UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

COMPUTADORA DE VUELO PARA UN SISTEMA DE CAPACITACIÓN DE RECURSOS HUMANOS EN EL MANEJO DE SATÉLITES

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO ELÉCTRICO ELECTRÓNICO

P R E S E N T A :

GÓMEZ ISLAS FRANCISCO JAVIER

DIRECTOR:

DR. ESAÚ VICENTE VIVAS

México D.F. 2007

AGRADECIMIENTOS

A mis padres Elvia Islas Martínez y Javier Gómez Sánchez y mis hermanos Mariel Gómez Islas y Carlos Adrián Gómez Islas por el cariño y apoyo que me han brindado

A la UNAM por darme el privilegio de formar parte de su comunidad

A los profesores de la Facultad de Ingeniería de la UNAM por compartir sus conocimientos y experiencias

A mis compañeros: Emilio Jiménez, Rodrigo Alva, Zaira Carrizales, Rodrigo Córdoba y Roy Rodríguez

A los miembros del jurado: Ing. Roberto Macías Pérez, M.I. Antonio Salvá Calleja, Ing. Alejandro Sosa Fuentes y M.I. Ricardo Garibay Jiménez

Al Dr. Esaú Vicente Vivas por dirigir esta tesis

COMPUTADORA DE VUELO PARA UN SISTEMA DE CAPACITACIÓN DE RECURSOS HUMANOS EN EL MANEJO DE SATÉLITES

Índice

Capítulo 1
Sistema de capacitación de recursos humanos en el manejo de satélites

1.1 Introducción

En México se han desarrollado proyectos satelitales como los UNAMSAT A, UNAMSAT B y el microsatélite SATEX. Estos proyectos son importantes para la formación de ingenieros y para reducir la brecha tecnológica que se tiene con los países desarrollados. Es por esto que se quiere desarrollar un sistema de capacitación de recursos humanos en el manejo de satélites el cual no es un satélite para lanzamiento pero si tiene el diseño y módulos que normalmente se encuentran en un picosatélite (0.1Kg. a 1Kg.). El objetivo que se persigue al desarrollar este sistema es que pueda ser utilizado en centros educativos para entrenar a los estudiantes en la operación de un satélite.

En este capítulo se describen la arquitectura propuesta para el sistema de entrenamiento y cada subsistema que lo compone.

1.2 Arquitectura propuesta

El sistema de entrenamiento (ver figura 1.1) está formado por circuitos impresos de 8.9cm. x 8.9cm. conectados uno encima del otro. El sistema cuenta con tarjetas de potencia, comunicaciones, computadora de vuelo, estabilización y sensores.

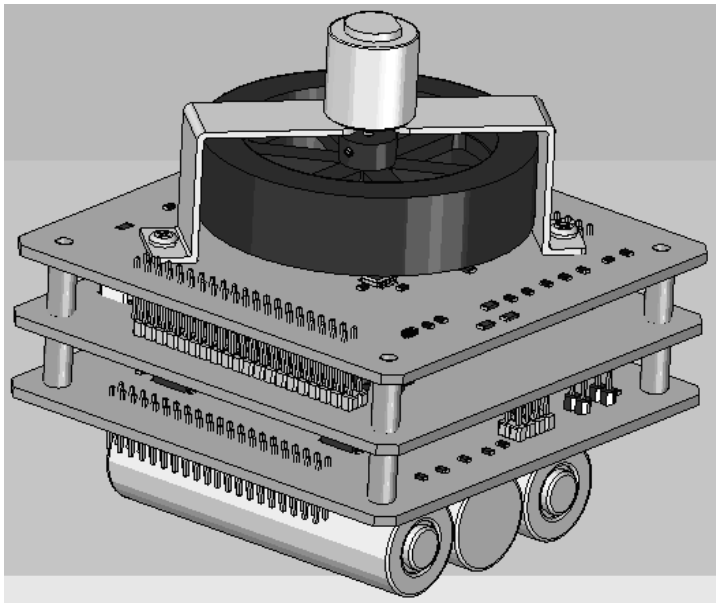


Figura 1.1: Arquitectura propuesta para el sistema de capacitación de recursos humanos en el manejo de satélites.

1.3 Subsistemas

1.3.1 Computadora de vuelo

La presente tesis trata sobre la computadora de vuelo la cual es la tarjeta electrónica que se encarga de controlar todas las tareas del sistema; entre ellas la recepción de comandos desde el software de PC, el envío de telemetría al software de PC, la comunicación por medio de comandos con los demás subsistemas, la ejecución de algoritmos de estabilización, el encendido o apagado de cualquiera de las tarjetas, el manejo de la memoria flash, la lectura de los sensores de temperatura y la lectura de la corriente eléctrica consumida por el sistema.

1.3.2 Subsistema de potencia

La tarjeta de potencia se encarga de energizar todo el sistema utilizando baterías de Litio recargables y celdas solares. Esta tarjeta puede desenergizar cualquier subsistema que le indique la computadora de vuelo y aloja la electrónica necesaria para interrumpir el suministro de energía en cuanto se presente el efecto latch-up en la computadora de vuelo. El efecto latch-up es un fenómeno que ocurre en los satélites reales y se manifiesta con el incremento en el consumo de corriente eléctrica cuando un dispositivo electrónico, como un microcontrolador, recibe radiación ionizante en exceso.

1.3.3 Subsistema de comunicaciones

El subsistema de comunicaciones es la tarjeta que cuenta con el transmisor y el receptor para que la computadora de vuelo se comunique inalámbricamente con el software de PC.

1.3.4 Subsistema de estabilización

Esta tarjeta cuenta con un microcontrolador que recibe comandos provenientes de la computadora de vuelo para controlar la velocidad de un motor de corriente directa y activar 6 bobinas de torque magnético durante la ejecución de algoritmos de estabilización. Para realizar pruebas de estabilización, el sistema se coloca en una estructura (ver figura 1.2) que consiste en tres anillos los cuales permiten que el sistema gire en cualquiera de los 3 ejes.

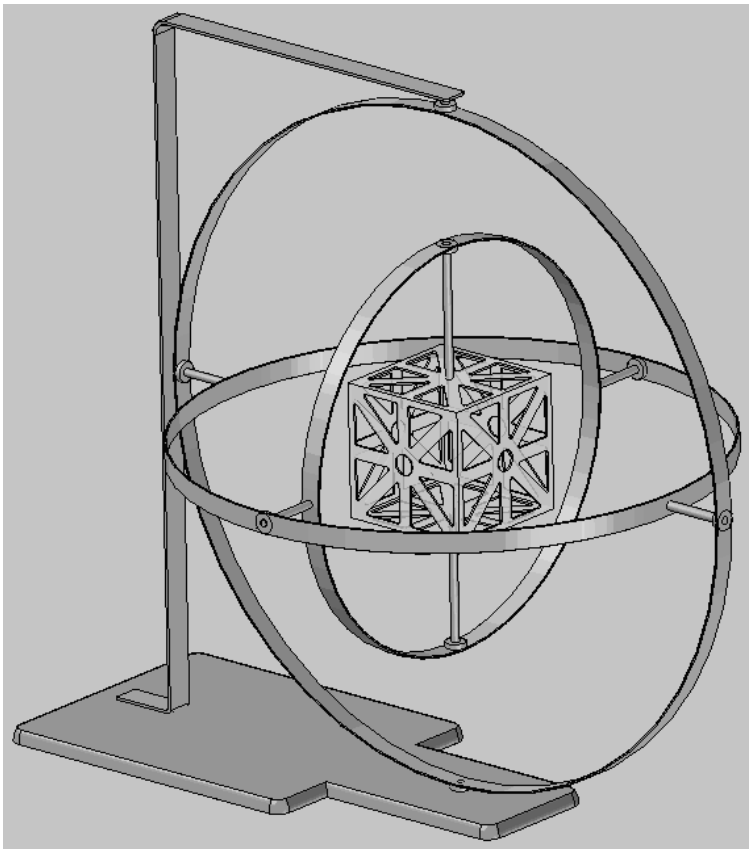


Figura 1.2: Estructura para realizar pruebas de estabilización.

1.3.5 Subsistema de sensores

En esta tarjeta hay 1 acelerómetro, 3 giróscopos, 1 brújula y un reloj en tiempo real cuya telemetría es enviada a la computadora de vuelo durante la ejecución de algoritmos de estabilización.

1.3.6 Software para la supervisión y comando desde la PC

Este software, también llamado “software de estación terrena”, se utiliza para cargar programas en la computadora de vuelo, para enviar comandos a la computadora y para recibir y desplegar la telemetría del sistema.

