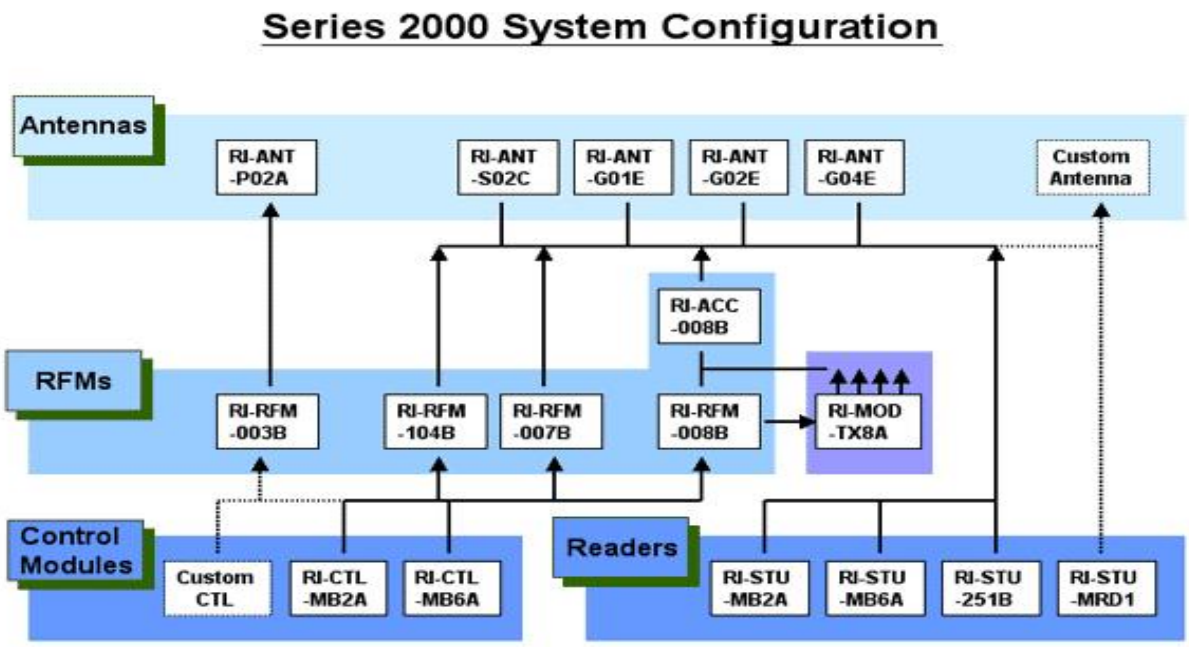


Fig. 5.1.4. Sistema de configuración serie 2000.

La configuración del sistema 2000 mostrado en la tabla 5.1.4, nos indica que las unidades lectoras corresponden a los modelos: RI-STU-MB2A, RI-STU-MB6A, RI-STU-251B y RU-STU-MRD1.

Se consultaron las hojas técnicas de cada una de ellas en las cuales se especifica que cuentan con todas las funciones de RF y control, puerto RS-232 y sincronización; sin embargo, sólo el lector RI-STU-251B cuenta con función de *Auto Tuning* que permite al lector sintonizarse de forma automática, función que como se mencionó es fundamental en el sistema de cronometraje deportivo, por lo que se seleccionó esta unidad para el sistema. En la fig. 5.1.5 se muestra una imagen de la unidad lectora seleccionada.

Fig. 5.1.5. Unidad Lectora RI-STU-251B.

Antenas

Criterios para su diseño:

- Su posición no deberá interferir con el paso de los corredores
- Su instalación deberá ser fácil, sin la necesidad de soportes o estructuras para su sujeción
- Sus dimensiones y peso deberán permitir que sean transportables
- Su radio de cobertura deberá ser el mayor posible para evitar una gran cantidad de antenas
- El patrón de radiación debe cubrir el movimiento del *transponder*, el cuál al ser sujetado en el tenis o en el tobillo pasará de una posición vertical a una horizontal y viceversa en el movimiento natural del pie.

El tipo de antena que se empleará y que cumple con los requerimientos planteados es la denominada Antena Tipo Cuadro o "*Loop*", que es una antena formada por una bobina de una o varias espiras enrolladas en un cuadro. Su funcionamiento es bi-direccional, tiene la posibilidad de instalarse rápidamente y a baja altura sin perder sus propiedades de radiación y portabilidad.

El patrón de radiación de estas antenas se presenta en la siguiente figura 5.1.6, como se puede observar el *transponder* podrá ser leído en cualquier posición, requerimiento obligado ya que el atleta portador de este dispositivo al paso por las antenas no tendrá que realizar acción que afecte el desarrollo natural de la competencia.

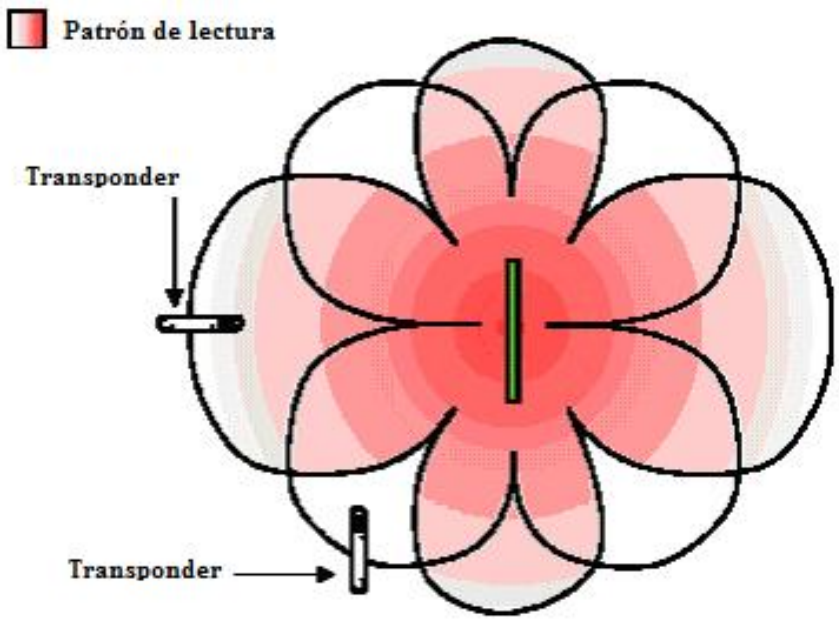


Fig. 5.1.6. Patrón de radiación de antena tipo *loop.*

Los parámetros establecidos por t*exas instruments* en el manual de especificaciones técnicas de la unidad lectora *Reader Reference Guide* para el diseño de la antena son (Fig. 5.1.7):

Parámetro	Mínimo	Máximo
Voltaje de resonancia	-	380V(pico)
Inductancia	26.0μH	27.9μH
Factor de calidad (Q)	40	350

Fig. 5.1.7. Parámetros para el diseño de la antena.

La impedancia de este tipo de antenas es de 100 Ω, el requerido por la unidad lectora es de 50 Ω, el sintonizador dinámico del lector se encargará de realizar su acoplamiento.

Las dimensiones de las antenas fueron definidas considerando 2 parámetros principales:

- El espacio promedio que abarca una persona con el movimiento de brazos y piernas en una competencia atlética.
- Las antenas puedan ser transportables y cargadas por una sola persona.

Las dimensiones obtenidas bajo esos criterios son: 60 cms de ancho x 90 cms de largo.

Por recomendación de *texas instruments* el alambre que se deberá emplear en la construcción de la bobina deberá ser de un material libre de oxigeno con 0.2 cm de radio, lo que ayudará a maximizar la eficiencia de la antena.

La ecuación 5.1 define el comportamiento de este tipo de antenas:

$$L=N^2(\mu/\pi)(-2(w+h)+(h^2+w^2)^{1/2}-h\ln((h+(h^2+w^2)^{1/2})/w) -w\ln((w+(h^2+w^2)^{1/2})/h)+h\ln(2h/a) +w\ln(2w/a)) \quad \text{ec (5.1)}$$

donde:

N = número de vueltas
a = radio del alambre = 0.2 cm
μ = permeabilidad relativa del medio = 1
L = 26 μH - 27.9 μH
w = 90 cms
h = 60 cms

Sustituyendo los datos en la ecuación anterior se obtienen el número de vueltas con que debe ser construida la antena:

N = 3

Entonces, la cantidad de cable a utilizar es: (90+90+60+60) cms x 3 = **900 cms**

La bobina de la antena se encapsulará en un material plástico y antiderrapante (tipo tapete), a fin de que los corredores puedan pasar sobre ella sin ningún tipo de riesgo o interferencia, la presentación final de la antena encapsulada se presenta en la Figura 5.1.8

Fig. 5.1.8 Antena encapsulada.

V.2 DISEÑO DEL CONTROLADOR LÓGICO DEDICADO (DLC)

Como se ha mencionado en capítulos anteriores, los Controladores Lógicos Dedicados, se diseñarán con base en un microcontrolador de la familia TMS370 de la firma t*exas instrument*. Los DLC tendrán la función de etiquetar, almacenar y transmitir la información recibida de las unidades lectoras a una unidad concentradora. Por cada antena que se instale se requiere un DLC.

El DLC estará conformado principalmente por: un microcontrolador, circuito de memoria externa, circuitos de comunicación y un reloj de tiempo real.

A continuación se presentan los criterios de diseño para el microcontrolador y los circuitos periféricos que conforman cada DLC los cuales son los siguientes: Circuitos de memoria externa, de comunicación, del reloj de tiempo real y de apoyo, así como la integración de los citados circuitos.

V.2.1. MICROCONTROLADOR

Como se estableció en capítulos anteriores el microcontrolador que se utilizará para el diseño pertenece a la familia de TMS370 de *texas instruments*. Para la selección del ***µC*** especificó se tomarán en cuenta los siguientes requerimientos:

- Un puerto serie para la comunicación con la unidad concentradora.
- Un puerto serie para la comunicación con la unidad lectora.
- Terminales de entrada y salida, para la lectura del teclado, desplegado de información en la pantalla y el control de circuitos periféricos.
- Permita el manejo de memoria externa.

Del manual *TMS370Cx5x 8-BIT MICROCONTROLLER*, se buscó el ***µC*** que cumpliera con los requerimientos establecidos, adicionalmente por la aplicación se debería considerar que el ***µC*** tuviera una capacidad grande de memoria para almacenar el programa. Como se muestra en la tabla 5.2.1 el ***µC*** seleccionado es el TMS370C758.

µC	**Memoria de programa (*bytes*)**		**Memoria de datos (*bytes*)**		**Expansión de memoria (*bytes*)**	**Interfaz**	**Módulos de reloj**
	ROM	**EPROM**	**EEPROM**	**RAM**			
TMS370C642	----	8K	----	256	----	SCI	T1/T2
TMS370C710	----	4K	256	128	----	SCI	T1
TMS370C722	----	8K	256	256	----	SPI/SCI	T1
TMS370C732	----	8K	256	256	----	PACT/SCI	PACT
TMS370C742	----	8K	256	256	----	SCI	T1/T2
TMS370C756	----	16K	512	512	----	SPI/SCI	T1/T2
TMS370C758	----	32K	256	1K	----	SPI/SCI	T1/T2

µC	**Interrupciones/reset**			**I/O *pins***	**No. de *pins***
	Externos	**Vectores**	**fuentes**		
TMS370C642	4	9	22	32/36	40 DIP/SDIP I 44 PLCC
TMS370C710	4	6	13	22	28 DIP/PLCC
TMS370C722	4	8	16	34	40 DIP/SDIP I 44 PLCC
TMS370C732	4	23	25	36	44 PLCC
TMS370C742	4	9	22	32/36	40 DIP/SDIP I 44 PLCC
TMS370C756	4	10	23	55/53	68 PLCC I 64 SDIP
TMS370C758	4	10	23	55/53	68 PLCC I 64 SDIP

Fig. 5.2.1 Datos para selección del microcontrolador

Para lograr la operación y adecuado funcionamiento del TMS370C758 se requieren las siguientes consideraciones:

- Frecuencia del cristal.

Para elegir la frecuencia del cristal se tomó en cuenta lo siguiente:

En promedio el TMS370C758 ejecuta una instrucción entre 10 y 20 ciclos, asimismo la información técnica de ese ***µC*** señala que un ciclo de máquina es igual a 4/clkin

Por la experiencia en aplicaciones similares se eligió un cristal con el cual se pudieran lograr tiempos de ejecución por instrucción de 5 μseg, el cristal seleccionado corresponde a una frecuencia de 20 MHz.

Tiempo de ejecución = ciclo de máquina / frecuencia del cristal

Tiempo de ejecución = 4 / 20 MHz = 0.2 μseg

La velocidad de procesamiento para el promedio en la ejecución de instrucciones sería de:

Velocidad = 10 ciclos x 0.2 μseg = 2 μseg
Velocidad = 20 ciclos x 0.2 μseg = 4 μseg

Por lo que se cumple con lo solicitado.

Ahora para el funcionamiento del ***μC*** se debe:

- Polarizar al dispositivo con +5 VDC.

- Fijar un nivel de 0 V en la terminal 6, para que el dispositivo funcione en el modo simple, en el cual todo el programa reside en la memoria interna del ***μC.***

- El cristal ya seleccionado de 20 MHz requiere de dos capacitores conectados a tierra de un valor recomendado por el fabricante de 15 pF.

Un circuito para reinicializar externamente el ***μC***. La señal de salida este circuito se debe conectar a la terminal 53 (*reset*).

Un pulso de 0 V inicia la secuencia de *reset* del sistema. El ***μC*** puede detectar pulsos de *reset* de duración corta de algunos nanosegundos, sin embargo, se necesita de un pulso de 0 V de por lo menos un ciclo de reloj, para garantizar su inicialización. El sistema se encuentra en *reset* hasta que el *pin* de *reset* se desactiva, es decir, vuelve a un nivel de 5 V. La señal de *reset* debe permanecer el tiempo necesario para inicializar los diferentes periféricos existentes. Para el caso del presente diseño, el circuito que necesita de un mayor tiempo para su reinicialización es el display, requiriendo de un mínimo de 4.2 ms. El circuito empleado para realizar la acción de *reset* se muestra después de la obtención de los valores de los elementos de *reset,* los cuales se calculan a partir de la ecuación 5.2.1

$$\tau = RC \qquad \text{ec. 5.2.1}$$

Por lo que:

$$C = \tau/R$$

Fijando el valor de $R_1 = 10\ k\Omega$ se calcula el valor de C.

$$C_4 = 4.2\ ms/10\ k\Omega$$

$$C_4 = .42\ \mu F$$

Por tanto, el valor comercial empleado es $C_4 = .47\ \mu F$

Debido a que la corriente máxima en la terminal de *reset* es de 2 mA, se requiere una resistencia que limite la corriente, el valor de esta resistencia se determina con la siguiente ecuación:

$$R = V/I \qquad \text{ec. 5.2.2}$$

= (5 V)/(2 mA)
= 2.5 kΩ

Por lo tanto R = 2.7 kΩ

En el circuito de *reset* se incluye un diodo D1, que permite al capacitor descargarse rápidamente durante una despolarización repentina del sistema. El diagrama eléctrico del ***µC*** se muestra en la siguiente figura 5.2.1:

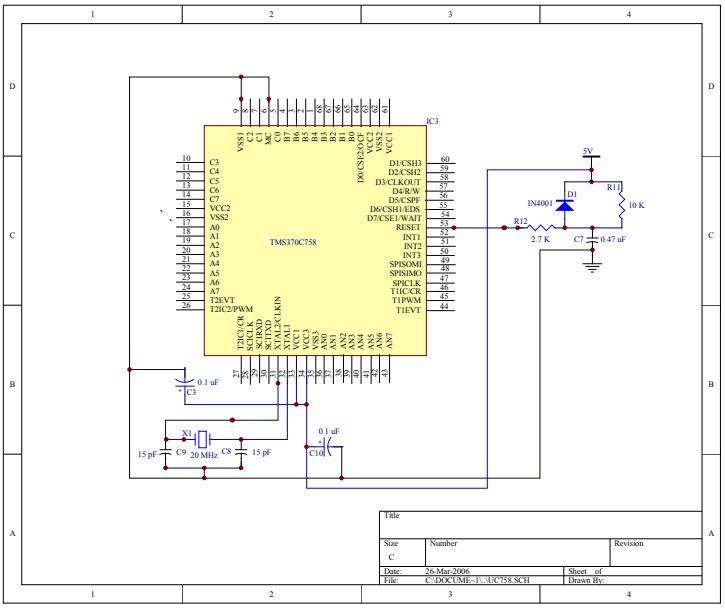


Fig. 5.2.1. Diagrama eléctrico del microcontrolador.

V.2.2. MEMORIA EXTERNA

Para el almacenamiento de las lecturas transmitidas por las unidades lectoras, cada DLC contará con una memoria externa. Ésta deberá ser capaz de:

- Almacenar una gran cantidad de información
- Grabado y borrado de la información con el voltaje de operación 5V.
- La información no debe perderse en caso de falta de polarización.

64 bytes es el tamaño de la cadena que en conjunto contiene la información del *transponder* y del reloj de tiempo real que se le asocia. Se hizo la consideración de que cada DLC pudiera almacenar hasta 5000 lecturas, por tanto:

Capacidad de la memoria = 5000 lecturas x 64 bytes = 320 kbytes

Entonces la memoria comercial que se utilizará corresponde a la de 512 kbytes de capacidad.

Para satisfacer el resto de los requisitos se utilizará una memoria del tipo *Flash EEPROM,* con la capacidad de almacenamiento definida y un voltaje de polarización de 5 V. Fabricante AMD modelo Am29F040B.

Para la conexión de la memoria se debe tomar en cuenta el modo de operación del ***µC***, el modo en el que se utilizará será el de microcomputador función ´´A´´ modo expandido. De las hojas de especificaciones del ***µC*** se obtiene las direcciones correspondientes a la memoria: C000h - FFFFh, un espacio de 16 k*bytes*. La memoria que se requiere conectar al ***µC*** como ya se definió tiene una capacidad de 512 k*bytes*, por lo que se tiene que dividir la memoria a conectar en 32 bloques de 16 k*bytes* cada uno. Considerando que el ***µC*** cuenta con 16 líneas de dirección se tendría una capacidad total de direccionamiento de 65536.

Como la memoria se va a dividir en bloques de 16 *kbytes,* para direccionar cada bloque se requerirá de una capacidad de direccionamiento de 16,384. Esta capacidad se logra utilizando 14 líneas de dirección (2^{14}=16384). Lo anterior implica que las líneas de direccionamiento C6 y C7 no serán utilizadas.

Se necesitan cinco terminales del ***µC*** (2^5=32) para manejar los 32 bloques de la memoria, cada uno de 16 kbytes. Estas terminales del ***µC*** se conectan a la memoria en sus conexiones identificadas como A14 a la A18.

Para el control de la memoria se necesitan tres terminales del ***µC***:

Las terminales C6 y C7 que están libres se utilizarán para habilitar el circuito de la memoria. El valor de estas terminales entre las direcciones C000h - FFFFh se garantiza siempre tendrán un "1" lógico y la memoria se habilita en su terminal $\overline{CE}$ con un “0” lógico, situación que se logra con el uso de una compuerta NAND.

La terminal D7 ($\overline{CSE1}$) se utilizará para habilitar la salida de datos de la memoria. Su conexión es directa.

La terminal D4 (R/$\overline{W}$) se utilizará para la lectura o escritura de la memoria, esta terminal del microcontrolador es configurada para lectura o escritura de periféricos. Su conexión es directa.

La polarización de la memoria se realiza conectando la terminal 16 a tierra y la terminal 32 al voltaje de polarización de la DLC. Entre estas terminales por recomendación del fabricante se conecta un capacitor para eliminar las señales armónicas del oscilador y el cristal utilizados en la DLC. El valor del capacitor es de 0.1 µF

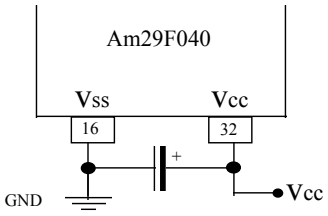


Fig. 5.2.2. Diagrama de polarización de la memoria.

En la fig. 5.2.3. se muestra la conexión entre la memoria Am29F040B y el *µC* TMS370C758.

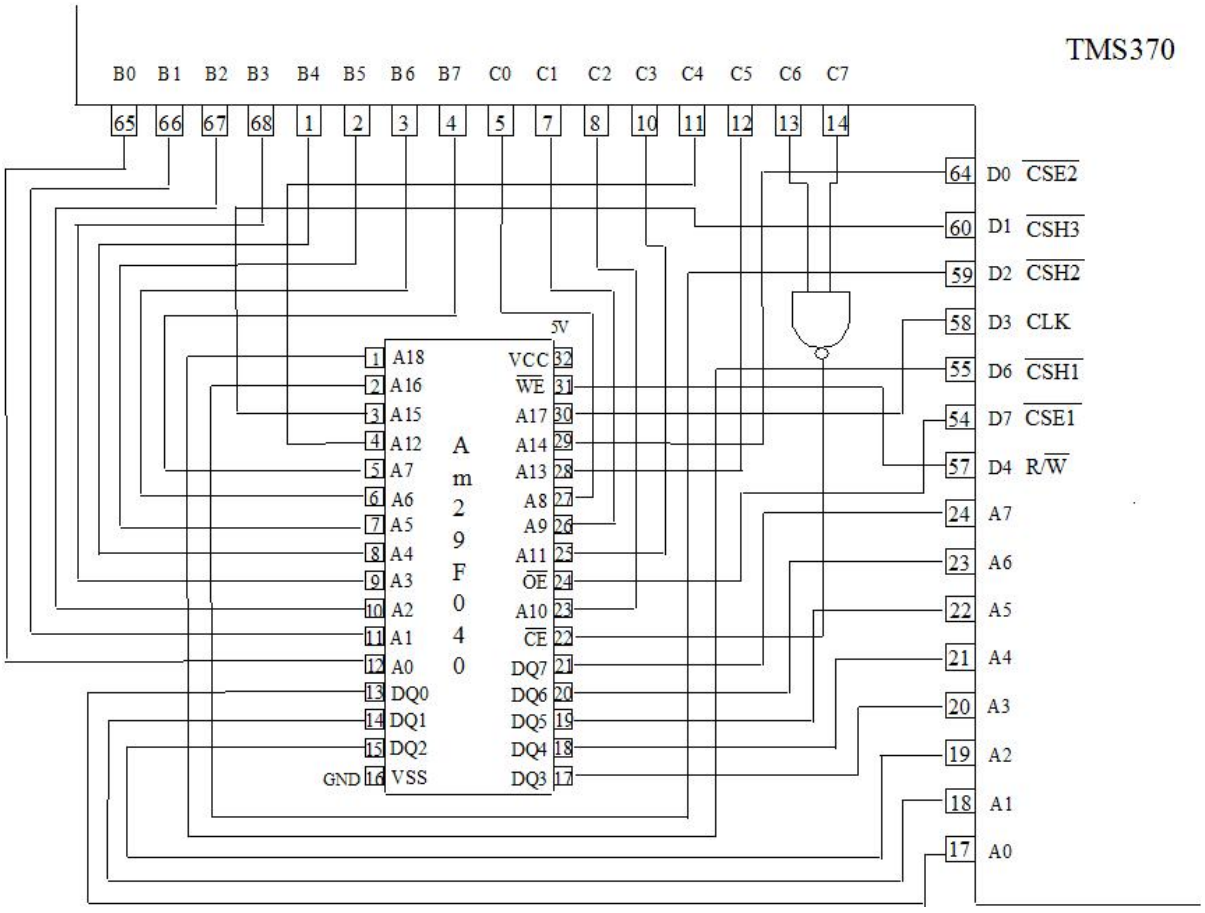


Fig. 5.2.3. Diagrama eléctrico conexión memoria y μC.

V.2.3. CIRCUITOS DE COMUNICACIÓN

El DLC se comunica con la unidad lectora para adquirir la información de los *transponders* y con la unidad concentradora para que se envíe dicha información hacia la *PC*, estos circuitos de comunicación se explican a continuación.

V.2.3.1 CIRCUITO DE COMUNICACIÓN DLC – UNIDAD LECTORA

La unidad lectora cuenta para su comunicación con 3 tipos de interfaz: RS-232, RS-422 o RS-485

Para el caso de nuestro diseño se tienen las siguientes condiciones a cumplir en la comunicación DLC - Unidad Lectora:

- Una DLC se comunica sólo con una unidad lectora a la vez.
- La distancia entre la unidad lectora y la DLC no es mayor a 20 cms.

Aunque las tres interfaces técnicamente pueden utilizarse, la comunicación entre el DLC y la unidad lectora es sólo entre ellas y están separadas por centímetros, condiciones idóneas para utilizar la RS-232.

Se elegirá la configuración estándar de la unidad lectora para la comunicación con el DLC, cuyas características son las siguientes:

- 9600 *baud*, 8 *bits* de datos, s/paridad, 1 *bit* de parada, *Xon/Xoff* habilitado
- Conector tipo *WECO* con la siguiente configuración (Fig. 5.2.4).

RS232 WECO Connector

Pin	Signal	Description	Direction
1	RxD	Receive Data	Input
2	DTR	Data Terminal Ready	Input
3	GND	Signal Ground	-
4	TxD	Transmit Data	Output
5	DSR	Data Set Ready	Output
6	GND	Signal Ground	-

Fig. 5.2.4. Conexión conector WECO interfaz RS-232.

Para implementar esta comunicación con el ***µC*** se emplea el módulo *SCI* del mismo, que cuenta con los elementos necesarios para realizar una implementación de software de manera directa, requiriendo únicamente un circuito para realizar la interfaz eléctrica.

Se empleará el circuito MAX232 para implementar la interfaz eléctrica entre el DLC y la unidad lectora, éste cuenta con las siguientes características (Fig. 5.2.5):

- Alimentación: 5V
- 2 Tx/2 Rx RS-232
- 4 capacitores externos
- 120 k*bps*

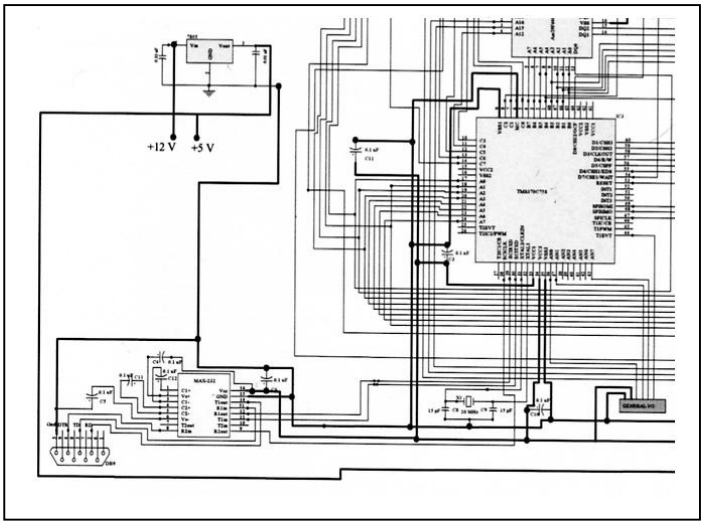


Fig. 5.2.5. Diagrama eléctrico conexión MAX232 y *µC*.

V.2.3.2 CIRCUITO DE COMUNICACIÓN DLC – UNIDAD CONCENTRADORA

La unidad concentradora se encargará, como se ha mencionado, de configurar a los DLCs y solicitar la información almacenada en ellas. Esta comunicación tendrá las siguientes características:

- La unidad concentradora coordina la comunicación con los DLC comunicándose con un DLC a la vez.
- La unidad concentradora siempre inicia la comunicación y establece la duración de la misma
- Para que las descargas de lecturas sean breves se requiere una velocidad de comunicación de 156 kbps

Se emplea el módulo *SPI* del *μC* para implementar la comunicación del DLC con la Unidad Concentradora como se muestra en la figura 5.2.6. En los DLCs el módulo *SPI* será configurado en modo esclavo.

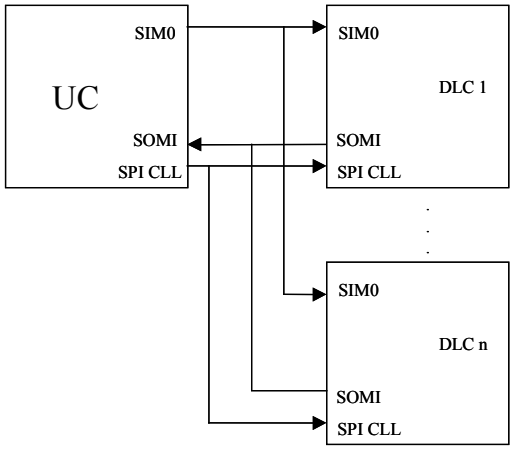


Fig. 5.2.6. Conexión módulo *SPI*. Unidad concentradora-DLCs.

V.2.4. RELOJ DE TIEMPO REAL (*RTC*)

Para que los DLCs puedan asociar el tiempo a cada *transponder* que reciban de la unidad lectora, será necesario que cada una cuente con un reloj de tiempo real. Más adelante se precisará que también la UC cuenta con un reloj de tiempo real y a través de éste, que se sincronizan todos los *RTC* de los controladores lógicos dedicados.

Para su selección, deberá cumplir con dos requerimientos:

-Rango de décimas de segundo a decenas de años en registros independientes
-Bajo costo

El circuito integrado MM58274C del cual se anexa su hoja de datos en el apéndice "A", satisface lo anterior, por lo que será el *RTC* utilizado en el desarrollo de este trabajo.

Para la conexión del *RTC* con el *μC* se tomó como referencia el siguiente diagrama proporcionado por el fabricante figura 5.2.7.

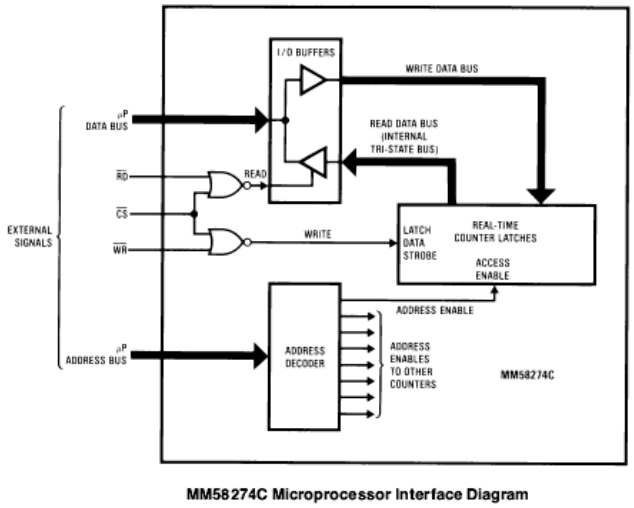


Fig. 5.2.7. Diagrama de conexión *RTC.*

El *μC* cuenta con un módulo de *software* para el control de periféricos, el cual se ubica en el rango de direcciones 10C0h – 10FFh, para el manejo del *RTC* se utilizan 16 direcciones por lo que se empleará el rango 10C0h – 10CFh. Con esto la interfaz entre el *μC* y *el RTC* queda de la siguiente forma:

- El bus de datos del *RTC* (DB0-DB3) se conecta directamente al bus de datos del *μC* (A0-A3)
- El bus de direcciones del *RTC* (AD0-AD3) se conecta directamente al bus de direcciones del *μC* (B0-B3)
- La terminal $\overline{WR}$ del *RTC* se conecta de forma directa a la terminal R/$\overline{W}$ del *μC*
- La terminal $\overline{RD}$ del *RTC* se conecta mediante una compuerta inversora a la terminal R/$\overline{W}$ del *μC* (Fig. 5.2.8)

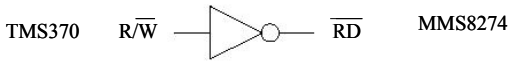


Fig. 5.2.8. Compuerta inversora *μC-RTC*

- La terminal $\overline{CS}$ del *RTC* se habilita con un "0" lógico, el cual se obtendrá de la siguiente forma:

En el rango de direcciones utilizadas para el *RTC* se asegura que la terminal B4 siempre tendrá un "0" lógico en su salida, el *μC* para el control de periféricos en el mismo rango de direcciones a través de su terminal D5 $\overline{(CSPF)}$ emite un pulso negativo. Con apoyo de una compuerta *OR* se conectarán en la entrada las terminales B4 y $\overline{CSPF}$ del *μC* ambas con un "0" lógico, obtenido a la salida de la compuerta un "0" lógico con el cual se habilitará la terminal $\overline{CS}$ del *RTC* (Fig.5.2.9).

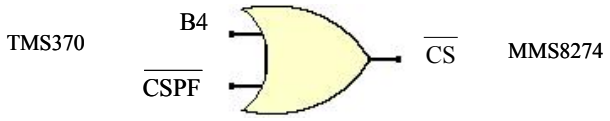


Fig. 5.2.9. Compuerta *OR*. *μC-RTC*

La conexión entre el *μC* y el *RTC* queda de la siguiente forma (Fig. 5.2.10):

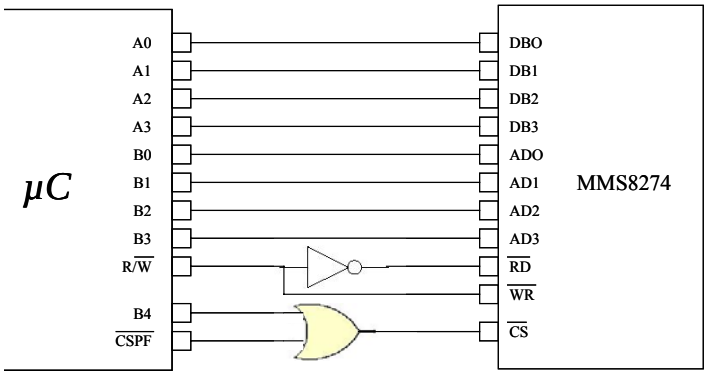


Fig. 5.2.10. Diagrama de conexión. *μC-RTC*

V.2.5. CIRCUITOS DE APOYO DEL SDR-1000

Para satisfacer las necesidades de los circuitos del DLC se requiere un voltaje de 5 Volts, con el cual operan todas las etapas de la misma. El consumo promedio de corriente del DLC se calcula a partir de los siguientes datos y se muestran en la tabla 5.2.2:

CIRCUITO	VOLTAJE (V)	CORRIENTE (m A)
Microcontrolador	5	50
MAX-232	5	15
MMS-8274	5	4
AM29FO40	5	40
7400	5	0.01
7432	5	0.01

Tabla. 5.2.2. Consumo de corriente DLC

Por lo tanto el consumo total de corriente del DLC es Itot = 109 mA

El módulo se alimenta con el regulador de voltaje L7805CV que tiene las siguientes características:

Vi= 7 – 25 V
Vo= 5 V
Io= 500 mA

Vi es el rango de voltaje en el que trabajará el regulador más adelante se determinará este dato.

El diagrama de conexión de acuerdo con la recomendación del fabricante se muestra en la figura 5.2.11:

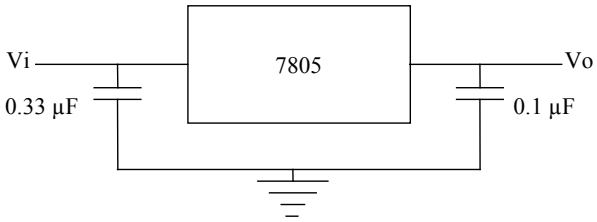


Fig: 5.2.11. Diagrama de conexión regulador.

V.2.6. INTEGRACIÓN DE LOS CIRCUITOS QUE CONFORMAN EL DLC

DIAGRAMA ELÉCTRICO DEL CONTROLADOR LÓGICO DEDICADO. La conexión de los circuitos que conforman cada uno de los controladores lógicos dedicados (DLCs) se muestra en la figura 5.2.12

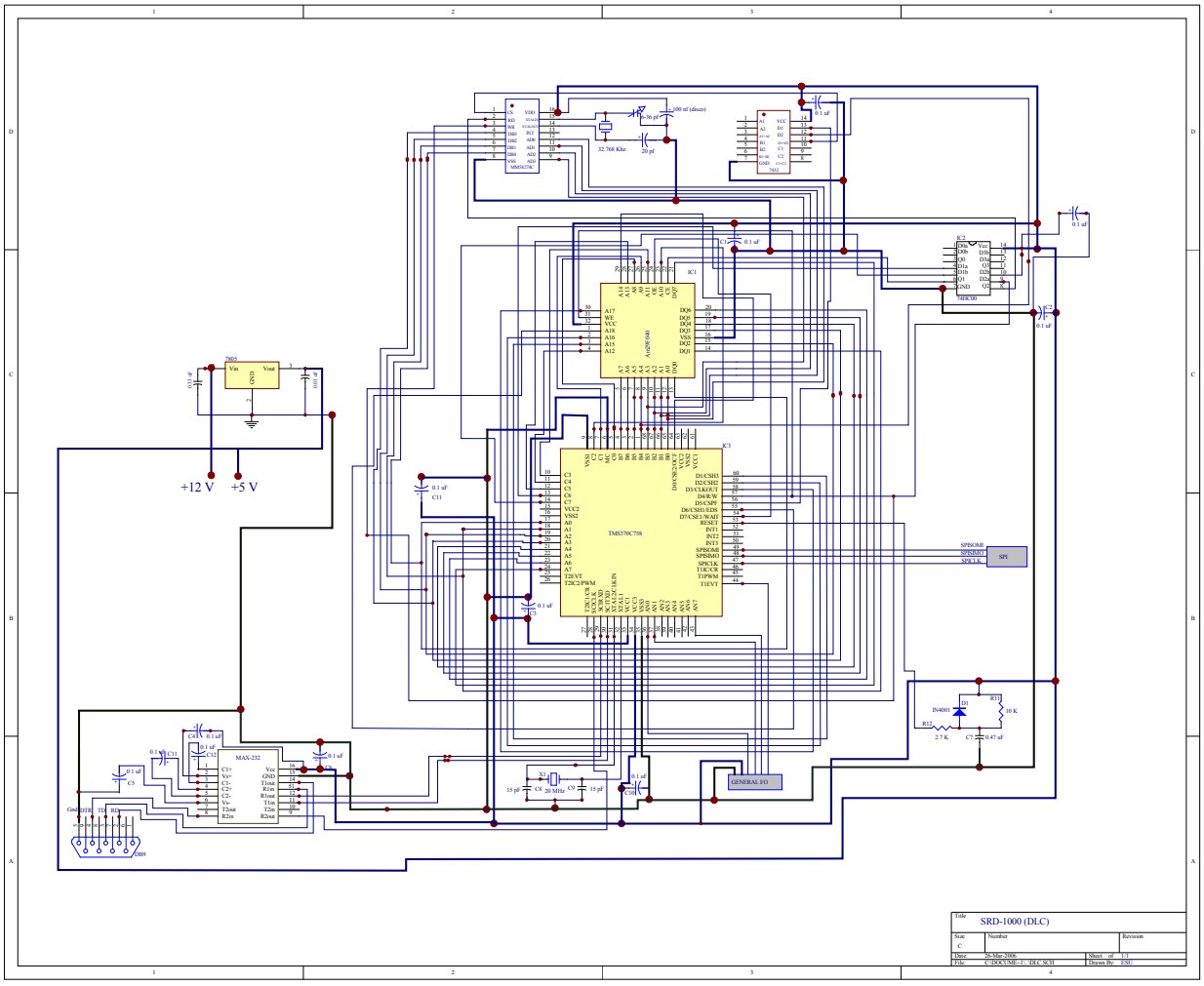


Fig. 5.2.12. Diagrama eléctrico del Controlador Lógico Dedicado.

V.3. DISEÑO DE LA UNIDAD CONCENTRADORA

Como se ha mencionado en capítulos anteriores, la unidad concentradora será la responsable de concentrar la información recibida por cada DLC, su resguardo, así como la transmisión de datos al equipo de cómputo de manera ordenada y dosificada. El número máximo de DLCs que manejará cada unidad concentradora será de seis. La unidad concentradora también basará su diseño en el microcontrolador TMS370C758 de la familia TMS370 de la firma *texas instruments*. Un equipo de cómputo conectado a la unidad concentradora a través del puerto serial RS-232 procesará la información para generar un archivo en formato txt.

A continuación se presentan los circuitos que conforman la unidad concentradora son, además del microcontrolador: de comunicación, pantalla, teclado, sonido, indicador de nivel de batería, de encendido y apagado de iluminación de teclado y pantalla y de alimentación.

V.3.1. MICROCONTROLADOR.

El microcontrolador utilizado para el diseño del concentrador es el mismo que el utilizado en el diseño del Controlador Lógico Dedicado (DLC), la selección y requerimientos para su operación fueron tratados en apartado V.2.1.

El microcontrolador en la unidad concentradora, será el encargado de procesar, administrar y transmitir toda la información recibida por los DLCs, así como del control de los dispositivos periféricos como el puerto serial, *display, buzzer*, teclado, etcétera.

V.3.2. CIRCUITOS DE COMUNICACIÓN

La unidad concentradora se comunica con las DLC para configurarlas y adquirir las lecturas almacenadas. De igual forma se comunica con un equipo de computo para enviar las lecturas para su posterior procesamiento.

V.3.2.1. CIRCUITO DE COMUNICACIÓN UNIDAD LECTORA-DLC

Este circuito de comunicación fue descrito en la sección V.2.3.1

V.3.2.2. CIRCUITO DE COMUNICACIÓN UNIDAD CONCENTRADORA-EQUIPO DE COMPUTO

La unidad concentradora, como ya se mencionó, es la encargada de transmitir en forma dosificada las lecturas almacenadas a un equipo de cómputo, el cual cuenta en forma general con un puerto serial que maneja el estándar RS-232. Este puerto será empleado para implementar la comunicación con la unidad concentradora, las características con las que trabajará el puerto son:

-9600 *bauds*, 8 bits de datos, s/paridad, 1 *bit* de parada, *Xon/Xoff* habilitado

El diseño con las características planteadas es el descrito en la sección V.2.3.1

V.3.3. CIRCUITO DEL DISPLAY

Este circuito sirve para visualizar los procesos ejecutados en el equipo, así como para su programación. Los requisitos que debe cumplir esta etapa son:

- Capacidad para desplegar caracteres alfanuméricos.
- Despliegue de un mínimo 32 caracteres simultáneamente.
- Tamaño compacto.
- Posibilidad de visualización en sitios poco iluminados.
- Voltaje de alimentación menor o igual a 5 V.

El circuito seleccionado para esta etapa es el **TM162G** de la marca **TIANMA** del cual se adjuntan su información técnica en el apéndice A. Este circuito tiene las siguientes características:

- Módulo *LCD* (*Liquid Crystal Display*).
- Presentación de caracteres en forma de matriz de puntos.
- Formato de carácter de 5 x 7 puntos.
- Voltaje de operación de +5 V.
- Interfaz directa para ***CPU*** de 4 u 8 *bits*.
- *Led* para iluminación de la pantalla.

De la misma forma que para el módulo *RTC,* para habilitar el *display* se emplearán terminales en el rango de direcciones que utilice éste para obtener el pulso adecuado, así como la terminal D5 ($\overline{\text{CSPF}}$) del *μC* que como ya se mencionó para el control de periféricos emite un pulso negativo.

Las direcciones que utiliza el *display* para su manipulación son dos: 10D0h Y 10D1h, en las cuales se puede garantizar que la terminal B4 del μC siempre tendrán un "1" lógico a la salida, con este dato, más el pulso negativo de la terminal del μC ($\overline{\text{CSPF}}$) se buscará por medio de compuertas obtener el "1" lógico que requiere el *display* para su activación

Como se recordará ya se cuenta con circuitos integrados de compuertas *NAND* y *OR* que tienen disponibilidad, por lo que se aprovecharán para habilitar el *display.* En la figura 5.3.1 se muestra el arreglo de compuertas, así como su tabla de verdad.

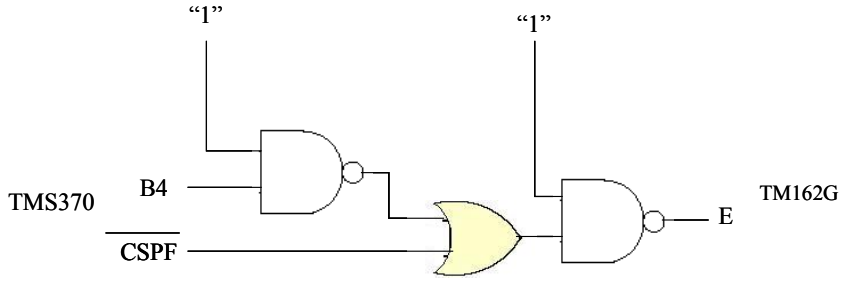


TABLA DE VERDAD

B4	CSPF	E
0	0	0
0	1	0
1	0	1
1	1	0

Fig. 5.3.1. Arreglo de compuertas *NAND* y *OR* y tabla de verdad.

