UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS

DISEÑO Y DESARROLLO DE UN CONCENTRADOR DE DATOS PARA LA EVALUACIÓN ENERGÉTICA EN EDIFICIOS

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO ELÉCTRICO ELECTRÓNICO

PRESENTA:

SERGIO AXEL SOTO BATALLA

DIRECTOR DE TESIS:

M.I. JOSÉ CASTILLO HERNÁNDEZ

CIUDAD UNIVERSITARIA, MÉXICO D.F. 26 / SEPTIEMBRE / 2015

A mis padres Daniel y Sofia, por siempre haber hecho su mejor esfuerzo y forjar el hombre que ahora soy, por su apoyo y su amor, les dedico este logro y todos los futuros, porque mis sueños son sus sueños.

Ítaca

Cuando emprendas el viaje hacia Ítaca,
ruega que tu camino sea largo
y rico en aventuras y descubrimientos.
No temas a lestrigones, a cíclopes o al fiero Poseidón;
no los encontrarás en tu camino
si mantienes en alto tu ideal,
si tu cuerpo y alma se conservan puros.
Nunca verás los lestrigones, los cíclopes o a Poseidón,
si de ti no provienen,
si tu alma no los invoca.

Ruega que tu camino sea largo,
que sean muchas las mañanas de verano,
cuando, con placer, llegues a puertos
que descubras por primera vez.
Ancla en mercados fenicios y compra cosas bellas:
madreperla, coral, ámbar, ébano
y voluptuosos perfumes de todas clases.
Compra todos los aromas sensuales que puedas;
ve a las ciudades egipcias y aprende de los sabios.

Siempre ten a Ítaca en tu mente;
llegar allí es tu meta; pero no apresures el viaje.
Es mejor que dure mucho,
mejor anclar cuando estés viejo.
Pleno con la experiencia del viaje
no esperes la riqueza de Ítaca.
Ítaca te ha dado un bello viaje.
Sin ella nunca lo hubieras emprendido;
pero no tiene más que ofrecerte,
y si la encuentras pobre, Ítaca no te defraudó.

Con la sabiduría ganada, con tanta experiencia,
habrás comprendido lo que las Ítacas significan.

Constantinos P. Caafis (1863-1933)

Agradecimientos

A mi viejita Carmen, por sus cuidados y su cariño, que aunque ya no se encuentra con nosotros físicamente, siempre estará en mi corazón.

A mi viejito Toyo, por ser un ejemplo de fuerza, sabiduría y de lo que es ser un hombre de bien.

A mi hermana Ixchel, por los momentos que hemos compartido a lo largo de nuestra vida.

A mis tíos Areli, Saúl, Chano y Aquilino por su amistad y su apoyo, a mis primos: Yuli, Deni, Emiliano, Valentina, Israel y Nancy por ser como mis segundos hermanos y hermanas.

A Mariana por su gran apoyo, ayuda e impulso para alcanzar el fin de este trabajo.

Al maestro José Castillo Hernández por darme la oportunidad de ser parte de este proyecto y confiar en mi talento para hacerlo realidad, por su apoyo y sus valiosas enseñanzas.

Al maestro Guillermo Sovero Ancheta mi compañero y amigo por sus enseñanzas y su amistad.

A los maestros del CCADET, el Dr. Nicolás Kemper Valverde y Sergio Quintana Thierry, por el apoyo y los consejos de vida brindados.

A mis amigos de la Universidad, en especial a Raúl Zepeda y a Diana Ramírez por todos los momentos y proyectos que emprendimos juntos a lo largo de nuestra carrera.

A mis maestros y a todas aquellas personas que estuvieron involucradas en el desarrollo de este trabajo y en mi formación profesional que por alguna u otra razón no menciono pero que igualmente agradezco y recuerdo con cariño.

A la Facultad de Ingeniería de la Universidad Nacional Autónoma de México por ser mi casa de estudios y al laboratorio de electrónica Del Centro de Ciencias Aplicadas y Desarrollo Tecnológico, por permitirme el uso de sus equipos e instalaciones llevar a cabo este trabajo.

Al gobierno del Distrito Federal, por otorgarme la beca y el apoyo para realizar este trabajo.

INDICE

INDICE DE FIGURAS

INDICE DE TABLAS

Introducción

"La energía que menos contamina es la que no se consume"

Arturo Romero Salvador
Catedrático de la Universidad Computense de Madrid

La energía es un factor de gran relevancia en el desarrollo económico de cualquier país, esta estrecha relación nace con la denominada revolución industrial en el siglo XVIII y se mantiene hasta nuestros días, prueba de ello ha sido el intenso crecimiento de la demanda energética, especialmente en las décadas posteriores a la segunda guerra mundial. La humanidad consume en la actualidad cincuenta veces más energía que hace un siglo y este aumento de las necesidades energéticas plantea hoy nuevos problemas. Entre ellos se pueden mencionar las limitadas reservas de los combustibles fósiles (carbón, petróleo y gas natural), recursos no renovables que representan el 80% de la producción mundial de energía primaria; así también en el mismo contexto la producción y el consumo de energía plantean grandes problemas de conservación del medio ambiente que dan lugar al cambio climático y por último, un problema reciente es la capacidad para aumentar la oferta de energía; sin embargo un tema de mayor importancia es el consumo responsable de los recursos energéticos, lo cual engloba un conjunto de actividades encaminadas a reducir el consumo de los mismos, desde la gestión de una política energética sostenible hasta una cultura de concientización de los consumidores.

En la actualidad cada vez es más necesario realizar un consumo correcto de la energía eléctrica, teniendo como principal objetivo la eficiencia en el aprovechamiento de los recursos energéticos y el ahorro económico sumado al trasfondo de lucha contra el cambio climático. De esta forma hay que actuar para mejorar los usos de la energía sin abandonar alguna opción tecnológica disponible (1).

[1] Comisión Nacional para el Uso Eficiente de la Energía (Conuee) (Barcón, 2011)

La gestión para un consumo energético responsable comienza a nivel internacional con el protocolo de Kyoto cuyo objetivo es reducir un 5.2% las emisiones de gases de efecto invernadero globales sobre los niveles de 1990 en países industrializados para el periodo de 2008-2012, posterior a ello en junio de 2011 fue emitida la norma internacional ISO 5001 que especifica los requisitos para establecer, implementar, mantener y mejorar un sistema de gestión de la energía, cuyo propósito es permitir a una organización seguir un enfoque sistemático para lograr la mejora continua de la eficiencia energética. En México, ha adquirido una gran importancia el aprovechamiento de la energía gracias a la publicación de la ley en materia de energía, esto a partir del 28 de noviembre del 2008, fecha de publicación de la ley para el aprovechamiento sustentable de la energía (2).

El control energético de los edificios es fundamental para la racionalización de los consumos, las tarifas eléctricas y los términos de potencia (3). Parte indispensable de nuestro sistema para la gestión de la eficiencia energética es el monitoreo constante del consumo de la red eléctrica por medio de medidores fijos, que obtendrán datos sobre el consumo de potencia en la red eléctrica. A su vez, estos constituyen una red en la cual existe un punto central que se encarga de la recepción de datos, es aquí donde se requiere un concentrador que almacene temporalmente dichos datos, para ser enviados en un conjunto ordenado a una computadora. Con los datos obtenidos es posible realizar un análisis estadístico sobre el comportamiento del consumo energético, para después tomar decisiones en la aplicación de reglas de gestión y ahorro energético y así generar un plan de consumo donde se tomarán en cuenta propuestas de opciones, cuantificación de ahorros y toma de decisiones ya sea correctivas o educativas.

El presente trabajo propone el diseño, desarrollo y construcción de un concentrador inalámbrico de datos referentes al consumo energético en un edificio, dicho instrumento se desarrolló en el Centro de Ciencias Aplicadas y Desarrollo Tecnológico de la Universidad Nacional Autónoma de México.

[2] (2010)
[3] ISO 50001. Gestión de Energía (Rebolledo, 2011)

1.1 Objetivos.

1.1.1 Objetivo general.

Diseñar, desarrollar y construir un dispositivo concentrador inalámbrico de datos que se desempeñe como parte central de una red de supervisión del consumo energético en edificios.

1.1.2 Objetivos particulares.

- Implementar comunicación inalámbrica de fácil manejo y bajo costo.
- Dotar de autonomía al concentrador.
- Capacidad de retención de la información
- Aprovechar al máximo posible las características de los elementos del hardware.
- Programación concisa y eficiente.
- Interfaz de control para el usuario de uso sencillo.

Capítulo 1 Conceptualización del concentrador

1.1 El concentrador en SIGEEN

El sistema para la gestión de la eficiencia energética (SIGEEN), el cual se muestra en la Figura 1.1 se conforma de tres partes: la red de medidores, el concentrador y la computadora. La red inalámbrica se conforma de los medidores de consumo energético los cuales están conectados en diferentes puntos de una estructura o edificio y que se comunican entre sí, formando una red con una topología de tipo multipunto que es estable y escalable, lo que permite que la información sea trasmitida al concentrador desde los nodos más alejados de la red y viceversa. En cuanto a los medidores, estos trabajan con completa autonomía y se encargan de la adquisición y procesamiento primario de información. El sistema de estos dispositivos mide los valores de voltaje y corriente AC en contactos de pared y apagadores de luces, en base a esta información determinan el voltaje y corriente RMS, potencia real y energía consumida, finalmente dicha información se transmite inalámbricamente al concentrador en un paquete de datos.

Por último, el concentrador se define como el dispositivo electrónico que se desempeña como el puente de comunicación entre la red inalámbrica de dispositivos de medición y la computadora. Las necesidades y requisitos del concentrador para su desempeño y trabajo dentro del SIGEEN se señalan en los siguientes puntos:

- Que su señal inalámbrica no interfiera con la de otro tipo de telecomunicaciones.
- La comunicación entre dispositivos no tenga problemas entre muros y pisos a distancias mínimas de entre 20 o 30 metros.
- Soportar como mínimo una red de 1000 medidores.
- Resolución de tiempo para recabar las mediciones configurable de 1 a 15 minutos.
- Capacidad mínima de memoria para retener información hasta por una hora.

- Que la información sea recuperable aún después de cortes inesperados del suministro de energía.
- Independencia de la conexión a la computadora.
- Interfaz de usuario de fácil manejo.

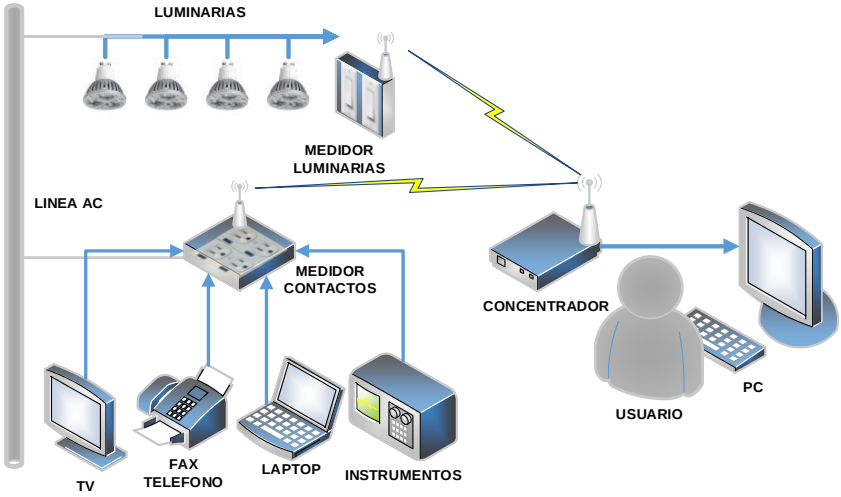

Figura 1.1 Sistema para la gestión de la eficiencia energética.

1.2 Desempeño y labor.

Concretando las funciones del concentrador, estas se basan en dos tipos como se observa en el diagrama de flujo (Figura 1.2), donde el recuadro RECOPILACIÓN agrupa las acciones para recibir la información de los medidores. El recuadro USUARIO ilustra las funciones de configuración que aplican desde la computadora.

La rutina de adquisición y almacenamiento de datos se realiza de manera automática, en intervalos de tiempo definidos por el usuario, en donde el concentrador establece comunicación con uno a uno de los medidores, guardando siempre el orden en la adquisición y evitando posibles colisiones de información. La recopilación de datos comienza cuando el concentrador solicita los valores al medidor en turno, a continuación este responde enviando un paquete con los valores de medición correspondientes, a dicho paquete se le agregan la identificación del medidor remitente (ID) y un apartado para los datos de minuto, hora y día de recepción (tiempo).

Toda la información se almacena en el banco de memoria interna del concentrador donde se guarda para su eventual extracción y envío a la computadora, que se encarga de generar el archivo de reporte final.

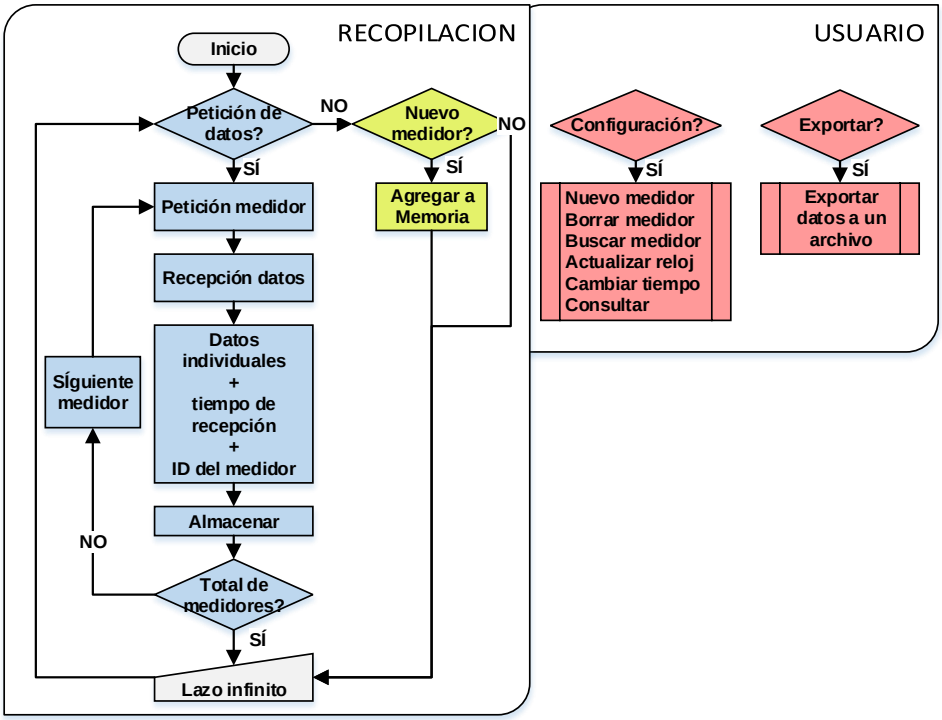

Figura 1.2 Diagrama de flujo del concentrador.

La función para la incorporación de nuevos dispositivos a la red es de forma automática, ya que cada vez que se inicia un medidor este envía su número de ID al concentrador, si el nuevo medidor no está en la lista actual de medidores, se anexa en ese mismo instante como un nuevo elemento de red.

Las funciones de configuración del concentrador se activan cuando se conecta el dispositivo a la computadora y estas se manejan por medio de la interfaz de usuario, dichas configuraciones se clasifican en los siguientes tipos:

- Lectura y actualización del reloj del concentrador.
- Agregar, borrar o buscar un medidor en red.
- Consultar el número de paquetes de mediciones almacenados.
- Cambio del intervalo de tiempo del modo automático.
- Realización de respaldo de información.

La función de respaldo, es la más importante ya que se encarga de la exportación de la información a la computadora con lo que concluye el trabajo del concentrador.

1.3 Arquitectura.

La arquitectura se basó en los requisitos y el diagrama de flujo del concentrador (Figura 1.2). El trabajo del concentrador se lleva a cabo en dos partes como se observa en la Figura 1.3 donde se ilustran los conjuntos de elementos que interactúan para realizar tareas en específico, dichas partes son referidas de aquí en adelante como:

- Sistema A: este sistema trabaja para interactuar con la computadora y realizar las configuraciones de usuario.
- Subsistema B: es la parte del concentrador que se relaciona con la red de medidores y se encarga de recopilar la información deseada.

Ambos sistemas cuentan con un microcontrolador (MCU) propio el cual se encarga de administrar los elementos correspondientes de cada uno de los dos sistemas.

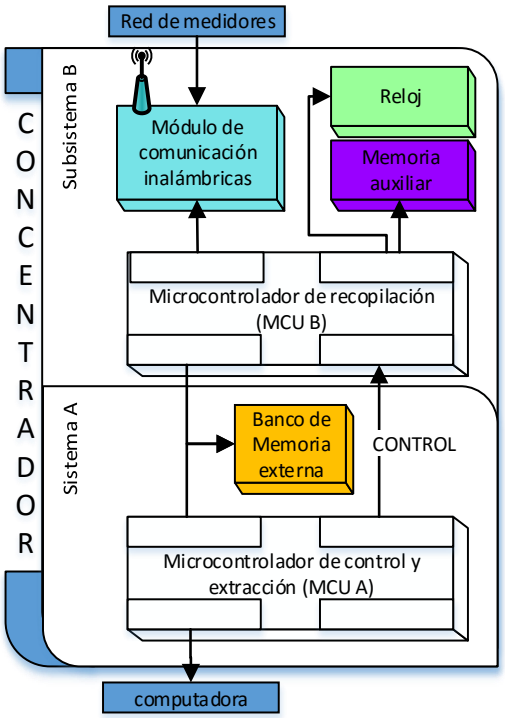

Figura 1.3 Arquitectura.

A continuación se describen las funciones de cada sistema, así como los requisitos de diseño para cada elemento que conforma el concentrador.

1.3.1 Sistema A.

Las tareas del sistema A son: realizar las configuraciones del usuario y exportar las mediciones a la computadora.

El sistema A se encarga de controlar a su semejante el subsistema B y solo interrumpe sus funciones cuando es necesario llevar a cabo una configuración del usuario, cuando el concentrador se encuentra trabajando con normalidad ya sea conectado o desconectado de la computadora, el sistema A solo administra los tiempos de petición del subsistema B. los elementos que conforman este sistema son:

- **El microcontrolador A**: (Figura 1.3 sistema A) es la pieza de control central que se encarga de la comunicación con la computadora y del control de subsistema B, por lo cual es indispensable que maneje puertos de protocolo para comunicaciones entre dispositivos lógicos externo y de conexión a la computadora. El MCU A requiere de gran capacidad de RAM (*Random-Access Memory)*, amplia memoria programable y alta velocidad de procesamiento, esto debido a que los algoritmos de programación son complejos y extensos, por estas características se optó por el uso del PIC18F4550 del fabricante Microchip Technology.
- **Banco de memoria:** (Figura 1.3 sistema A) es el bloque de memoria secundaria no volátil donde se almacena temporalmente la información obtenida de los medidores. La capacidad de almacenamiento de este debe ser suficiente para 60,000 paquetes de mediciones, que es el total de información generada por una red de mil medidores funcionando por una hora a la máxima resolución de tiempo, es decir, 1000 medidores con un intervalo de petición de 1 minuto. El chip adecuado para esta función es el 24FC1025 EEPROM (Electrically Erasable Programmable Read-Only Memory) de la compañía Microchip Technology.

1.3.2 Subsistema B.

Este sistema es el conjunto de componentes encargado de realizar la recopilación de información proveniente de los medidores y a la cual se le debe agregar la ID y el tiempo de recepción antes de guardarla en el banco de memoria. Para realizar esta tarea el sistema se constituye de las siguientes partes:

- **El microcontrolador B**: (Figura 1.3 subsistema B) es el dispositivo principal del sistema de recopilación y es el encargado de manejar los elementos del módulo de radiofrecuencia, el reloj y la memoria auxiliar. Es necesario que cuente con puertos para manejo de protocolos de comunicación con dispositivos lógicos externos. Este microcontrolador al igual que su similar en el sistema A, también requiere de gran capacidad de RAM, amplia memoria programable y alta velocidad de procesamiento, por lo cual se usó el mismo dispositivo PIC18F4550.

- **El módulo de comunicaciones inalámbricas**: (Figura 1.3 subsistema B) es el nodo principal de la red inalámbrica, este dispositivo se encarga de ser el origen y/o destino de información, además debe de cumplir con características específicas como: operar en una banda de radiofrecuencia libre, comunicación estable a distancias mínimas de 30 metros entre muros y que su señal no interfiera con otro tipos de señales de telecomunicaciones, como por ejemplo: Wi-Fi, bluetooth, telefonía celular, etc. El módulo de comunicación inalámbrico XBee del fabricante DIGI International cumple con las características requeridas.

- **El reloj:** (Figura 1.3 subsistema B) es el dispositivo encargado de proporcionar la fecha y hora de recepción de paquetes. Se buscó un circuito integrado externo, autónomo, confiable y de fácil acceso. El reloj de tiempo real DS1307 cubre las necesidades del sistema. Además es un integrado que puede ser utilizado como un oscilador externo ya que cuenta con una señal de salida programable a 1Hz, dicha señal se aprovecha para contabilizar los minutos transcurridos entre los intervalos de recopilación de mediciones.

- **Memoria auxiliar**: (Figura 1.3 subsistema B) es la memoria secundaria que se encarga de retener información importante permanentemente, como por ejemplo, la lista de direcciones de los medidores. El concentrador debe soportar como mínimo una red de 1,000 medidores, lo que exige al menos un tamaño de memoria de 4,000 bytes, por lo cual el chip adecuado para la memoria auxiliar es el 24FC1025 EEPROM, que es el mismo integrado de memoria no volátil usado para el banco de memoria del concentrador.

Capítulo 2 Hardware

2.1 Descripción de componentes lógicos.

De acuerdo a la arquitectura y los requisitos del concentrador, el diseño del hardware del mismo, se conforma de los componentes lógicos que a continuación se enlistan y se describen:

- 2 microcontroladores PIC18F4550.
- 1 reloj de tiempo real DS1307.
- 1 módulo de radiofrecuencia XBee-pro.
- 17 memorias no volátiles EEPROM 24FC1025.

2.1.1 Microcontrolador PIC18F4550.

El PIC18F4550 es un MCU con memoria de programación tipo flash, ideal para aplicaciones lógicas de alto rendimiento y para procesos sensibles a la energía. Cuenta con tecnología nanowatt que reduce significativamente el consumo de energía durante el funcionamiento y permite al usuario incorporar ideas de ahorro de energía por software en el diseño de aplicación. También maneja velocidades de reloj de hasta 48 MHz. Sus puertos programables permiten al usuario desarrollar aplicaciones intensivas de tipo entrada/salida. Además cuenta con diversos módulos para comunicaciones en los cuales destacan:

- **Universal Serial Bus (USB):** es el tipo de puerto más común en toda computadora actual, diseñado para transferencia serial de datos a alta velocidad, comparable a la velocidad de los puertos en paralelo. Incorpora su propio regulador a 3.3 V para uso de transmisores-receptores externos.

- **Universal Synchronous/Asynchronous Receiver/Transmitter (USART)**: capaz de operar con el protocolo estándar RS-232, con detección automática de velocidad de *baud* y hasta 16 bits para mejorar la resolución.

- **Master Synchronous Serial Port (MSSP):** útil para la comunicación con otros dispositivos periféricos o microcontrolador, puede funcionar en uno de dos modos: Interfaz Periférico Serial (SPI) o Circuito inter-integrado (I^2C [4]). La interfaz soporta los siguientes modos de hardware:
    - Modo Maestro.
    - Modo Múltimaestro.
    - Modo esclavo.

2.1.2 Reloj DS1307.

El reloj de tiempo real DS1307 (RTC) es un reloj/calendario de 56 bytes de SRAM. Se maneja como esclavo de protocolo I^2C y proporciona segundos, minutos, horas, días, fecha, mes y año. El final de la fecha de mes se ajusta automáticamente a los meses con menos de 31 días, mismo caso para los años bisiestos. El reloj funciona tanto en el formato de 24 horas o de 12 horas con indicador AM/PM. El DS1307 tiene un circuito integrado que detecta fallas de energía con el cual cambia automáticamente a la alimentación de reserva.

2.1.3 Módulo de radiofrecuencia XBee.

XBee es un módulo de comunicaciones inalámbricas, fácil de utilizar y de sencillo montaje. Opera en una banda libre de 2.4 GHz de frecuencia que no requiere permiso legal para operar. La velocidad de transmisión de datos de una red es de hasta 256 kbps y es capaz de conformar redes flexibles y extensibles de hasta 65535 equipos (2^{16}), número máximo limitado por los bits de dirección (4 Bytes). El XBee permite diseñar redes con configuración en

[4] Anexo A.3 Bus I^2C Pág. 75

malla, punto a punto y punto a multipunto donde cada módulo puede ser configurado como coordinador, *routers* o *end devices* y su alcance está sujeto a:

- La serie de fabricación del módulo (serie 1 o 2).
- Tipo de módulo (XBee o XBee-Pro).
- Tipo de antena del módulo (chip, cable o dipolo).

Dependiendo de estas características la distancia de transmisión puede ser desde 30 hasta 100 metros en interiores (entre muros) y desde 100 hasta 1200 metros en exteriores (línea de visión directa).

2.1.4 Memoria EEPROM 24FC1025.

El microchip 24FC1025 EEPROM tiene una capacidad de almacenamiento de 128 kbytes. Ha sido desarrollado para aplicaciones avanzadas de baja potencia, como por ejemplo las comunicaciones personales o de adquisición de datos. Este dispositivo tiene la capacidad de escribir hasta 128 bytes de datos en una sola secuencia e igualmente realiza lecturas secuenciales o al azar. Sus líneas físicas de direccionamiento permiten hasta cuatro dispositivos en un mismo canal bus de datos, lo que equivale hasta 4 Mbits de memoria total en un bus.

2.2 Diseño del PCB.

La placa de circuito impreso (PCB) que se usó para la fabricación del concentrador es doble capa de cobre (top layer y bottom layer)[5] y se interconectan mediante la técnica de fabricación *through hole* (a través del orificio), con canales hechos con fresadora mecánica y estañada para evitar la degradación del depósito de cobre.

Dado que todos los dispositivos lógicos son encapsulados tipo DIP (Dual in-line package) cada uno emplea una base, las cuales fueron fijadas eléctrica y mecánicamente a la tarjeta con

[5] *Figura B.1 Top layer concentrador 3.0. Figura B.2 Bottom layer concentrador 3.0.* Pág. 78

soldadura de estaño, así es más fácil poder montar y desmotar un chip cuando requiera ser remplazado.

La distribución de los elementos del concentrador sobre el PCB se observa en la Figura 2.1. El subsistema B se encuentra en la parte izquierda de la imagen que corresponde físicamente a la parte frontal del concentrador, es donde está ubicada la antena externa del módulo XBee y el display de leds indicadores. El sistema A ilustrado en la parte derecha de la imagen y ubicado físicamente en la parte posterior de la placa, donde se encuentra el conector USB y el *plug* de la fuente de alimentación de voltaje directo.

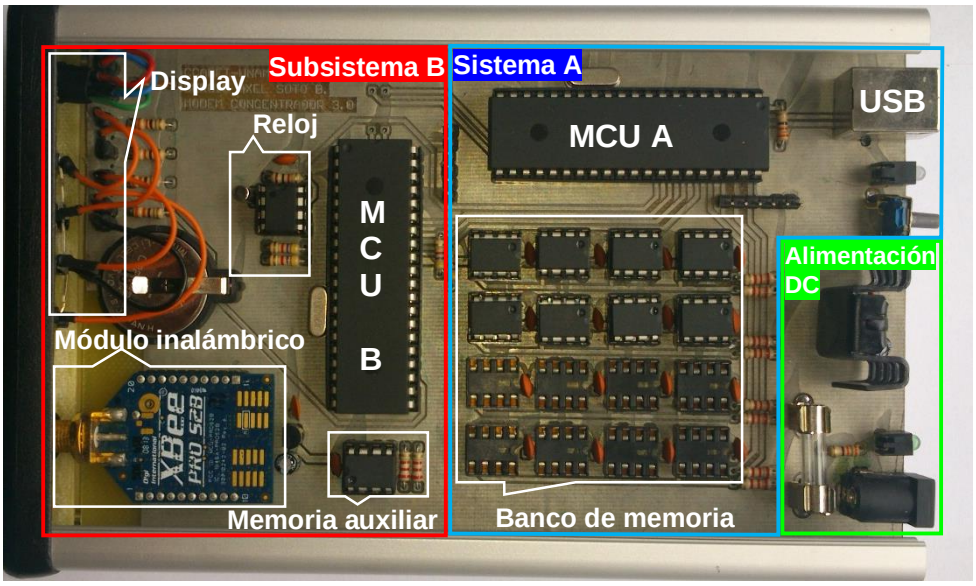

Figura 2.1 Imagen del hardware del concentrador.

El diseño electrónico del concentrador se observa en el esquemático general ubicado en apéndice B de este trabajo [6]. A continuación se explican las conexiones eléctricas para los elementos que conforman el concentrador así como los requisitos para el óptimo funcionamiento de los mismos.

[6] *Figura B.5 esquemático completo del concentrador 3.0*, pág. 80

2.2.1 Microcontrolador A y B.

El PIC18F4550 requiere un voltaje de alimentación de 5V y un cristal de cuarzo de 20 MHz que junto con el PLL programable del microcontrolador alcanza una frecuencia de reloj interno de 48 MHz. Los puertos múlti-propósito (port A, B, C, D y E) se usan para controlar los periféricos de cada sistema como se describe a continuación.

2.2.1.1 Microcontrolador A

El MCU A se encarga de administrar el banco de memoria y el subsistema B por medio de puertos virtuales, multímaestro y maestro respectivamente (Tabla 2.1). Se usan dos terminales de interrupciones externas como entradas para:

- INT0, conectada a la salida del oscilador del reloj DS1307, su usa para llevar acabo la cuenta del tiempo del intervalo de petición de datos de modo automático.
- INT1, se usa para coordinar la comunicación entre el MCU A y B, por medio del envió de pulsos eléctricos.

Puerto	**Pines**	**Etiqueta en la Figura 2.2**	**uso**
Virtuales multímaestro I^2C	puerto B y D	naranja	banco de memoria
Virtual I^2C	RB2, RB3	amarilla	salida de datos destinario MCU A
Pin de entrada	RB1/INT1	amarilla	Entrada interrupción de aviso del MCU A
Pin de entrada	RB0/INT0	morada	Entrada interrupción del oscilador DS1307
USB	RC4/D-, RC5/D+	azul	Conexiones del puerto USB
Pin de salida	RC0	azul	Led indicador del puerto USB

Tabla 2.1 uso de puertos del MCU B.

Para la comunicación USB las terminales D+ y D- están enlazadas directamente al conector USB, dicho conector no une el voltaje de puerto de la computadora (Vbus) con el voltaje del concentrador (VCC), ya que si bien ambos voltajes son de 5 V, la variación de voltaje en cualquiera de los dos puertos podría crear un circuito corto y causar daño en el puerto de la computadora, en el concentrador o en ambos sistemas.

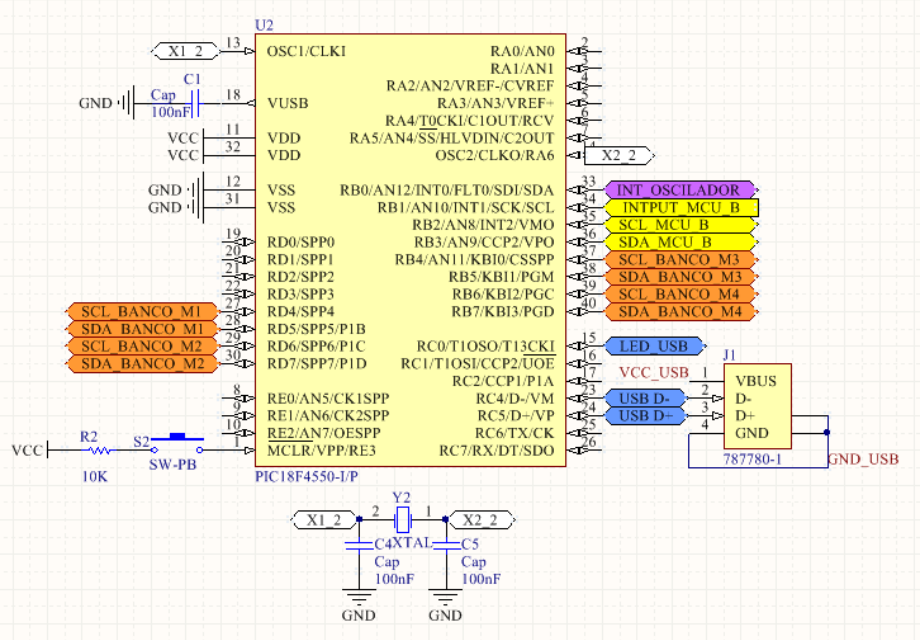

Figura 2.2 Diagrama de conexión para el MCU A.

2.2.1.2 Microcontrolador B.

En el MCU B los puertos se usan para conectar el módulo XBee, el reloj de tiempo real, la memoria auxiliar y el banco de memoria (Tabla 2.2*)*, estos tres últimos son manejados por medio de puertos virtuales simulados por programación, ya que el único puerto MSSP del microcontrolador se usa para las comunicaciones con el MCU A, donde el B funge como un dispositivo esclavo I^2C.

Puerto	**Pines**	**Etiqueta en la Figura 2.3**	**uso**
serial USART	RC6/Tx, RC7/Rx	azul	módulo XBee
virtual I^2C	RE0, RE1	verde	reloj DS1307
virtual I^2C	RD2, RD3	rosa	memoria auxiliar
virtuales multímaestro I^2C	puerto D y B	naranja	banco de memoria
MSPP I^2C (hardware)	RB0/SDA, RB1/CLS	amarilla	entrada de datos remitentes del MCU A
Pin de salida	RB2	amarilla	pulso de aviso a MCU A
Pin de salida	Puerto A	gris	Display y LED de estatus

Tabla 2.2 uso de puertos del MCU A.

El MCU B incluye un *display* de leds el cual controla desde su puerto A, en él se indica el estado de funcionamiento y la cantidad de memoria interna utilizada por el concentrador en la recaudación de mediciones.

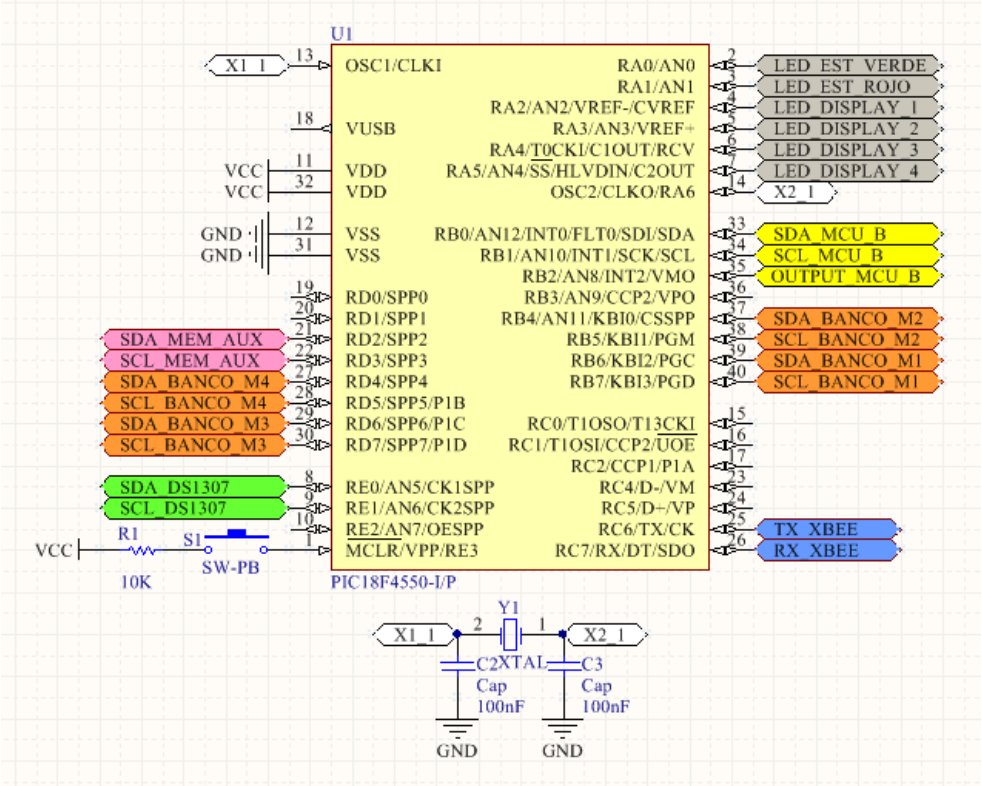

Figura 2.3 Diagrama de conexión para el MCU B.