Capítulo 2 Computadora de vuelo

2.1 Introducción

La presente tesis trata sobre la computadora de vuelo la cual es la tarjeta electrónica que se encarga de controlar todas las tareas del sistema; entre ellas la recepción de comandos desde el software de PC, el envío de telemetría al software de PC, la comunicación por medio de comandos con los demás subsistemas, la ejecución de algoritmos de estabilización, el encendido o apagado de cualquiera de las tarjetas, el manejo de la memoria flash, la lectura de los sensores de temperatura y la lectura de la corriente eléctrica consumida por el sistema.

El objetivo de esta tesis es el diseño de la computadora de vuelo y el desarrollo del software para el cargado de programas, la lectura de los sensores de temperatura, el manejo de la memoria flash y la lectura del convertidor A/D. A partir de este software se podrá desarrollar la aplicación completa para la computadora de vuelo.

En este capítulo se describen las partes que componen a la computadora de vuelo.

2.2 Especificaciones y diagrama a bloques

La computadora de vuelo (ver figura 2.1) tiene una forma cuadrada de 8.9cm x 8.9cm con componentes en ambos lados. El microcontrolador principal es un SAB80C166 que tiene conectada una memoria RAM de 256 kBytes. En la computadora hay un microcontrolador PIC16F876A cuya función principal es activar el modo de cargar programa en el SAB80C166 a través de los pines NMI#, ALE y RSTIN#. La computadora cuenta con 32 MBytes de memoria flash, tres sensores de temperatura 1-Wire y dos multiplexores para conectar el puerto serial 1 del SAB80C166 con hasta 8 subsistemas. El SAB80C166, la memoria RAM y el PIC16F876A están protegidos contra efecto latch-up.

La computadora cuenta con dos conectores tarjeta a tarjeta (ver apéndice A.1 y A.2) de 40 pines cada uno. Estos conectores proporcionan los voltajes de alimentación para todo el sistema y permiten la comunicación de los subsistemas con la computadora de vuelo.

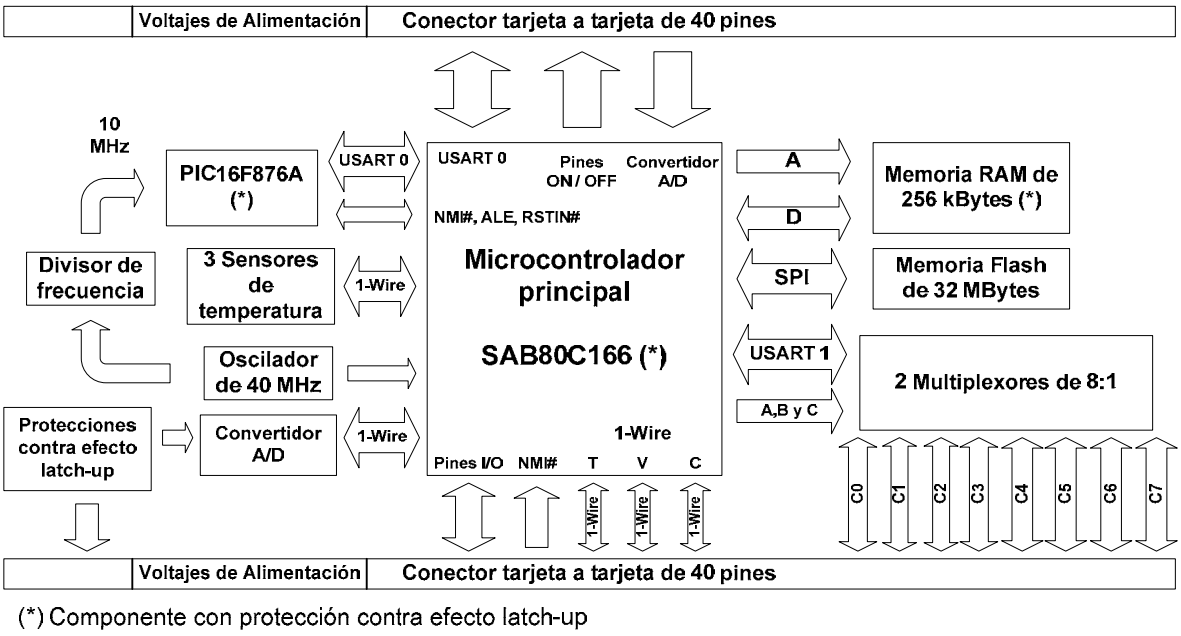


Figura 2.1: Diagrama de bloques de la computadora de vuelo.

2.3 Microcontrolador SAB80C166

2.3.1 Especificaciones

El microcontrolador SIEMENS SAB80C166 cuenta con un CPU de 16 bits, pipeline de 4 niveles, 100 ηs/instrucción con una señal de reloj de 40 MHz, memoria RAM interna de 1 kByte, capacidad de manejar memoria externa de hasta 256 kBytes en modo multiplexado y demultiplexado, sistema de interrupción con 16 niveles de prioridad, diez canales de convertidor A/D de 10 bits con tiempo de conversión de 9.7 us, unidad CAPCOM de 16 canales, 5 timers de propósito general, dos canales seriales USART, modo de cargar programas o “Bootstrap loader”, 76 pines de propósito general y encapsulado MQFP de 100 pines.

En la computadora de vuelo cada pin del SAB80C166 tiene una función específica (ver apéndice A.3) que puede ser el manejo de sensores 1-Wire, el manejo de la memoria flash, el apagado o encendido de alguna tarjeta del sistema o la selección de algún subsistema para comunicarse mediante el canal serial 1 (ver tabla 2.1).

C	B	A	Tarjeta
0	0	0	Estabilización
0	0	1	Sensores
0	1	0	Expansión 0
0	1	1	Expansión 1
1	0	0	Expansión 2
1	0	1	Expansión 3
1	1	0	Expansión 4
1	1	1	Expansión 5

Tabla 2.1: Valores de A, B y C en los multiplexores para tener comunicación con cada tarjeta del sistema mediante el canal serial 1 del SAB80C166.

2.3.2 Memoria RAM externa

El SAB80C166 de la computadora está conectado a 256 kBytes de memoria RAM externa (ver figura 2.2). Los pines EBC0 y EBC1 están conectados a 1 y el pin BUSACT# a 0 para así tener una configuración de bus externo demultiplexado. En el puerto 0 se encuentra el bus de datos de 16 bits y en el puerto 1 y 4 el bus de direcciones de 18 bits. La memoria RAM externa necesita de las señales WR# y RD# generadas por el SAB80C166 cuando escribe o lee la memoria.