Las demás conexiones son:

- El bus de datos del *display* se implementa con 8 *bits* conectándolo directamente al bus de datos del *µC* (A0-A7).
- La terminal RS del *display* será manejada por la terminal B0 del *µC* que proporciona los cambios lógicos de nivel (1 ó 0) para las funciones del *display*
- La terminal *R/W* será manejada de forma directa por la señal *R/W* del *µC*

El diagrama de conexiones se muestra en la figura 5.3.2

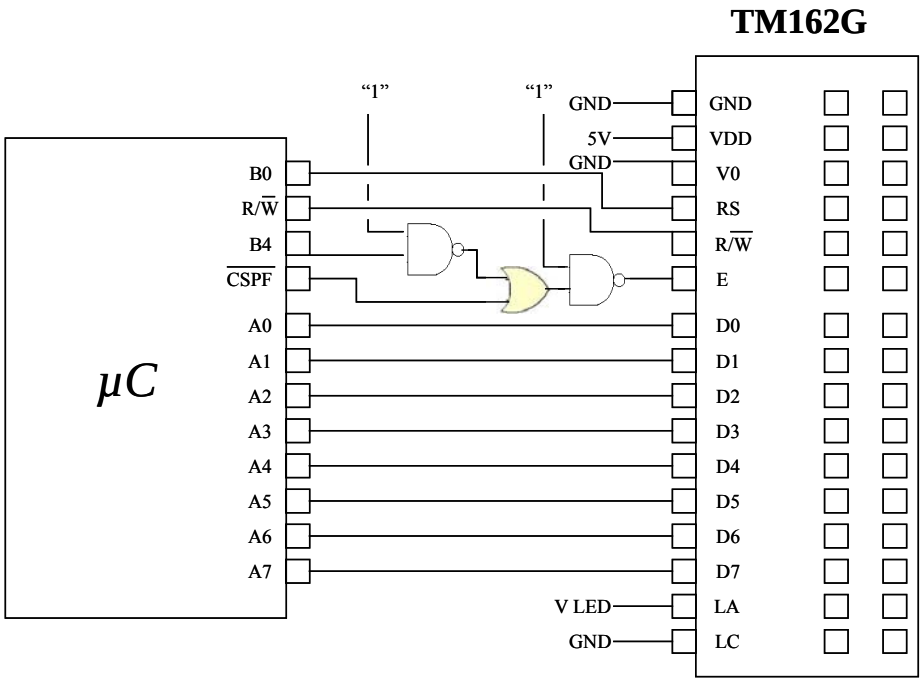


Fig. 5.3.2. Diagrama conexión del *display-µC*

V.3.4. CIRCUITO DEL TECLADO

La configuración del equipo se realiza mediante un teclado que consta de 25 teclas. Para esta función se implementa un arreglo de interruptores en forma de matriz de 5 x 5, de tal forma que para conocer que un interruptor se encuentra cerrado, se enviará un voltaje a través de cada una de las líneas del arreglo matricial, y se buscará a través de las columnas qué interruptor está cerrado, lo cual indicará la coordenada de la tecla oprimida.

Para la implementación del teclado se utilizarán interruptores del tipo "contacto momentáneo normalmente abierto", el material de estos interruptores son de hule de silicón con contactos de carbón, los cuales actúan sobre una tarjeta de circuito impreso, donde se encuentran las terminales normalmente abiertas. Este tipo de teclado se eligió debido a su bajo costo y durabilidad. La fig. 5.3.3 muestra el diagrama de la conexión del teclado.

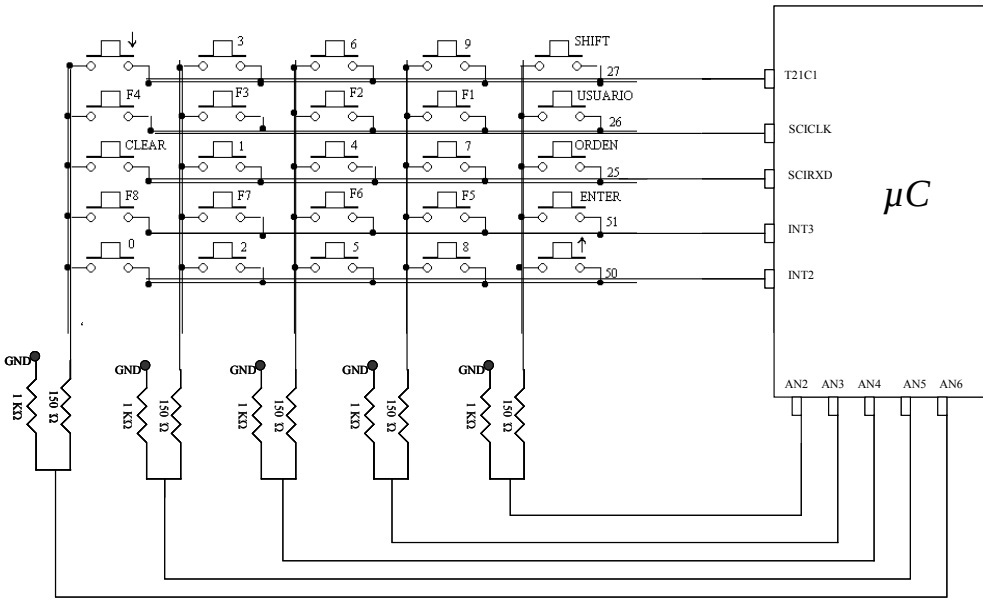


Fig. 5.3.3. Diagrama conexión del teclado-*μC.*

V.3.5. CIRCUITO DEL SONIDO

Esta etapa es incluida en la unidad concentradora, para realizar las siguientes funciones:

- Generar una señal audible mientras se mantenga sostenida una tecla.
- Generar una señal audible cuando se necesite que el operador atienda algún aviso mostrado en la pantalla.

Esta etapa es implementada con un multivibrador astable, el cual genera una señal que al ser aplicada a un *buzzer* emite un sonido audible.

El multivibrador astable mostrado en la figura 5.3.4 se implementa en forma discreta, se emplean dos transistores **BC547C**. La frecuencia de oscilación elegida para el circuito es de 1.25 kHz. La activación de la señal audible la realiza el ***μC*** enviando un pulso de +5 Vdc.

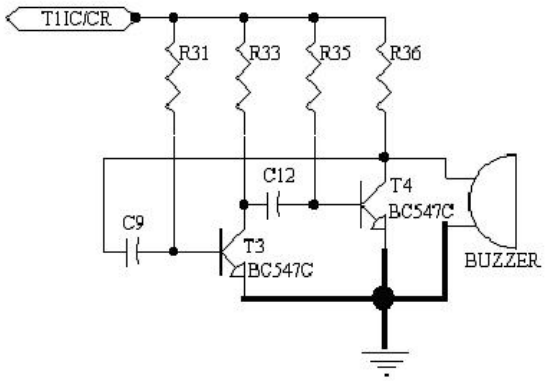


Fig. 5.3.4. Diagrama circuito de sonido.

Considerando los siguientes parámetros de operación:

$V_{T1C/CR}$= +5 V_{DC}
$I_{T1C/CR}$ = 0.5 mA
V_{CE} = 2 V
$I_{CT4} = I_{CT3} = I_{T1C/CR}/2$ = 0.25 mA
β = 400

Entonces, de la ecuación de malla de colector de T4:

$$V_{T1C/CR} = R_{36} \times I_{CT4} + V_{CE} \qquad \text{ec. 5.3.1a}$$

Despejando R36 de la ecuación anterior y sustituyendo valores:

$R_{36} = (V_{T1C/CR} - V_{CE})/ I_{CT4}$
= (5 V - 2 V)/.25 mA
= 12 kΩ

Por lo tanto valor comercial para $R_{33} = R_{36}$ = 15 kΩ.

De la ecuación de malla de base de T4: ec. 5.3.1b

$$V_{T1C/CR} = R_{35} \times I_B + V_{BE}$$

Despejando R35 de la ecuación anterior y sustituyendo valores:

$R_{35} = (V_{T1C/CR} - V_{BE})/ I_B$
= (5 V - .7 V)/(.25 mA/400) = 6.88 MΩ

Por lo tanto $R_{35} = R_{31}$ = 6.8 MΩ

Las ecuaciones para un multivibrador astable están dadas por:

$$T = t1 + t2 \qquad \text{ec. 5.3.1c}$$

Donde: $t1 = 0.69 R_{31} C_9$ y $t2 = 0.69 R_{35} C_{12}$ ec. 5.3.1d

Y el periodo está dado por $T = 1/f$ ec. 5.3.1e

De las ecuaciones anteriores y el dato de la frecuencia de oscilación (f) del circuito, se obtienen los valores de C_9 y C_{12}:

$$f = 1.4/(R_{31} \times C_9 + R_{35} \times C_{12}) \qquad \text{ec. 5.3.1f}$$

$C_9 = C_{12} = 1.4/(2 f R_{31})$
= 1.4/(2 x 1.25 kHz x 6.8 MΩ)
= 82 pF.

V.3.6. CIRCUITO INDICADOR DEL NIVEL DE LA BATERÍA

Es necesario avisar al operador del equipo de cronometraje, que el voltaje proporcionado por la batería está cerca de ser insuficiente para que el equipo funcione adecuadamente. Esté indicador consta de un circuito conectado desde la batería que proporciona su voltaje máximo (13.8 V) cuando tiene una carga mínima que se presenta en el momento en que las unidades lectoras están apagadas, y se conecta del lado del ***μC*** a la terminal 37 (AN0) del Convertidor Analógico/Digital.

El nivel de voltaje máximo proporcionado por la batería debe de ser acoplado al nivel máximo de voltaje permitido por el Convertidor Analógico/Digital (5 V). Para realizar este acoplamiento se utiliza un divisor de voltaje como el mostrado en la figura 5.3.5:

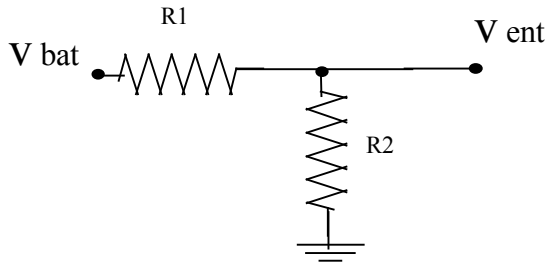


Fig. 5.3.5. Divisor de voltaje del circuito indicador del nivel de batería.

$$Vbat = V\ ent + I1R1 \qquad \text{Ecu. 5.3.2 a}$$

$$I_1 = (Vbat - Vent) / R_1 \qquad \text{Ecu. 5.3.2 b}$$

De la Ecu. 5.3.2a, despejando el valor de la resistencia se tiene:

$$R1 = (Vbat - Vent) / I1 \qquad \text{Ecu 5.3.2 c}$$

En el manual de aplicaciones de la familia **TMS370** de ***μC***, se tiene el valor de la corriente que entra a la terminal del convertidor A/D.

$$Ient = 1.44\ mA$$

Sustituyendo este valor por I_1, y los valores Vbat = 13.8V y Vent= 5 V, la Ecu. 5.3.2c se obtiene que el valor de la resistencia R1

$$R_1 = 6111\ \Omega$$

Utilizando el valor comercial de la resistencias se tiene que:

$$R_1 = 6.8\ K\Omega$$

V.3.7. CIRCUITO DE ENCENDIDO Y APAGADO DE ILUMINACIÓN DEL TECLADO Y LA PANTALLA

Debido a que uno de los requisitos es que la pantalla cuente con iluminación para los casos donde sea necesaria ésta, se incluye una etapa que permite mantener iluminado el teclado y la pantalla. Para la implementación de esta etapa se considera lo siguiente:

- La iluminación de la pantalla se efectúa con el ***led*** contenido en el propio dispositivo.
- Debido a que el teclado está fabricado con un material translúcido, este será iluminado por un arreglo de ***leds*** ubicados en la placa de su circuito impreso.

Con el objeto de economizar la energía proveniente de la batería, el encendido de la iluminación la realiza el ***µC***, únicamente cuando es necesario. Debido a que éte no puede proporcionar directamente el voltaje y la corriente demandada por los ***led***s del teclado y de la pantalla, se emplea una etapa que a partir de los voltajes de 5 y 0 V realice el encendido y apagado de los dispositivos de iluminación. La alimentación para la iluminación, se interrumpe simultáneamente en el teclado y la pantalla mediante la operación en las regiones de corte y activa directa de los transistores T1 y T2 mostrados en la figura 5.3.6 El empleo de dos transistores se debe a que los niveles de voltaje de encendido son diferentes en el teclado y la pantalla. La iluminación del teclado se realiza mediante 14 ***led***s, conectados en paralelo, con su ánodo conectado al emisor de T1 y el cátodo conectado a tierra. La terminal denominada ánodo se conecta al emisor de T2 y la terminal denominada cátodo va conectada a tierra. La acción de encendido de la iluminación se efectúa aplicando un nivel de +5 V en las resistencias de base de T1 y T2, para que estos pasen de la región de corte a la región de activa directa, encendiéndose así la iluminación. La señal requerida para el encendido es enviada desde el ***µC*** terminal T1 PWM. El apagado de la iluminación se efectúa con un nivel de 0 V aplicado en las resistencias de base de los transistores T1 y T2, provocando que pasen a la región de corte, lográndose así el apagado de la iluminación. Figura 5.3.6

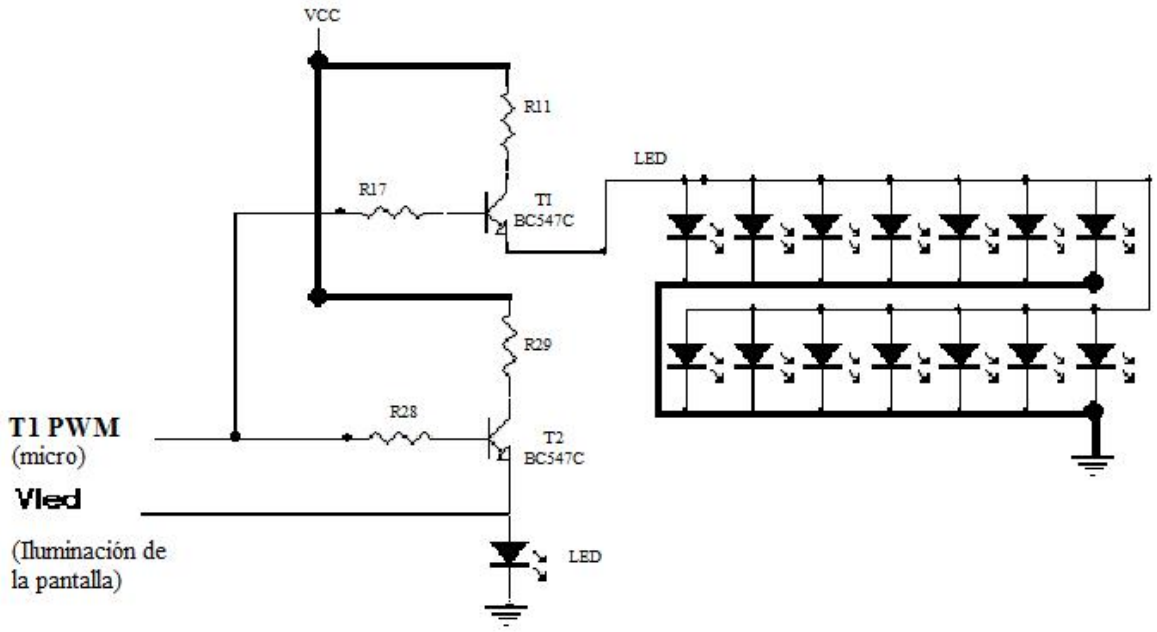


Fig. 5.3.6. Circuito de encendido y apagado de la iluminación del teclado y la pantalla.

A continuación se presenta el cálculo de los elementos de la etapa de control de iluminación del teclado y pantalla (encendido y apagado).

Para que exista una buena iluminación en el teclado, se estableció que los ***led***s operen con los siguientes valores de polarización, mismos que fueron el resultados en las pruebas de iluminación que se efectuaron:

$$V_{LED} = 2.2 \text{ V}$$
$$I_{LED} = 20 \text{ mA} = I_E$$

Como se mencionó, para encender la iluminación se requiere que el transistor T1 opere en la región de activa directa. En esta etapa se emplea el transistor **BC547C**, que tiene una ganancia de corriente típica de 400, un voltaje $V_{BE} = 0.7$ V, $I_{MAX} = 500$ mA, $V_{CE\,MAX} = 45$ V

Considerando $V_{CE} = 4.5$ V, para que exista una caída de voltaje en la resistencia de colector muy pequeña.

De las ecuaciones de malla obtenemos:

$$V_{T1PWM} = R_{17} I_B + V_{BE} + V_{LED} \qquad \text{ec. 5.3.3a}$$

$$V_{CC} = R_{11} I_C + V_{CE} + V_{LED} \qquad \text{ec. 5.3.3b}$$

Despejando R_{17} de la ec. 5.3.3a

.

$$R_{17} = (V_{T1PWM} - V_{BE} - V_{LED})/(I_E/401)$$
$$= (5\text{V} - 0.7\text{V} - 2.2\text{V})/(20\text{mA}/401) = 42 \text{ k}\Omega$$

El valor comercial que se utiliza es: $R_{17} = 39$ kΩ

Despejando R_{11} de la ec. 5.3.3b

$$R_{11} = (V_{CC} - V_{CE} - V_{LED})/ I_C$$
$$= (12\text{V} - 4.5\text{V} - 2.2\text{V})/(20\text{mA}) = 265 \text{ Ω}$$

El valor comercial que se empleará para $R_{11} = 270$ Ω.

En el caso de la pantalla, para que exista una buena iluminación, se requieren de las siguientes características:

$$V_{LED} = 4 \text{ V}$$
$$I_{LED} = 15 \text{ mA} = I_E$$
$$V_{CE} = 3 \text{ V (propuesto)}$$

Empleando las ecuaciones 5.3.3a y 5.3.3b, se calcula los valores de R_{28} y R_{29}:

$$R_{28} = (V_{CONTROL} - V_{BE} - V_{LED})/ I_B$$
$$= (5\text{V} - 0.7\text{V} - 4\text{V})/(15\text{mA}/400) = 8.02 \text{ k}\Omega$$

El valor comercial que se utiliza es: $R_{28} = 8.2\ k\Omega$

Para el cálculo de R_{29},

$$R_{29} = (V_{CC} - V_{CE} - V_{LED})/ I_C$$
$$= (12V - 3V - 4V)/(15mA) = 333.33\ \Omega$$

El valor comercial que se utiliza es: $R_{29} = 330\ \Omega$

V.3.8. CIRCUITO DE ALIMENTACIÓN

El sistema SDR-1000 tiene la opción de operar con alimentación eléctrica de 127 VCA o de ser necesario contar autonomía eléctrica por medio de una batería recargable de ácido-plomo.

Para el caso de operar con alimentación eléctrica en 127 VCA el circuito de alimentación contempla reguladores de voltajes para satisfacer la necesidades de polarización de los circuitos que son diferentes como se observa en la tabla 5.3.1.

CIRCUITO	CANTIDAD	VOLTAJE (V)	CORRIENTE (mA)	CORRIENTE TOTAL (mA)
LECTORES	6	12	300	1800
DLC	7	5	110	770
ILUMINACIÓN	7	5	20	20
			TOTAL:	**2590**

Tabla 5.3.1. Polarización y consumo de corriente sistema SDR-1000.

Se requieren de dos niveles de voltaje: 5 V y 12 V, el voltaje de 5 V lo obtenemos empleando el regulador de voltaje **CI L7805CV** y el voltaje de 12 V se obtiene con el regulador de voltaje **CI L7812CT**.

Ahora, para el caso de operar con la batería, las consideraciones para su selección fueron las siguientes:

Las competencias tiene un tiempo aproximado de 2 a 2.5 horas, para el caso del diseño se consideraron 4 horas de autonomía. El consumo de corriente se muestra en la tabla 5.3.1

Entonces tenemos: 2590 mA x 4 horas = 10.36 Amp

Por tanto, la batería deberá tener las siguientes características:

- Voltaje nominal, V = 12 V
- Capacidad nominal, I = 12 A/h

Cabe mencionar que cuando el sistema opera con la batería, éste cuenta con un relevador que realiza un *by-pass* al regulador **CI L7812CT** para alimentar directamente a las unidades lectoras que trabajan en 12 V.

Con el objeto de proteger a los circuitos del sistema de posibles sobre corrientes, se incluirá un fusible cuyas características se muestran a continuación:

Como protección al sistema se consideró el siguiente fusible:

I_{TOT} = 2.6 A. Fusible valor comercial I = 3 A
V= 127 V V = 250 V

En la figura 5.3.7 se muestran los circuitos eléctricos de alimentación y del cargador de baterías por la relación que existe entre ambos.

V.3.9. CARGADOR DE BATERIA.

En el punto anterior se definió que la batería a utilizar para la autonomía eléctrica que requiere el SDR-1000 tiene las siguientes características:

Voltaje nominal, V = 12 V
Capacidad nominal, I = 12 A/h

Su autonomía es de 4 horas por lo que es necesario recargar la batería, a continuación se muestran los criterios para el circuito cargador.

Las baterías de plomo ácido requieren un voltaje regulado con corriente limitada a 0.1 veces la capacidad de la batería, por lo que en este caso esta será de 1.2 A Icarga. Se requiere bajar el voltaje, si ésta corriente sube del valor mencionado. Conforme la batería se vaya cargando la corriente disminuirá, hasta que el voltaje alcance el voltaje deseado, para cumplir con estos requisitos se empleará el regulador LM317T. La información técnica se localiza en el apéndice A.

El voltaje de carga especificado para esta batería es de Vcarga = 13.8 V.

Para realizar la recarga de la batería se consideran 127 Vac por lo que se requerirá un transformador para proporcionar el voltaje adecuado para el circuito, a continuación se presenta el cálculo de este voltaje.

$V_{transformador} = (V_{bat} + V_{caída\ regulador} + V_{caída\ rectificador}) + 10\%$ ec. 5.3.4
$V_{transformador} = (13.8\ V + 3V + 1.4\ V) = 18.2\ V + 10\%$
$V_{transformador} = 20.02\ V$

El valor comercial que se emplea es:

$V_{transformador} = 24\ V$

El valor de la resistencia de la carga (*RESCHG*) limitará la corriente de carga del circuito, este resistor permitirá que el transistor entre en la región activa requiriendo 0.6 V para ello. Entre 0 V y 6 V el transistor ajustará al regulador para incrementar o decrementar el voltaje dependiendo de la corriente que pase, de acuerdo a lo explicado el valor de la resistencia es:

RESCHG = 0.6V/ máxima corriente = 0.6 V / 1.2 A = 0.5 Ω

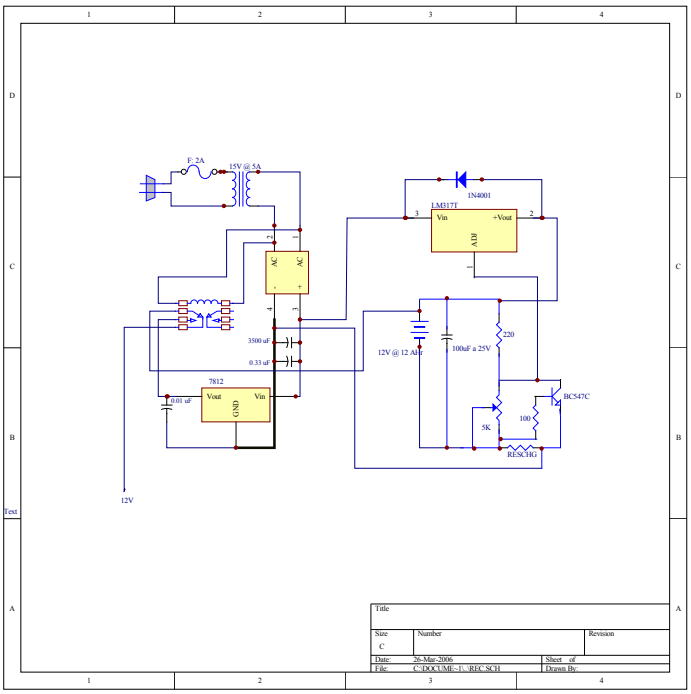


Figura 5.3.7. Circuito de alimentación y recarga de la batería.

V.3.10 INTEGRACIÓN DE LOS CIRCUITOS QUE CONFORMAN EL CONCENTRADOR

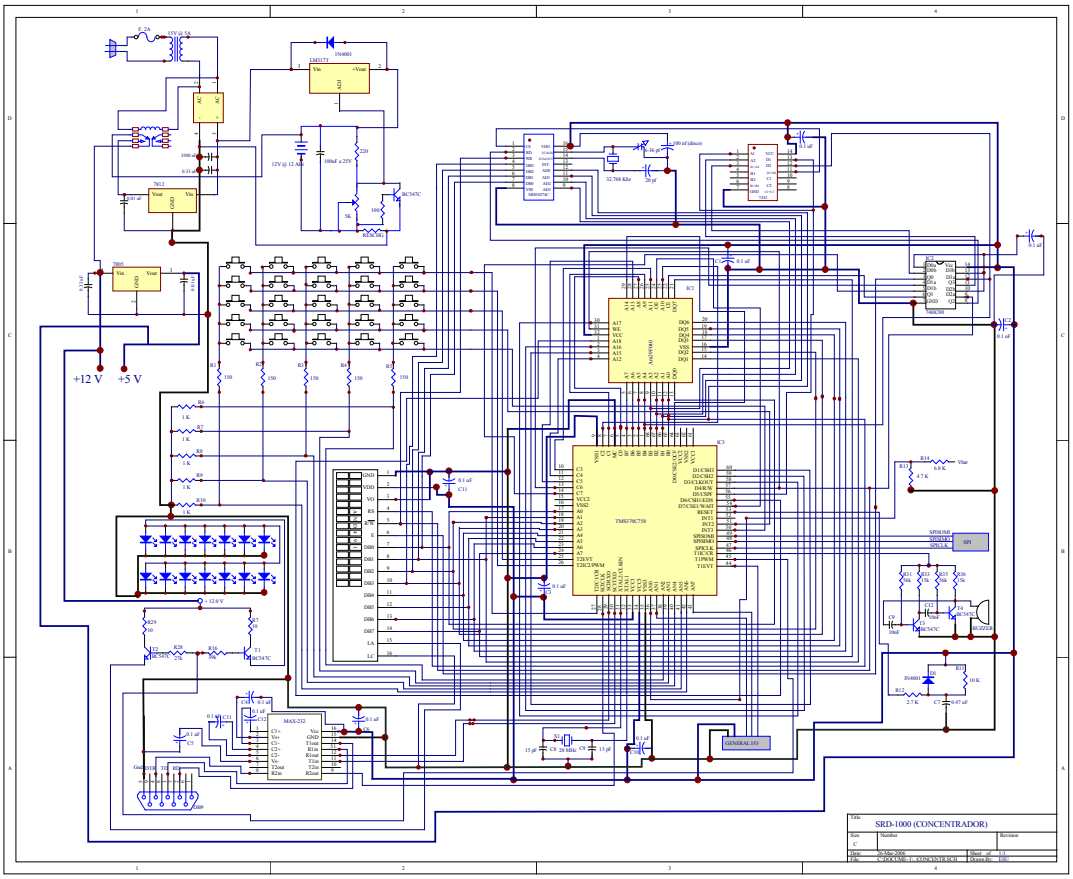


Fig. 5.3.8. Diagrama Eléctrico de la Unidad Concentradora.

V.4. EQUIPO SDR-100

A continuación se presentan imágenes del sistema (Figura 5.4.1):

SISTEMA SDR-1000
(Vista frontal)

COMPETENCIA EN LA ESEF

(Vista del SDR-1000 y antenas registrando información al paso de los corredores)

COMPETENCIA EN CU

(Adquisición de datos del sistema SDR-1000 a la conclusión de una carrera)

Fig. 5.4.1. Imágenes del sistema SDR-1000.

V.5. ANALISIS DE COSTOS

En esta sección se presenta la lista detallada de los componentes que conforman el sistema con sus costos:

Para este análisis de costo – beneficio se consideró lo siguiente:

- Se tomó como base el costo unitario de un equipo SDR-1000 con capacidad para 6 DLCs (unidad lector+antena)
- Carrera promedio con 10,000 competidores
- En promedio en una carrera se utilizan 3 sistemas SDR-1000, inicio y meta de la competencia y toma de tiempos en puntos intermedios.

En la tabla 5.5.1 se muestra el detalle del análisis de costos para un sistema SDR-1000.

Entonces:

Sistema SDR-1000:	72,072.87
x 3 equipos:	216,218.61
Subtotal:	**216,218.61**
Costo *transponder*	35.10
10,000 *Transponders*:	351,000.00
Subtotal:	**351,000.00**
Costo total:	**567,218.61**

El costo unitario por competidor sería entonces de:

567,218.61

Entre 10,000 corredores:

Costo ($/competidor): 56.72

Por lo tanto, el sistema podría ofrecerse a una renta de $100/atleta, precio que estaría muy por debajo de la renta de las empresas que actualmente ofrecen el servicio de cronometraje automático y representaría que la inversión podría recuperarse en un tiempo muy corto y tener utilidades desde un inicio.

COMPONENTE	CANTIDAD	UNIDAD	PRECIO UNIT mon nac	PRECIO TOTAL mon nac
TRANSFORMADOR 15 V, 5 A	1	PZA	$95.99	$95.99
INTERRUPTOR LLAVE (LS-09)	1	PZA	$66.00	$66.00
PORTA LED 5mm	14	PZA	$1.00	$14.00
LED ROJO 5mm (ES/ROJ-D)	6	PZA	$2.00	$12.00
LED VERDE 5mm (ES/VER-D)	7	PZA	$2.00	$14.00
LED AMBAR 5mm (ES/AMB-D)	1	PZA	$3.00	$3.00
PORTAFUSIBLE (FUS-AUB)	1	PZA	$4.00	$4.00
VENT117 VAC 150mA (VN4-117M)	1	PZA	$160.00	$160.00
BATERIA ACIDO-PLOMO 12V @ 12A	1	PZA	$260.00	$260.00
DISPLAY TM162JBC6-1	1	PZA	$135.00	$135.00
CONECTOR DB9 HEMBRA (500-109)	1	PZA	$15.00	$15.00
CON TIPO PUSH (250-700)	4	PZA	$11.00	$44.00
CON TIPO JACK (250-720)	4	PZA	$8.77	$35.08
CON TIPO JACK (250-705)	1	PZA	$13.00	$13.00
CON TIPO PUSH (250-581)	3	PZA	$5.00	$15.00
CABLE ARSA M-02X18MM	16	MTS	$16.00	$256.00
CABLE M-03X18MM	3	MTS	$20.00	$60.00
CABLE TR2X24-100	2	MTS	$4.00	$8.00
CABLE POT22BI-100	10	MTS	$3.00	$30.00
CABLE A22N-1000	11	MTS	$3.00	$33.00
CABLE A22R-1000	11	MTS	$3.00	$33.00
PIJAS PANEL	1	PZA	$20.00	$20.00
CTO. IMPRESO	7	PZA	$230.00	$1,610.00
BASE DE (32 pin) PLCC 32	7	PZA	$6.10	$42.70
BASE DE (68 pin) 68PC	7	PZA	$11.50	$80.50
C.I. 7400	7	PZA	$7.00	$49.00
C.I. 74LS32	7	PZA	$2.50	$17.50
C.I. MM58274CN	7	PZA	$147.00	$1,029.00
C.I. MAX232N	7	PZA	$7.00	$49.00
C.I. AM29F040B-90JI	7	PZA	$29.00	$203.00
C.I. TMS370C758AFNT	7	PZA	$100.00	$700.00
DIODO 1N4002	14	PZA	$0.40	$5.60
DIODO ZENER 5.6 V @ 1/2W 1N752A	1	PZA	$1.50	$1.50
TRANSISTOR C106B	1	PZA	$3.50	$3.50
TRANSISTOR C122B1	1	PZA	$14.50	$14.50
REGULADOR L7805CV	7	PZA	$3.00	$21.00
REGULADOR L7812CT	1	PZA	$23.00	$23.00
PUENTE DE DIODOS 4A RS402L	1	PZA	$10.00	$10.00
CRISTAL 20 MHz	7	PZA	$10.00	$70.00
CRISTAL 32.768 kHz	7	PZA	$7.50	$52.50
PORTAFUS T EURO V520101	1	PZA	$17.50	$17.50
CAP. CER. 15 pF	14	PZA	$1.50	$21.00
CAP. CER. 22p	7	PZA	$1.00	$7.00
CAP. CER. 10 nF	14	PZA	$1.00	$14.00
CAP. CER. ,1µF	90	PZA	$1.00	$90.00
CAP. CER. ,22µF	8	PZA	$1.50	$12.00
CAP. ELEC. .47µF @ 12 V	7	PZA	$0.70	$4.90
CAP. ELEC. 10µF @ 63 V	1	PZA	$1.00	$1.00
CAP. ELEC. 4700µF @ 35 V	1	PZA	$17.00	$17.00
CAP. VAR. 6-36 pf	7	PZA	$3.50	$24.50

Fig. 5.5.1. Costos componentes sistema SDR-1000 (continua).

COMPONENTE	CANTIDAD	UNIDAD	PRECIO UNIT mon nac	PRECIO TOTAL mon nac
RES. 470 Ω α 1/2	1	PZA	$0.30	$0.30
RES. 560 Ω α 1/2	1	PZA	$0.30	$0.30
RES.1 kΩ α 1/2	2	PZA	$0.30	$0.60
RES. 1.5 kΩ α 1/2	7	PZA	$0.30	$2.10
RES. 10 kΩ α 1/4	8	PZA	$0.30	$2.40
RES. 2.7 kΩ α 1/4	7	PZA	$0.30	$2.10
RES. 3.3 kΩ α 1/2	7	PZA	$0.30	$2.10
RES. 1 a 10 kΩ 10 α 25	1	PZA	$10.00	$10.00
POT. 110-5K	1	PZA	$4.00	$4.00
DISIPADOR TO-220 C/C	9	PZA	$9.00	$81.00
DISIPADOR TO-3	1	PZA	$10.00	$10.00
RELEVADOR HC2-H-AC115V	1	PZA	$111.00	$111.00
CONECTOR 2 POSICIONES MACHO-HEMBRA	39	PZA	$1.80	$70.20
CONECTOR 4 POSICIONES MACHO-HEMBRA	14	PZA	$3.50	$49.00
CONECTOR 6 POSICIONES MACHO-HEMBRA	7	PZA	$4.50	$31.50
CONECTOR 16 POSICIONES MACHO-HEMBRA	1	PZA	$15.00	$15.00
PINES CONECTORES	288	PZA	$0.50	$144.00
THERMOFIT NEGRO GPN 1/16	1	PZA	$6.00	$6.00
CHASIS	1	PZA	$2,160.00	$2,160.00
PANEL	1	PZA	$2,500.00	$2,500.00
TABLERO	1	PZA	$700.00	$700.00
CABLE ANTENAS	6	MTS	$104.00	$624.00
VULVANIZADO ANTENAS	6	PZA	$1,800.00	$10,800.00
UNIDAD LECTORAS	6	PZA	$8,205.00	$49,230.00
			TOTAL:	$72,072.87

Fig. 5.5.1. Costos componentes sistema SDR-1000.

CAPÍTULO VI

PROGRAMACIÓN DEL SISTEMA

En este capítulo se describe la programación necesaria para el funcionamiento de la Unidad Concentradora (UC) y de los Controladores Lógicos Dedicados (DLCs).

La figura 6.1.1 muestra el lenguaje en que serán programados los diferentes componentes del sistema automático para cronometrar los tiempo de corredores en competencias de maratón SDR-1000.

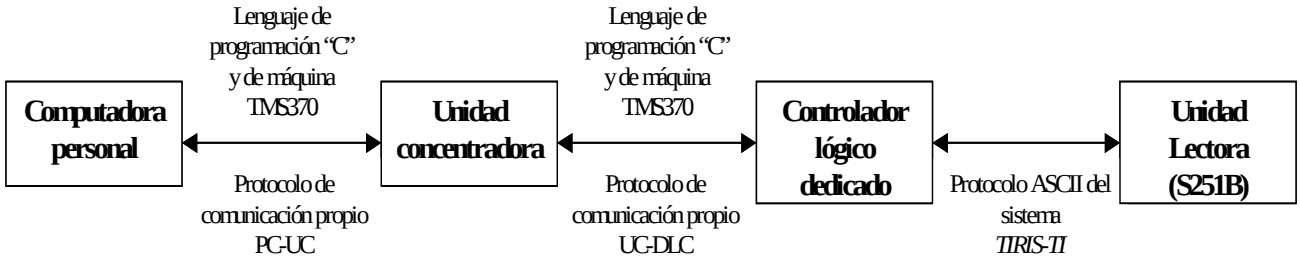


Fig. 6.1.1. Lenguajes de programación sistema SDR-1000.

Cabe señalar que se presentan los protocolos de comunicación conforme al diagrama 6.1.1, haciendo la hincapié que la empresa que contrate el sistema será quien desarrolle el *software* que se comunicará con la UC y que procesará la información en una computadora personal conforme a sus requerimientos y necesidades.

Este capítulo está dividido en dos partes: Programación del microcontrolador de la Unidad Concentradora y programación del microcontrolador de los Controladores Lógicos Dedicados. El código fuente de los programas se anexan en el apéndice B.

VI.1. PROGRAMACIÓN DEL MICROCONTROLADOR DE LA UNIDAD CONCENTRADORA

La función principal del sistema de cronometraje deportivo SDR-1000 como ya se mencionó será el de llevar el control del registro de datos, los cuales son introducidos al sistema mediante la lectura de un *transponder* por medio de una unidad lectora al cual se le asocia la fecha y la hora para su almacenamiento y posterior transmisión a una computadora.

En el contexto de este sistema, el microcontrolador de la unidad concentradora tendrá las siguientes funciones:

- Interpretar la información introducida por el teclado
- Desplegar en el formato requerido por la pantalla la información generada localmente y la transmitida por la *PC*
- Poner en modo lectura, las lecturas del sistema *TIRIS* y de los DLCs
- Configura el *RTC* del sistema
- Solicitar a cada DLC las lecturas almacenadas
- Transmitir de forma ordenada y dosificada los datos a la *PC*

VI.1.1. PARÁMETROS DE LA PROGRAMACIÓN DEL MICROCONTROLADOR DE LA UNIDAD CONCENTRADORA

Los parámetros que se considerarán para la programación del *µC* de la unidad concentradora son los siguientes:

- El teclado se configura de tal forma que queda una matriz de 5x5, en la cual se manejarán 5 columnas y 5 renglones
- Para la presentación de la información se empleará una pantalla de cristal líquido de 16 caracteres por 2 líneas

La pantalla tendrá la siguiente presentación:

		G	r	u	p	o		D	e	t	e	s	a		
h	h	:	m	m	:	s	s			D	D	/	M	M	

- En la comunicación con la *PC* se utilizará una interfaz serial RS-232.

El puerto debe ser configurado con las siguientes características:
- Velocidad: 9600 *bauds*
- paridad: ninguna
- bits de datos: 8
- bits de parada: 1
- protocolo: *xon/xoff*

- En la comunicación con los DLCs se utiliza la interfaz serial del módulo SPI configurado con las siguientes características:
- bits de datos: 8
- estado inactivo: alto
- velocidad: 156 k *bauds*

- Se emplea un Reloj de Tiempo Real (*RTC*) con las siguientes características:

FECHA: DD/MM/AA

Donde: Día (DD): 1 *byte* (1-31)
Mes (MM): 1 *byte* (1-12)
Año (AA): 1 *byte* (00-99)

HORA: hh/mm/ss/d

Donde: Hora (hh): 1 *byte* (0-23)
Minutos (mm): 1 *byte* (0-59)
Segundos (ss): 1 *byte* (0-59)
Décimas de segundo: 1 *byte* (0-9)

- Se implementarán dos modos de lectura; descritas a continuación:

a) MODO DE LECTURA 1, el sistema leerá los datos provenientes del *transponder* los etiquetará con la hora y fecha y los almacenará en la memoria del sistema.
b) MODO DE LECTURA 2, el sistema leerá los datos provenientes del *transponder* los etiquetará con la hora y fecha y los enviará a la computadora personal.

La cadena de datos proporcionada por los *transponders* será etiquetada con la hora y fecha en el siguiente formato:

T	b	A1	A2	A3	A4	b	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14

C15	C16	D1	D2	M1	M2	A1	A1	h1	h2	m1	m2	s1	s2	d

En la tabla 6.1.1 se describen cada uno de los elementos que conforman el formato mencionado

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
T	77(M), 82(R), 87(W)	1 *byte*	Byte de tipo de *transponder* R (*read only*),W(*read -write*), M(*multipage)*	Indica el tipo de *transponder*
b	32	1 *byte*	Valor de espacio en blanco para separar campos	Separador de campos
A1-A4	ASCII DEL 48(0) AL 57(9)	4 *bytes*	Clave de aplicación en la que se emplea el *transponder*	Aplicación de *transponder*
C1-C16	ASCII DEL 48(0) AL 57(9)	16 *bytes*	Clave almacenada en *transponder*	Clave *transponder*
D1,D2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Día en que se registro el evento D1= decenas de la cantidad día D2= unidades de la cantidad día	Día de registro del evento
M1,M2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Mes en que se registro el evento M1= decenas de la cantidad mes M2= unidades de la cantidad mes	Mes de registro del evento
A1,A2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Año en que se registro el evento A1= decenas de la cantidad año A2= unidades de la cantidad año	Año de registro del evento
h1,h2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Hora en que se registro el evento h1= decenas de la cantidad hora h2= unidades de la cantidad hora	Hora de registro del evento
m1,m2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Minutos en que se registro el evento m1= decenas de la cantidad minutos m2= unidades de la cantidad minutos	Minutos de registro del evento
s1,s2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Segundos en que se registro el evento s1= decenas de la cantidad segundos s2= unidades de la cantidad segundos	Segundos de registro del evento
d	ASCII DEL 48(0) AL 57(9)	1 *bytes*	d= Décimas en que se registro el evento	Décimas de registro del evento

Tabla 6.1.1. Cadena que proporcionan los *transponders.*

Con objeto de implementar la comunicación entre el equipo de cronometraje deportivo SDR-1000 y la computadora personal se implementará un protocolo que incluye comandos de manejo de lecturas, comandos para configuración de los DLCs y comandos para unidades lectoras, mismos que aplicarán según la función que se realice. Tabla 6.1.2

Función	Dirección	TIPO DE COMANDOS		
		De manejo de lecturas	De configuración DLCs	De manejo Unidades Lectoras
Solicitud de información al SDR-1000	**PC⇒ SDR-1000**	**X**		
Cantidad de lecturas almacenadas	**SDR-1000⇒PC**	**X**		
Lecturas almacenadas	**SDR-1000⇒PC**	**X**		
Borrado de lecturas	**PC⇒ SDR-1000**	**X**		
Configuración del *RTC*	**PC⇒ SDR-1000**		**X**	
Solicitud del *RTC*	**PC⇒ SDR-1000**		**X**	
Informe del *RTC*	**SDR-1000⇒PC**		**X**	
Solicitud de *reset flash*	**PC⇒ SDR-1000**		**X**	
Configuración modo lectura	**PC⇒ SDR-1000**			**X**
Confirmación modo lectura	**SDR-1000⇒PC**			**X**
Salida de operación en línea	**PC⇒ SDR-1000**			**X**
Confirmación salida de operación en línea	**SDR-1000⇒PC**			**X**
Ejecución de comando unidad lectora *TIRIS*	**PC⇒ SDR-1000**			**X**

Tabla 6.1.2. Tipos de comando en el protocolo de comunicación.

Como parte del protocolo mencionado es necesario el uso de comandos, los cuales se presentan y describen en la tabla 6.1.3

Elemento de Trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	**254**	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_EINF	**253**	1 *byte*	Comando de solicitud de envío de lecturas	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que le envíe la información almacenada
C_FMSG	**252**	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje
C_CLT	**251**	1 *byte*	Comando de contador de lecturas almacenadas	*Byte* empleado por la SDR-1000 para informarle a la *PC* que se envía la cantidad de lecturas almacenadas que se van a enviar
C_LECT	**250**	1 *byte*	Comando de lectura	*Byte* empleado por la SDR-1000 para informarle a la *PC* que se envía unas lecturas
C_BLT	**249**	1 *byte*	Comando de borrado de lecturas almacenadas	*Byte* empleado por la *PC* para informarle al SDR-1000 que se deben borrar las lecturas almacenadas

Tabla 6.1.3. Comandos del protocolo de comunicación entre la computadora personal y el SDR-1000 (continua).

C_DRTC	**248**	1 *byte*	Comando de configuración de reloj de tiempo real	*Byte* empleado por la *PC* para configurar el reloj de tiempo real del SDR-1000
C_ACG	**247**	1 *byte*	Comando de aviso de carga de memorias	*Byte* empleado por la *PC* para informarle al SDR-1000 que se van a cargar "L" páginas en la memoria
C_CMEM	**246**	1 *byte*	Comando de carga de memoria	*Byte* empleado por la PC para cargar la memoria del SDR-1000
C_SRTC	**245**	1 *byte*	Comando de solicitud de envío de información del *RTC*	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que le envíe la información del reloj de tiempo real
C_RSTFLASH	244	1 *byte*	Comando de solicitud de reset flash	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que limpie la memoria flash
C_ELINE	243	1 *byte*	Comando de solicitud de entrada al estado en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que opere en uno de los 3 estados de lectura de transponders
C_SLINE	242	1 *byte*	Comando de solicitud de salida al estado en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que deje de operar en uno de los 3 estados de lectura de transponders
C_PRG	241	1 *byte*	Comando de programación de transponder	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que programe un transponder
C_BUFOK	240	1 *byte*	Comando de solicitud de información a un *UC*	*Byte* empleado por el *UC* para solicitarle a algún DLC enviarle información
C_NOBUF	239	1 *byte*	Comando para avisar a *UC* que no hay información	*Byte* empleado por algún *UC* para avisarle al DLC que no hay información
C_ERRORBUF	238	1 *byte*	Comando para avisar a *UC* información llegó con error	*Byte* empleado por algún *UC* para avisarle al DLC la información recibida contiene errores.
C_DUMMY	237	1 *byte*	Comando para adquisición de datos del *UC*	*Byte* empleado por el *UC* para solicitarle información al DLC.
C_ELECT	236	1 *byte*	Comando para avisar a *UC* que no se ejecutó el comando	*Byte* empleado por algún *UC* para avisarle al DLC que no se ejecutó el comando indicado.
C_ONT	238	1 *byte*	Comando para avisar a DLC el encendido de la terminal *T1EVT*	*byte* empleado por el *UC* para indicar a algún DLC que encienda la terminal T1EVT.

Tabla 6.1.3. Comandos del protocolo de comunicación entre la computadora personal y el SDR-1000.