2.2.2 Módulo de radiofrecuencia.

El dispositivo de radiofrecuencia empleado en el diseño del concentrador es un Xbee-pro 2.4 RF de la serie 2 con antena tipo dipolo, este modelo es el que tiene mayor distancia de transmisión en entornos cerrados y/o abiertos dentro de la serie de módulos XBee.

El hardware del módulo XBee no requiere usar circuitos externos y es capaz de cumplir sus funciones solo acoplándolo correctamente. Se alimenta con 3.3 V y se comunica por medio de cables unidireccionales tipo serial RS232, con sus terminales DOUT y DIN que son los canales de salida y de entrada de información del módulo respectivamente. Para conectar dichas terminales al MCU B es indispensable respetar los valores de voltaje y corriente que manejan, de lo contrario se corre el riesgo de dañar el módulo, dichos valores son:

- DOUT – 45 mA a 3.3 V.
- DIN – 55 mA a 3.3 V.

En el caso del DOUT la conexión de la terminal con el Tx del MCU B no presenta problema ya que el MCU B soporta hasta 5.5 V y 250 mA. Para la terminal DIN, el caso es distinto, ya que se encuentra comprometida por el voltaje de 5 V de salida en el Rx, por lo cual es necesario un divisor de tensión para ajustar el voltaje de 5 V a 3.3 V (Figura 2.4).

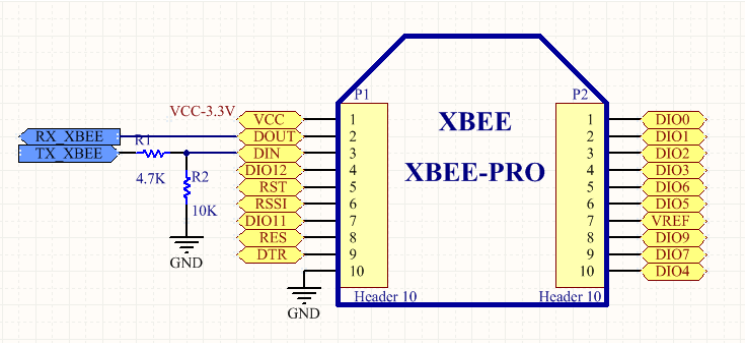

Figura 2.4 Diagramas de conexión del módulo XBee.

En la figura 2.5, se aprecia el área libre de metales respecto a la ubicación de la antena del XBee, el segmento es aproximadamente de 84 mm de largo y 15 mm de ancho. Los materiales metálicos cerca de la antena pueden interferir en el patrón de radiación, bloqueando la trayectoria del mismo o reduciendo la distancia de transmisión.

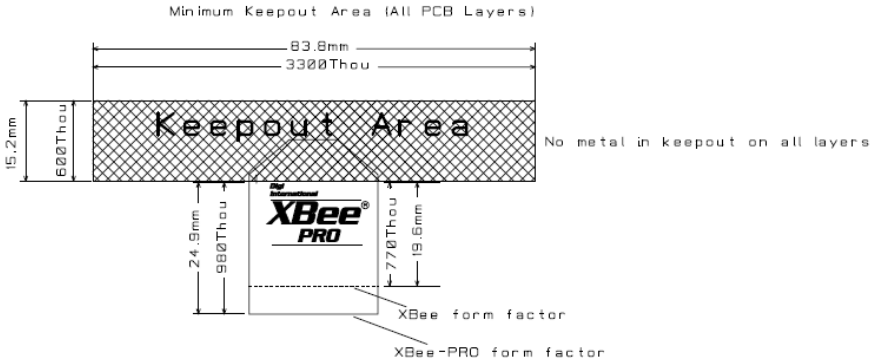

Figura 2.5 Área mínima de PCB libre de cobre para el módulo XBee. Nota de diseño: el fabricante recomienda mantener libre de cobre el área de PCB bajo la antena del módulo.

2.2.3 Reloj de tiempo real.

El circuito integrado DS1307 (RTC) funciona con un suministro de energía principal de 5V, con el cual el chip es totalmente accesible a escritura y lectura de información, cuando dicho suministro de energía falla o decae, el integrado cambia automáticamente a la fuente de

alimentación de respaldo (batería) e inhibe sus funciones de comunicación. Para trabajar correctamente el DS1307 requiere un cristal de cuarzo de 32.7 kHz y una celda estándar de litio de 3V tipo *coin*, que actúa como la fuente de energía de respaldo y asegura que el reloj siga funcionando cuando el concentrador esté apagado (Figura 2.6).

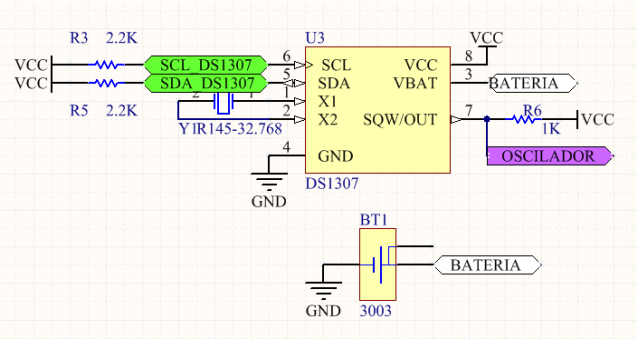

Figura 2.6 Diagrama de conexión del reloj.

El MCU B y el RTC se comunican por medio de un bus bidireccional I^2C donde las terminales SDA y SCL son de tipo *open drain* y requieren resistencias *pull-up* externas (Figura 2.6, etiqueta verde). La terminal de salida de onda cuadrada del reloj (SQW/OUT) está programada a 1 Hz [7] y se utiliza como un oscilador que activa una interrupción externa en el MCU A (Figura 2.2 etiqueta morada).

2.2.3.1 Resistencias *pull-up*.

En el cálculo de la resistencia *pull-up* se realizó para manejar la máxima velocidad de transferencia de datos. Primero usando la ecuación 2.1, donde el valor de la resistencia es estimado respecto a la frecuencia de la señal de conmutación a través de la línea:

$$R_{pull-up} = \frac{t_R}{C_B} \qquad \textit{Ecuación 2.1}$$

Donde:

- $R_{pull-up}$: Resistencia pull-up [Ω]

[7] A.1 Registros DS1307 pág. 64.

- t_R: tiempo de rizo [s]
- C_B: capacitancia máxima del bus [F]

Sustituyendo los valores nominales [8] para el protocolo I²C de C_B y t_R en la ecuación 2.1$\boldsymbol{R_{pull-up} = \frac{t_R}{C_B}}$. Se obtiene que:

$$R_{pull-up} = \frac{1000\ ns}{400\ pF} = 2500\Omega \qquad (2.1.a)$$

El resultado del cálculo de la resistencia *pull-up* (2.1.a) indica que el valor óptimo de resistencia es aproximadamente de 2.5 kΩ. Antes de fijar el valor de la resistencia se confirma que este es válido para polarizar las terminales *open drain* del integrado, para lo cual se emplea la Ecuación 2.2.

$$\boldsymbol{R_{pull-up} \geq \frac{VCC_{máx} - V_{OL}}{I_{OL}}} \qquad \textit{Ecuación 2.2}$$

Donde:

- $VCC_{máx}$: valor máximo del voltaje de alimentación [V]
- V_{0L}: voltaje lógico de 0 en terminal de salida [V]
- I_{0L}: corriente de terminal de salida [A]

De la Ecuación 2.2 se sustituyen $VCC_{máx}$, V_{0L} y $I_{0L,}$ por los valores del fabricante [9] en la hoja de especificaciones, de este modo se obtiene que:

$$R_{pull-up} \geq \frac{VCC_{máx} - V_{OL}}{I_{OL}} = \frac{5.5V - 0.4V}{5mA} = 1020\ \Omega \qquad (2.2.a)$$

Como lo indica el resultado del cálculo 2.2.a, el valor de la resistencia debe ser mayor a 1 kΩ. Observando los resultados de las ecuaciones 2.1 y 2.2, el valor óptimo para la resistencia *pull-up* que cumple ambas condiciones es de 2.2KΩ.

Todos los buses I²C del diseño del concentrador se calcularon para la máxima velocidad de transición, por lo tanto, todos los canales manejan el mismo valor de las resistencias *pull-up*.

[8] Datos obtenidos de la hoja de especificaciones pagina 3 - Bibliografía (maximintegrated.com, 2015)
[9] Datos obtenidos de la hoja de especificaciones pagina 2 - Bibliografía (microchip.com, 2015)

2.2.4 Memoria auxiliar.

La memoria auxiliar es un circuito integrado EEPROM 24FC1025, el cual se alimenta con 5 V y trasmite información por medio de bus serial bidireccional I^2C, con terminales tipo *open drain* (SDA, SCL) que requieren de resistencias *pull-up* externas [10], dichas terminales se unen a un puerto virtual I^2C del MCU B (Figura 2.3, etiqueta rosa), el chip usa la dirección física de 0xb00 (A0, A1) y es el único elemento en este bus (Figura 2.7).

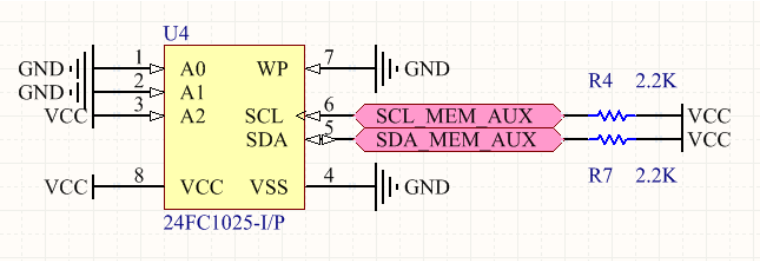

Figura 2.7 Diagrama de conexión para la memoria auxiliar.

El circuito integrado 24FC1025 distribuye su capacidad de memoria en dos bloques de 64 kbytes cada uno, como se muestra en la Tabla 2.3. Los primeros 16 bytes (x0000 - x000F) del bloque 1 están reservados para guardar los apuntadores de lista o estatus de lista. A partir del byte 17 del bloque 1, inicia el espacio de memoria asignado para guardar las IDs de los módulos XBee activos en la red de medidores (x0010- xFA00), como cada ID es de un tamaño de 4 bytes (celda), entonces la capacidad máxima del concentrador para la lista de módulos de radiofrecuencia es de hasta 15,996.

[10] Página anterior

Bloque físico	Aplicación	hexadecimal
Bloque 1	Espacio para datos de estatus de lista	x0000 x0001 x0002 : x000F
	Inicio del lista Celda ID # 1	x0010 x0011 x0012 x0013
	Celda ID # 2	x0014 x0015 x0016 x0017
	: Celda ID # 15,996 Fin de lista	: xF9FF xFA00
Bloque 2	memoria auxiliar para guardar apuntadores a lista de celdas	xFA01 xFA02 : x1F400

Tabla 2.3 Mapa de memoria auxiliar.

El bloque 2 se emplea como una memoria auxiliar en la cual se guardan las direcciones de las celdas que van quedando vacías al ser borradas, los apuntadores a celdas vacías en la lista se usan dentro de la programación del concentrador para mantener el orden secuencial en la lista, esta información se detallará en el capítulo 3.

2.2.5 Banco de memoria.

El banco de memoria es el sector del hardware dedicado al almacenamiento temporal de las mediciones, este se conforma de 16 circuitos integrados EEPROM 24FC1025 distribuidos en cuatro buses I^2C nombrados dentro del plano esquemático[11] como M1, M2, M3 y M4, cada bus tiene un capacidad de 512 kbytes y en total suman 2 Mbyte de memoria. Un bus consta de cuatro chips y cada circuito integrado ocupa una dirección física dentro del bus U1 - 0xb00, U2 - 0xb01, U3 - 0xb10, U4 - 0xb11 (Figura 2.8).

[11] *Figura B.5 esquemático completo del concentrador 3.0* , pág. 80

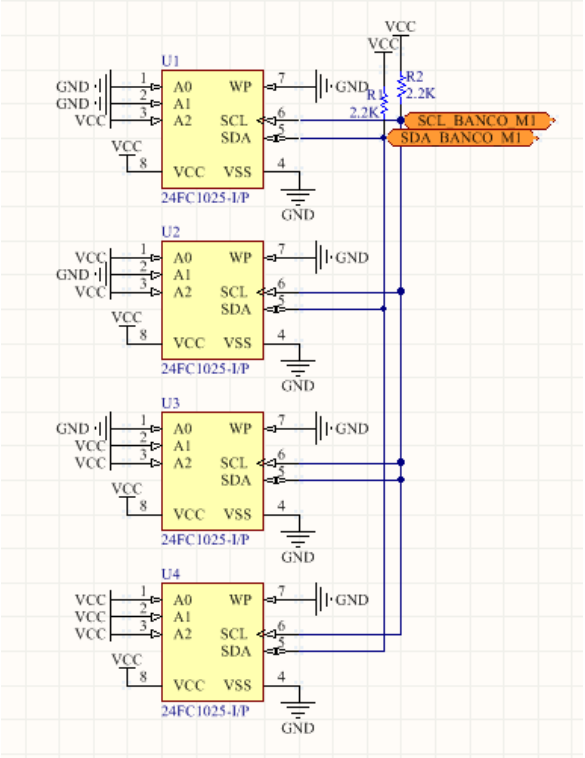

Figura 2.8 Conexión de bus M1 (Banco de memoria).

Las terminales de comunicación del bus M1 (SDA, SCL), están conectadas a los puertos virtuales I^2C en los MCUs A y B (etiquetas anaranjadas Figura 2.2 y Figura *2.3*), lo que define a este bus como un canal multímaestro I^2C[12], quiere decir que ambos microcontroladores comparten el dominio del bus y se turnan para el control de envío y lectura de información. El objetivo es que el MCU B pueda escribir la información recibida de los medidores y el MCU A substraiga la misma para exportarla a la computadora. El caso es igual para los buses M2, M3 y M4.

El volumen total del banco de memoria es de 2048 kbytes y en su totalidad se emplea para guardar paquetes de mediciones de 32 bytes de tamaño, es decir la capacidad de almacenamiento del concentrador es de 64,000 paquetes de mediciones.

[12] A.3 Bus I^2C pág. 75

2.2.6 Fuente de alimentación eléctrica.

La fuente de suministro de energía se conforma de tres etapas en cascada, primero un convertidor de voltaje de AC-DC externo y después dos reguladores de voltaje 5 V y 3.3 V. Los dispositivos que conforman el circuito del concentrador funcionan con un voltaje de alimentación de 5 V a excepción del módulo XBee, que funciona con un voltaje 3.3 V.

El convertidor AC-DC de tipo plug-in es la fuente de alimentación del concentrador (Figura 2.9), es un componente externo de conexión a pared que elimina la necesidad de dispositivos de refrigeración internos, reduciendo así el nivel de ruido, tamaño y peso del concentrador, además de mantener el calor fuera de los circuitos sensibles. Su entrada es 127 VAC a 60 Hz y proporciona una salida de 7.5 V a 800 mA.

Figura 2.9 Convertidor AC-DC de pared tipo plug-in.

El primer elemento dentro del PCB del concentrador es el fusible, el cual protege y evita destrucción total del circuito a causa de sobrecargas y/o corto circuito. Tomando en cuenta los valores de consumo de corriente nominal de los dispositivos electrónicos e incluyendo las resistencias de *pull-up*. Como se observa en la Tabla 2.4, el cálculo total aproximado de consumo de corriente en el concentrador es de 294.5 mA. Sin embargo el valor del fusible debe soportar el 80% más de valor nominal de la corriente total, esto es 530.1 mA, por lo tanto un fusible de 500 mA protege adecuadamente el circuito.

Dispositivo	cantidad	Corriente nominal (mA)	Total (mA)
MCU A (USB)	1	65	65.0
MCU B	1	50	50.0
DS1307	1	5	5.0
24FC1025	17	5	85.0
Resistencia pull-up	15	2.3	34.5
Módulo XBee	1	55	55.0
Total			294.5

Tabla 2.4 *Corriente nominal por dispositivo.*

La segunda etapa en cascada es el regulador de voltaje LM340T–5, un circuito integrado monolítico de encapsulado TO-22 (Figura 2.10). Este regulador tiene una salida de VCC = 5 V, con la disipación térmica adecuada permite una corriente de salida de hasta 1A. No necesita componentes externos y opera con temperatura de juntura de -40 °C a +125 °C.

Para saber si se necesitaría un disipador de calor se estimó la temperatura de juntura de acuerdo con la potencia del regulador, apoyando el cálculo en la siguiente ecuación;

$$T_J - T_A = P \times R\theta_{total} \quad \text{(Ecuación 2.3)}$$

Donde:

- T_J: temperatura máxima de la juntura [ºC]
- T_A: temperatura ambiental [ºC]
- P: potencia consumida por el disipador [W]
- $R\theta_{total}$: resistencia térmica desde la juntura al ambiente [ºC/W]

Despejando $T_{J\ estimada}$ y sustituyendo $R\theta_{total}$, por la sumatoria de la resistencia térmica de juntura a casco $R\theta_{J-C}$ y la de casco al ambiente $R\theta_{C-A}$ de la ecuación 2.3.

$$T_{J\ estimada} = \left[P \times \left(R\theta_{J-C} + R\theta_{C-A}\right)\right] + T_A \quad \text{(2.3.a)}$$

Sustituyendo en (2.3.a) los valores para el regulador LM340T–5 y la potencia a través del regulador, se tiene que:

$$T_{J\ estimada} = \left[\left[(7.5V - 5V) \times 0.5A\right] \times \left(5\,\tfrac{℃}{W} + 65\tfrac{℃}{W}\right)\right] + 25\ ℃ = \ 112.5\ ℃ \quad \text{(2.3.b)}$$

Con el cálculo 2.3b se observa que la temperatura estimada de la juntura del regulador es de 112.5°C. El regulador no alcanza la temperatura de juntura máxima de 150°C, por lo tanto no es necesario un disipador de calor. Solo con el propósito de reducir la temperatura del encapsulado se usó un disipador de tipo clip TO-22 cuya resistencia térmica es de 8°C/W.

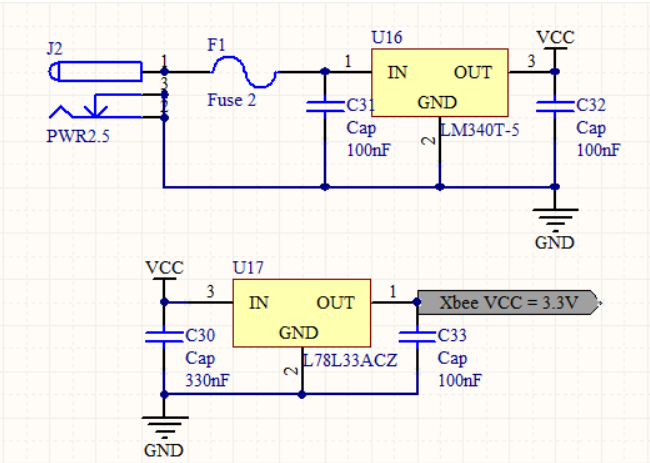

Figura 2.10 Reguladores LM340T–5 (5V) y L78L33ACZ (3.3V).

El último elemento de la fuente es el regulador de voltaje L78L33ACZ de encapsulado TO-92 (Figura 2.10), el cual adapta el voltaje principal VCC de 5V a 3.3V para energizar el módulo XBee. Este regulador no requiere de disipador de calor ya que como máximo entrega hasta 100mA de corriente y su potencia es de 0.17 Watts.

2.3 Montaje y ensamblado del concentrador

La tarjeta PCB de concentrador se montó dentro de una caja de aluminio (Figura 2.11), esto ayuda a disipar el calor que generan los componentes, además actúa como un blindaje electromagnético que aísla la antena del módulo XBee del resto del circuito, para evitar interferencias con el espectro de radiación. En el panel frontal de la caja de aluminio se encuentra un led bicolor (verde/rojo) que funge como indicador de estado, junto a una barra de 4 leds que se encienden secuencialmente conforme se va llenando el banco de memoria, lo que indica al usuario cuando es necesario realizar un respaldo. En la parte posterior de la caja de

aluminio se encuentra el *plug* para la fuente externa, el botón de *reset* y el conector tipo B de puerto USB.

Figura 2.11 concentrador final.

La caja de aluminio provee un blindaje al concentrador, el cual es tan bueno como sea la capacidad del material para atenuar los campos electromagnéticos en valores de decibeles [13], su efectividad total (S) es igual a la siguiente suma:

$$S = A + R + B\ [dB] \qquad \text{(Ecuación 2.4)}$$

Donde:

- A: son las pérdidas por absorción [dB]
- R: son las pérdidas por reflexión [dB]
- B: factor por múltiples reflexiones blindaje [dB]

El valor de R es de 60 dB para materiales metálicos no magnéticos como el aluminio a frecuencias mayores a 1 MHz [14]. Cuando una onda electromagnética pasa a través del blindaje esta se atenúa por el efecto Joule, esta pérdida representa el valor de A y depende de la profundidad de penetración, como se demuestra en la ecuación 2.5:

[13] A.4 Efectividad de los blindajes Pág 76.
[14] Figura A.4 Resumen de características de efectividad de los blindajes, *sin tener en cuenta las discontinuidades debidas a ranuras o juntas.*

$$A = 1314.3 * t \sqrt{(\mu_r * \sigma_r * f)}$$ (Ecuación 2.5)

Donde:

- t: es el espesor de material [cm]
- f: la frecuencia de la señal [MHz]
- μ_r: permeabilidad relativa del material.
- σ_r: conductividad relativa del material.

Sustituyendo en la ecuación 2.5 los valores de μ_r, σ_r, del aluminio, el espesor de la caja y la frecuencia de operación de módulo XBee, se tiene que:

$$A = 1314.3 * 0.1 \sqrt{(1 * 0.036 * 2400)} = 1221.66\ dB$$ (2.5.a)

Si el valor de A > 9 dB entonces el factor B es despreciable. Sustituyendo los valores de A y R en la ecuación 2.4, se tiene que:

$$S = 1221.66 + 60 + 0 = 1281.66\ dB$$ (2.4.a)

De esta manera se comprueba que la caja de aluminio cumple satisfactoriamente con su función de blindaje altamente atenuante al espectro electromagnético emitido desde la antena del módulo XBee.

2.4 Prototipos anteriores del concentrador.

Antes del prototipo final del concentrador se experimentó con dos diseños. El primer diseño fue planteado en protobord (Figura 2.12) y sirvió para realizar experimentos, ensayos de control de los diversos circuitos periféricos y pruebas de los protocolos de comunicación, como por ejemplo: serial RS232, I^2C virtual, I^2C múltimaestro, USB serial, etc. Dicho prototipo contaba con una pantalla LCD que servía de apoyo para saber lo que ocurría durante cada uno de los experimentos, en la siguiente versión se retiró el LCD del diseño.

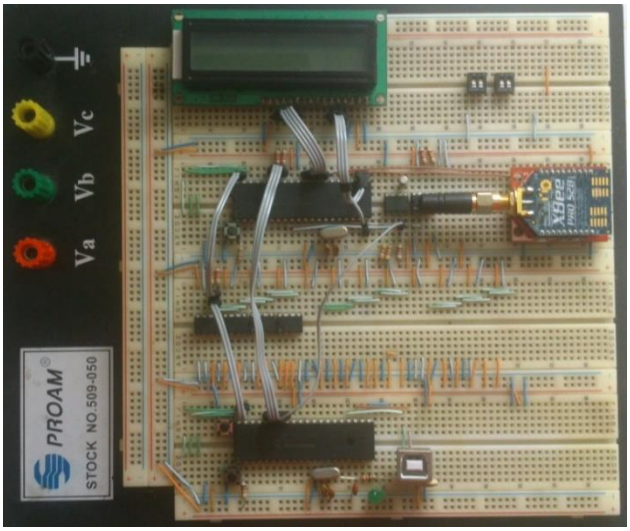

Figura 2.12 Prototipo de protobord.

Después de elegir las funciones y dispositivos que conformarían el diseño y arquitectura del concentrador, se desarrolló un prototipo de pruebas, siendo este el primero ya fabricado en PCB (Figura 2.13). Este diseño contaba con *headers* y puentes que se usaban para aislar circuitos periféricos y fuentes durante las pruebas. Además la base del módulo XBee era para usar el adaptador FTDI, el cual se conectaba a la computadora y mostraba los datos que se enviaban y recibían a través de la red. Las pruebas de red se realizaron con un banco de PICs y módulos XBee los cuales simulaban la red de medidores.

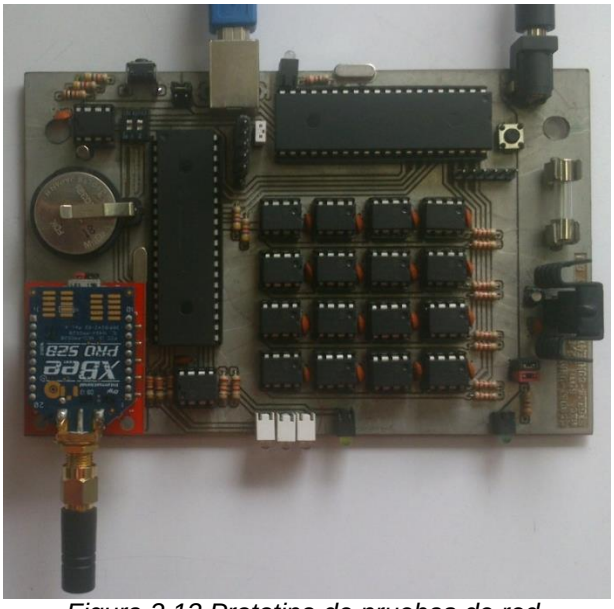

Figura 2.13 Prototipo de pruebas de red.

Capítulo 3 Software

La parte de software se maneja por medio de tres conceptos que son: las librerías de control de los elementos planteados en la arquitectura, las rutinas de trabajo para las funciones de configuración desde computadora y la rutina automática para la recopilación de mediciones. Estos tres elementos forman la programación del concentrador, en el cual, cada sistema y su microprocesador colabora con actividades que se interrelacionan entre sí, con el objetivo de llevar a cabo las tareas deseadas del concentrador.

3.1 Librerías de control

Las librerías de control son segmentos de código en lenguaje C estructurados para manejar de forma transparente los circuitos periféricos de cada sistema (reloj de tiempo real, módulo XBee, banco de memoria, etc.). A continuación se hace la descripción de cada librería basada en las características del circuito periférico que controla.

3.1.1 XBEE_S2.C [15]

Esta librería está diseñada para manejar el módulo XBee serie 2, en este dispositivo el intercambio de información entre el módulo y cualquier microcontrolador, es a través del envío y recepción de paquetes seriales de bytes en formato API (*Application Programming Interface*)[16]. Estos paquetes se usan para las funciones de lectura/configuración de parámetros, envío/recepción de información y respuestas de control a distancia. (Módulo – microcontrolador o módulo – módulo).

[15] El código de esa librería se muestra en el apéndice en la sección - Código C 4 pág. 99
[16] Consultar el apéndice en la sección - A.2 Protocolo API. Pág. 74

La lectura y configuración de parámetros, se manejan por medio de comandos AT (Attention) que son instrucciones codificadas que conforman un lenguaje de comunicación entre el usuario y la terminal o módem. Dichos comandos se utilizan para realizar consultas o cambios en las funciones internas del módulo. En la programación del concentrador se requieren dos comandos AT:

- ATID: se usa para dar valor al identificador de red (LAN), el cual es necesario para que un módulo XBee reconozca su pertenencia a una red en específico.
- ATSL: se usa para solicitar al módulo XBee los cuatro bytes bajos de su dirección única, lo que hasta el momento se ha manejado como el valor de ID o dirección única de un medidor.

La librería XBee está constituida por rutinas API de transmisión y recepción de paquetes así como funciones auxiliares para interpretar la información útil para el concentrador y los medidores. Las funciones de transmisión se clasifican de la siguiente forma:

- Envío de comandos AT – comunicación del microcontrolador a módulo XBee.
  - ATCommand(): Envío de un comando AT al módulo.
  - WriteATCommand(): Envío de un comando AT con cambio o configuración de parámetro.
- Envío de paquetes – comunicación de XBee a XBee.
  - CommandTransmission(): Envía un comando de librería de una ID remitente a otra ID de destinatario.
  - FrameTransmission(): Envía una trama de mediciones de una ID remitente a otra ID de destinatario.

Además de las funciones mencionadas, la librería usa palabras reservadas para facilitar la comunicación entre los módulos que conforman una red, principalmente entre el concentrador y los medidores. Las palabras reservadas son comandos que se usan para identificar el tipo de paquete que se recibe o se envía por parte de un módulo, tal como se muestra en la siguiente tabla (Tabla 3.1):

Comando de librería	Descripción
PETICION_EST	Se usa para buscar medidores en la red.
PETICION_MED	Se usa para realizar una petición de medición.
EST_OK	Respuesta ok.
EST_ERROR	Respuesta error
MED_NUEVO	Solicitud del medidor al concentrador para ser incorporado a la red
MED_ENVIO	Identifica un paquete API que contiene mediciones

Tabla 3.1 Comandos de librería Xbee_s2.C.

Otras funciones importantes para el funcionamiento de la librería son las que se describen a continuación.

3.1.1.1 Función IntXBee().

Esta función se encarga de recibir paquetes API, para lo cual primero reconoce el inicio de una secuencia API y después recupera la información contenida en el mismo. En la figura a continuación (Figura 3.1) se observa el diagrama de flujo que ilustra cómo trabaja esta función.

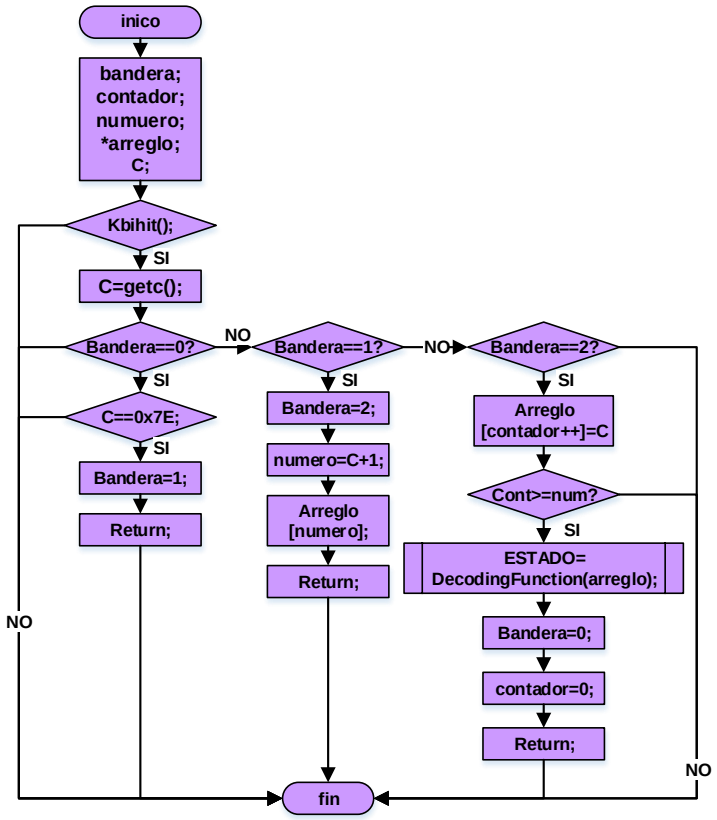

Figura 3.1 diagrama de función IntXbee().

De acuerdo a la figura 3.1, el algoritmo que describe la rutina IntXbee() es la secuencia de instrucciones siguiente:

1) La función kbihit() indica un nuevo byte en el puerto serial del microcontrolador, este byte se recupera en la variable 'C' usando la función getc().
2) Se realiza el reconocimiento del byte por comparación, si C es igual a 0x7E, este byte es el inicio de un paquete API, por lo cual la bandera cambia a '1'.
3) Con la bandera en '1' el siguiente byte recibido es el número total de bytes del paquete API, por lo cual este se guarda en la variable 'numero' más uno (por el byte de checksum), con este valor se crea un arreglo dinámico de dicho tamaño, para concluir con la creación del arreglo, la bandera cambia a valor '2'.
4) Con la bandera en '2' los siguientes bytes recibidos en el puerto son los datos de interés del paquete API, estos se almacenan en el arreglo dinámico conforme son recibidos, por cada byte recibido incrementa la variable 'contador'.
5) Cuando el 'contador' es igual a la variable 'numero' se envía el arreglo dinámico a la sub-rutina de decodificación (DecodingFunction()), la cual se encarga de interpretar la naturaleza del mensaje.
6) Una vez que se ha decodificado el mensaje, el arreglo dinámico se libera y se reinician todas las variables de la función para la próxima recepción de paquete.

La función DecodingFunction() sirve para interpretar la información recibida, para esto usa las tres siguientes funciones en las cuales se clasifica el tipo de paquete recibido y después cada una ejerce un acción correspondiente de la siguiente manera:

- ATCR(): (AT Command Response) se usa cuando el paquete API es una respuesta del módulo XBee a un comando AT enviado por el usuario anteriormente.
- TS(): (Transmit Starus) se usa cuando el paquete API es la respuesta de aviso al envío previo de un paquete de información.
- TP(): (Transmit Packet) se usa cuando el paquete API es el envío de información importante como por ejemplo un comando de librería o los valores de medición.

Toda la información obtenida por las funciones anteriores se copia en las variables globales de la librería XBee_S2.C, las cuales se describen en la Tabla 3.2. De esta forma se

pueden interpretar las acciones de envío y recepción de información dentro de la programación principal del concentrador, permitiendo manejar los valores de medición de manera más directa.

Tipo	Nombre	Descripción
int	ESTADO	Indica el tipo paquete API recibido por el módulo.
int32	IDENTIDAD	Almacena el ID propio del módulo en uso, ya sea el caso del concentrador o un medidor.
int32	NUEVA_ID	Contiene la ID del módulo remitente del paquete.
Float	VALORES	Almacena las mediciones, en el caso del medidor para enviarlas, y en caso del concentrador para recibirlas.

Tabla 3.2 Variables globales XBee.

3.1.2 RTC_DS1307.C [17]

La librería del reloj DS1307 utiliza secuencias lineales del protocolo I^2C para leer y escribir los valores tiempo y fecha (segundo, minuto, hora, día de la semana, día del mes, mes y año), tales valores se encuentran en formato de código binario decimal (BCD)[18] y se colocan en los registros propios del circuito integrado DS1307. El formato BCD es importante de considerar al interpretar los valores en formato decimal.