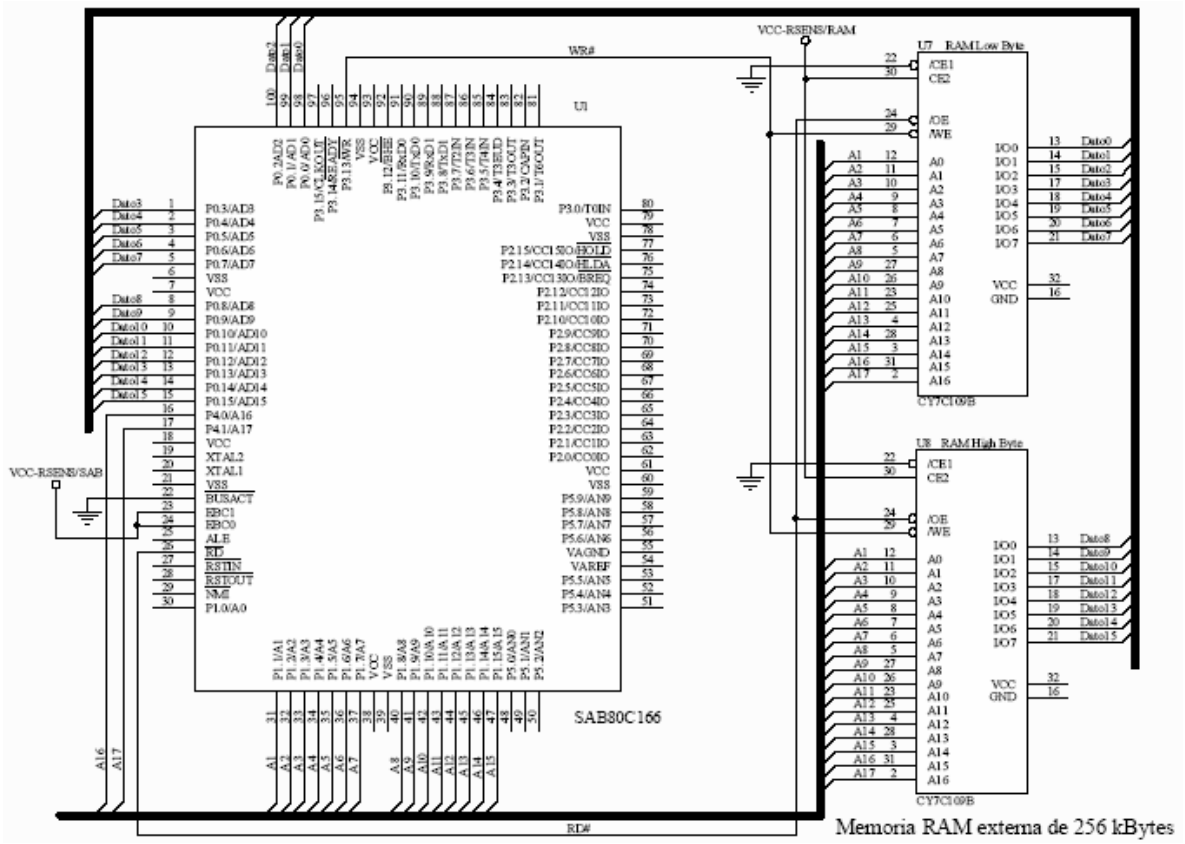


Figura 2.2: Conexión de la memoria RAM externa al SAB80C166.

Los 256 KBytes de memoria que puede direccionar el SAB80C166 están divididos en 4 segmentos de 64 KBytes cada uno (ver figura 2.3).

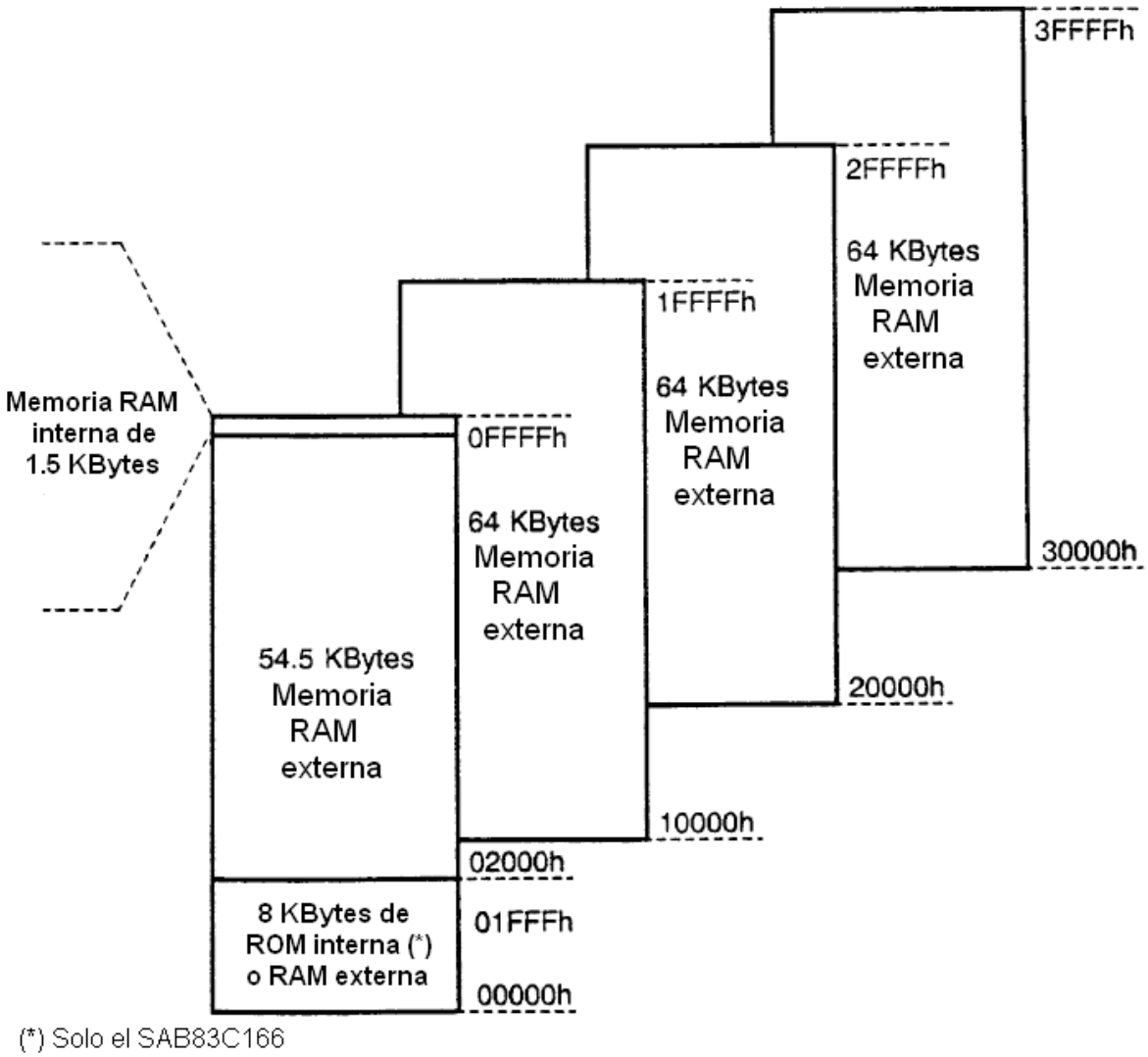


Figura 2.3: Organización de la memoria del SAB80C166.

Cada segmento de memoria está dividido en 4 páginas de 16 KBytes. El SAB80C166 cuenta con 4 registros llamados “data page pointers” que son utilizados implícitamente por el CPU para acceder a cualquier página de la memoria. Cada “data page pointer” es de 4 bits, los 2 bits más significativos representan el segmento y los 2 bits menos significativos la página a seleccionar dentro del segmento. Después de un reset los 4 “data page pointers” se ubican en el segmento 0 (ver figura 2.4). Si el usuario quiere tener acceso a otros segmentos de memoria tiene que modificar el valor de los “data page pointers”.

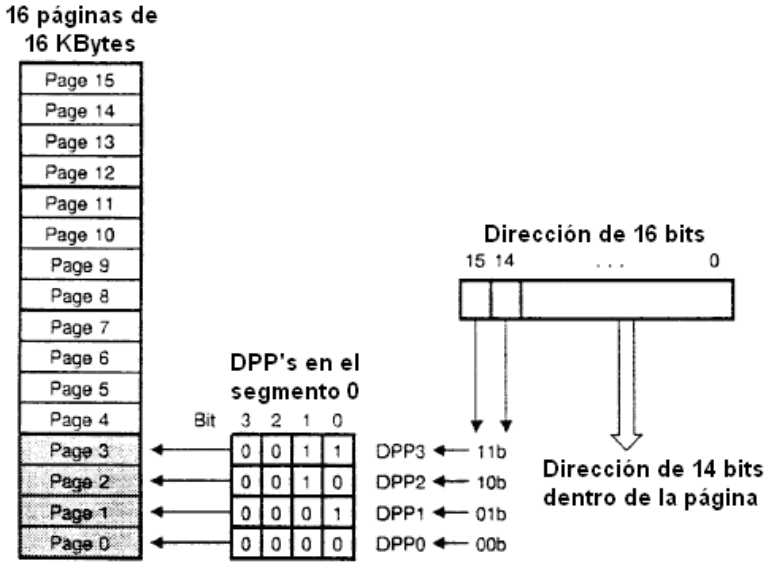


Figura 2.4: “Data page pointers” en el segmento 0 de la memoria.

En el segmento 0 se tienen 1.5KBytes de memoria RAM interna (ver figura 2.5) en la cual se encuentran todos los registros de funciones especiales, espacio para software de usuario, para el precargador, para el stack y para los registros de propósito general.