VI.1.1. PROTOCOLO DE COMUNICACIÓN ENTRE LA PC Y EL SDR-1000

El protocolo de comunicación entre el equipo SDR-1000 y la computadora personal se describe a continuación. Los tipos comandos para el envío y recepción de información de acuerdo con las instrucciones a realizar son los indicados en la tablas 6.1.2 y 6.1.3.

Solicitud de información al SDR-1000

Cuando la computadora personal desea obtener la información almacenada en la unidad SDR-1000 debe enviar la siguiente cadena de *bytes* al SDR -1000.

Solicitud de información a lector (PC⇒ SDR-1000)

C_IMSG	C_EINF	C_FMSG

En la tabla 6.1.4 se describen los campos del mensaje

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_EINF	253	1 *byte*	Comando de solicitud de envío de lecturas	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que le envíe la información almacenada
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.4. Comandos para solicitud de información de la PC al SDR-1000.

Al recibir el mensaje de solicitud de información, la unidad SDR-1000 envía una primera cadena de datos (cantidad de lecturas almacenadas) a la PC, seguida de "n" cadenas (lecturas) con las lecturas almacenadas de acuerdo a los siguientes formatos:

Cantidad de lecturas almacenadas (SDR-1000⇒PC)

C_IMSG	C_CLT	#L1	#L2	#L3	#L4	#L5	C_FMSG

En la tabla 6.1.5 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	Byte para identificar inicio de mensaje

Tabla 6.1.5. Comandos para solicitar la cantidad de lecturas almacenadas de la PC al SDR-1000 (continua).

C_CLT	251	1 *byte*	Comando de contador de lecturas almacenadas	Byte empleado por la SDR-1000 para informarle a la *PC* que se envía la cantidad de lecturas almacenadas que se van a enviar
L1	*ascii* 48-57	1 *byte*	Decenas de millares de cantidad de lecturas	En este *byte* se envía n las decenas de millar de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "1" se envía un 49d)
#L2	*ascii* 48-57	1 *byte*	Millar de cantidad de lecturas	En este *byte* se envía el millar de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "1" se envía un 49d)
#L3	*ascii* 48-57	1 *byte*	Centenas de cantidad de lecturas	En este *byte* se envía la centena de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "0" se envía un 48d)
#L4	*ascii* 48-57	1 *byte*	Decenas de cantidad de lecturas	En este *byte* se envía la decena de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "7" se envía un 55d)
#L5	*ascii* 48-57	1 *byte*	Unidades de cantidad de lecturas	En este *byte* se envía la unidad de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "9" se envía un 57d)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	Byte para identificar fin de mensaje

Tabla 6.1.5. Comandos para solicitar la cantidad de lecturas almacenadas de la PC al SDR-1000.

LECTURAS (SDR-1000⇒PC)

C_IMSG	**C_LECT**	**NL**	**T**	**b**	**A1**	**A2**	**A3**	**A4**	**b**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	
C15	**C16**	**D1**	**D2**	**M1**	**M2**	**A1**	**A1**	**h1**	**h2**	**m1**	**m2**	**s1**	**s2**	**d**	**C_FMSG**									

En la tabla 6.1.6 se describen los campos del mensaje para las lecturas almacenadas.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_LECT	250	1 *byte*	Comando de lectura	*Byte* empleado por la SDR-1000 para informarle a la *PC* que se envía una lectura
NL	48-53	1 *byte*	Indica la antena en la que se detectó el *transponder*	Permite identificar la antena en la que detectó el *transponder*, las antenas se numeran del 0 al 5, por ejemplo, si el *transponder* es detectado en la antena 3 se envía el carácter "2" 50d.
b	32	1 *byte*	Valor de espacio en blanco para separar campos	Separador de campos

Tabla 6.1.6. Comandos del mensaje para lecturas almacenadas que envía el SDR-1000 a la *PC* (continua).

A1-A4	ASCII DEL 48(0) AL 57(9)	4 *bytes*	Clave de aplicación en la que se emplea el *transponder*	Aplicación de *transponder*; por ejemplo, si el número de aplicación es 2345 se envían los caracteres A1=2(50), A2=3(51), A3=4(52) y A4=5(53)
C1-C16	ASCII DEL 48(0) AL 57(9)	16 *bytes*	Clave almacenada en el *transponder*	Clave *transponder*; por ejemplo, si la clave es 0000000000000007 se envían los caracteres C1 AL C15=0(48) Y C16=7(55)
D1,D2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Día en que se registro el evento D1= decenas de la cantidad día D2= unidades de la cantidad día	Día de registro del evento; por ejemplo, si el día es 12 se envían los caracteres D1=1(49) y D2=2(50)
M1,M2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Mes en que se registro el evento M1= decenas de la cantidad mes M2= unidades de la cantidad mes	Mes de registro del evento; por ejemplo, si el mes es febrero se envían los caracteres M1=0(48) y M2=2(50)
A1,A2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Año en que se registro el evento A1= decenas de la cantidad año A2= unidades de la cantidad año	Año de registro del evento; por ejemplo, si el año es 2004, se envían los caracteres A1=0(48) y A2=4(52)
h1,h2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Hora en que se registro el evento h1= decenas de la cantidad hora h2= unidades de la cantidad hora	Hora de registro del evento; por ejemplo, si la hora es 6 p.m, se envían los caracteres h1=1(49) y h2=8(56)
m1,m2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Minutos en que se registro el evento m1= decenas de la cantidad minutos m2= unidades de la cantidad minutos	Minutos de registro del evento; por ejemplo, si los minutos son 59, se envían los caracteres m1=5(53) y m2=9(57)
s1,s2	ASCII DEL 48(0) AL 57(9)	2 *bytes*	Segundos en que se registro el evento s1= decenas de la cantidad segundos s2= unidades de la cantidad segundos	Segundos de registro del evento; por ejemplo, si los segundos son32, se envían los caracteres s1=3(51) y s2=2(50)
d	ASCII DEL 48(0) AL 57(9)	1 *bytes*	Decenas en que se registro el evento	Décimas de registro del evento; por ejemplo, si las décimas son 6, se envía el carácter d=6(54)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	Byte para identificar fin de mensaje

Tabla 6.1.6. Comandos del mensaje para lecturas almacenadas
que envía el SDR-1000 a la *PC*.

Cuando la *PC* termina de recibir las n lecturas que esperaba, debe enviar al SDR-1000 la confirmación de que llegaron bien las lecturas, para que éstos sean eliminados de la memoria, esto se hace enviando inmediatamente después de recibir las lecturas el siguiente mensaje:

Borrar últimas n lecturas (PC⇒SDR-1000)

C_IMSG	C_BLT	#L1	#L2	#L3	#L4	#L5	C_FMSG

En la tabla 6.1.7 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje

Tabla 6.1.7. Comandos del mensaje que envía la PC al SDR-1000 confirmando
que las lecturas se recibieron correctamente (continua).

C_BLT	249	1 *byte*	Comando de contador de lecturas almacenadas.	*Byte* empleado por la SDR-1000 para informarle a la *PC* que se envía la cantidad de lecturas almacenadas que se van a enviar
#L1	*ascii* 48-57	1 *byte*	Decenas de millares de cantidad de lecturas	En este *byte* se envían las decenas de millar de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "1" se envía un 49d)
#L2	*ascii* 48-57	1 *byte*	Millar de cantidad de lecturas	En este *byte* se envía el millar de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "1" se envía un 49d)
#L3	*ascii* 48-57	1 *byte*	Centenas de cantidad de lecturas	En este *byte* se envía la centena de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "0" se envía un 48d)
#L4	*ascii* 48-57	1 *byte*	Decenas de cantidad de lecturas	En este *byte* se envía la decena de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "7" se envía un 55d)
#L5	*ascii* 48-57	1 *byte*	Unidades de cantidad de lecturas	En este *byte* se envía la unidad de la cantidad de lecturas a enviar en formato *ascii* (ejemplo, si es "9" se envía un 57d)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.7. Comandos del mensaje que envía la PC al SDR-1000 confirmando que las lecturas se recibieron correctamente.

Configuración de *RTC*

Para configurar la fecha y la hora de la unidad SDR-1000 se emplea el siguiente formato el cuál lo debe enviar la *PC*.

CONFIGURACION (PC⇒SDR-1000)

C_IMSG	C_DRTC	D1	D2	M1	M2	A1	A2	h1	h2	m1	m2	s1	s2	C_FMSG

En la tabla 6.1.8 se describen los campos del mensaje.

Elemento De Trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_DRTC	248	1 *byte*	Comando de configuración del reloj de tiempo real	*Byte* empleado por la *PC* para configurar el reloj de tiempo real del SDR-1000
Dn	48-57	2 *bytes*	Día del reloj de tiempo real D1= decenas en cantidad día D2= unidades en cantidad día	En estos *bytes* se envían las decenas y unidades de la cantidad del día del *RTC* del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "12" se envía d1= 49d, d2=50d)

Tabla 6.1.8. Comandos para que la *PC* configure el *RTC* del SDR-1000 (continua).

Mn	48-57	2 *bytes*	Mes del reloj de tiempo real M1= decenas en cantidad mes M2= unidades en cantidad mes	En estos *bytes* se envían las decenas y unidades de la cantidad del mes del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el mes "enero=01" se envía m1= 48d, m2=49d)
An	48-57	4 *bytes*	Año del reloj de tiempo real A1= decenas en cantidad año A2= unidades en cantidad año	En estos *bytes* se envían las decenas y unidades de la cantidad del año del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "2001" se envía a1=48d y a2=49d)
hn	48-57	2 *bytes*	Hora del reloj de tiempo real h1= decenas en cantidad hora h2= unidades en cantidad hora	En estos *bytes* se envían las decenas y unidades de la cantidad de las horas del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra la hora 17, se envía h1= 49d, h2=55d)
mn	48-57	2 *bytes*	Minutos del reloj de tiempo real m1= decenas en cantidad minutos m2= unidades en cantidad minutos	En estos *bytes* se envían las decenas y unidades de la cantidad de los minutos del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra los minutos 59, se envía m1= 53d, m2=575d)
sn	48-57	2 *bytes*	Segundos del reloj de tiempo real s1= decenas en cantidad segundos s2= unidades en cantidad segundos	En estos *bytes* se envían las decenas y unidades de la cantidad de los segundos del *RTC* del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra los segundos 59, se envía s1= 53d, s2=575d)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.8. Comandos para que la *PC* configure el *RTC* del SDR-1000.

Solicitud del *RTC*

Cuando la *PC* desea obtener la configuración del reloj de tiempo real almacenada en la unidad SDR-1000 debe enviar la siguiente cadena de *bytes* al SDR-1000.

Solicitud de información de RTC (PC⇒ SDR-1000)

C_IMSG	**C_SRTC**	**C_FMSG**

En la tabla 6.1.9 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando para inicio de mensaje	*Byte* para identificar inicio de mensaje
C_SRTC	245	1 *byte*	Comando de solicitud de envío de información del *RTC*	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que le envíe la información del reloj de tiempo real
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.9. Comandos para que la *PC* solicite la configuración el *RTC* al SDR-1000.

El equipo SDR-1000 contesta con esta cadena para informar a la *PC* el *RTC* que tiene configurado:

CONFIGURACION (SDR-1000⇒PC)

C_IMSG	C_DRTC	D1	D2	M1	M2	A1	A2	h1	h2	m1	m2	s1	s2	C_FMSG

En la tabla 6.1.10 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_DRTC	248	1 *byte*	Comando de configuración del reloj de tiempo real	*Byte* empleado por la *PC* para configurar el reloj de tiempo real del SDR-1000
Dn	48-57	2 *bytes*	Día del reloj de tiempo real D1= decenas en cantidad día D2= unidades en cantidad día	En estos *bytes* se envían las decenas y unidades de la cantidad del día del *RTC* del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "12" se envía d1= 49d, d2=50d)
Mn	48-57	2 *bytes*	Mes del reloj de tiempo real M1= decenas en cantidad mes M2= unidades en cantidad mes	En estos *bytes* se envían las decenas y unidades de la cantidad del mes del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el mes "enero=01" se envía m1= 48d, m2=49d)
An	48-57	4 *bytes*	Año del reloj de tiempo real A1= decenas en cantidad año A2= unidades en cantidad año	En estos *bytes* se envían las decenas y unidades de la cantidad del año del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "2001" se envía a1=48d y a2=49d)
hn	48-57	2 *bytes*	Hora del reloj de tiempo real h1= decenas en cantidad hora h2= unidades en cantidad hora	En estos *bytes* se envían las decenas y unidades de la cantidad de las horas del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra la hora 17, se envía h1= 49d, h2=55d)
mn	48-57	2 *bytes*	Minutos del reloj de tiempo real m1= decenas en cantidad minutos m2= unidades en cantidad minutos	En estos *bytes* se envían las decenas y unidades de la cantidad de los minutos del *RTC* del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra los minutos 59, se envía m1= 53d, m2=575d)
sn	48-57	2 *bytes*	Segundos del reloj de tiempo real s1= decenas en cantidad segundos s2= unidades en cantidad segundos	En estos *bytes* se envían las decenas y unidades de la cantidad de los segundos del *RTC* del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra los segundos 59, se envía s1= 53d, s2=575d)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.10. Comandos para que el SDR-1000 envíe la configuración de su *RTC* a la *PC*

Solicitud de *reset* de *Flash*.

Cuando la *PC* desea *resetear* la memoria *flash* en la unidad SDR-1000 debe enviar la siguiente cadena de *bytes* al SDR-1000.

Solicitud de *reset* de *flash* (PC⇒ SDR-1000)

C_IMSG	C_RSTFLASH	C_FMSG

En la tabla 6.1.11 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_RSTFLASH	244	1 *byte*	Comando de solicitud de *reset flash*	*Byte* empleado por la *PC* para solicitarle al SDR-1000 que limpie la memoria *flash*
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.11. Comandos para que la *PC* envíe el *reset* de la memoria *flash* del SDR-1000.

Configuración de modo de lectura

Cuando la *PC* desea poner en modo lectura a la unidad SDR-1000 debe enviar la siguiente cadena de *bytes* al SDR-1000.

Entrada a operación en línea (PC⇒ SDR-1000)

C_IMSG	C_ELINE	EDO	C_FMSG

En la tabla 6.1.12 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_ELINE	243	1 *byte*	Comando de solicitud de ingreso al modo lectura en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 active las antenas y comience a registrar los *transponder*
EDO	“1”(49) “2”(50)	1 *byte*	Modo de lectura de antenas de equipo **EDO=1** en este modo el equipo lee los *transponder* y los almacena en la memoria del equipo **EDO=2** en este modo el equipo lee los *transponder* y envía las lecturas a la PC	*Byte* empleado para que la *PC* avise al equipo en que modo de adquisición de datos debe trabajar, en el caso de maratones por ejemplo, se trabajará mayormente en el estado 1 leer y almacenar en este caso el *byte* enviado debe ser el carácter “1” 49 en decimal.
49C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.12. Comandos para que la *PC* ponga en modo lectura al SDR-1000.

El equipo confirma que entró a operar en el estado indicado regresando la misma cadena de datos.

Confirmación de operación en línea. (SDR-1000⇒ PC)

C_IMSG	C_ELINE	EDO	C_FMSG

En la tabla 6.1.13 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_ELINE	243	1 *byte*	Comando de solicitud de ingreso al modo lectura en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 active las antenas y comience a registrar los *transponder*
EDO	“1”(49) “2”(50)	1 *byte*	Modo de lectura de antenas de equipo **EDO=1** en este modo el equipo lee los *transponder* y los almacena en la memoria del equipo **EDO=2** en este modo el equipo lee los *transponder* y envía las lecturas a la *PC*	*Byte* empleado para que la *PC* avise al equipo en que modo de adquisición de datos debe trabajar, en el caso de maratones por ejemplo, se trabajará mayormente en el estado 1 leer y almacenar en este caso el *byte* enviado debe ser el carácter “1” 49 en decimal.
49C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.13. Comandos que envía el SDR-1000 a la *PC* para confirmar que entró en operación.

Salida de operación de modo “Línea”

Cuando la *PC* desea salir del modo lectura a la unidad SDR-1000 debe enviar la siguiente cadena de *bytes* al SDR-1000.

Salida a operación en línea (PC⇒ SDR-1000)

C_IMSG	C_SLINE	C_FMSG

En la tabla 6.1.14 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_SLINE	242	1 *byte*	Comando de solicitud de salida al modo lectura en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 desactive las antenas y comience a registrar *transponder*
49C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.14. Comandos para que la *PC* indique al SDR-1000 que salga del modo lectura

El equipo confirma que salió de operación en línea regresando la misma cadena de datos.

Confirmación de salida de operación en línea. (SDR-1000⇒ PC)

C_IMSG	C_SLINE	C_FMSG

En la tabla 6.1.15 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
C_SLINE	242	1 *byte*	Comando de solicitud de salida al modo lectura en línea	*Byte* empleado por la *PC* para solicitarle al SDR-1000 desactive las antenas y comience a registrar *transponder*
49C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.15. Comandos para que el SDR-1000 indique a la PC que salió de operación.

Ejecución de comando

Cuando la *PC* desea que algún lector *TIRIS* ejecute un comando envía el siguiente formato al equipo.

Ejecución de comando (PC⇒SDR-1000)

C_IMSG	COMANDO	NL	D1	…	....	DN	C_FMSG

En la tabla 6.1.16 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
COMANDO	*Ascii* C, B, J, Y, H, Z, X, P, L, ESC	1 *byte*	Comando a ejecutar en algún lector	*Byte* empleado por la *PC* para informarle al SDR-1000 que ejecute el lector de *TIRIS* seleccionado algún comando

Tabla 6.1.16. Comandos para que la *PC* envíe algún comando a ejecutar a un lector *TIRIS* (continua).

NL	*ascii* 48-57	1 *byte*	Número de lector que se desea ejecute el comando los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para envíos globales	En este *byte* se envía el lector destinatario del mensaje en formato *ascii* (ejemplo, si es para el lector 3 se envía un "2" 50d)
DN	*ascii*	N *bytes*	Datos necesarios para ejecución de comando (referirse a manual de *ascii* de *TEXAS INSTRUMENTS*)	En estos *bytes* se envían los datos relacionados al comando por ejemplo, si se envía el comando para grabar el código 15 en un *transponder* se envía la siguiente cadena: 000000000000000f (para grabar se envía siempre el código en formato hexadecimal)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.16. Comandos para que la *PC* envíe algún comando a ejecutar a un lector *TIRIS*.

Si al ejecutar el comando el lector de *TIRIS* genera información está es enviada a la *PC* en el siguiente formato:

Ejecución de comando (SDR-1000⇒PC)

C_IMSG	**COMANDO**	**NL**	**D1**	**…**	**....**	**DN**	**C_FMSG**

En la tabla 6.1.17 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
COMANDO	*Ascii* C, B, J, Y, H, Z, X, P, L, ESC	1 *byte*	Comando a ejecutar en algún lector	*Byte* empleado por la *PC* para informarle al SDR-1000 que ejecute el lector de *TIRIS* seleccionado algún comando
NL	*ascii* 48-57	1 *byte*	Número de lector que se desea ejecute el comando los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para envíos globales	En este *byte* se envía el lector destinatario del mensaje en formato *ascii* (ejemplo, si es para el lector 3 se envía un "2" 50d)
DN	*ascii*	N *bytes*	Datos necesarios para ejecución de comando (referirse a manual de *ascii* de *TEXAS INSTRUMENTS)*	En estos *bytes* se envían los datos relacionados al comando por ejemplo, si se envía el comando para grabar el código 15 en un *transponder* se envía la siguiente cadena: 000000000000000f (para grabar se envía siempre el código en formato hexadecimal)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.17. Comandos que envía el lector *TIRIS* a la *PC* si generó información.

VI.1.2 PROTOCOLO DE COMUNICACIÓN ENTRE LA UNIDAD CONCENTRADORA Y LAS DLCs.

Se implementa un protocolo de comunicación entre la unidad concentradora y las DLCs, este se realizará empleando los módulos *SPI* de los microcontroladores.

Este protocolo incluye los siguientes tipo de comandos para las siguientes funciones. Tabla 6.1.18:

Función	Dirección	TIPO DE COMANDOS		
		De manejo de lecturas	**De configuración DLCs**	**De manejo Unidades Lectoras**
Borrado de *buffer* lectora	**UC⇒ DLCs**	**X**		
Confirmación borrado de *buffer*	**DLCs⇒ UC**	**X**		
Lectura contenido *buffer*	**UC⇒ DLCs**	**X**		
Configuración de borrado de *buffer*	**DLCs⇒ UC**	**X**		
Cambio de periodo de carga	**UC⇒ DLCs**	**X**		
Confirmación del periodo de carga	**DLCs⇒ UC**	**X**		
Entrada a modo ejecución	**UC⇒ DLCs**	**X**		
Confirmación de modo a entrada ejecución	**DLCs⇒ UC**	X		
Solicitud de lecturas	**UC⇒ DLCs**		X	
Lecturas almacenadas	**DLCs⇒ UC**		**X**	
Borrado de memoria	**DLCs⇒ UC**		**X**	
Configuración *RTC*	**UC⇒ DLCs**			**X**
Confirmación configuración *RTC*	**UC⇒ DLCs**			**X**
Programación de *transponder*	**DLCs⇒ UC**	**X**		
Modo de lectura	**UC⇒ DLCs**	**X**		
Confirmación modo de lectura	**UC⇒ DLCs**	X		

Tabla 6.1.18. Tipos de comando en el protocolo de comunicación.

Los protocolos de comunicación entre el ***μC* de la unidad concentradora (UC)- *μC* DLCs** son los siguientes:

Borrado de *buffer* de almacenamiento

Borrado del *buffer* de almacenamiento **(UC⇒ DLC´s)**

C_IMSG	No. LECT	C	C_FMSG

En la tabla 6.1.19 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje

Tabla 6.1.19. Comandos que envía la *UC* a los DLCs para el borrado del *buffer* de almacenamiento (continua).

No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo si es para el DLC 3 se envía un "2" 50d)
C	67	1 *byte*	Comando de borrado de buffer	*Byte* empleado para borrar el buffer del lector de *TIRIS* empleado en el modo "normal"
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.19. Comandos que envía la *UC* a los DLCs para el borrado del *buffer* de almacenamiento.

Los DLCs confirman que borraron el *buffer* de almacenamiento regresando la misma cadena de datos.

Confirmación de borrado de *buffer* de almacenamiento. (DLCs⇒ UC)

C_IMSG	No. LECT	C	C_FMSG

En la tabla 6.1.20se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo si es para el DLC 3 se envía un "2" 50d)
C	67	1 *byte*	Comando de borrado de *buffer*	*Byte* empleado para borrar el buffer del lector de *TIRIS* empleado en el modo "normal"
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.20. Comandos que envían los DLCs a la *UC* confirmando que fue borrado el *buffer* de almacenamiento.

Lectura del contenido del *buffer*

Lee contenido del *buffer* **(UC⇒ DLCs)**

C_IMSG	No. LECT	B	C_FMSG

En la tabla 6.1.21 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
B	66	1 *byte*	Comando de lectura de *buffer*	*Byte* empleado para leer el contenido del *buffer* del lector de *TIRIS*
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.21. Comandos que envía la *UC* a los DLCs para leer el contenido del *buffer*.

Los DLCs confirman que se realizó la lectura del *buffer* de almacenamiento regresando la misma cadena de datos.

Confirmación de borrado de *buffer* de almacenamiento. (DLCs⇒ *UC*)

C_IMSG	No. LECT	B	C_FMSG

En la tabla 6.1.22 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
B	66	1 *byte*	Comando de lectura de *buffer*	*Byte* empleado para leer el contenido del *buffer* del lector de *TIRIS*
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.22. Comandos que envían los DLCs a la *UC* confirmando que se llevó a cabo la lectura del *buffer* de almacenamiento.

Cambio del periodo de carga

Cambia periodo de carga **(UC⇒ DLC)**

C_IMSG	No. LECT	Z	PER MSB h	PER LSB h	C_FMSG

En la tabla 6.1.23 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
Z	90	1 *byte*	Comando de cambio de periodo de carga	*Byte* empleado para cambiar el periodo de carga en el lector en el rango de 15 a 255 mseg, por ejemplo, si se quiere que el periodo de carga sea de 32 mseg se envía la cadena "2"(50d), "0"(48d) (20h = 32d)
PER	ASCII	2 *bytes*	Valor de periodo de carga en lector de *TIRIS*	
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.23. Comandos que envía la *UC* a los DLCs para modificar el periodo de carga.

Los DLCs confirman que se realizaron las modificaciones al periodo de carga regresando la siguiente cadena de datos.

Confirmación de modificaciones al periodo de carga. (DLC⇒ UC)

C_IMSG	No. LECT	Z	C_FMSG

En la tabla 6.1.24 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje

Tabla 6.1.24. Comandos que envían los DLCs a la *UC* confirmando que se llevó a cabo la modificación al periodo de carga (continua).

No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
Z	90	1 *byte*	Comando de cambio de periodo de carga	*Byte* empleado para cambiar el periodo de carga en el lector en el rango de 15 a 255 mseg, por ejemplo, si se quiere que el periodo de carga sea de 32 mseg se envía la cadena "2"(50d), "0"(48d) (20h = 32d)
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.24. Comandos que envían los DLCs a la *UC* confirmando que se llevó a cabo la modificación al periodo de carga.

Entrada a modo de ejecución

Entrar a modo de ejecución **(UC⇒ DLC)**

C_IMSG	No. LECT	X	C_FMSG

En la tabla 6.1.25 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
X	88	1 *byte*	Comando de lectura instantánea de *transponders*	*Byte* empleado para forzar al lector a realizar una lectura y entrar al estado de espera
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.25. Comandos que envía la *UC* a los DLCs para entrar en modo de ejecución.

Los DLCs confirman que se realizaron la ejecución del comando regresando la identificación de los *transponders* detectados con la siguiente cadena de datos.

Confirmación de entrada a modo de ejecución (DLC⇒ UC)

C_IMSG	No. LECT	X	D1	……	D21	C_FMSG

Solicitud de lecturas

Solicitud de lecturas **(UC⇒ DLC)**

C_IMSG	No. LECT	C_EINF	C_FMSG

En la tabla 6.1.26 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo si es para el DLC 3 se envía un "2" 50d)
C_EINF	253	1 *byte*	Comando de solicitud de lecturas	*Byte* empleado para solicitar al DLC todas las lecturas almacenadas en su memoria flash
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.26. Comandos que envía la *UC* a los DLCs para solicitar la cantidad de lecturas que tienen almacenadas.

Los DLCs envían una primera cadena informando la cantidad de lecturas almacenadas seguido de las cadenas correspondientes de lecturas.

Información de cantidad de lecturas almacenadas (DLC⇒ UC)

C_IMSG	No. LECT	C_CLT	# LECT MSB	# LECT LSB	C_FMSG

Lecturas almacenadas (DLC⇒ UC)

C_IMSG	No. LECT	C_LECT	D1	……..	D36	C_FMSG

.
.
.

C_IMSG	No. LECT	C_LECT	D1	……..	D36	C_FMSG

Borrado de memoria

Borrado de memoria **(UC⇒ DLC)**

C_IMSG	No. LECT	C_RST FLASH	C_FMSG

En la tabla 6.1.27 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato ascii (ejemplo, si es para el DLC 3 se envía un "2" 50d)
C_RSTF LASH	244	1 *byte*	Comando de borrado de memoria	*Byte* empleado para borrar todo el contenido de la memoria flash del DLC
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.27. Comandos que envía la *UC* a los DLCs para el borrado de la memoria.

Los DLCs confirman que realizaron el borrado de la memoria regresando la siguiente cadena de datos.

Confirmación de borrado de memoria. (DLC⇒ UC)

C_IMSG	No. LECT	C_RST FLASH	C_FMSG

En la tabla 6.1.28 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
C_RSTF LASH	244	1 *byte*	Comando de borrado de memoria	*Byte* empleado para borrar todo el contenido de la memoria flash del DLC
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.28. Comandos que envían los DLCs a la *UC* confirmando que se borró la memoria.

Configuración de *RTC*

Configuración de *RTC* **(UC⇒ DLC)**

C_IMSG	No. LECT	C_DRTC	D	M	A	h	m	s	C_FMSG

En la tabla 6.1.29 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
C_DRTC	248	1 *byte*	Comando de configuración de reloj de tiempo real	*Byte* empleado para configurar el reloj de tiempo real
Dn	48-57	2 *bytes*	Día del reloj de tiempo real D1= decenas en cantidad día D2= unidades en cantidad día	En estos *bytes* se envían las decenas y unidades de la cantidad del día del RTC del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "12" se envía d1= 49d, d2=50d)
Mn	48-57	2 *bytes*	Mes del reloj de tiempo real M1= decenas en cantidad mes M2= unidades en cantidad mes	En estos *bytes* se envían las decenas y unidades de la cantidad del mes del RTC del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el mes "enero=01" se envía m1= 48d, m2=49d)
An	48-57	4 *bytes*	Año del reloj de tiempo real A1= decenas en cantidad año A2= unidades en cantidad año	En estos *bytes* se envían las decenas y unidades de la cantidad del año del RTC del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra el día "2001" se envía a1=48d y a2=49d)
hn	48-57	2 *bytes*	Hora del reloj de tiempo real h1= decenas en cantidad hora h2= unidades en cantidad hora	En estos *bytes* se envían las decenas y unidades de la cantidad de las horas del RTC del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra la hora 17, se envía h1= 49d, h2=55d)
mn	48-57	2 *bytes*	Minutos del reloj de tiempo real m1= decenas en cantidad minutos m2= unidades en cantidad minutos	En estos *bytes* se envían las decenas y unidades de la cantidad de los minutos del RTC del SDR -1000 los números se envían en formato *ascii* (ejemplo, si se registra los minutos 59, se envía m1= 53d, m2=575d)
sn	48-57	2 *bytes*	Segundos del reloj de tiempo real s1= decenas en cantidad segundos s2= unidades en cantidad segundos	En estos *bytes* se envían las decenas y unidades de la cantidad de los segundos del RTC del SDR-1000 los números se envían en formato *ascii* (ejemplo, si se registra los segundos 59, se envía s1= 53d, s2=575d)
C_FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.29. Comandos que envía la *UC* para configurar el *RTC* de los DLCs.

Los DLCs confirman que realizaron la configuración del *RTC* regresando la siguiente cadena de datos.

Confirmación de configuración de *RTC*. (DLC⇒ UC)

C_IMSG	No. LECT	C_DRTC	D	M	A	h	m	s	C_FMSG

Programación de *transponder*

Programación de *transponder* **(UC⇒ DLC)**

C_IMSG	No. LECT	P	D1h	D2h	D3h	D4h	D5h	…	…	D15h	D16h	C_FMSG

En la tabla 6.1.30 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje
No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
P	80	1 *byte*	Comando de programación de *transponder*	*Byte* empleado para grabar la memoria de *transponders* del tipo *R/W*
D	ASCII	16 *bytes*	Datos a grabar en *transponder*	Cadena de datos a grabar en *transponder* por ejemplo, para grabar en un *transponder* el número 2048 0274603029037288(d) se envía la cadena"8000f9c00000000e8" (56(d),48(d), 48(d), 48(d),70(d),57(d),67(d), 48(d), 48(d), 48(d), 48(d), 48(d), 48(d), 48(d), 48(d), 69(d),56(d))
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.30. Comandos que envía la *UC* a los DLCs para configurar los *transponders.*

Los DLCs confirman que se realizó la configuración de los transpoders regresando la siguiente cadena de datos.

Confirmación de ejecución de comando. (DLC⇒ UC)

C_IMSG	No. LECT	P	C_FMSG

Modo de lectura

Modo lectura **(UC⇒ DLC)**

C_IMSG	No. LECT	L/ <ESC>	EDO	C_FMSG

En la tabla 6.1.31 se describen los campos del mensaje.

Elemento de trama	Valor de elemento (decimal)	Longitud máxima	Descripción	Función
C_IMSG	254	1 *byte*	Comando de inicio de mensaje	*Byte* para identificar inicio de mensaje

Tabla 6.1.31. Comandos que envía la *UC* a los DLCs para indicar el modo de lectura en que trabajarán (continua).

No. LECT	48-53	1 *byte*	Número de lector de *TIRIS* asociado al DLC que se desea ejecute el comando. los lectores de *TIRIS* serán numerados del 0 al 5, correspondiendo el 0 al lector 1 y el 5 al lector 6, si se desea que todos los lectores ejecuten el comando a la vez se enviará el número 8 para instrucciones globales	En este *byte* se envía el DLC destinatario del mensaje en formato *ascii* (ejemplo, si es para el DLC 3 se envía un "2" 50d)
L/<ESC>	76/69	1 *byte*	Comando de programación de modo de lectura de lector de *TIRIS*	*Byte* empleado para definir el modo de lectura del lector de *TIRIS*, L = pone al lector en modo de lectura continua, E = el lector sólo envía un "nuevo *transponder* detectado"
EDO	49,50	1 *byte*	Modo de lectura del DLC	Define al DLC que hacer cuando el lector le envía el identificador de un *transponder* detectado 1= lo etiqueta con *RTC* y lo almacena en la memoria del DLC, 2= lo etiqueta y lo envía al concentrador
C-FMSG	252	1 *byte*	Comando de fin de mensaje	*Byte* para identificar fin de mensaje

Tabla 6.1.31. Comandos que envía la *UC* a los DLCs para indicar el modo de lectura en que trabajarán.

Los DLCs confirman que entraron en la opción indicada del modo de lectura regresando la siguiente cadena de datos.

Confirmación de modo de lectura (DLC⇒ UC)

C_IMSG	No. LECT	L/ <ESC>	EDO	C_FMSG

VI.2. DESARROLLO DE LA PROGRAMACIÓN DEL *μC* DE LA UNIDAD CONCENTRADORA

La programación del *μC* de la unidad concentradora como se mencionó anteriormente se realizará en los lenguajes ensamblador y C.

Las rutinas que se realizan en lenguaje ensamblador, son las que requieren el manejo directo con los módulos del microcontrolador estas son:

- Definición de módulos *(intvecs.asm)*
- Comunicación unidad concentradora-DLCs *(intps2.asm)*
- Escritura de memoria EEPROM *(eeprom.asm)*
- Rutina de comunicación unidad concentradora-PC *(sciints.asm)*
- Rutina de atención a interrupción de apagado de iluminación *(itimluz.asm)*

Las rutinas que se realizaron en lenguaje C son:

- Sistema operativo (*os.cpp)*
- Rutina principal *(sartci.cpp)*
- Rutina de comunicaciones *(sci.cpp)*

A continuación se describen las rutinas del programa del microcontrolador de la unidad concentradora

Definición de módulos

En esta rutina se definen e inicializan los módulos empleados del ***μC*** estos son***:***

- *Timer 2*: Se emplea para definir un *timer* que servirá para apagar la iluminación del equipo
- SCI: Empleado en la comunicación con la PC
- *Timer 1*: Empleado en la comunicación con la PC para establecer un tiempo límite para intentar la comunicación
- SP1: Empleado en la comunicación con los DLCs

VI.2.1. COMUNICACIÓN UNIDAD CONCENTRADORA-DLCs

Esta rutina empleará el protocolo de comunicación entre la unidad concentradora y los DLCs descritos anteriormente:

Para implementar la comunicación se define el *buffer* “bufspi1” en la memoria RAM del ***μC,*** el tamaño será de 22 *bytes* este *buffer* se empleará para almacenar la información a transmitir a los DLCs. De igual forma se define el *buffer* “bufsp12” de 42 *bytes* de longitud para almacenar la información procedente de los DLCs.

El diagrama de flujo de esta rutina se muestra en la siguiente figura 6.2.1

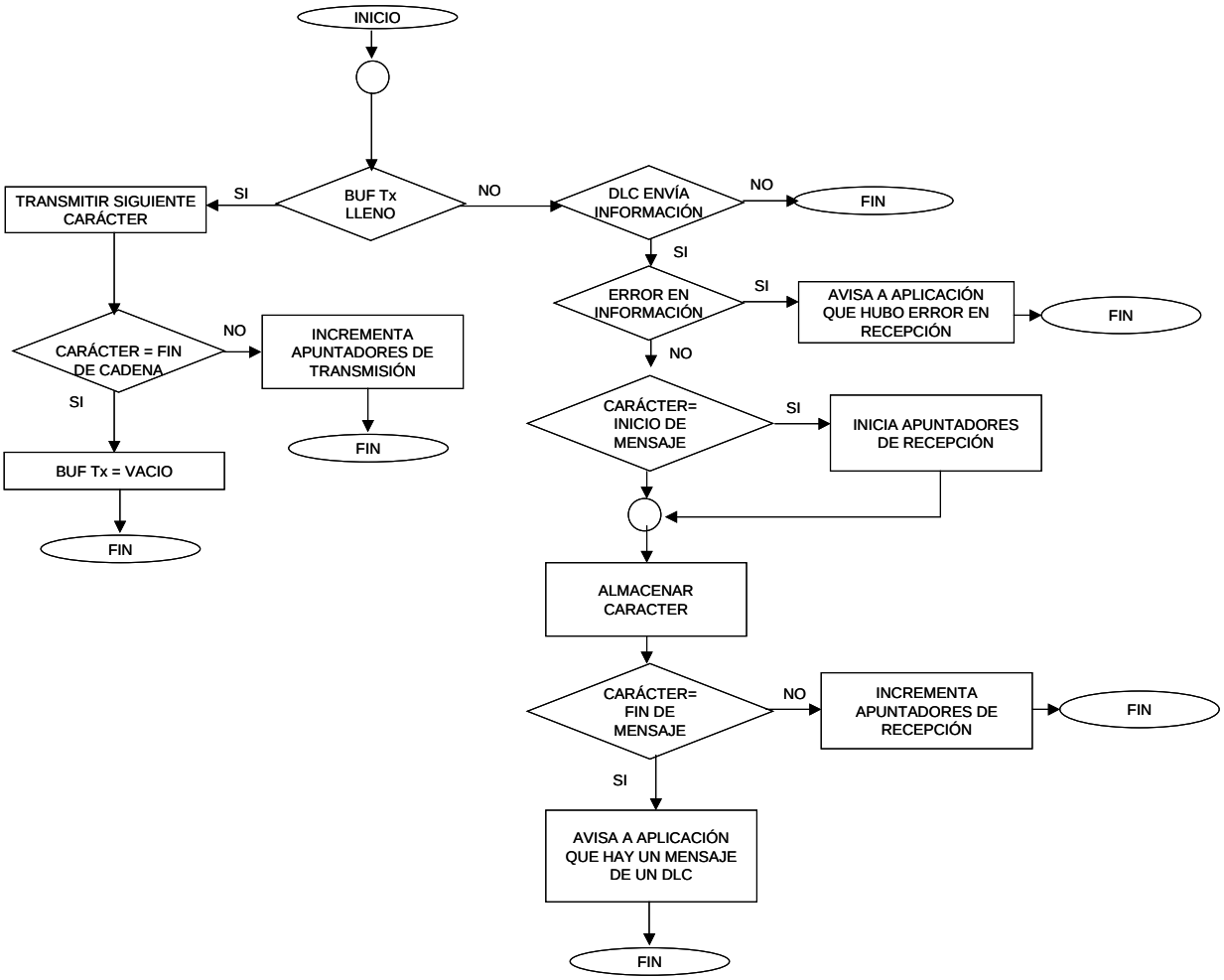


Fig. 6.2.1. Rutina de comunicación unidad concentradora-DLCs.

VI.2.2. ESCRITURA DE MEMORIA *EEPROM*

Esta rutina se encargará de realizar la escritura de la memoria *EEPROM* del microcontrolador dónde residirá la configuración del sistema.

VI.2.3. RUTINA DE ATENCIÓN A INTERRUPTORES DE APAGADO DE ILUMINACIÓN

El equipo cuenta con la iluminación en el *display* y en el teclado, este se encenderá automáticamente cuando se pulse una tecla o se reciba información de la PC, con objeto de economizar batería una vez que se encienda se iniciará el *timer* para apagar de forma automática la luz 10 segundos después.

Esta rutina atiende a la interrupción generada por el *timer* para reiniciar el mismo o desactivarlo y apagar la iluminación una vez concluido el periodo establecido.

VI.2.4. RUTINA DE COMUNICACIÓN UNIDAD CONCENTRADORA - PC

En esta implementación se empleará el protocolo de comunicación “Unidad concentradora – PC” descrito anteriormente.

Se definen en esta comunicación los *buffers* para almacenar la información recibida y la información a transmitir. Estos son: *buftx* de 70 *bytes* de longitud empleado para almacenar la información a transmitir y *bufrx* de 72 *bytes* empleado para almacenar el mensaje recibido de la *PC*.

Los diagramas de flujo de estas rutinas se muestran a continuación en la figura 6.2.2

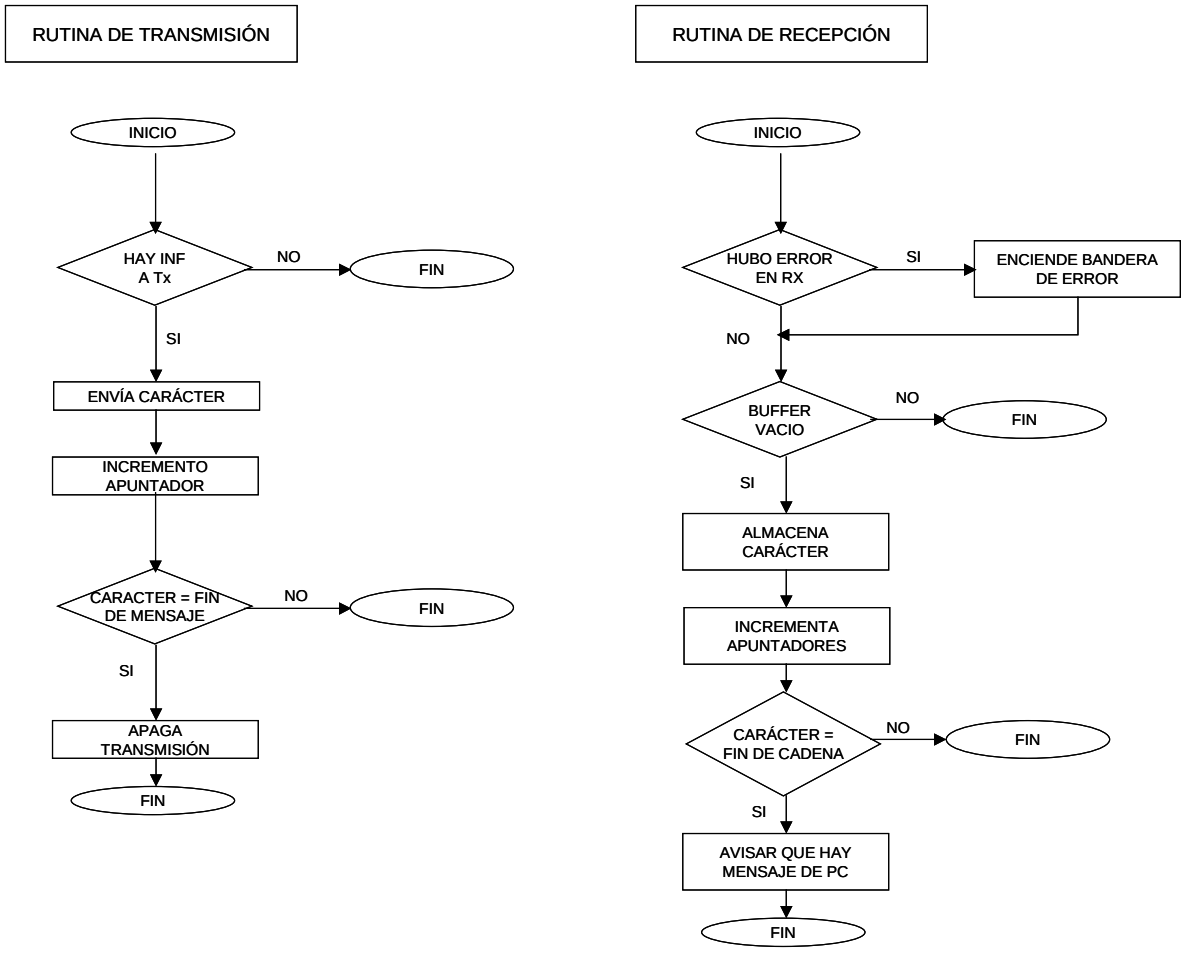


Fig. 6.2.2. Rutina de comunicación unidad concentradora – *PC*.

VI.2.5. SISTEMA OPERATIVO

Esta rutina se encarga de configurar e inicializar los puertos del ***μC,*** así como ofrecer los servicios del teclado, *display* del equipo y el reloj de tiempo real.