Las funciones de la librería son tres:

- WriteDate(). Sirve para configurar el reloj. Además esta rutina activa la salida del oscilador de onda cuadrada programada a 1 Hz.
- ReadDate(). Sirve para leer todos los registros del reloj.
- ReadTime(). Sirve para leer solo los registros de minuto, hora y día.

3.1.3 MEMORIA_AUXILIAR_24FC1025.C [19]

La parte más importante de la memoria auxiliar es la lista de direcciones de módulos XBee, la cual se conforma por un arreglo lineal tipo pila, con la peculiaridad que se ha adaptado a las

[17] Consultar el código completo en el apéndice - Código C 3 Pág. 98
[18] Consultar en el apéndice - Tabla A.1 Registros RTC. Pág. 73
[19] Consultar el código completo en el apéndice - Código C 5 Pág. 102

características de acceso del dispositivo periférico EEPROM 24FC1025. Como en el protocolo I^2C los registros se usan de manera serial para la lectura y escritura, se aprovechó esta característica para crear celdas de 4 bytes y usarlas secuencialmente, como se muestra en la Figura 3.2 en su inciso A. Esta forma evita la necesidad de usar métodos de localización de datos más elaborados o de direccionamientos igualmente complejos.

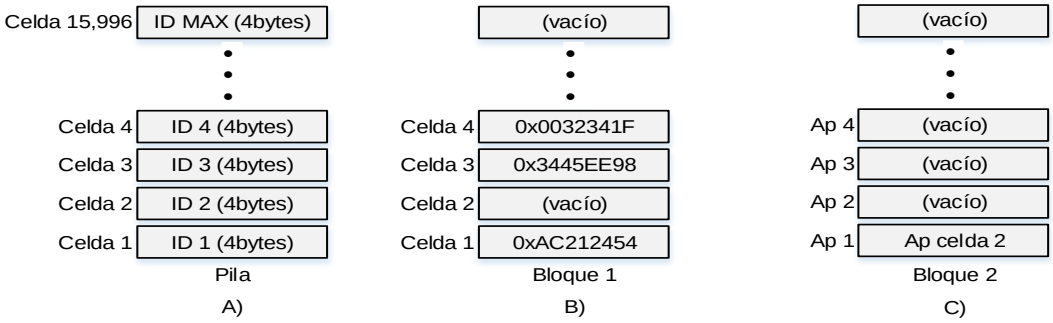

Figura 3.2 Pila en la memoria auxiliar.

La eficiencia en el uso de la memoria es muy importante, por lo cual las celdas borradas que quedan vacías implican perder tiempo en la lectura lineal, para evitar esto, el algoritmo de la librería considera ocupar estos espacios vacíos antes que uno nuevo, esto con ayuda de una lista auxiliar en el bloque 2 de memoria, la cual guarda los apuntadores de celdas vacías conforme fueron borradas. Un ejemplo se ilustra en la Figura 3.2 en su inciso B, donde se representa una pila con tres celdas ocupadas identificadas como 1, 3, 4 y una celda borrada identificada como 2. En el momento que se borra la celda 2 se guarda en la lista auxiliar (Figura 3.2 inciso C) un apuntador a dicha celda, de esta forma no se pierde la ubicación de la celda borrada y así para guardar la próxima ID nueva, primero se checa la lista y en este ejemplo la celda a ocupar será la número 2 de la lista. Este método asegura el aprovechamiento del espacio vacío.

Para manejar las listas de IDs se utilizan las siguientes funciones;

- IdXbeeSearch(). Busca una ID en lista, de existir la ID en alguna celda de la lista la función retorna un TRUE junto al número de celda donde ocurrió la coincidencia de búsqueda.

- SaveNewIdXbee(). Busca la ID en lista, de no existir la nueva ID en lista la función la guarda respetando el ocupar una celda nueva o una vacía disponible.
- DeleteIdXbee(). Busca la ID en lista, de si existir la ID en lista borra la celda y genera el apuntador en la lista auxiliar de celdas vacías.

Estas funciones utilizan un conjunto de valores de estatus de pila que se muestran en la Tabla 3.3, estos son apuntadores y contadores de IDs en la pila, dada la gran importancia de estas variables estas se guardan dentro de la memoria auxiliar.

Variables de estatus		**Descripción**
Apuntadores	Inicio	Marca la primera celda y es donde comienza la lista
	Fin	Apunta a la última celda de las ID guardadas
	Nueva celda	Apunta a la próxima celda nueva disponible para usar
Número IDs		Guarda el número total de direcciones de medidores en la red
Número celdas vacías		Es el número de celdas que han sido borradas en la lista

Tabla 3.3 variables de estatus de lista.

Para leer, escribir y reiniciar el estatus de la lista se usan las siguientes funciones:

- LoadListStatus().
- SaveListStatus().
- RebootListStatus().

Otro valor importante es el número de tramas almacenadas en el banco de memoria, dicho contador también se guarda en la memoria auxiliar, ya que si se da el caso de que el concentrador se apague inesperadamente antes de realizar el respaldo de la información en la computadora, al reiniciar el concentrador nuevamente sus función no pierde la información previamente almacenada y continua su trabajo gracias al contador de tramas.

Para leer, guardar y reiniciar dicho valor se usan las siguientes funciones:

- LoadNumer().
- SaveNumber().
- RebootNumber().

3.1.4 BANCO_MEMORIA_24CF1025.C [20]

Esta librería se utiliza para escribir y recuperar las tramas de mediciones en la memoria interna del concentrador. Una trama se conforma de un conjunto de variables de diferentes tipos, tal como se muestra en la Tabla 3.4, esta información se conforma por la dirección ID del medidor remitente (tipo entero de 32bits), el conjunto de valores enviados por este (cinco variables tipo flotante de 4bytes) y el tiempo de recepción (variable tipo entero de 8bytes). Todas estas variables conforman un arreglo unidimensional de datos de 27 bytes de longitud, en informática, el tamaño de palabra más cercano a dicha longitud es de 32 bytes, es por eso que las celdas para almacenar las mediciones en el banco de memoria sean de esta longitud.

Estructura	Tipo	Bytes	Nombre
ID XBEE	Int32	4	ID
TRAMA	Float	4	Voltaje
	Float	4	Corriente
	Float	4	Potencia
	Float	4	Factor de Potencia
	Float	4	Energía
TIEMPO	int	1	Minuto
	int	1	Hora
	int	1	Día

Tabla 3.4 trama de mediciones

Las funciones principales para manejar la información contenida en el banco de memoria desde programación principal de los microcontroladores son:

- FourDataFramesSave(). Se utiliza para almacena cuatro tramas de mediciones, aprovechando el modo de escritura por página característico del integrado 24FC1025 (hasta 128 bytes por rutina de escritura).
- SinglyDataFrameLoad(). Lee una trama de mediciones.

Estructuras	rango	Nombres
APUNTADOR	1-4	canal
	1-8	block
	0x0 - 0xFA00	dirección

Tabla 3.5 Apuntador de celda banco de memoria.

[20] Código C 6 Pág. 105

Como se observa en la Tabla *3.5* una celda puede ser ubicada en uno de los cuatro buses que a su vez cada uno se conforma de conforma por ocho bloques de memoria y cada bloque cuenta con 64mil registros cada uno (0xFA00). Los bloques tienen direcciones física ya definidas desde hardware como se muestra en la Tabla 3.6. Estas direcciones definen el orden de los dispositivos y son necesarias para realizar las comunicaciones I^2C.

Dispositivo	Bloque	Dirección
Chip 1	BLOCK1	0xA0
	BLOCK2	0xA8
Chip 2	BLOCK3	0xA2
	BLOCK4	0xAA
Chip 3	BLOCK5	0xA4
	BLOCK6	0xAC
Chip 4	BLOCK7	0xA6
	BLOCK8	0xAE

Tabla 3.6 Bloques de un bus EEPROM 24FC1025.

Para decodificar el apuntador de celda se utiliza la función MemoryAddressDecoder(), se utiliza para que dado un número de celda dentro 0-64000 posibles, esta retorna el apuntador correspondiente de la posición de celda deseada, lo cual hace trasparente para el usuario el manejo de la información al momento de leer o escribir dentro del banco de memoria. El algoritmo se asemeja al método de búsqueda en árbol como se observa en la figura a continuación (

Figura *3.3*).

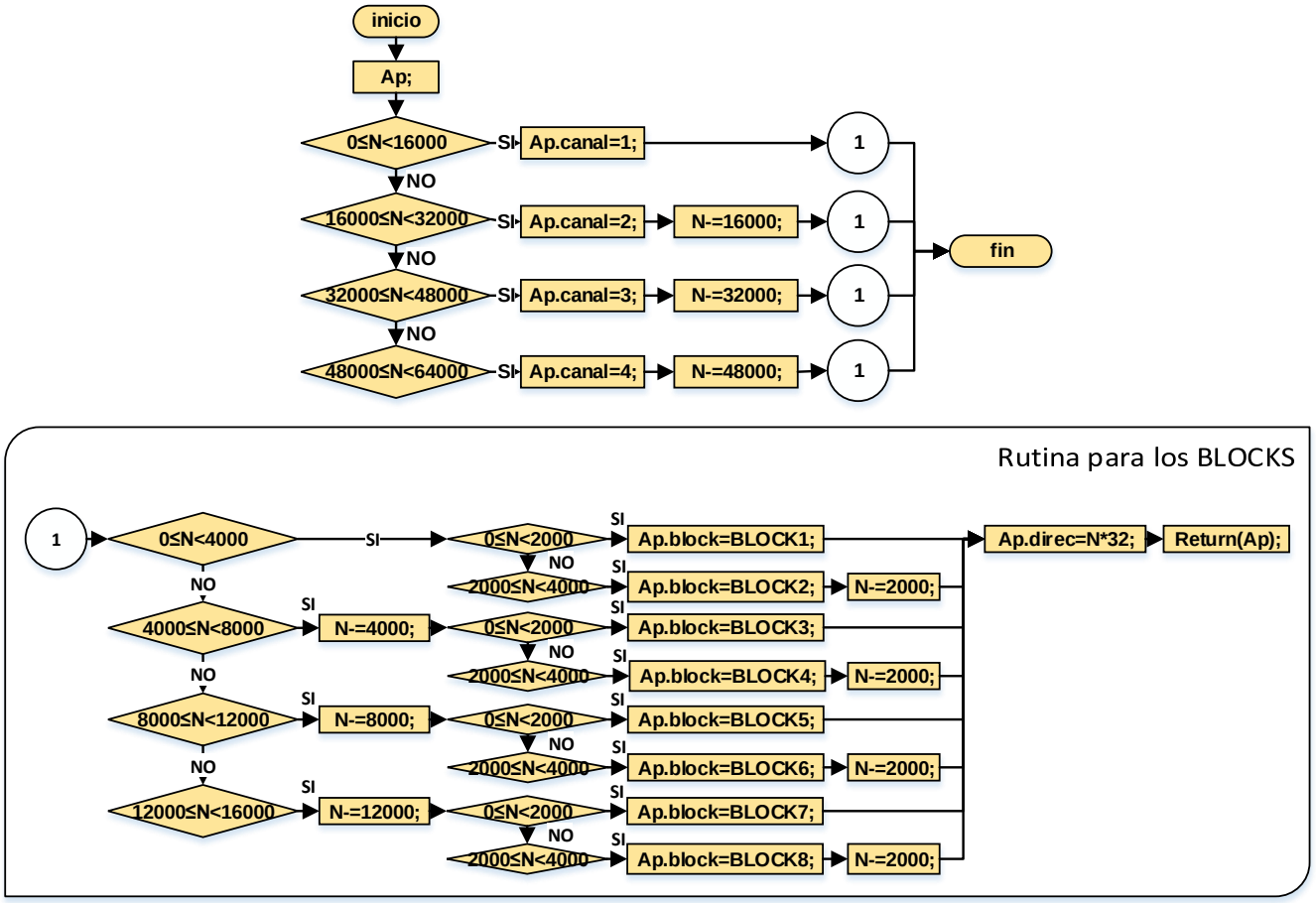

Figura 3.3 Diagrama de flujo de la función MemoryAddressDecoder().

El diagrama de la función de MemoryAddressDecoder() (figura 3.3) se interpreta con ayuda del algoritmo siguiente:

1) La entrada de la función es un número de celda ‘N’.
2) Primero se buscar el bus para N, se evalúa el valor de N para los intervalos [0,16mil), [16mil, 32mil), [32mil, 48mil) y [48mil, 64mil), que delimitan los canal 1, 2, 3 y 4 respectivamente. Una vez ubicado a N dentro de un intervalo, se asigna el número de bus a ‘Ap.canal’ y se le resta a N el límite inferior del intervalo.
3) El nuevo valor de N se envía a una segunda función para buscar el bloque y dirección en el bus.
4) Ahora se busca N dentro de cuatro rangos posibles [0, 4mil), [4mil, 8mil), [8mil, 12mil) y [12mil, 16mil), estos representan los chips en un bus y los ocho bloques de memoria que

lo conforman, una vez ubicado N en un dispositivo nuevamente se le resta el límite inferior de su rango.

5) Ya seleccionado el chip el nuevo valor de N está dentro uno de los dos posibles rangos [0,2mil) y [2mil, 4mil), que representan los bloques del chip, se asigna el correspondiente a 'Ap.block' y se le resta a N el límite inferior nuevamente.
6) El valor final de N se multiplica por el tamaño de una celda *(*32 bytes), para obtener la dirección del registro y se asigna a 'Ap.dirección'*.*
7) La función retorna los valores del apuntador ya con el canal, el bloque y la dirección de registro deseada y finaliza.

3.1.5 USB_CDC.H comunicación USB

Esta librería diseñada por CCS Inc. (*Custom Computer Services, Inc*), es abierta para estudiantes e investigadores, su aplicación es para simular un enlace RS232 virtual entre la computadora y cualquier microcontrolador de la compañía Microchip perteneciente a las familias de PICs 18F4450, 2450, 87J50, 46J50, 13K50, 14K50.

Esta librería requiere de un driver complementario llamado USB_DESC_CDC.H el cual debe ser instalado en la computadora y es compatible con la mayoría de las versiones de Windows Microsoft. Este driver contiene un conjunto de comandos que describen un dispositivo tipo CDC (*Communication Device Class*) para crear un puerto COM virtual que simula un puerto COMx, desde el cual se puede escribir y leer en el dispositivo microcontrolador de forma semejante como cualquier dispositivo serie que tenga un puerto COMx.

El programa principal del concentrador utiliza esta librería para comunicarse con la computadora usando tres de sus funciones:

- Usb_cdc_kbhit(). Devuelve TRUE si hay uno o más caracteres recibidos y esperando en el buffer de recepción.
- Usb_cdc_getc(). Obtiene un carácter del buffer de recepción, si no hay datos en el mismo se esperará hasta haya algún valor para leer.

- Usb_cdc_putc(char c). Coloca un carácter en el buffer de transmisión, si este está lleno esperará hasta que se desocupe antes de poner el nuevo carácter.

El intercambio de información entre el MCU B y la computadora es por medio de paquetes seriales en el formato de API, tal como lo es para el manejo de módulo XBee. Por lo cual la función IntUSBLabVIEW()[21] propia para la comunicación de la computadora con el MCU B es similar a la función IntXbee()[22], la diferencia es que cambian las funciones de espera y recepción de un carácter kbhit() y getc() por Usb_cdc_kbhit() y Usb_cdc_getc() respectivamente.

3.2 Programación.

3.2.1 Comunicación entre el Sistema A y subsistema B.

Los dos sistemas que conforman la arquitectura del concentrador (sistema A y subsistema B) tienen una relación de *maestro y esclavo,* esto quiere decir que el MCU A es el dispositivo *maestro* que tiene el domino y control del funcionamiento del subsistema B. Para esto el MCU B tiene una estructura de control similar al de un dispositivo *esclavo,* ya que cuenta con un buffer de registros el cual se usa para recibir órdenes, datos o alistar información que puede substraerse en cualquier momento del MCU A.

El buffer del MCU B es un arreglo global unidimensional de 18 bytes que simula registros de control los cuales se identifican cada uno con una etiqueta dependiendo del propósito o de la variable que manejen en específico, tal como se observa en la Tabla 3.7.

Los primeros tres registros del buffer son para el control de las actividades de MCU B.

- CONTROL. Se usa para las órdenes principales.
- FUNCION. Es donde se indica la tarea a llevar a cabo por el subsistema.

[21] Rutina del modo automático. Pág. 56
[22] Función IntXBee(). Pág. 35

- EST_F. Una vez que se haya realizado una tarea solicitada este registro indica el estatus con que finalizo dicha función.

Después de los registros para el control siguen los dedicados a la información.

- N_ID_H, N_ID_L. Son un entero de 16 bits para indicar el número total de ID almacenadas en la memoria auxiliar.
- ID_BUF0, ID_BUF1, ID_BUF2, ID_BUF3. Cuatro registros para recibir una dirección de módulo XBee.
- REGISTROS DE RELOJ: siete registros para enviar o recibir los valores de configuración del reloj de tiempo real.
- NUMT_H, NUMT_L: juntos son un entero de 16 bits para indicar el número de tramas de mediciones almacenadas en el banco de memoria.

nombre de registro		**Etiqueta**	**dirección**
Control		CONTROL	0x00
Función		FUNCION	0x01
Estado de función		EST_F	0x04
Número total de medidores	IDs byte alto	N_ID_H	0x02
	IDs byte bajo	N_ID_L	0x03
Dirección de módulo XBee	ID Byte 0	ID_BUF0	0x05
	ID Byte 1	ID_BUF1	0x06
	ID Byte 2	ID_BUF2	0x07
	ID Byte 3	ID_BUF3	0x08
Valores para el reloj	segundos	SEGUNDO	0x09
	minutos	MINUTO	0x0A
	hora	HORA	0x0B
	día de la semana	DIA_SEM	0x0C
	día del mes	DIA_MES	0x0D
	mes	MES	0x0E
	año	ANO	0x0F
Numero de tramas almacenadas	Tramas byte alto	NUMT_H	0x10
	Tramas byte bajo	NUMT_L	0x11

Tabla 3.7 Registros del MCU A

Para manejar los registros son importantes también las etiquetas que dentro de la programación y que ambos MCU conocen y manejan (tabla 3.8). Las etiquetas se dividen en tres grupos; funciones, control y respuestas básicas, respuestas de búsqueda de ID y por ultimo elementos de red, los cuales son necesarios para configurar la misma.

Constante		Etiqueta	Valor byte
Funciones para el MCU A	Leer Reloj	F1	0xF1
	Configurar Reloj	F2	0xF2
	Agregar ID XBee	F3	0xF3
	Borrar ID XBee	F4	0xF4
	Buscar ID XBee	F5	0xF5
	Reiniciar Memoria ID XBee	F6	0xF6
	Petición Tramas	F9	0xF9
Control y respuestas básicas		STOP	0x80
		PLAY	0x84
		OK	0x88
		ERROR	0x89
Respuestas de búsqueda	Si está en red, Si en memoria	YY	0x8A
	Si está en red, No en memoria	YN	0x8B
	No está en red, SI en memoria	NY	0x8C
	No está en red, No en memoria	NN	0x8D
Elementos de red	Identificador de red	NETWORK	0x3032
	Dirección coordinador	COORDINADOR	0x4091D114

Tabla 3.8 etiquetas de los MCU A y B.

Como se menciona previamente en el capítulo 2, la comunicación entre los microcontroladores es por medio de un bus bidireccional I^2C, donde el puerto virtual es definido por software en caso del MCU A, ya que solo se necesita para generar los estados de protocolo como *maestro*, sin embargo en caso del MCU B el puerto es el MSSP incluido en hardware, es completamente necesario que sea así, ya que para el manejo del microcontrolador como un *esclavo* se requiere la interrupción propia del puerto.

La parte del MCU B involucrada en la comunicación entre sistemas es la interrupción SSP, cuyo *identificador de esclavo* es definido para el puerto en uso en el *setup* de mismo, en este caso ‘Slave 0xC0’, todo esto está ligado a la función ssp_interrupcion() que junto al buffer de registros constituyen la arquitectura de *esclavo*. La función se caracteriza por poder detectar el tipo de byte recibido en una serie en protocolo I^2C, además administra los registros ya sea para escritura o lectura. En la figura a continuación se describe su diagrama de flujo (figura 3.4).

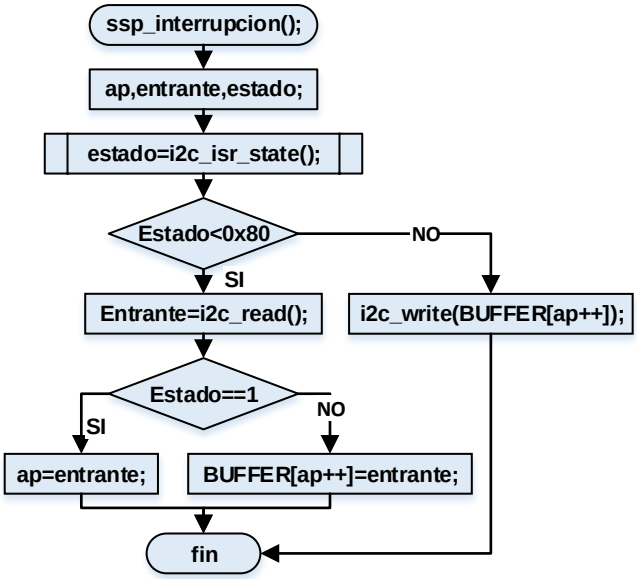

Figura 3.4 Diagrama interrupción SSP

Descripción de algoritmo:

1) La función ocupa tres variables estáticas '*ap'*, '*entrante'* y '*estado'*.
2) i2c_isr_state(); indica el estado el recepción de byte, el valor leído se almacena en *estado*. Existen dos casos para este valor.
3) **Si *estado* es menor a 0x80:** significa que *el maestro* envió un byte en modo escritura.
    a. se lee el valor enviado y se guarda en la variable *entrante.*
    b. si *estado* es igual a 1: *entrante* es un apuntador de registro y se guarda en la variable *ap*,
    c. si *estado* es mayor a 1: *entrante* es información, usando *ap* se direcciona registro del buffer y se guarda el byte, al final se incrementa el apuntador para el caso de escritura continua.
4) **Si *estado* es mayor a 0x80:** *el maestro* solicita modo lectura.
    a. Con cada *estado* entrante en modo lectura la función envía el valor del registro del buffer al que apunta ap y para lectura continua el apuntador se decrementa con cada intervención.

En la contraparte al *esclavo*, las funciones del *maestro* (MCU A) son secuencias de protocolo I^2C, direccionadas al *identificador de dispositivo esclavo,* las cuales se describen en la siguiente tabla (Tabla 3.9)

Función	Descripción
WriteByteSlave	Escribe un dato en un registro deseado
ReadByteSlave	lee el contenido de un registro deseado
WriteDateSlave	Envía la fecha y hora a los registros correspondientes
ReadDateSlave	lee los registros de fecha y hora
WriteIDSlave	Envía una ID a los registros correspondientes
NumID	lee los registros del número de IDs

Tabla 3.9 Rutinas básicas para el canal de control.

3.2.2 Programa del microcontrolador A [23]

La programación del MCU A sigue el diagrama de flujo siguiente (figura 3.5), el cual se conforma de una cabecera (*setup*) y un ciclo o lazo infinito:

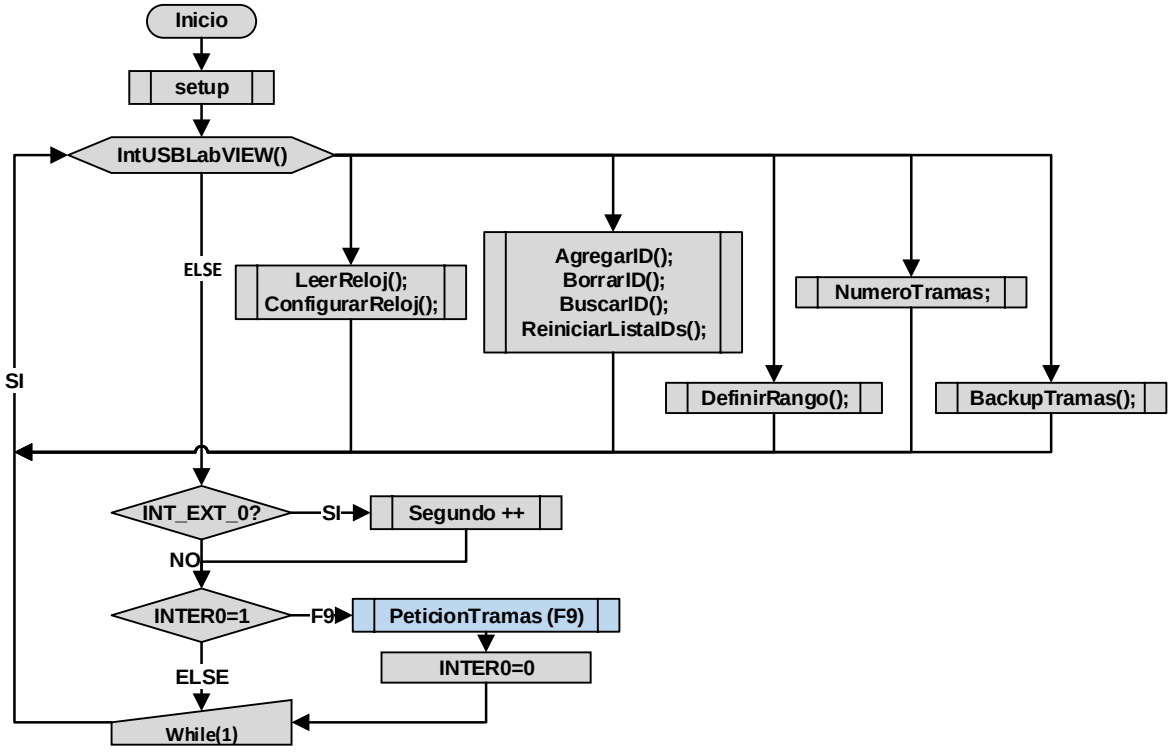

Figura 3.5 diagrama de flujo del microcontrolador A.

[23] Código C 1 Pág. 88

- *Setup*: en esta parte se realizan la configuración del reloj interno, los puertos y las terminales de entrada/salida, además se considera que las configuraciones de inicio pertenecen a esta parte.
- Lazo infinito: es el ciclo de actividades principales, cuando el concentrador recibe órdenes de la computadora y cuando solo se dedica a la recopilación de información.
    - Actividad con conexión a la computadora: es cuando la interfaz del usuario envía un comando para solicitar una función o realizar una configuración.
    - Actividad en modo automático: donde la interrupción conectada al reloj externo con una señal de 1Hz cuenta los minutos y cada que el conteo es igual a el RANGO se activa una bandera (INTER0) y el MCU A ordena al subsistema de recopilación que inicie un ciclo de petición.

En el *setup* se configuran las dos librerías que el MCU A requiere, estas son:

- USB_CDC.H
- BANCO_24CF1025.C

La primera librería se utiliza para la comunicación con la computadora y la segunda para extraer e importar los datos almacenados en el banco de memoria, esta se conecta al programa por medio de las etiquetas de la configuración de puertos multímaestro virtuales (Tabla 3.10).

dispositivo	**Tipo**	**SDA**	**SCL**	**configuración**	**Etiqueta**
MCU B	master	RB3	RB2	software	I2C_CTRL
M1	multi_master	RD5	RD4	software	I2C_BANCO_M1
M2	multi_master	RD7	RD6	software	I2C_BANCO_M2
M3	multi_master	RB5	RB4	software	I2C_BANCO_M3
M4	multi_master	RB7	RB6	software	I2C_BANCO_M4

Tabla 3.10 Configuración de puertos multímaestro para el MCU A

En lazo infinito lo primero son las rutinas compuestas para realizar tareas en específico y estas se clasifican en cuatro tipos:

- **Administración del reloj:** LeerReloj() y ConfigurarReloj().
- **Manejo de medidores:** AgregarID(), BorrarID(), BuscarID() y ReiniciarListaIDs().
- **Parámetro de tiempo**: DefinirRango().
- **Reportes y respaldo**: NumeroTramas() y BackupTramas().

Cada una de estas rutinas inicia con la recepción de un comando-API enviado desde la interfaz de usuario, donde la función IntUSBLabVIEW() identifica un paquete y substrae la información, después se lleva a cabo la rutina deseada, para finalmente retornar la información a la interfaz usando nuevamente un paquete-API, para esto el MCU A usa las siguientes funciones.

- FechaHora_PortUSB(): Envía la fecha y hora leída o actualizada del reloj RTC.
- ID_PortUSB(): Envía el número de IDs, el estatus de ID y una ID.
- RANGO_PortUSB(): Retorna a la computadora el valor actual del RANGO.
- NumT_PortUSB(): Retorna el número de tramas de mediciones almacenadas hasta el momento.

Por último, cuando el concentrador se encuentra trabajando en modo automático el MCU A se encarga de llevar el control del intervalo de tiempo. Usando la interrupción externa y la función ligada a esta Oscilador_Interrupcion(), la cual se encarga de contar los minutos trascurrido en tiempo real, cuando el valor de contador alcanza el valor de la variable RANGO, el MCU A da la orden al subsistema B para iniciar un ciclo de peticiones.

3.2.3 Rutinas para realizar las configuraciones desde la interfaz de usuario.

3.2.3.1 Administración del reloj.

En esta rutina intervienen la interfaz de usuario y los dos sistemas del concentrador como se muestra en la figura 3.6. Primero el usuario selecciona la función para leer o configurar el reloj, la interfaz envía un paquete API por el puerto USB al MCU A, el cual ejecuta la tarea deseada, este empieza escribiendo los comando de control STOP y función (F1, F2) al buffer del MCU B para deshabilitar su trabajo, en el caso de que el proceso sea de configuración también se copian el tiempo y fecha dados por la interfaz. El sistema B lleva a cabo la lectura o configuración del reloj RTC y reasigna los nuevos valores leídos en su buffer, para finalizar su intervención manda un pulso de aviso al MCU A, este lee los nuevos valores y los manda a la computadora usando la función FechaHora_PortUSB(), dichos valores los recupera la interfaz

y los muestra al usuario. Por último el MCU A vuelve a habilitar al subsistema B cambiando el estado del registro de control a PLAY y el concentrador regresa a su trabajo automático de recaudación.

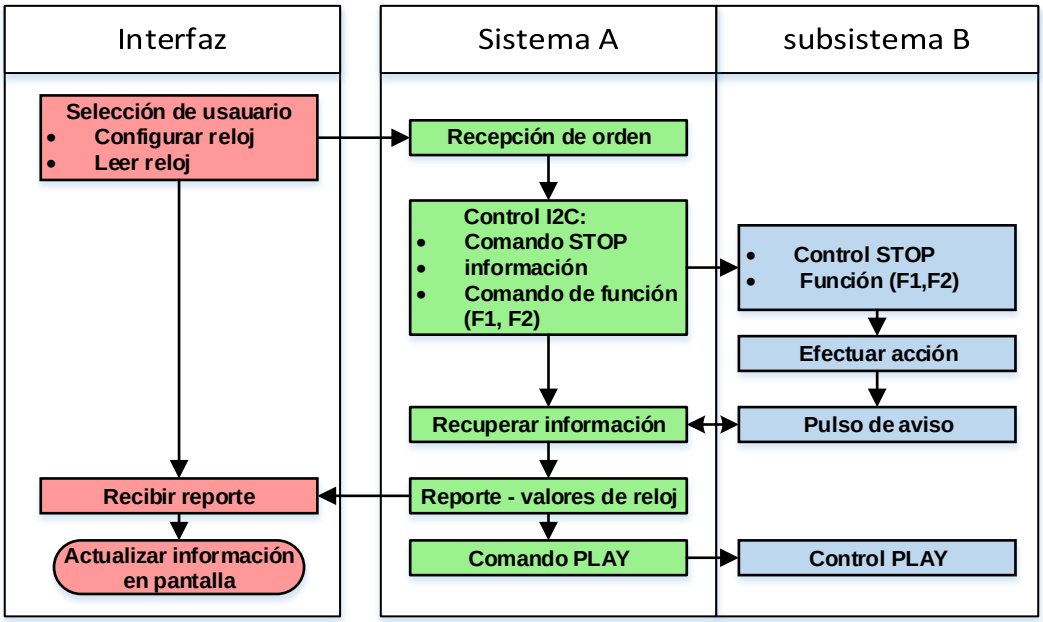

Figura 3.6 Rutina para el manejo del reloj RTC.

3.2.3.2 Manejo de medidores.

En la figura 3.7 se ilustran las rutinas para agregar, borrar y buscar un medidor en red, en estas rutinas intervienen el usuario, el concentrador y la red de medidores. Primero la interfaz envía al concentrador un paquete API el cual contiene una ID y un comando para indicar una de las tres tareas, dicha información la recibe el MCU A que comienza la función deseada detenido la activad de subsistema B y escribe la ID en el buffer del MCU B junto con el comando de función (F3, F4, F5), con esta orden el subsistema B realiza la búsqueda del medidor en red y lista de IDs, de existir la dirección de módulo esta se agrega o borra de la lista según sea el caso, por ultimo este MCU reporta el resultado de operación en su buffer y envía el pulso de aviso, después el MCU A lee los registros del estado lista y los manda a la computadora usando la función ID_PortUSB(), la interfaz interpreta dicho resultado y muestra la respuesta al usuario. Por último el MCU A envía el comando de PLAY al registro de control del subsistema B y el concentrador regresa a su estado automático.

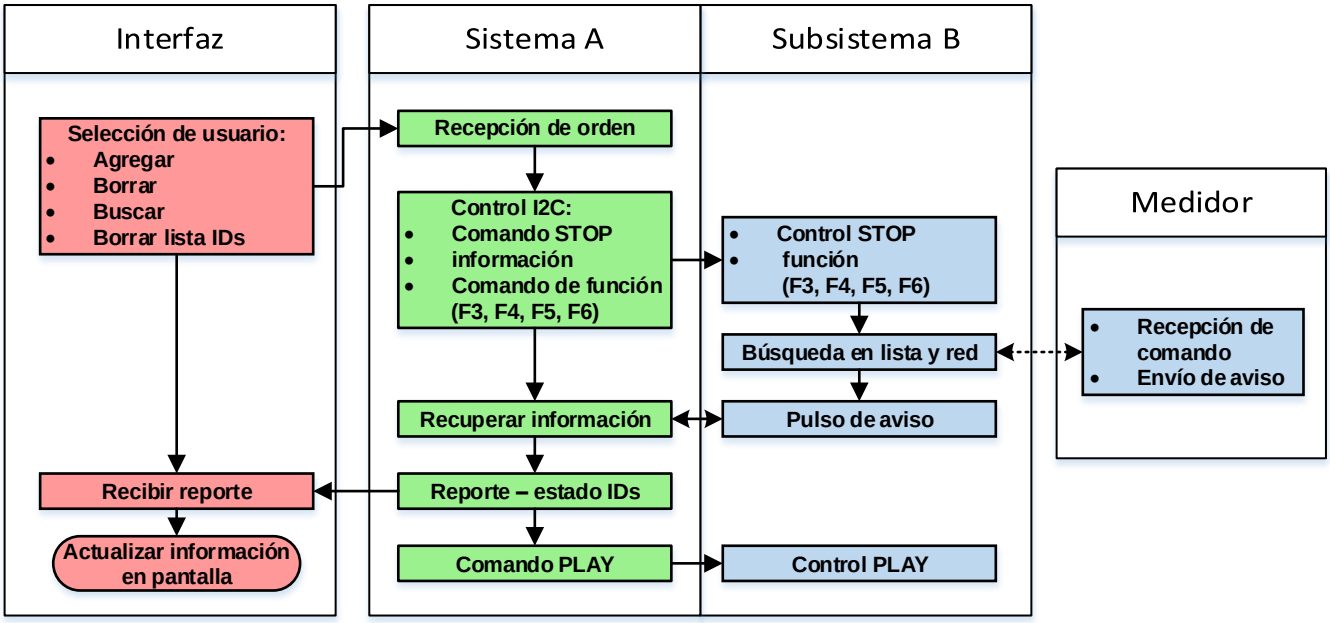

Figura 3.7 Rutina para el manejo de la direcciones de medidores

La rutina para borrar la lista de direcciones de medidores es igual a las demás hasta que la orden llega al MCU B quien cambia al valor cero los apuntadores de la lista reiniciando así la lista.