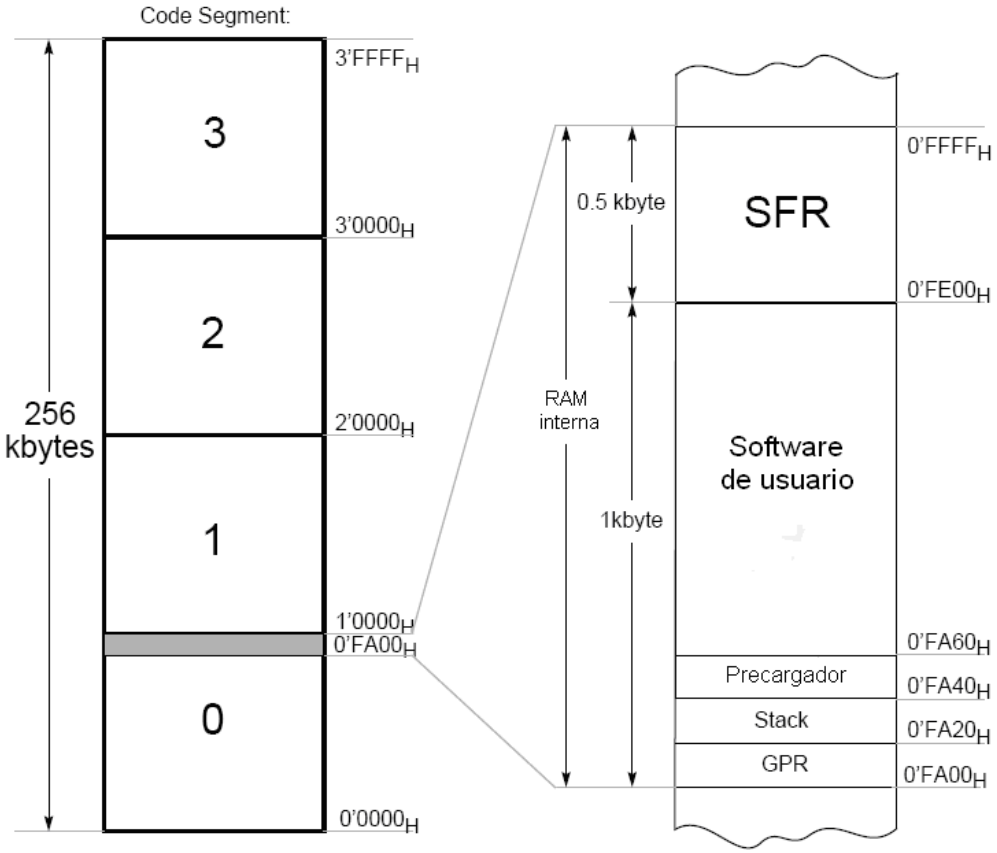


Figura 2.5: Memoria RAM interna del SAB80C166.

2.3.3 Modo de cargar programa

Para cargar software en el SAB80C166 es necesario activar el modo de cargar programa utilizando el PIC16F876A. El modo de cargar programa (ver figura 2.6) se activa cuando durante una señal de reset por hardware el pin ALE se encuentra en 1 y el pin NMI# cambia de 1 a 0. En modo de cargar programa el SAB80C166 espera por el canal de comunicaciones 0 el byte 00H con un bit de inicio y uno de paro. Este byte 00H le sirve al SAB80C166 para calcular el baudaje del dispositivo que le va a enviar el software. Una vez recibido el byte 00H, el SAB80C166 envía como respuesta el byte 55H y espera un programa de 32 bytes que se guarda en las localidades 0FA40H a 0FA5FH. El programa de 32 bytes o precargador se ejecuta automáticamente después de ser recibido. Este programa de 32 bytes puede ser utilizado para cargar un programa de hasta 256 kBytes en la memoria RAM externa.

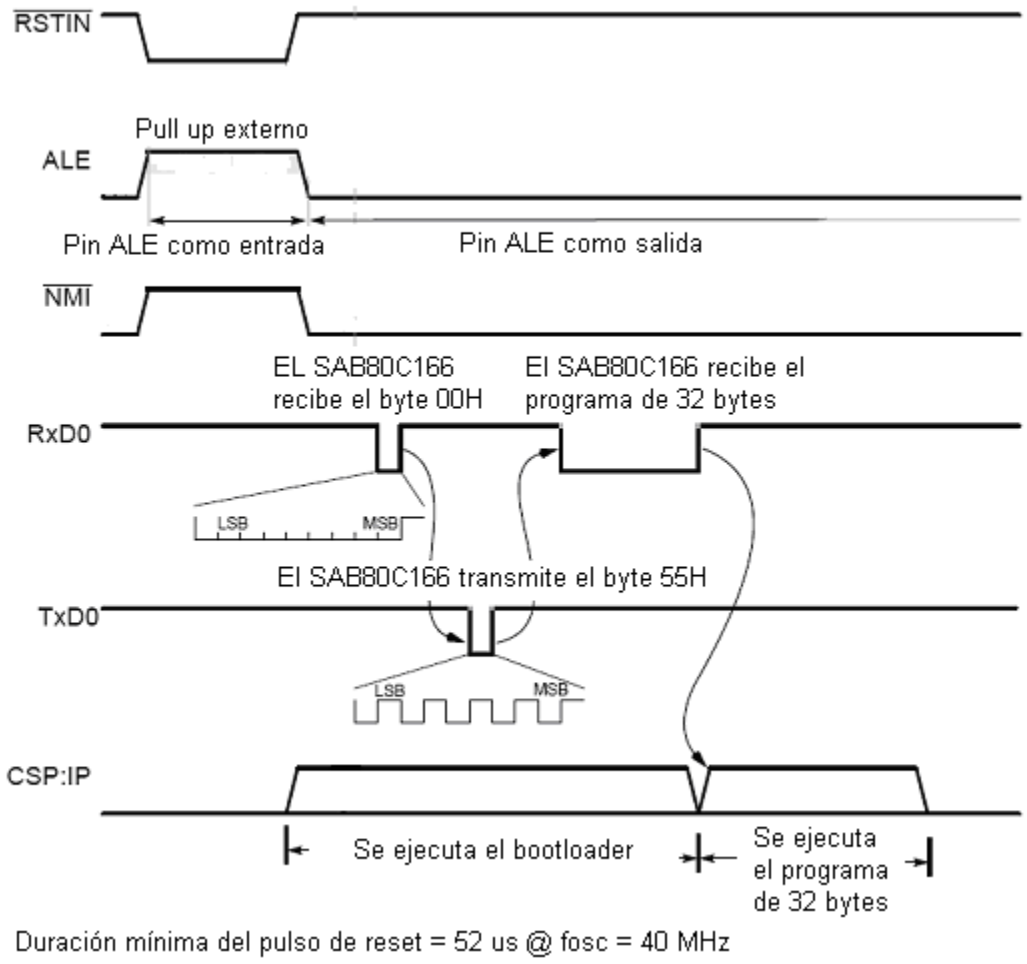


Figura 2.6: Modo de cargar programa en el SAB80C166.

2.4 Sensores de temperatura 1-Wire

La computadora de vuelo cuenta con 3 sensores de temperatura 1-Wire DS18S20 conectados al pin 3 del puerto 3 del SAB80C166 (ver figura 2.7). El SAB80C166 lee cada

sensor utilizando comandos y señales 1-Wire. Cada sensor tiene un número de identificación único de 64 bits llamado ROM Code, voltaje de alimentación de 5 V, rango de medición de -55°C a +125°C, precisión de ±0.5°C, resolución de 9 bits y tiempo máximo de conversión A/D de la lectura de temperatura de 750ms.