A continuación se muestran las principales rutinas del sistema operativo del microcontrolador de la unidad concentradora

VI.2.6. MANEJO DE TECLADO

En esta parte del programa se lleva a cabo la lectura del teclado esto se logra encendiendo la terminal asociada a cada renglón, al encenderse la terminal se realiza una lectura de las columnas para ver si alguna está encendida lo que equivaldría a una tecla oprimida, si en el renglón revisado no hay

una tecla pulsada se revisa el siguiente, realizando un poleo de todo el teclado hasta encontrar un tecla pulsada, cuando se detecta una tecla en esa condición se revisa siete veces el teclado para detectar si la tecla sigue pulsada y evitar señales de rebote que equivaldrían a una lectura errónea. Al detectar una tecla que está siendo oprimida se revisa una bandera que indica si hay un mensaje en formación, en cuyo caso el programa va a la rutina abierta anteriormente para seguir formando el mensaje, si no se estaba formando un mensaje el programa va a una rutina que asigna una tarea de acuerdo a la tecla oprimida. En esta rutina también se lleva a cabo el encendido del circuito de sonido mientras la tecla permanece oprimida así como el encendido de la iluminación del equipo, el diagrama de flujo de esta rutina se muestra en la fig. 6.2.3

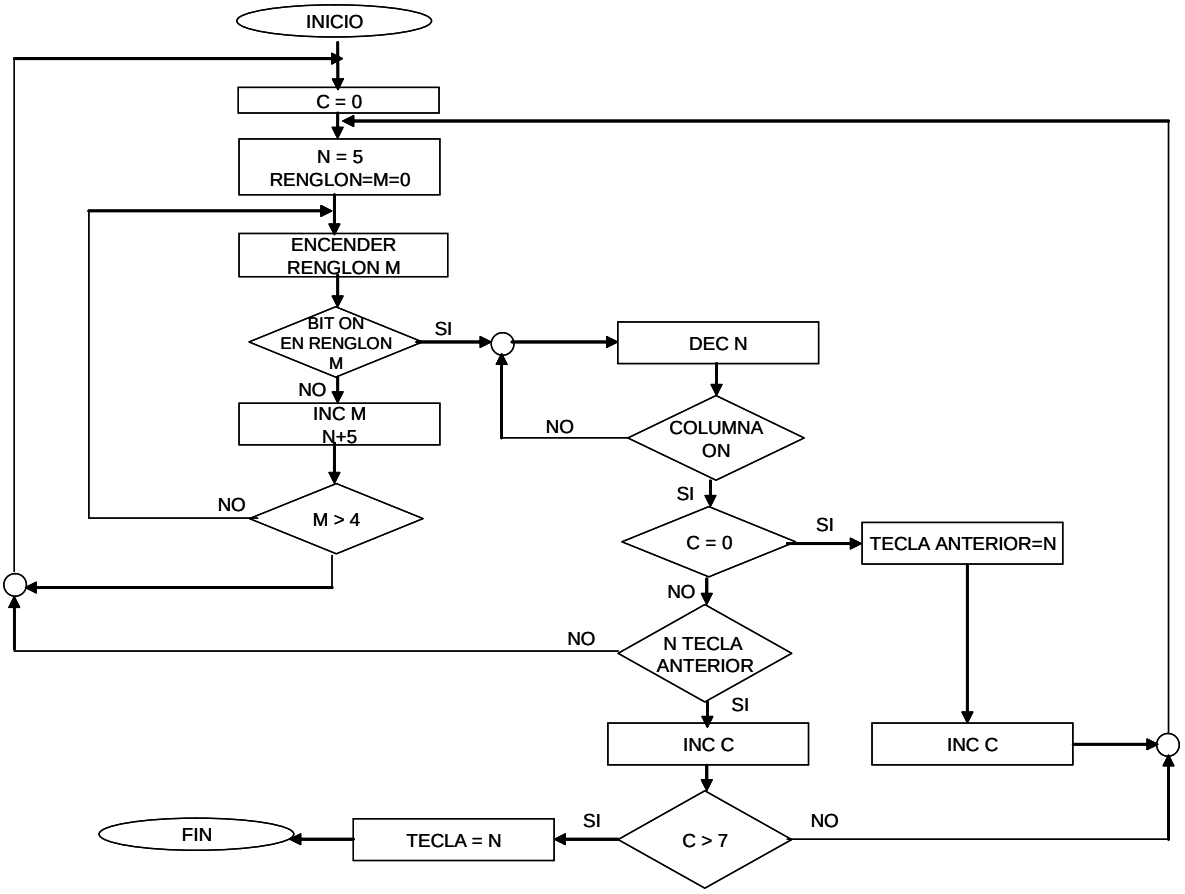


Fig. 6.2.3. Rutina de lectura de teclado.

VI.2.7. LIMPIAR UN CARÁCTER A LA IZQUIERDA

Esta rutina se encarga de borrar el carácter situado a la izquierda de la posición del cursor verificando si la posición del cursor es la primera columna antes de borrar el carácter para que se no pierda la dirección en la memoria de la pantalla. El diagrama de flujo de esta rutina se muestra en la fig. 6.2.4

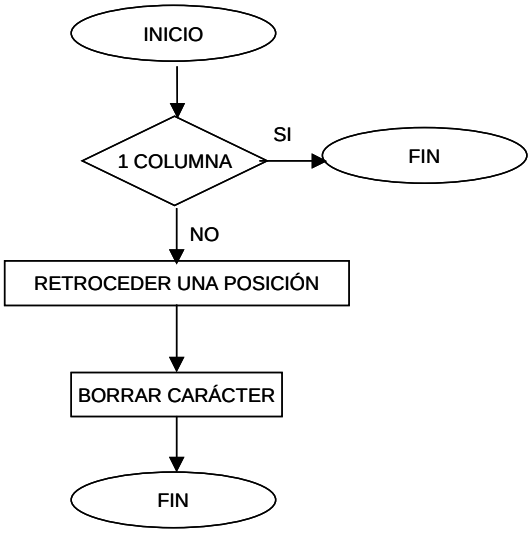


Fig. 6.2.4. Rutina de borrado de un carácter.

VI.2.8. TAREA *SHIFT*

Esta rutina que se muestra en la figura 6.2.5 tiene por objeto asignar una doble función a algunas teclas del equipo pulsando la tecla *"shift"* seguida de una tecla que tiene doble función, se tienen las siguientes posibilidades de tecla alterna:

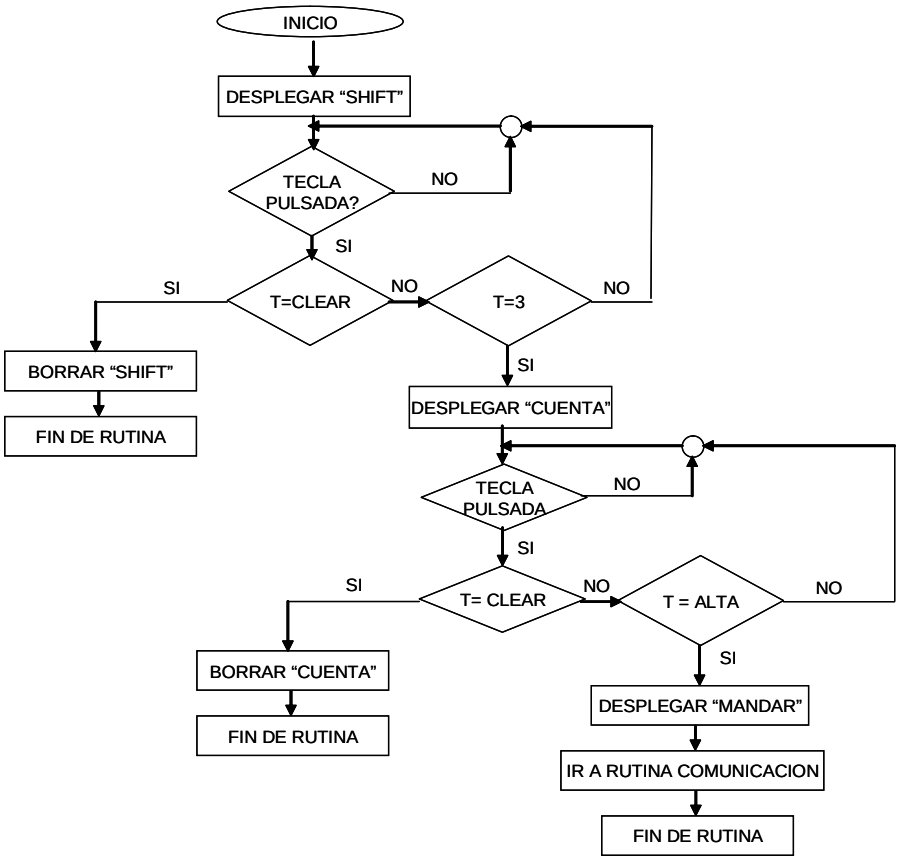


Fig. 6.2.5. Rutina para que una tecla tenga doble función.

VI.2.9. TIEMPO DE ESPERA PARA INICIALIZAR LA PANTALLA

Esta rutina sirve para proporcionarle a la pantalla los comandos necesarios en el tiempo que requiere para inicializar la pantalla, el diagrama de flujo de esta rutina se muestra en la fig. 6.2.6

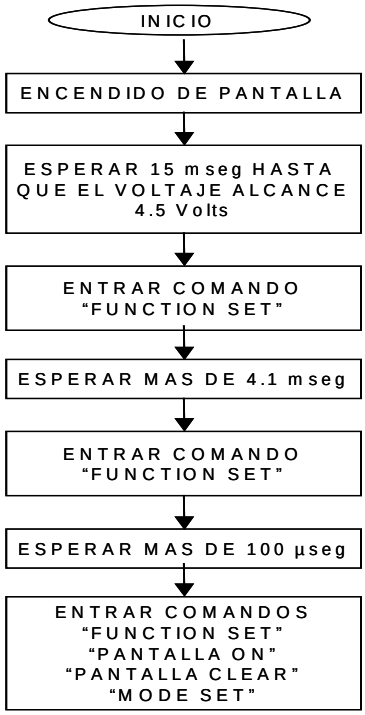


Fig. 6.2.6. Rutina de tiempo de espera para inicializar.

VI.2.10. ENVÍO DE DATOS A PANTALLA

Esta rutina lleva a cabo el envío de información a la pantalla ya sean datos para ser desplegados o comandos de control para la pantalla. Con esta rutina se proporcionan las señales *RS, R/W* y *E* requeridas por la pantalla para que la información enviada a las líneas de datos sea reconocida como un carácter a desplegar.

VI.2.11. RUTINA PRINCIPAL

Esta es la rutina que se ejecuta continuamente en el equipo y que se encarga de dirigir la información proveniente ya sea del teclado o de la *PC* que pone en marcha al sistema o lo configura.

El diagrama de flujo se muestra en la figura 6.2.7

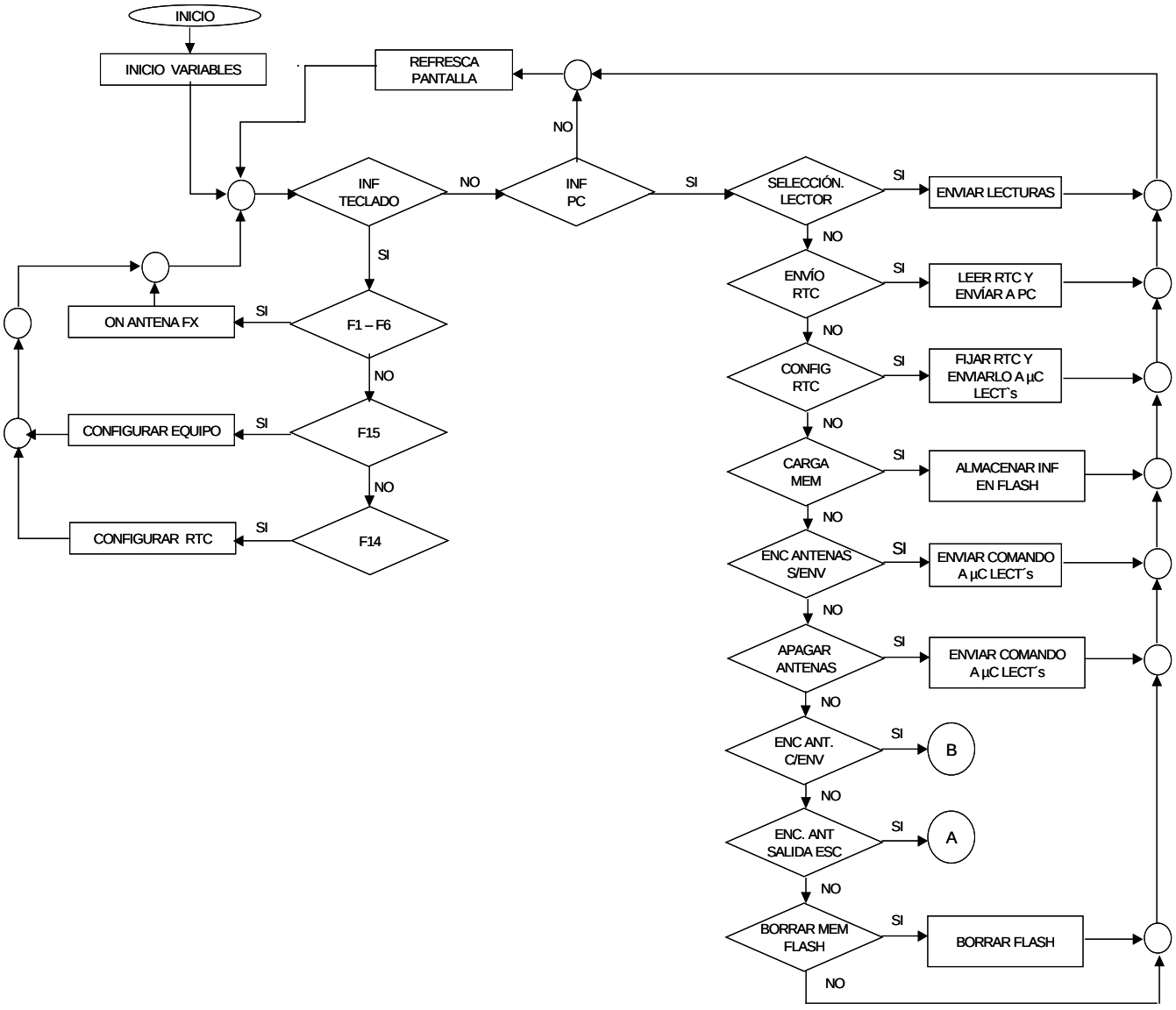


Fig. 6.2.7. Rutina principal Unidad Concentradora.

VI.2.12. RUTINA DE COMUNICACIONES

En esta rutina se lleva a cabo la inicialización de los puertos de comunicación *SCI* y *SPI* empleados en la comunicación con la *PC* y los DLCs respectivamente.

Se incluye también en esta rutina las funciones empleadas en la rutina principal.

VI.3 PROGRAMACIÓN DEL MICROCONTROLADOR DE LOS CONTROLADORES LÓGICOS DEDICADOS (DLCS)

Como se ha mencionado anteriormente, el microcontrolador de cada DLC tendrá las siguientes funciones:

- Comunicarse con la lectora de *TIRIS* para su activación y recepción de lecturas
- Etiquetar la información contenida en cada *transponder* asociando un tiempo de acuerdo con el reloj de tiempo real
- Almacenar cada una de las lecturas recibidas y etiquetadas
- Transmitir la información cuando le sea requerida por la unidad concentradora

De acuerdo con lo anterior, se presenta en la fig. 6.3.1 el diagrama de flujo con la rutina principal que ejecutará el microcontrolador de cada DLC

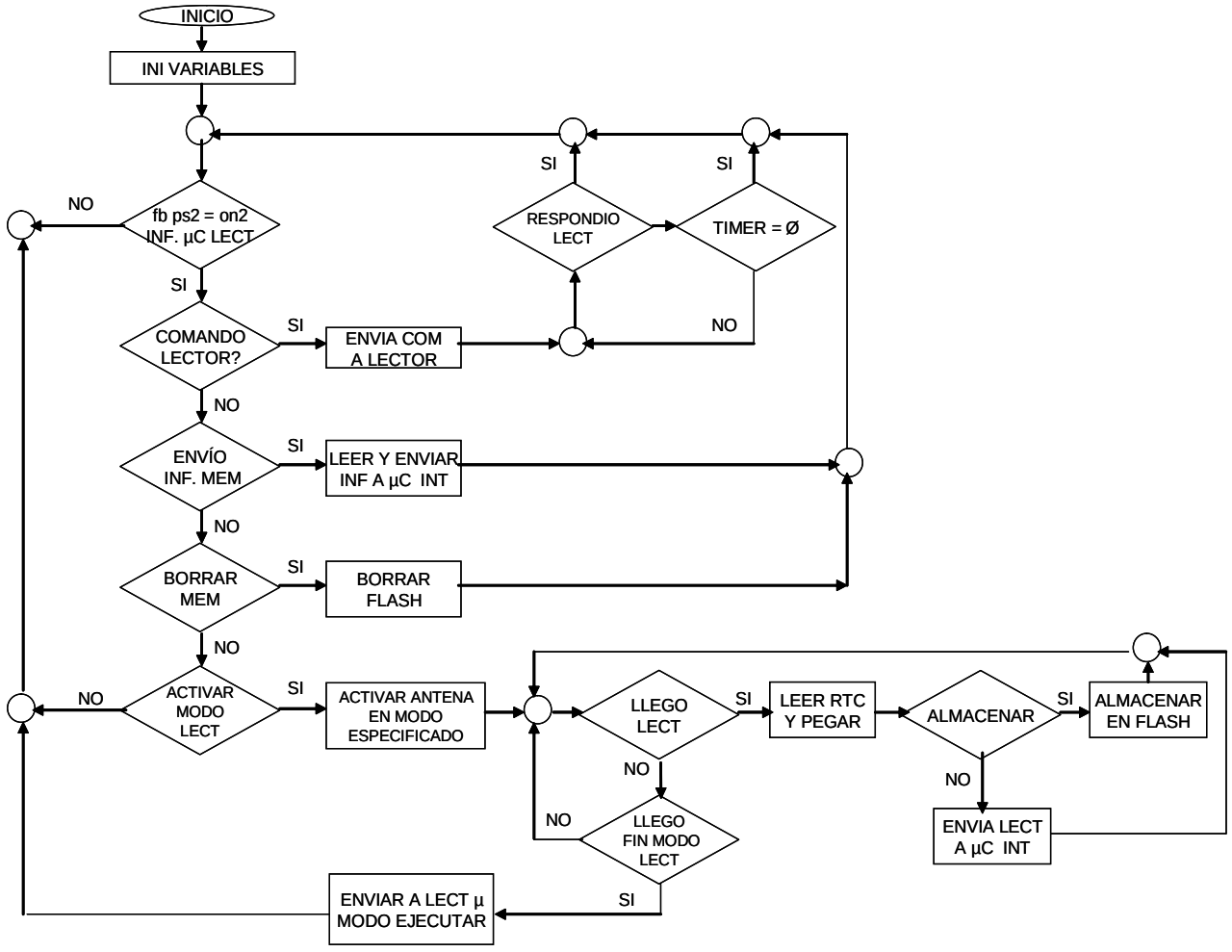


Fig. 6.3.1. Rutina principal del DLC.

Cabe señalar que las rutinas para la programación del microcontrolador de los DLCs son exactamente las mismas que para el microcontrolador de la unidad concentradora, a excepción de la rutina principal para los DLCs, misma que se mostró en la figura 6.3.1

RESULTADOS Y CONCLUSIONES

RESULTADOS Y CONCLUSIONES

Se realizaron pruebas por separado para determinar la eficiencia de lectura del conjunto antenas-*transponder*, así como pruebas de campo para el sistema completo SDR-1000.

Al conjunto antena-*transponder* se le realizaron dos pruebas:

1.- Eficiencia de lectura del *transpoder* con la antena colocada en diferentes medios.

Medio	**Tipo antena**	**Lectora**	***Alcance transponder***				**Resultados**
			Long	**Frecuencia**	**Teórico**	**Real**	
Tartán	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	75.1 cms	Aceptable
Asfalto	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.2 cms	Aceptable
Pasto	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	71.1 cms	Aceptable
Concreto armado	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	70.6cms	Aceptable

2.- Eficiencia de lectura del *transpoder* con diferentes orientaciones.

Medio: Tartán

Orientación	**Tipo antena**	**Lectora**	***Alcance transponder***				**Resultados**
			Long	**Frecuencia**	**Teórico**	**Real**	
Horizontal	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	75.0 cms	Aceptable
Vertical	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	75.3 cms	Aceptable
45°	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.9 cms	Aceptable
- 45°	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	75.2 cms	Aceptable

Medio: Asfalto

Orientación	**Tipo antena**	**Lectora**	***Alcance transponder***				**Resultados**
			Long	**Frecuencia**	**Teórico**	**Real**	
Horizontal	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.1 cms	Aceptable
Vertical	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.1 cms	Aceptable
45°	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.3 cms	Aceptable
- 45°	cuadro	RI-STU-251B	32 mm	134.2 kHz	100 cms	74.2 cms	Aceptable

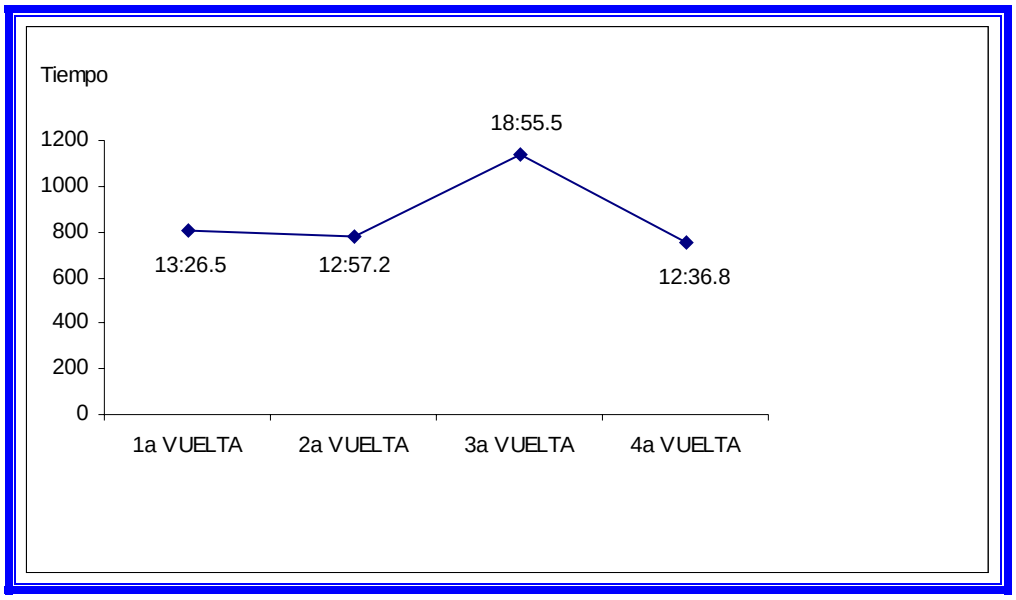
En las pruebas se aprecia que la máxima eficiencia de lectura que se alcanzó fueron en promedio 70 centímetros, alcance que se determinó suficiente con las pruebas de campo que se realizaron con el sistema completo.

Las pruebas de campo se realizaron en el parque viveros de Coyoacán, con apoyo de los corredores que asisten los fines de semana, se presentan los resultados.

LUGAR: PARQUE VIVEROS DE COYOACÁN
FECHA: 22 DE AGOSTO DEL 2006
NOMBRE: JOSÉ RENTERIA
CORREO ELECTRÓNICO:jrrenteria@hotmail.com
NUMERO DE CHIP: 222

ARRANQUE	07:56:44.0		
1a VUELTA	08:10:10.5	TIEMPO ENTRE ARRANQUE Y 1a VUELTA 13:26.5	13:26.5
2a VUELTA	08:23:07.7	TIEMPO ENTRE 1a Y 2a VUELTA	12:57.2
3a VUELTA	08:42:03.2	TIEMPO ENTRE 2a Y 3a VUELTA	18:55.5
4a VUELTA	08:54:40.0	TIEMPO ENTRE 2a Y 3a VUELTA	12:36.8
		TIEMPO TOTAL	00:57:56.0

SI SU RECORRIDO FUE POR LA RUTA MARCADA POR LA SEMARNAT (2.1 km)
RECORRIDO TOTAL: 08.4 km
VELOCIDAD PROMEDIO: 08.69 km/hr



Los resultados obtenidos de manera general, son como los reflejados en la tabla y gráfica anteriores, se tuvo una eficiencia excelente de lectura en corredores que incluso daban varias vueltas a la pista como se observa. Con la información registrada por el sistema SDR-1000 se pueden generar reportes como el que se muestra que son de gran utilidad para el atleta para estudiar su desempeño.

Finalmente, el sistema SDR-1000 fue probado en una competencia formal, los resultados son los siguientes:

Se presenta sólo una muestra de los datos registrados por el sistema, en ellos se puede apreciar como se registra el número de *transponder* y el tiempo asociado, así como incluso ya transferidos los datos a la *PC* se pueden ligar con el número y nombre de competidor

Lector	Tipo	Clase	Chip	Fecha	Valor	Horas	Corredor	Numero	Nombre
0	R	0000	0000000131760019	15.03.06	7094	00:11:49.4	135	1322	CENTENO JOSE
0	R	0000	0000000131760048	15.03.06	7682	00:12:48.2	295	1311	DAVID JOEL
0	R	0000	0000000131760030	15.03.06	7728	00:12:52.8	132	1361	GUERRERO OMAR
0	R	0000	0000000131760053	15.03.06	8414	00:14:01.4	293	1319	PEREZ TOMAS
0	R	0000	0000000131760048	15.03.06	8491	00:14:09.1	295	1311	DAVID JOEL
0	R	0000	0000000131760037	15.03.06	9168	00:15:16.8	294	1345	RIVERA JORGE
0	R	0000	0000000131760023	15.03.06	9171	00:15:17.1	129	1342	MORALES JONNATAN
0	R	0000	0000000131760010	15.03.06	9176	00:15:17.6	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760030	15.03.06	9346	00:15:34.6	132	1361	GUERRERO OMAR
0	R	0000	0000000131760019	15.03.06	9594	00:15:59.4	135	1322	CENTENO JOSE
0	R	0000	0000000131760010	15.03.06	9936	00:16:33.6	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760048	15.03.06	10127	00:16:52.7	295	1311	DAVID JOEL
0	R	0000	0000000131760030	15.03.06	10178	00:16:57.8	132	1361	GUERRERO OMAR
0	R	0000	0000000131760034	15.03.06	10567	00:17:36.7	133	1341	VALLE GERARDO
0	R	0000	0000000131760053	15.03.06	10704	00:17:50.4	293	1319	PEREZ TOMAS
0	R	0000	0000000131760048	15.03.06	10968	00:18:16.8	295	1311	DAVID JOEL
0	R	0000	0000000131760037	15.03.06	11462	00:19:06.2	294	1345	RIVERA JORGE
0	R	0000	0000000131760053	15.03.06	12229	00:20:22.9	293	1319	PEREZ TOMAS
0	R	0000	0000000131760037	15.03.06	13004	00:21:40.4	294	1345	RIVERA JORGE
0	R	0000	0000000131760053	15.03.06	13007	00:21:40.7	293	1319	PEREZ TOMAS
0	R	0000	0000000131760010	15.03.06	13013	00:21:41.3	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760019	15.03.06	13062	00:21:46.2	135	1322	CENTENO JOSE
0	R	0000	0000000131760030	15.03.06	13464	00:22:26.4	132	1361	GUERRERO OMAR
0	R	0000	0000000131760053	15.03.06	13776	00:22:57.6	293	1319	PEREZ TOMAS
0	R	0000	0000000131760030	15.03.06	15130	00:25:13.0	132	1361	GUERRERO OMAR
0	R	0000	0000000131760019	15.03.06	15722	00:26:12.2	135	1322	CENTENO JOSE
0	R	0000	0000000131760030	15.03.06	16805	00:28:00.5	132	1361	GUERRERO OMAR
0	R	0000	0000000131760023	15.03.06	16845	00:28:04.5	129	1342	MORALES JONNATAN
0	R	0000	0000000131760030	15.03.06	17655	00:29:25.5	132	1361	GUERRERO OMAR
0	R	0000	0000000131760010	15.03.06	18529	00:30:52.9	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760037	15.03.06	19034	00:31:43.4	294	1345	RIVERA JORGE
0	R	0000	0000000131760053	15.03.06	19079	00:31:47.9	293	1319	PEREZ TOMAS
0	R	0000	0000000131760030	15.03.06	19365	00:32:16.5	132	1361	GUERRERO OMAR
0	R	0000	0000000131760019	15.03.06	20977	00:34:57.7	135	1322	CENTENO JOSE
0	R	0000	0000000131760024	15.03.06	959	00:01:35.9	159	1344	CRUZ ARACELI
0	R	0000	0000000131760050	15.03.06	978	00:01:37.8	309	1352	JUAREZ JUANA
0	R	0000	0000000131760063	15.03.06	1795	00:02:59.5	*	*	*
0	R	0000	0000000131760027	15.03.06	1927	00:03:12.7	163	1336	GONZALEZ ELIZABEDTH
0	R	0000	0000000131760024	15.03.06	1934	00:03:13.4	159	1344	CRUZ ARACELI
0	R	0000	0000000131760016	15.03.06	1937	00:03:13.7	164	1322	PLATA ARACELI
0	R	0000	0000000131760050	15.03.06	1987	00:03:18.7	309	1352	JUAREZ JUANA
0	R	0000	0000000131760063	15.03.06	2734	00:04:33.4	*	*	*
0	R	0000	0000000131760024	15.03.06	2880	00:04:48.0	159	1344	CRUZ ARACELI
0	R	0000	0000000131760034	15.03.06	767	00:01:16.7	133	1341	VALLE GERARDO
0	R	0000	0000000131760019	15.03.06	785	00:01:18.5	135	1322	CENTENO JOSE
0	R	0000	0000000131760053	15.03.06	1516	00:02:31.6	293	1319	PEREZ TOMAS
0	R	0000	0000000131760010	15.03.06	1520	00:02:32.0	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760030	15.03.06	1539	00:02:33.9	132	1361	GUERRERO OMAR
0	R	0000	0000000131760034	15.03.06	2263	00:03:46.3	133	1341	VALLE GERARDO
0	R	0000	0000000131760053	15.03.06	2274	00:03:47.4	293	1319	PEREZ TOMAS
0	R	0000	0000000131760048	15.03.06	2283	00:03:48.3	295	1311	DAVID JOEL
0	R	0000	0000000131760053	15.03.06	3027	00:05:02.7	293	1319	PEREZ TOMAS
0	R	0000	0000000131760030	15.03.06	3048	00:05:04.8	132	1361	GUERRERO OMAR
0	R	0000	0000000131760019	15.03.06	3088	00:05:08.8	135	1322	CENTENO JOSE
0	R	0000	0000000131760037	15.03.06	3780	00:06:18.0	294	1345	RIVERA JORGE
0	R	0000	0000000131760010	15.03.06	3787	00:06:18.7	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760048	15.03.06	3795	00:06:19.5	295	1311	DAVID JOEL
0	R	0000	0000000131760030	15.03.06	3822	00:06:22.2	132	1361	GUERRERO OMAR
0	R	0000	0000000131760037	15.03.06	4543	00:07:34.3	294	1345	RIVERA JORGE
0	R	0000	0000000131760034	15.03.06	4558	00:07:35.8	133	1341	VALLE GERARDO
0	R	0000	0000000131760048	15.03.06	4565	00:07:36.5	295	1311	DAVID JOEL
0	R	0000	0000000131760030	15.03.06	4597	00:07:39.7	132	1361	GUERRERO OMAR
0	R	0000	0000000131760053	15.03.06	5315	00:08:51.5	293	1319	PEREZ TOMAS
0	R	0000	0000000131760034	15.03.06	5374	00:08:57.4	133	1341	VALLE GERARDO
0	R	0000	0000000131760010	15.03.06	6084	00:10:08.4	130	1346	RAMIREZ MARCO
0	R	0000	0000000131760048	15.03.06	6095	00:10:09.5	295	1311	DAVID JOEL
0	R	0000	0000000131760030	15.03.06	6148	00:10:14.8	132	1361	GUERRERO OMAR
0	R	0000	0000000131760034	15.03.06	6227	00:10:22.7	133	1341	VALLE GERARDO
0	R	0000	0000000131760037	15.03.06	6855	00:11:25.5	294	1345	RIVERA JORGE

Conclusiones:

El sistema SDR-1000 cumplió las expectativas:

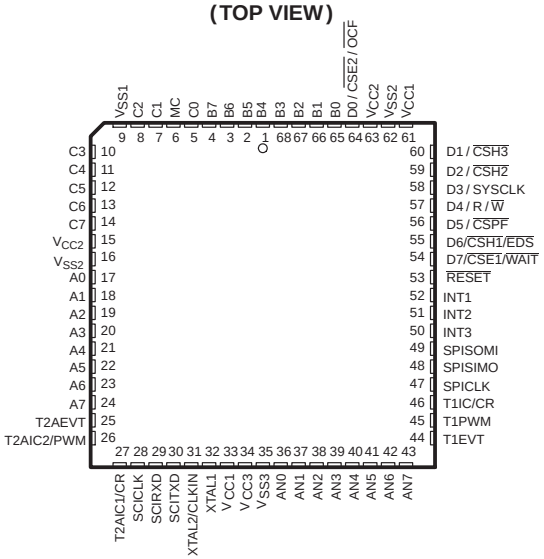
- Se logró un sistema sumamente eficiente de lecturas al paso de un contingente importante de atletas.
- El sistema registra la información necesaria para entregar resultados confiables y en tiempo real.
- Cumple con los requerimientos establecidos por la Federación de Atletismo, proporciona resultados hasta décimas de segundo y no interfiere ni obliga acción que afecte el desarrollo natural de la competencia
- El sistema ofrece autonomía eléctrica en caso de no contar con alimentación en 127 V.
- Se obtuvo un equipo completamente transportable y robusto para soportar de ser el caso el traslado en camión, avión, etcétera.
- El estudio de costo-beneficio reflejó un proyecto viable, dado que la posibilidad de comercializarlo mediante el modo de renta, ofrecería al cliente precios por debajo de los que se manejan actualmente en el mercado y al dueño del equipo la posibilidad de recuperar la inversión en tiempos muy razonables con ganancias de un principio.

APÉNDICE A

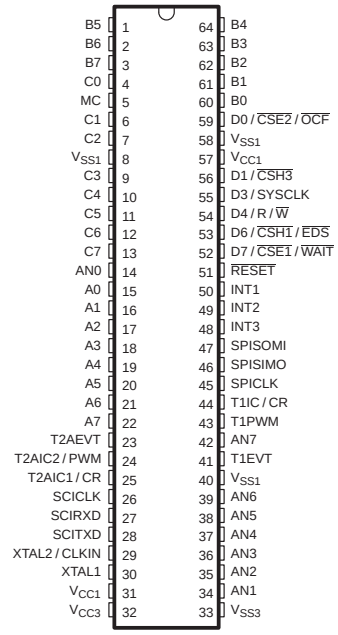
HOJAS DE ESPECIFICACIONES TÉCNICAS

- **CMOS/EEPROM/EPROM Technologies on a Single Device**
  - **Mask-ROM Devices for High-Volume Production**
  - **One-Time-Programmable (OTP) EPROM Devices for Low-Volume Production**
  - **Reprogrammable EPROM Devices for Prototyping Purposes**
- **Internal System Memory Configurations**
  - **On-Chip Program Memory Versions**
    - **ROM: 4K to 48K Bytes**
    - **EPROM: 16K to 48K Bytes**
    - **ROM-less**
  - **Data EEPROM: 256 or 512 Bytes**
  - **Static RAM: 256 to 3.5K Bytes**
  - **External Memory/Peripheral Wait States**
  - **Precoded External Chip-Select Outputs in Microcomputer Mode**
- **Flexible Operating Features**
  - **Low-Power Modes: STANDBY and HALT**
  - **Commercial, Industrial, and Automotive Temperature Ranges**
  - **Clock Options**
    - **Divide-by-4 (0.5 MHz – 5 MHz SYSCLK)**
    - **Divide-by-1 (2 MHz – 5 MHz SYSCLK)**
    - **Phase-Locked Loop (PLL)**
  - **Supply Voltage (V_{CC}): 5 V ± 10%**
- **Eight-Channel 8-Bit Analog-to-Digital Converter 1 (ADC1)**
- **Two 16-Bit General-Purpose Timers**
- **On-Chip 24-Bit Watchdog Timer**
- **Two Communication Modules**
  - **Serial Communications Interface 1 (SCI1)**
  - **Serial Peripheral Interface (SPI)**
- **Flexible Interrupt Handling**
- **TMS370 Series Compatibility**
- **CMOS/Package/TTL-Compatible I/O Pins**
  - **64-Pin Plastic and Ceramic Shrink Dual-In-Line Packages/44 Bidirectional, 9 Input Pins**
  - **68-Pin Plastic and Ceramic Leaded Chip Carrier Packages/46 Bidirectional, 9 Input Pins**
  - **All Peripheral Function Pins Are Software Configurable for Digital I/O**

FN/FZ PACKAGE
(TOP VIEW)



JN/NM PACKAGE
(TOP VIEW)



- **Workstation/PC-Based Development System**
  - **C Compiler and C Source Debugger**
  - **Real-Time In-Circuit Emulation**
  - **Extensive Breakpoint/Trace Capability**
  - **Software Performance Analysis**
  - **Multi-Window User Interface**
  - **Microcontroller Programmer**

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

Pin Descriptions

PIN				I/O†	DESCRIPTION‡
NAME	ALTERNATE FUNCTION	SDIP (64)	LCC (68)		
A0	DATA0	15	17	I/O	Single-chip mode: Port A is a general-purpose bidirectional I/O port. Expansion mode: Port A can be individually programmed as the external bidirectional data bus (DATA0–DATA7).
A1	DATA1	16	18		
A2	DATA2	17	19		
A3	DATA3	18	20		
A4	DATA4	19	21		
A5	DATA5	20	22		
A6	DATA6	21	23		
A7	DATA7	22	24		
B0	ADDR0	60	65	I/O	Single-chip mode: Port B is a general-purpose bidirectional I/O port. Expansion mode: Port B can be individually programmed as the low-order address output bus (ADDR0–ADDR7).
B1	ADDR1	61	66		
B2	ADDR2	62	67		
B3	ADDR3	63	68		
B4	ADDR4	64	1		
B5	ADDR5	1	2		
B6	ADDR6	2	3		
B7	ADDR7	3	4		
C0	ADDR8	4	5	I/O	Single-chip mode: Port C is a general-purpose bidirectional I/O port. Expansion mode: Port C can be individually programmed as the high-order address output bus (ADDR8–ADDR15).
C1	ADDR9	6	7		
C2	ADDR10	7	8		
C3	ADDR11	9	10		
C4	ADDR12	10	11		
C5	ADDR13	11	12		
C6	ADDR14	12	13		
C7	ADDR15	13	14		
INT1	NMI	50	52	I	External (nonmaskable or maskable) interrupt/general-purpose input pin
INT2	—	49	51	I/O	External maskable interrupt input/general-purpose bidirectional pin
INT3	—	48	50	I/O	External maskable interrupt input/general-purpose bidirectional pin
AN0	E0	14	36	I	ADC1 analog input (AN0–AN7) or positive reference pins (AN1–AN7) Port E can be individually programmed as general-purpose input pins if not used as ADC1 analog input or positive reference input.
AN1	E1	34	37		
AN2	E2	35	38		
AN3	E3	36	39		
AN4	E4	37	40		
AN5	E5	38	41		
AN6	E6	39	42		
AN7	E7	42	43		
V_{CC3}		32	34		ADC1 positive-supply voltage and optional positive-reference input pin
V_{SS3}		33	35		ADC1 ground reference pin
$\overline{\text{RESET}}$		51	53	I/O	System reset bidirectional pin. $\overline{\text{RESET}}$, as an input, initializes the microcontroller; as open-drain output, $\overline{\text{RESET}}$ indicates an internal failure was detected by the watchdog or oscillator fault circuit.
MC		5	6	I	Mode control (MC) pin. MC enables EEPROM write-protection override (WPO) mode, also EPROM V_{PP}.
XTAL2/CLKIN		29	31	I	Internal oscillator crystal input/external clock source input
XTAL1		30	32	O	Internal oscillator output for crystal
V_{CC1}		31, 57	33, 61		Positive supply voltage
V_{CC2}		—	15,63		Positive supply voltage

† I = input, O = output

‡ Ports A, B, C, and D can be configured only as general-purpose I/O pins. Also, port D3 can be configured as SYSCLK.

Pin Descriptions (Continued)

PIN				I/O†	DESCRIPTION‡
NAME	ALTERNATE FUNCTION	SDIP (64)	LCC (68)		
V_{SS1}		8, 58,40	9		Ground reference for digital logic
V_{SS2}		—	16,62		Ground reference for digital I/O logic
	FUNCTION			I/O	Single-chip mode: Port D is a general-purpose bidirectional I/O port. Each of the port D pins can be individually configured as a general-purpose I/O pin, primary memory control signal (function A), or secondary memory control signal (function B). All chip selects are independent and can be used for memory bank switching. Refer to Table 1 for function A memory accesses.
	A / B				
D0	$\overline{CSE2}$ / $\overline{OCF}$	59	64		I/O pin A: Chip select eighth output 2 goes low during memory accesses I/O pin B: Opcode fetch goes low during the opcode fetch memory cycle.
D1	$\overline{CSH3}$ / —	56	60		I/O pin A: Chip select half output 3 goes low during memory accesses. I/O pin B: Reserved
D2	$\overline{CSH2}$ / —	—	59		I/O pin A: Chip select half output 2 goes low during memory accesses. I/O pin B: Reserved
D3	SYSCLK / SYSCLK	55	58		I/O pin A, B: Internal clock signal is 1/1 (PLL) or 1/4 XTAL2/CLKIN frequency.
D4	R/$\overline{W}$ / R/$\overline{W}$	54	57		I/O pin A, B: Read/write output pin
D5	$\overline{CSPF}$ / —	—	56		I/O pin A: Chip select peripheral output for peripheral file goes low during memory accesses. I/O pin B: Reserved
D6	$\overline{CSH1}$ / $\overline{EDS}$	53	55		I/O pin A: Chip select half output 1 goes low during memory accesses. I/O pin B: External data strobe output goes low during memory accesses from external memory and has the same timings as the five chip selects.
D7	$\overline{CSE1}$ / $\overline{WAIT}$	52	54		I/O pin A: Chip select eighth output goes low during memory accesses. I/O pin B: Wait input pin extends bus signals.
SCITXD	SCIIO1	28	30	I/O	SCI transmit data output pin/general-purpose bidirectional pin (see Note 1)
SCIRXD	SCIIO2	27	29		SCI receive data input pin/general-purpose bidirectional pin
SCICLK	SCIIO3	26	28		SCI bidirectional serial clock pin/general-purpose bidirectional pin
T1IC/CR	T1IO1	44	46	I/O	Timer1 input capture/counter reset input pin/general-purpose bidirectional pin
T1PWM	T1IO2	43	45		Timer1 pulse-width-modulation (PWM) output pin/general-purpose bidirectional pin
T1EVT	T1IO3	41	44		Timer1 external event input pin/general-purpose bidirectional pin
T2AIC1/CR	T2AIO1	25	27	I/O	Timer2A input capture 1/counter reset input pin/general-purpose bidirectional pin
T2AIC2/PWM	T2AIO2	24	26		Timer2A input capture 2/PWM output pin/general-purpose bidirectional pin
T2AEVT	T2AIO3	23	25		Timer2A external event input pin/general-purpose bidirectional pin
SPISOMI	SPIIO1	47	49	I/O	SPI slave output pin, master input pin/general-purpose bidirectional pin
SPISIMO	SPIIO2	46	48		SPI slave input pin, master output pin/general-purpose bidirectional pin
SPICLK	SPIIO3	45	47		SPI bidirectional serial clock pin/general-purpose bidirectional pin

† I = input, O = output

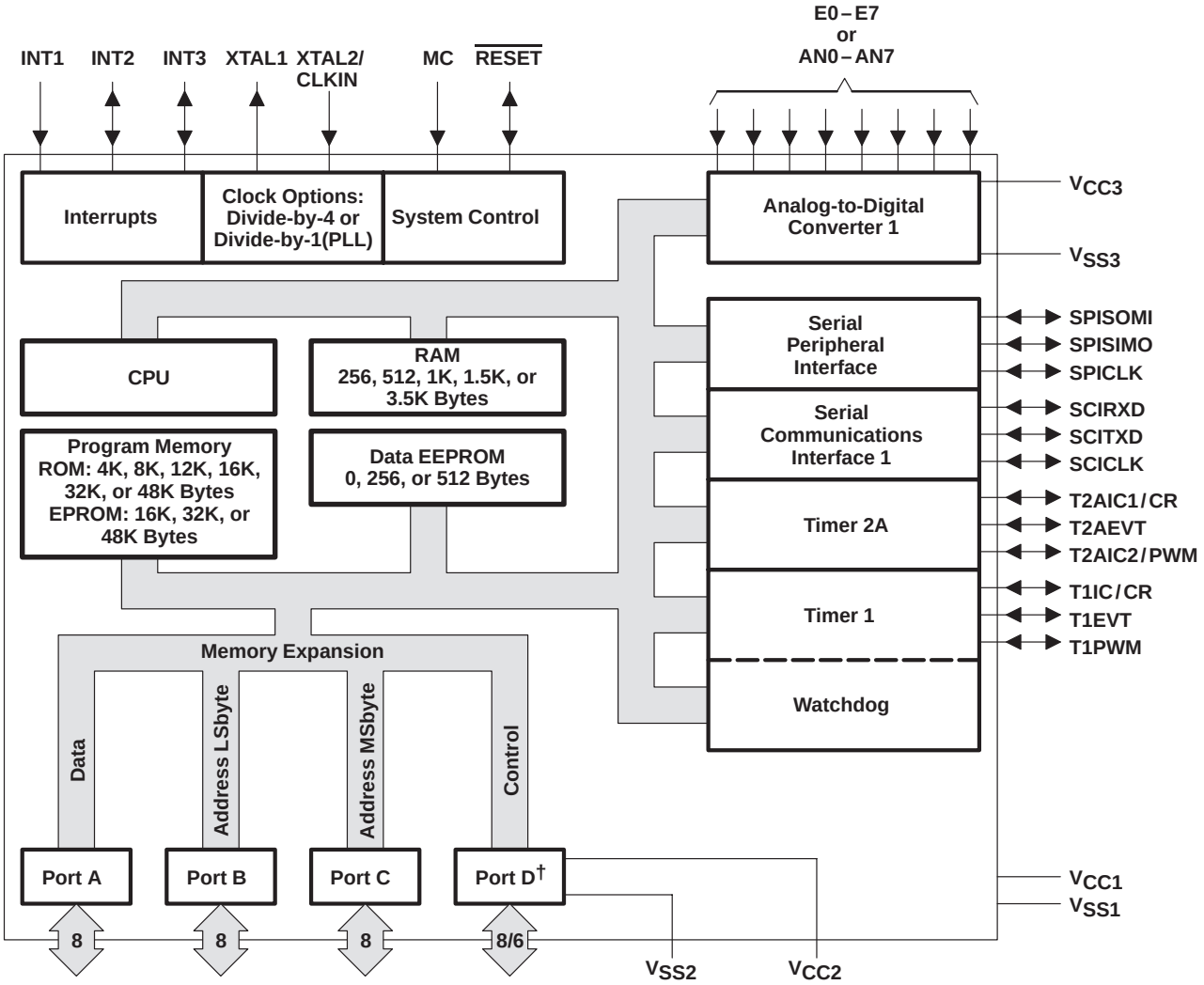
‡ Ports A, B, C, and D can be configured only as general-purpose I/O pins. Port D3 also can be configured as SYSCLK.