3.2.3.3 Parámetro de tiempo

Esta función se usa para configurar el intervalo de tiempo, por default el valor de este rango es de 15 minutos pero el usuario puede cambiarlo desde la interfaz, la cual envía el valor deseado al MCU A y este asigna el nuevo valor su variable global de RANGO y después retorna el mismo a la computadora usando RANGO_PortUSB(), esto para confirmar al usuario que el intervalo de tiempo efectivamente fue cambiado. Esta rutina se ilustra en la figura 3.8.

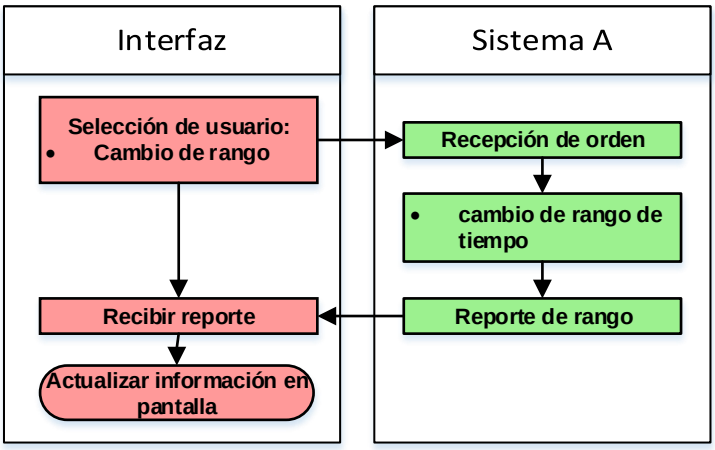

Figura 3.8 Rutina de cambio de rango de tiempo.

3.2.3.4 Reporte y respaldo de tramas

La rutina NumeroTramas() se usa para realizar una consulta del total de tramas que han sido ya guardadas en el banco de memoria. Cuando el usuario desea saber dicha información desde la interfaz se manda un comando para dicha acción. El MCU A recibe la orden y lee del buffer del MCU B el número de tramas, después manda este valor a la computadora usando la función NumT_PortUSB().

La función BackupTramas() se encarga hacer el reporte de información cuando el usuario lo desee. Primero la interfaz manda el comando de función al concentrador, donde el MCU A la recibe y detiene la actividad del subsistema B para leer el valor de numero de tramas del buffer y tomar el control de los buses del banco de memoria, después lee cuatro tramas y las manda a la computadora, el mismo microcontrolador repite esta acción cíclicamente hasta haber trasladado el total de datos a la computadora. Cuando el respaldo termina se cierra la comunicación con la computador y el MCU A ordena al subsistema B retomar su trabajo de recaudación iniciando una nueva cuanta de tramas.

En la figura 3.9 se ilustran las rutinas de consulta y respaldo de tramas.

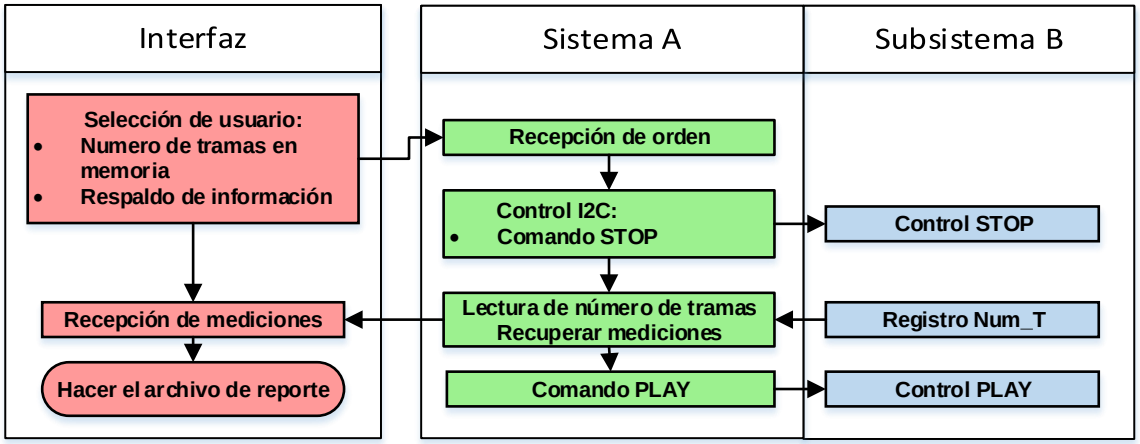

Figura 3.9 Rutinas de consulta y respaldo de tramas.

3.2.4 Programación del microcontrolador B [24]

El programa del MCU B se describe en el diagrama de flujo propio (Figura 3.10). De la misma forma que para el microcontrolador A, este se conforma de un *setup* y un ciclo de lazo infinito.

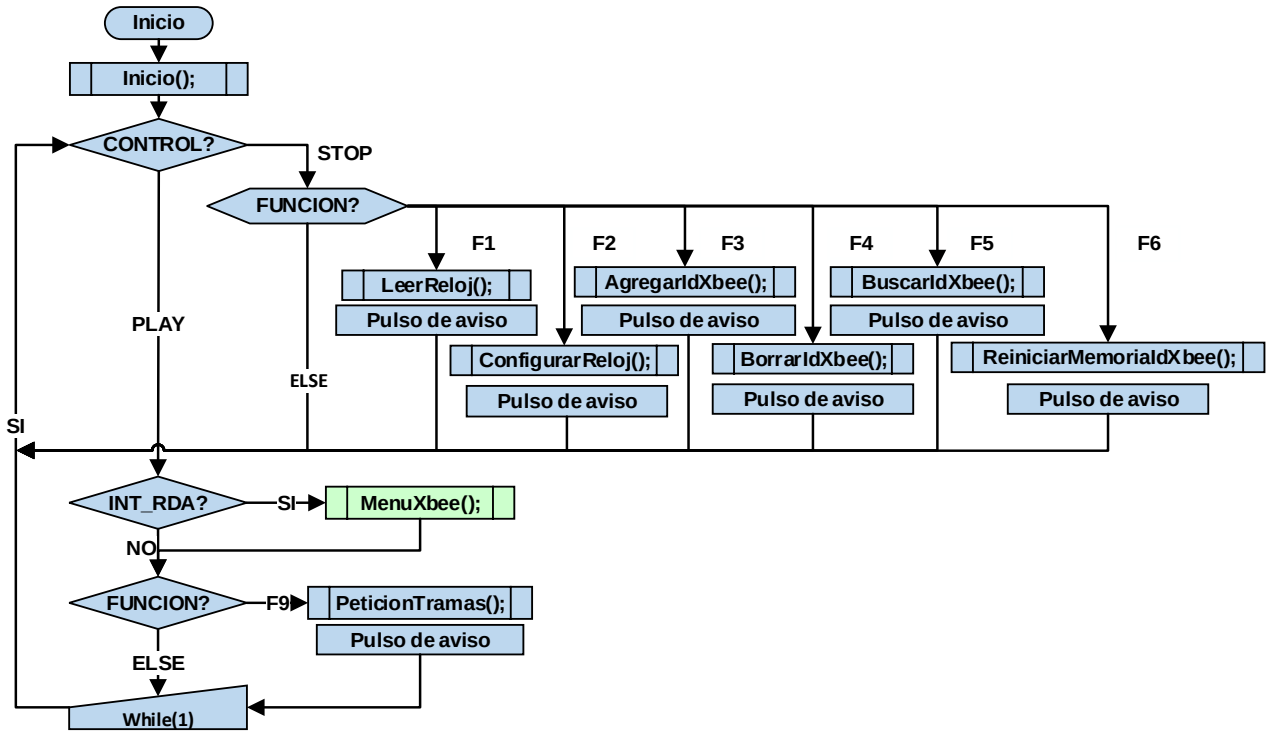

Figura 3.10 Diagrama de flujo MCU B.

[24] Código C 2 Pág. 93

En el *setup* se colocan las librerías de control que utiliza el MCU B para manejar los dispositivos del reloj de tiempo real, memoria auxiliar, banco de memoria y módulo XBee. Estas son:

- RTC_DS1307.C
- MEMORIA_AUXILIAR_24FC1025.C
- BANCO_MEMORIA_24CF1025.C
- XBEE_S2.C

Estos dichos código se conectan usando las etiquetas de la siguiente tabla (Tabla 3.11), para la configuración de cada puerto.

dispositivo	**Modo**	**SDA**	**SCL**	**configuración**	**Etiqueta**
MCU A	Slave (0xC0)	RB0	RB1	hardware	I2C_CTRL
Reloj (RTC)	master	RE0	RE1	software	I2C_RTC
Memoria auxiliar	master	RD2	RD3	software	I2C_MEM_AUX
M1	multi_master	RB6	RB7	software	I2C_BANCO_M1
M2	multi_master	RB4	RB5	software	I2C_BANCO_M2
M3	multi_master	RD6	RD7	software	I2C_BANCO_M3
M4	multi_master	RD4	RD5	software	I2C_BANCO_M4

Tabla 3.11 Configuración de puertos I^2C.

El lazo infinito es el ciclo principal y se rige por el estado del registro CONTROL con ayuda de dos comandos:

- PLAY: En este modo el subsistema B está listo para recibir la orden del MCU A para iniciar un ciclo de petición de tramas de mediciones.
- STOP: En este estado el subsistema B está pausado y se pone a disposición de las órdenes del MCU B. el cual puede solicitar algunas tareas enviando un comando (F1, F2, F3, F4, F5, F6) al registro de FUNCION, al finalizar su labor el MCU B envía un pulso de aviso al microcontrolador. Las funciones que puede realizar el subsistema B en modo pausado son:
    - LeerReloj(): Recupera la fecha y hora del reloj y verifica que los valores no sean nulos (0xFF).
    - ConfigurarReloj(): Escribe los vales de tiempo en el reloj, verifica la acción con la función de lectura y retorna los nuevos valores del reloj.

- AgregarXbee(): Busca una ID en la red y en lista, si el módulo está activo en la red y la ID no existe el lista la agrega como una dirección nueva.
- borrarXbee(): Busca una ID la lista, si existe la borra.
- BuscarIdXbee(): Busca una ID en la red y lista, retorna una respuesta para la interfaz del usuario.
- ReiniciarMemoriaIdXbee(): Reinicia la lista de IDs, borrando los apuntadores del estatus de lista.
- Backup(): Reinicia el contador de tramas, lo cual ocurre después de que el sistema A ya ha respaldado la información que contenía el banco de memoria.

3.2.4.1 Rutina del modo automático.

Esta rutina comienza cuando el subsistema B se encuentra en estado de PLAY y el MCU A envía el comando de inicio para un ciclo de petición (F9), entonces el MCU B usa la función PeticionTramas(), la cual se encarga de recaudar la información de todos los medidores en la red. Primero lee una dirección de medidor de la lista de IDs, después con la función Conexion() establece la comunicación con el dispositivo en la red, enviando el comando de petición, a lo cual el medidor responde mandando los valores descriptivos del consumo energético que previamente ya obtuvo. Dicha información la recibe el MCU B y la complementa con la ID remitente y el tiempo, después, usando la función GuardarTrama() almacena la trama en un arreglo provisional, en este arreglo la información aguarda hasta reunir cuatro tramas, entonces se almacena todo el arreglo en una sola secuencia de escritura en el banco de memoria, después de esto el MCU B actualiza el display de leds y valor del número total de tramas almacenas, tanto en los registros de buffer como también en la memoria auxiliar (figura 3.11).

La rutina se repite desde la lectura de la dirección de medidor hasta el fin, y el ciclo termina hasta que se haya recorrido toda la lista de IDs. Al terminar la recaudación el MCU B envía un pulso de aviso al MCU A, que al recibirlo reinicia el conteo de los minutos para la siguiente ciclo de petición.

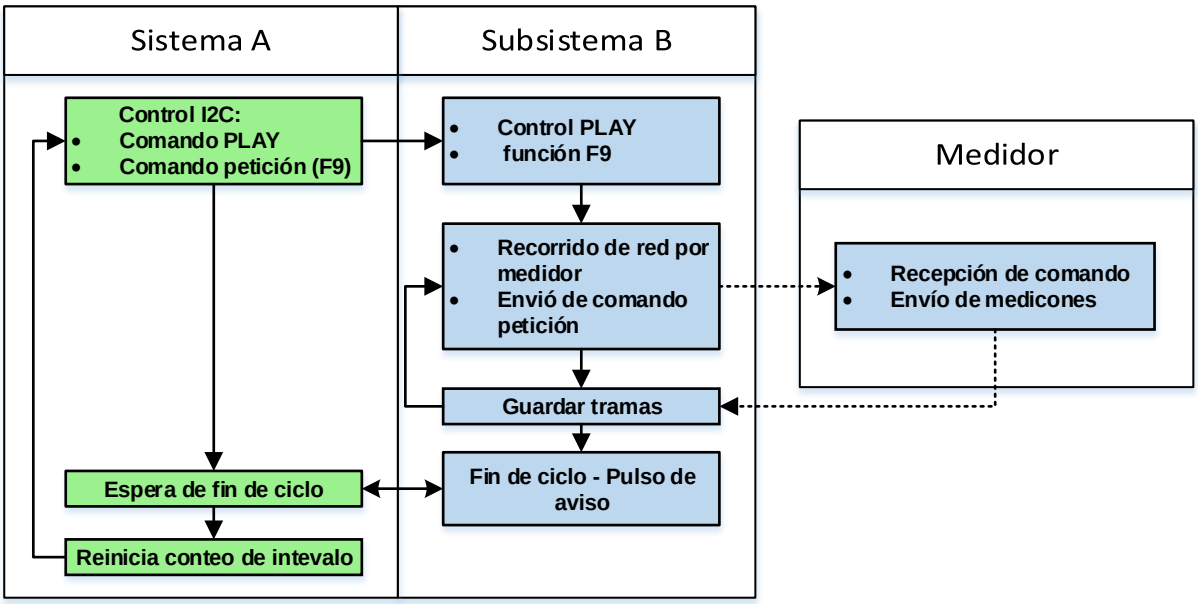

Figura 3.11 Rutina de modo automático (recaudación).

Capítulo 4 Interfaz de Usuario

Para crear la interfaz de usuario se empleó el entorno grafico de programación LabVIEW, el cual comúnmente se utiliza para crear sistemas virtuales de control, instrumentación, comunicaciones etc. El ambiente de LabVIEW se conforma de dos partes, el panel frontal y el panel de programación. En el panel frontal se colocan los elementos visibles que conforman una interfaz de usuario como por ejemplo botones, interruptores, ventanas etc. cada uno de estos elementos genera un bloque de control en el panel de programación. Los bloques de control trabajan interactuando por medio de conexiones o hilos donde el sentido de la función obedece el orden secuencial en el que se han conectados [25].

Cuando el concentrador se conecta al puerto USB de la computadora, esta reconoce la conexión como un puerto virtual serial COMx. Por medio de esta unión la interfaz y el concentrador intercambian información usando paquetes seriales en formato API[26]. Así el usuario tiene acceso a la configuración de paramentos del dispositivo y puede realizar el respaldo de mediciones según se requería. La interfaz de usuario mostrada en la Figura 4.1 se observan cinco elementos:

- 2 botones para controlar la interfaz:
  - **Stop:** se usa para detener todo proceso en cualquier momento.
  - **Lock:** bloquea todos los demás botones del panel.
- 2 casillas para configurar el puerto serial entre el concentrador y la computadora:
  - **PUERTO COM USB (I/O**): casilla de entra donde se selecciona el puerto COMx asignado al concentrador por la computadora.
  - **Estado de Conexión:** este elemento muestra el estado de la conexión y lo actualiza cada vez que se ha utilizado dicho enlace para realizar un envió o recepción de paquetes.

[25] Cita (José Rafael Lajara Vizcaíno, 2011)
[26] Figura B.10 Diagramas de bloques para envío y recepción de paquetes API. Pág. 84

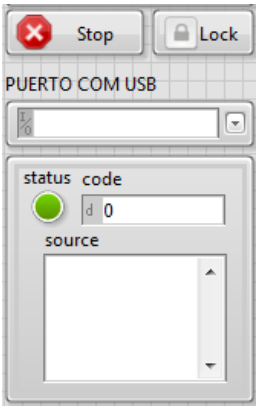

Figura 4.1 ventana de control de puerto COM.

- Las ventana de funciones de control y configuración, esta se conforma de tres pestañas:
  - Control de tiempo.
  - Control de mediciones.
  - Control de IDs.

Las cuales contienen botones que funciona para realizar tareas en específico y toda forman parte de un mismo diagrama de bloques en el panel de programación [27].

4.1 Pestaña de control de tiempo

Esta pestaña se conforma de dos conjuntos de elementos como se observa en la Figura 4.2, donde el primero grupo enmarcado en la parte superior de la ventana se constituye de dos botones, un interruptor y dos casillas de salida tipo reloj/calendario, estos elementos sirven para leer o configurar el reloj interno del concentrador y sus funciones se describen a continuación:

- **Leer:** al oprimir este botón la interfaz envía un comando API al concentrador, el cual retorna un paquete de valores que obtuvo de la lectura del reloj RTC, estos datos se encuentran en formato BCD por lo cual la interfaz los castea a formato decimal[28] antes de colocarlos en la casilla reloj/calendario correspondiente al concentrador.

[27] Figura B.6 Diagrama de bloques general de la interfaz. Pág. 81
[28] Figura B.7 Diagrama de bloques de casteo de tiempo decimal a BCD. Pág. 82

Configurar: al oprimir este botón la interfaz toma los valores de tiempo y de fecha de la computadora y los cambia del formato DCD al decimal[29] para mandarlos en un paquete-API al concentrador. Finalmente la interfaz coloca los nuevos valores de tiempo en la casilla correspondiente.

- **Personalizado?:** es un interruptor que hablita la configuración del reloj con valores de tiempo personalizados dados por el usuario en la casilla de entrada tipo reloj/calendario, después para hacer efectivo este cambio se oprime el botón *configurar*.

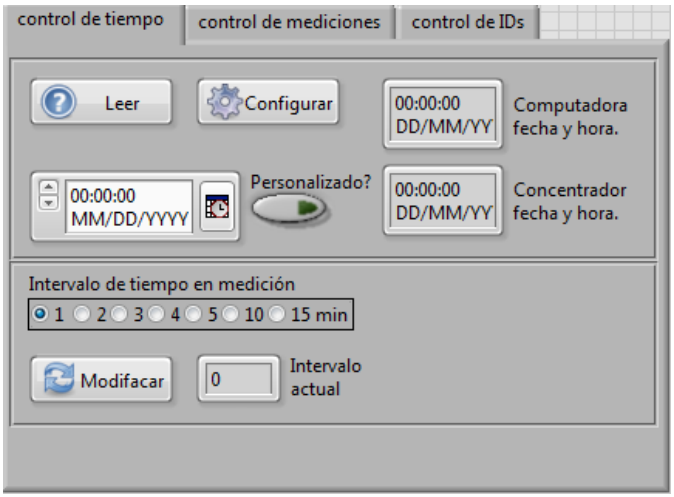

Figura 4.2 pestaña de control de tiempo.

Los elementos del recuadro inferior de la Figura 4.2 se usan para definir el intervalo de petición. De inicio el concentrador está programado para recabar información cada de 15 minutos, pero si el usuario desea modificar este rango, primero se selección una de las opciones de intervalo (1-15 minutos) y después se oprime el botón **Modificar**, así la interfaz envía el dato al concentrador y este realizar el cambio de la variable general de RANGO, por último la interfaz coloca el nuevo valor en la casilla 'intervalo actual'.

[29] Figura B.8 Diagrama de bloques de casteo de tiempo de BCD a decimal. Pág. 82

4.2 Pestaña de control de mediciones.

Esta pestaña controla todo lo relacionado con el respaldo de las tramas de mediciones. Como se observa en la Figura 4.3 la pestaña se conforma de dos grupos de elementos, uno es el conjunto enmarcado en el recuadro superior, el cual se usa para definir la carpeta de destino, nombre y tipo de archivo, donde cada botón aporta un parámetro:

- **Path de Carpeta:** se usa para escribir o seleccionar la carpeta de destino del archivo.
- **Box tipo archivo:** sirve para seccionar uno de los dos tipos de extensión txt (texto) o xls (Excel).
- **Menú de nombre**: el nombre del archivo incluye la fecha y hora en la que se realiza el respaldo, por lo cual en este botón se selecciona una de tres opciones posibles:
    - Corto: ejemplo - 29-01-14 Hr 16-15
    - Largo: ejemplo - 29 de enero del 2014 Hr 16-15
    - Abreviado: ejemplo - 29 de ene de 2014 Hr 16-15

El segundo recuadro de la Figura 4.3 es dedicado a los elementos para realizar el respaldo de mediciones. Antes de hacer el reporte de la información se debe de definir la carpeta de destino para archivo, si no la interfaz no puede ejecutar la acción.

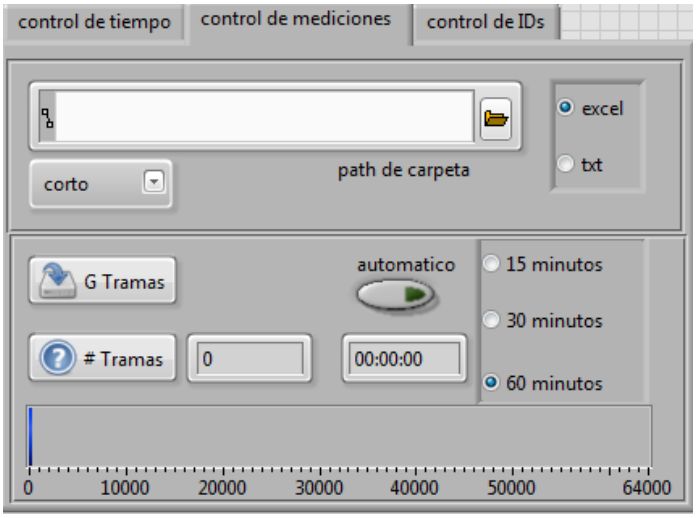

Figura 4.3 Pestaña de control de mediciones.

El botón **#Tramas** se usa para consultar al concentrador el número de mediciones que hasta ese momento han sido guardadas en el banco de memoria interna. Para esto la interfaz manda un comando-API al dispositivo, el cual responde con la cifra actual deseada, este dato se visualiza gráfica y numéricamente en el tanque de tramas, ubicado en la parte inferior de la pestaña (Figura 4.3).

Para realizar el respaldo de las tramas de medición existen dos formas, la primera de manera instantánea cuando el usuario oprime el botón **G Tramas** y la segunda cuando se habilita el modo automático oprimiendo el interruptor del mismo nombre, así la función de respaldo se sincroniza con el reloj de la computadora y se efectúa al cruce de los minutos 15, 30 o 60 de cada hora [30]. La casilla tipo reloj bajo el interruptor muestra una cuenta regresiva en minutos y segundos para el siguiente cruce de respaldo automático.

Ambos modos utilizan el mismo diagrama de bloques para crear el archivo final, en el cual la interfaz inicia una comunicación de manera cíclica con el concentrador. Por cada ciclo de conexión el concentrador envía cuatro tramas a la computadora, las cuales la interfaz va copiando en el archivo del reporte. Cuando el total de información ha sido respaldada, la interfaz cierra el ciclo de comunicación y guarda el archivo. Por su parte el concentrador retoma su labor de recopilación, iniciado desde un nuevo conteo de tramas.

4.3 Pestaña de control de IDs.

La última pestaña de la ventana de control mostrada en la Figura 4.4 se encarga del manejo de los medidores en red, para esto se emplean cuatro botones: **Agregar, Borrar, Buscar** y **Reiniciar**. Los tres primeros efectúan dichas acciones usando la dirección en hexadecimal única de cada módulo, la cual se coloca previamente en la casilla *ID.* Dicha dirección es enviada al concentrador y este se encarga de buscar la ID en la red y la lista de medidores, después de ejercer la acción deseada por el usuario, retorna una respuestas a la

[30] Figura B.9 Diagrama de bloques para back-up automático. Pág. 83

interfaz, finalmente esta despliega una ventana emergente con un mensaje que indica al usuario el resultado de la función.

El botón **Reiniciar** sirve para borra la lista de IDs en el concentrador. Dado que al oprimir este botón se pude perder información importante sin querer, antes de efectuar la acción, la interfaz despliega una ventana emergente de *warning,* donde se explica las consecuencias de borrar la lista y para que la operación continúe se requiere que el usuario confirme la acción o en caso contrario decline la decisión.

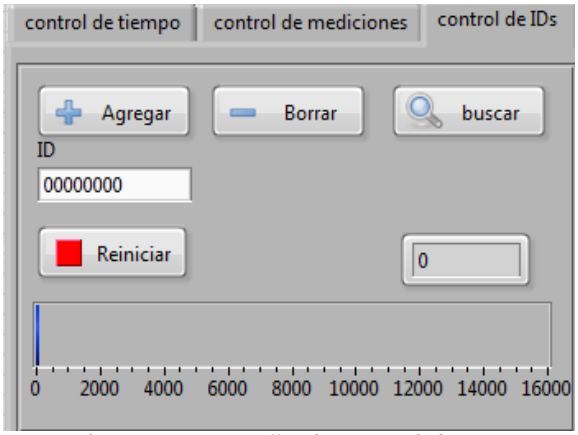

Figura 4.4 Pestaña de control de IDs.

Todos los botones de esta pestaña (Figura 4.4) incluyen la función para actualizar el número de medidores en lista. Este dato se muestra gráfica y numéricamente en el tanque y la casilla, ubicados en la parte inferior de la pestaña.

Capítulo 5 Pruebas

La prueba consistió en poner a funcionar al concentrador junto a un dispositivo medidor, recabando información cada minuto durante un día y medio. La carga del medidor fue un foco incandescente de 75 Watts.

El diseño del dispositivo medidor es de la autoría del M.I. José Castillo (Figura 5.1), dentro de su diseño se incluye la base para montar un módulo Xbee, que junto con la librería de control del mismo, otorgan al dispositivo la capacidad para relaciónese con el concentrador y demás medidores. La programación del medidor toma en cuenta los puntos siguientes.

- incluir en la cabecera del código la librería XBEE_S2.C.
- configuración del puerto serial USART:
    - Baud: 57600
    - Parity: N
    - Bits: 8
    - bit Stop: 1
- Habilitar la interrupción INT_RDA del puerto USART y ligar a esta la función IntXbee ()[31].
- anexar al código las definiciones de red:
    - NETWORK 0x3032
    - COORDINADOR 0x4091D11
- incluir las funciones en sectores específicos:
    - Inicio_Xbee (): se incorpora al setup del programa y son de las configuraciones iniciales para manejar y sincronizar el módulo Xbee con el concentrador.
    - Menu_Xbee (): se incluye en el siclo infinito del programa y se encarga de interpretar los comandos enviados por concentrador.

[31] Función IntXBee(). Pág. 35

- Medicion_Xbee (): es una función adicional y se encarga de enviar las mediciones realizadas.[32]

Figura 5.1 Medidor inalámbrico.

Para el desarrollo de la prueba se conectó el concentrador en un punto apartado del medidor (aproximadamente 15 metros el línea de visión directa). Ya energizado se verifica que haya inicializado correctamente y se encuentre funcionando con normalidad, esto se puede observar en su display (Figura 5.2), donde en la izquierda se observa el led de estatus enciende en color verde y en la parte izquierda donde el led de energía está activo al igual que el del USB.

[32] Código C 7 Pág. 109

Figura 5.2 dispositivo concentrador.

Para configurar el concentrador primero se conecta la terminal USB al puerto de la computadora, tal como se observa en la Figura 5.3, la casilla PUERTO COM USB de la interfaz se selecciona el puerto asociado al centrador (en este caso COM7). Con la conexión reconocida por la interfaz ahora se pueden manejar las funciones de la ventana de control. Lo principal es configurar los parámetros de la pestaña de tiempo, sincronizar el reloj del concentrador con la computadora y definir el intervalo de mediciones (un minuto). Después de configurar el concentrador se puede desconectar de la computadora y dejar que continúe con su labor de manera autónoma.

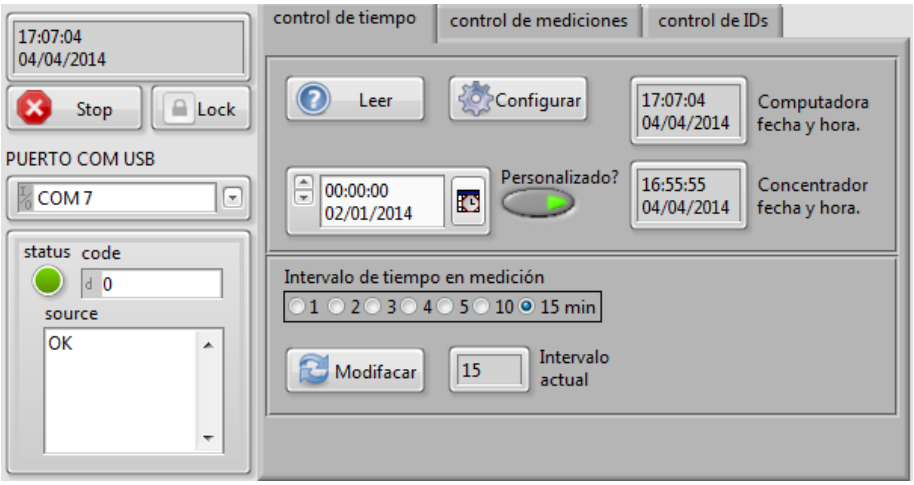

Figura 5.3 Interfaz de control de tiempo.

Una vez que transcurrió el día y medio de esta prueba, nuevamente se conectó el concentrador a la computadora, ahora para realizar el respaldo de información, para estos en la pestaña de Control (Figura 5.4) se define la carpeta en donde se guardará el reporte (C:\Users\axel\Desktop\Mediciones), el tipo (Excel) y nombre con los que se manejara el archivo. Finalmente los datos obtenidos se muestran en la sección de resultados.

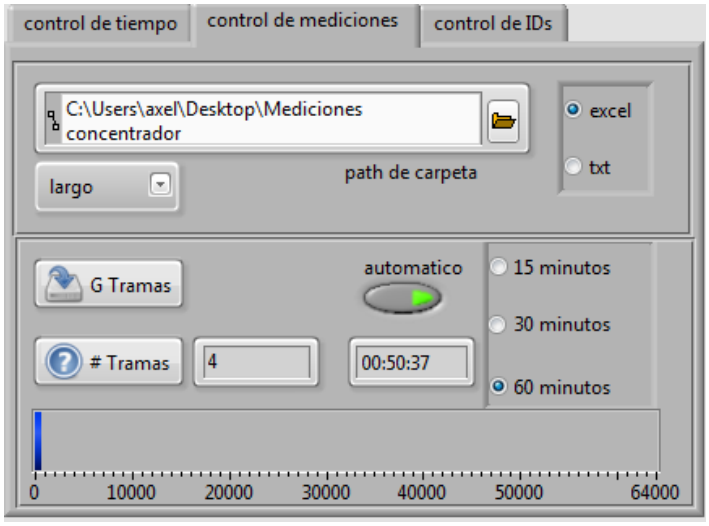

Figura 5.4. Panel de control de mediciones.

Capítulo 6 Resultados

El archivo de Excel creado por el concentrador del **06 de abril de 2014Hr 14-0** es el reporte de 1440 tramas de datos, de las cuales se tomó el intervalo significativo del día 5 de abril (0 - 23 horas), con estos valores se obtuvieron las siguientes graficas de Voltaje RMS, corriente RMS (Figura 6.1), Potencia y energía (Figura 6.2).

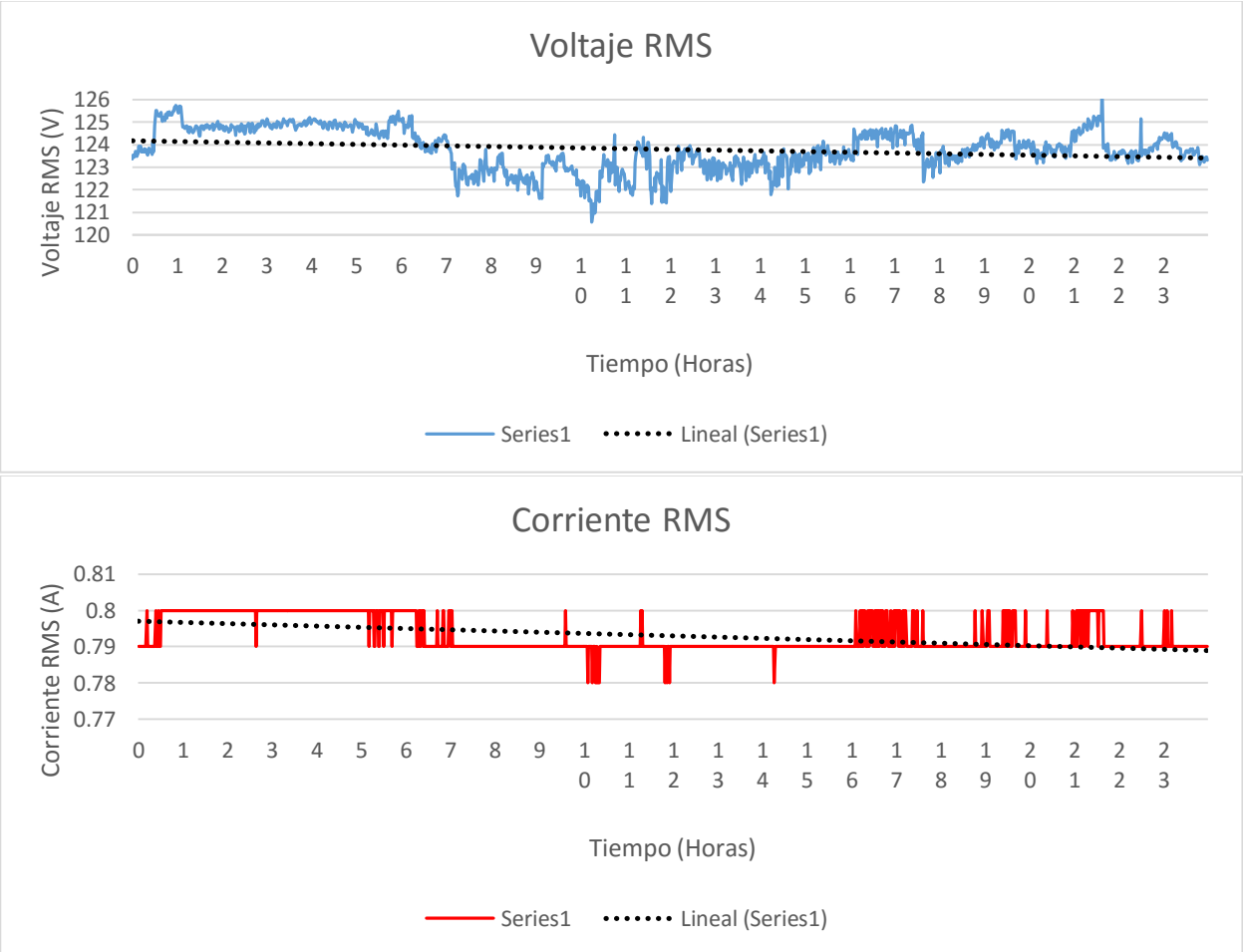

Figura 6.1 graficas de voltaje y corriente RMS.