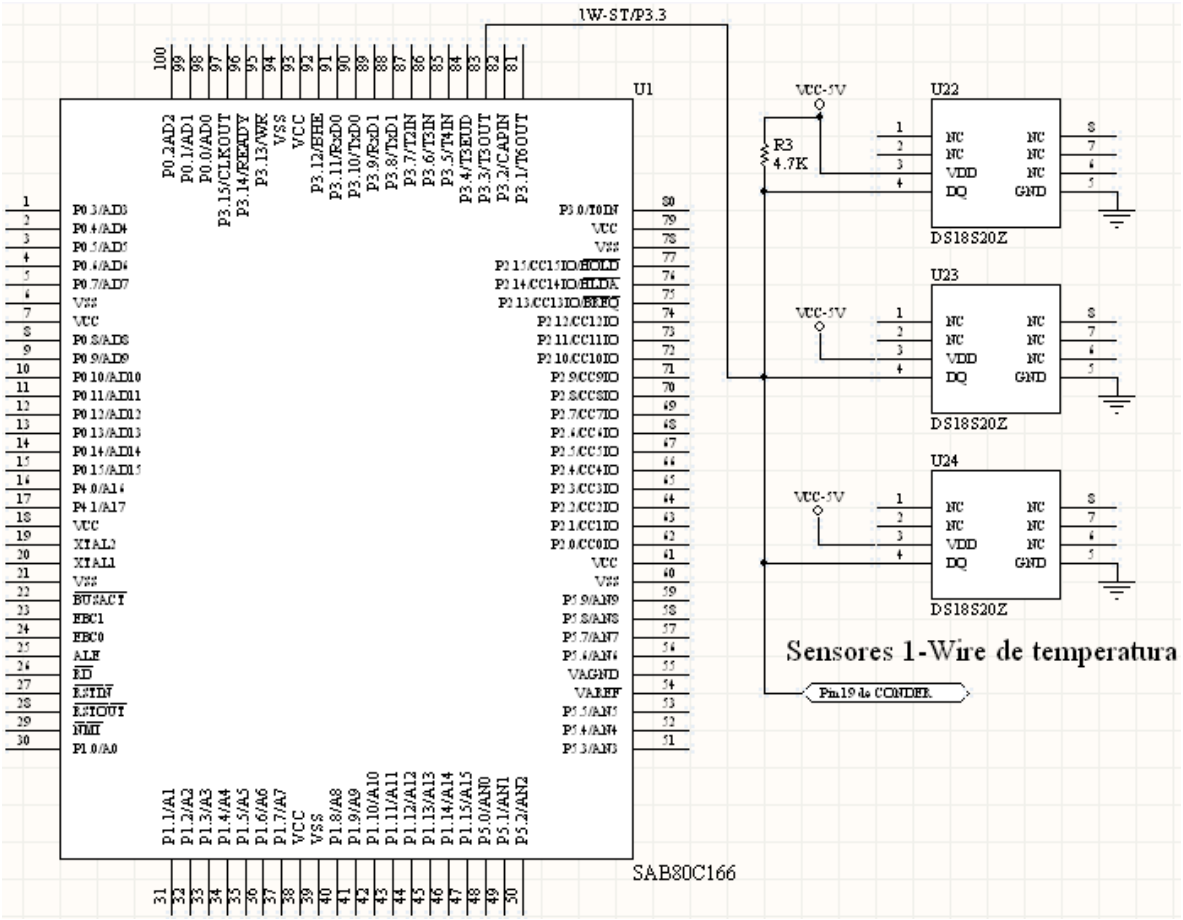


Figura 2.7: Conexión de los 3 sensores de temperatura 1-Wire al SAB80C166.

El pin 3 del puerto 3 del SAB80C166 está conectado al pin 19 del conector derecho de la computadora para permitir la conexión de más sensores 1-Wire de temperatura que se encuentren en otras tarjetas del sistema.

2.5 Memoria flash

La computadora de vuelo cuenta con 32 MBytes de memoria flash formada por 4 circuitos integrados AT45DB642D. Las 4 memorias flash están conectadas en los pines 6 a 12 del puerto 2 del SAB80C166 (ver figura 2.8). El circuito integrado AT45DB642D tiene una capacidad de 8 MBytes, funciona como un dispositivo SPI esclavo de 4 líneas (CS#, SCK, SI y SO), soporta una interfaz SPI de hasta 66 MHz, voltaje de alimentación de 3.3V y rango de operación de –40°C a +85°C.

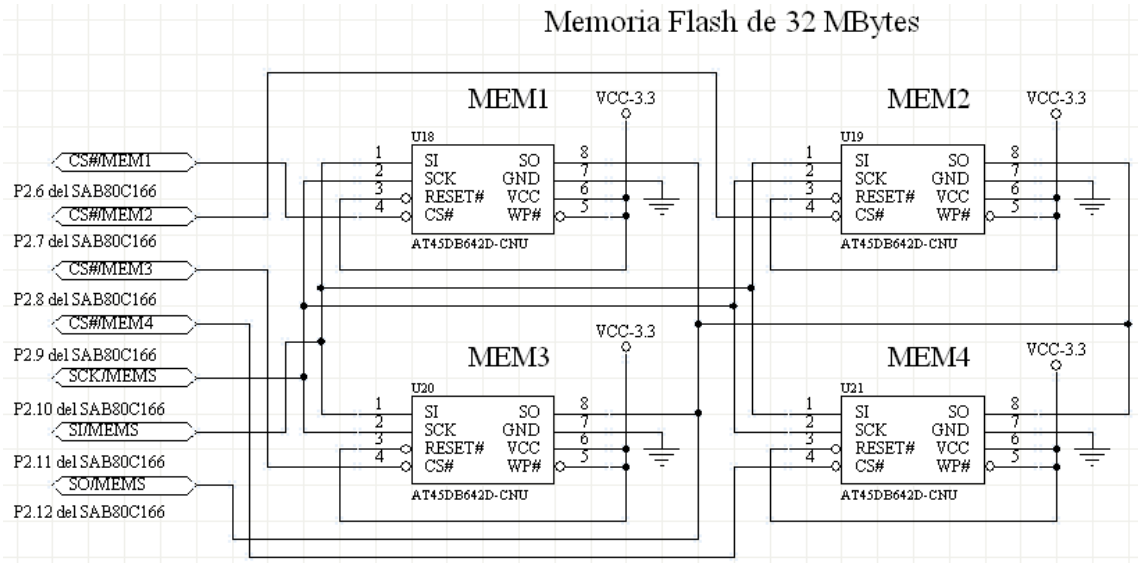


Figura 2.8: Conexión de los 32 MBytes de memoria flash al SAB80C166.

El SAB80C166 habilita la memoria flash que quiere manejar llevando el pin CS# correspondiente a bajo. Las transferencias de datos entre la memoria habilitada y el SAB80C166 se realizan a través de los pines SI y SO en sincronía con la señal de reloj SCK generada por el SAB80C166.

2.6 Protecciones contra efecto latch-up

Cuando los semiconductores son expuestos a radiación ionizante en exceso presentan el efecto latch-up que se manifiesta con la elevación en el consumo de corriente eléctrica del dispositivo. Entre mayor nivel de integración tenga el dispositivo el efecto latch-up es mayor. Si esta elevación en el consumo de la corriente eléctrica no se detiene el dispositivo puede dañarse permanentemente. Una forma de detener el efecto latch-up es desenergizando el dispositivo electrónico. En la computadora de vuelo el SAB80C166, la memoria RAM y el PIC16F876A están protegidos contra efecto latch-up. Esta protección monitorea la corriente eléctrica que consume cada dispositivo y en el momento en que esta rebasa el umbral máximo de operación se genera una señal verificada alta en los pines Latch-up/pulso-SAB, Latch-up/pulso-RAM o Latch-up/pulso-PIC de los conectores tarjeta a tarjeta. Esta señal le indica a la tarjeta de potencia que debe apagar la computadora de vuelo. La protección contra efecto latch-up esta formada por los circuitos integrados MAX4071 y LM6511 (ver figura 2.9). El MAX4071 se utiliza para obtener lecturas de voltaje que son proporcionales a la corriente eléctrica consumida y el LM6511 es un comparador de voltaje que sirve para generar la señal verificada alta. En la computadora de vuelo hay un convertidor A/D 1-Wire DS2450 que sirve para obtener lecturas de la corriente eléctrica consumida por los dispositivos con protección contra efecto latch-up. El convertidor A/D DS2450 cuenta con un número único de identificación de 64 bits, cuatro canales de conversión A/D, resolución de hasta 16 bits, voltaje de alimentación de 5 V y rango de operación de –40°C a +85°C.

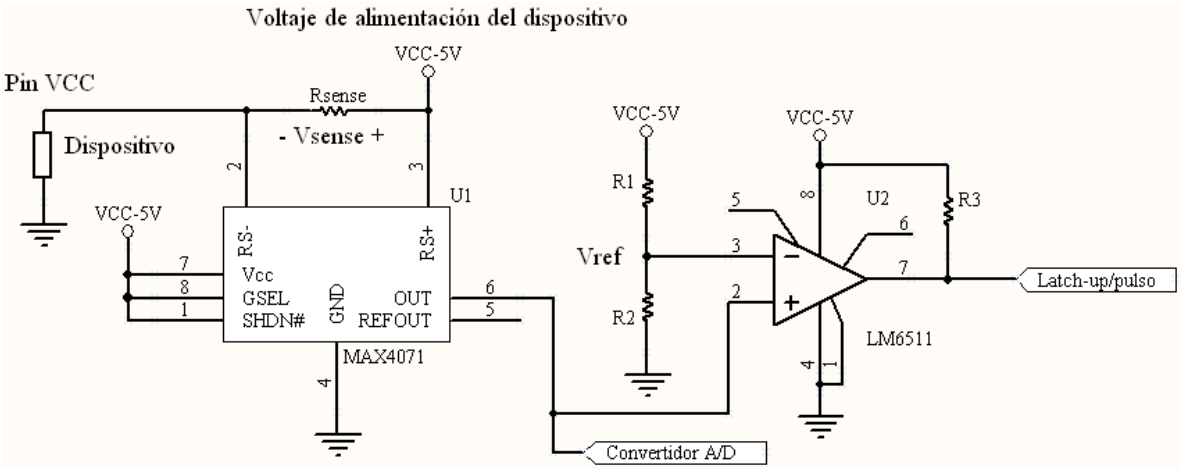


Figura 2.9: Protección de un dispositivo contra efecto latch-up.