NOTE 1: The three-pin configuration SCI is referred to as SCI1.

Table 1. Function A: Memory Accesses Locations for ‘x5x Devices

FUNCTION A	‘X50, ‘X52, ‘X53, AND ‘X56	‘X58	‘X59
$\overline{CSEx}$	2000h – 3FFFh (8K bytes)	A000h – BFFFh (8K bytes)	E000h – EFFFh (4K bytes)
$\overline{CSHx}$	8000h – FFFFh (32K bytes)	C000h – FFFFh (16K bytes)	F000h – FFFFh (4K bytes)
$\overline{CSPF}$	10C0h – 10FFh (64 bytes)	10C0h – 10FFh (64 bytes)	10C0h – 10FFh (64 bytes)

functional block diagram



† For the 64-pin devices, there are only six pins for port D.

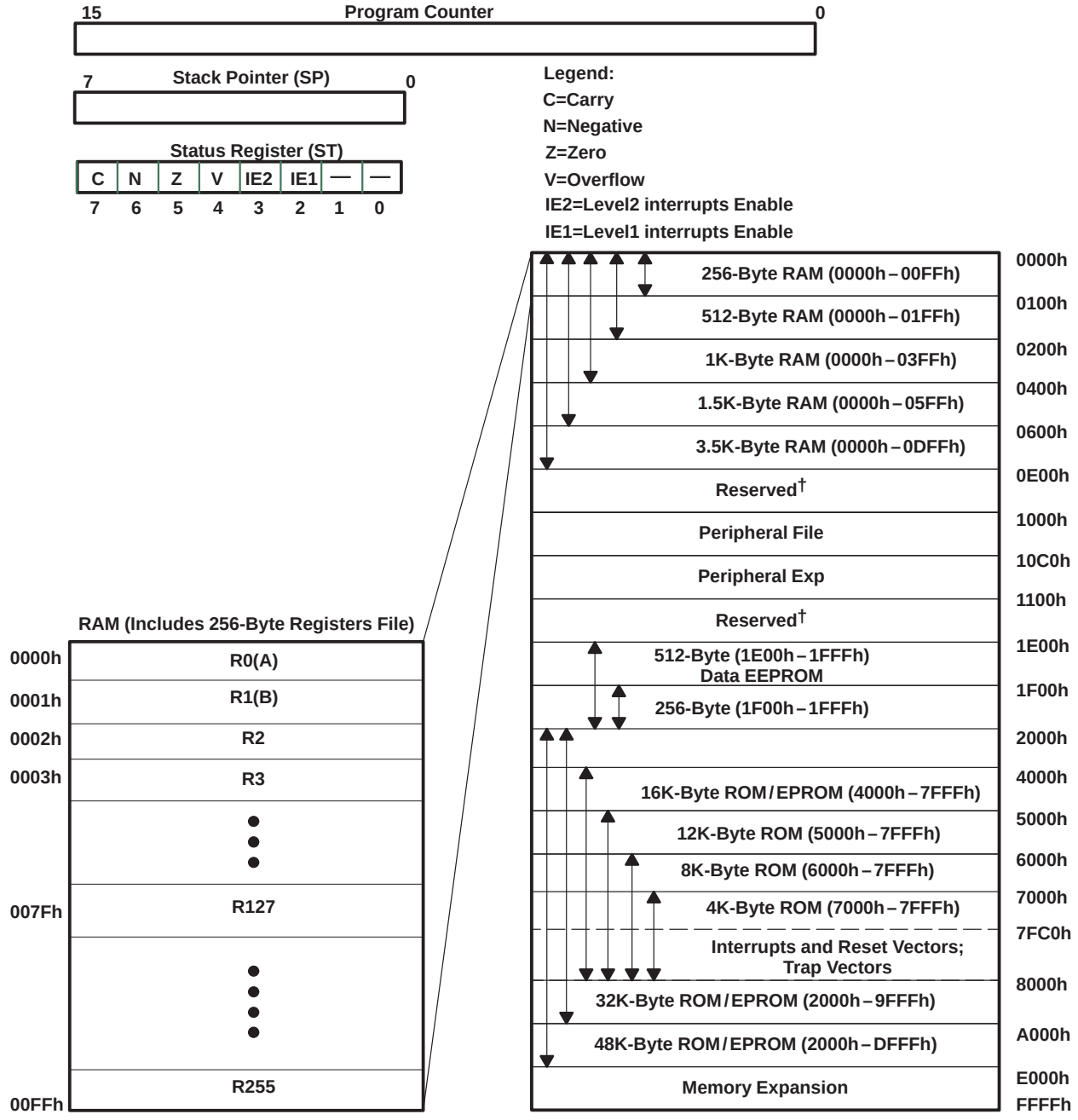
description

The TMS370Cx5x family of single-chip 8-bit microcontrollers provides cost-effective real-time system control through integration of advanced peripheral function modules and various on-chip memory configurations. The TMS370Cx5x family presently consists of twenty-one devices which are grouped into seven main sub-families: the TMS370Cx50, TMS370Cx52, TMS370Cx53, TMS370Cx56, TMS370Cx58, TMS370Cx59, and SE370C75x.

The TMS370Cx5x family of devices is implemented using high-performance silicon-gate CMOS EPROM and EEPROM technologies. The low-operating power, wide-operating temperature range, and noise immunity of CMOS technology, coupled with the high performance and extensive on-chip peripheral functions, make the TMS370Cx5x devices attractive in system designs for automotive electronics, industrial motor control, computer peripheral control, telecommunications, and consumer application. Table 2 provides a memory configuration overview of the TMS370Cx5x devices.

CPU

The CPU used on TMS370Cx5x devices is the high-performance 8-bit TMS370 CPU module. The ’x5x implements an efficient register-to-register architecture that eliminates the conventional accumulator bottleneck. The complete ’x5x instruction set is summarized in Table 23. Figure 1 illustrates the CPU registers and memory blocks.



† Reserved means the address space is reserved for future expansion.

Figure 1. Programmer’s Model

RAM/register file (RF)

Locations within RAM address space can serve as either register file or general-purpose read/write memory, program memory, or stack instructions. The TMS370Cx50 and TMS370Cx52 devices contain 256 bytes of internal RAM, mapped beginning at location 0000h and continuing through location 00FFh which is shown in Table 5 along with other ’x5x devices.

Table 5. RAM Memory Map

	‘x50 and ‘x52	‘x56	‘x58	‘x53	‘x59
RAM Size	256 Bytes	512 Bytes	1K Bytes	1.5K Bytes	3.5K Bytes
Memory Mapped	0000h – 00FFh	0000h – 01FFh	0000h – 03FFh	0000h – 05FFh	0000h – 0DFFh

The first 256 bytes of RAM (0000h – 00FFh) are register files, R0 through R255 (see Figure 1). The first two registers, R0 and R1, are also called register A and B, respectively. Some instructions implicitly use register A or B; for example, the instruction LDSP (load SP) assumes that the value to be loaded into the stack pointer is contained in register B. Registers A and B are the only registers cleared on reset.

peripheral file (PF)

The TMS370Cx5x control registers contain all the registers necessary to operate the system and peripheral modules on the device. The instruction set includes some instructions that access the PF directly. These instructions designate the register by the number of the PF relative to 1000h, preceded by P0 for a hexadecimal designator or by P for a decimal designator. For example, the system control register 0 (SCCR0) is located at address 1010h; its peripheral file hexadecimal designator is P010, and its decimal designator is P16. Table 6 shows the TMS370Cx5x peripheral files.

Table 6. TMS370Cx5x Peripheral File Address map

ADDRESS RANGE	PERIPHERAL FILE DESIGNATOR	DESCRIPTION
1000h–100Fh	P000–P00F	Reserved for factory test
1010h–101Fh	P010–P01F	System and EEPROM/EPROM control registers
1020h–102Fh	P020–P02F	Digital I/O port control registers
1030h–103Fh	P030–P03F	Serial peripheral interface registers
1040h–104Fh	P040–P04F	Timer 1 registers
1050h–105Fh	P050–P05F	Serial communication interface 1 registers
1060h–106Fh	P060–P06F	Timer 2A registers
1070h–107Fh	P070–P07F	Analog-to-digital converter 1 registers
1080h–10BFh	P080–P0BF	Reserved
10C0h–10FFh	P0C0–P0FF	External peripheral control

data EEPROM

The TMS370Cx56 devices contain 512 bytes of data EEPROM, which are memory mapped beginning at location 1E00h and continuing through location 1FFFh as shown in Table 7 along with other ‘x5x devices.

Table 7. Data-EEPROM Memory Map

	‘x50, ‘x52, ‘x58, and ‘x59	‘x56	‘X53
Data-EEPROM Size	256 Bytes	512 Bytes	None
Memory Mapped	1F00h–1FFFh	1E00h–1FFFh	None

PRELIMINARY

AMD

Am29F040B

4 Megabit (512 K x 8-Bit)
CMOS 5.0 Volt-only, Uniform Sector Flash Memory

Distinctive Characteristics

- **5.0 V ± 10% for read and write operations**
  - — Minimizes system level power requirements
- **Manufactured on 0.35 µm process technology**
  - — Compatible with 0.5 µm Am29F040 device
- **High performance**
  - — Access times as fast as 55 ns
- **Low power consumption**
  - — 20 mA typical active read current
  - — 30 mA typical program/erase current
  - — 1 µA typical standby current (standard access time to active mode)
- **Flexible sector architecture**
  - — 8 uniform sectors of 64 Kbytes each
  - — Any combination of sectors can be erased
  - — Supports full chip erase
  - — Sector protection:

    A hardware method of locking sectors to prevent any program or erase operations within that sector
- **Embedded Algorithms**
  - — Embedded Erase algorithm automatically preprograms and erases the entire chip or any combination of designated sectors
  - — Embedded Program algorithm automatically writes and verifies bytes at specified addresses
- **Minimum 1,000,000 program/erase cycles per sector guaranteed**
- **Package options**
  - — 32-pin PLCC, TSOP, or PDIP
- **Compatible with JEDEC standards**
  - — Pinout and software compatible with single-power-supply Flash standard
  - — Superior inadvertent write protection
- **Data# Polling and toggle bits**
  - — Provides a software method of detecting program or erase cycle completion
- **Erase Suspend/Erase Resume**
  - — Suspends a sector erase operation to read data from, or program data to, a non-erasing sector, then resumes the erase operation

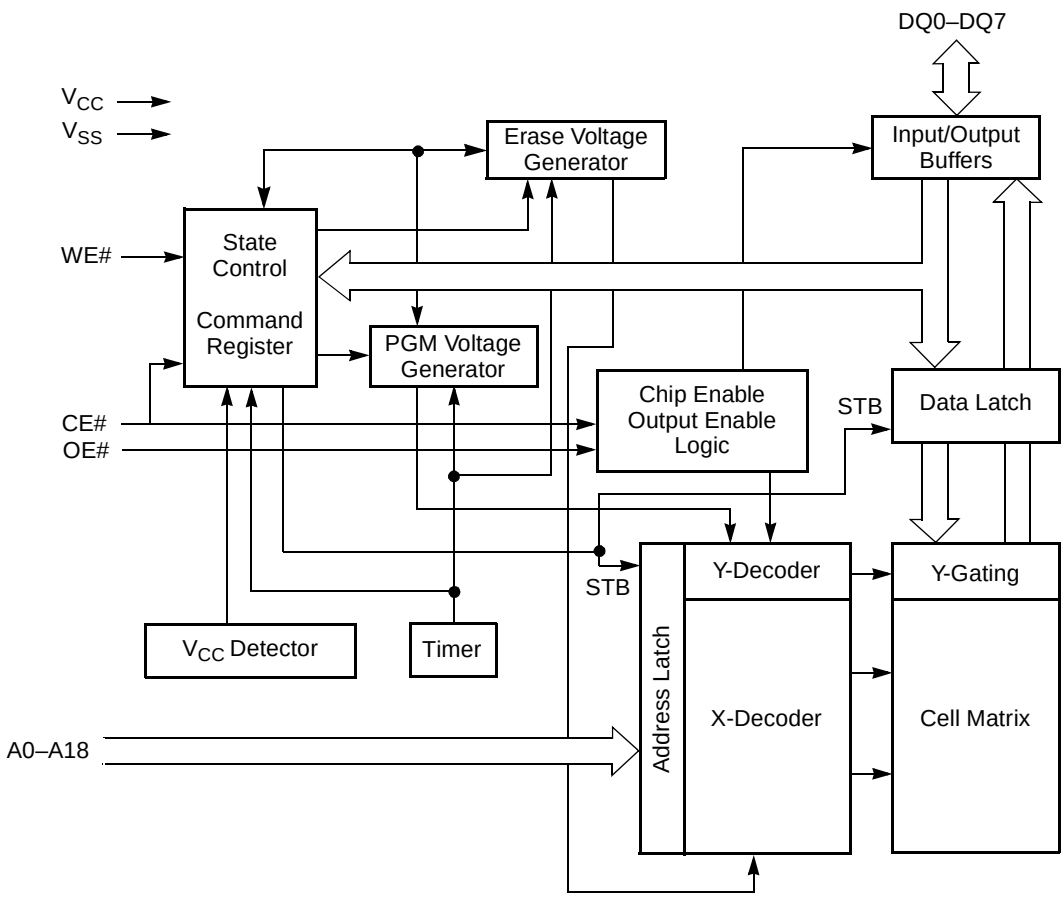
Publication# **21445** Rev: **B** Amendment/**+2**
Issue Date: **April 1998**

PRODUCT SELECTOR GUIDE

Family Part Number		Am29F040B				
Speed Option	V_{CC} = 5.0 V ± 5%	**-55**				
	V_{CC} = 5.0 V ± 10%		**-70**	**-90**	**-120**	**-150**
Max access time, ns (t_{ACC})		55	70	90	120	150
Max CE# access time, ns (t_{CE})		55	70	90	120	150
Max OE# access time, ns (t_{OE})		25	30	35	50	55

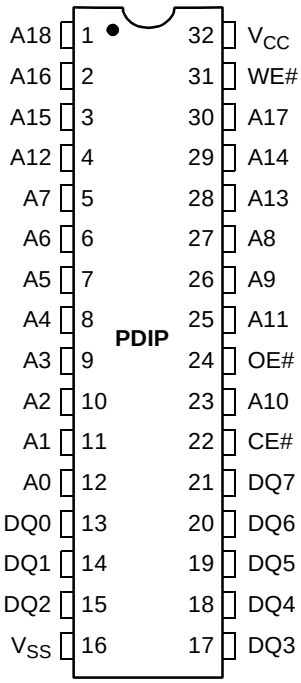
Note: *See the "AC Characteristics" section for more information.*

BLOCK DIAGRAM

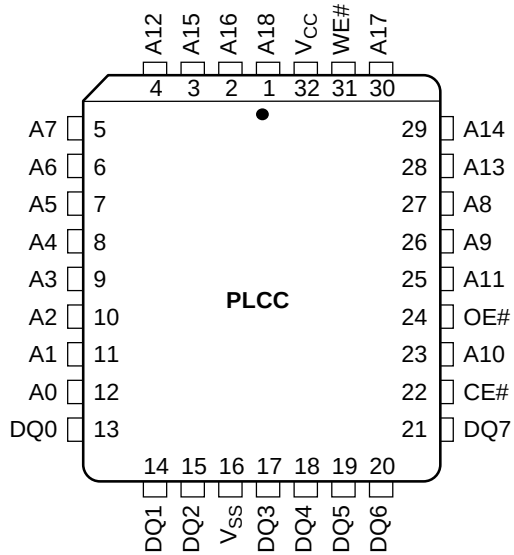


21445B-1

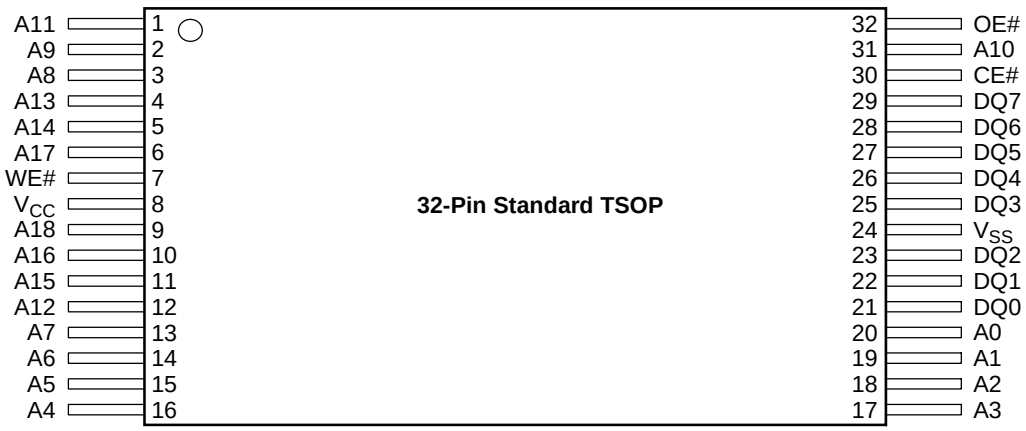
CONNECTION DIAGRAMS

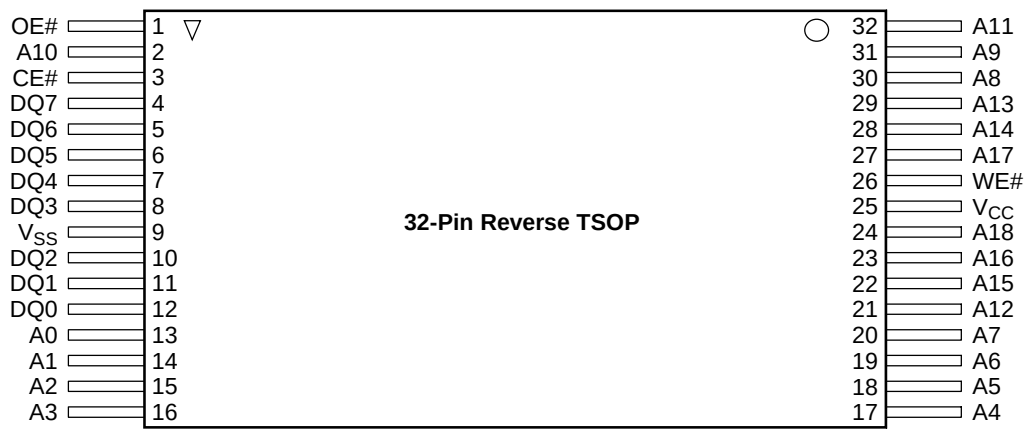


21445B-2



21445B-3





21445B-4

PIN CONFIGURATION

A0–A18 = Address Inputs

DQ0–DQ7 = Data Input/Output

CE# = Chip Enable

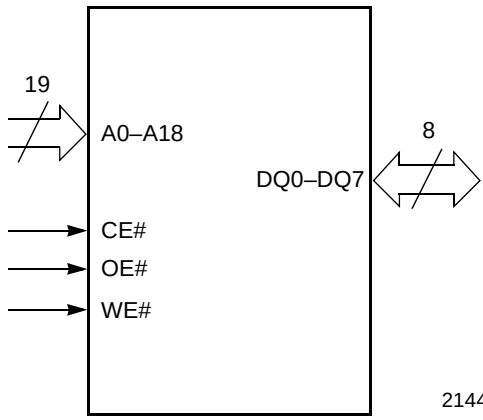
WE# = Write Enable

OE# = Output Enable

V_{SS} = Device Ground

V_{CC} = +5.0 V single power supply (see Product Selector Guide for device speed ratings and voltage supply tolerances)

LOGIC SYMBOL



ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of the following:

Am29F040B -55 E C B

OPTIONAL PROCESSING
Blank = Standard Processing
B = Burn-in
(Contact an AMD representative for more information)

TEMPERATURE RANGE
C = Commercial (0°C to +70°C)
I = Industrial (–40°C to +85°C)
E = Extended (–55°C to +125°C)

PACKAGE TYPE
P = 32-Pin Plastic DIP (PD 032)
J = 32-Pin Rectangular Plastic Leaded Chip Carrier (PL 032)
E = 32-Pin Thin Small Outline Package (TSOP) Standard Pinout (TS 032)
F = 32-Pin Thin Small Outline Package (TSOP) Reverse Pinout (TSR032)

SPEED OPTION
See Product Selector Guide and Valid Combinations

DEVICE NUMBER/DESCRIPTION
Am29F040B
4 Megabit (512 K x 8-Bit) CMOS 5.0 Volt-only Sector Erase Flash Memory
5.0 V Read, Program, and Erase

Valid Combinations	
Am29F040B-55	JC, JI, JE, EC, EI, EE, FC, FI, FE
Am29F040B-70	
Am29F040B-90	PC, PI, PE, JC, JI, JE, EC, EI, EE, FC, FI, FE
Am29F040B-120	
Am29F040B-150	

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations and to check on newly released combinations.

www.agelectronica.com

National Semiconductor

April 1991

MM58274C
Microprocessor Compatible Real Time Clock

General Description

The MM58274C is fabricated using low threshold metal gate CMOS technology and is designed to operate in bus oriented microprocessor systems where a real time clock and calendar function are required. The on-chip 32.768 kHz crystal controlled oscillator will maintain timekeeping down to 2.2V to allow low power standby battery operation. This device is pin compatible with the MM58174A but continues timekeeping up to tens of years. The MM58274C is a direct replacement for the MM58274 offering improved Bus access cycle times.

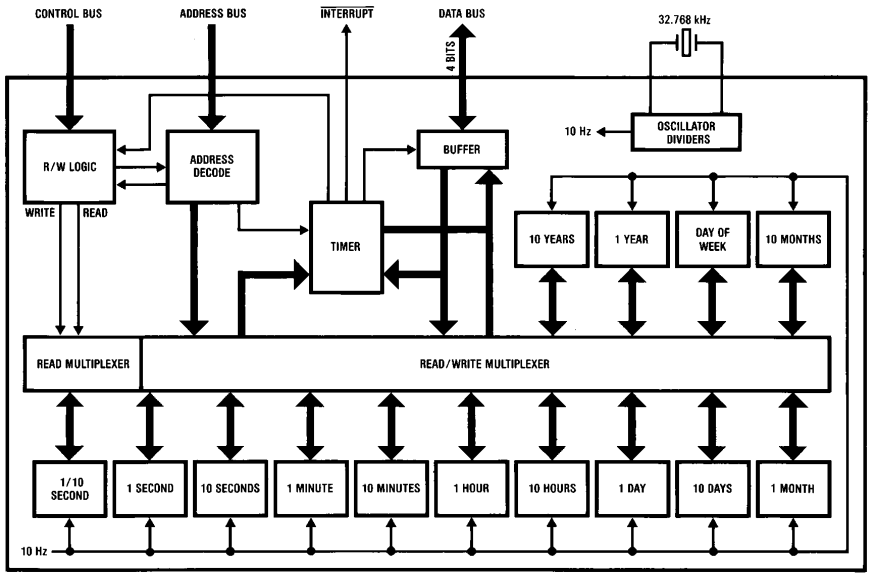
Applications

- Point of sale terminals
- Teller terminals
- Word processors
- Data logging
- Industrial process control

Features

- Same pin-out as MM58174A, MM58274B, and MM58274
- Timekeeping from tenths of seconds to tens of years in independently accessible registers
- Leap year register
- Hours counter programmable for 12 or 24-hour operation
- Buffered crystal frequency output in test mode for easy oscillator setting
- Data-changed flag allows simple testing for time rollover
- Independent interrupting time with open drain output
- Fully TTL compatible
- Low power standby operation (10 μA at 2.2V)
- Low cost 16-pin DIP and 20-pin PCC

Block Diagram



TL/F/11219–1

FIGURE 1

TRI-STATE® is a registered trademark of National Semiconductor Corp.
Microbus™ is a trademark of National Semiconductor Corp.

 TL/F/11219

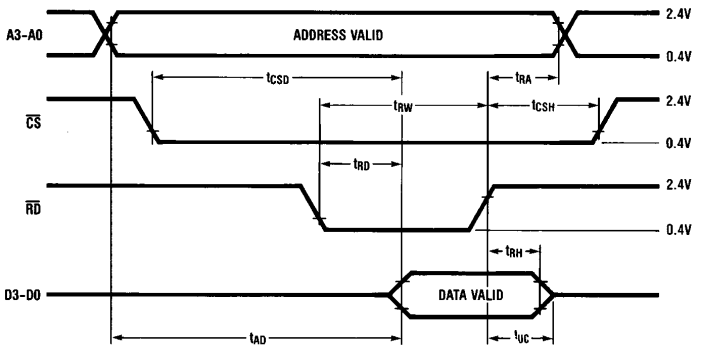
RRD-B30M105/Printed in U. S. A.

www.agelectronica.com

www.agelectronica.com

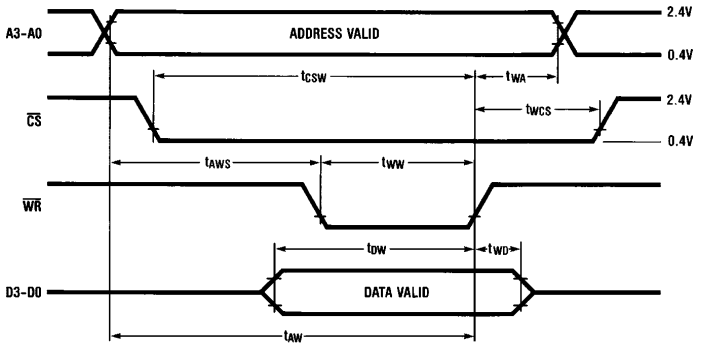
Switching Time Waveforms

Read Cycle Timing (Notes 5 and 7)



TL/F/11219–2

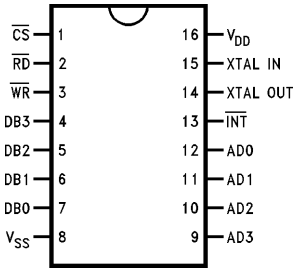
Write Cycle Timing (Notes 5 and 6)



TL/F/11219–3

Connection Diagrams

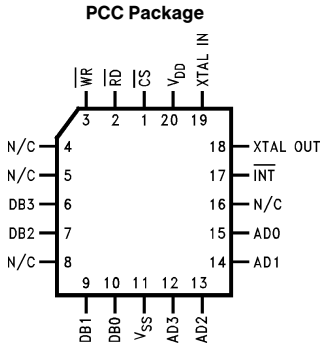
Dual-In-Line Package



Top View

TL/F/11219–4

PCC Package



Top View

TL/F/11219–5

FIGURE 2

Order Number MM58274CJ, MM58274CN or MM58274CV
See NS Package J16A, N16A, or V20A

www.agelectronica.com

19-4323; Rev 15; 1/06

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multidrop RS-232 Networks

AutoShutdown and UCSP are trademarks of Maxim Integrated Products, Inc.

Next-Generation Device Features

- **For Low-Voltage, Integrated ESD Applications MAX3222E/MAX3232E/MAX3237E/MAX3241E/ MAX3246E: +3.0V to +5.5V, Low-Power, Up to 1Mbps, True RS-232 Transceivers Using Four 0.1µF External Capacitors (MAX3246E Available in a UCSP™ Package)**
- **For Low-Cost Applications MAX221E: ±15kV ESD-Protected, +5V, 1µA, Single RS-232 Transceiver with AutoShutdown™**

Ordering Information

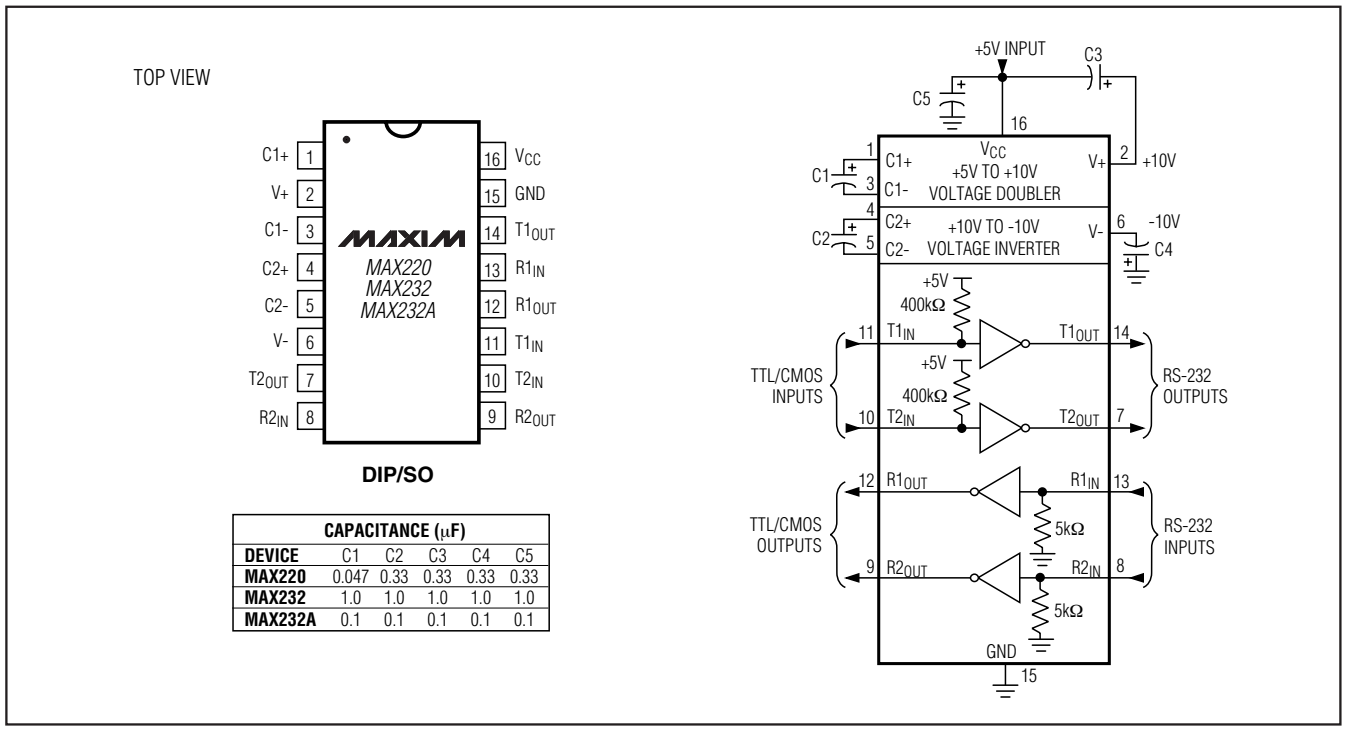
PART	TEMP RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	0.047/0.33	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package



CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

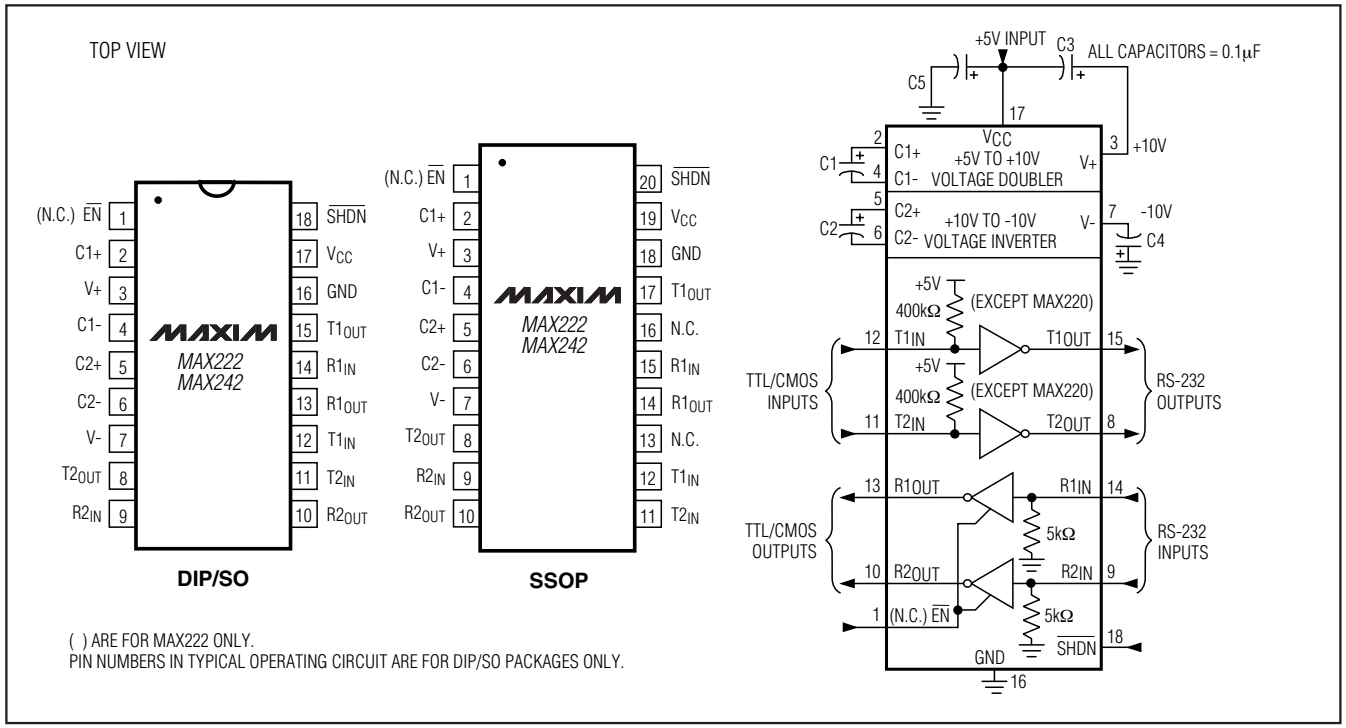


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

National Semiconductor

August 1999

LM117/LM317A/LM317
3-Terminal Adjustable Regulator

General Description

The LM117 series of adjustable 3-terminal positive voltage regulators is capable of supplying in excess of 1.5A over a 1.2V to 37V output range. They are exceptionally easy to use and require only two external resistors to set the output voltage. Further, both line and load regulation are better than standard fixed regulators. Also, the LM117 is packaged in standard transistor packages which are easily mounted and handled.

In addition to higher performance than fixed regulators, the LM117 series offers full overload protection available only in IC's. Included on the chip are current limit, thermal overload protection and safe area protection. All overload protection circuitry remains fully functional even if the adjustment terminal is disconnected.

Normally, no capacitors are needed unless the device is situated more than 6 inches from the input filter capacitors in which case an input bypass is needed. An optional output capacitor can be added to improve transient response. The adjustment terminal can be bypassed to achieve very high ripple rejection ratios which are difficult to achieve with standard 3-terminal regulators.

Besides replacing fixed regulators, the LM117 is useful in a wide variety of other applications. Since the regulator is "floating" and sees only the input-to-output differential voltage, supplies of several hundred volts can be regulated as long as the maximum input to output differential is not exceeded, i.e., avoid short-circuiting the output.

Also, it makes an especially simple adjustable switching regulator, a programmable output regulator, or by connecting a fixed resistor between the adjustment pin and output, the LM117 can be used as a precision current regulator. Supplies with electronic shutdown can be achieved by clamping the adjustment terminal to ground which programs the output to 1.2V where most loads draw little current.

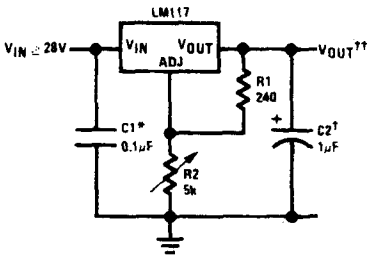
For applications requiring greater output current, see LM150 series (3A) and LM138 series (5A) data sheets. For the negative complement, see LM137 series data sheet.

Features

- Guaranteed 1% output voltage tolerance (LM317A)
- Guaranteed max. 0.01%/V line regulation (LM317A)
- Guaranteed max. 0.3% load regulation (LM117)
- Guaranteed 1.5A output current
- Adjustable output down to 1.2V
- Current limit constant with temperature
- P+ Product Enhancement tested
- 80 dB ripple rejection
- Output is short-circuit protected

Typical Applications

1.2V–25V Adjustable Regulator



DS009063-1

Full output current not available at high input-output voltages

*Needed if device is more than 6 inches from filter capacitors.

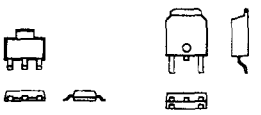
†Optional — improves transient response. Output capacitors in the range of 1 μF to 1000 μF of aluminum or tantalum electrolytic are commonly used to provide improved output impedance and rejection of transients.

$$\dagger\dagger V_{OUT} = 1.25V\left(1 + \frac{R2}{R1}\right) + I_{ADJ}(R_2)$$

LM117 Series Packages

Part Number Suffix	Package	Design Load Current
K	TO-3	1.5A
H	TO-39	0.5A
T	TO-220	1.5A
E	LCC	0.5A
S	TO-263	1.5A
EMP	SOT-223	1A
MDT	TO-252	0.5A

SOT-223 vs D-Pak (TO-252) Packages



DS009063-54

Scale 1:1

SPECIFICATION FOR LCD MODULE

Model No.__ TM162IBCU6 __

Prepared by:	**Date:**
Checked by :	**Date:**
Verified by :	**Date:**
Approved by:	**Date:**

TIANMA MICROELECTRONICS CO., LTD

1. General Specifications:

1.1 Display type: STN

1.2 Display color*:

Display color: Blue-Black

Background: Yellow-Green

1.3 Polarizer mode: Transflective/Positive

1.4 Viewing Angle: 12:00

1.5 Driving Method: 1/16 Duty 1/5 Bias

1.6 Backlight: LED

1.7 Display Fonts: 5 x 7 dots + 5 x 1 Cursor (1 Character)

1.8 Controller: S6A0069X01-C0CX(KS0066UP-00CC)

1.9 Data Transfer: 8 Bit Parallel

1.10 Operating Temperature: 0----+50 ℃

Storage Temperature: -20----+60 ℃

1.11 Outline Dimensions: Refer to outline drawing on next page

1.12 Dot Matrix: 16 Characters X 2 Lines

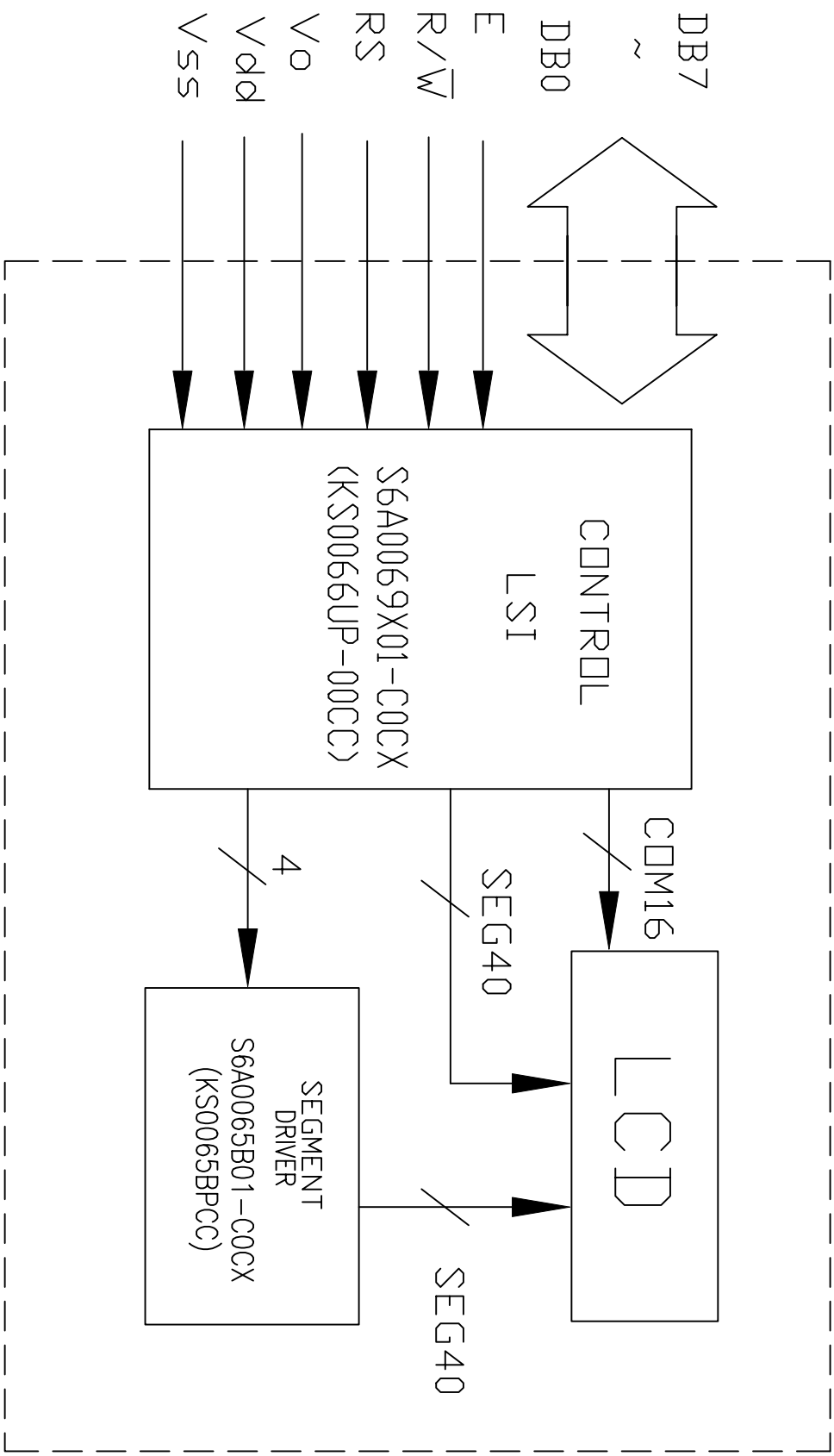
1.13 Dot Size: 0.55X0.55(mm)

1.14 Dot Pitch: 0.60X0.60 (mm)

1.15 Weight: 35g (Approx.)

* Color tone is slightly changed by temperature and driving voltage.

4. Circuit Block Diagram



6.2 Interface Signals

Pin No.	Symbol	Level	Description
1	Vss	0V	Ground
2	Vcc	5.0V	Power supply voltage for logic and LCD(+)
3	Vee	0.3V	Power supply voltage for LCD(-)
4	RS	H/L	Selects registers (H: Data L: Instruction)
5	R/W	H/L	Selects read or write
6	E	H/L	Data read/write enable signal
7	DB0	H/L	Data bit0
8	DB1	H/L	Data bit1
9	DB2	H/L	Data bit2
10	DB3	H/L	Data bit3
11	DB4	H/L	Data bit4
12	DB5	H/L	Data bit5
13	DB6	H/L	Data bit6
14	DB7	H/L	Data bit7
A	LED+	4.2V	Power supply voltage for LCD(+)
K	LED-	0V	Power supply voltage for LCD(-)

6.4 Instruction Code

Instruction Table

Instruction	Instruction Code										Description	Execution time (fosc= 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write “20H” to DDRAM and set DDRAM address to “00H” from AC	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to “00H” from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 μs
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	39 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5×11dots/5×8 dots)	39 μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39 μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 μs

* “-“: don’t care

NOTE: When an MPU program with checking the Busy Flag(DB7) is made, it must be necessary 1/2Fosc is necessary for executing the next instruction by the falling edge of the 'E' signal after the Busy Flag (DB7) goes to “Low”.

Data Sheet

Eco - Line

Low Frequency 32mm Glass Transponder

Specifications:

	RI-TRP-RE2B	*RI-TRP-WE2B*
Functionality	Read Only	Read/Write
Memory (Bits)	64	80*
Memory (Pages)	1	1
Operating Frequency	134.2 kHz	
Modulation	FSK (Frequency Shift Keying) 134.2 kHz / 123.2 kHz	
Transmission Principle	HDX (Half Duplex)	
Power Source	Powered from the reader signal (batteryless)	
Typical Reading Range	≤ 100 cm**	
Typical Programming Range	---	30 % of typical reading range
Typical Read Time	70 ms	
Typical Programming Time	---	309 ms
Typical Programming Cycles	---	10,000
Operating Temperature (Read)	-25 to +70°C	
Operating Temperature (Program)	---	-25 to +70°C
Storage Temperature	-40 to +85°C	
Case Material	Glass	
Protection Class	Hermetically sealed	
EMC	Programmed code is not affected by normal electromagnetic interference or x-rays	
Signal Penetration	Transponder can be read through virtually all non-metallic material	
Mechanical Shock	IEC 68-2-27, Test Ea;	
Vibration	IEC 68-2-6, Test Fc;	
Dimensions	∅ 3.85 mm ± 0.05 mm x 31.2 ± 0.6 mm	
Weight	0.8 g	

* We recommend that you split each 80 bit page into 64 user programmable bits plus a 16 bit wide CRC CCITT Block Check Character as is done by TI-RFID LF readers.

** Depending on RF regulation in country of use, the Reader Antenna configuration used, and the environmental conditions.

For more information, contact the sales office or distributor nearest you. This contact information can be found on our web site at: **http://www.ti-rfid.com**

1.1 General

This document provides information about the S251B Reader. It describes the reader and how to install it.

1.2 System Description

A TIRIS system comprises a reader connected to a control device (usually a host computer) via an RS232, or an RS422/RS485 interface, an antenna and a transponder. It is used for wireless identification of TIRIS LF transponders.