Los valores mostrados son discretos y se puede observar los cambios de consumos de la carga. En la gráfica de corriente de la Figura 6.1 los cambios son mínimos pero coinciden con los incrementos de voltaje.

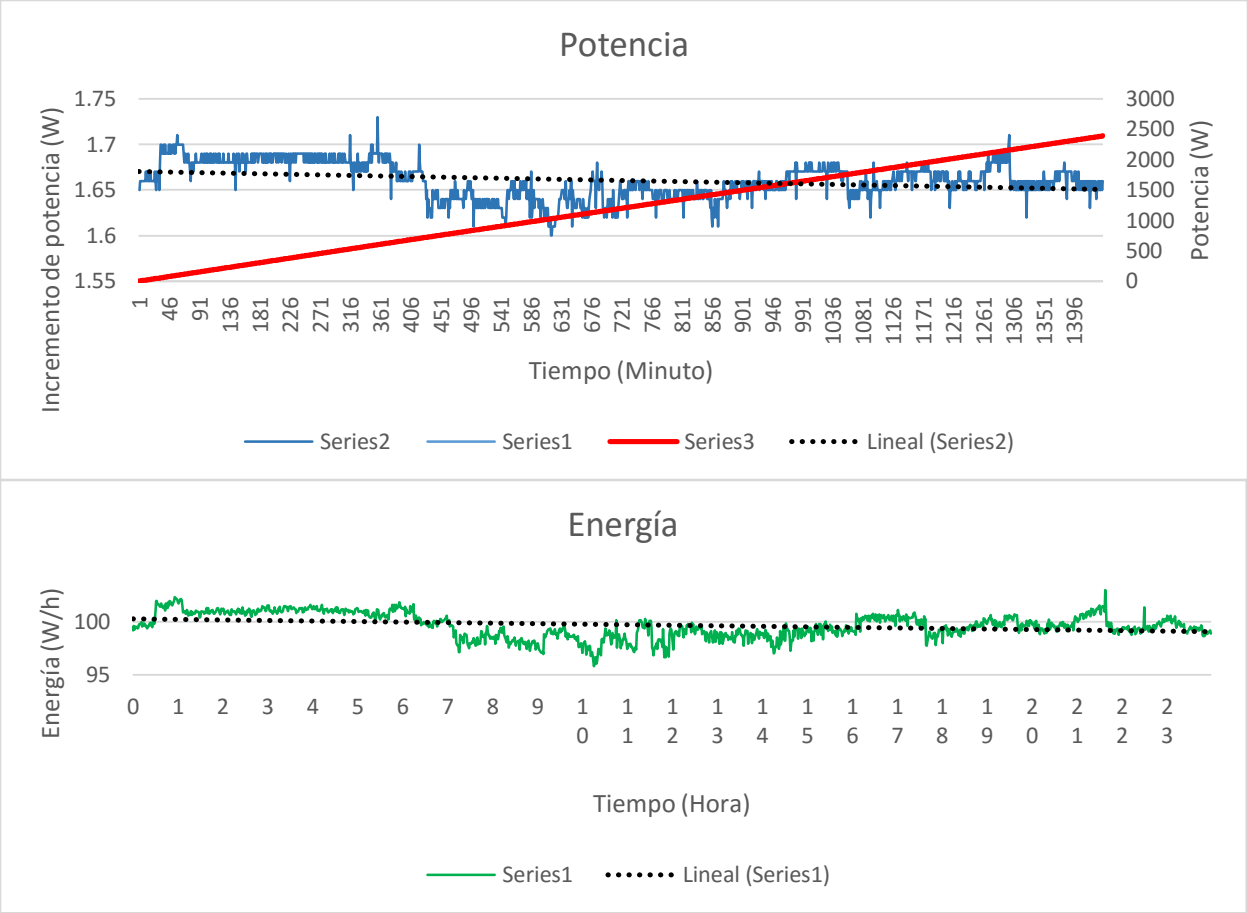

Figura 6.2 graficas de potencia y energía.

Es importante mencionar que estos datos solo son de carácter demostrativos para este trabajo. La verificación de los mismos es tema para un trabajo posterior y enfocado a este objetivo en particular.

Conclusiones

Se diseñó, desarrolló y construyó un dispositivo concentrador inalámbrico, totalmente portable (10 x 15 x 3 cm y 500 g. aproximadamente) que fue utilizado como la parte central de una red de monitoreo de consumo energético en edificios.

El concentrador fue diseñado con la finalidad de que funcionara como un dispositivo autónomo para evitar la supervisión humana permanente y/o la dependencia con un sistema secundario (computadora). El concentrador por si solo puede reunir las mediciones y almacenarlas en su banco de memoria, además es capaz de agregar automáticamente nuevos medidores a la red con solo encender estos, lo que permite crear una red flexible que soporta hasta 16,000 elementos ubicados a distancias de hasta 30 metros entre muros o piso y hasta 100 metros en visión directa en exteriores.

El concentrador y los medidores se comunican inalámbricamente por medio de la señal del módulo XBee cuya gran ventaja para el concentrador es que se propaga a través de la banda de 2.4 MHz, la cual es libre y no requiere de la compra de licencias especiales para su manejo, esto convierte al concentrador en una opción económicamente atractiva. Además el protocolo ZigBee garantiza la no interferencia con otras señales de telecomunicaciones como lo son Wi-Fi, Bluetooth y de telefonía celular. Adicionalmente, la caja de aluminio que contiene la placa PCB brinda al sistema un blindaje contra interferencias electromagnéticas del entorno.

El concentrador está diseñado para almacenar hasta 64,000 paquetes de mediciones. Los chips de memoria tienen una capacidad de retención de hasta 200 años y un millón de ciclos de escritura, lo cual indica que los chips tienen un ciclo de vida útil de 100 años promedio. Gracias a que los chips son de memoria no volátil, la información está protegida contra apagones y cortes de suministro de energía inesperados.

En la programación, el establecer el uso de las librerías para controlar cada dispositivo electrónico, hizo eficiente la comunicación interna entre dispositivos lo que permitió mantener un orden de programación que facilito el diseño de los algoritmos generales de funcionamiento para el sistema A y subsistema B.

La conexión entre el concentrador y la interfaz es por medio de un puerto USB de la computadora, siendo esta característica una gran ventaja debido a que es el tipo de puerto más común en cualquier tipo de sistema informático. Con respecto a la interfaz esta es sencilla y totalmente trasparente para el usuario, no requiere más que solo una breve explicación y hasta de cierto modo es intuitiva. Sus funciones son configurar parámetros como el tiempo de resolución, actualización de fecha y hora, entre otros. Además permite hacer el respaldo en cualquier instante de tiempo o usando un “modo automático”. Finalmente, la interfaz concluye sus funciones generando un reporte con las mediciones reunidas por el concentrador en un archivo de Excel. Como propuesta para una futura revisión de la interfaz, se podría incorporar otras tareas como por ejemplo; una función para respaldar la lista de medidores o también un modo de comunicación en tiempo real con un medidor en específico desde la interfaz misma.

Si bien el concentrador se diseñó para desempeñarse dentro del proyecto SIGEEN, su función podría expandirse para otro tipo de aplicaciones, como por ejemplo en sistemas de monitoreo de procesos industriales, de instrumentación en plantas y hasta en sistemas de control a distancia. Todo esto es posible gracias a la arquitectura y diseño del concentrador, el cual forma junto con los medidores redes de topologías en malla extensibles, que permiten enviar y recibir cualquier tipo de información de manera inalámbrica. Al final los datos que se manejen dependerán del dispositivo de medición, adquisición o instrumentación, el cual cense las variables físicas como por ejemplo temperatura, presión, flujo o cualquier otra variable deseada.

Anexo A

A.1 Registros DS1307

Los registros se localizan entre las direcciones 0x00 y 0x06 segundos, minutos, horas, día, día del mes, mes y año tal como lo muestra la siguiente tabla. Como se había mencionado anteriormente dichos valores tiene el formato *BCD* (Binary-Coded Decimal) esto es que por ejemplo: 10 segundo, 30 minutos y 2 horas serán escritos y leídos como; 0x10, 0x30 y 0x02 en sus respectivos registros.

El bit 7 del registro 0x00 (CH), es el bit de paro ‘0’ y marcha del reloj ‘1’. El bit 6 del registro 0x02, configura el formato de la horas ‘0’ para 24 horas, ‘1’ para 12 horas. En este último formato el reloj nos indicara AM/PM en el bit 6 del mismo registro AM ‘0’ o PM ‘1’.

registro	**dato**	**rango**	**Bit 7**	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
0x00	segundos	00-59	CH	10 segundos			Segundos			
0x01	minutos	00-59	x	10 minutos			minutos			
0x02	horas	00-24 00-12	x	24 12	20 hrs. AM/PM	10 horas	horas			
0x03	día	01-07	x	x	x	x	x	día		
0x04	día del mes	01-31	x	x	10 día del mes		Día del mes			
0x05	mes	01-12	x	x	x	10 mes	mes			
0x06	Año	00-99	10 años				años			
0x07	control	-	OUT	x	x	SQWE	x	x	RS1	RS0
0x08- 0x3F	RAM 56x8	00-FF								

Tabla A.1 Registros RTC.

El registro 0x07 es el control para el pin de onda cuadrada, el bit 7 (OUT) fija valor en esta salida ya sea ‘0’ o ‘1’ lógico, mientas que el bit 4 (SQWE) activa el oscilador con un ‘1’. Por último los bits 1 y 0 (RS1 y RS0) configuran la frecuencia del oscilador como lo muestra la tabla siguiente

SQWE	OUT	RS1	RS0	SQW/OUT
1	X	0	0	1Hz
1	X	0	1	4.096kHz
1	X	1	0	8.192kHz
1	X	1	1	32.768kHz
0	0	x	x	0
0	1	x	x	1

Tabla A.2 Control de la salida onda cuadrada.

A.2 Protocolo API.

El *API* es un tipo de secuencia serial para el envío de un paquete de información en bytes.

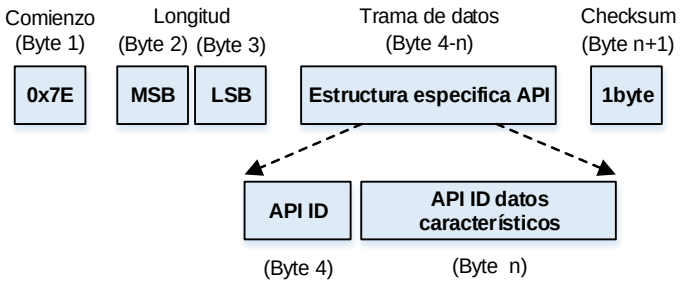

Figura A.1 Secuencia serial API.

- El primer byte indica el comienzo del mensaje, todo valor antes de este byte será descartado, normalmente su valor es 0x7E.
- Los bytes 2 y 3 conforman un número entero de 16bits que indica la longitud del mensaje a partir del byte 4 hasta *'n'* elementos.
- La trama es la parte principal del API, es donde se coloca la información, esta inicia del byte 4 hasta 'n' siendo este el último byte de información. Para tener orden en la trama normalmente esta se conforma por dos partes.
    1. la identidad de la *API (API ID)* que indica la naturaleza de la trama o mensaje.
    2. Los datos que conforman en sí el mensaje.
- Por último se requiere de un byte de comprobación, cuyo valor es el byte menos significativo de la suma de los valores de los bytes que conforman la trama restados a 0xFF. De no se correcto el checksum el paquete es invalido.

$$checksum = 0xFF - \sum_{i=4}^{n} Trama[i] \quad \text{(Ecuación A.1)}$$

A.3 Bus I²C

Se denomina “bus I²C” a un bus bidireccional de dos hilos para conectar distintos circuitos integrados. Las características principales son:

- Se emplean dos líneas, la de datos-dirección (SDA) y la de reloj (SCL), conectadas a tensión positiva mediante resistencias de *pull-up.* Los pines de salida que los dispositivos conecten al *bus* deben ser del tipo *open drain*.
- Cada dispositivo conectado al bus tiene un código seleccionable mediante configuración.
- La estructura de comunicación es de tipo *maestro - esclavo.* Habitualmente un dispositivo *maestro* (generalmente un microcontrolador) y varios *esclavos.* También pueden existir varios dispositivos *maestros* mediante un sistema de detección de colisiones (multímaestro).
- El número de interfaces conectados al *bus* únicamente depende de la capacidad límite asignada al *bus* (400pf).
- El dispositivo *maestro* es el generador del reloj, establece un pulso de reloj por cada bit transferido.
- Los datos y las direcciones se transmiten como palabras de 8 bits.

A.3.1 Secuencias de escritura y lectura.

Utilizando las condiciones propias del protocolo I²C. La comunicación comienza con *START* por parte del *maestro*, seguido de la dirección del esclavo de 7 bits junto a un bit (*R/W*) en modo escritura (0). El esclavo responde enviando un acuse de recepción (*ACK)*, hecho lo anterior el *maestro* transmite una dirección de registro, estableciendo así un puntero de registro que se incrementa automáticamente después de cada byte escrito exitosamente. Para finalizar la comunicación el *maestro* genera un estado de *STOP*.

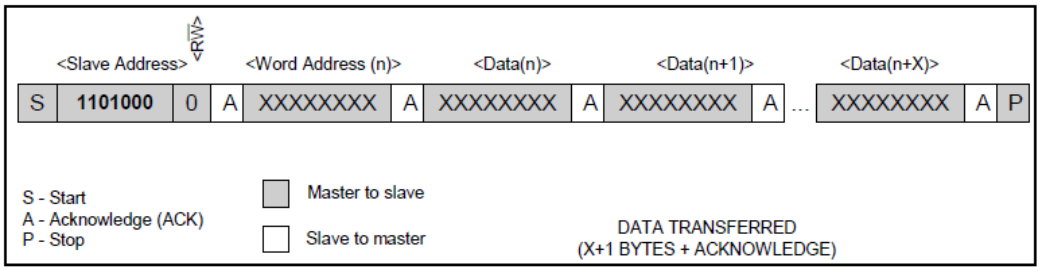

Figura A.2 Secuencia de modo escritura I2C

El modo de lectura repite los primeros pasos del anterior, sin embargo el bit de dirección *R/W* cambia a 1. Después del acuse de envió (*ACK)* del *esclavo,* el *maestro* envía la dirección del puntero a registro y comienza la lectura, el puntero se incrementa automáticamente después de leer un byte. Para finalizar la comunicación el esclavo tiene que recibir un NO acuse de recepción (NACK) seguido de la condición de STOP por parte del *maestro*.

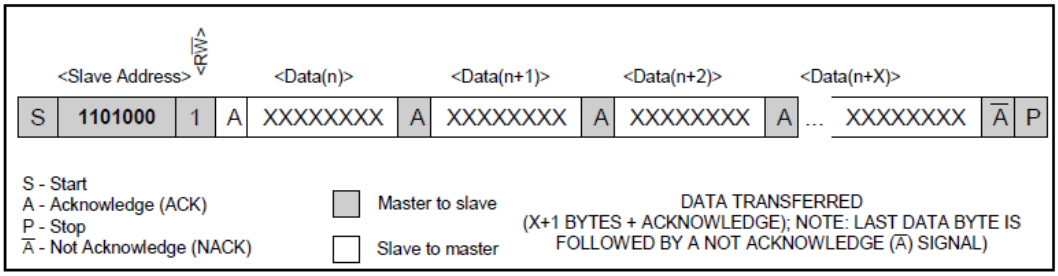

Figura A.3 Secuencia de modo lectura I2C.

A.4 Efectividad de los blindajes

La efectividad de un blindaje puede especificarse en términos de atenuación en dB la intensidad del campo. Así la efectividad S está definida para campos eléctricos por:

$$S = 20 \log \left(\frac{E_0}{E_1} \right) [dB] \qquad \text{(Ecuación A.2)}$$

Y para campos magnéticos como:

$$S = 20 \log \left(\frac{H_0}{H_1} \right) [dB] \qquad \text{(Ecuación A.3)}$$

Donde:

- E_0, H_0: son las intensidades del campo incidente.
- E_1, H_1: la intensidad de campo que trapazas el blindaje.

La intensidad del blindaje varía con la frecuencia. La geometría del campo y la posición desde donde el campo es medido.

Al inducir una onda electromagnética en una superficie metálica existen dos efectos. La onda es parcialmente reflejada por la superficie, y la parte trasmitida (no reflejada) que es atenuada al pasar a través del blindaje, lo que es mejor conocido como perdidas por absorción y son las mismas para un campo cercano o lejano. La energía trasmitida puede asimismo reflejarse en la superficie del blindaje contraria y volverse a reflejar múltiples veces en las dos superficies lo que se conoce como efecto de múltiples reflexiones.

Material	*Frecuencia (kHz)*	*Pérd. de absorción (todos los campos)*	*Pérdidas de reflexión*		
			campos magnéticos	*campos eléctricos*	*ondas planas*
magnético $\mu_r = 1000$ $\sigma_r = 0{,}1$	<1	*A–B*	*A*	*E*	*E*
	1–10	*C–D*	*A–B*	*E*	*E*
	10–100	*E*	*B*	*E*	*D*
	>100	*E*	*B–C*	*D*	*C–D*
no magnético $\mu_r = 1$ $\sigma_r = 1$	<1	*A*	*B*	*E*	*E*
	1–10	*A*	*C*	*E*	*E*
	10–100	*B*	*C*	*E*	*E*
	>100	*C–D*	*D*	*E*	*E*

EFECTIVIDAD DE LOS BLINDAJES

	Atenuación (dB)	*Característica*	
A	0–10 dB	muy inefectivo	(muy malo)
B	10–30 dB	inefectivo	(malo)
C	30–60 dB	medio	(normal)
D	60–90 dB	efectivo	(bueno)
E	>90 dB	muy efectivo	(excelente)

Figura A.4 Resumen de características de efectividad de los blindajes, sin tener en cuenta las discontinuidades debidas a ranuras o juntas.

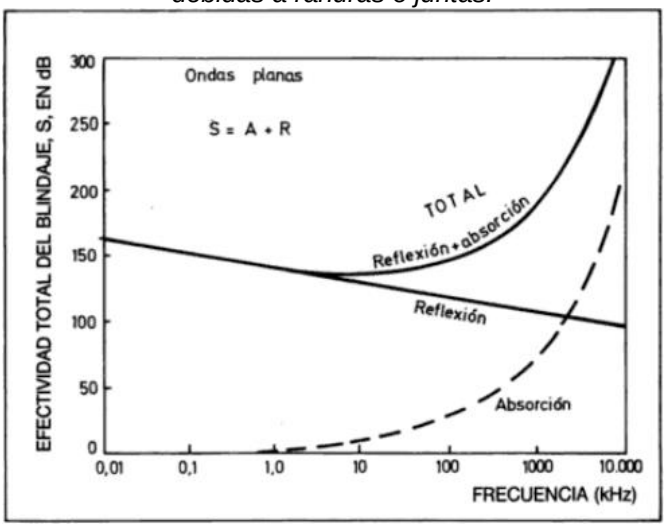

Figura A.5 Efectividad de un blindaje de metal no magnético de 0.5mm de espesor.

Anexo B

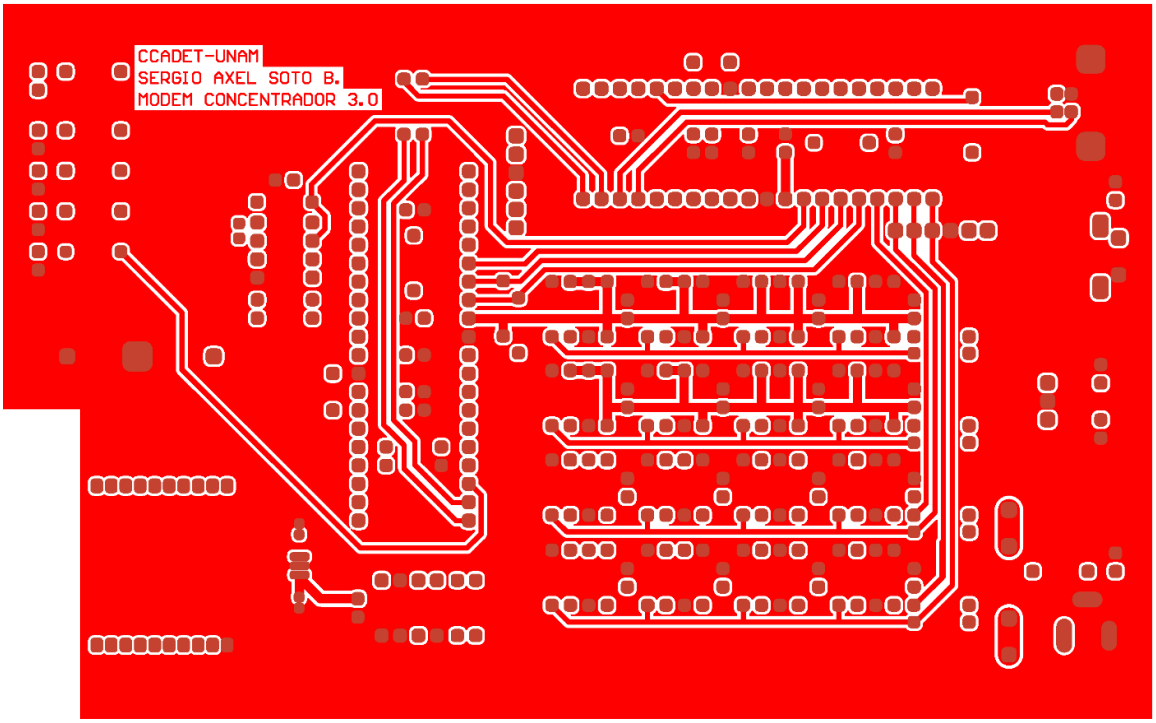

Figura B.1 Top layer concentrador 3.0.

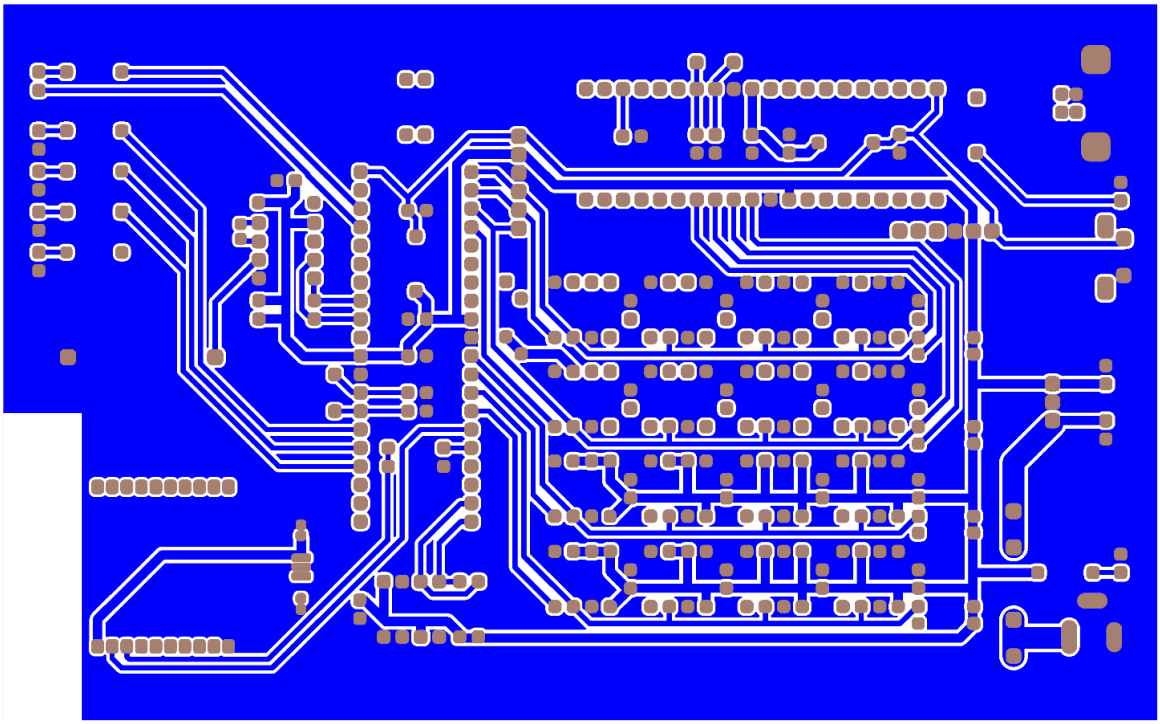

Figura B.2 Bottom layer concentrador 3.0.

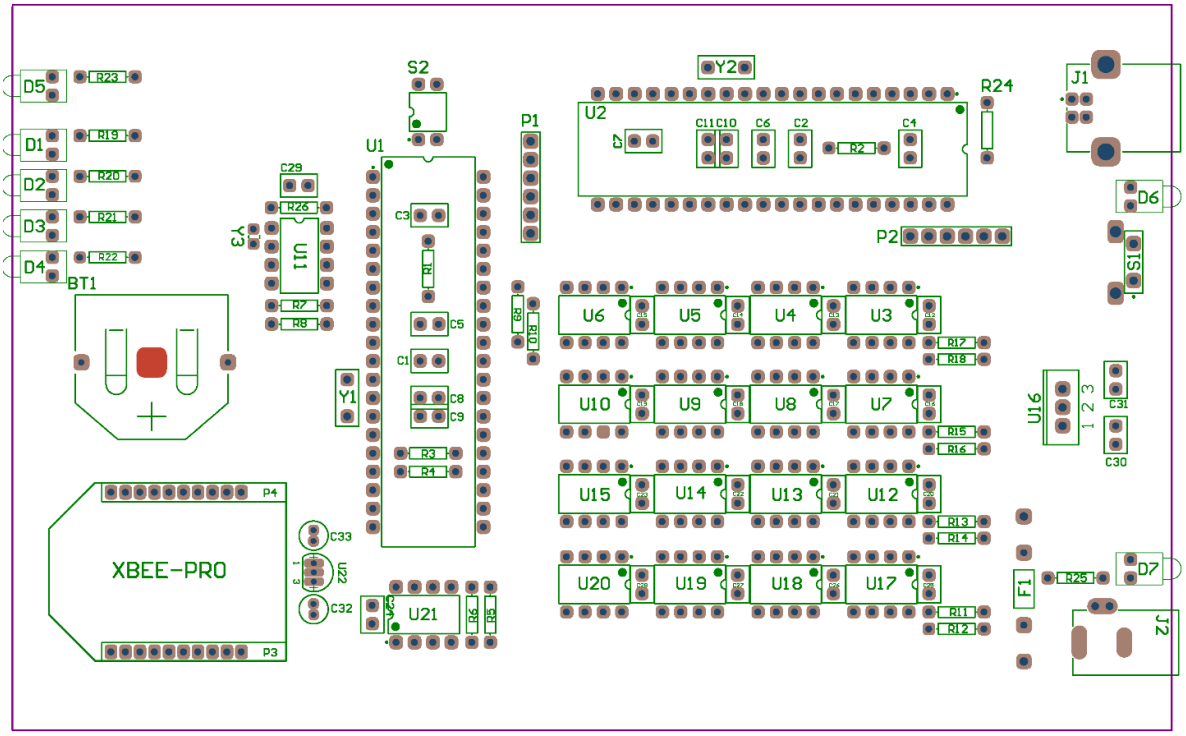

Figura B.3 top Overlay y Drill concentrador 3.0.

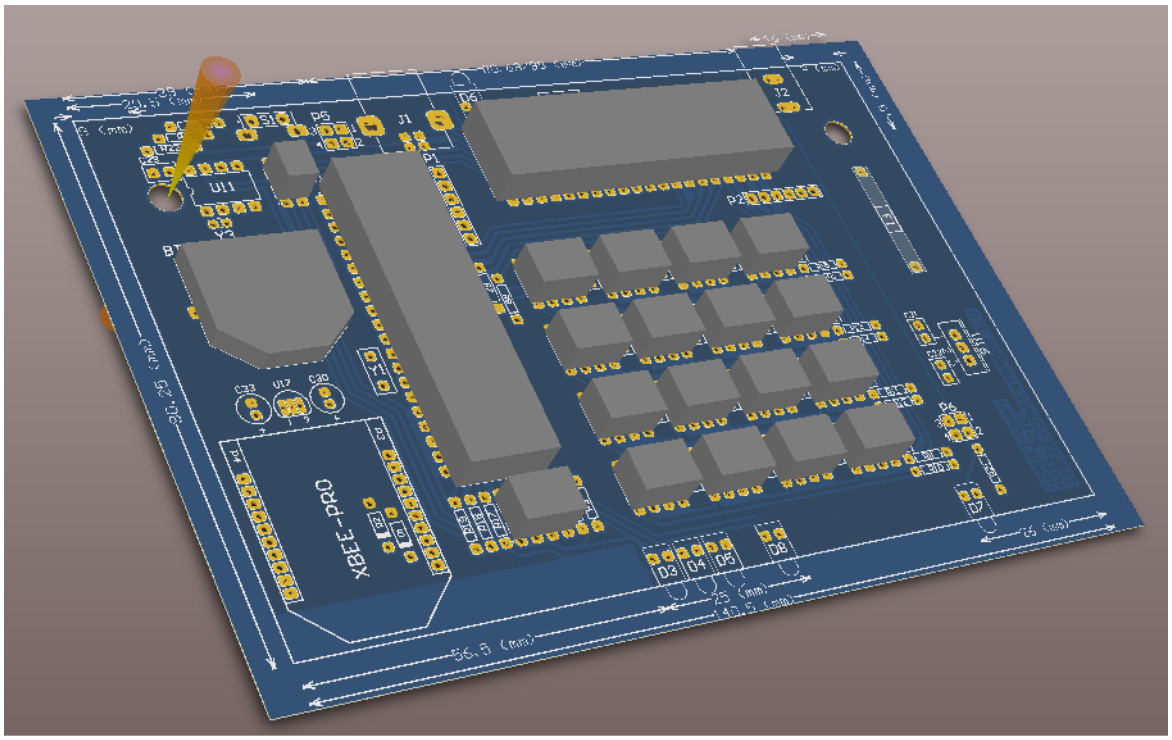

Figura B.4 PCB concentrador 2.0 visión 3D.

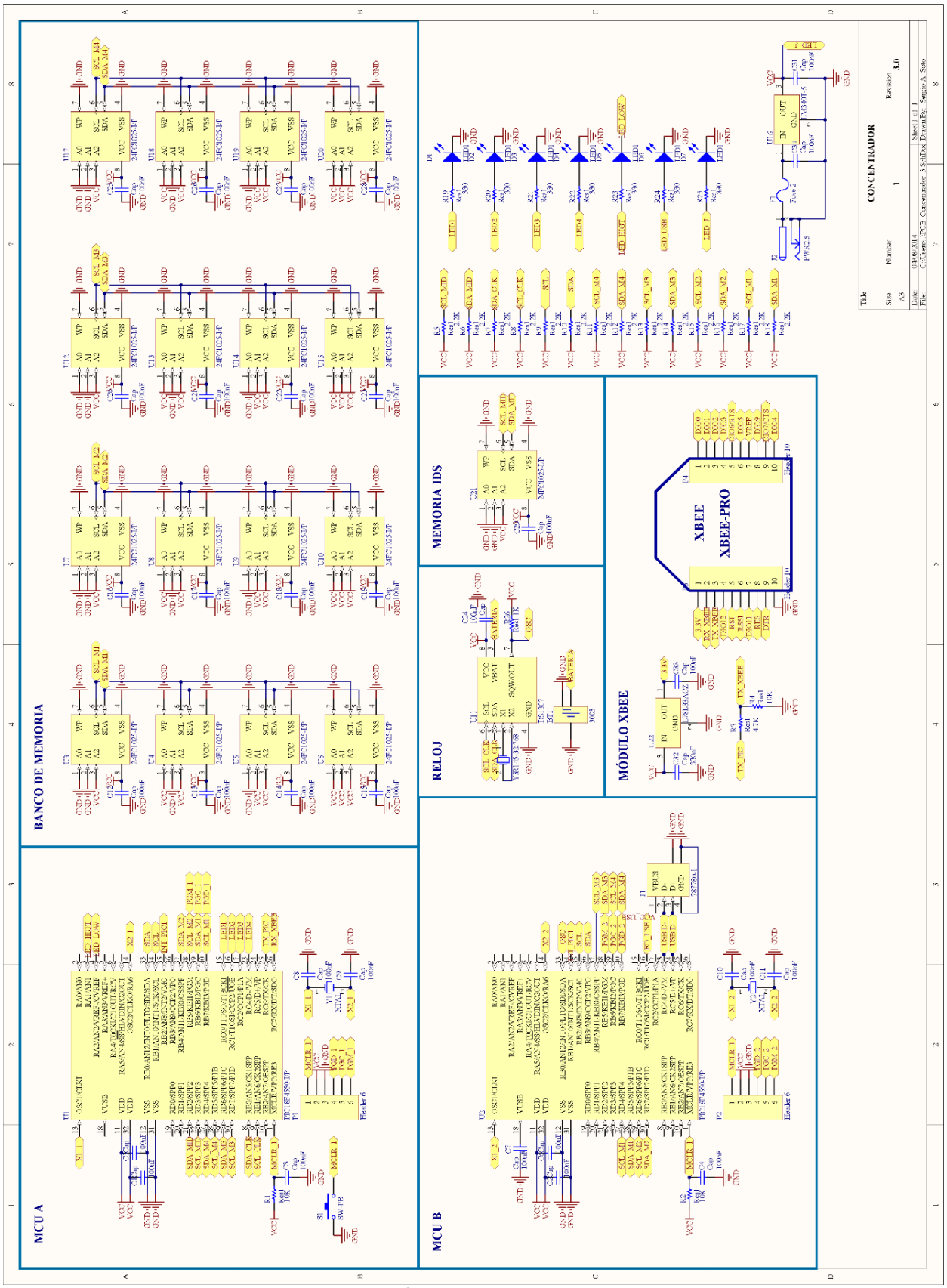

Figura B.5 esquemático completo del concentrador 3.0

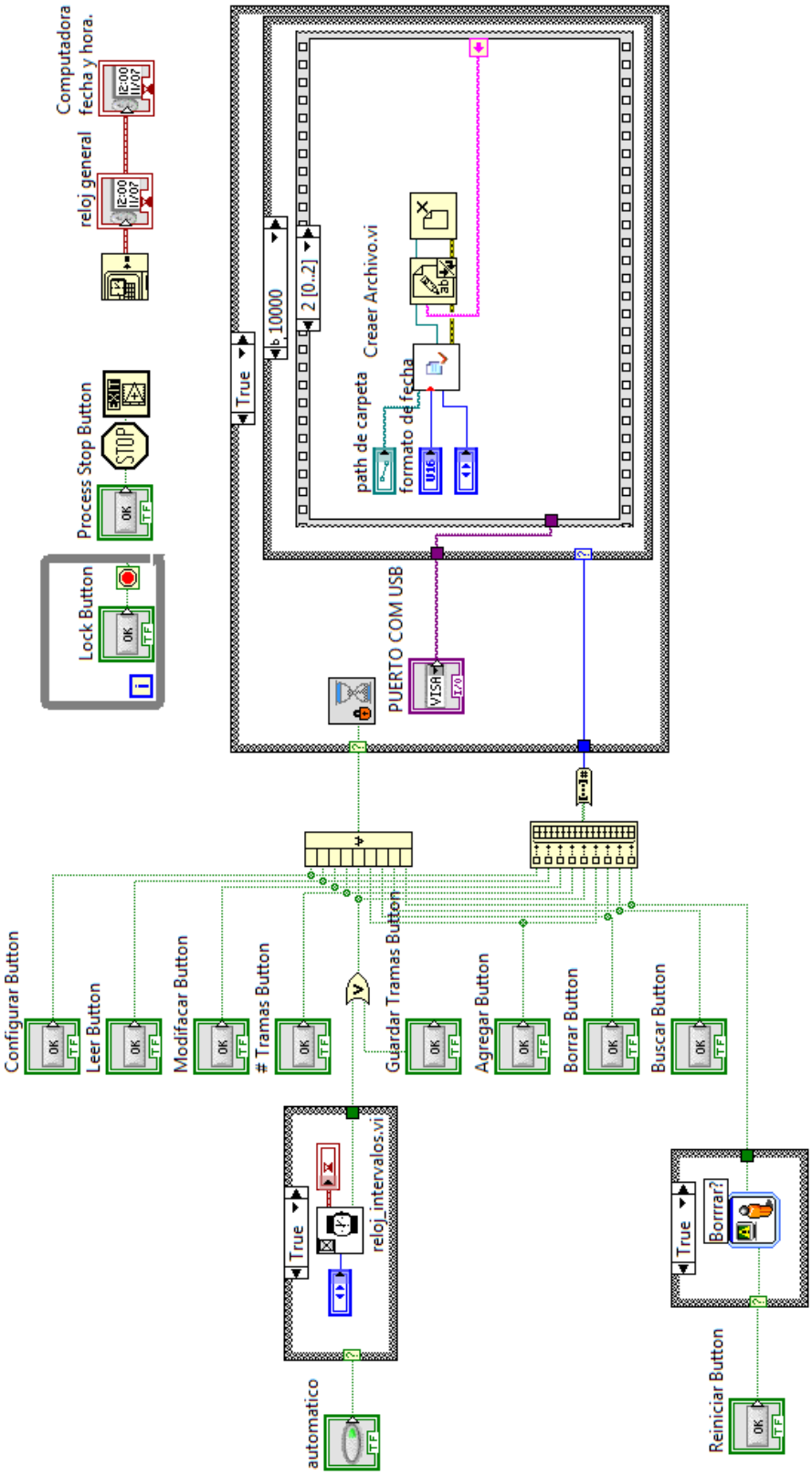

Figura B.6 Diagrama de bloques general de la interfaz.