Capítulo 3
Cargado de software en la computadora de vuelo

3.1 Introducción

La computadora de vuelo del sistema de entrenamiento se puede reprogramar a través del software de PC en el momento en que el usuario lo requiera y sin mover elementos de hardware. Para cumplir con este propósito se colocó un microcontrolador PIC16F876A cuya función principal es poner al SAB80C166 en modo de cargar programa.

En este capítulo se habla del software y hardware para programar el PIC16F876A; después se explica el proceso a seguir para cargar un programa en el SAB80C166 a través de LabVIEW.

3.2 Microcontrolador PIC16F876A

3.2.1 Especificaciones

El microcontrolador PIC16F876A (ver figura 3.1) trabaja con una señal de reloj de hasta 20 MHz, tiene una memoria para programas tipo flash de 14.3 KBytes, memoria de datos tipo SRAM de 368 Bytes, memoria EEPROM de 256 Bytes, cuenta con 5 convertidores A/D de 10 bits cada uno, 2 canales de CCP (PWM), 2 timers de 8 bits y 1 de 16 bits, 2 comparadores, 14 fuentes de interrupción, MSSP tipo SPI, Master I²C y USART.

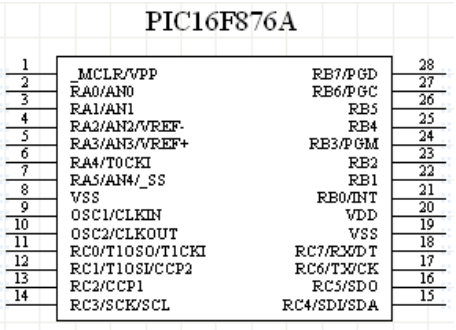


Figura 3.1: Patigrama del PIC16F876A.

Las principales razones por las cuales se eligió este microcontrolador para cumplir con la función de cargar programa en el SAB80C166 son: su disponibilidad en encapsulado SSOP de 28 pines el cual ocupa poco espacio, posee 14.3 KBytes de memoria programa tipo flash la cual es más que suficiente para nuestro propósito y que cuenta con un pin de salida tipo “open drain” que generará la señal en el pin ALE del SAB80C166.

3.2.2 Herramientas de software

El desarrollo del software para el PIC16F876A se hizo utilizando el compilador de C para microcontroladores PIC marca CCS y MPLAB IDE. El cargado de software en el PIC16F876A se hace utilizando una modificación del programador Multi PIC Programmer y el software IC–Prog.

Los pasos para programar el PIC16F876A de la computadora de vuelo son:

1. Diseñar el software en lenguaje C.

2. Compilar el software.

3. Desenergizar la computadora de vuelo y quitar los jumpers W1 y W2.

4. Conectar el header ICSP del programador de PIC´s con el header PROG de la computadora de vuelo.

5. Abrir el correspondiente archivo HEX con el software IC–Prog y programarlo en el PIC.

6. Desconectar el programador de PIC´s.

7. Poner los jumpers W1 y W2.

8. Al energizar la computadora de vuelo el software programado en el PIC16F876A se ejecuta automáticamente.

3.2.3 Programador

El programador Multi PIC Programmer (ver figuras 3.2, 3.3 y 3.4) es básicamente un circuito electrónico que sirve para ajustar los voltajes del puerto serial de la PC a los voltajes que necesita el PIC para ser programado. Este circuito utiliza programación de alto voltaje que consiste en aplicar 14V en el pin VPP del microcontrolador, 5V en el pin VDD mientras que por el pin PGD se transfiere el software en sincronía con la señal de reloj en el pin PGC. Para programar el PIC16F876A el interruptor S1 del programador debe estar abierto.

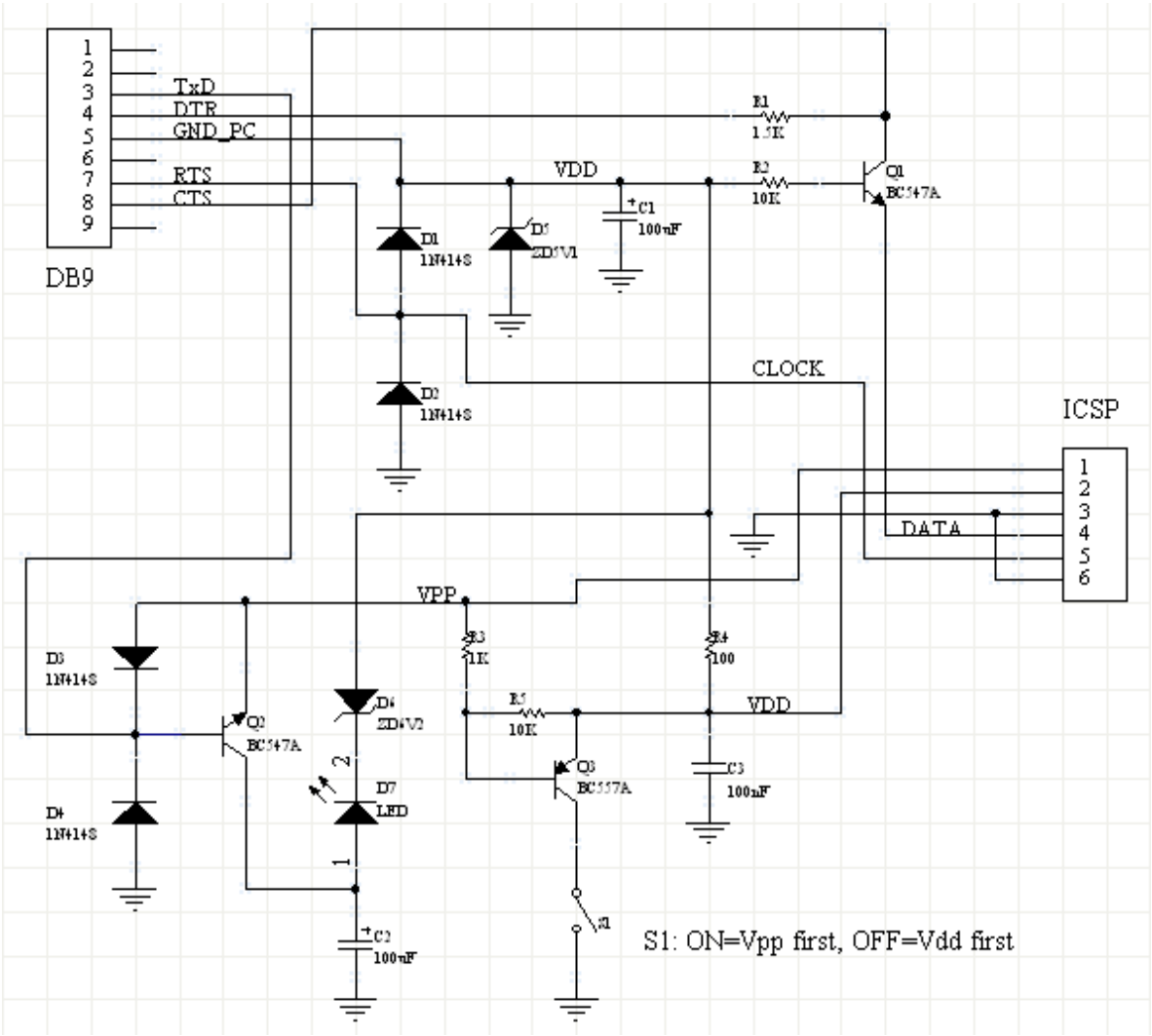


Figura 3.2: Diagrama electrónico del programador de PIC's.

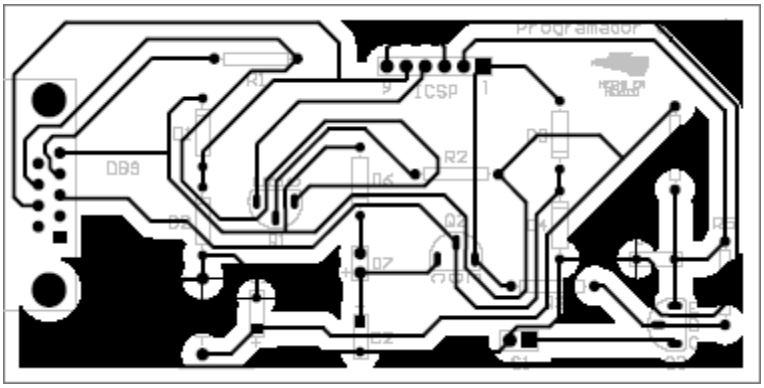


Figura 3.3: PCB del programador de PIC's.