The reader sends a 134.2 kHz power pulse to a transponder, the energy of the generated magnetic field is stored in the capacitor in the transponder and when the power pulse has finished the transponder immediately sends its data back to the reader.

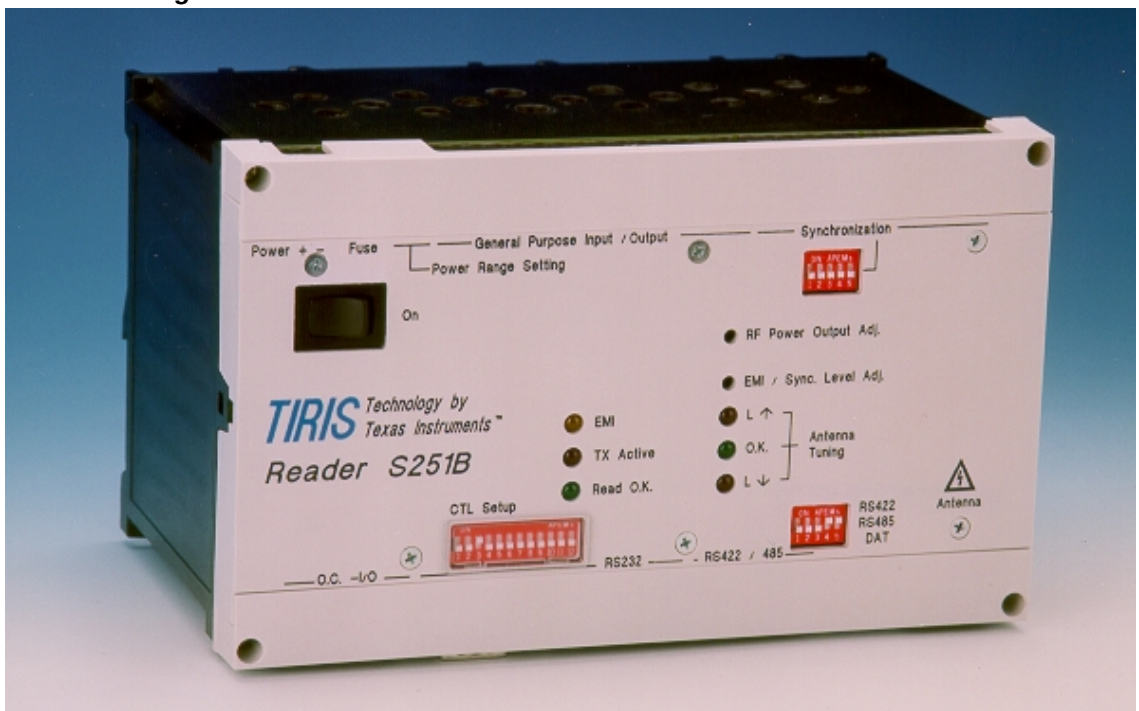
1.3 Product Description

The Reader is an integral part of a TIRIS system, it provides all of the RF and control functions required to communicate with TIRIS LF transponders.

The main task of the Reader is to send a power pulse via the antenna to initialize the transponder, to demodulate the received identification signal and then send the data to a control device. It is also used to send programming data to Read/Write and Multipage transponders.

The Reader is housed in an IP20 Polycarbonate box as shown Figure 1.

Figure 1: S251B Reader

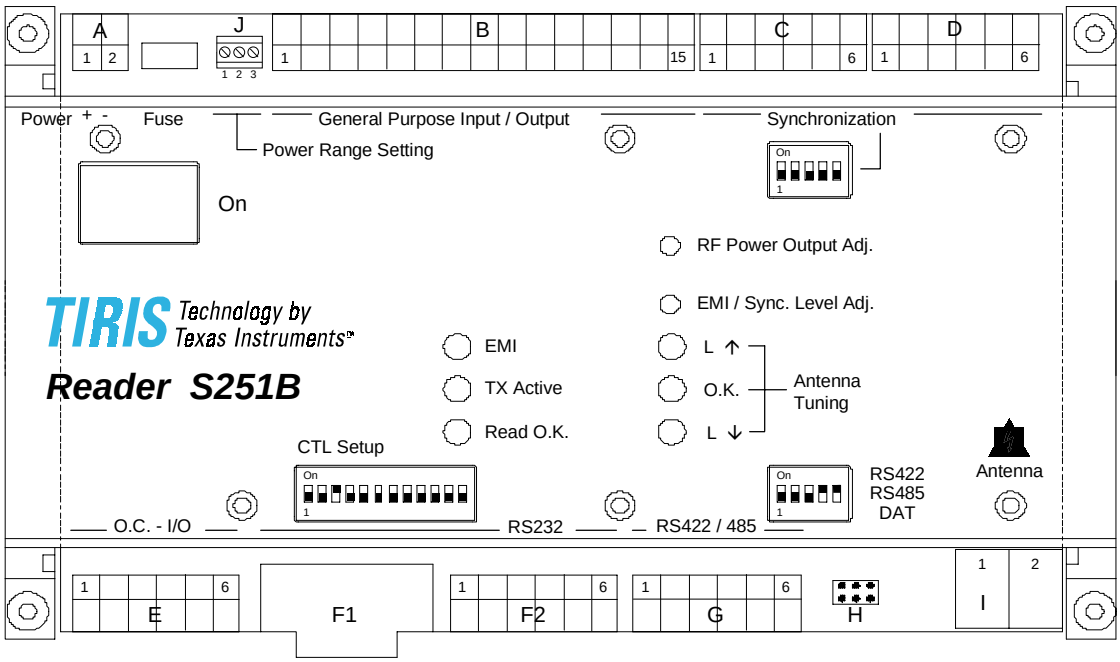


If connected via an RS232 or an RS422/RS485 interface the computer sends commands to the reader using one of the two protocols used by the system (ASCII or TIRIS Bus Protocol), and the reader then communicates via its antenna with any transponders within that antenna's range. The antenna can be mounted up to 5 m (depending on the antenna) away from the reader.

2.2.1 Connectors

There are 10 connectors on the S251B, 7 WECO connectors, the antenna connector, a 9-pin sub-D RS232 connector, a 6-pin connector for the indicator outputs and a 2-pin connector for the antenna. The function of each pin on each connector (except the RS232 sub-D connector) is described in the following paragraphs. Their location is shown in Figure 3.

Figure 3: S251B Connector Locations



In order to gain access to the fuse and connector J you must first remove the upper two screws holding the front panel on, remove the plastic cover strip and then replace the two screws. To gain access to the connectors H and I you must first remove the lower two screws holding the front panel on, remove the plastic cover strip and then replace the two screws.

The pins are not individually numbered on the connectors themselves (just on Figure 3 for your convenience).

The connectors are all marked on Figure 3 with a letter (from A to I) and are listed in Table 1 which also shows the section that describes them

Table 1: List of Connectors

Identifying Letter	Function	Section
A	Supply Connector	2.2.1.1
B	General Purpose Inputs/Outputs	2.2.1.2
C	Synchronization Interface	2.2.1.3
D	Carrier Phase Synchronization Interface	2.2.1.4
E	Open Collector Inputs/Outputs	2.2.1.5
F1	RS232 Connector (9-pin SEB-D)	2.2.1.6

Table 1: List of Connectors

Identifying Letter	Function	Section
F2	RS232 Connector	2.2.1.6
G	RS422/RS485 Connector	2.2.1.7
H	Indicator Outputs	2.2.1.8
I	Antenna Connector	2.2.1.9

2.2.1.1 A - Supply Connector

The Reader requires a single DC supply voltage (10 to 24 V) through a 2-pin connector marked with + for positive and – for negative

The Power Range Setting wired jumpers (marked J in Figure 3) and the actual power supply have a direct consequence on the operating temperature of the reader as shown in Table 2.

Table 2: Power Range Settings

Setting	Input Power	Operating Temperature Range
Pins 1 + 2 connected	10 - 15 V	-20º to +70º C
"	15 - 24 V	-20º to +70º C (max. I_VSP = 0.9 A_{peak} see caution 1).
Pins 2 + 3 connected	18 - 24 V	-20º to +70º C

CAUTION:

1. **In order to operate the reader over the full temperature range with pins 1 + 2 connected (15 to 24 V), the maximum current consumption must not exceed 0.9 A_{peak}. Exceeding this value could result in unreliable functioning of the dynamic auto tuning, or sharp limitation of the transmitter output power because of internal protection. If either of these should occur, switch the device off and allow it time to recover; and then when it is switched on again it will revert to normal operation. Note that if either of these occur it is an indication that the reader is not being operated within its specifications.**
2. **The reader itself generates heat, therefore if it is incorporated into a housing you must ensure (by proper design and/or cooling) that the temperature immediately surrounding the reader does not exceed the operating temperature range.**

1..

Table 3: Supply Connector

Pin	Signal	Description	Direction
1	+	Positive supply	input
2	-	Ground	input

2.2.1.6 F1 & F2 - RS232 Communication Interface

Depending on the DIP-Switch configuration, the Reader will either communicate via the RS232, RS422 or RS485 interface.

There are two interface connectors either of which can be used for an RS232C connection. They are: a standard RS232 Interface 9-pin SUB-D male connector (F1 on Figure 3) and a 6-pin WECO connector (F2 on Figure 3). Both of these connectors allow communication between the reader and a controlling device. The pin assignment for the SUB-D connector is given in Table 13 and the pin assignment for the WECO connector is given in Table 14.

Both, the ASCII and TIRIS Bus protocol can be used with the RS232 interface.

Table 13: RS232 SUB-D Connector

Pin	Signal	Description	Direction
1	-	Not connected	-
2	TxD	Transmit Data	Output
3	RxD	Receive Data	Input
4	DTR	Data Terminal Ready	Input
5	GND	Signal Ground	-
6	DSR	Data Set Ready	Output
7	-	Not connected	-
8	-	Not connected	-
9	-	Not connected	-

Table 14: RS232 WECO Connector

Pin	Signal	Description	Direction
1	RxD	Receive Data	Input
2	DTR	Data Terminal Ready	Input
3	GND	Signal Ground	-
4	TxD	Transmit Data	Output
5	DSR	Data Set Ready	Output
6	GND	Signal Ground	-

All interface parameters are according to the RS232 specification and are not given in detail in this manual. The DTR and DSR lines are currently not used for any purpose.

APÉNDICE B

LISTADOS DE PROGRAMAS DE LOS CONTROLADORES LÓGICOS DEDICADOS Y DE LA UNIDAD CONCENTRADORA

OS.CPP

```
#include "OS.H"
#include "SCI.H"
#include <stdlib.h>
#include "ports.h"
BYTE FcOS1;  /*x0=control de LUZ, x1=control de SONIDO */
/* Checa que display termino de ejecutar una instruccción*/
void Disocup(void)
{
 #ifdef _SIMULADOR_
 unsigned int i=0x3FF;
 BYTE x;
 while (i) {i--; x = ADIN;}
 #else
 while (DISCON & 0X80) {}
 #endif
}

void InitOS(void)
{
 unsigned int  i;
 APORT2 = 0XFF;    /*puerto A definido como bus de datos */
 DPORT1 = 0X00;   /*modo de operación de microcontrolador como función A*/
 DPORT2 = 0X30;   /*bit 4 y 5 de puerto D definidos como read&write y selección de periferico  */
 SCCR0 |= 0X2C;    /* HABILITAR CICLO DE ESPERA PARA PERIFERICOS*/
 SCCR1 &= 0XEF;    /*HABILITACIÓN DEL CICLO DE ESPERA*/
 T1PC1 = 0X01;    /*t1evt (salida prop. general) definido como salida digital TECLADO*/
 T1PC2 = 0X11;    /*t1ic/cr (on buzzer) y t1pwm (on luz)definidos como salida digital */
 ADENA = 0XFF;    /*pto E def como ent dig bits 0,1 Y 7 selec. dir de SPI y del  2-6 p/teclado*/
 INT2  = 0X10;     /* habilita terminales de teclado como salida -columnas TECLADO */
 INT3  = 0X10;
 T2PC1 = 0X01;
 T2PC2 = 0X11;
 T2CMSB = 0XFF;   /*INI CONTADOR PARA TIEMPO DE OFF LUZ*/
 T2CLSB = 0XFF;   /*T=13 mS*/
 T2PRI  = 0X40;   /* DEFINE BAJA PRIORIDAD PARA INT DE TIMER*/
 T2CTL1 = 0X00;   /* DEFINE PARAMETROS DE TIMER*/
 T2CTL2 = 0X00;
 T2CTL3 = 0X00;
 for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar display*/
 INT2  |= 0X08;     /* QUITAR     PRUEBA ********************************** */
 Ntci  = ADIN;      /*lectura de numero de tarjeta*/
 Ntci &= 0x03;
 if (ADIN & 0X80) Ntci |= 0X4;
 if(Ntci == 0x7)
     {
      FcOS1 = erByte(EE_SIST);  /*lee banderas de luz y sonido*/
      DISCON = 0X38 ;  /*config. display para realizar la interfaz ccon el microcontrolador8bits de 8 bits y caracteres de 5X7 y dos lineas*/
                       Disocup();
      DISCON = 0X0D ;   /* enciende display y cursor */
                       Disocup();
      DISCON = 0X06 ;   /* define desplazamiento de cursor a la derecha*/
                       Disocup();
     }
 asm("     EINT");
 IniRTC(0, 0, 0, 1, 1, 4, 0, 1);
}

BYTE INP(void)
{
 BYTE r,c,bx;
 for (r=0; r<5; r++)
 {
            switch (r)
            {
                       case 0 : T2PC2 |= 0x04; break;
                       case 1 : T2PC2 |= 0x40; break;
                       case 2 : T2PC1 |= 0x04; break;
                       case 3 : INT2  |= 0x08; break;
                       case 4 : INT3  |= 0x08; break;
            }

                      bx = ADIN;
```

```
                        switch (r)
                        {
                          case 0 : T2PC2 &= 0xFB; break;
                          case 1 : T2PC2 &= 0xBF; break;
                          case 2 : T2PC1 &= 0xFB; break;
                          case 3 : INT2  &= 0xF7; break;
                          case 4 : INT3  &= 0xF7; break;
                        }

                        if (r == 0) bx &= 0x78;
                        else bx &= 0x7c;

                        if (bx != 0)
                        {
                          bx = bx>>2;
                          for (c=0; c<5; c++)
                          {
                                  if(bx & 0x01) return (r*10+c);
                                  bx= bx>>1;
                          }
                        }
                }
               return 0;
}

BOOL TestShift(void)
{
 BOOL fShift;
 T2PC2 |= 0x04;
 fShift = (ADIN & 0x04);
 T2PC2 &= 0xFB;
 return fShift;
}

BYTE TestKey(void)
{
 unsigned int i;
 BYTE Button;
 BYTE Key;
 Button = INP();
 if (!Button) return 0;

 if (FcOS1&0x2) T1PC2 |= 0X4; /*ON SONIDO*/

 #ifdef _TECBOTONES_
 switch (Button)
 {
                case 10 :   Key = VK_USUARIO;               break;
                case 20 :   Key = VK_ORDEN;                 break;
                case 30 :   Key = VK_RETURN;                break;
                case 12 :   if (TestShift()) Key = VK_F9;
                else Key = VK_F1;                           break;
                case 11 :   if (TestShift()) Key = VK_F10;
                else Key = VK_F2;                           break;
                case 14 :   if (TestShift()) Key = VK_F11;
                else Key = VK_F3;                           break;
                case 13 :   if (TestShift()) Key = VK_F12;
                else Key = VK_F4;                           break;
                case 32 :   if (TestShift()) Key = VK_F13;
                else Key = VK_F5;                           break;
                case 31 :   if (TestShift()) Key = VK_F14;
                else Key = VK_F6;                           break;
                case 34 :   if (TestShift()) Key = VK_F15;
                else Key = VK_F7;                           break;
                case 33 :   if (TestShift()) Key = VK_F16;
                else Key = VK_F8;                           break;
                case 43 :   if (TestShift()) Key = VK_SPACE;
                else Key = VK_0;                            break;
                case 24 :   if (TestShift()) Key = VK_ABC;
                else Key = VK_1;                            break;
                case 44 :   if (TestShift()) Key = VK_DEF;
                else Key = VK_2;                            break;
```

```
            case 04 :   if (TestShift()) Key = VK_GHI;
            else Key = VK_3;                        break;
            case 21 :  if (TestShift()) Key = VK_JKL;
            else Key = VK_4;                        break;
            case 41 :   if (TestShift()) Key = VK_MNO;
            else Key = VK_5;                        break;
            case 01 :   if (TestShift()) Key = VK_PQR;
            else Key = VK_6;                        break;
            case 22 :   if (TestShift()) Key = VK_STU;
            else Key = VK_7;                        break;
            case 42 :   if (TestShift()) Key = VK_VWX;
            else Key = VK_8;                        break;
            case 02 :   if (TestShift()) Key = VK_YZ;
            else Key = VK_9;                        break;
            case 23 :   Key = VK_UP;                break;
            case 03 :   Key = VK_DOWN;              break;
            case 40 :   if (TestShift()) Key = VK_CLALL;
            else Key = VK_CLEAR;                    break;
}
#else
switch (Button)
{
            case 10 :   Key = VK_USUARIO;           break;
            case 20 :   Key = VK_ORDEN;             break;
            case 30 :   Key = VK_RETURN;            break;
            case 11 :   if (TestShift()) Key = VK_F9;
            else Key = VK_F1;                       break;
            case 12 :   if (TestShift()) Key = VK_F10;
            else Key = VK_F2;                       break;
            case 13 :   if (TestShift()) Key = VK_F11;
            else Key = VK_F3;                       break;
            case 14 :   if (TestShift()) Key = VK_F12;
            else Key = VK_F4;                       break;
            case 31 :   if (TestShift()) Key = VK_F13;
            else Key = VK_F5;                       break;
            case 32 :   if (TestShift()) Key = VK_F14;
            else Key = VK_F6;                       break;
            case 33 :   if (TestShift()) Key = VK_F15;
            else Key = VK_F7;                       break;
            case 34 :   if (TestShift()) Key = VK_F16;
            else Key = VK_F8;                       break;
            case 44 :   if (TestShift()) Key = VK_SPACE;
            else Key = VK_0;                        break;
            case 23 :   if (TestShift()) Key = VK_ABC;
            else Key = VK_1;                        break;
            case 43 :   if (TestShift()) Key = VK_DEF;
            else Key = VK_2;                        break;
            case 3  :   if (TestShift()) Key = VK_GHI;
            else Key = VK_3;                        break;
            case 22 :   if (TestShift()) Key = VK_JKL;
            else Key = VK_4;                        break;
            case 42 :   if (TestShift()) Key = VK_MNO;
            else Key = VK_5;                        break;
            case 2  :   if (TestShift()) Key = VK_PQR;
            else Key = VK_6;                        break;
            case 21 :   if (TestShift()) Key = VK_STU;
            else Key = VK_7;                        break;
            case 41 :   if (TestShift()) Key = VK_VWX;
            else Key = VK_8;                        break;
            case 1  :   if (TestShift()) Key = VK_YZ;
            else Key = VK_9;                        break;
            case 40 :   Key = VK_UP;                break;
            case 4  :   Key = VK_DOWN;              break;
            case 24 :   if (TestShift()) Key = VK_CLALL;
            else Key = VK_CLEAR;                    break;
}
#endif

i = TIEMPO_RETARDO_TECLADO;
while (i) {i--; if (FcOS1&0x2) T1PC2 |= 0X4;}
while (Button == INP()) {}
if(FcOS1&0x2) T1PC2 &= 0XFB;   /*OFF SONIDO*/
if(FcOS1&0x1)
```

```
 {
   T1PC2 |=0X40;  /*ENCIENDE ILUMINACION*/
   T2CTL1=0X0;   /*RESETEA TIMER*/
   T2CTL2=0X1;   /*HABILITA INTERRUPCION DE TIMER 2*/
   ContSeg=770;  /*TIMER DE APAGADO DE LUZ T=(13mS)(ContSeg)*/
 }

 return Key;
}

/*  Polea el teclado hasta recibir una tecla, si la tecla es SHIFT se espera */
/*  a recibir otra tecla y la interpreta como shift-tecla. */
BYTE GetKey(void)
{
 BYTE Key = TestKey();
 while (!Key) {Key=TestKey();}
 return Key;
}

/* Bucle de teclas alfanumericas */
void  AlfaShiftLoop(BYTE r, BYTE c, BYTE* Key)
{
 BYTE LastKey=*Key;
 BYTE rep=0;
 BYTE Char=*Key;
 WriteChar(r,c,*Key);
 do
 {
            *Key = TestKey();
            if ((*Key) && ALFA(*Key))
            {
                        if (*Key == LastKey)
                        {
                                    if (*Key != VK_YZ) rep = (++rep % 3);
                                    else rep = (++rep % 2);
                                    if (*Key == VK_SPACE)
        {if (rep==0) Char=*Key; else if (rep==1) Char='.'; else Char='-';}
                                    else Char = *Key + rep;
                        }
                        else  {rep=0; Char=*Key;}

                        WriteChar(r,c,Char);
                        LastKey = *Key;
            }
 }
 while (TestShift());

 *Key = Char;
}
/* Recibe un string del teclado */
BOOL InputString(BYTE r, BYTE c, BYTE maxlen, char* Data,BOOL fAlfa)
{
 BYTE ptr=strlen(Data);
 BYTE Key,test=0;
 maxlen--;
 WriteString(r,c, Data);
 if (ptr) c+=strlen(Data);
 CursorOn();
 do
 {
  Key = GetKey();
  if ((!test)&&(!RETURN(Key))) {while (ptr) {WriteChar(r,c,0x20); c--; CursorPrior(); ptr--;} WriteChar(r,c,0x20); Data[ptr]=0;}
  if (NUMBER(Key)) WriteChar(r,c,Key);
  if ((fAlfa) && (ALFA(Key))) AlfaShiftLoop(r,c,&Key);

            if (((fAlfa && ALFANUM(Key)) || (!fAlfa && NUMBER(Key))) && (ptr<=maxlen))
                        {Data[ptr]=Key; Data[ptr+1]=0; if (ptr<maxlen) {test++; ptr++; c++; CursorNext();}}

            if (CLEAR(Key))
            {
                        if (ptr>0)
                        {
                          if (Key == VK_CLEAR) {if (!Data[ptr]) {ptr--; c--; CursorPrior();} Data[ptr]=0; WriteChar(r,c,0x20);}
```

```
                        if (Key == VK_CLALL) {while (ptr) {WriteChar(r,c,0x20); c--; CursorPrior(); ptr--;} WriteChar(r,c,0x20); Data[ptr]=0;}
                        }
                        else break;
                }
 }
 while (!RETURN(Key));
 CursorOff();
 return (RETURN(Key));
}
/* Escribe en el display y lee del teclado simutaneamente una cadena alfanum*/
BOOL ReadString(BYTE r, BYTE c, char* Text, BYTE Long, char* Data)
{
 ClearRow(r);
 WriteString(r,c,Text);
 return InputString(r,c+strlen(Text),Long,Data,TRUE);
}

/* Escribe en el display y lee del teclado simutaneamente una cadena numerica */
BOOL ReadNumber(BYTE r, BYTE c, char* Text, unsigned int Min, unsigned int Max, unsigned int* val)
{
 char Tmp[LEN_DISPLAY];
 while (1)
 {
  ClearRow(r);
  WriteString(r,c, Text);
  if (*val) ltoa(*val,Tmp);
  else Tmp[0]=0;
                if (!InputString(r,c+strlen(Text),5,Tmp,FALSE)) return FALSE;
                *val = atol(Tmp);
                if ((*val >= Min) && (*val <= Max)) return TRUE;
                WriteError(r,"Fuera de rango");
  *val = 0;
 }
}
/* Escribe una cadena como pregunta. Con shift se intercambian las respuestas */
BOOL Question(BYTE r, BYTE c, char* Text, BOOL def, BOOL del)
{
 BYTE Key;
 if (del)  ClearRow(r);
 WriteString(r,c, Text);
 if (def) WriteString(r,c+strlen(Text),"Si");
 else WriteString(r,c+strlen(Text),"No");
 do
 {
                if (TestShift())
                {
                        def = !def;
                        if (def) WriteString(r,c+strlen(Text),"Si");
                        else WriteString(r,c+strlen(Text),"No");
    while (TestShift()) {};
                }
                Key = TestKey();
 }
 while (!RETURN(Key));
 return def;
}
/* Limpia el display */
void ClearDisplay(void)
{
                DISCON = 0X01 ;
                Disocup();
}
/* Limpia una Linea del Display */
void ClearRow(BYTE r)
{
 BYTE i;
 for (i=1;i<=LEN_DISPLAY;i++) WriteChar(r,i,0x20);      /* 16 por 21 */
}
/* Escribe un caracter */
void WriteChar(BYTE r, BYTE c, BYTE ch)
{
 DISCON = (((r - 1)*64)+(c - 1))+128 ;
 Disocup();
```

```
 DISDATA = ch;
 Disocup();
 DISCON = 0x10;
 Disocup();
}
/* Imprime la cadena str en la columna c renglon r. sin avanzar el cursor */
void WriteString(BYTE r, BYTE c, char* str)
{
 BYTE i = 0;
 while (str[i])
 {
            WriteChar(r,c,str[i++]);
  c++;
 }
 DISCON = (((r - 1)*64)+(c - 1))+128 ;
 Disocup();
}
/* Imrime la cadena en la columna c renglon r. Espea cualquier tecla para continuar */
void WriteError(BYTE r, char* str)
{
 ClearRow(r);
 WriteString(r,1,str);
 GetKey();
}
/* Posiciona el cursor en renglon r columna c */
void CursorPos(BYTE r, BYTE c)
{
 DISCON = (((r - 1)*64)+(r - 1))+128 ;
 Disocup();
}
/* Avanza el cursor una posicion */
void CursorNext(void)
{
 DISCON = 0x14 ;
 Disocup();;
}
/* Retraza el cursor una posicion */
void CursorPrior(void)
{
 DISCON = 0x10 ;
 Disocup();
}
/* Enciende el cursor */
void CursorOn(void)
{
 DISCON = 0x0E ;
 Disocup();
}
/*ENCIENDE*/
void CursorBlink(void)
{
 DISCON = 0x0D ;
 Disocup();
}
/* Apaga el cursor */
void CursorOff(void)
{
 DISCON = 0x0c ;
 Disocup() ;
}

/*****************************************************************************/
/***********              Manejo de memoria EEProm           ************/
/*****************************************************************************/
unsigned int erInt(unsigned int addr)
{
 return eRead(addr)*0x100+eRead(addr+1);
}

BYTE erByte(unsigned int addr)
{
 return eRead(addr);
}
```

```
void erStr(unsigned int addr, char* buffer)
{
 unsigned int  cont=0;
 do
 {
                if ((buffer[cont]=eRead(addr+cont)) == 0xFF) buffer[cont]=0;
                cont++;
 }
 while (buffer[cont-1]);
}

void ewInt(unsigned int addr, unsigned int data)
{
 ewByte(addr,data/0x100);
 ewByte(addr+1,data%0x100);
}

void ewStr(unsigned int addr, char* data)
{
 unsigned int cont=0;
 unsigned int x;
 while (data[cont])
 {
                eWrite(addr+cont,data[cont]);
                cont++;
 }
}

/*rutina de programacion de opciones de encendido de luz y sonido*/

void LuzSonido(void)
{
        BOOL def=TRUE;
        ClearDisplay();
        if (!(FcOS1&0X1)){def=FALSE;}
        if (Question(1,1,"Encender Luz? ",def,TRUE)) {FcOS1 |= 0x1;}
        else {FcOS1 &= 0xFE; T1PC2 &= 0XBF;}
        def=TRUE;
        ClearDisplay();
        WriteString(1,1,"Encender Sonido?");
        if (!(FcOS1&0X2)) {def=FALSE;}
        if (Question(2,1," ",def,TRUE)) {FcOS1 |= 0x2;}
        else {FcOS1 &= 0xFD; T1PC2 &= 0XFB;}
        ewByte(EE_SIST,FcOS1);
}

/*RUTINAS DE MANEJO DE RELOJ DE TIEMPO REAL*/

void IniRTC(BYTE Segundos, BYTE Minutos, BYTE Horas, BYTE Dias, BYTE Mes, BYTE Ano, BYTE Bis, BYTE Formato)
{
 unsigned int i;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar display*/
 CRRTC = 0x0F; /*TEST MODE, CLOCK STOP, INTERRUPT REG, INTERRUPT STOP*/
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar display*/
 CSRTC = 0x0;  /*DESHABILITA INTERRUPCIONES*/
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar display*/
 CRRTC = 0x05; /*NORMAL MODE, CLOCK STOP, CLOCK SETTING REG, INTERRUPT STOP*/
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 TSRTC = Segundos/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 USRTC = Segundos%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 TMRTC = Minutos/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 UMRTC = Minutos%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 THRTC = Horas/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 UHRTC = Horas%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
 TDRTC = Dias/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
```

```
  UDRTC = Dias%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  TMORTC = Mes/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  UMORTC = Mes%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  TYRTC = Ano/10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  UYRTC = Ano%10;
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  CSRTC = ((Bis*4)+Formato);
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
  CRRTC = 0x01; /*NORMAL MODE, CLOCK RUN, CLOCK SETTING REG, INTERRUPT STOP*/
/*  for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para iniciar RTC*/
}

BOOL ReadRTC(char* Rtc)
{
 BYTE x,y,z;
 x=CRRTC;
 x=10;
 while(x)
  {
    y=THSRTC;                  /*DECIMAS DE SEGUNDOS*/
    y&=0xF;                   /*DECIMAS DE SEGUNDOS*/
    Rtc[0]=y;                 /*DECIMAS DE SEGUNDOS*/
    y=TSRTC;                   /*DECENAS DE SEGUNDOS*/
    y&=7;                     /*DECENAS DE SEGUNDOS*/
    y*=10;                    /*DECENAS DE SEGUNDOS*/
    z=USRTC;                    /*UNIDADES DE SEGUNDOS*/
    z&=0xF;                   /*UNIDADES DE SEGUNDOS*/
    y+=z;                     /*SEGUNDOS*/
    Rtc[1]=y;                  /*SEGUNDOS*/
    y=TMRTC;                    /*DECENAS DE MINUTOS*/
    y&=7;                     /*DECENAS DE MINUTOS*/
    y*=10;                    /*DECENAS DE MINUTOS*/
    z=UMRTC;                    /*UNIDADES DE MINUTOS*/
    z&=0xF;                   /*UNIDADES DE MINUTOS*/
    y+=z;                     /*MINUTOS*/
    Rtc[2]=y;                  /*MINUTOS*/
    y=THRTC;                    /*DECENAS DE HORAS*/
    y&=3;                     /*DECENAS DE HORAS*/
    y*=10;                    /*DECENAS DE HORAS*/
    z=UHRTC;                    /*UNIDADES DE HORAS*/
    z&=0xF;                   /*UNIDADES DE HORAS*/
    y+=z;                     /*HORAS*/
    Rtc[3]=y;                  /*HORAS*/
    y=TDRTC;                    /*DECENAS DE DIAS*/
    y&=3;                     /*DECENAS DE DIAS*/
    y*=10;                    /*DECENAS DE DIAS*/
    z=UDRTC;                    /*UNIDADES DE DIAS*/
    z&=0xF;                   /*UNIDADES DE DIAS*/
    y+=z;                     /*DIAS*/
    Rtc[4]=y;                  /*DIAS*/
    y=TMORTC;                    /*DECENAS DE MESES*/
    y&=1;                     /*DECENAS DE MESES*/
    y*=10;                    /*DECENAS DE MESES*/
    z=UMORTC;                    /*UNIDADES DE MESES*/
    z&=0xF;                   /*UNIDADES DE MESES*/
    y+=z;                     /*MESES*/
    Rtc[5]=y;                  /*MESES*/
    y=TYRTC;                    /*DECENAS DE AÑOS*/
    y&=0XF;                   /*DECENAS DE AÑOS*/
    y*=10;                    /*DECENAS DE AÑOS*/
    z=UYRTC;                    /*UNIDADES DE AÑOS*/
    z&=0xF;                   /*UNIDADES DE AÑOS*/
    y+=z;                     /*AÑOS*/
    Rtc[6]=y;                  /*AÑOS*/
    if(!(CRRTC&0x8)) return TRUE;
    x--;
  }
  return FALSE;
}
```