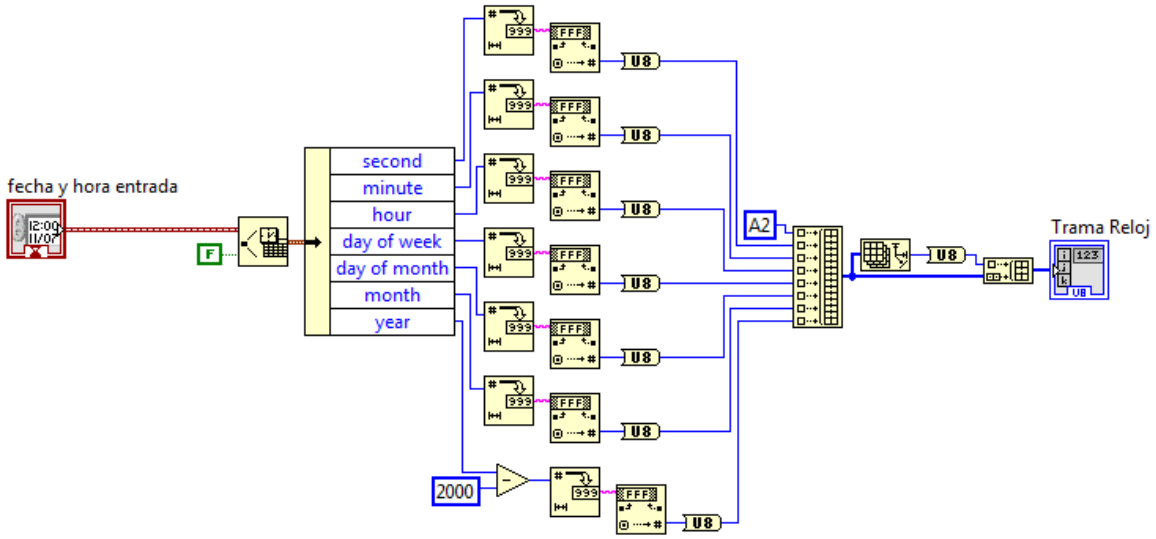

Figura B.7 Diagrama de bloques de casteo de tiempo decimal a BCD.

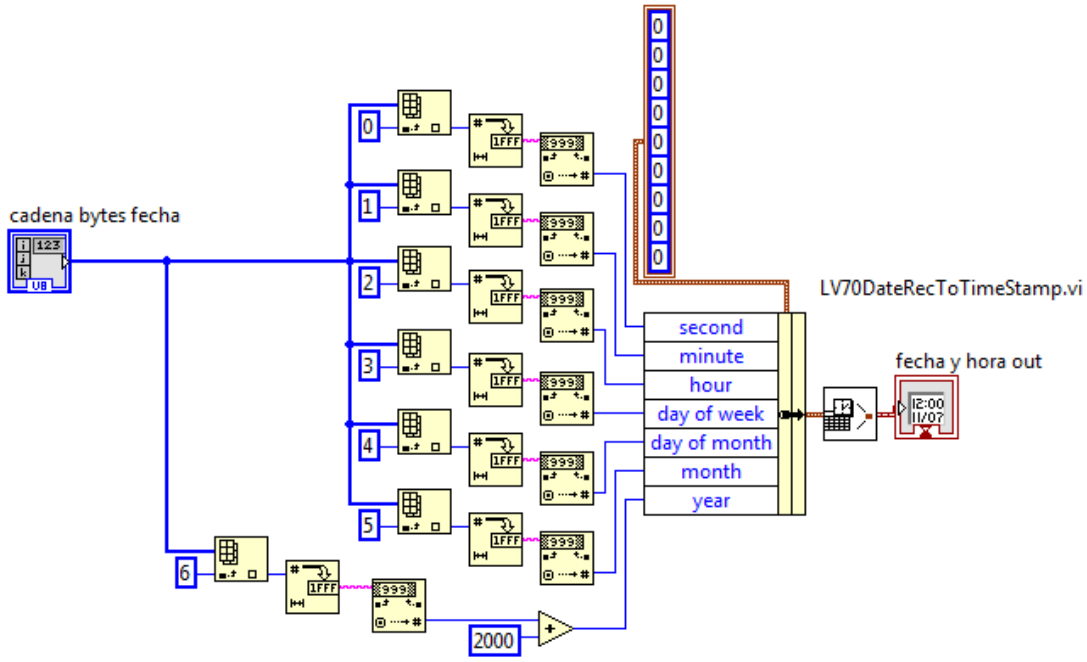

Figura B.8 Diagrama de bloques de casteo de tiempo de BCD a decimal.

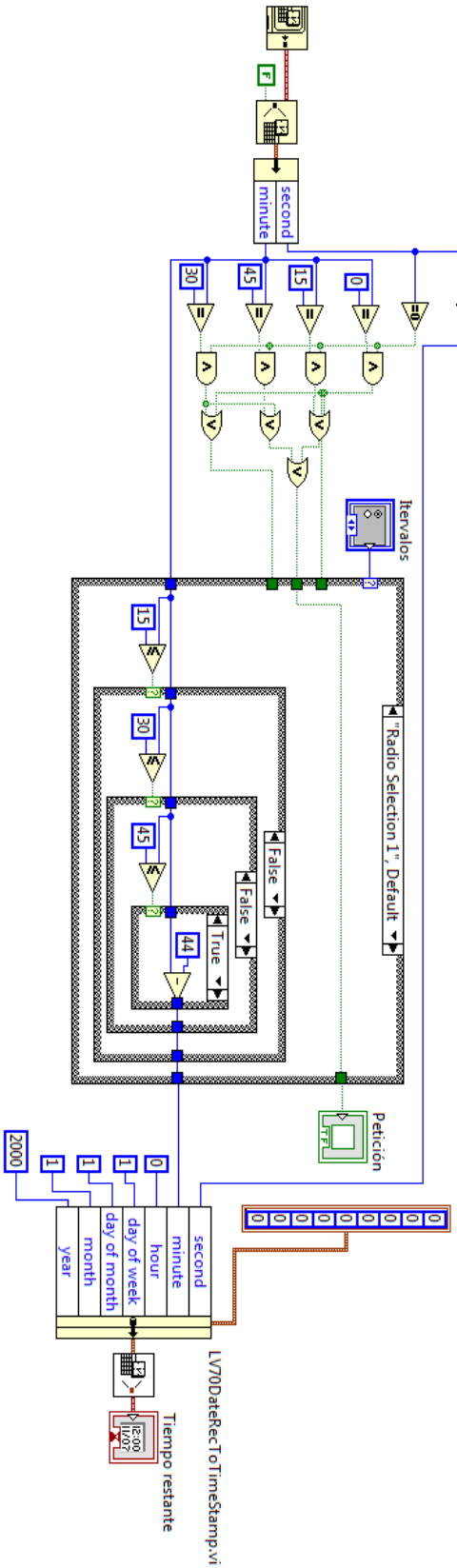

Figura B.9 Diagrama de bloques para back-up automático.

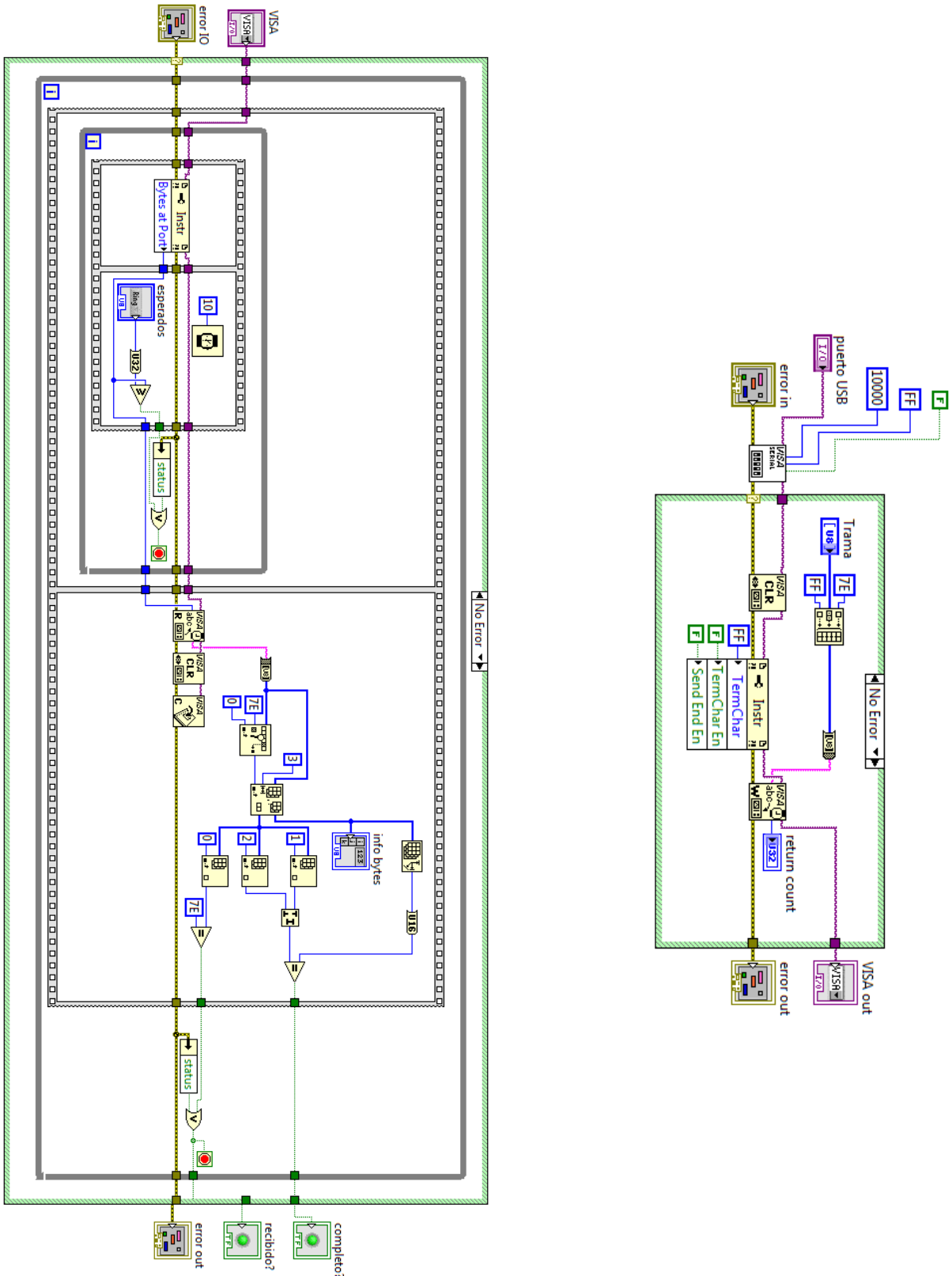

Figura B.10 Diagramas de bloques para envío y recepción de paquetes API.

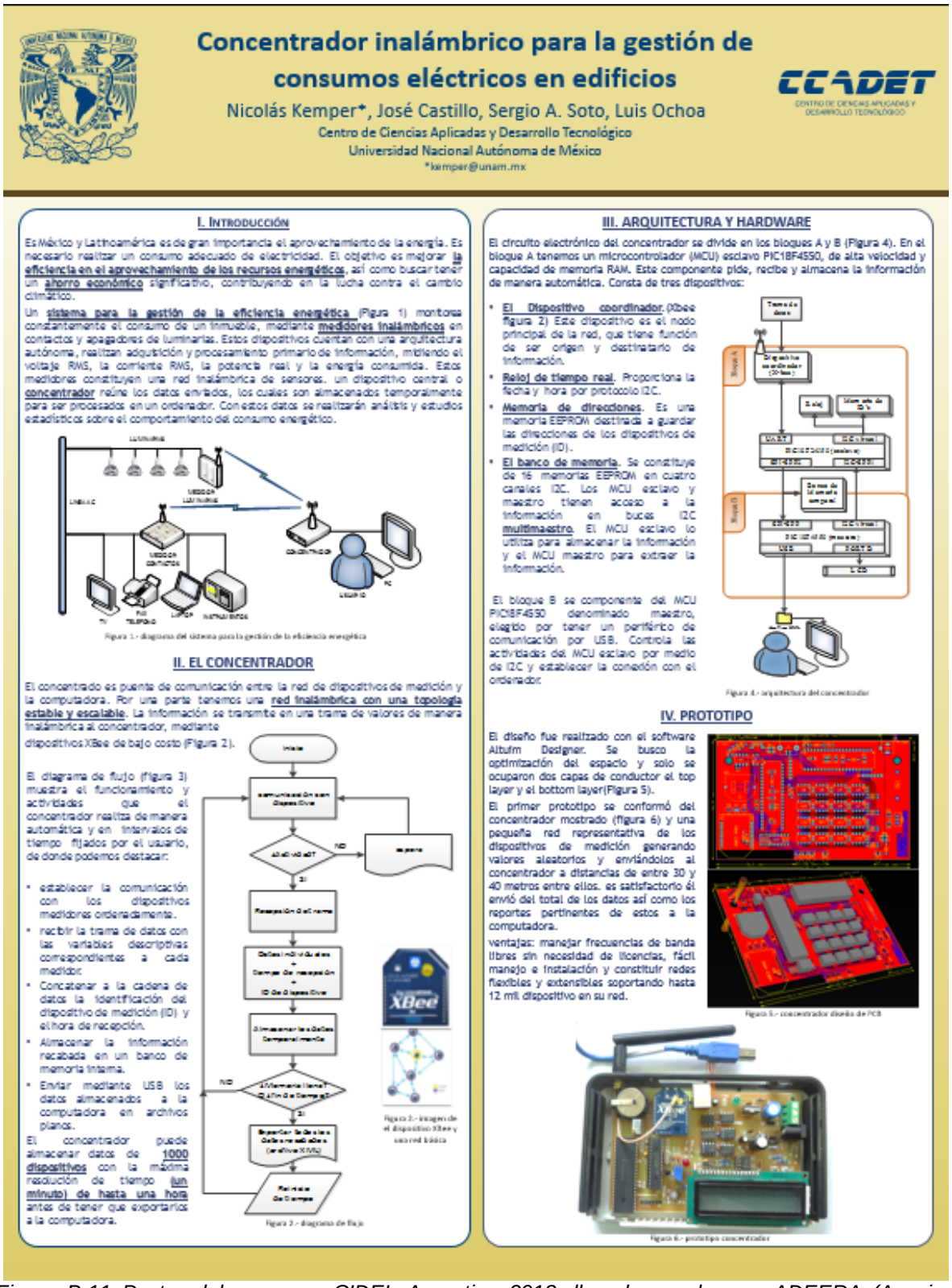

Figura B.11 Poster del congreso CIDEL Argentina 2013, llevado a cabo por ADEERA (Asociación de Distribuidores de Energía Eléctrica de la República Argentina) y CACIER (Comité Argentino de la Comisión de Integración Energética Regional).

Anexo C

Código de programación en lenguaje C

Código C 1 MCU A PIC18FC4550 (USB).

```
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ MCU A PIC18F4550 (USB) ++++++////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
#include <18F4550.h>
#include <stdlibm.h>
#include <STDLIB.h>
#CASE
#ZERO_RAM
#PRIORITY USB,EXT1,EXT                          //prioridad de interrupción
#define __USB_PIC_PERIF__ 1                     //configuración puerto USB
#define __USB_PIC_EXPLORER16__ 1
#if (defined(__PCM__) || defined(__PCB__))
#error USB CDC Library requires PIC18, PIC24, dsPIC30 or dsPIC33
#endif
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48M)
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Configuración de puertos ++++++///////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
#use I2C(MASTER        ,SDA=PIN_B3,SCL=PIN_B2,FORCE_SW,FAST=100000,STREAM=I2C_CTRL)       //Canal I2C de control
#use I2C(MULTI_MASTER,SDA=PIN_D5,SCL=PIN_D4,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M1)   //Canales I2C de Banco de memoria
#use I2C(MULTI_MASTER,SDA=PIN_D7,SCL=PIN_D6,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M2)
#use I2C(MULTI_MASTER,SDA=PIN_B5,SCL=PIN_B4,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M3)
#use I2C(MULTI_MASTER,SDA=PIN_B7,SCL=PIN_B6,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M4)
#use fast_io(b)     //entradas de interrupciones
#use fast_io(c)     //led USB
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Funciones +++++////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
//***rutinas I2C de control***
void WriteByteSlave(BYTE direc,BYTE dato);
BYTE ReadByteSlave(BYTE direc);
void WriteDateSlave(BYTE *arr);
void ReadDateSlave(void);
void WriteIDSlave(BYTE *arr);
void NumID(void);

//***rutinas usuario-concentrador***
void LeerReloj(void);
void ConfigurarReloj(BYTE *arr);
void AgregarIdXbee(BYTE *arr);
void BorrarIdXbee(BYTE *arr);
void BuscarIdXbee(BYTE *arr);
void ReiniciarMemoriaIdXbee(void);
void DefinirRango(BYTE *arr);
void NumeroTramas(void);

//***rutinas seriales USB***
void IntUSBLabVIEW(void);
void BackupTramas(void);
void ID_PortUSB(BYTE aux);
void FechaHora_PortUSB(BYTE aux);

//***interrupciones y auxiliares***
void Decoding(BYTE *arr);
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Librerías ++++++//////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
#include <usb_cdc.h>                #include <BANCO_24CF1025.C>
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Definiciones ++++++///////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
//***registros para el Buffer***
#define CONTROL  0x00
#define FUNCION  0x01
#define N_ID_H   0x02
#define N_ID_L   0x03
#define EST_F    0x04
#define ID_BUF0  0x05
#define ID_BUF1  0x06
#define ID_BUF2  0x07
#define ID_BUF3  0x08
#define SEGUNDO  0x09
#define MINUTO   0x0A
#define HORA     0x0B

#define DIA_SEM  0x0C
#define DIA_MES  0x0D
#define MES      0x0E
#define ANO      0x0F
#define NUMT_H   0x10
#define NUMT_L   0x11

//***funciones entre el MCU A y B***
#define F1       0xF1
#define F2       0xF2
#define F3       0xF3
#define F4       0xF4
#define F5       0xF5
#define F6       0xF6
#define F7       0xF7
#define F9       0xF9

//***órdenes y respuestas***
#define STOP     0x80
#define PLAY     0x84
#define OK       0x88
#define ERROR    0x89

//***dirección de MCU B***
#define SLAVE   0XC0

//***LED de USB***
#define LED1     PIN_C1
#define LED_OFF  output_low
#define LED_ON   output_high
#define LED_TOG  output_toggle
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Variables globales ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BYTE BUFFER[0x12];          short INTER0;          short INTER1;          int RANGO=1;
////////////////////////////////////////////////////////////////////////////
//++++++ Funciones de interrupción +++++////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
#INT_EXT                         //pulso del oscilador DS1307
void ext_interrupcion_0(void){
  static int segundo;            //variables locales
  static int minuto;
  LED_ON(LED1);
  delay_ms(1);
  segundo++;                     //incremento de segundos
  if(segundo>=60){               //minuto completo
    segundo=0;
    minuto++;                    //incremento de minutos
    if(minuto>=RANGO){           //minuto igual al RANGO(usuario)
      minuto=0;
      INTER0=1;                  //bandera petición habilitada
    }
  }
  LED_OFF(LED1);
}

#INT_EXT1                        //pulso de aviso MCU B
void ext_interrupcion_1(void){
          INTER1=0x01;           //bandera de aviso habilitada
}
```

```
////////////////////////////////////////////////////////////////////////////
//++++++ Función principal +++++///////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void main()
{
    set_tris_b(0x03);
    set_tris_c(0x00);
    port_b_pullups(TRUE);
    ext_int_edge(L_TO_H);
    enable_interrupts(INT_EXT);
    enable_interrupts(INT_EXT1);
    enable_interrupts(INT_USB);
    enable_interrupts(GLOBAL);
    usb_init_cs();               //habilitación interrupción USB
    LED_OFF(LED1);               //Led apagado
    while(1){                    //ciclo infinito
       IntUSBLabVIEW();          //recepción de datos del usuario(LabVIEW)
//*******autorización a MCU A para petición de datos***********
       if(INTER0==1){                              //si la bandera está habilitada petición de datos
          INTER0=0;                                //reinicia bandera
          if(ReadByteSlave(CONTROL)==PLAY){ //si MCU B se encuentra desocupado (PLAY)
            WriteByteSlave(FUNCION,F9);            //comando de ciclo de petición para el MCU B
             while(INTER1!=1){}INTER1=0;           //recepción de pulso al MCU A
          }
       }
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función IntUSBLabVIEW +++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void IntUSBLabVIEW(void)
{
   static BYTE bandera;                //variables locales
   static BYTE contador;
   static BYTE numero;
   static BYTE *arreglo;
   BYTE c;
   usb_task();
   if(usb_cdc_kbhit()){                //si Byte recivido
      disable_interrupts(INT_EXT);     //deshabilitar interrupciones
      LED_ON(LED1);
      c=usb_cdc_getc();                //leer Byte en C
      if(bandera==0){                  //si C es primer Byte de mensaje (bandera 0)
         if(c==0x7E){                  //confirmación primer byte 0x7E
           bandera=1;                  //cambio de bandera
           return;
         }
      }»1
```

```
»1  if(bandera==1){                                      //si C es tamaño de mensaje (bandera 1)
        bandera=2;                                       //cambio de bandera
        numero=c+1;                                      //C es el tamaño de mensaje
        arreglo=(BYTE*)calloc(numero,sizeof(BYTE));      //creación arreglo dinámico
        return;
    }
    if(bandera==2){                  //si C es Byte de mensaje (bandera 2)
        arreglo[contador++]=c;       //guardar Byte en arreglo dinámico
        if(contador>=numero){        //si C es ultimo Byte de mensaje
           Decoding(arreglo);        //decodificar mensaje
           free(arreglo);            //liberar arreglo dinámico
           contador=0x00;            //reiniciar bandera y contador local
           bandera=0;
        }
    }
  LED_OFF(LED1);
  enable_interrupts(INT_EXT);        //habilitar interrupciones
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función Decoding +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Decoding(BYTE *arr)
{
   BYTE est;
   est=ReadByteSlave(FUNCION);       //leer registro de FUNCION del MCU B
   switch(arr[0x00]){
      case 0xA1:    if(est==0)LeerReloj();                  //rutina leer reloj
                    else FechaHora_PortUSB(est);            //aviso concentrador ocupado
                    break;
      case 0xA2:    if(est==0)ConfigurarReloj(arr);         //rutina configurar reloj
                    else FechaHora_PortUSB(est);            //aviso concentrador ocupado
                    break;
      case 0xB1:    if(est==0)AgregarIdXbee(arr);           //rutina agregar ID
                    else ID_PortUSB(est);                   //aviso concentrador ocupado
                    break;
      case 0xB2:    if(est==0)BorrarIdXbee(arr);            //rutina borrar ID
                    else ID_PortUSB(est);                   //aviso concentrador ocupado
                    break;
      case 0xB3:    if(est==0)BuscarIdXbee(arr);            //rutina buscar ID
                    else ID_PortUSB(est);                   //aviso concentrador ocupad
                    break;
      case 0xB4:    if(est==0)ReiniciarMemoriaIdXbee();     //rutina reiniciar memoria ID
                    else ID_PortUSB(est);                   //aviso concentrador ocupado
                    break;
      case 0xC1:    DefinirRango(arr);                      //Nuevo rango de tiempo
                    break;
      case 0xD1:    if(est==0)NumeroTramas();               //envío al usuario de número tramas
                    break;
      case 0xD2:    if(est==0)BackupTramas();               //envío al usuario de datos
                    break;
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función WriteByteSlave +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void WriteByteSlave(BYTE direc,BYTE dato)
{                                                   //escribir un dato en el BUFFER-MCU B
   i2c_start(I2C_CTRL);
   i2c_write(I2C_CTRL,SLAVE);
   i2c_write(I2C_CTRL,direc);
   i2c_write(I2C_CTRL,dato);
   i2c_stop(I2C_CTRL);
   delay_ms(5);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReadByteSlave +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BYTE ReadByteSlave(BYTE direc)
{                                                   //leer un dato del BUFFER-MCU B
    BYTE dato;
    i2c_start(I2C_CTRL);
    i2c_write(I2C_CTRL,SLAVE);
    i2c_write(I2C_CTRL,direc);
    i2c_start(I2C_CTRL);
    i2c_write(I2C_CTRL,(SLAVE+0X01));
    dato=i2c_read(I2C_CTRL,0);
     i2c_stop(I2C_CTRL);
     delay_ms(5);
     return(dato);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función WriteDateSlave +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////
void WriteDateSlave(BYTE *arr)
{  //escribir fecha completa en el BUFFER-MCU B
   i2c_start(I2C_CTRL);
   i2c_write(I2C_CTRL,SLAVE);
   i2c_write(I2C_CTRL,SEGUNDO);
   i2c_write(I2C_CTRL,arr[0x01]);
   i2c_write(I2C_CTRL,arr[0x02]);
   i2c_write(I2C_CTRL,arr[0x03]);
   i2c_write(I2C_CTRL,arr[0x04]);
   i2c_write(I2C_CTRL,arr[0x05]);
   i2c_write(I2C_CTRL,arr[0x06]);
   i2c_write(I2C_CTRL,arr[0x07]);
   i2c_stop(I2C_CTRL);
   delay_ms(5);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReadDateSlave +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ReadDateSlave(void)
{  //leer fecha completa del BUFFER-MCU B
   i2c_start(I2C_CTRL);
   i2c_write(I2C_CTRL,SLAVE);
   i2c_write(I2C_CTRL,SEGUNDO);
   i2c_start(I2C_CTRL);
   i2c_write(I2C_CTRL,(SLAVE+0x01));
   BUFFER[SEGUNDO]=i2c_read(I2C_CTRL,1);
   BUFFER[MINUTO   ]=i2c_read(I2C_CTRL,1);
   BUFFER[HORA     ]=i2c_read(I2C_CTRL,1);
   BUFFER[DIA_SEM  ]=i2c_read(I2C_CTRL,1);
   BUFFER[DIA_MES  ]=i2c_read(I2C_CTRL,1);
   BUFFER[MES      ]=i2c_read(I2C_CTRL,1);
   BUFFER[ANO      ]=i2c_read(I2C_CTRL,0);
   i2c_stop(I2C_CTRL);
   delay_ms(5);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función WriteIDSlave +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void WriteIDSlave(BYTE *arr)
{           //escribir una ID en el BUFFER-MCU B
           i2c_start(I2C_CTRL);
           i2c_write(I2C_CTRL,SLAVE);
           i2c_write(I2C_CTRL,EST_F);
           i2c_write(I2C_CTRL,arr[0x01]);
           i2c_write(I2C_CTRL,arr[0x02]);
           i2c_write(I2C_CTRL,arr[0x03]);
           i2c_write(I2C_CTRL,arr[0x04]);
           i2c_write(I2C_CTRL,arr[0x05]);
           i2c_stop(I2C_CTRL);
           delay_ms(5);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función NumID +++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void NumID(void)
{  //leer número de IDs  del BUFFER-MCU B
    i2c_start(I2C_CTRL);
    i2c_write(I2C_CTRL,SLAVE);
    i2c_write(I2C_CTRL,N_ID_H);
    i2c_start(I2C_CTRL);
    i2c_write(I2C_CTRL,(SLAVE+0X01));
    BUFFER[N_ID_H]=i2c_read(I2C_CTRL,1);
    BUFFER[N_ID_L]=i2c_read(I2C_CTRL,1);
    BUFFER[EST_F]=i2c_read(I2C_CTRL,0);
          i2c_stop(I2C_CTRL);
          delay_ms(5);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función LeerReloj +++++////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void LeerReloj(void)
{
  WriteByteSlave(CONTROL,STOP);    //Pausa del MCU B
  WriteByteSlave(FUNCION,F1);      //comando de función para el MCU B
  while(INTER1!=1){}INTER1=0;      //recepción de pulso al MCU A
  ReadDateSlave();                 //leer fecha y hora del MCU B
  BUFFER[EST_F]=ReadByteSlave(EST_F);//leer estado de función
  WriteByteSlave(CONTROL,PLAY);    //Play MCU B
  FechaHora_PortUSB(0xA0);         //envío de fecha y hora al usuario(LabVIEW)
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función AgregarIdXbee +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void AgregarIdXbee(BYTE *arr)
{
   WriteByteSlave(CONTROL,STOP);   //Pausa del MCU B
   WriteIDSlave(arr);              //escribir ID al MCU B
   WriteByteSlave(FUNCION,F3);     //comando de función para el MCU B
   while(INTER1!=1){}INTER1=0;     //recepción de pulso al MCU A
   NumID();                        //leer número de IDs del MCU B
   WriteByteSlave(CONTROL,PLAY);   //Play del MCU B
   ID_PortUSB(0xB1);               //envío de número de IDs al usuario(LabVIEW)
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función BuscarIdXbee +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void BuscarIdXbee(BYTE *arr)
{
  WriteByteSlave(CONTROL,STOP);    //Pausa del MCU B
  WriteIDSlave(arr);               //escribir ID al MCU B
  WriteByteSlave(FUNCION,F5);      //comando de función para el MCU B
  while(INTER1!=1){}INTER1=0;      //recepción de pulso al MCU A
  NumID();                         //leer número de IDs del MCU B
  WriteByteSlave(CONTROL,PLAY);    //Play del MCU B
  ID_PortUSB(0xB3);                //envío de número de IDs al usuario(LabVIEW)
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función DefinirRango +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DefinirRango(BYTE *arr)
{
  RANGO=arr[0x01];                 //cambiar el valor de RANGO por el nuevo
  usb_task();
  while(usb_cdc_putready()!=TRUE){}
  usb_cdc_putc(0x7E);              //confirmación del cambio al usuario(LabVIEW)
  usb_cdc_putc(0x00);
  usb_cdc_putc(0x01);
  usb_cdc_putc(0xC1);
  usb_cdc_putc(RANGO);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función ConfigurarReloj +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ConfigurarReloj(BYTE *arr)
{
  WriteByteSlave(CONTROL,STOP);    //Pausa del MCU B
  WriteDateSlave(arr);             //escribir nueva fecha y hora al MCU B
  WriteByteSlave(FUNCION,F2);      //comando de función para el MCU B
  while(INTER1!=1){}INTER1=0;      //recepción de pulso al MCU A
  ReadDateSlave();                 //leer fecha y hora del MCU B
  BUFFER[EST_F]=ReadByteSlave(EST_F);//leer estado de función
  WriteByteSlave(CONTROL,PLAY);    //Play del MCU B
  FechaHora_PortUSB(0xA0);         //retorno de fecha y hora al usuario(LabVIEW)
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función BorrarIdXbee +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void BorrarIdXbee(BYTE *arr)
{
  WriteByteSlave(CONTROL,STOP);    //Pausa del MCU B
  WriteIDSlave(arr);               //escribir ID al MCU B
  WriteByteSlave(FUNCION,F4);      //comando de función para el MCU B
  while(INTER1!=1){}INTER1=0;      //recepción de pulso al MCU A
  NumID();                         //leer número de IDs del MCU B
  WriteByteSlave(CONTROL,PLAY);    //Play del MCU B
  ID_PortUSB(0xB2);                //envío de número de IDs al usuario(LabVIEW)
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReiniciarMemoriaIdXbee +++++////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ReiniciarMemoriaIdXbee(void)
{
  WriteByteSlave(CONTROL,STOP);    //Pausa del MCU B
  WriteByteSlave(FUNCION,F6);      //comando de función para el MCU B
  while(INTER1!=1){}INTER1=0;      //recepción de pulso al MCU A
  NumID();                         //leer número de IDs del MCU B
  WriteByteSlave(CONTROL,PLAY);    //Play del MCU B
  ID_PortUSB(0xB4);                //envío de número de IDs al usuario(LabVIEW)
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función NumeroTramas +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void NumeroTramas(void)
{
  WriteByteSlave(CONTROL,STOP);                //Pausa del MCU B
  BUFFER[NUMT_H]=ReadByteSlave(NUMT_H);//leer número de tramas guardadas
  BUFFER[NUMT_L]=ReadByteSlave(NUMT_L);
  WriteByteSlave(CONTROL,PLAY);                //Play del MCU B
  usb_task();
  while(usb_cdc_putready()!=TRUE){}
  usb_cdc_putc(0x7E);                          //envío de número de tramas al usuario(LabVIEW)
  usb_cdc_putc(0x00);
  usb_cdc_putc(0x04);
  usb_cdc_putc(0xD1);
  usb_cdc_putc(BUFFER[EST_F]);
  usb_cdc_putc(BUFFER[NUMT_H]);
  usb_cdc_putc(BUFFER[NUMT_L]); }
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función BackupTramas +++++/////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void BackupTramas(void)
{
  static ADDRESS num_tramas;
  static unsigned long ap;
  DATAFRAME d;
  int i;
  if(ap==0){                                          //apuntador posición inicial
    WriteByteSlave(CONTROL,STOP);                     //Pausa del MCU B
    num_tramas.b[1]=BUFFER[NUMT_H];                   //leer número de tramas guardadas
    num_tramas.b[0]=BUFFER[NUMT_L];
  }
  for(i=0;i<4;i++){
     d=LoadDataFrameMemory(ap++,num_tramas.i);        //leer una trama del banco de memoria (Librería BANCO_MEMORIA_24CF1025.C)
     usb_task();
     while(usb_cdc_putready()!=TRUE){}
     printf(usb_cdc_putc,"%x\t%x\t%x\t%lX\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n",d.Ti.dia_mes,d.Ti.hora,d.Ti.minuto,d.Id,d.Tr.V,d.Tr.I,d.Tr.FP,d.Tr.E,d.Tr.W);//envío de los valores guardados por renglón x4
   }
   if(ap>=num_tramas.i){                              //si el apuntador está en posición fin de lista
     ap=0;
     WriteByteSlave(FUNCION,F7);                      //comando reiniciar lista de tramas del MCU B
     while(INTER1!=1){}INTER1=0;                      //recepción de pulso al MCU A
     WriteByteSlave(CONTROL,PLAY);                    //Play del MCU B
         }
}
```

```
////////////////////////////////////////////////////////////////////////////
//++++++ Función ID_PortUSB +++++///////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void ID_PortUSB(BYTE aux)
{                                  //envío de número de IDs al usuario (LabVIEW)
  usb_task();
  while(usb_cdc_putready()!=TRUE){}
  usb_cdc_putc(0x7E);
  usb_cdc_putc(0x00);
  usb_cdc_putc(0x04);
  usb_cdc_putc(aux);
  usb_cdc_putc(BUFFER[EST_F  ]);
  usb_cdc_putc(BUFFER[N_ID_H ]);
  usb_cdc_putc(BUFFER[N_ID_L ]);
}
```

```
////////////////////////////////////////////////////////////////////////////
//++++++ Función FechaHora_PortUSB +++++////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void FechaHora_PortUSB(BYTE aux)
{                                  //envío de fecha y hora al usuario (LabVIEW)
  usb_task();
  while(usb_cdc_putready()!=TRUE){}
  usb_cdc_putc(0x7E);
  usb_cdc_putc(0x00);
  usb_cdc_putc(0x09);
  usb_cdc_putc(aux);
  usb_cdc_putc(BUFFER[EST_F       ]);
  usb_cdc_putc(BUFFER[SEGUNDO     ]);
  usb_cdc_putc(BUFFER[MINUTO      ]);
  usb_cdc_putc(BUFFER[HORA        ]);
  usb_cdc_putc(BUFFER[DIA_SEM     ]);
  usb_cdc_putc(BUFFER[DIA_MES     ]);
  usb_cdc_putc(BUFFER[MES         ]);
  usb_cdc_putc(BUFFER[ANO         ]);
}
```