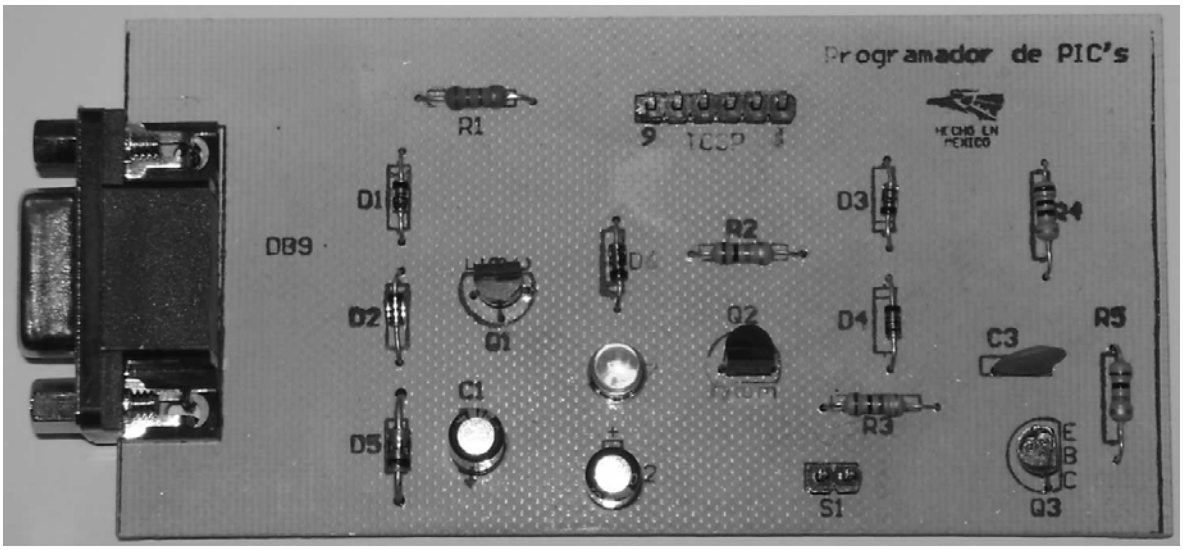


Figura 3.4: Programador de PIC's.

3.2.4 Funciones del PIC16F876A en la computadora de vuelo

La memoria para programas del SAB80C166 es RAM y por lo tanto al energizar la computadora de vuelo el microcontrolador no ejecuta ningún programa. Esto es una desventaja ya que obviamente es necesario que en el encendido se ejecute un programa que realice funciones básicas como el apagado o encendido de ciertas tarjetas del sistema o el envío de mensajes que indiquen al software de PC que el sistema de entrenamiento se ha encendido. En la figura 3.5 se muestra el diagrama de conexiones del PIC16F876A en la computadora de vuelo.

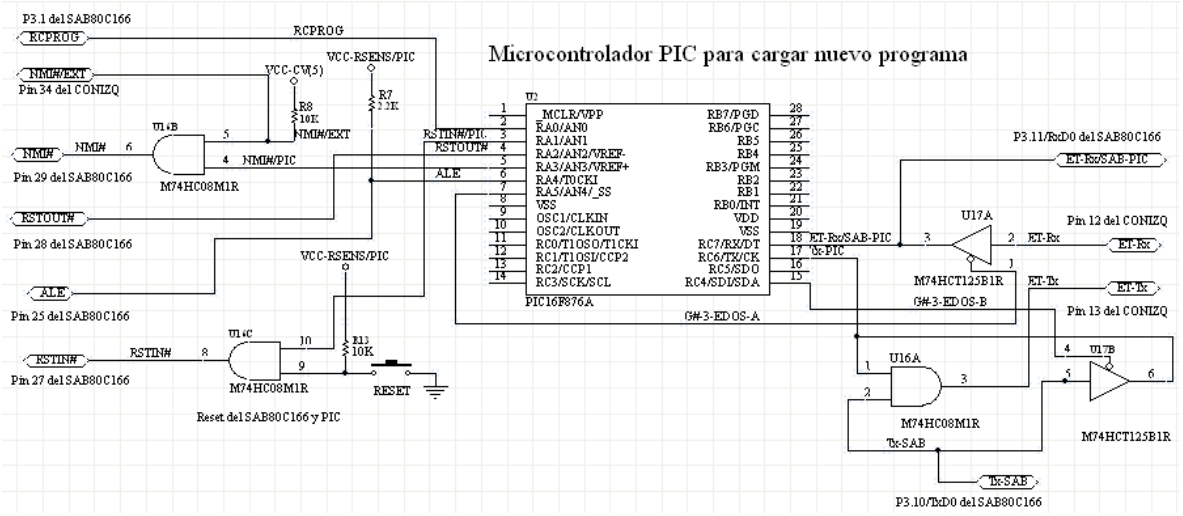


Figura 3.5: Conexión del PIC16F876A para cargar nuevo programa en el SAB80C166.

El PIC16F876A genera las señales RSTIN#, NMI# y ALE que activan el modo de cargar programa del SAB80C166. El pin Rx del PIC16F876A esta conectado con el pin Rx0 del SAB80C166 para que ambos microcontroladores puedan recibir comandos del

software de PC. El pin Tx del PIC16F876A y Tx0 del SAB80C166 están conectados a la entrada de la compuerta AND U16A cuya salida es el transmisor hacia el software de PC y así ambos microcontroladores puedan transmitir. Los buffers tres estados U17A y U17B sirven para que en el encendido el PIC16F876A pueda cargar un programa en la memoria RAM interna del SAB80C166. Este programa se puede utilizar para configurar los puertos del microcontrolador o también puede servir para transferir la aplicación completa de la computadora de vuelo de la memoria flash a la memoria RAM externa para su ejecución. Si el buffer tres estados U17A desconecta el transmisor del software de PC de los receptores de los 2 microcontroladores entonces en el pin Rx del PIC16F876A debe implementarse un Tx por software. Si el buffer tres estados U17B une los pines transmisores de los 2 microcontroladores entonces en el pin Tx del PIC16F876A debe implementarse un Rx por software. Para ejemplificar el cargado de software en el encendido (ver figura 3.7) supongamos que el PIC16F876A pone al SAB80C166 en modo de cargar programa y le envía el siguiente software de inicialización de puertos:

```
MOV    STKUN,#0FA40h      ; Registro Stack Underflow
MOV    STKOV,#0FA20h      ; Registro Stack Overflow
MOV    SYSCON,#062C0h     ; Pila: 32 words, Sin bus externo
MOV    DP2,#0EFFFh        ; Configuración del puerto 2
MOV    DP3,#0F5FFh        ; Configuración del puerto 3
MOV    P2,#003C0h         ; Valores en el puerto 2
MOV    P3,#02580h         ; Valores en el puerto 3
BSET   P3.1               ; Le indica al PIC16F876A que finalizo el programa
fin:   JMPR    fin        ; fin del programa
```

El PIC16F876A envía este software al SAB80C166 como la serie de bytes:

E6, 0B, 40, FA, E6, 0A, 20, FA, E6, 86, C0, 62, E6, E1, FF, EF, E6, E3, FF, F5,E6, E0, C0, 03, E6, E2, 80, 25, 1F, E2, 0D, FF

Una vez ejecutado este programa el PIC16F876A envía al software de PC la cadena de caracteres: ‘E’, ‘N’, ‘C’, ‘E’, ‘N’ y espera 1 minuto por el byte de respuesta ‘K’. El PIC16F876A envía la cadena de caracteres y espera 1 minuto las veces que sea necesario hasta recibir el byte de respuesta proveniente del software de PC. Una vez recibido el byte de respuesta, el PIC16F876A espera la cadena de caracteres: ‘R’, ‘E’, ‘C’, ‘I’ y después se pone en modo de recepción de comandos. Cada comando del sistema consiste en una cadena de 14 bytes (ver figura 3.6). El primer byte o byte de despertar siempre es ‘D’ y sirve para que en el PIC16F876A ocurra una interrupción por recepción en el puerto serial y se ejecute la rutina que recibirá el resto del comando. El segundo byte es el número de comando. Los siguientes 11 bytes son los parámetros del comando. El último byte es el valor del checksum calculado a partir del número de comando y los parámetros del comando. Para calcular el checksum hay que sumar los bytes, tomar el byte menos significativo del resultado y calcularle el complemento a2.

Comando de 14 bytes

Byte de despertar: ‘D’	Número de comando (1 byte)	Parámetros del comando (11 bytes)	Checksum (1 byte)

Figura 3.6: Estructura de un comando.