SARTCI.CPP

```
#include "OS.H"
#include "SARTCI.H"
#include "FLASH.H"
#include "SCI.H"
#include <ports.h>
#include <stdlib.h>
BYTE bx;
void InitAntenas(void)
{
 OnAntena(0x1,0);
 OnAntena(0x1,0);
 OnAntena(0x2,1);
 OnAntena(0x2,1);
 OnAntena(0x4,2);
 OnAntena(0x4,2);
 OnAntena(0x8,3);
 OnAntena(0x8,3);
 OnAntena(0x10,4);
 OnAntena(0x10,4);
 OnAntena(0x20,5);
 OnAntena(0x20,5);
}
void MainDisplay(void)
{
  char Tmp1[4];
  char Tmp[8];
  char Buf[17];
  unsigned int val;
  ClearDisplay();
  CursorOff();
  erStr(EE_NOMBRE,Buf);
  WriteString(1,(((16-strlen(Buf))/2)+1),Buf);
  DispRTC(2);
}
void DispRTC(BYTE renglon)
{
  char Tmp[7];
  char Tmp1[4];
  char Buf[17];
  unsigned int val;
  if(ReadRTC(Tmp))
   {
     val=(0+Tmp[3]);       /*HORA*/
     ltoa(val,Tmp1);
     Buf[0]=0x30;
     Buf[1]=Tmp1[0];
     if(strlen(Tmp1)==2) {Buf[0]=Tmp1[0]; Buf[1]=Tmp1[1];}
     val=(0+Tmp[2]);      /*MINUTOS*/
     ltoa(val,Tmp1);
     Buf[2]=':';
     Buf[3]=0x30;
     Buf[4]=Tmp1[0];
     if(strlen(Tmp1)==2) {Buf[3]=Tmp1[0]; Buf[4]=Tmp1[1];}
     val=(0+Tmp[1]);      /*SEGUNDOS*/
     ltoa(val,Tmp1);
     Buf[5]=':';
     Buf[6]=0x30;
     Buf[7]=Tmp1[0];
     if(strlen(Tmp1)==2) {Buf[6]=Tmp1[0]; Buf[7]=Tmp1[1];}
     val=(0+Tmp[4]);      /*DIA*/
     ltoa(val,Tmp1);
     Buf[8]=' ';
     Buf[9]=' ';
     Buf[10]=0x30;
     Buf[11]=Tmp1[0];
     if(strlen(Tmp1)==2) {Buf[10]=Tmp1[0]; Buf[11]=Tmp1[1];}
     val=(0+Tmp[5]);      /*MES*/
     ltoa(val,Tmp1);
     Buf[12]='/';
     Buf[13]=0x30;
     Buf[14]=Tmp1[0];
     if(strlen(Tmp1)==2) {Buf[13]=Tmp1[0]; Buf[14]=Tmp1[1];}
```

```
    Buf[15]=' ';
    Buf[16]=0;
    WriteString(renglon,1,Buf);
   }
}
#ifdef _APLICACION_
#endif
void ProcessNombre(void)
{
 char tmp[15];
 BYTE cont=0;
 tmp[0]=0;
 ClearDisplay();
 WriteString(1,1,"Nombre");
 ReadString(2,1,">",14,tmp);
 if (tmp[0]!=0)
  {
   cont=0;
   while (cont<14){eWrite((EE_NOMBRE+cont),0x0); cont++;}
   ewStr(EE_NOMBRE,tmp);
  }
}
void MainKeyProcess(key)
{
 if (key == VK_F1)      OnAntena(0x1,0);
 if (key == VK_F2)      OnAntena(0x2,1);
 if (key == VK_F3)      OnAntena(0x4,2);
 if (key == VK_F4)      OnAntena(0x8,3);
 if (key == VK_F5)      OnAntena(0x10,4);
 if (key == VK_F6)      OnAntena(0x20,5);
 if (key == VK_F8)      Sensor();
 if (key == VK_F9)      ConfigRTC();
 if (key == VK_F10)     LuzSonido();
 if (key == VK_F11)     ProcessNombre();
 if (key == VK_F12)     TechSup();
 MainDisplay();
}
void main(void)
{
 BYTE key;
 char tmp [5];
 InitFlash();
 InitOS();
 IniSCI();
 IniSPI();
 if (Ntci!=7)
  {
     FF_STAR=0;
     FF_ENV=0;
     while (frByte(FF_STAR,0)!=0xFF)
     {
      FF_STAR+=1;
     }
     FF_ENV=FF_STAR;
    while(1)
     {
      if (Fcps2 & 0x1)  Slave(); /* fBPS2=on? llego inf de uC interfaz?*/
     }
  }
 InitAntenas();
 CursorOff();
 MainDisplay();
 while (1)
 {
   DispRTC(2);
   key=TestKey();
              if (key) MainKeyProcess(key);
   if(FcSCI&0x1)
     {
      RS232();
      MainDisplay();
     }
 }
```

```
}
SCI.CPP
#include "SCI.H"
#include "OS.H"
#include "FLASH.H"
#include "SARTCI.H"
#include "ports.H"
/*RUTINA DE INICIALIZACION DE MODULO SCI*/
void IniSCI(void)
{
  long cont;
            T1PC1 = 0X01;    /*config pin T1EVT (pin 44) como salida para pulso de torniquete(slave) alarma(master)*/
  T1CTL1 = 0X04;   /*configura timer 1 de ON TORNIQUETE/ALARMA*/
  T1CTL2 = 0X0;    /*reseta contador de timer 1*/
  T1CTL3 = 0X0;    /*deshabilita interrupcion de timer 1*/
  T1CMSB = 0X061;  /*carga contador para tiempo de 20 milisegundos*/
  T1CLSB = 0X0A7;
  SCICTL = 0X0;    /*realiza un switch reset a SCI*/
  SCICCR = 0X17;   /*config SCI con 1 bit de parada, 8 datos, IDLE mode, asinc, S/PARIDAD*/
            BAUDMSB = HIBAUD; /*config vel de sci a 9600 baudios*/
  BAUDLSB = LOBAUD;
            SCIPC1 = 0X00;    /*define SCICLK (pin 28) como entrada digital*/
  SCIPC2 = 0X22;    /*define funcion de pines rx y tx en sci*/
  SCIPRI = 0X0;     /*define interrupciones de sci con alta prioridad*/
            TXCTL = 0X0;      /*deshabilita interrupción de Tx*/
  if (!(TBufTx=eRead(EE_TBufTx))) TBufTx=1;   /*factor x 20 mS para activar torniquete*/
  Antenas=0;      /*ini estatus de antenas*/
  FcSCI     = 0X14; /* x0=fBufRx=off, x1=fBufTx=off, x2=fPermisoTx=on, x3=ferrorRx=off,
               x4=flectact=OFF, x5=LIBRE, x6=LIBRE, x7=LIBRE*/
  SCICTL = 0X33;   /*habilita tx y rx de sci, sleep y reloj*/
  AdRx=0;
  RXCTL = 0X1;     /*habilita interrupción de recepción*/
}
void IniSPI(void)
{
  SPICCR = 0X0E7; /*configura 8 datos, edo. inactivo alto y vel 156 kbauds*/
  SPIPC1 = 0X02;  /*habilita la función de spiclk como reloj del spi*/
  SPIPC2 = 0X022; /*habilita el pin spisimo y spisomi*/
  SPIPRI = 0X0;   /*habilita función de prioridad de interrupcion*/
  SPICTL = 0X01;    /*habilita interrupción*/
  INT1 = 0X0;       /*ini pin int 1 empleado para sensar arranque de maraton*/
  Fcps2 = 0x010;   /*x0=fBufSPI1(edo. buf. 0-vacio 1-lleno), x1=fBufSPI2(edo. buf. 0-vacio 1-lleno),
              x2=error spibuf (0=dato ok 1=dato sobreescrito) x3=contestar a uC int
              x4=master/slave(0=master 1=slave) x5=RX NOBUF EN MASTER(0=NO RX 1=RX)
              x6=RX ERRORBUF EN MASTER(0=NO RX 1=RX) x7=TXBYTEMASTER (1=TERMINO DE TX BYTE)
              */
  Fcps21 = 0x00;   /*x0=fESPINFMASTER(0-NO ESP 1-ESPERA INF), x1=LIBRE,
              x2=LIBRE x3=LIBRE
              x4=LIBRE x5=LIBRE
              x6=LIBRE x7=LIBRE
              */
  if(Ntci == 0x7)
     {
      SPICTL |= 0X06;    /*habilita talk bit y master mode*/
      Fcps2  &= 0XEF;    /*bit master/slave=master*/
     }
  SPICCR &= 0X07F;   /*sw reset*/
  BufSPI1[21]=C_FMSG;
  BufSPI2[41]=C_FMSG;
}
/*****************************************************************************/
/*rutina de transmision comunicacion uC Lector-Lector-uC Interfaz*/
/*****************************************************************************/
void Slave(void)
{
 unsigned char x;
 x=BufSPI1[2];
   switch(x)
   {
    case 'C': case 'B': case 'J': case 'X': case 'F':
         {
          BYTE cont,seg1,seg2;
          unsigned char r;
```

```
        Fcps2  &= 0XFA;             /*fBufSPI1=vacio, error spibuf=off*/
        FcSCI  &= 0XF6;             /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;                   /*ini apunt de rx de lector tiris*/
        TXBUF=x;                    /*envia comando a lector de tiris*/
        seg2=USRTC;                 /*lectura de unidades de segndos de lector tiris*/
        seg2&=0xF;
        seg1=(seg2+3);              /*fija ciclo de 2 seg de espera de resp de lector tirs*/
        seg1&=0xF;
        if(seg1>9) seg1=2;
        while((!(FcSCI&0x1))&&(seg1!=seg2)){seg2=USRTC; seg2&=0xF;}   /*espera respuesta de lector de tiris*/
        cont=0;
        BufSPI2[cont++]=C_IMSG;                         /*prepara buffer de resp p/uC int*/
        BufSPI2[cont++]=Ntci;
        if ((FcSCI&0X8)||(seg1==seg2))
           {
            BufSPI2[cont++]=C_ELECT;
           }
        else
           {
             seg1=0;
             while(BufRx[seg1]!=CR)
              {
                BufSPI2[cont++]=BufRx[seg1];
                seg1++;
              }
           }
        BufSPI2[cont++]=C_FMSG;
        ABufSPI2=0;                 /*INI APUNTADOR*/
        Fcps2  |= 0X2;              /*fBufSPI2=lleno*/
        break;
       }
  case C_RSTFLASH:
       {
        unsigned char r;
        BufSPI2[0]=C_IMSG;
        BufSPI2[1]=Ntci;
        BufSPI2[2]=x;
        Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
        x=frByte(0,0);     /*LEE PRIMER BYTE DE FLASH*/
        if(x==0XFF) fwByte(0,0,0XAA);
        ChipErase();         /*RUTINA DE BORRADO DE MEMORIA*/
        FF_STAR=0; /*INI APUNTADOR DE COLA*/
        FF_ENV=0;  /*INI APUNTADOR DE BYTES ENVIADOS*/
        x=frByte(0,0);     /*LEE PRIMER BYTE DE FLASH*/
        if(x!=0XFF)
          {
            ChipErase();
            x=frByte(0,0);     /*LEE PRIMER BYTE DE FLASH*/
            if(x!=0XFF) {BufSPI2[2]=C_ELECT;}
          }
        BufSPI2[3]=C_FMSG;
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        break;
       }
  case C_DRTC:
       {
         IniRTC(BufSPI1[8], BufSPI1[7], BufSPI1[6], BufSPI1[3], BufSPI1[4], BufSPI1[5], (BufSPI1[5]%4), 1);
         Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
         break;
       }
  case 'H': case 'Y': case 'Z': case 'P':
       {
        BYTE y, z, cont,seg1,seg2;
        unsigned char r;
        FcSCI  &= 0XF6;             /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;                    /*ini apunt de rx de lector tiris*/
        TXBUF=x;                  /*TRANSMITE COMANDO A LECTOR TIRIS*/
        seg2=USRTC;                /*LECTURA DE SEGUNDO DE RTC*/
        seg2&=0xF;
        seg1=(seg2+4);           /*FIJA EN 2 SEGUNDOS EL TIEMPO DE ESPERA*/
        seg1&=0xF;
        if(seg1>9) seg1=3;
```

```
        while((AdRx!=1)&&(seg1!=seg2)) {seg2=USRTC; seg2&=0xF;} /*ESPERA RESPUESTA DE LECTOR*/
        r=BufRx[0];                          /*LECTURA DE COMANDO ENVIADO POR LECTOR*/
        cont=0;
        BufSPI2[cont++]=C_IMSG;
        BufSPI2[cont++]=Ntci;
        if( (r!=x) || (seg1==seg2) )
          {
          BufSPI2[cont++]=C_ELECT;
          Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
          goto SALPROG;
          }
        y=3;
        AdTx=0;
        z=1;
        while(z)
          {
          BufTx[AdTx]=BufSPI1[y];
          AdTx+=1;
          y+=1;
          z=(252-BufSPI1[y]);
          }
        Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
        if(x=='Y') BufTx[1]=C_FMSG;
        if((x=='H')||(x=='Z')) BufTx[2]=C_FMSG;
        if(x=='P') BufTx[16]=C_FMSG;
        AdTx=1;
        FcSCI |= 0x2;      /*habilita bufer de tx*/
        TXBUF= BufTx[0];   /*TRANSMITE EL PRIMER DATO*/
        TXCTL |= 0x01;     /*HABILITA INT DE TX*/
        FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;
        seg2=USRTC;
        seg2&=0xF;
        seg1=(seg2+4);
        seg1&=0xF;
        if(seg1>9) seg1=3;
        while((!(FcSCI&0x1))&&(seg1!=seg2)){seg2=USRTC; seg2&=0xF;}
        if ((FcSCI&0X8)||(seg1==seg2)) BufSPI2[cont++]=C_ELECT;
        else
          {
            BufSPI2[cont++]=x;
            if((x=='H')||(x=='P')) BufSPI2[cont++]=BufRx[0];
            if(x=='H') BufSPI2[cont++]=BufRx[1];
          }
        SALPROG:
        BufSPI2[cont++]=C_FMSG;
        FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;
        Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        break;
       }

case C_EINF:
    {
        BYTE y, z, cont,seg1,seg2;
        unsigned char r;
        unsigned int val,aux;
        BYTE cont0, cont1, cont2, segundero, memseg, contseg;
        Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
        BufSPI2[0]=C_IMSG;
        BufSPI2[1]=Ntci;
        BufSPI2[2]=C_CLT;
        val=abs(FF_STAR);
        BufSPI2[3]=(val/256);
        BufSPI2[4]=(val%256);
        BufSPI2[5]=C_FMSG;
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        if(!val){break;}
        aux=0;
        while(val)
```

```
          {
           while(Fcps2 & 0X2) {};
           cont=0;
           BufSPI2[cont++]=C_IMSG;
           BufSPI2[cont++]=Ntci;
           BufSPI2[cont++]=C_LECT;
           frPage(aux, 36, BufSPI2+3);
           BufSPI2[39]=C_FMSG;
           ABufSPI2=0;       /*INI APUNTADOR*/
           Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
           val--;
           aux++;
          }
        {break;}
      }
case 'L': case ESC:
      {
        BYTE y, z, cont,seg1,seg2, F_ENV=0;
        unsigned char r;
        char dat;
        unsigned int val,aux, POINTDAT, CONTDATTX=0, FF_STARX=0;
        char Tmp[10];
        char Tmp1[5];
        BYTE cont0, cont1, cont2, segundero, memseg, contseg;
        y=BufSPI1[3] ;  /*MODO  1=ALM+RTC+DES 2=ALM+RTC+DES+RST 3=ALM+RTC+LINE 4=ALM+RTC+LINE+RST*/
        if(y==3||y==4)  F_ENV=1;   /*MODO 3 O 4 ACTIVA TX EN LINEA*/
        cont0=x;
        Fcps2  &= 0XFA;       /*fBufSPI1=vacio, error spibuf=off*/
        FcSCI  &= 0XF6;       /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;
        if (x==ESC) x='E';
        TXBUF=cont0;             /*TRANSMITE COMANDO A LECTOR*/
        seg2=USRTC;
        seg2&=0xF;
        seg1=(seg2+3);
        seg1&=0xF;
        if(seg1>9) seg1=2;
        while( (!(FcSCI&0x1))&&(seg1!=seg2) ){seg2=USRTC; seg2&=0xF;}
        r=BufRx[0];
        cont=0;
        BufSPI2[cont++]=C_IMSG;
        BufSPI2[cont++]=Ntci;
        if ((r!=x)||(seg1==seg2))
          {
           BufSPI2[cont++]=C_ELECT;
           goto SALIRM;
          }
        BufSPI2[cont++]=x;
        BufSPI2[cont++]=y;
        BufSPI2[cont++]=C_FMSG;
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;
        while (1)
         {
          if(F_ENV)
           {
            if(y==1||y==2)
              {
                  if(!(Fcps2&0X2))
                   {
                     cont1=0;
                     BufSPI2[cont1++]=C_IMSG;
                     BufSPI2[cont1++]=Ntci;
                     BufSPI2[cont1++]=C_LECT;
                     frPage(POINTDAT, 36, BufSPI2+3);
                     BufSPI2[39]=C_FMSG;
                     POINTDAT++;
                     ABufSPI2=0;       /*INI APUNTADOR*/
                     Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
                     if(POINTDAT==FF_STARX) F_ENV=0;
                   }
```

```
   }
  if(y==3||y==4)
    {
     if(FF_STAR!=FF_ENV)
       {
        if(!(Fcps2&0X2))
         {
           cont1=0;
           BufSPI2[cont1++]=C_IMSG;
           BufSPI2[cont1++]=Ntci;
           BufSPI2[cont1++]=C_LECT;
           frPage(FF_ENV, 36, BufSPI2+3);
           BufSPI2[39]=C_FMSG;
           ABufSPI2=0;       /*INI APUNTADOR*/
           Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
           FF_ENV++;
         }
       }
    }
  }
 if(Fcps2&0x1)
  {
   Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
   dat=254-BufSPI1[2];
   if(BufSPI1[2]==0x58)
     {
       FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
       AdRx=0;
       TXBUF='X';         /*TRANSMITE COMANDO PARA APAGAR ANTENAS*/
       seg2=USRTC;
       seg2&=0xF;
       seg1=(seg2+3);
       seg1&=0xF;
       if(seg1>9) seg1=2;
       while((!(FcSCI&0x1))&&(seg1!=seg2)){seg2=USRTC; seg2&=0xF;}
       cont=0;
       BufSPI2[cont++]=C_IMSG;
       BufSPI2[cont++]=Ntci;
       dat=(0x58-BufRx[0]);
       if ((dat)||(seg1==seg2))
         {
           BufSPI2[cont++]=C_ELECT;
           goto SALIRM;
         }
       BufSPI2[cont++]='X';
       goto SALIRM;
     }
   if(dat==6)                  /*checa si llego comando de config de rtc*/
     {
      IniRTC(BufSPI1[8], BufSPI1[7], BufSPI1[6], BufSPI1[3], BufSPI1[4], BufSPI1[5], (BufSPI1[5]%4), 1);
     }
   if(dat==1)                  /*checa si llego comando de solicitud de inf*/
     {
       BufSPI2[0]=C_IMSG;
       BufSPI2[1]=Ntci;
       BufSPI2[2]=C_CLT;
       FF_STARX=FF_STAR;
       POINTDAT=FF_ENV;
       CONTDATTX=abs(FF_STARX-POINTDAT);
       BufSPI2[3]=(CONTDATTX/256);
       BufSPI2[4]=(CONTDATTX%256);
       BufSPI2[5]=C_FMSG;
       ABufSPI2=0;       /*INI APUNTADOR*/
       Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
       if(CONTDATTX){F_ENV=1;}
     }
    if(dat==10)                  /*checa si llego comando RX CORRECTA*/
     {
       BufSPI2[0]=C_IMSG;
       BufSPI2[1]=Ntci;
       BufSPI2[2]=C_RSTFLASH;
       BufSPI2[3]=C_FMSG;
       ABufSPI2=0;       /*INI APUNTADOR*/
```

```
      Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
      FF_ENV=FF_STARX;
     }
  if(dat==23)                  /*checa si llego comando de ACTIVACION DE LECT*/
     {
      FcSCI&=0xEF;       /*factlect=on*/
     }
 }
if(FcSCI&0x1)    /*llego inf de lector*/
 {
  if (!(FcSCI&0x018))    /*hubo error en la lectura, lectura activa*/
   {
    cont0=0;
    if(x=='L') cont0+=1;
    if((BufRx[cont0]!='I')&&(BufRx[cont0]!='C')&&(BufRx[cont0]!=CR))
     {
      if(ReadRTC(Tmp))
       {
        if(y==2||y==4)  TXBUF='C';   /*MODO 2 O 4 TRANSMITE COMANDO CLEAR BUFF*/
        cont1=cont0+23;
        val=(0+Tmp[4]);       /*día*/
        ltoa(val,Tmp1);
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        val=(0+Tmp[5]);      /*mes*/
        ltoa(val,Tmp1);
        cont1+=1;
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        val=(0+Tmp[6]);      /*año*/
        ltoa(val,Tmp1);
        cont1+=1;
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        val=(0+Tmp[3]);      /*horas*/
        ltoa(val,Tmp1);
        cont1+=1;
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        val=(0+Tmp[2]);      /*minutos*/
        ltoa(val,Tmp1);
        cont1+=1;
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        val=(0+Tmp[1]);      /*segundos*/
        ltoa(val,Tmp1);
        cont1+=1;
        BufRx[cont1]=0x30;
        cont1+=1;
        BufRx[cont1]=Tmp1[0];
        if(strlen(Tmp1)==2) {cont1-=1; BufRx[cont1]=Tmp1[0]; cont1+=1; BufRx[cont1]=Tmp1[1];}
        cont1+=1;           /*decimas*/
        BufRx[cont1]=(0x30+Tmp[0]);
        aux=abs(FF_STAR);
        if(aux<8190)    /*checa si no llego a limite de memoria*/
          {
           fwPage(aux, (BufRx+cont0), 36);
           aux+=1;
           FF_STAR=abs(aux); /*INC APUNTADOR DE COLA*/
          }
       }
     }
   }
```

```
            FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
            AdRx=0;
           }
         }
        SALIRM:
        BufSPI2[cont++]=C_FMSG;
        FcSCI  &= 0XF6;    /*fBufSCIRX=vacio, error SCIBUF=off*/
        AdRx=0;
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        FcSCI  |= 0X10;    /* flectact=off */
        break;
       }
    default:
       {
        unsigned char cont=0;
        BufSPI2[cont++]=C_IMSG;
        BufSPI2[cont++]=Ntci;
        BufSPI2[cont++]=C_ELECT;
        BufSPI2[cont++]=C_FMSG;
        Fcps2  &= 0XFA;    /*fBufSPI1=vacio, error spibuf=off*/
        ABufSPI2=0;       /*INI APUNTADOR*/
        Fcps2  |= 0X2;    /*fBufSPI2=lleno*/
        break;
       }
    {break;}
   }
}
/*****************************************************************************/
/*RUTINA DE ATENCION AL PUERTO RS232 DE LA PC (MASTER)*/
/*****************************************************************************/
/*RUTINA DE ESPERA DE CONFIRMACION DE MICROCONTROLADORES LECTORES*/
unsigned int Esp_uclect(void)
{
   BYTE seg1, seg2, cont1, cont2, val, dat, esp, cont=20, NuC=0;
   unsigned int i, aux, numlect=0;
   char Tmp[7];
   while(cont)
    {
      for (i = 0; i < 0xff5f; ++ i) ; /*espera 4.8 S*/
      cont-=1;
    }
   cont=6;
   ClearRow(2);
   while(cont)
     {
       GOESP:
       BufSPI2[2]=9;          /*ini comando de respuesta*/
       Fcps21=0;             /*ini comando d espera de frame*/
       BufSPI1[0]=NuC;         /*caracter para solicitar conformacion*/
       BufSPI1[1]=C_BUFOK;
       BufSPI1[2]=C_FMSG;
       SPIDAT=C_IMSG;
       Fcps2  |= 0X1;          /*fBufSPI1=lleno*/
       ABufSPI1=0;             /*INI APUNTADOR*/
       while(Fcps2 & 0X1) {};
       val=250;
       while(val) {val--;}
       WriteString(1,1,"         ");
       WriteString(1,1,"Com uC:");
       ltoa(NuC,Tmp);
       WriteString(1,8,Tmp);
       SPIDAT=C_DUMMY;         /* TRANSMITE DATO A SLAVE PARA RX CARACTER */
       Fcps2 &= 0x7F;          /* fTXDUMMY=OFF*/
       while(!( Fcps2 & 0x80) ){};
       val=10;
       while(val) {val--;}
       if (Fcps21&0X1)
          {
           seg2=USRTC;
           esp=1;
           seg2&=0xF;
           seg1=(seg2+3);
```

```
    seg1&=0xF;
    if(seg1>9) seg1=2;
    while( esp && (seg1!=seg2) )  {seg2=USRTC; seg2&=0xF; esp=(Fcps21&0x1);}
    dat=(255-BufSPI2[2]);
    if (dat==4)        /* cacarter igual a C_CLT=251 */
     {
       aux=(Convertir(BufSPI2[3])*256);
       aux+=Convertir(BufSPI2[4]);
       numlect+=aux;
       WriteString(1,10,"L:");
       ltoa(numlect,Tmp);
                                    WriteString(1,12,Tmp);
       ltoa(aux,Tmp);
       WriteString(2,((NuC*4)+1),Tmp);
       Fcps2  &= 0X9D;    /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
     }
    if (dat==5)   /* cacarter igual a C_LECT=250 */
     {
       BYTE segundero, memseg, contseg;
       FcSCI &= 0x9F;  /*FRXOK=FRXRPT=OFF*/
       BufTx[0]=C_LECT;
       BufTx[1]=BufSPI2[1];
       BufTx[1]+=0X30;     /*NUMERO DE LECTOR DEL QUE PROVIENE EL MENSAJE*/
       cont1=2;
       cont2=3;
       dat=1;
       while (dat)
         {
          BufTx[cont1]=BufSPI2[cont2];
          dat=(252-BufTx[cont1]);   /*checa comando de fin de mensaje*/
          cont1+=1;
          cont2+=1;
         }
       BufSPI2[26]=0;   /*limpia caract para leer buffer*/
       WriteString(2,1,BufSPI2+10);
       AdTx=0;
       FcSCI |= 0x2;  /*fBufTx=On*/
       TXBUF=C_IMSG;
       TXCTL = 0X1;      /*habilita interrupción de Tx*/
       segundero=5;
       memseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
       contseg=memseg;
       while(FcSCI&0x2)
         {
          if (memseg!=contseg) {memseg=(((TSRTC&0xF)*10)+(USRTC&0xF)); segundero--;}
          contseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
          if (!segundero) {break;}
         }
       segundero=7;
       while(segundero==7)
         {
          if(FcSCI&0x20) segundero=3;
          if(FcSCI&0x40) segundero=6;
         }
       if(FcSCI&0x20)
         {
          BufSPI1[0]=NuC;      /*caracter para solicitar conformacion*/
          BufSPI1[1]=C_RXOK;
          Fcps21&=0xFE;
          BufSPI1[2]=C_FMSG;
          SPIDAT=C_IMSG;
          Fcps2  |= 0X1;     /*fBufSPI1=lleno*/
          ABufSPI1=0;        /*INI APUNTADOR*/
          while(Fcps2 & 0X1) {};
          for (i = 0; i < 0x0a28; ++ i) ; /*espera 9.3 mS PARA QUE SLAVE CHEQUE SI TIENE INF A TX */
         }
       Fcps2  &= 0X9D;    /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
       goto GOESP;
     }
   }
Fcps2  &= 0X9D; /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
BufSPI1[0]=NuC;      /*caracter para solicitar conformacion*/
BufSPI1[1]=C_RXOK;
```

```
        Fcps21&=0xFE;
        BufSPI1[2]=C_FMSG;
        SPIDAT=C_IMSG;
        Fcps2  |= 0X1;     /*fBufSPI1=lleno*/
        ABufSPI1=0;       /*INI APUNTADOR*/
        while(Fcps2 & 0X1) {};
        val=200;       /*TIEMPO PARA QUE SLAVE CHEQUE SI TIENE INF A TX*/
        while(val) {val--;}
        NuC+=1;
        cont--;
      }
    return (abs(numlect));
}
/*RUTINA DE CONFIGURACION DE RELOJ DE TIEMPO REAL*/
void Config_RTC(void)
{
    char Tmp[5];
    BYTE DIA, MES, ANO, HORAS, MINUTOS, SEGUNDOS;
    unsigned int val;
    Tmp[0]=BufRx[2];
    Tmp[1]=BufRx[3];
    Tmp[2]=0;
                val = atol(Tmp);
    DIA=val%256;              /* DIA */
    Tmp[0]=BufRx[4];
    Tmp[1]=BufRx[5];
    Tmp[2]=0;
                val = atol(Tmp);
    MES=val%256;              /* MES */
    Tmp[0]=BufRx[6];
    Tmp[1]=BufRx[7];
    Tmp[2]=0;
                ANO = atol(Tmp);          /* AÑO */
    Tmp[0]=BufRx[8];
    Tmp[1]=BufRx[9];
    Tmp[2]=0;
                val = atol(Tmp);
    HORAS=val%256;            /* HORAS */
    Tmp[0]=BufRx[10];
    Tmp[1]=BufRx[11];
    Tmp[2]=0;
                val = atol(Tmp);
    MINUTOS=val%256;          /* MINUTOS */
    Tmp[0]=BufRx[12];
    Tmp[1]=BufRx[13];
    Tmp[2]=0;
                val = atol(Tmp);
    SEGUNDOS=val%256;         /* SEGUNDOS */
    BufSPI1[0]=0x8;           /*caracter para envio GLOBAL*/
    BufSPI1[1]=C_DRTC;
    BufSPI1[2]=DIA;
    BufSPI1[3]=MES;
    BufSPI1[4]=ANO;
    BufSPI1[5]=HORAS;
    BufSPI1[6]=MINUTOS;
    BufSPI1[7]=SEGUNDOS;
    BufSPI1[8]=C_FMSG;
    SPIDAT=C_IMSG;
    Fcps2  |= 0X1;            /*fBufSPI1=lleno*/
    ABufSPI1=0;               /*INI APUNTADOR*/
    while(Fcps2 & 0X1) {};
    IniRTC(SEGUNDOS, MINUTOS, HORAS, DIA, MES, ANO, (ANO%4), 1);
    FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
    AdRx=0;
    TXBUF=XON;
}
/*RUTINA DE ENVIO DE RELOJ DE TIEMPO REAL DE SCE-1000 A PC*/
void Get_RTC(void)
{
    char Tmp[5];
    char Tmp1[8];
    unsigned int val;
    ClearRow(1);
```

```
    WriteString(1,1,"SOLICITUD DE RTC");
    FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
    AdRx=0;
    TXBUF=XON;
    ReadRTC(Tmp1);
    BufTx[0]=C_DRTC;
    val=(0+Tmp1[4]);    /*DIA*/
    ltoa(val,Tmp);
    BufTx[1]=0x30;
    BufTx[2]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[1]=Tmp[0]; BufTx[2]=Tmp[1];}
    val=(0+Tmp1[5]);    /*MES*/
    ltoa(val,Tmp);
    BufTx[3]=0x30;
    BufTx[4]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[3]=Tmp[0]; BufTx[4]=Tmp[1];}
    val=(0+Tmp1[6]);    /*AÑO*/
    ltoa(val,Tmp);
    BufTx[5]=0x30;
    BufTx[6]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[5]=Tmp[0]; BufTx[6]=Tmp[1];}
    val=(0+Tmp1[3]);    /*HORA*/
    ltoa(val,Tmp);
    BufTx[7]=0x30;
    BufTx[8]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[7]=Tmp[0]; BufTx[8]=Tmp[1];}
    val=(0+Tmp1[2]);    /*MINUTOS*/
    ltoa(val,Tmp);
    BufTx[9]=0x30;
    BufTx[10]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[9]=Tmp[0]; BufTx[10]=Tmp[1];}
    val=(0+Tmp1[1]);    /*SEGUNDOS*/
    ltoa(val,Tmp);
    BufTx[11]=0x30;
    BufTx[12]=Tmp[0];
    if(strlen(Tmp)==2) {BufTx[11]=Tmp[0]; BufTx[12]=Tmp[1];}
    BufTx[13]=C_FMSG;
    AdTx=0;
    FcSCI |= 0x2;  /*fBufTx=On*/
    TXBUF=C_IMSG;
    TXCTL = 0X1;    /*habilita interrupción de Tx*/
}
/*RUTINA DE ENVIO DE LECTURAS A PC*/
void Env_Inf(void)
{
    char Tmp[5];
    char dat;
    unsigned int val,aux;
    BYTE cont, cont1, cont2, segundero, memseg, contseg;
    FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
    AdRx=0;
    TXBUF=XON;
    ClearRow(1);
    WriteString(1,1,"DESCARGA MEMORIA");
    ClearRow(2);
    WriteString(2,1,"uC:  LECT:");
    BufSPI1[0]=0x8;     /*caracter para envio GLOBAL*/
    BufSPI1[1]=C_EINF;
    BufSPI1[2]=C_FMSG;
    SPIDAT=C_IMSG;
    Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
    ABufSPI1=0;      /*INI APUNTADOR*/
    while(Fcps2 & 0X1) {};
    BufTx[0]=C_CLT;
    val= abs(Esp_uclect());
    aux=val;
    BufTx[1]=0x30;
    BufTx[2]=0x30;
    BufTx[3]=0x30;
    BufTx[4]=0x30;
    BufTx[5]=0x30;
    ltoa(val,Tmp);
    cont=strlen(Tmp);
```

```
   cont1=5;
   cont2=(cont-1);
   while(cont)
     {
       BufTx[cont1]=Tmp[cont2];
       cont--;
       cont1--;
       cont2--;
     }
   BufTx[6]=C_FMSG;
   AdTx=0;
   FcSCI |= 0x2;  /*fBufTx=On*/
   TXBUF=C_IMSG;
   TXCTL = 0X1;      /*habilita interrupción de Tx*/
   segundero=30;
   memseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
   contseg=memseg;
   while(FcSCI&0x2)
         {
           if (memseg!=contseg) {memseg=(((TSRTC&0xF)*10)+(USRTC&0xF)); segundero--;}
           contseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
           if (!segundero) return;
         }
   if(!val){return;}
   Esp_uclect();         /*envia datos de uC lects a pc*/
   ClearRow(1);
   segundero=30;
   memseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
   contseg=memseg;
   while(!(FcSCI&0x1))
     {
       if (memseg!=contseg) {memseg=(((TSRTC&0xF)*10)+(USRTC&0xF)); segundero--;}
       contseg=(((TSRTC&0xF)*10)+(USRTC&0xF));
       if (!segundero) { return;}
     }
   dat=(255-BufRx[1]);  /*checa si es comando C_BLT=249*/
   WriteChar(1,1,'A');
   if ((dat!=6)&&(dat!=25))  { FcSCI &= 0xF6; AdRx=0; TXBUF=XON; return;}
   Tmp[0]=BufRx[2];
   Tmp[1]=BufRx[3];
   Tmp[2]=BufRx[4];
   Tmp[3]=BufRx[5];
   Tmp[4]=BufRx[6];
   Tmp[5]=0;
               val = atol(Tmp);
   WriteChar(1,2,'B');
   if (val!=aux) { FcSCI &= 0xF6; AdRx=0; TXBUF=XON; return;}
   switch(dat)
    {
     case  6  : {Rst_Flash(); break;}
     case  25 : {Ok_Flash();  break;}
    }
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
   TXBUF=XON;
}

/*RUTINA DE BORRADO DE MEMORIAS FLASH DE uC LECT*/
void Rst_Flash(void)
{
   unsigned int cont, i;
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
   TXBUF=XON;
   ClearDisplay();
   WriteString(1,1,"BORRADO DE LECT.");
   WriteString(2,1,"T. aprox=2 min.");
   BufSPI1[0]=0x8;      /*caracter para envio GLOBAL*/
   BufSPI1[1]=C_RSTFLASH;
   BufSPI1[2]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
```

```
   while(Fcps2 & 0X1) {};
   cont=0x170;    /*define tiempo de espera de 90 seg*/
   while(cont)
    {
      for (i = 0; i < 0xff5f; ++ i) ; /*espera 240 mS para */
      cont-=1;
    }
   Esp_uclect();
}
/*RUTINA DE BORRADO DE MEMORIAS FLASH DE uC LECT*/
void Ok_Flash(void)
{
   unsigned int cont, i;
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
   TXBUF=XON;
   ClearDisplay();
   WriteString(1,1,"CONFIRM DE RX");
   WriteString(2,1,"DE LECT FLASH");
   BufSPI1[0]=0x8;      /*caracter para envio GLOBAL*/
   BufSPI1[1]=C_RSTFLASH;
   BufSPI1[2]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   cont=0x1;    /*define tiempo de espera de 1 seg*/
   while(cont)
    {
      for (i = 0; i < 0xff5f; ++ i) ; /*espera 471mS para */
      cont-=1;
    }
   Esp_uclect();
}

/*RUTINA DE ACTIVACION DE REGISTRO DE LECTURAS DE TRPs*/
void Act_Lect(void)
{
   unsigned int cont, i;
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
   TXBUF=XON;
   ClearDisplay();
   WriteString(1,1,"ACTIVACION DE R");
   BufSPI1[0]=0x8;      /*caracter para envio GLOBAL*/
   BufSPI1[1]=C_ACTL;
   BufSPI1[2]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   cont=0x2;    /*define tiempo de espera de 1 seg*/
   while(cont)
    {
      for (i = 0; i < 0xff5f; ++ i) ; /*espera 471mS para */
      cont-=1;
    }
}

/*RUTINA DE ENVIO DE COMANDO A uC*/
void Comando(void)
{
   BYTE z, cont, cont1, val, seg1, seg2, seg3, esp, Lector;
   BufSPI1[0]=(BufRx[2]-0X30);  /*almacena num de uC lect*/
   Lector=BufSPI1[0];
   BufSPI1[1]=BufRx[1];  /*almacena comando*/
   ClearRow(1);
   WriteString(1,1,"TX:   ID uC:");
               WriteChar(1,4,BufRx[1]);
               WriteChar(1,13,BufRx[2]);
   cont=2;
   cont1=3;
   z=1;
```

```
while(z)
  {
    BufSPI1[cont]=BufRx[cont1];
    z=(252-BufSPI1[cont]);
    cont+=1;
    cont1+=1;
  }
FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
AdRx=0;
TXBUF=XON;
SPIDAT=C_IMSG;
Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
ABufSPI1=0;       /*INI APUNTADOR*/
while(Fcps2 & 0X1) {};
seg2=USRTC;
seg2&=0xF;
seg1=(seg2+3);
seg1&=0xF;
if(seg1>9) seg1=2;
while(seg1!=seg2){seg2=USRTC; seg2&=0xF;}
seg3=1;
if(Lector==8)  {seg3=6; Lector=0;}
while(seg3)
   {
     BufSPI1[0]=Lector;          /*caracter para solicitar conformacion*/
     Fcps21&=0xFE;
     FcSCI &= 0x9F;  /*FRXOK=FRXRPT=OFF*/
     BufSPI1[1]=C_BUFOK;
     BufSPI1[2]=C_FMSG;
     SPIDAT=C_IMSG;
     Fcps2  |= 0X1;           /*fBufSPI1=lleno*/
     ABufSPI1=0;              /*INI APUNTADOR*/
     while(Fcps2 & 0X1) {};
     val=200;
     while(val) {val--;}
     SPIDAT=C_DUMMY;       /* TRANSMITE DATO A SLAVE PARA RX CARACTER */
     Fcps2 &= 0x7F;        /* fTXDUMMY=OFF*/
     while(!( Fcps2 & 0x80 ) ){};
     val=10;
     while(val) {val--;}
     if (Fcps2  & 0X62)  {Fcps2  &= 0X9D; goto SALRCOM;} /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
     if (!(Fcps21  & 0X1))       /*no llego informacion*/
       {
         goto SALRCOM;
       }
     seg2=USRTC;
     esp=1;
     seg2&=0xF;
     seg1=(seg2+3);
     seg1&=0xF;
     if(seg1>9) seg1=2;
     while( esp && (seg1!=seg2) )  {seg2=USRTC; seg2&=0xF; esp=(Fcps21&0x1);}
     BufTx[0]=BufSPI2[2];
     BufTx[1]=BufSPI2[1];
     cont=2;
     z=1;
     while(z)
        {
          BufTx[cont]=BufSPI2[cont+1];
          z=(252-BufTx[cont]);
          cont+=1;
        }
     AdTx=0;
     FcSCI |= 0x2;  /*fBufTx=On*/
     BufTx[1]+=0X30;
     TXBUF=C_IMSG;
     TXCTL = 0X1;      /*habilita interrupción de Tx*/
     while(FcSCI&0x2) {};
     seg1=7;
     while(seg1==7)
        {
          if(FcSCI&0x20) seg1=3;
          if(FcSCI&0x40) seg1=6;
```

```
          }
        if(FcSCI&0x20)
            {
             SALRCOM:
             BufSPI1[0]=Lector;      /*caracter para solicitar conformacion*/
             BufSPI1[1]=C_RXOK;
             Fcps21&=0xFE;
             BufSPI1[2]=C_FMSG;
             SPIDAT=C_IMSG;
             Fcps2  |= 0X1;     /*fBufSPI1=lleno*/
             ABufSPI1=0;        /*INI APUNTADOR*/
             while(Fcps2 & 0X1) {};
             seg3-=1;
             Lector+=1;
             val=100;        /*TIEMPO PARA QUE SLAVE CHEQUE SI TIENE INF A TX*/
             while(val) {val--;}
            }
      }
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
}
/*RUTINA DE ENVIO DE LECTURAS ENTRE uCLECT Y PC*/
void Operacion(void)
{
   BYTE seg1, esp, seg2, uC, FcLS, cont, cont1, dat, val, edo=(BufRx[2]-0x30);
   BYTE DAT;
   FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
   AdRx=0;
   DAT=ESC;
   val=BufRx[1];
   BufTx[0]=C_IMSG;
   BufTx[1]=BufRx[1];
   BufTx[2]=BufRx[2];
   BufTx[3]=C_FMSG;
   AdTx=0;
   FcSCI |= 0x2;         /*fBufTx=On*/
   TXBUF=XON;
   TXCTL = 0X1;          /*habilita interrupción de Tx*/
   if(val==C_ELINEL) DAT='L';
   ClearDisplay();
   WriteString(1,1,"LECTURA DE TRPs ");
   WriteString(2,1,"MODO:  NUMERO:  ");
   WriteChar(2,6,DAT);
   WriteChar(2,15,edo+0x30);
   BufSPI1[0]=0x8;      /*caracter para envio GLOBAL*/
   BufSPI1[1]=DAT;
   BufSPI1[2]=edo;
   BufSPI1[3]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
   uC=0;             /*INI APUNT uC*/
   while(Fcps2 & 0X1) {};
   Esp_uclect();
   ClearDisplay();
   if ((edo==1)||(edo==2)) return;
   WriteString(1,1,"ACTIVAR REGISTRO");
   uC=0;
   while(1)
    {
     if(FcSCI&0x1)   /*CHECA SI LLEGO INF DE PC*/
      {
       dat=(255-BufRx[1]);
       WriteChar(1,15,dat+0x30);
       if(dat==13)    /*checa si es comando C_SLINE=242*/
        {
         FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
         AdRx=0;
         TXBUF=XON;           /*HABILITA RX DE PC*/
         BufTx[0]=C_IMSG;
         BufTx[1]=C_SLINE;
         BufTx[2]=C_FMSG;
         AdTx=0;
```

```
          FcSCI |= 0x2;         /*fBufTx=On*/
          TXBUF=XON;
          TXCTL = 0X1;          /*habilita interrupción de Tx*/
          while(FcSCI & 0x2) {};
          BufSPI1[0]=0x8;      /*caracter para envio GLOBAL*/
          BufSPI1[1]='X';
          BufSPI1[2]=C_FMSG;
          SPIDAT=C_IMSG;
          Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
          ABufSPI1=0;       /*INI APUNTADOR*/
          uC=0;            /*INI APUNT uC*/
          while(Fcps2 & 0X1) {};
          Esp_uclect();
          return;
        }
        if(dat==7)    /*checa si es comando C_DRTC=248 CONFIG RTC*/
        {
         Config_RTC();
         TXBUF=XOFF;           /*DESHABILITA RX DE PC*/
        }
        if(dat==24)    /*checa si es comando C_ACTL=231 ACIVA REGISTRO DE LECT*/
        {
         Act_Lect();
         TXBUF=XOFF;           /*DESHABILITA RX DE PC*/
        }
       FcSCI &= 0xF6;  /*fBufRx=OFF, fERRORRX=OFF*/
       AdRx=0;
       TXBUF=XON;           /*HABILITA RX DE PC*/
     }
    FcLS = erByte(EE_SIST);  /*lee banderas de luz y sonido*/
    ClearDisplay();
    WriteString(1,1,"Antena:");
    WriteChar(1,8,(uC+0x31));
    BufSPI1[0]=uC;      /*caracter para solicitar conformacion*/
    BufSPI1[1]=C_BUFOK;
    Fcps21&=0xFE;
    BufSPI1[2]=C_FMSG;
    SPIDAT=C_IMSG;
    Fcps2  |= 0X1;     /*fBufSPI1=lleno*/
    ABufSPI1=0;       /*INI APUNTADOR*/
    while(Fcps2 & 0X1) {};
    val=100;        /*TIEMPO PARA QUE SLAVE CHEQUE SI TIENE INF A TX*/
    while(val) {val--;}
    SPIDAT=C_DUMMY;       /* TRANSMITE DATO A SLAVE PARA RX CARACTER */
    Fcps2 &= 0x7F;        /* fTXDUMMY=OFF*/
    while(!( Fcps2 & 0x80 )){};
    val=10;
    while(val) {val--;}
    if (Fcps21  & 0X1)
      {
       seg2=USRTC;
       esp=1;
       seg2&=0xF;
       seg1=(seg2+3);
       seg1&=0xF;
       if(seg1>9) seg1=2;
       while( esp && (seg1!=seg2) )  {seg2=USRTC; seg2&=0xF; esp=(Fcps21&0x1);}    /*SI NO LLEGA INF NO PUSE NADA PARA
VERIFICAR*/
        switch(BufSPI2[1])
         {
          case 0 : case 1: case 2: case 3: case 4: case 5:
            {
              if (FcLS&0x2) T1PC2 |= 0X4;   /*on sonido si esta configurado*/
              FcSCI &= 0x9F;  /*FRXOK=FRXRPT=FBUBRX=OFF*/
              BufTx[0]=BufSPI2[2];
              BufTx[1]=BufSPI2[1];
              BufTx[1]+=0X30;
              cont=2;
              cont1=3;
              dat=1;
              while(dat)
                  {
                    BufTx[cont]=BufSPI2[cont1];
```

```
                    dat=252-BufTx[cont];
                    cont++;
                    cont1++;
                  }
              BufSPI2[26]=0;
              WriteString(2,1,BufSPI2+10);
              AdTx=0;
              FcSCI |= 0x2;  /*fBufTx=On*/
              TXBUF=C_IMSG;
              TXCTL = 0X1;      /*habilita interrupción de Tx*/
              cont=7;
              while(cont==7)
                     {
                      if(FcSCI&0x20) cont=3;
                      if(FcSCI&0x40) cont=6;
                      if(FcLS&0x2) T1PC2 &= 0XFB;   /*OFF SONIDO SI ESTA CONFIGURADO*/
                     }
              if(FcSCI&0x20)
                 {
                    BufSPI1[0]=uC;      /*caracter para solicitar conformacion*/
                    BufSPI1[1]=C_RXOK;
                    Fcps21&=0xFE;
                    BufSPI1[2]=C_FMSG;
                    SPIDAT=C_IMSG;
                    Fcps2  |= 0X1;     /*fBufSPI1=lleno*/
                    ABufSPI1=0;        /*INI APUNTADOR*/
                    while(Fcps2 & 0X1) {};
                    val=100;       /*TIEMPO PARA QUE SLAVE CHEQUE SI TIENE INF A TX*/
                    while(val) {val--;}
                 }
            }
          }
        }
     if (Fcps2  & 0X62)   Fcps2  &= 0X9D;   /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
     uC+=1;
     if(FcLS&0x2) T1PC2 &= 0XFB;   /*OFF SONIDO SI ESTA CONFIGURADO*/
     if (uC==6) uC=0;
   }
}
void RS232(void)
{
  BYTE Head=BufRx[1];
              switch (Head)
              {
                        case C_DRTC     : Config_RTC(); break;
                        case C_SRTC     : Get_RTC();    break;
                        case C_EINF     : Env_Inf();    break;
                        case C_ACTL     : Act_Lect();   break;
                        case C_RSTFLASH : Rst_Flash();  break;
                        case C_AVZ      : Ok_Flash();   break;
                        case C_ELINE    : case C_ELINEL   :Operacion();  break;
                        case 'C' : case 'B' : case 'J' : case 'Y' : case 'H' : case 'Z' :
     case 'X' :case 'P' : Comando();  break;
     default:         {FcSCI &= 0xF6; AdRx=0; TXBUF=XON;}
              }
}

/*****************************************************************************/
/***********                RUTINAS VARIAS                ******************/
/*****************************************************************************/
/*RUTINAS DE SOPORTE TECNICO*/
void ConfigSCI(void)
{
    BYTE key;
    unsigned int x;
              ClearDisplay();
    x= TBufTx+0;
    WriteString(1,1,"Pulso torniquete");
              if (!ReadNumber(2,1,"1-255(20mS)>",1,255,&x)) return;
              eWrite(EE_TBufTx,x);
              TBufTx= x;
              return;
}
```

```
void TechSup(void)
{
 BYTE key, ptr;
 char tmp[11];
 char clave[]="950512";
 ClearDisplay();
 WriteString(1,1,"Sop. Tec.");
 key=GetKey();
 if(key != VK_RETURN) return;
 ClearDisplay();
 tmp[0]=0;
 if (!ReadString(1,1,"Clave>",6,tmp)) return;
 ptr=0;
 while(ptr!=6)
  {
   if (tmp[ptr]!=clave[ptr]) return;
   ptr++;
  }
 SOPTEC:
 ClearDisplay();
 WriteString(1,1,"Op. F1-F7?");                /* se cambio F4 por F11  */
 key=GetKey();
 switch(key)
 {
              case VK_F1    :  { ConfigSCI();   goto SOPTEC; }
              case VK_F2    :  { Rst_Flash();   goto SOPTEC; }
              case VK_CLEAR :  { return; }
 }
 goto SOPTEC;
}

/*RUTINA DE ENVIO DE COMANDOA uC*/
void OnAntena(BYTE bit_Antenas, BYTE Lector)
{
   BYTE seg1, esp, seg2, cont=0, val, edo=0;
   unsigned int i;
   BufSPI1[0]=Lector;  /*almacena num de uC lect*/
   BufSPI1[1]='X';     /*almacena com de apagado de antena*/
   if( !(bit_Antenas & Antenas) )  { Antenas|=bit_Antenas; BufSPI1[1]=ESC;  BufSPI1[2]=1; BufSPI1[3]=C_FMSG; edo=1;}
   if( BufSPI1[1]=='X') {BufSPI1[2]=C_FMSG; val=(255-bit_Antenas); Antenas&=val;}
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   ClearRow(1);
   WriteString(1,1,"Antena   Off");
   cont=(Lector+0x30);
   WriteChar(1,8,cont);
   if(edo)  WriteString(1,10,"On ");
   cont=10;
   while(cont)
    {
      for (i = 0; i < 0xff5f; ++ i) ; /*espera 15 mS para */
      cont-=1;
    }
   BufSPI1[0]=Lector;          /*caracter para solicitar conformacion*/
   Fcps21&=0xFE;
   BufSPI1[1]=C_BUFOK;
   BufSPI1[2]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;           /*fBufSPI1=lleno*/
   ABufSPI1=0;              /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   val=200;
   while(val) {val--;}
   SPIDAT=C_DUMMY;       /* TRANSMITE DATO A SLAVE PARA RX CARACTER */
   Fcps2 &= 0x7F;        /* fTXDUMMY=OFF*/
   while(!( Fcps2 & 0x80 ) ){};
   val=10;
   while(val) {val--;}
   if (Fcps2  & 0X62)  {Fcps2  &= 0X9D; return;} /*fBufSPI2=VACIO, FNOBUF=OFF, FNOERROR=OFF*/
   if (!(Fcps21  & 0X1))  return;       /*no llego informacion*/
```

```
   seg2=USRTC;
   esp=1;
   seg2&=0xF;
   seg1=(seg2+3);
   seg1&=0xF;
   if(seg1>9) seg1=2;
   while( esp && (seg1!=seg2) )  {seg2=USRTC; seg2&=0xF; esp=(Fcps21&0x1);}
   Fcps2  &= 0XFD;                    /*fBufSPI2=VACIO*/
}
void ConfigRTC(void)
{
   unsigned int x,i;
   BOOL def=TRUE;
   BYTE DIA, MES, ANO, HORAS, MINUTOS, SEGUNDOS, NuC;
   WriteString(1,1,"Conf. RTC ");
   CRRTC = 0x0F; /*TEST MODE, CLOCK STOP, INTERRUPT REG, INTERRUPT STOP*/
   CSRTC = 0x0;  /*DESHABILITA INTERRUPCIONES*/
   SEGUNDOS=0;
   x=0;
            if (ReadNumber(2,1,"Segundos>",0,59,&x)) SEGUNDOS=x%256;
   CRRTC = 0x05; /*NORMAL MODE, CLOCK STOP, CLOCK SETTING REG, INTERRUPT STOP*/
   TSRTC = SEGUNDOS/10;
   USRTC = SEGUNDOS%10;
   MINUTOS=0;
   x=0;
            if (ReadNumber(2,1,"Minutos>",0,59,&x)) MINUTOS=x%256;
   TMRTC = MINUTOS/10;
   UMRTC = MINUTOS%10;
   HORAS=0;
   x=0;
            if (ReadNumber(2,1,"Horas>",0,23,&x)) HORAS=x%256;
   THRTC = HORAS/10;
   UHRTC = HORAS%10;
   DIA=1;
   x=1;
            if (ReadNumber(2,1,"Dia>",1,31,&x)) DIA=x%256;
   TDRTC = DIA/10;
   UDRTC = DIA%10;
   MES=1;
   x=1;
            if (ReadNumber(2,1,"Mes>",1,12,&x)) MES=x%256;
   TMORTC = MES/10;
   UMORTC = MES%10;
   ANO=1;
   x=1;
            if (ReadNumber(2,1,"Ano>",0,99,&x)) ANO=x%256;
   TYRTC = ANO/10;
   UYRTC = ANO%10;
   CSRTC = (((ANO%4)*4)+1);
   BufSPI1[0]=0x8;          /*caracter para envio GLOBAL*/
   BufSPI1[1]=C_DRTC;
   BufSPI1[2]=DIA;
   BufSPI1[3]=MES;
   BufSPI1[4]=ANO;
   BufSPI1[5]=HORAS;
   BufSPI1[6]=MINUTOS;
   BufSPI1[7]=SEGUNDOS;
   BufSPI1[8]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;           /*fBufSPI1=lleno*/
   ABufSPI1=0;              /*INI APUNTADOR*/
   NuC=0;                  /*INI APUNT uC*/
   while(Fcps2 & 0X1) {};
   CRRTC = 0x01; /*NORMAL MODE, CLOCK RUN, CLOCK SETTING REG, INTERRUPT STOP*/
   Esp_uclect();
}
void Sensor(void)
{
   unsigned int x,i;
   BOOL def=TRUE;
   ClearDisplay();
   if (!Question(1,1,"Activa sensor? ",def,TRUE)) {return;}
   ClearRow(1);
```

```
   WriteString(1,1,"SENSOR LISTO");
   BufSPI1[0]=0x8;          /*caracter para envio GLOBAL*/
   BufSPI1[1]=C_DRTC;
   BufSPI1[2]=1;
   BufSPI1[3]=1;
   BufSPI1[4]=1;
   BufSPI1[5]=0;
   BufSPI1[6]=0;
   BufSPI1[7]=0;
   BufSPI1[8]=C_FMSG;
   while(!(INT1&0x40)) {};
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;           /*fBufSPI1=lleno*/
   ABufSPI1=0;              /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   IniRTC(0, 0, 0, 1, 1, 1, 1, 1);
   Esp_uclect();
   BufSPI1[0]=0x8;  /*almacena num de uC lect*/
   BufSPI1[1]=ESC;
   BufSPI1[2]=1;
   BufSPI1[3]=C_FMSG;
   SPIDAT=C_IMSG;
   Fcps2  |= 0X1;    /*fBufSPI1=lleno*/
   ABufSPI1=0;       /*INI APUNTADOR*/
   while(Fcps2 & 0X1) {};
   Antenas=0x3F;   /*enciende bits de banderas*/
   Esp_uclect();
}
```

FLASH.CPP

```
#include "FLASH.H"
#include <ports.h>
#include "SCI.H"
#include "OS.H"

void InitFlash(void)
{
 BPORT2 = 0XFF;   /*puerto B definido como bus de direcciones FLASH */
 CPORT2 = 0XFF;   /*puerto C definido como bus de direcciones FLASH */
 DDATA = 0X80;    /*inicialización de los datos del puerto D FLASH */
 DDIR  = 0XCF;     /*bit0-3, 6 y 7 definidos como salida digital FLASH */
}

/*****************************************************************************/
/***********              Manejo de Tablas en Flash          ************/
/*****************************************************************************/

unsigned int frInt(unsigned int Pagina, BYTE Offset)
{
 unsigned int x=frByte(Pagina,Offset)*256;
 Offset++;
 x += frByte(Pagina,Offset);
 Offset--;
 return x;
}

BYTE frStr(unsigned int Pagina, BYTE Offset, char* Buffer, BYTE SizeBuffer)
{
 BYTE cont=0;
 do
 {
   Buffer[cont]=frByte(Pagina,Offset);
   if (frByte(Pagina,Offset)==0xFF) Buffer[cont]=0;
   Offset++;
              cont++;
 }
 while ((Buffer[cont-1])||(cont==SizeBuffer));
 return(cont-1);
}

void frPage(unsigned int Pagina, BYTE Limite, char* Buffer)
{
```

```
 BYTE offset=0;
 do
 {
  Buffer[offset]=frByte(Pagina,offset);
  offset++;
            Limite--;
 }
 while (Limite);
}

void  fwPage(unsigned int Pagina, char* data, BYTE len)
{
 BYTE Offset;

 for (Offset=0;Offset<len;Offset++) fwByte(Pagina,Offset,data[Offset]);
}
```