Código C 2 MCU B PIC18FC4550.

```
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ MCU B PIC18F4550. ++++++/////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <18F4550.h>
#CASE
#ZERO_RAM
#PRIORITY RDA,SSP
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48M)
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Configuración de puertos ++++++///////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#use I2C(SLAVE ,SDA=PIN_B0,SCL=PIN_B1,FORCE_HW,FAST=100000,ADDRESS=0xC0,STREAM=I2C_CTRL)      //Canal I2C de control
#use I2C(MASTER,SDA=PIN_E0,SCL=PIN_E1,FORCE_SW,FAST=100000,STREAM=I2C_RTC)                   //Canal I2C de reloj
#use I2C(MULTI_MASTER,SDA=PIN_D2,SCL=PIN_D3,FORCE_SW,FAST=100000,STREAM=I2C_MEM_ID)          //Canal I2C de memoria ID
#use I2C(MULTI_MASTER,SDA=PIN_B6,SCL=PIN_B7,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M1)        //Canales I2C de Banco de memoria
#use I2C(MULTI_MASTER,SDA=PIN_B4,SCL=PIN_B5,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M2)
#use I2C(MULTI_MASTER,SDA=PIN_D6,SCL=PIN_D7,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M3)
#use I2C(MULTI_MASTER,SDA=PIN_D4,SCL=PIN_D5,FORCE_SW,FAST=100000,STREAM=I2C_BANCO_M4)
#use rs232(XMIT=PIN_C6,RCV=PIN_C7,BAUD=57600,PARITY=N,BITS=8,STOP=1)                          //Canal serie módulo Xbee
#use fast_io(c)
#use fast_io(a)
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Funciones +++++///////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Inicio(void);          void ConfigurarReloj(void);     void BuscarIdXbee(void);      void GuardarTrama(void);      short Conexion(int32 id,BYTE comando);
int  MenuXbee(void);        void AgregarIdXbee(void);       void ReiniciarIdXbee(void);   void PeticionTramas(void);    void DisplayLeds(long numero);
void LeerReloj(void);       void BorrarIdXbee(void);        void Backup(void);
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Librerías ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <RTC_DS1307.C>              #include <MEMORIA_AUXILIAR_24FC1025.C>    #include <BANCO__MEMORIA_24CF1025.C>     #include <XBEE_S2.C>
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Definiciones ++++++///////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//***registros para el Buffer***       //***funciones entre el MCU A y B***     //***conexiones con la red de Xbee***
#define CONTROL  0x00                  #define F1         0xF1                   #define NETWORK          0x3032
#define FUNCION  0x01                  #define F2         0xF2                   #define COORDINADOR      0x40A11D82
#define N_ID_H   0x02                  #define F3         0xF3                   //***display de leds***
#define N_ID_L   0x03                  #define F4         0xF4                   #define LED1       PIN_C0
#define EST_F    0x04                  #define F5         0xF5                   #define LED2       PIN_C1
#define ID_BUF0  0x05                  #define F6         0xF6                   #define LED3       PIN_C2
#define ID_BUF1  0x06                  #define F7         0xF7                   #define LED4       PIN_C4
#define ID_BUF2  0x07                  #define F9         0xF9                   #define LEDR       PIN_A1
#define ID_BUF3  0x08                  //***órdenes y respuestas***              #define LEDV       PIN_A2
#define SEGUNDO  0x09                  #define STOP       0x80                   #define OFF        output_low
#define MINUTO   0x0A                  #define PLAY       0x84                   #define ON         output_high
#define HORA     0x0B                  #define OK         0x88                   #define TOG        output_toggle
#define DIA_SEM  0x0C                  #define ERROR      0x89                   #define LEDV_ON          OFF(LEDR);ON(LEDV)
#define DIA_MES  0x0D                  //***búsqueda de ID***                    #define LEDR_ON          OFF(LEDV);ON(LEDR)
#define MES      0x0E                  #define YY         0x8A                   #define LEDV_TOG         LEDV_ON;delay_ms(500);TOG(LEDV);delay_ms(500)
#define ANO      0x0F                  #define YN         0x8B                   #define LEDR_TOG         LEDR_ON;delay_ms(500);TOG(LEDR);delay_ms(500)
#define NUMT_H   0x10                  #define NY         0x8C                   #define LED_VR_TOG       LEDV_ON;delay_ms(500);LEDR_ON;delay_ms(500)
#define NUMT_L   0x11                  #define NN         0x8D                   #define OUT_INT          ON(PIN_B2);delay_ms(5);OFF(PIN_B2)
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Variables globales ++++++/////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
BYTE BUFFER[0x12];
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Funciones de interrupción +++++////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#int_SSP                                        //interrupción canal I2C
void ssp_interrupcion(void)
{
   static BYTE direccion;
   BYTE entrante,estado;
   estado=i2c_isr_state(I2C_CTRL);              //retorna el tipo de byte recibido
   if(estado<0x80){                             //si el maestro envío un dato
         entrante=i2c_read(I2C_CTRL);           //se lee el byte entrante
         if(estado==1)                          //si el primer BYTE recibido es una dirección
         direccion=entrante;                    //se lee el byte como una dirección
         if(estado>=2 && estado!=0x80)          //si el segundo BYTE recibido es un dato
         BUFFER[direccion++]=entrante;          //se lee el dato y se incrementa el apuntador de
dirección
  }
  if(estado>=0x80)                              //si el maestro requiere un dato
     i2c_write(I2C_CTRL,BUFFER[direccion++]); //envía el dato referenciado a la dirección
}
```

```
#INT_RDA            //interrupción puerto UART
void rda_interrupcion(void)
{
   IntXbee();       //conexión con librería XBEE_S2.C
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función principal +++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void main()
{
   set_tris_c(0x00);
   set_tris_a(0x00);
   enable_interrupts(INT_SSP);
   enable_interrupts(INT_RDA);
   enable_interrupts(GLOBAL);
   OFF(LED1);      //apagar los leds
   OFF(LED2);
   OFF(LED3);
   OFF(LED4);
   OFF(LEDV);
   OFF(LEDR);
   Inicio();       //configuración
»1
```

```
»1 while(1){                            //ciclo infinito
      if(BUFFER[CONTROL]==STOP){        //actividad pausada
         LED_VR_TOG;                    //led estatus rojo/verde
         switch(BUFFER[FUNCION]){       //selección de función
               case F1:  LeerReloj();          break;
               case F2:  ConfigurarReloj();    break;
               case F3:  AgregarIdXbee();      break;
               case F4:  BorrarIdXbee();       break;
               case F5:  BuscarIdXbee();       break;
               case F6:  ReiniciarIdXbee();    break;
               case F7:  Backup();             break;
               }
      }
»2
```

```
»2  if(BUFFER[CONTROL]==PLAY){              //actividad normal
         MenuXbee();                        //estado del módulo Xbee
         if(BUFFER[FUNCION]==F9){           //petición de datos activa
               LEDR_ON;                     //led estatus rojo
               PeticionTramas();            //petición de trama
               LEDV_ON;                     //led estatus verde
         }
         if(BUFFER[EST_F]==ERROR){          //si ocurre error
               LEDR_TOG;                    //led estatus rojo
         }else{                             //sin errores
               LEDV_TOG;                    //led estatus verde
            }
         }
     }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función Inicio +++++///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Inicio(void)
{
  ADDRESS numero;
  int i;
  for(i=CONTROL;i<=NUMT_H;i++)      //limpieza de BUFFER
        BUFFER[i]=0x00;
  numero.i=LoadNumer();             //recupera el número de tramas almacenadas
  BUFFER[NUMT_H ]=numero.b[1];      //          escribe tal valor en los registros.
  BUFFER[NUMT_L ]=numero.b[0];
  DisplayLeds(numero.i);            //habilita display de leds
»1
```

```
»1 WriteATCommand(ID,NETWORK);   //configura la LAN de la red Xbee
   WriteATCommand(AO,0);         //habilita comandos tipo API
   ATCommand(SL);                //lee el ID Xbee puerto UART
   delay_ms(10);
   if(IDENTIDAD==COORDINADOR){   //si el Xbee es el correcto
     ON(LEDV);                   //led estatus verde
     BUFFER[CONTROL]=PLAY;       //actividad normal habilitada
   }else{                        //error, Xbee incorrecto o comunicación nula
     ON(LEDR);                   //led estatus rojo
     while(1);                   //ciclo de error infinito
   }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función LeerReloj ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void LeerReloj(void)
{
   int i;
   FECHA f;
   f=ReadDate();                                  //lectura de fecha(librería  RTC_DS1307.C).
   BUFFER[SEGUNDO]=     f.segundo;                //asignación de valores de reloj al BUFFER
   BUFFER[MINUTO   ]=   f.minuto;
   BUFFER[HORA     ]=   f.hora;
   BUFFER[DIA_SEM  ]=   f.dia_sem;
   BUFFER[DIA_MES  ]=   f.dia_mes;
   BUFFER[MES      ]=   f.mes;
   BUFFER[ANO      ]=   f.ano;
   for(i=SEGUNDO;i<=ANO;i++){                     //chequeo de los valores
     if(BUFFER[i]==0xFF){                         //si son nulos
        BUFFER[EST_F]=ERROR;                      //reporte de error
        return;                                   //finaliza la función
      }
   }
  BUFFER[EST_F  ]=OK;                             //finalización correcta
  BUFFER[FUNCION]=0x00;                           //finaliza la acción
  OUT_INT;                                        //pulso de aviso al MCU A
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función AgregarIdXbee ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void AgregarIdXbee(void)
{
   IDXBEE id;
   STATUS st;
   id.b[0x00]=BUFFER[ID_BUF0];                    //Asignación de los valores de ID
   id.b[0x01]=BUFFER[ID_BUF1];                    //     en una variable tipo IDXBEE.
   id.b[0x02]=BUFFER[ID_BUF2];
   id.b[0x03]=BUFFER[ID_BUF3];
   if(Conexion(id.i,PETICION_STD)==TRUE){         //buscar ID en red
     if(IdXbeeSearchFast(id.i)==TRUE)             //buscar ID en memoria
           BUFFER[EST_F]=YY;                      //SI red y memoria
     else  BUFFER[EST_F]=YN;                      //SI red, NO memoria
    }else{
     if(IdXbeeSearchFast(id.i)==TRUE)             //buscar ID en memoria
           BUFFER[EST_F]=NY;                      //NO red, SI memoria
      else BUFFER[EST_F]=NN;                      //NO red y memoria
   }
   if(BUFFER[EST_F]==YN||BUFFER[EST_F]==NN)//si la ID no existe en memoria
     SaveNewIdXbee(id.i);                         //guardar ID en memoria

  st=LoadListStatus();
  BUFFER[N_ID_H ]=(st.total>>8);                  //número total de ID en memoria
  BUFFER[N_ID_L ]=st.total;
  BUFFER[FUNCION]=0x00;                           //finaliza la acción
  OUT_INT;                                        //pulso de aviso al MCU A
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función LeerReloj ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ConfigurarReloj(void)
{
   FECHA f;
   f.segundo=BUFFER[SEGUNDO];     //Asignación de los valores de reloj
   f.minuto   =BUFFER[MINUTO];    //        en una variable tipo FECHA.
   f.hora     =BUFFER[HORA];
   f.dia_sem=BUFFER[DIA_SEM];
   f.dia_mes=BUFFER[DIA_MES];
   f.mes      =BUFFER[MES];
   f.ano      =BUFFER[ANO];
   WriteDate(f);                  //escritura de fecha(Librería RTC_DS1307.C)
   LeerReloj();                   //función del lectura confirmar la acción.
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función BorrarIdXbee ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void BorrarIdXbee(void)
{
   IDXBEE id;
   STATUS st;
   id.b[0x00]=BUFFER[ID_BUF0];    //Asignación de los valores de ID
   id.b[0x01]=BUFFER[ID_BUF1];    //     en una variable tipo IDXBEE.
   id.b[0x02]=BUFFER[ID_BUF2];
   id.b[0x03]=BUFFER[ID_BUF3];
   if(DeleteIdXbee(id.i)==1)      //buscar ID en memoria y borrar
      BUFFER[EST_F]=NY;           //NO red, SI memoria
   else
      BUFFER[EST_F]=NN;           //NO red, NO memoria

   st=LoadListStatus();
   BUFFER[N_ID_H]=(st.total>>8);  //número total de ID en memoria
   BUFFER[N_ID_L]=st.total;
   BUFFER[FUNCION]=0x00;          //finaliza la acción
   OUT_INT;                       //pulso de aviso al MCU A
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función MenuXbee ++++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int MenuXbee(void)
{
   if(ESTADO>=0x20&&ESTADO<0x30){//variable global ESTADO (librería XBEE_S2.C)
      switch(ESTADO){
            case 0x28:          ESTADO=0x00;     //Comunicación fallida
                                return(1); break;
            case 0x29:          ESTADO=0x00;     //Comunicación fallida
                                return(1); break;
      }
   }
   if(ESTADO>=0x30&&ESTADO<0x40){
      LEDR_ON;
      switch(ESTADO){
            case STD_OK:        ESTADO=0x00;     //Recepción comando OK
                                return(2); break;
            case STD_ERROR: ESTADO=0x00;     //Recepción comando ERROR
                                return(3); break;
            case STD_OCUPADO:ESTADO=0x00; //Recepción comando OCUPADO
                                return(4); break;
            case MED_NUEVO:ESTADO=0x00;     //Recepción comando MED_NUEVO
                                SaveNewIdXbee(NUEVA_ID);
                                             //guardar nueva ID (Librería ID_24CF1025.C)
                                CommandTransmission(IDENTIDAD,NUEVA_ID,STD_OK);
                                             //envío comando OK (librería XBEE_S2.C)
                                return(5); break;
            case MED_ENVIO: ESTADO=0x00;     //Recepción comando MD_ENVIO
                                GuardarTrama();  //guardar valores recibidos
                                return(6); break;
      }
   }
   return(0);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función Conexion ++++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
short Conexion(int32 id,BYTE comando)
{
   long tiempo=0;                    //condición de tiempo
   if(comando==PETICION_STD){        //caso petición de estado
      CommandTransmission(IDENTIDAD,id,PETICION_STD);
                                     //envío de orden a módulo Xbee (librería XBEE_S2.C)
      while(tiempo++<5000){          //contador de tiempo
         delay_ms(1);
         switch(MenuXbee()){         //espera de respuesta
            case 1:return(0);break;  //respuesta deseada return 1
            case 2:return(1);break;
            case 3:return(1);break;
            case 4:return(1);break;
            case 5:return(0);break;
            case 6:return(0);break;
         }
      }
      return(0);
   }
   if(comando==PETICION_MED){
      CommandTransmission(IDENTIDAD,id,PETICION_MED);
                                     //envío de orden a módulo Xbee (librería XBEE_S2.C)
      while(tiempo++<5000){          //contador de tiempo
         delay_ms(1);
         switch(MenuXbee()){         //espera de respuesta
            case 1:return(0);break;  //respuesta deseada return 1
            case 2:return(0);break;
            case 3:return(0);break;
            case 4:return(0);break;
            case 5:return(0);break;
            case 6:return(1);break;
         }
      }
      return(0);
   }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función DisplayLeds ++++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void DisplayLeds(long numero)
{
   if(numero>=0x0000&&numero<0x3E80){  //1er rango 0-25%
      OFF(LED1);delay_ms(250);TOG(LED1);
      OFF(LED2);
      OFF(LED3);
      OFF(LED4);
   }
   if(numero>=0x3E80&&numero<0x7D00){  //2do rango 25-50%
      ON(LED1);
      OFF(LED2);delay_ms(250);TOG(LED2);
                  OFF(LED3);
                  OFF(LED4);
   } »1
```

```
»1 if(numero>=0x7D00&&numero<0xBB80){  //3er rango 50-75%
      ON(LED1);
      ON(LED2);
      OFF(LED3);delay_ms(250);TOG(LED3);
      OFF(LED4);
   }
   if(numero>=0xBB80&&numero<0xFA00){  //4to rango 75-100%
      ON(LED1);
      ON(LED2);
      ON(LED3);
      OFF(LED4);delay_ms(250);TOG(LED4);
   } »2
```

```
»2 if(numero>=0xFA00){    //100% llena
      ON(LED1);
      ON(LED2);
      ON(LED3);
      ON(LED4);
   }
```

Código C 3 Reloj de Tiempo Real RTC_DS1307.C

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Reloj De Tiempo Real RTC_DS1307.C ++++++////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//++++++ Estructuras ++++++//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
typedef struct{
   BYTE segundo;
   BYTE minuto;
   BYTE hora;
   BYTE dia_mes;
   BYTE dia_sem;
   BYTE mes;
   BYTE ano;
}FECHA;

typedef struct{
   BYTE segundo;
   BYTE minuto;
   BYTE hora;
   BYTE dia_mes;
}TIEMPO;
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Funciones ++++++////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void WriteByteClock(int direc,int dato);   //escribir un byte
BYTE ReadByteClock(int direc);             //leer un byte
void WriteDate(FECHA f);                   //escribir fecha completa
FECHA ReadDate(void);                      //leer fecha completa
TIEMPO ReadTime(void);                     //leer tiempo
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función WriteByteClock ++++++///////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void WriteByteClock(int direc,int dato)
{
   i2c_start(I2C_RTC);          //inicio rutina
   i2c_write(I2C_RTC,0xd0);     //dirección del DS1307 modo Write
   i2c_write(I2C_RTC,direc);    //dirección del registro
   i2c_write(I2C_RTC,dato);     //byte de información
   i2c_stop(I2C_RTC);           //fin de rutina
   delay_ms(5);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReadByteClock ++++++////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
BYTE ReadByteClock(int direc)
{
   BYTE dato;
   i2c_start(I2C_RTC);          //inicio rutina
   i2c_write(I2C_RTC,0xd0);     //dirección del DS1307 modo write
   i2c_write(I2C_RTC,direc);    //dirección del registro
   i2c_start(I2C_RTC);          //reinicio
   i2c_write(I2C_RTC,0xd1);     //dirección del DS1307 modo read
   dato=i2c_read(I2C_RTC,0); //lectura de byte
   i2c_stop(I2C_RTC);           //fin de rutina
   delay_ms(5);
   return(dato);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReadTime ++++++/////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
TIEMPO ReadTime(void)
{
   TIEMPO t;
   i2c_start(I2C_RTC);          //inicio rutina
   i2c_write(I2C_RTC,0xd0);     //dirección del DS1307 modo write
   i2c_write(I2C_RTC,0x00);     //apuntador a registro 0x00
   i2c_start(I2C_RTC);          /condición de reinicio
   i2c_write(I2C_RTC,0xd1);     //dirección del DS1307 modo read
   t.segundo      =i2c_read(I2C_RTC,1);      //lectura
   t.minuto       =i2c_read(I2C_RTC,1);
   t.hora         =i2c_read(I2C_RTC,1);
                   i2c_read(I2C_RTC,1);
   t.dia_mes      =i2c_read(I2C_RTC,0);
   i2c_stop(I2C_RTC);           //fin de rutina
   delay_ms(5);
   return(t);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función writeDate ++++++////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void WriteDate(FECHA f)
{
   i2c_start(I2C_RTC);                 inicio rutina
   i2c_write(I2C_RTC,0xd0);            //dirección del DS1307 modo write
   i2c_write(I2C_RTC,0x00);            //dirección del registro
   i2c_write(I2C_RTC,0x80);            //paro del oscilador
   i2c_write(I2C_RTC,f.minuto);        //escritura fecha y hora.
   i2c_write(I2C_RTC,f.hora);
   i2c_write(I2C_RTC,f.dia_sem);
   i2c_write(I2C_RTC,f.dia_mes);
   i2c_write(I2C_RTC,f.mes);
   i2c_write(I2C_RTC,f.ano);
   i2c_write(I2C_RTC,0x10);            //habilita salida del oscilador
   i2c_stop(I2C_RTC);                  //fin de rutina
   delay_ms(5);
   WriteByteClock(0x00,f.segundo);     //inicio del segundero
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función ReadAllDate ++++++//////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
FECHA ReadDate(void)
{
   FECHA f;
   i2c_start(I2C_RTC);                           //inicio rutina
   i2c_write(I2C_RTC,0xd0);                      //dirección del DS1307 modo write
   i2c_write(I2C_RTC,0x00);                      //apuntador a registro 0x00
   i2c_start(I2C_RTC);                           //condición de reinicio
   i2c_write(I2C_RTC,0xd1);                      //dirección del DS1307 modo read
   f.segundo      =i2c_read(I2C_RTC,1);          //lectura fecha y hora.
   f.minuto       =i2c_read(I2C_RTC,1);
   f.hora         =i2c_read(I2C_RTC,1);
   f.dia_sem      =i2c_read(I2C_RTC,1);
   f.dia_mes      =i2c_read(I2C_RTC,1);
   f.mes          =i2c_read(I2C_RTC,1);
   f.ano          =i2c_read(I2C_RTC,0);
   i2c_stop(I2C_RTC);                            //fin de rutina
   delay_ms(5);
   return(f);
}
////////////////////////////////////////////////////////////////////////////////
```

```c
////////////////////////////////////////////////////////////////////////////////
//++++++ Módulo Inalámbrico XBEE_S2.C ++++++////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//++++++ Librerías ++++++///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
#include <stdlibm.h> //cabecera para uso de malloc (memoria dinámica)
////////////////////////////////////////////////////////
//++++++ Estructuras y uniones ++++++///////////////////
////////////////////////////////////////////////////////
typedef union{
   float f;
   BYTE b[0x04];
}DATO;

typedef union{
   int32 i;
   BYTE b[0x04];
}IDXBEE;

typedef struct{
   float V;
   float I;
   float FP;
   float E;
   float W;
}TRAMA;
```

```c
////////////////////////////////////////////////////////
//++++++ Definiciones ++++++////////////////////////////
////////////////////////////////////////////////////////
//***comandos API Xbee***
#define AC          0x4143
#define DB          0x4442
#define AP          0x4150
#define ID          0x4944
#define CH          0x4348
#define SH          0x5348
#define SL          0x534C
#define DH          0x4448
#define DL          0x444C
#define AO          0x414F
//***comandos concentrador-medidor***
#define PETICION_STD      0x30
#define PETICION_MED      0x31
#define STD_OK            0x32
#define STD_ERROR         0x33
#define STD_OCUPADO       0x34
#define MED_NUEVO         0x35
#define MED_ENVIO         0x36
```

```c
////////////////////////////////////////////////////////
//++++++ Funciones +++++////////////////////////////////
////////////////////////////////////////////////////////
//***transmición***
void ATCommand(long at);
void WriteATCommand(long at,int32 valor);
void CommandTransmission(int32 id,int32 destino,BYTE comando);
void FrameTransmission(int32 id,int32 destino,TRAMA tr);

//***recepción***
void IntXbee(void);
int DecodingFunction(BYTE *arr);
int ATCR(BYTE *arr);          //AT Command Response
int TS(BYTE *arr);            //Transmit Starus
int TP(BYTE *arr);            //Transmit Packet
////////////////////////////////////////////////////////
//++++++ Variables globales ++++////////////////////////
////////////////////////////////////////////////////////
BYTE ESTADO;                  //tipo de mensaje
int32 IDENTIDAD;              //ID de modulo
int32 NUEVA_ID;               //ID auxiliar de receptor/emisor
TRAMA VALORES;                //mediciones
```

```c
////////////////////////////////////////////////////////
//++++++ Función ATCommand ++++++///////////////////////
////////////////////////////////////////////////////////
void ATCommand(long at)
{
   BYTE checksum=0;
   checksum=0xFF-(0x09+(at>>8)+at);
   putc(0x7E);        //byte de inicio
   putc(0x00);        //número de bytes high
   putc(0x04);        //número de bytes low
   putc(0x08);        //tipo de trama(comandos AT)
   putc(0x01);        //ack o no ack
   putc(at>>8);       //comando AT High
   putc(at);          //comando AT low
   putc(checksum);    //checksum
}
```

```c
////////////////////////////////////////////////////////
//++++++ Función WriteATCommand ++++++//////////////////
////////////////////////////////////////////////////////
void WriteATCommand(long at,int32 valor)
{
   BYTE checksum=0;
   checksum=0xFF-(0x0A+(at>>8)+(at)+
                (valor>>24)+(valor>>16)+(valor>>8)+valor);
   putc(0x7E);        //byte de inicio
   putc(0x00);        //número de bytes high
   putc(0x08);        //número de bytes low
   putc(0x09);        //tipo de trama(comandos at)
   putc(0x01);        //ack o no ack
   putc(at>>8);       //comando at HIGH
    putc(at);         //comando at LOW
    putc(valor>>24);  //valor 4
    putc(valor>>16);  //valor 3
    putc(valor>>8);   //valor 2
    putc(valor);      //valor 1
    putc(checksum);//checksum
}
```

```c
////////////////////////////////////////////////////////
//++++++ Función CommandTransmission ++++++/////////////
////////////////////////////////////////////////////////
void CommandTransmission(int32 id,int32 destino,BYTE comando)
{
    BYTE checksum; IDXBEE aux; int i;
    checksum=(0x02C3+(destino>>24)+(destino>>16)+(destino>>8)+destino+comando);
    putc(0x7E);               //byte de inicio
    putc(0x00);               //número de bytes high
    putc(0x13);               //número de bytes low
    putc(0x10);               //tipo de trama
    putc(0x01);               //ack o no ack
    putc(0x00);               //address high 4
    putc(0x13);               //address high 3
    putc(0xA2);               //address high 2
    putc(0x00);               //address high 1
    putc(destino>>24);        //address low 4
    putc(destino>>16);        //address low 3
    putc(destino>>8);         //address low 2
    putc(destino);            //address low 1
    putc(0xFF);               //reservado
    putc(0xFE);               //reservado
    putc(0x00);               //broadcast radius
    putc(0x00);               //opción de trasmisión
    putc(comando);            //comando de usuario
    aux.i=id;                 //ID remitente
    for(i=0;i<4;i++){
       checksum+=aux.b[i]; putc(aux.b[i]);}
    putc(0xFF-checksum);      //checksum
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función FrameTransmission ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void FrameTransmission(int32 id,int32 destino, TRAMA tr)
{
   BYTE checksum;
   IDXBEE aux_id;
   DATO aux;
   int i;
   checksum=(0x2C3+(destino>>24)+(destino>>16)+(destino>>8)+destino+MED_ENVIO);
   putc(0x7E);                  //byte de inicio
   putc(0x00);                  //número de bytes high
   putc(0x27);                  //número de bytes low
   putc(0x10);                  //tipo de trama
   putc(0x01);                  //ack o no ack
   putc(0x00);                  //address high 4
   putc(0x13);                  //address high 3
   putc(0xA2);                  //address high 2
   putc(0x00);                  //address high 1
   putc(destino>>24);           //address low 4
   putc(destino>>16);           //address low 3
   putc(destino>>8);            //address low 2
   putc(destino);               //address low 1
   putc(0xFF);                  //reservado
   putc(0xFE);                  //reservado
   putc(0x00);                  //broadcast radius
   putc(0x00);                  //opcion de trasmición
   putc(MED_ENVIO);             //comando de usuario
   aux_id.i=id;                 //ID remitente
   for(i=0;i<4;i++){
     checksum+=aux_id.b[i];
     putc(aux_id.b[i]);}
   aux.f=tr.V;                  //voltaje
   for(i=0;i<4;i++){
     checksum+=aux.b[i];
     putc(aux.b[i]);}
   aux.f=tr.I;                  //corriente
   for(i=0;i<4;i++){
     checksum+=aux.b[i];
     putc(aux.b[i]);}
   aux.f=tr.FP;                 //factor de potencia
   for(i=0;i<4;i++){
     checksum+=aux.b[i];
     putc(aux.b[i]);}
   aux.f=tr.W;                  //potencia
   for(i=0;i<4;i++){
     checksum+=aux.b[i];
     putc(aux.b[i]);}
   aux.f=tr.E;                  //energía
     for(i=0;i<4;i++){
     checksum+=aux.b[i];
     putc(aux.b[i]);}
  putc(0xFF-checksum);          //checksum
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ IntXbee ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void IntXbee(void)
{
  static BYTE bandera;          //variables locales
  static BYTE contador;
  static BYTE numero;
  static BYTE *arreglo;
  BYTE c;
  if(kbhit()){                                  //si Byte recibido
    c=getc();                                   //leer Byte en C
    if(bandera==0)                              //si C es primer Byte de mensaje (bandera 0)
      if(c==0x7E){                              //confirmación primer byte 0x7E
        bandera=1;                              //cambio de bandera
        return;}
    if(bandera==1)                              //si C es tamaño de mensaje (bandera 1)
      if(c!=0x7E&&c!=0x00){
        bandera=2;                                                  //cambio de bandera
        numero=c+1;                                                 //C es el tamaño de mensaje
        arreglo=(BYTE*)calloc(numero,sizeof(BYTE));                 //creación arreglo dinámico
        return;}
   if(bandera==2){                                  //si C es Byte de mensaje (bandera 2)
      arreglo[contador++]=c;                        //guardar Byte en arreglo dinámico
      if(contador>=numero){                         //si C es ultimo Byte de mensaje
        ESTADO=DecodingFunction(arreglo);           //decodificar mensaje
        free(arreglo);                              //liberar arreglo dinámico
        contador=0;                                 //reiniciar bandera y contador local
        bandera=0;
        return;}
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función DecodingFunction ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int DecodingFunction(BYTE *arr)
{
   switch(arr[0x00])
   {
     case 0x88:     return(ATCR(arr));  //respuesta de comandos at
                    break;
     case 0x8A:     return(0x10);       //estado de modem
                    break;
     case 0x8B:     return(TS(arr));    //estado de transmisión
                    break;
      case 0x90:    return(TP(arr));    //recepción de paquete
                    break;
      case 0x97:    return(0x40);       //respuesta de comando at remoto
                    break;
      case 0x91:    return(0x50);       //recepción de paquete con AO=1
                    break;
      case 0x92:    return(0x60);       //indicador de datos Rx
                    break;
      case 0x95:    return(0x70);       //indicador de identificación de nodo
            break;
   }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función ATCR ++++++////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int ATCR(BYTE *arr)             //AT Command Response
{
   IDXBEE aux;
   switch(arr[0x04])
   {
      case 0x00:     if(arr[0x02]=='S' && arr[0x03]=='L'){
                       aux.b[0x03]=arr[0x05];          // si la respuesta API es "SL"
                       aux.b[0x02]=arr[0x06];
                       aux.b[0x01]=arr[0x07];
                       aux.b[0x00]=arr[0x08];
                     }
                     IDENTIDAD=aux.i;  //guardar ID recibida en IDENTIDAD
                     return(0x00);          //OK
                     break;
      case 0x01:     return(0x01);          //ERROR
                     break;
      case 0x02:     return(0x02);          //COMANDO INVALIDO
                     break;
      case 0x03:     return(0x03);          //PARAMETRO INVALIDO
                     break;

   }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función TS ++++++////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int TS(BYTE *arr)    //Transmit Status
{
   switch(arr[0x05])
   {
      case 0x00:     if(arr[0x06]==0x00) return(0x20);          //Success and No Discovery Overhead
                     else                return(0x21);          //Success and Route Discovery
                     break;
      case 0x01:     if(arr[0x06]==0x00) return(0x22);          //MAC ACK Failure and No Discovery Overhead
                     else                return(0x23);          //MAC ACK Failure and Route Discovery
                     break;
      case 0x15:     if(arr[0x06]==0x00) return(0x24);          //Invalid destination endpoint and No Discovery
Overhead
                     else                return(0x25);          //Invalid destination endpoint and Route Discovery
                     break;
      case 0x21:     if(arr[0x06]==0x00) return(0x26);          //Network ACK Failure and No Discovery Overhead
                     else                return(0x27);          //Network ACK Failure and Route Discovery
                     break;
      case 0x25:     if(arr[0x06]==0x00) return(0x28);          //Route Not Found No Discovery Overhead
                     else                return(0x29);          //Route Not Found Route Discovery
                     break;
   }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función TP ++++++////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int TP(BYTE *arr)// Transmit Packet
{
   IDXBEE aux;
   DATO da;
   int i;
   BYTE j=0x0D;
   switch(arr[0x0C])
   {
      case PETICION_STD:                          //petición de estado medidor
                     return(PETICION_STD);        //retorno en ESTADO
                     break;
      case PETICION_MED:                          //petición de mediciones medidor
                     return(PETICION_MED);        //retorno en ESTADO
                     break;
      case STD_OK:                                //recepción de estado OK
                     for(i=0;i<4;i++)aux.b[i]=arr[j++];
                     NUEVA_ID=aux.i;              //guardar ID del remitente en NUEVA_ID
                     return(STD_OK);              //retorno en ESTADO
                     break;
     case STD_ERROR:                              //recepción de estado ERROR
                     for(i=0;i<4;i++)aux.b[i]=arr[j++];
                     NUEVA_ID=aux.i;              //guardar ID del remitente en NUEVA_ID
                     return(STD_ERROR);           //retorno en ESTADO
                     break;
     case STD_OCUPADO:                            //recepción de estado OCUPADO
                     for(i=0;i<4;i++)aux.b[i]=arr[j++];
                     NUEVA_ID=aux.i;   /          /guardar ID del remitente en NUEVA_ID
                     return(STD_OCUPADO);         //retorno en ESTADO
                     break;
          »1
```