INTVECS.ASM

```
;*******************************************************************************
;***** RUTINAS DE ATENCION A LA INTERRUPCION DE SPI PARA PS2            *****
;*******************************************************************************
;*******************************************************************************
; declaración de registros y perifericos
;*******************************************************************************
SPICCR     .equ P030  ; REG. 1 DE CONF. DE SPI (# CARACT., VEL., POLARIDAD,RST)
SPICTL     .equ P031  ; REG. 2 DE CONF. DE SPI (INTERRUPCION, MASTER/SLAVE)
SPIBUF     .equ P037  ; REGISTRO DE DATO RECIBIDO EN SPI
SPIDAT     .equ P039  ; REGISTRO SERIAL DE DATO (DATO TX. POR SPI)
SPIPC1     .equ P03D  ; REGISTRO DE CONTROL DEL PIN CLOCK SPICLK
SPIPC2     .equ P03E  ; REGISTRO DE CONTROL DE LOS PINES SPISIMO Y SPISOMI
SPIPRI     .equ P03F  ; REGISTRO DE CONTROL DE PRIORIDADES
C_IMSG      .equ 254   ; VALOR DE COMANDO DE INICIO DE MENSAJE
C_EINF      .equ 253   ; VALOR DE COMANDO DE ENVIO DE LECTURAS
C_FMSG      .equ 252   ; VALOR DE COMANDO DE FIN DE MENSAJE
C_CLT       .equ 251   ; VALOR DE COMANDO DE CONTADOR DE LECTURAS
C_LECT      .equ 250   ; VALOR DE COMANDO DE LECTURA
C_BLT       .equ 249   ; VALOR DE COMANDO DE BORRADO DE LECTURAS ALMACENADAS
C_DRTC      .equ 248   ; VALOR DE COMANDO DE CONFIGURACION DE RELOJ DE TIEMPO REAL (RTC)
C_ACG       .equ 247   ; VALOR DE COMANDO DE AVISO DE CARGA DE MEMORIA
C_CMEM      .equ 246   ; VALOR DE COMANDO DE CARGA DE MEMORIA
C_SRTC      .equ 245   ; VALOR DE COMANDO DE SOLICITUD DE RTC DE UNIDAD
C_RSTFLASH  .equ 244   ; COMANDO DE SOLICITUD DE RESET FLASH
C_ELINE     .equ 243   ; VALOR DE COMANDO DE SOLICITUD DE ENTRAR A EDO LINEA
C_SLINE     .equ 242   ; VALOR DE COMANDO DE SOLICITUD DE SALIR EDO LINEA
C_PRG       .equ 241   ; COMANDO DE PROGRAMACION DE TRANSPONDER
C_BUFOK     .equ 240   ; COMANDO DE SOLICITAR INF DE uC LECTOR
C_NOBUF     .equ 239   ; COMANDO DE AVISAR A uC INT QUE NO HAY INF
C_ERRORBUF  .equ 238   ; VALOR DE COMANDO DE QUE SE ENV A MASTER P/AVISAR QUE BUF SE ENVIO MAL
C_DUMMY     .equ 237   ; VALOR DE COMANDO DE SEPARACION DE CAMPOS
C_ELECT     .equ 236   ; VALOR DE COMANDO DE AVISO A uC LECT QUE LECT NO EJECUTO COMANDO
C_ONT       .equ 235   ; VALOR DE COMANDO DE AVISO DE ENCENDIDO DE TORNIQUETE
C_ELINEL    .equ 234   ; VALOR DE COMANDO DE SOLICITUD DE ENTRAR A EDO LINEA MODO LINE
C_RXOK      .equ 233   ; VALOR DE COMANDO DE CONFIRMACION DE RX CORRECTA DE INF
C_RXRPT     .equ 232   ; VALOR DE COMANDO DE AVISO DE REPETRIR ENVIO DE ULTMO REGISTRO
C_ACTL      .equ 231   ; VALOR DE COMANDO DE ACTIVACION DE REGISTRO DE LECTURAS
C_AVZ       .equ 230   ; VALOR DE COMANDO DE AVANCE DE COLA EN MODO 1 DE LECT
        .bss _BufSPI1,22       ; DEFINE BUFFER DE CARACT. PARA SPI
        .global _BufSPI1       ; DECLARA BUFFER EXTERNO
        .bss _BufSPI2,42       ; DEFINE BUFFER DE CARACT. PARA SPI
        .global _BufSPI2       ; DECLARA BUFFER EXTERNO
                .reg _Ntci,1          ; DEFINE VARIABLE PARA NUMERO DE uC
        .globreg _Ntci         ; DECLARA VARIABLE EXTERNA
                .reg _Fcps2,1         ; DEFINE VARIABLE PARA BANDERAS DE CTRL. RUT PS2
        ; x0=fBufSPI1(edo. buf. 0-vacio 1-lleno), x1=fBufSPI2(edo. buf. 0-vacio 1-lleno),
        ; x2=error spibuf (0=dato ok 1=dato sobreescrito) x3=contestar a uC int
        ; x4=master/slave(0=master 1=slave) x5=RX NOBUF EN MASTER(0=NO RX 1=RX)
        ; x6=RX ERRORBUF EN MASTER(0=NO RX 1=RX) x7=FTXDUMMY (1=TERMINO DE TX DUMMY)
        .globreg _Fcps2        ; DECLARA VARIABLE EXTERNA
                .reg _Fcps21,1         ; DEFINE VARIABLE PARA BANDERAS DE CTRL. RUT PS2
        ; x0=fESPINFMASTER(0-NO ESP 1-ESPERA INF), x1=LIBRE,
        ; x2=LIBRE x3=LIBRE
        ; x4=LIBRE x5=LIBEE
        ; x6=LIBRE x7=LIBRE
```

```
        .globreg _Fcps21        ; DECLARA VARIABLE EXTERNA
        .reg _ABufSPI1,1        ; APUNTADOR DE BUFFER SPI 1
        .globreg _ABufSPI1      ; DECLARA VARIABLE EXTERNA
        .reg _ABufSPI2,1        ; APUNTADOR DE BUFFER SPI 2
        .globreg _ABufSPI2      ; DECLARA VARIABLE EXTERNA

;
;*******************************************************************************
; RUTINA DE INTERFAZ PS2
;*******************************************************************************
        .global COMPS2
COMPS2:     PUSH A                  ; ALMACENA EN STACK REG A Y B
        PUSH B
        BTJZ #080h,SPICTL,NOOVERRUN  ; SE SOBREESCRIBIO SPIBUF?
        OR #04h,_Fcps2              ; fEBUF=ON
NOOVERRUN:  MOV SPIBUF,A                ; LEE DATO DE BUF SPI (PS2)
        BTJZ #010h,_Fcps2,MASTER    ; CHECA SI uC ES MASTER?
        CMP #C_DUMMY,A              ; CHECA SI ESPERA uC INT ESPERA DATOS
        JNE RXINISPI
        BTJZ #08h,_Fcps2,SALIRSPI   ; CONTESTAR A uC INT?
        BTJZ #02h,_Fcps2,TXNOINFSPI  ; HAY INF A TX DE SLAVE?
        MOV _ABufSPI2,B
        MOV *_BufSPI2[B],A          ; LEE DATO
        MOV A,SPIDAT                ; TRANSMITE DATO A MASTER
        INC _ABufSPI2
        CMP #C_FMSG,A               ; CHECA SI ERA ULTIMO DATO A TX
        JNE SALIRSPI
WAITSPI:    BTJZ #040h,SPICTL,WAITSPI   ; ESPERA A QUE TERMINE DE TRANSMITIR
        MOV SPIBUF,A                ; LLE DATO PARA APAGAR INT
TXNOINFSPI: AND #0F7h,_Fcps2            ; CONTESTAR A uCINT=OFF
        AND #0FDh,SPICTL            ; APAGA TALK BIT
SALIRSPI:   POP B                   ; REGRESA VALORES DE STACK
        POP A
        RTI
RXINISPI:   CMP #C_IMSG,A               ; CHECA SI ES DATO DE INI DE CADENA
        JNE RXFINSPI
        BTJZ #01h,_Fcps2,RXINISPI1   ; BUF1(RX SLAVE) LLENO?
        OR #04h,_Fcps2              ; fEBUF=ON
        JMP SALIRSPI
RXINISPI1:  CLR _ABufSPI1               ; INI APUNT DE BUF SPI (RX SLAVE)
RXINISPI2:  MOV _ABufSPI1,B
        MOV A,*_BufSPI1[B]          ; GUARDA DATO
        INC _ABufSPI1               ; INC APUNTADOR
        JMP SALIRSPI
RXFINSPI:   BTJO #01h,_Fcps2,SALIRSPI    ; BUFFER DE RX LLENO?
        CMP #C_FMSG,A               ; CHECA SI ES DATO DE FIN DE CADENA
        JNE RXINISPI2
        MOV #01h,B
        MOV *_BufSPI1[B],A          ; LEE DATO DE DIRECCION DE uC
        CMP A,_Ntci                 ; CHECA SI PREGUNTA ES PARA ESTE uC
        JEQ RXFINSPI8
        CMP #08h,A                  ; CHECA SI PREGUNTA ES GLOBAL
        JNE TXNOINFSPI
RXFINSPI8:  INC B                       ; APUNTA A COMANDO
        MOV *_BufSPI1[B],A          ; LEE DATO DE COMANDO
        CMP #C_BUFOK,A              ; CHECA SI SOLICITA CONFIRMACION DE ALGUN COMANDO ANTERIOR
        JEQ RXINFSPI3
        CMP #C_RXOK,A               ; CHECA SI AVISA CONFIRM. DE ENVIO CORRECTO DE LECTURA TRP
        JNE RXINFSPI8A
        AND #0FDh,_Fcps2            ; INF TX SLAVE=OFF
        JMP TXNOINFSPI
RXINFSPI8A: BTJZ #04h,_Fcps2,RXINFSPI4   ; HUBO ERROR EN RX?
        AND #0FBh,_Fcps2            ; ERROR EN RX=OFF
        JMP SALIRSPI
RXINFSPI4:  OR #01h,_Fcps2              ; BUF1=LLENO
        JMP RXINISPI2
RXINFSPI3:  BTJZ #02h,_Fcps2,RXINFSPI7   ; BUF2=LLENO?
        MOV #01h,_ABufSPI2          ; INI APUNT DE BUF DE TX A UC INT
        MOV #C_IMSG,A               ; TRANSMITE CARACT. DE INI DE BUF
        JMP RXINFSPI6
RXINFSPI7:  MOV #C_NOBUF,A              ; TRANSMITE CARACT. DE NO BUF PARA ENVIAR
RXINFSPI6:  OR #02h,SPICTL              ; ENCIENDE TALK BIT
        OR #08h,_Fcps2              ; PERMISO DE TX=ON
```

```
                                        MOV A,SPIDAT
        JMP SALIRSPI
; *COMUNICACION SPI MASTER*
MASTER:    OR #080h,_Fcps2          ; FTXDUMMY=ON
        PUSH A
        MPY B,A                  ; INSTRUCCION PARA HACER TIEMPO
        POP A
        BTJZ #01h,_Fcps2,RXMASTER    ; BUF1=LLENO?
        MOV _ABufSPI1,B
        MOV *_BufSPI1[B],A          ; LEE DATO
        MOV A,SPIDAT               ; TRANSMITE DATO A MASTER
        INC _ABufSPI1
        CMP #C_FMSG,A              ; CHECA SI ERA ULTIMO DATO A TX
        JNE SALIRSPI
        AND #0FEh,_Fcps2           ; fBufSPI1=OFF?
        JMP SALIRSPI
RXMASTER:  CMP #C_NOBUF,A
        JNE RXEBUFMAS
        OR #020h,_Fcps2            ; fRX NOBUF=ON
        JMP SALIRSPI
RXEBUFMAS:  CMP #C_ERRORBUF,A
        JNE RXINIMAS
        OR #040h,_Fcps2            ; fRX ERRORBUF=ON
        JMP SALIRSPI
RXINIMAS:  CMP #C_IMSG,A
        JNE RXFINMAS
        AND #0F9h,_Fcps2           ; fBufSPI2=ferrorSPIBUF=OFF
        OR #01h,_Fcps21            ; fESPINFMASTER=1
        CLR _ABufSPI2
RXDATMAS:  BTJZ #01h,_Fcps21,SALIRSPI
        MOV _ABufSPI2,B
        MOV A,*_BufSPI2[B]         ; GUARDA DATO
        INC _ABufSPI2
        CMP #029h,_ABufSPI2
        JEQ RXFINMAS1
        MOV #C_DUMMY,SPIDAT        ; TRANSMITE DATO A SLAVE PARA RX SIG CARACTER
        JMP SALIRSPI
RXFINMAS:  CMP #C_FMSG,A              ; CHECA SI ES FIN DE CADENA
        JNE RXDATMAS
        MOV _ABufSPI2,B
        MOV A,*_BufSPI2[B]         ; GUARDA DATO
RXFINMAS1:  AND #0FEh,_Fcps21          ; fESPINFMASTER=0
        OR #02h,_Fcps2             ; fBufSPI2=ON
        JMP SALIRSPI

EEPROM.ASM
;*****************************************************************************
;** RUTINA DE LECTURA DE UN BYTE DE MEMORIA EEPROM                          **
;*****************************************************************************
FP         .equ R31             ; DEFINE FRAME POINTER
STK        .equ R33              ; DEFINE SOFTWARE STACK

          .global _eRead        ; DECLARA FUNCION EXTERNA
          .global $epilog       ; DECLARA FUNCION DE RESPALDO DE ENTRADA
          .global $prolog       ; DECLARA FUNCION DE RESPALDO DE SALIDA
          .reg _Veeprom2,1
          .globreg _Veeprom2
          .reg _Veeprom3,1
          .globreg _Veeprom3
          .reg _Veeprom4,1
          .globreg _Veeprom4
          .reg _Veeprom5,1
          .globreg _Veeprom5
_eRead:      CALL $prolog          ; LLAMA A RUTINA DE RESPALDO DE "OLDFP"
          PUSH A               ; RESPALDA REG AUXILIARES
leRead:     ; BTJO #080h,P01A,leRead ; CHECA SI MEMORIA EEPROM PUEDE SER ACCESADA
          ; *******quitar ; de arriba en programa final*******
          mov #0ch,A           ; quitar  ***********
l1eread:     djnz A,l1eread        ; quitar **************
          MOV *-3[FP],A         ; CARGA "addr" (DIRECCION DE DATO A LEER)
          MOV A,_Veeprom4     ; LECTURA DE MSB addr
          MOV *-2[FP],A
          MOV A,_Veeprom5     ; LECTURA DE LSB addr
```

```
        MOV *_Veeprom5,A    ; LECTURA DE DATO
        MOV A,R8            ; CARGA DATO EN REGISTRO DE RETORNO
        POP A
        BR $epilog          ; SALE A RESTAURAR "FP"
;*****************************************************************************
;** RUTINA DE ESCRITURA DE UN BYTE DE MEMORIA EEPROM                        **
;*****************************************************************************
        .global _eWrite     ; DECLARA FUNCION EXTERNA
_eWrite:    CALL $prolog        ; LLAMA A RUTINA DE RESPALDO DE "OLDFP"
        PUSH A                                                  ; RESPALDA REG AUXILIARES
        DINT                                                    ; DESHABILITACION GLOBAL DE INTERRUPCIONES
        MOV *-3[FP],A       ; CARGA "addr" (DIRECCION DE DATO A ESC)
        MOV A,_Veeprom4     ; LECTURA DE MSB addr
        MOV *-2[FP],A
        MOV A,_Veeprom5     ; LECTURA DE LSB addr
        MOV *-4[FP],A       ; CARGA "data" DATO A GRABAR
            MOV A,*_Veeprom5    ; ESCRIBE "data" EN "addr"
        MOV #03h,P01A       ; ESCRIBE "1s" W1W0=1, EXE=1
        EINT                                                    ; HABILITACION GLOBAL DE INTERRUPCIONES
        MOVW #0ADAh,_Veeprom3   ; DEFINE RETARDO DE 10 mS-
leWrite:    INCW #-1,_Veeprom3      ; DECREMENTA CONTADOR
        JC leWrite
        MOV #00h,P01A       ; APAGA ESCRITURA EN EEPROM, EXE=0
        DINT                                                    ; DESHABILITACION GLOBAL DE INTERRUPCIONES
                MOV *-3[FP],A       ; CARGA "addr" (DIRECCION DE DATO A ESC)
        MOV A,_Veeprom4         ; LECTURA DE MSB addr
                                    MOV *-2[FP],A
        MOV A,_Veeprom5         ; LECTURA DE LSB addr
        MOV *-4[FP],A       ; CARGA "data" DATO A GRABAR
            MOV A,*_Veeprom5        ; ESCRIBE "data" EN "addr"
        MOV #01h,P01A       ; ESCRIBE "0s" W1W0=0, EXE=1
        EINT                                                    ; HABILITACION GLOBAL DE INTERRUPCIONES
        MOVW #0ADAh,_Veeprom3   ; DEFINE RETARDO DE 10 mS
l1eWrite:   INCW #-1,_Veeprom3      ; DECREMENTA CONTADOR
        JC l1eWrite
        MOV #00h,P01A           ; APAGA ESCRITURA EN EEPROM, EXE=0
        MOV #01h,R8             ; CARGA VALOR DE RETORNO "TRUE"
        POP A
        BR $epilog          ; SALE A RESTAURAR "FP"
```

MEMFLASH.ASM

```
;****************************************
;*****   DECLARACION DE VARIABLES   *****
;****************************************
        .global _ChipErase   ; DEFINICION DE VARIABLE GLOBAL
        .global _CleanSector ; DEFINICION DE VARIABLE GLOBAL
        .global _frByte     ; DEFINICION DE VARIABLE GLOBAL
        .global _fwByte     ; DEFINICION DE VARIABLE GLOBAL
        .global $prolog     ; DEFINICION DE VARIABLE GLOBAL
        .global $epilog     ; DEFINICION DE VARIABLE GLOBAL
FP      .equ   R31          ; DEFINICION DE REGISTRO
STK     .equ   R33          ; DEFINICION DE REGISTRO
DDATA   .equ   P02E         ; DEFINICION DE REGISTRO
        .reg _Vflash8,1     ; DEFINICION DE NUMERO DE BYTE A UTILIZAR
        .globreg _Vflash8   ; DEFINICION DE REGISTRO GLOBAL
        .reg _Vflash9,1     ; DEFINICION DE NUMERO DE BYTE A UTILIZAR
        .globreg _Vflash9   ; DEFINICION DE REGISTRO GLOBAL
        .reg _Vflash0,1     ; DEFINICION DE NUMERO DE BYTE A UTILIZAR
        .globreg _Vflash0   ; DEFINICION DE REGISTRO GLOBAL
        .reg _Vflash1,1     ; DEFINICION DE NUMERO DE BYTE A UTILIZAR
        .globreg _Vflash1   ; DEFINICION DE REGISTRO GLOBAL
        .globreg _Veeprom2  ; DEFINICION DE REGISTRO GLOBAL
        .globreg _Veeprom3  ; DEFINICION DE REGISTRO GLOBAL
        .globreg _Veeprom4  ; DEFINICION DE REGISTRO GLOBAL
        .globreg _Veeprom5  ; DEFINICION DE REGISTRO GLOBAL



        .bss _Pagina,2      ; DEFINICION DE NUM. BYTE A UTILIZAR
        .global _Pagina     ; DEFINICION DE VARIABLE GLOBAL

        .bss _Offset,1      ; DEFINICION DE NUM. BYTE A UTILIZAR
```

```
        .global _Offset     ; DEFINICION DE VARIABLE GLOBAL

;*******************************************************
;*****   RUTINA DE BORRAR BLOQUE DE MEMORIA FLASH  *****
;*******************************************************


_ChipErase: CALL $prolog   ; LLAMA A RUTINA DE RESPALDO DE "OLDEP"
        PUSH A          ; ALMACENAR EN STACK EL REGISTRO A
        MOVW #0D555h,_Veeprom5   ; GRABFL1MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0EAAAh,_Veeprom3   ; GRABFL2MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0C000h,_Vflash1    ; DEFINICION DE DIR. MEMORIA
        MOV *-2[FP],A    ; LEER DATO DE BLOQUE A BORRAR
        RL A            ; ROTACION A LA IZQUIERDA DEL REGISTRO A
        RL A
        CMP #010h,A
        JL  LDSC
        ADD #030h,A
LDSC:       MOV A,_Vflash9   ; ALMACENAR EL REGISTRO A
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        OR #001h,DDATA   ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%%%
        MOV #0AAh,A      ; ALMACENAR EN REG. A LOS DATOS 'AA'
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1MS
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        MOV #055h,A      ; ALMACENAR EN REG. A LOS DATOS '55'
        MOV A,*_Veeprom3 ; GRABFL2MS ; ALMACENAR REG. A EN FRABFL2MS
        OR #001h,DDATA   ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%%%
        MOV #080h,A      ; ALMACENAR EN REG. A LOS DATOS '80'
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1S
        MOV #0AAh,A      ; ALMACENAR EN REG. A LOS DATOS 'AA'
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1S
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        MOV #055h,A      ; ALMACENAR EN REG. A LOS DATOS '55'
        MOV A,*_Veeprom3 ; GRABFL2MS ; ALMACENAR REG. A EN GRABFL1S
        MOV #010h,A      ; ALMACENAR EN REG. A LOS DATOS '10'
        MOV A,*_Veeprom5  ; ALMACENAR REG. A EN TYNI
        MOVW #0129h,_Veeprom3  ; CICLO DE 70 SEGUNDOS (236 ms X 5)
LET4C:      MOVW #0FFFFh,_Veeprom5 ; CICLO DE 236 mSEGUNDOS
LET3C:      INCW #-1,_Veeprom5     ; DECREMENTAR EL CONTADOR EN UNA UNIDAD
        JC LET3C             ; SI EL CARRY ES 1 SALTA A LET3
        INCW #-1,_Veeprom3     ; DECREMENTA EL CONTADOR
        JC LET4C
        POP A           ; OBTENER DEL STACK EL VALOR DEL REGISTRO A
        BR $epilog      ; SALE A RESTAURAR "FP"

;*******************************************************
;*****   RUTINA DE BORRAR BLOQUE DE MEMORIA FLASH  *****
;*******************************************************


_CleanSector: CALL $prolog   ; LLAMA A RUTINA DE RESPALDO DE "OLDEP"
        PUSH A          ; ALMACENAR EN STACK EL REGISTRO A
        MOVW #0D555h,_Veeprom5   ; GRABFL1MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0EAAAh,_Veeprom3   ; GRABFL2MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0C000h,_Vflash1    ; DEFINICION DE DIR. MEMORIA
        MOV *-2[FP],A    ; LEER DATO DE BLOQUE A BORRAR
        RL A            ; ROTACION A LA IZQUIERDA DEL REGISTRO A
        RL A
        CMP #010h,A
        JL  LDS
        ADD #030h,A
LDS:            MOV A,_Vflash9   ; ALMACENAR EL REGISTRO A
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        OR #001h,DDATA   ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%%%
        MOV #0AAh,A      ; ALMACENAR EN REG. A LOS DATOS 'AA'
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1MS
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        MOV #055h,A      ; ALMACENAR EN REG. A LOS DATOS '55'
        MOV A,*_Veeprom3 ; GRABFL2MS ; ALMACENAR REG. A EN FRABFL2MS
        OR #001h,DDATA   ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%%%
        MOV #080h,A      ; ALMACENAR EN REG. A LOS DATOS '80'
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1S
        MOV #0AAh,A      ; ALMACENAR EN REG. A LOS DATOS 'AA'
```

```
        MOV A,*_Veeprom5 ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1S
        AND #0B0h,DDATA  ; INICIALIZAR BLOQUE DE MEM. FLASH
        MOV #055h,A      ; ALMACENAR EN REG. A LOS DATOS '55'
        MOV A,*_Veeprom3 ; GRABFL2MS ; ALMACENAR REG. A EN GRABFL1S
        MOV _Vflash9,A   ; LEER DATO DE BLOQUE A BORRAR
        OR A,DDATA       ; HABILITAR BLOQUE A BORRAR
        MOV #030h,A      ; ALMACENAR EN REG. A LOS DATOS '30'
        MOV A,*_Vflash1  ; ALMACENAR REG. A EN TYNI
        MOV #023h,_Veeprom3 ; CICLO DE 8.26 SEGUNDOS (236 ms X 35)
LET4:   MOVW #0FFFFh,_Veeprom5 ; CICLO DE 236 mSEGUNDOS
LET3:   INCW #-1,_Veeprom5 ; DECREMENTAR EL CONTADOR EN UNA UNIDAD
        JC LET3          ; SI EL CARRY ES 1 SALTA A LET3
        DJNZ _Veeprom3,LET4 ; DECREMENTA EL CONTADOR
        POP A            ; OBTENER DEL STACK EL VALOR DEL REGISTRO A
        BR $epilog       ; SALE A RESTAURAR "FP"


;*****************************************************************
;*****   RUTINA DE ESCRITURA DE UN BYTE EN MEMORIA FLASH   *****
;*****************************************************************


_fwByte:  CALL $prolog     ; LLAMA A RUTINA DE RESPALDO DE "OLDEP"
        PUSH A           ; ALMACENAR EN STACK EL REGISTRO A
        PUSH B           ; ALMACENAR EN STACK EL REGISTRO B

        MOVW #0D555h,_Veeprom5  ; GRABFL1MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0EAAAh,_Veeprom3  ; GRABFL2MS ;DEFINICION DE DIR. DE MEM. FLASH

; LEE PAGINA Y LA MULTIPLICA POR 64 PARA APUNTAR AL PRIMER BYTE DE LA PAGINA

         MOV *-2[FP],A    ; OBTENER EL VALOR MSBYTE
         MOV A,_Vflash1   ; ALMACENAR EN REGIRTO          0
         MOV *-3[FP],A    ; OBTENER EL VALOR LSBYTE
         MOV A,_Vflash0   ; ALMACENAR EN REGISTRO         1
         CLR _Vflash9     ; LIMPIAR REGISTRO
         MOV #05h,_Vflash8 ; ALMACENAR EN CONTADOR EL VALOR 05
         RLC _Vflash1     ; ROTACION A LA IZQUIERDA CON CARRY
         RLC _Vflash0     ; ROTACION A LA IZQUIERDA CON CARRY
LAP:     RLC _Vflash1     ; ROTACION A LA IZQUIERDA CON CARRY
         RLC _Vflash0     ; ROTACION A LA IZQUIERDA CON CARRY
         DAC _Vflash9,_Vflash9  ; SUMAR REGISTRO CON CARRY
         DJNZ _Vflash8,LAP ; DECREMENTA CONTADOR, SALTA SI ES CERO

; SUMA OFSET A DIRECCION

ALA:     MOV *-4[FP],A    ; ALMACENAR EN REGISTRO A EL OFFSET
        MOV A,B          ; ALMACENAR EN REGISTRO B
        CLR A            ; LIMPIAR REGISTRO A
        ADD _Vflash1,B   ; SUMAR OFFSET Y DIRECCION
        ADC _Vflash0,A   ; SUMAR CARRY A DIRECCION
        ADC #00h,_Vflash9 ;
        MOV B,_Vflash1   ; ALMACENAR REGISTRO B EN DIRECCION
        MOV A,_Vflash0   ; ALMACENAR REGISTRO A EN DIRECCION
; ACOMODO DE MSB EN REGISTRO PUERTO D _Vflash8=PD= XA18XXA17A16A15A14

ECIE:    CLR A            ; LIMPIA REGISTRO PARA FORMAR NUMERO
        MOV _Vflash0,B   ; LEER BYTE INTERMEDIO DIREC. TABLA MEM. FLASH
        SWAP B           ; 4 ROTACIONES HACIA LA IZQUIERDA DE REG. A
        AND #0Ch,B       ; AISLA A14 Y A15
        RR B             ; PONE A14 Y A15 EN LOS BITS MENOS SIGNIFICATIVOS
        RR B
        OR B,A           ; GUARDA LOS 2 LSb EN REGISTRO
        MOV _Vflash9,B   ; LEE LOS 3 MSb
        RL B             ; ACOMODA LOS BITS EN POSICIONES CORRESPONDIENTES
        RL B
        MOV B,_Vflash8   ; RESPALDA REGISTRO ROTADO
        AND #0Ch,B       ; AISLA BITS
        OR B,A           ; AGREGA BITS EN REGISTRO
        RL _Vflash8      ; ACOMODA MSb
        RL _Vflash8
        AND #040h,_Vflash8 ; ASILA MSb
        OR _Vflash8,A    ; AGREGA MSb A REGISTRO
```

```
        MOV A,_Vflash8    ; GUARDA DATO

; PONE LA DIRECCION A LEER EN EL RANGO DE C000 PARA PODER MANEJAT LA FLASH

DRI:    AND #03Fh,_Vflash0
        OR #0C0h,_Vflash0

; ESCRITURA DE DATO

        AND #0B0h,DDATA   ; INICIALIZAR BLOQUE DE MEM.FLASH
        OR #001h,DDATA    ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%
        MOV #0AAh,A       ; ALMACENAR EN REG. A LOS DATOS 'AA'
        MOV A,*_Veeprom5  ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1MS
        AND #0B0h,DDATA   ; INICIALIZAR BLOQUE DE MEM.FLASH
        MOV #055h,A       ; ALMACENAR EN REG. A EL DATO '55'
        MOV A,*_Veeprom3  ; GRABFL2MS ; ALMACENAR REG. A EN GRABFL2MS
        OR #001h,DDATA    ; BLOQUE 1 DE MEM.FLASH   %%%%%%%%%%%%%%%%%%
        MOV #0A0h,A       ; ALMACENAR EN REG. A LOS DATOS 'A0'
        MOV A,*_Veeprom5  ; GRABFL1MS ; ALMACENAR REG. A EN GRABFL1MS
        AND #0B0h,DDATA   ; INICIALIZAR BLOQUE DE MEM.FLASH
        MOV _Vflash8,A    ; REGISTRO A = BRICK
        OR A,DDATA        ; HABILITAR BLOQUE DE MEM. FLASH
        MOV *-5[FP],A     ; REGISTRO A = GU6 'DATO 6'
        MOV A,*_Vflash1   ; ALMACENAR DATO DEL  REG. A EN MEM. FLASH

        POP B           ; OBTENER DEL STACK EL VALOR DEL REGISTRO B
        POP A           ; OBTENER DEL STACK EL VALOR DEL REGISTRO A
        BR $epilog       ; SALE A RESTAURAR "FP"


;************************************************************
;*****  RUTINA DE LECTURA DE UN BYTE EN MEMORIA FLASH  *****
;************************************************************


_frByte:  CALL $prolog     ; LLAMA A RUTINA DE RESPALDO DE "OLDEP"
        PUSH A          ; ALMACENAR EN STACK EL REGISTRO A
        PUSH B          ; ALMACENAR EN STACK EL REGISTRO B


        MOVW #0D555h,_Veeprom5  ; GRABFL1MS ;DEFINICION DE DIR. DE MEM. FLASH
        MOVW #0EAAAh,_Veeprom3  ; GRABFL2MS ;DEFINICION DE DIR. DE MEM. FLASH



         MOV *-2[FP],A    ; OBTENER EL VALOR MSBYTE      &_Pagina,A
         MOV A,_Vflash1   ; ALMACENAR EN REGIRTO         0
         MOV *-3[FP],A    ; OBTENER EL VALOR LSBYTE      &_Pagina+1
         MOV A,_Vflash0   ; ALMACENAR EN REGISTRO        1
         CLR _Vflash9     ; LIMPIAR REGISTRO
         MOV #05h,_Vflash8 ; ALMACENAR EN CONTADOR EL VALOR 05
         RLC _Vflash1     ; ROTACION A LA IZQUIERDA CON CARRY
         RLC _Vflash0     ; ROTACION A LA IZQUIERDA CON CARRY
LAP1:    RLC _Vflash1     ; ROTACION A LA IZQUIERDA CON CARRY
        RLC _Vflash0     ; ROTACION A LA IZQUIERDA CON CARRY
        DAC _Vflash9,_Vflash9  ; SUMAR REGISTRO CON CARRY
        DJNZ _Vflash8,LAP1 ; DECREMENTA CONTADOR, SALTA SI ES CERO

; SUMA OFSET A DIRECCION

ALA1:    MOV *-4[FP],A    ; ALMACENAR EN REGISTRO A EL OFFSET
        MOV A,B         ; ALMACENAR EN REGISTRO B
        CLR A           ; LIMPIAR REGISTRO A
        ADD _Vflash1,B   ; SUMAR OFFSET Y DIRECCION
        ADC _Vflash0,A   ; SUMAR CARRY A DIRECCION
        ADC #00h,_Vflash9 ;
        MOV B,_Vflash1   ; ALMACENAR REGISTRO B EN DIRECCION
        MOV A,_Vflash0   ; ALMACENAR REGISTRO A EN DIRECCION
; ACOMODO DE MSB EN REGISTRO PUERTO D _Vflash8=PD= XA18XXA17A16A15A14

ECIE1:   CLR A           ; LIMPIA REGISTRO PARA FORMAR NUMERO
        MOV _Vflash0,B   ; LEER BYTE INTERMEDIO DIREC. TABLA MEM. FLASH
        SWAP B          ; 4 ROTACIONES HACIA LA IZQUIERDA DE REG. A
```

```
        AND #0Ch,B       ; AISLA A14 Y A15
        RR B             ; PONE A14 Y A15 EN LOS BITS MENOS SIGNIFICATIVOS
        RR B
        OR B,A           ; GUARDA LOS 2 LSb EN REGISTRO
        MOV _Vflash9,B   ; LEE LOS 3 MSb
        RL B             ; ACOMODA LOS BITS EN POSICIONES CORRESPONDIENTES
        RL B
        MOV B,_Vflash8   ; RESPALDA REGISTRO ROTADO
        AND #0Ch,B       ; AISLA BITS
        OR B,A           ; AGREGA BITS EN REGISTRO
        RL _Vflash8      ; ACOMODA MSb
        RL _Vflash8
        AND #040h,_Vflash8 ; ASILA MSb
        OR _Vflash8,A    ; AGREGA MSb A REGISTRO
        MOV A,_Vflash8   ; GUARDA DATO

; PONE LA DIRECCION A LEER EN EL RANGO DE C000 PARA PODER MANEJAT LA FLASH

DRI1:     AND #03Fh,_Vflash0
        OR #0C0h,_Vflash0

; LECTURA DE DATO

        AND #07Fh,DDATA   ; HABILITACION DE SALIDA DE MEMORIA FLASH
        AND #0B0h,DDATA   ; INICIALIZAR BLOQUE DE MEM. FLASH
        MOV _Vflash8,A    ; REGISTRO A = BRICK
        OR A,DDATA        ; HABILITAR BLOQUE DE MEM. FLASH
        MOV *_Vflash1,A   ; LEER BYTE DE FLASH PARA ALMACENAR EN MEM. RAM
        MOV A,R8          ; ALMACENAR REGISTRO A EN DATO 6
        OR #080h,DDATA    ; DESHABILITACION DE SALIDA DE MEMORIA FLASH
        POP B             ; OBTENER DEL STACK EL VALOR DEL REGISTRO B
        POP A             ; OBTENER DEL STACK EL VALOR DEL REGISTRO A
        BR $epilog        ; SALE A RESTAURAR "FP"

SCIINTS.ASM

*******************************************************************************
;***** RUTINAS DE ATENCION A LA INTERRUPCION DE SCI TX Y RX            *****
;*******************************************************************************
;*******************************************************************************
; declaración de registros y perifericos
;*******************************************************************************
T1CTL1     .equ P049  ; REGISTRO 1 DE CONTROL DE TIMER 1
T1CTL2     .equ P04A  ; REGISTRO 2 DE CONTROL DE TIMER 1
T1CTL3     .equ P04B  ; REGISTRO 3 DE CONTROL DE TIMER 1
T1PC1      .equ P04D  ; PIN T1EVT EMPLEADO PARA GIRAR TORNIQUETE
T1CMSB     .equ P042  ; REGISTRO COMPARADOR DE TIMER 1 MSB
T1CLSB     .equ P043  ; REGISTRO COMPARADOR DE TIMER 1 LSB
SCICCR     .equ P050  ; REG. CONT. SCI (#BITs, PARIDAD, SINC., BITs PARADA)
SCICTL     .equ P051  ; REG. CONT. SCI (BIT SLEEP, HABILIT. TX-RX,RESET, ETC)
BAUDMSB    .equ P052  ; REG. CONT. SCI (MSB DE VELOCIDAD)
BAUDLSB    .equ P053  ; REG. CONT. SCI (LSB DE VELOCIDAD)
TXCTL      .equ P054  ; REG. CONT. SCI (CONTROL Y EDO. DE TX)
RXCTL      .equ P055  ; REG. CONT. SCI (CONTROL Y EDO. DE RX)
RXBUF      .equ P057  ; REG. CONT. SCI (BUFFER DE RECEPCION DE DATO)
TXBUF      .equ P059  ; REG. CONT. SCI (BUFFER DE TRANSMISION DE DATO)
SCIPC1     .equ P05D  ; REG. CONT. SCI (CONTROL DE RELOJ)
SCIPC2     .equ P05E  ; REG. CONT. SCI (DEF. DE PINES RX Y TX)
SCIPRI     .equ P05F  ; REG. CONT. SCI (DEF. DE PRIORIDAD SCI)
C_IMSG     .equ 254   ; VALOR DE COMANDO DE INICIO DE MENSAJE
C_EINF     .equ 253   ; VALOR DE COMANDO DE ENVIO DE LECTURAS
C_FMSG     .equ 252   ; VALOR DE COMANDO DE FIN DE MENSAJE
C_CLT      .equ 251   ; VALOR DE COMANDO DE CONTADOR DE LECTURAS
C_LECT     .equ 250   ; VALOR DE COMANDO DE LECTURA
C_BLT      .equ 249   ; VALOR DE COMANDO DE BORRADO DE LECTURAS ALMACENADAS
C_DRTC     .equ 248   ; VALOR DE COMANDO DE CONFIGURACION DE RELOJ DE TIEMPO REAL (RTC)
C_ACG      .equ 247   ; VALOR DE COMANDO DE AVISO DE CARGA DE MEMORIA
C_CMEM     .equ 246   ; VALOR DE COMANDO DE CARGA DE MEMORIA
C_SRTC     .equ 245   ; VALOR DE COMANDO DE SOLICITUD DE RTC DE UNIDAD
C_RSTFLASH .equ 244   ; COMANDO DE SOLICITUD DE RESET FLASH
C_ELINE    .equ 243   ; VALOR DE COMANDO DE SOLICITUD DE ENTRAR A EDO LINEA
C_SLINE    .equ 242   ; VALOR DE COMANDO DE SOLICITUD DE SALIR EDO LINEA
C_PRG      .equ 241   ; COMANDO DE PROGRAMACION DE TRANSPONDER
```

```
C_BUFOK     .equ 240   ; COMANDO DE SOLICITAR INF DE uC LECTOR
C_NOBUF     .equ 239   ; COMANDO DE AVISAR A uC INT QUE NO HAY INF
C_ERRORBUF  .equ 238   ; VALOR DE COMANDO DE QUE SE ENV A MASTER P/AVISAR QUE BUF SE ENVIO MAL
C_DUMMY     .equ 237   ; VALOR DE COMANDO DE SEPARACION DE CAMPOS
C_ELECT     .equ 236   ; VALOR DE COMANDO DE AVISO A uC LECT QUE LECT NO EJECUTO COMANDO
C_ONT       .equ 235   ; VALOR DE COMANDO DE AVISO DE ENCENDIDO DE TORNIQUETE
C_ELINEL    .equ 234   ; VALOR DE COMANDO DE SOLICITUD DE ENTRAR A EDO LINEA MODO LINE
C_RXOK      .equ 233   ; VALOR DE COMANDO DE CONFIRMACION DE RX CORRECTA DE INF
C_RXRPT     .equ 232   ; VALOR DE COMANDO DE AVISO DE REPETRIR ENVIO DE ULTMO REGISTRO
C_ACTL      .equ 231   ; VALOR DE COMANDO DE ACTIVACION DE REGISTRO DE LECTURAS
C_AVZ       .equ 230   ; VALOR DE COMANDO DE AVANCE DE COLA EN MODO 1 DE LECT
XON         .equ 17    ; VALOR DE INDICA A PC QUE ENVIE MAS DATOS
XOFF        .equ 19    ; VALOR DE INDICA A PC QUE NO ENVIE MAS DATOS
CR          .equ 13    ; VALOR DE COMANDO CARRIAGE RETURN <CR>
LF          .equ 10    ; VALOR DE COMANDO LINE FEED <LF>
ESC         .equ 27    ; VALOR DE COMANDO ESCAPE <ESC>
;
;*******************************************************************************
; RUTINA DE TRANSMISION DE INFORMACION A IPC
;*******************************************************************************
            .global TXINFPC
TXINFPC:    BTJO #04h,_FcSCI,TXINFPC1    ; FPERMISOTX=ON?
            RTI
TXINFPC1:   BTJO #02h,_FcSCI,TXINFPC2    ; FBUFTX=ON?
            MOV #00h,TXCTL               ; INTTX=OFF
            RTI
TXINFPC2:   PUSH A                       ; ALMACENA EN STACK REG A Y B
            PUSH B
            MPY B,A                  ; RUTINA PARA HACER TIEMPO
            MOV _AdTx,B              ; LEE APUNTADOR
            MOV *_BufTx[B],A
TXDAT4:     MOV A,TXBUF              ; CARGA DATO EN BUFFER A TX (SCI)
            INC _AdTx                ; INC APUNTADOR DE SIG DAT A TX
            CMP #C_FMSG,A            ; CHECA SI ES FIN DE MENSAJE A PC
            JNE SALINTTX             ; SI ES IGUAL TERMINA TRANSMISION
            MOV #00h,TXCTL               ; INTTX=OFF
            AND #0FDh,_FcSCI             ; FBUFTX=OFF
SALINTTX:   POP B                        ; REGRESA VALORES DE REG A Y B
            POP A
            RTI                      ; RETORNO DE INTERRUPCION DE TX (SCI)
;
;*******************************************************************************
; RUTINA DE RECEPCION DE INFORMACION DE IPC
;*******************************************************************************
            .global RXINFPC
RXINFPC:    BTJZ #080h,RXCTL,NOERRORRX   ; CHECA SI EL DATO LLEGO BIEN
            OR #08h,_FcSCI               ; FERRORRX=ON
NOERRORRX:  MOV RXBUF,_DATORX            ; LEE EL BYTE RX DEL BUFFER (DATO)
            CMP #C_RXOK,_DATORX          ; CHECA SI LLEGO COMANDO RXOK
            JNE RXRPT
            OR #020h,_FcSCI              ; FRXOK=ON
            RTI
RXRPT:      CMP #C_RXRPT,_DATORX         ; CHECA SI LLEGO COMANDO RXRPT
            JNE RXDATA1
            OR #040h,_FcSCI              ; FRXRPT=ON
            RTI
RXDATA1:    BTJZ #01h,_FcSCI,BUFVACIO    ; FBUFRX=OFF?
            RTI
BUFVACIO:   CMP #C_FMSG,_DATORX          ; CHECA FIN DE CADENA PC
            JNE RXLF
            MOV #XOFF,TXBUF              ; CARGA DATO XOFF EN BUFFER A TX (SCI)
            JMP SBUFFULL
RXLF:       CMP #LF,_DATORX              ; CHECA FIN DE CADENA LECTOR
            JNE RXXOFF                   ; SI NO ES PRUEBA OTRA OPCION
SBUFFULL:   OR #01h,_FcSCI               ; FBUFRX=ON
            JMP RXDAT                    ; SALE DE INTERRUPCION
RXXOFF:     CMP #XOFF,_DATORX            ; CHECA SI LLEGO COMANDO XOFF
            JNE RXXON
            AND #0FBh,_FcSCI             ; FPERMISOTX=OFF
            RTI
RXXON:      CMP #XON,_DATORX             ; CHECA SI LLEGO COMANDO XON
            JNE RXDAT
            OR #04h,_FcSCI               ; FPERMISOTX=ON
```

```
        RTI
RXDAT:     CMP #049h,_AdRx
        JGE SALINTRX2
        PUSH A
        PUSH B
        MOV _DATORX,A              ; CARGA DATO EN REG. AUXILIAR
        MOV _AdRx,B
        MOV A,*_BufRx[B]           ; ALMACENA DATO EN BUFFER
        INC _AdRx                  ; APUNTA A LA SIG LOCALIDAD DE MEM.
SALRXCMD1:  POP B                  ; RETORNA VALORES RESP. A REG A Y B
        POP A
SALINTRX2:  RTI                    ; TERMINA LA INTERRUPCION


;*******************************************************************************
; rutina de atencion a interrupcion
;*******************************************************************************
                                        .global TBUFTX
TBUFTX:    CMP _ContmSeg,_TBufTx
        JNE SALBUFTX
        AND #0FBh,T1PC1       ; TERMINA PULSO PARA GIRAR/TORNIQUETE ON/ALARMA
        OR #01h,T1CTL2        ; RESETEA CONTADOR
        AND #0DEh,T1CTL3      ; LIMPIA INTERRUPCION PENDIENTE
        RTI
SALBUFTX:  INC _ContmSeg
        OR #01h,T1CTL2        ; RESETEA CONTADOR
        AND #0DFh,T1CTL3      ; LIMPIA INTERRUPCION PENDIENTE
                RTI
```

BIBLIOGRAFÍA

BIBLIOGRAFÍA

LIBROS

- A. Sedra /K. C. Smith, *dispositivos electrónicos y amplificación de señales*, McGraw-Hill, México 1989.

- Brian W. Kernighan/Dennis M. Ritchie, *El lenguaje de programación C*, segunda edición, Printence Hall.

- Douglas V. Hall, *Microprocessors and interfacing. Programming and hardware*, Editorial McGraw-Hill, USA, 1991.

- Herbert Taub, *Circuitos digitales y microprocesadores*, McGraw-Hill, México, 1990.

- M. Morris Mano, *Diseño digital*, 1ª edición, Printence Hall, México, 1987.

- Robert F. Coughlin/Frederick F. Driscoll, *Circuitos integrados lineales y amplificadores operacionales*, 2ª edición, Printence Hall, México, 1987.

- Andrew S. Tanembaum, *Structured computer organization*, third edition, Printence Hall, USA, 1990.

MANUALES

- Data Manual, 8-bit microcontroller family, TMS370 Family, Texas Instruments, 1993.

- TIRIS Reference Guide, Series 2000 Reader System RI-STU-251B, Texas Instruments, 2000.

- TIRIS Reference Guide, 32 mm Glass transponder, Texas Instruments, 2001.

- TIRIS Reference Guide, Series 2000 Reader System ASCII protocol, Texas Instruments, 2000.