```
»1   case MED_NUEVO:                              //recepción de ID medidor nuevo en red
                     for(i=0;i<4;i++)aux.b[i]=arr[j++];
                     NUEVA_ID=aux.i;              //guardar ID del remitente en NUEVA_ID
                     return(MED_NUEVO);           //retorno en ESTADO
                     break;
      case MED_ENVIO:      //recepción de mediciones
                     for(i=0;i<4;i++)aux.b[i]=arr[j++];
                     NUEVA_ID=aux.i;                          //ID remitente en NUEVA_ID
                     for(i=0;i<4;i++)da.b[i]=arr[j++];
                     VALORES.V=da.f;                          //voltaje VALORES.V
                     for(i=0;i<4;i++)da.b[i]=arr[j++];
                     VALORES.I=da.f;                          //corriente VALORES.I
                     for(i=0;i<4;i++)da.b[i]=arr[j++];
                     VALORES.FP=da.f;             //factor de potencia VALORES.FP
                     for(i=0;i<4;i++)da.b[i]=arr[j++];
                     VALORES.W=da.f;                          //potencia VALORES.W
                     for(i=0;i<4;i++)da.b[i]=arr[j++];
                     VALORES.E=da.f;                          //energía VALORES.E
                     return(MED_ENVIO);                       //retorno en ESTADO
                     break;
        }
      }
```

Código C 5 Memoria auxiliar MEMORIA_AUXILIAR_24FC1025.C

```c
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Memoria Auxiliar MEMORIA_AUXILIAR_24FC1025.C ++++++//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Definiciones ++++++////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
#define MAX_ID      0x3E7D    //0x3E7D, hasta 15,997 direcciones
/////////////////////////////////////////////////
//++++++ Estructuras ++++++/////////
/////////////////////////////////////////////////
typedef union{
   int32 i;
   BYTE b[0x04];
}IDXBEE;

typedef union{
   long i;
   BYTE b[0x02];
}ADDRESS;

typedef struct{
   long total;
   long inicio;
   long fin;
   long libre;
   long borrados;
}STATUS;
```

```c
/////////////////////////////////////////////////////////////////////////////
//++++++ Funciones +++++/////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////
short SaveNewIdXbee(int32 id);
short DeleteIdXbee(int32 id);
short IdXbeeSearch(STATUS st,int32 id,long &numero);
short IdXbeeSearchFast(int32 id);
long  MemoryAddress(long numero);
void  Save(long numero,int32 id);
int32 Load(long numero);
void  IdXbeeStartProtocol(long inicio);
int32 IdXbeeLoadProtocol(void);
void  IdXbeeEndProtocol(void);
STATUS LoadListStatus(void);
void   SaveListStatus(STATUS st);
void RebootListStatus(void);
void SaveAddressDeleted(long borrado,long numero);
long LoadAddressDeleted(long numero);
long LoadNumer(void);
void SaveNumber(long numero);
void RebootNumber(void);
```

```c
/////////////////////////////////////////////////////////////////
//++++++ Función Save ++++++/////////////////////////////
/////////////////////////////////////////////////////////////////
void Save(long numero,int32 id)
{      //guardar ID en la celda 'numero'
   long direc;
   IDXBEE aux;
   direc=MemoryAddress(numero);
   aux.i=id;
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,direc>>8);
   i2c_write(I2C_MEM_ID,direc);
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   i2c_write(I2C_MEM_ID,aux.b[0x02]);
   i2c_write(I2C_MEM_ID,aux.b[0x03]);
   i2c_stop(I2C_MEM_ID);
   delay_ms(5);
   return;
}
```

```c
/////////////////////////////////////////////////////////////////////
//++++++ Función Load ++++++/////////////////////////////////////
/////////////////////////////////////////////////////////////////////
int32 Load(long numero)
{      //leer ID de la celda 'numero'
   long direc;
   IDXBEE aux;
   direc=MemoryAddress(numero);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,direc>>8);
   i2c_write(I2C_MEM_ID,direc);
   i2c_stop(I2C_MEM_ID);
    i2c_start(I2C_MEM_ID);
    i2c_write(I2C_MEM_ID,0xA1);
    aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
    aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
    aux.b[0x02]=i2c_read(I2C_MEM_ID,1);
    aux.b[0x03]=i2c_read(I2C_MEM_ID,0);
    i2c_stop(I2C_MEM_ID);
    delay_ms(5);
    return(aux.i);   }
```

```c
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función SaveNewIdXbee ++++++//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
short SaveNewIdXbee(int32 id)
{
   STATUS st;
   long numero;
   st=LoadListStatus();                              //cargar estatus de lista
   if(IdXbeeSearch(st,id,numero)==1){                //buscar ID en lista
      return(0);
   }else if(st.borrados>0x0000){                               //si existe celda borrada disponible
         numero=LoadAddressDeleted(st.borrados);               //leer apuntador de celda disponible
         Save(numero,id);                                      //guardar ID en celda disponible
         st.total+=0x01;                             //incrementar número total de IDs
         st.borrados-=0x01;                          //borrar apuntador de celda ahora ocupada
         SaveListStatus(st);                         //salvar estatus de lista
         return(1);
         }else if(st.libre<=MAX_ID){                 //si se necesita una celda nueva
                  Save(st.libre,id);                 //guardar ID en siguiente celda libre
                  st.total+=0x01;                    //incrementar número total de IDs
                  st.fin=st.libre;                   //apuntador libre asignado a fin de lista
                  st.libre+=0x01;                    //incrementar el apuntador de libre
                  SaveListStatus(st);                //salvar estatus de lista
                  return(1);
            }
      }
}
```

```c
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función DeleteIdXbee ++++++///////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
short DeleteIdXbee(int32 id)
{
    STATUS st;
    long numero;
    st=LoadListStatus();                                     //cargar estatus de lista
    if(IdXbeeSearch(st,id,numero)==1){                       //buscar ID en lista
       Save(numero,0x00000000);                              //limpiar la celda
       st.borrados+=0x01;                                    //incrementar contador de celdas borradas
       SaveAddressDeleted(st.borrados,numero);               //guardar el apuntador de celda borrada
       st.total-=0x01;                                       //decremento de total IDs
       SaveListStatus(st);                                   //salvar el estatus de lista
       return(1);
    }
    return(0);
}
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función MemoryAddress ++++++//////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
long MemoryAddress(long numero)
{
   long direc;
   direc=numero*0x04;          //dirección según tamaño y número de celda
   direc+=0x0C;                //desplazar 10 bytes (para librar los índices iniciales)
   return(direc);              //retorna la dirección
}
////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función IdXbeeSearch ++++++//////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
short IdXbeeSearch(STATUS st,int32 id,long &numero)
{
   long ap;
   ap=st.inicio;                          //apuntador a inicio de lista
   if(st.total==0x0000){                  //si la lista está vacía
     return(0);
   }
   if(st.total>0x0000){                   //lista no vacía
     IdXbeeStartProtocol(st.inicio);      //iniciar protocolo de lectura
     do{
        if(id==IdXbeeLoadProtocol()){ //lectura y comparación de ID
           IdXbeeEndProtocol();          //fin de protocolo de lectura
           numero=ap;                    //posición de ID en lista
           return(1);
        }
        ap++;                       //incremento del apuntador
     }while(ap<=st.fin);            //si apuntador menor a fin de lista
     IdXbeeEndProtocol();           //fin de protocolo de lectura
     return(0);
   }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función IdXbeeSearchFast ++++++//////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
short IdXbeeSearchFast(int32 id)
{
   long ap;
   STATUS st;
   ap=st.inicio;                          //apuntador a inicio de lista
   if(st.total==0x0000){                  //si la lista está vacía
     return(0);
   }
   if(st.total>0x0000){                   //lista no vacía
     IdXbeeStartProtocol(st.inicio);      //iniciar protocolo de lectura
     do{
        if(id==IdXbeeLoadProtocol()){     //lectura y comparación de ID
           IdXbeeEndProtocol();           //fin de protocolo de lectura
           return(1);
        }
        ap++;                           //incremento del apuntador
     }while(ap<=st.fin);                //si apuntador menor fin de lista
     IdXbeeEndProtocol();               //fin de protocolo de lectura
     return(0);
   }
}
////////////////////////////////////////////////////////////////////////
//++++++ Función IdXbeeStartProtocol ++++++//
////////////////////////////////////////////////////////////////////////
void IdXbeeStartProtocol(long inicio)
{      //inicio de protocolo de lectura I2C
   long direc;
   direc=MemoryAddress(inicio);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,direc>>8);
   i2c_write(I2C_MEM_ID,direc);
   i2c_stop(I2C_MEM_ID);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA1);
}
////////////////////////////////////////////////////////////////////////
//++++++ Función IdXbeeLoadProtocol ++++++//
////////////////////////////////////////////////////////////////////////
int32 IdXbeeLoadProtocol(void)
{       //protocolo de lectura I2C
   IDXBEE aux;
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x02]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x03]=i2c_read(I2C_MEM_ID,1);
   return(aux.i);
}
////////////////////////////////////////////////////////////////////////
//++++++ Función IdXbeeEndProtocol ++++++//
////////////////////////////////////////////////////////////////////////
void IdXbeeEndProtocol(void)
{      //fin de protocolo de lectura I2C
   i2c_read(I2C_MEM_ID,0);
   i2c_stop(I2C_MEM_ID);}
////////////////////////////////////////////////////////////////////////
//++++++ Función LoadListStatus ++++++//////////
////////////////////////////////////////////////////////////////////////
STATUS LoadListStatus(void)
{         //cargar estatus de lista
   ADDRESS aux;
   STATUS st;        //variable tipo STATUS
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,0X00);
   i2c_write(I2C_MEM_ID,0x00);
   i2c_stop (I2C_MEM_ID);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA1);
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
   st.total=aux.i;
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
   st.inicio=aux.i;
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
   st.fin=aux.i;
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,1);
   st.libre=aux.i;
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,0);
   st.borrados=aux.i;
   i2c_stop(I2C_MEM_ID);
   delay_ms(5);
   return(st);
}
////////////////////////////////////////////////////////////////////////
//++++++ Función SaveListStatus ++++++//////////
////////////////////////////////////////////////////////////////////////
/
void SaveListStatus(STATUS st)
{            //salvar estatus de lista
   ADDRESS aux;
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,0X00);
   i2c_write(I2C_MEM_ID,0x00);
   aux.i=st.total;
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   aux.i=st.inicio;
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   aux.i=st.fin;
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   aux.i=st.libre;
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   aux.i=st.borrados;
    i2c_write(I2C_MEM_ID,aux.b[0x00]);
    i2c_write(I2C_MEM_ID,aux.b[0x01]);
    i2c_stop (I2C_MEM_ID);
    delay_ms(5);
return;
}
////////////////////////////////////////////////////////////////////////
//++++++ Función RebootListStatus ++++++//////////
////////////////////////////////////////////////////////////////////////
void RebootListStatus(void)
{            //reinicio de estatus de lista
   STATUS st;
   st.total=0x0000;
   st.borrados=0x0000;
   st.inicio=0x0000;
   st.fin=0x0000;
   st.libre=0x0000;
   SaveListStatus(st);
   return;
}
////////////////////////////////////////////////////////////////////////
//++++++ Función LoadAddressDeleted ++++++//
////////////////////////////////////////////////////////////////////////
long LoadAddressDeleted(long numero)
{      //lectura dirección de celdas borradas
   long direc;
   ADDRESS aux;
   direc=MemoryAddress(numero);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA8);
   i2c_write(I2C_MEM_ID,direc>>8);
   i2c_write(I2C_MEM_ID,direc);
   i2c_stop (I2C_MEM_ID);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA9);
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,0);
   i2c_stop (I2C_MEM_ID);
   delay_ms(5);
   return(aux.i);}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función SaveAddressDeleted ++++++//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void SaveAddressDeleted(long borrado,long numero)
{               //guardar dirección de celdas borradas
   long direc;
   ADDRESS aux;
   direc=MemoryAddress(borrado);
   aux.i=numero;
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA8);
   i2c_write(I2C_MEM_ID,direc>>8);
   i2c_write(I2C_MEM_ID,direc);
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   i2c_stop (I2C_MEM_ID);
   delay_ms(5);
   return;
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función LoadNumer ++++++///////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
long LoadNumer(void)
{               //leer número total de tramas en banco de memoria
   ADDRESS aux;
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,0X00);
   i2c_write(I2C_MEM_ID,0x0A);
   i2c_stop (I2C_MEM_ID);
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA1);
   aux.b[0x00]=i2c_read(I2C_MEM_ID,1);
   aux.b[0x01]=i2c_read(I2C_MEM_ID,0);
   i2c_stop(I2C_MEM_ID);
   delay_ms(5);
   return(aux.i);
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función SaveNumber ++++++//////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void SaveNumber(long numero)
{               //guardar número total de tramas en banco de memoria
    ADDRESS aux;
    aux.i=numero;
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,0X00);
   i2c_write(I2C_MEM_ID,0x0A);
   i2c_write(I2C_MEM_ID,aux.b[0x00]);
   i2c_write(I2C_MEM_ID,aux.b[0x01]);
   i2c_stop (I2C_MEM_ID);
   delay_ms(5);
   return;
}
```

```
////////////////////////////////////////////////////////////////////////////////
//++++++ Función RebootNumber ++++++////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
void RebootNumber(void)
{               //reiniciar número total de tramas en banco de memoria
   i2c_start(I2C_MEM_ID);
   i2c_write(I2C_MEM_ID,0xA0);
   i2c_write(I2C_MEM_ID,0X00);
   i2c_write(I2C_MEM_ID,0x0A);
   i2c_write(I2C_MEM_ID,0x00);
   i2c_write(I2C_MEM_ID,0x00);
   i2c_stop (I2C_MEM_ID);
   delay_ms(5);
   return;
}
////////////////////////////////////////////////////////////////////////////////
```

Código C 6 Banco de memoria BANCO_MEMORIA _24FC1025.C

```c
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Banco de memoria interna BANCO_MEMORIA_24FC1025.C ++++++//////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Definiciones ++++++////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
#define TRAMA_MAX 0xFA00
#define BLOCK1 0xA0    #define BLOCK2 0xA8    #define BLOCK3 0xA2    #define BLOCK4 0xAA    #define BLOCK5 0xA4    #define BLOCK6 0xAC    #define BLOCK7 0xA6    #define BLOCK8 0xAE
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Estructuras ++++++/////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
typedef union{                typedef union{                typedef struct{                typedef struct{
   float f;                      int32 i;                     TRAMA    Tr;                   BYTE canal;
   BYTE b[0x04];                 BYTE b[0x04];                long  Id;                     BYTE block;
}DATO;                        }IDXBEE;                        TIEMPO   Ti;                   long direc;
                                                            }DATAFRAME;                    }APUNTADOR;
```

```c
//////////////////////////////////////////////////////////////////////
//++++++ Funciones ++++++/////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
APUNTADOR MemoryAddressDecoder(long numero);
Void Decoder_2(long numero, BYTE block, BYTE direc)
long FourDataFramesSave(long numero,DATAFRAME *trama);
DATAFRAME SinglyDataFrameLoad (long numero);
void DataFrameSave_1(BYTE block,long direc,DATAFRAME *trama);
void DataFrameSave _2(BYTE block,long direc,DATAFRAME *trama);
void DataFrameSave _3(BYTE block,long direc,DATAFRAME *trama);
void DataFrameSave _4(BYTE block,long direc,DATAFRAME *trama);
DATAFRAME DataFrameLoad_1(BYTE block,long direc);
DATAFRAME DataFrameLoad_2(BYTE block,long direc);
DATAFRAME DataFrameLoad_3(BYTE block,long direc);
DATAFRAME DataFrameLoad_4(BYTE block,long direc);
```

```c
//////////////////////////////////////////////////////////////////////
//++++++ Función FourDataFramesSave ++++++////////////////////////////
//////////////////////////////////////////////////////////////////////
long FourDataFramesSave (long numero,DATAFRAME *trama)
{
   APUNTADOR ap;
   ap=MemoryAddressDecoder(numero);      //obtención de dirección física en banco de memoria
   switch (ap.canal){                    //coloca 4 tramas en memoria de forma consecutiva
      case 0x01:  DataFrameSave _1(ap.block,ap.direc,trama);      //guardar en memoria 1-4
               return(numero+4);                                  //avanzar cuatro celdas
               break;
      case 0x02:  DataFrameSave _2(ap.block,ap.direc,trama);      //guardar en memoria 4-8
               return(numero+4);                                  //avanzar cuatro celdas
               break;
      case 0x03:  DataFrameSave _3(ap.block,ap.direc,trama);      //guardar en memoria 8-12
               return(numero+4);                                  //avanzar cuatro celdas
               break;
      case 0x04:  DataFrameSave _4(ap.block,ap.direc,trama);      //guardar en memoria 12-16
               return(numero+4);                                  //avanzar cuatro celdas
               break;
   }
}
```

```c
////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ Función SinglyDataFrameLoad ++++++/////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
DATAFRAME SinglyDataFrameLoad (long numero)
{
   APUNTADOR ap;
   DATAFRAME trama;
   ap=MemoryAddressDecoder(numero);                      //obtención de dirección física en banco de memoria
   switch (ap.canal){                                    //ubica el canal
      case 0x01: trama=DataFrameLoad_1(ap.block,ap.direc);     //leer trama
                 return(trama);      break;
      case 0x02: trama=DataFrameLoad_2(ap.block,ap.direc);
                 return(trama);      break;
      case 0x03: trama=DataFrameLoad_3(ap.block,ap.direc);
                 return(trama);      break;
      case 0x04: trama=DataFrameLoad_4(ap.block,ap.direc);
                 return(trama)       break;
   }
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameSave_1 +++//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void DataFrameSave_1
(BYTE block,long direc,DATAFRAME *trama)
{   //guardar arreglo de 4 tramas para el canal 1
    int i,j;
    DATO dato;
    IDXBEE id;
    i2c_start(I2C_BANCO_M1);
    i2c_write(I2C_BANCO_M1,block);
    i2c_write(I2C_BANCO_M1,direc>>8);
    i2c_write(I2C_BANCO_M1,direc);
    for(i=0;i<4;i++){
        dato.f=trama[i].Tr.V;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,dato.b[j]);
        dato.f=trama[i].Tr.I;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,dato.b[j]);
        dato.f=trama[i].Tr.FP;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,dato.b[j]);
        dato.f=trama[i].Tr.E;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,dato.b[j]);
        dato.f=trama[i].Tr.W;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,dato.b[j]);
        id.i=trama[i].Id;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M1,id.b[j]);
i2c_write(I2C_BANCO_M1,trama[i].Ti.segundo);
i2c_write(I2C_BANCO_M1,trama[i].Ti.minuto);
i2c_write(I2C_BANCO_M1,trama[i].Ti.hora);
i2c_write(I2C_BANCO_M1,trama[i].Ti.dia_mes);
        for(j=0;j<4;j++)
        i2c_write(I2C_BANCO_M1,0x00);
    }
    i2c_stop(I2C_BANCO_M1);
    delay_ms(5);
    return;
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameSave_2 +++//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void DataFrameSave_2
(BYTE block,long direc,DATAFRAME *trama)
{   //guardar arreglo de 4 tramas para el canal 2
    int i,j;
    DATO dato;
    IDXBEE id;
    i2c_start(I2C_BANCO_M2);
    i2c_write(I2C_BANCO_M2,block);
    i2c_write(I2C_BANCO_M2,direc>>8);
    i2c_write(I2C_BANCO_M2,direc);
    for(i=0;i<4;i++){
        dato.f=trama[i].Tr.V;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,dato.b[j]);
        dato.f=trama[i].Tr.I;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,dato.b[j]);
        dato.f=trama[i].Tr.FP;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,dato.b[j]);
        dato.f=trama[i].Tr.E;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,dato.b[j]);
        dato.f=trama[i].Tr.W;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,dato.b[j]);
        id.i=trama[i].Id;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M2,id.b[j]);
i2c_write(I2C_BANCO_M2,trama[i].Ti.segundo);
i2c_write(I2C_BANCO_M2,trama[i].Ti.minuto);
i2c_write(I2C_BANCO_M2,trama[i].Ti.hora);
i2c_write(I2C_BANCO_M2,trama[i].Ti.dia_mes);
        for(j=0;j<4;j++)
        i2c_write(I2C_BANCO_M2,0x00);
    }
    i2c_stop(I2C_BANCO_M2);
    delay_ms(5);
    return;
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameSave_3 +++//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void DataFrameSave_3
(BYTE block,long direc,DATAFRAME *trama)
{   //guardar arreglo de 4 tramas para el canal 3
    int i,j;
    DATO dato;
    IDXBEE id;
    i2c_start(I2C_BANCO_M3);
    i2c_write(I2C_BANCO_M3,block);
    i2c_write(I2C_BANCO_M3,direc>>8);
    i2c_write(I2C_BANCO_M3,direc);
    for(i=0;i<4;i++){
        dato.f=trama[i].Tr.V;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,dato.b[j]);
        dato.f=trama[i].Tr.I;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,dato.b[j]);
        dato.f=trama[i].Tr.FP;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,dato.b[j]);
        dato.f=trama[i].Tr.E;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,dato.b[j]);
        dato.f=trama[i].Tr.W;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,dato.b[j]);
        id.i=trama[i].Id;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M3,id.b[j]);
i2c_write(I2C_BANCO_M3,trama[i].Ti.segundo);
i2c_write(I2C_BANCO_M3,trama[i].Ti.minuto);
i2c_write(I2C_BANCO_M3,trama[i].Ti.hora);
i2c_write(I2C_BANCO_M3,trama[i].Ti.dia_mes);
        for(j=0;j<4;j++)
        i2c_write(I2C_BANCO_M3,0x00);
    }
    i2c_stop(I2C_BANCO_M3);
    delay_ms(5);
    return;
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameSave_4 +++//////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
void DataFrameSave_4
(BYTE block,long direc,DATAFRAME *trama)
{   //guardar arreglo de 4 tramas para el canal 4
    int i,j;
    DATO dato;
    IDXBEE id;
    i2c_start(I2C_BANCO_M4);
    i2c_write(I2C_BANCO_M4,block);
    i2c_write(I2C_BANCO_M4,direc>>8);
    i2c_write(I2C_BANCO_M4,direc);
    for(i=0;i<4;i++){
        dato.f=trama[i].Tr.V;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,dato.b[j]);
        dato.f=trama[i].Tr.I;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,dato.b[j]);
        dato.f=trama[i].Tr.FP;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,dato.b[j]);
        dato.f=trama[i].Tr.E;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,dato.b[j]);
        dato.f=trama[i].Tr.W;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,dato.b[j]);
        id.i=trama[i].Id;
        for(j=0;j<4;j++)
            i2c_write(I2C_BANCO_M4,id.b[j]);
i2c_write(I2C_BANCO_M4,trama[i].Ti.segundo);
i2c_write(I2C_BANCO_M4,trama[i].Ti.minuto);
i2c_write(I2C_BANCO_M4,trama[i].Ti.hora);
i2c_write(I2C_BANCO_M4,trama[i].Ti.dia_mes);
        for(j=0;j<4;j++)
        i2c_write(I2C_BANCO_M4,0x00);
    }
    i2c_stop(I2C_BANCO_M4);
    delay_ms(5);
    return;
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameLoad_1 +++//////
////////////////////////////////////////////////////////////////////////////
DATAFRAME DataFrameLoad_1
(BYTE block,long direc)
{   // I2C para lectura canal 1
    int i;
    DATO dato;
    IDXBEE id;
    DATAFRAME trama;
    i2c_start(I2C_BANCO_M1);
    i2c_write(I2C_BANCO_M1,block);
    i2c_write(I2C_BANCO_M1,direc>>8);
    i2c_write(I2C_BANCO_M1,direc);
    i2c_stop (I2C_BANCO_M1);
    i2c_start(I2C_BANCO_M1);
    i2c_write(I2C_BANCO_M1,(block+0x01));
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Tr.V=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Tr.I=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Tr.FP=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Tr.E=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Tr.W=dato.f;
    for(i=0;i<4;i++)
       id.b[i]=i2c_read(I2C_BANCO_M1,1);
    trama.Id=id.i;
trama.Ti.segundo=i2c_read(I2C_BANCO_M1,1);
trama.Ti.minuto   =i2c_read(I2C_BANCO_M1,1);
trama.Ti.hora     =i2c_read(I2C_BANCO_M1,1);
trama.Ti.dia_mes=i2c_read(I2C_BANCO_M1,0);
    i2c_stop(I2C_BANCO_M1);
    delay_ms(5);
    return(trama);
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameLoad_2 +++//////
////////////////////////////////////////////////////////////////////////////
DATAFRAME DataFrameLoad_2
 (BYTE block,long direc)
{   // I2C para lectura canal 2
    int i;
    DATO dato;
    IDXBEE id;
    DATAFRAME trama;
    i2c_start(I2C_BANCO_M2);
    i2c_write(I2C_BANCO_M2,block);
    i2c_write(I2C_BANCO_M2,direc>>8);
    i2c_write(I2C_BANCO_M2,direc);
    i2c_stop (I2C_BANCO_M2);
    i2c_start(I2C_BANCO_M2);
    i2c_write(I2C_BANCO_M2,(block+0x01));
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Tr.V=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Tr.I=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Tr.FP=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Tr.E=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Tr.W=dato.f;
    for(i=0;i<4;i++)
       id.b[i]=i2c_read(I2C_BANCO_M2,1);
    trama.Id=id.i;
trama.Ti.segundo=i2c_read(I2C_BANCO_M2,1);
trama.Ti.minuto   =i2c_read(I2C_BANCO_M2,1);
trama.Ti.hora     =i2c_read(I2C_BANCO_M2,1);
trama.Ti.dia_mes=i2c_read(I2C_BANCO_M2,0);
    i2c_stop(I2C_BANCO_M2);
    delay_ms(5);
    return(trama);
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameLoad_3 +++//////
////////////////////////////////////////////////////////////////////////////
DATAFRAME DataFrameLoad_3
(BYTE block,long direc)
{   // I2C para lectura canal 3
int i;
    DATO dato;
    IDXBEE id;
    DATAFRAME trama;
    i2c_start(I2C_BANCO_M3);
    i2c_write(I2C_BANCO_M3,block);
    i2c_write(I2C_BANCO_M3,direc>>8);
    i2c_write(I2C_BANCO_M3,direc);
    i2c_stop (I2C_BANCO_M3);
    i2c_start(I2C_BANCO_M3);
    i2c_write(I2C_BANCO_M3,(block+0x01));
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Tr.V=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Tr.I=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Tr.FP=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Tr.E=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Tr.W=dato.f;
    for(i=0;i<4;i++)
       id.b[i]=i2c_read(I2C_BANCO_M3,1);
    trama.Id=id.i;
trama.Ti.segundo=i2c_read(I2C_BANCO_M3,1);
trama.Ti.minuto   =i2c_read(I2C_BANCO_M3,1);
trama.Ti.hora     =i2c_read(I2C_BANCO_M3,1);
trama.Ti.dia_mes=i2c_read(I2C_BANCO_M3,0);
    i2c_stop(I2C_BANCO_M3);
    delay_ms(5);
    return(trama);
}
```

```
////////////////////////////////////////////////////////////////////////////
//+++ Función DataFrameLoad_4 +++//////
////////////////////////////////////////////////////////////////////////////
DATAFRAME DataFrameLoad_4
BYTE block,long direc)
{   // I2C para lectura canal 4
    int i;
    DATO dato;
    IDXBEE id;
    DATAFRAME trama;
    i2c_start(I2C_BANCO_M4);
    i2c_write(I2C_BANCO_M4,block);
    i2c_write(I2C_BANCO_M4,direc>>8);
    i2c_write(I2C_BANCO_M4,direc);
    i2c_stop (I2C_BANCO_M4);
    i2c_start(I2C_BANCO_M4);
    i2c_write(I2C_BANCO_M4,(block+0x01));
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Tr.V=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Tr.I=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Tr.FP=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Tr.E=dato.f;
    for(i=0;i<4;i++)
       dato.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Tr.W=dato.f;
    for(i=0;i<4;i++)
       id.b[i]=i2c_read(I2C_BANCO_M4,1);
    trama.Id=id.i;
trama.Ti.segundo=i2c_read(I2C_BANCO_M4,1);
trama.Ti.minuto   =i2c_read(I2C_BANCO_M4,1);
trama.Ti.hora     =i2c_read(I2C_BANCO_M4,1);
trama.Ti.dia_mes=i2c_read(I2C_BANCO_M4,0);
    i2c_stop(I2C_BANCO_M4);
    delay_ms(5);
    return(trama);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//+++ Función MemoryAddressDecoder +++//////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
APUNTADOR MemoryAddressDecoder(long numero)
{                                          //recibe un número y retorna su dirección física en memoria
   APUNTADOR ap;                           //ubica el canal definido por rangos
   if(numero>=0x0000&&numero<0x3E80){      //si 0>=numero<16000 canal 1
      ap.canal=1;                          //asignación de canal
      Decoder_2(numero,ap.block,ap.direc)  //buscar bloque y dirección
      return;
   }
   if(numero>=0x3E80&&numero<0x7D00){      //si 16000>=numero<32000 canal 2
      numero-=0x3E80;                      //resta el valor inferior
      ap.canal=2;                          //asignación de canal
      Decoder_2(numero,ap.block,ap.direc)  //buscar bloque y dirección
      return;
   }
   if(numero>=0x7D00&&numero<0xBB80){      //si 32000>=numero<48000 canal 3
      numero-=0x7D00;                      //resta el valor inferior
      ap.canal=3;                          //asignación de canal
      Decoder_2(numero,ap.block,ap.direc)  //buscar bloque y dirección
      return;
   }
   if(numero>=0xBB80&&numero<0xFA00){      //si 48000>=numero<64000 canal 4
      numero-=0xBB80;                      //resta el valor inferior
      ap.canal=4;                          //asignación de canal
      Decoder_2(numero,ap.block,ap.direc)  //buscar bloque y dirección
      return;
   }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//+++ Función Decoder_2 +++/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Decoder_2(long numero,long &block,long &direc)
{                                          // Retorna el bloque y la dirección del número dado
   if(numero>=0x0000&&numero<0x0FA0){
      if(numero>=0x0000&&numero<0x07D0){
         block=BLOCK1;
         direc=numero*0x20;
         return;
      }
      if(numero>=0x07D0&&numero<0x0FA0){
         block=BLOCK2;
         direc=(numero-0x07D0)*0x20;
         return;
      }
   }
   if(numero>=0x0FA0&&numero<0x1F40){
      numero-=0x0FA0;
      if(numero>=0x0000&&numero<0x07D0){
         block=BLOCK3;
         direc=numero*0x20;
         return;
      }
      if(numero>=0x07D0&&numero<0x0FA0){
         block=BLOCK4;
         direc=(numero-0x07D0)*0x20;
         return;
      }
   }
   if(numero>=0x1F40&&numero<0x2EE0){
      numero-=0x1F40;
      if(numero>=0x0000&&numero<0x07D0){
         block=BLOCK5;
         direc=numero*0x20;
         return();
      }
      if(numero>=0x07D0&&numero<0x0FA0){
         block=BLOCK6;
         direc=(numero-0x07D0)*0x20;
         return();
      }
   }
   if(numero>=0x2EE0&&numero<0x3E80){
      numero-=0x2EE0;
      if(numero>=0x0000&&numero<0x07D0){
         block=BLOCK7;
         direc=numero*0x20;
         return();
      }
      if(numero>=0x07D0&&numero<0x0FA0){
         block=BLOCK8;
         direc=(numero-0x07D0)*0x20;
         return();
      }
   }
}
```

Código C 7 MCU medidor

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ MCU medidor +++++++////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ función Inico_Xbee ++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Inico_Xbee(void)
{
  WriteATCommand(ID,NETWORK);                                           //configuración de la LAN de red
  WriteATCommand(AO,0);                                                 //habilitar comandos API
  ATCommand(SL);                                                        //petición de ID del módulo propio
 CommandTransmission (IDENTIDAD,COORDINADOR, MED_NUEVO);                //envío de ID al concentrador como módulo nuevo
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ función Menu_Xbee +++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
 void Menu_Xbee(void)
{
  if(ESTADO>=0x30&&ESTADO<0x40){                                                        //estado de módulo solo de recepción de paquetes
    switch(ESTADO){
      case PETICION_STD:   CommandTransmission(IDENTIDAD,COORDINADOR,STD_OK);           //petición de estado, responder con OK
                           ESTADO=0x00;                                                 //mensaje recibido limpiar bandera de ESTADO
                           break;
      case PETICION_MED:   Medicion_Xbee();                                             //petición de mediciones, responder con envío de mediciones
                           ESTADO=0x00;
                           break;
      case STD_OK:         ESTADO=0x00;                                                 //otros estados ignorarlos
                           break;
      case STD_ERROR:      ESTADO=0x00;
    }
  }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//++++++ función Medicion_Xbee +++++++//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Medicion_Xbee(void)
{
   VALORES.V      =VOLTAJE;                //medidas obtenidas asignadas a la estructura global VALORES de la librería XBEE_S2.C
   VALORES.I      =CORRIENTE;
   VALORES.FP     =FACTOR_POTENCIA;
   VALORES.W      =POTENCIA;
   VALORES.E      =ENERGIA;
   FrameTransmission(IDENTIDAD,COORDINADOR,VALORES);      //función de envío de mediciones
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Bibliografía

1. **Adel S. Sedra, Kenneth C. Smith. 2007.** *Circuitos Microelectrónicos.* México : Mc Graw Hill, 2007.
2. **Aitzol Zuloaga Izaguirre, Armando Astarloa Cuéllar. 2008.** *Sistemas de Prcesamiento Digital.* Primera. Madrid (España) : Delta Publicaciones, 2008.
3. **Augenstein, Aaron M. Tenenbaum Yedidyah Langsam Moshe J. 1993.** *Estructuras de datos en C.* México : Prentice-Hall Hispanoamericana, 1993.
4. **Barcón, Santiago. 2011.** Comisión Nacional para el Uso Eficiente de la Energía (Conuee). [En línea] 2011. [Citado el: 10 de enero de 2014.] http://www.conuee.gob.mx/work/sites/CONAE/resources/LocalContent/7472/3/2011_09_26_AMESCO.pdf.
5. **Breijo, Eduardo García. 2009.** *Compilador C CCS y Simulador Proteus para Microcontroladores PIC.* Madrid (Eapaña) : MARCOMBO, 2009.
6. **Faludi, Robert. 2010.** *Building Wireless Sensor Networks.* L.A. (United States of America) : O'Reilly, 2010.
7. **José Rafael Lajara Vizcaíno, José Pelegrí Sebastiá. 2011.** *LabVIEW: Entorno gráfico de programación.* segunda. Barcelona (España) : MARCOMBO, 2011.
8. **Josep, Balcells. 1992.** *Interferencias Electromagnéticas en Sistemas Electrónicos .* Barcelona, España : MARCOMBO, 1992.
9. **Martin, James. 1987.** *Organización de las bases de datos.* México : Prentice-Hall Hispanoamericana, 1987.
10. **maximintegrated.com. 2015.** http://datasheets.maximintegrated.com/en/ds/DS1307.pdf. [En línea] 2015. [Citado el: 01 de 08 de 2015.]

11. **microchip.com. 2015.** http://ww1.microchip.com/downloads/en/DeviceDoc/20001941L.pdf. [En línea] 2015. [Citado el: 01 de 08 de 2015.]

12. **2010.** Protocolo de Kyoto. Secretaría del Medio Ambiente GDF. [En línea] 2010. [Citado el: 10 de enero de 2014.] http://www.sma.df.gob.mx/cclimatico/ciudadanos01_c.html.

13. **Rebolledo, Higinio Acoltzi Acoltzi y Hugo Pérez. 2011.** ISO 50001. Gestión de Energía. [En línea] junio de 2011. [Citado el: 10 de enero de 2014.] http://www.iie.org.mx/boletin042011/tecnico.pdf.