El PIC16F876A puede ejecutar los comandos 0, 1 y 2, todos los demás comandos los ignora ya que son ejecutados por el SAB80C166. El comando 0 pone al SAB80C166 en modo de cargar programa proveniente del software de PC, el comando 1 ejecuta un reset por hardware en el SAB80C166 y el comando 2 activa la interrupción por NMI en el SAB80C166.

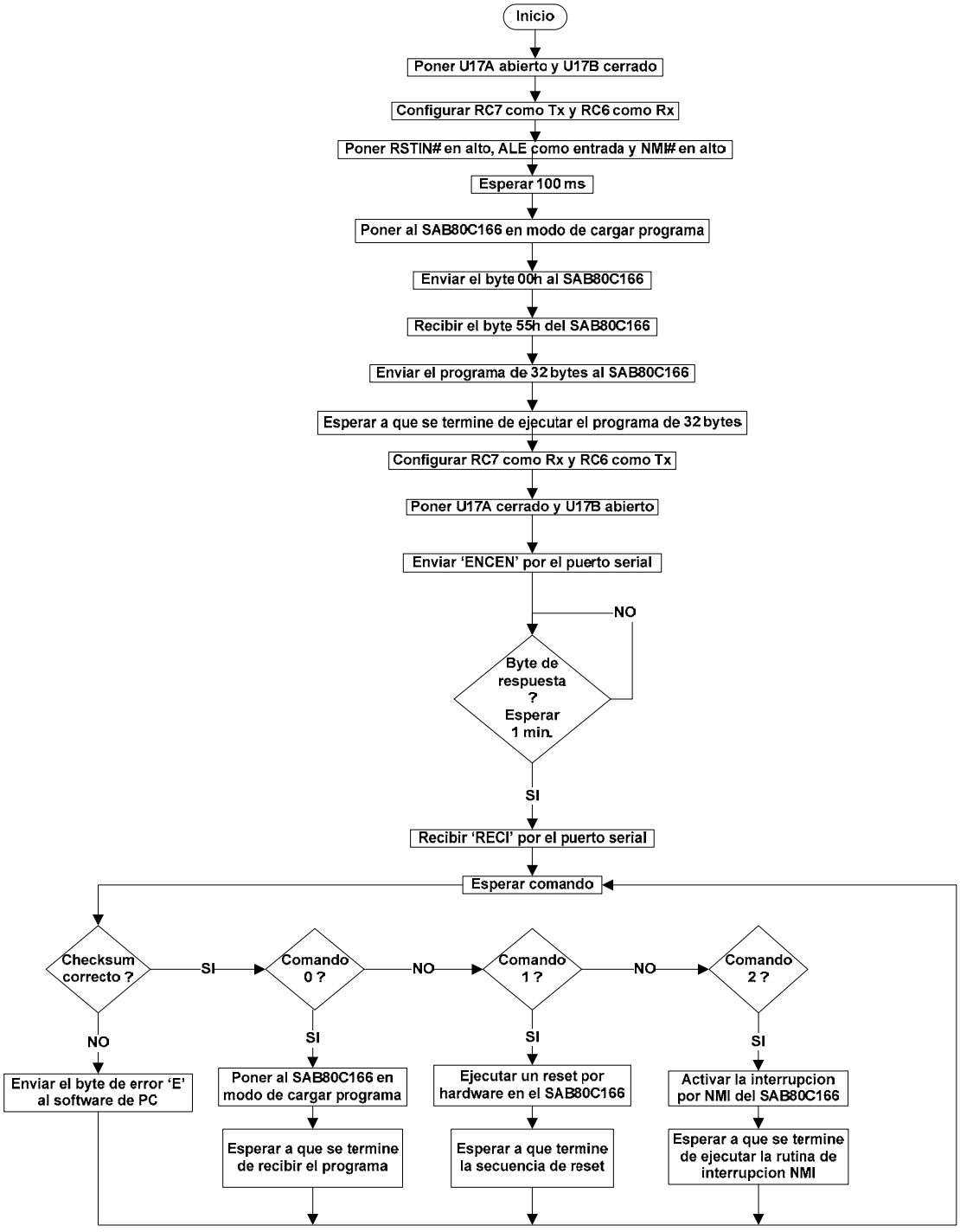


Figura 3.7: Ejemplo de software en el PIC16F876A (código en el apéndice B).

3.3 Tarjeta de comunicaciones entre la computadora de vuelo y el software de PC

En la computadora de vuelo se incluyó un conector RJ11 en el cual se conecta una tarjeta electrónica (ver figuras 3.8, 3.9 y 3.10) que proporciona 5V, 3.3V y contiene un circuito MAX232 que sirve de interfaz entre el canal serial 0 del SAB80C166 y el software de PC.

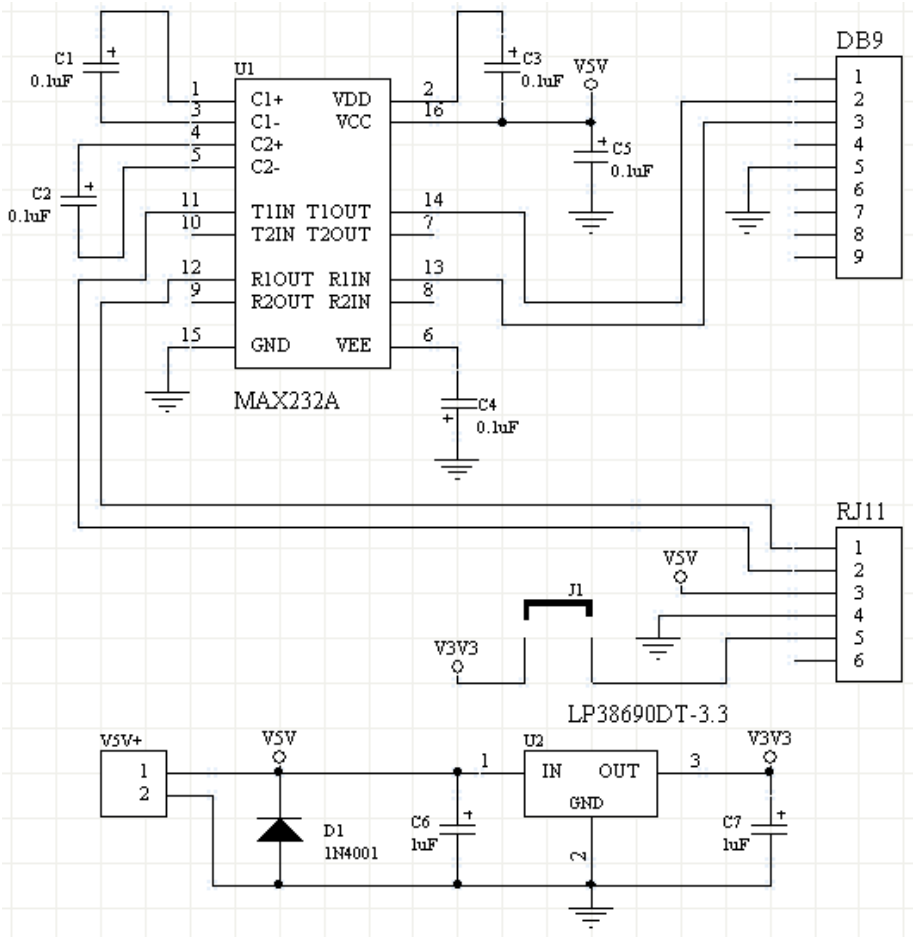


Figura 3.8: Diagrama electrónico de la tarjeta de comunicaciones.

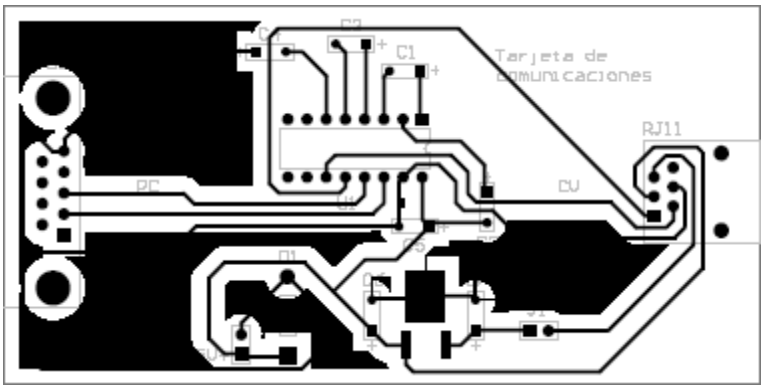


Figura 3.9: PCB de la tarjeta de comunicaciones.

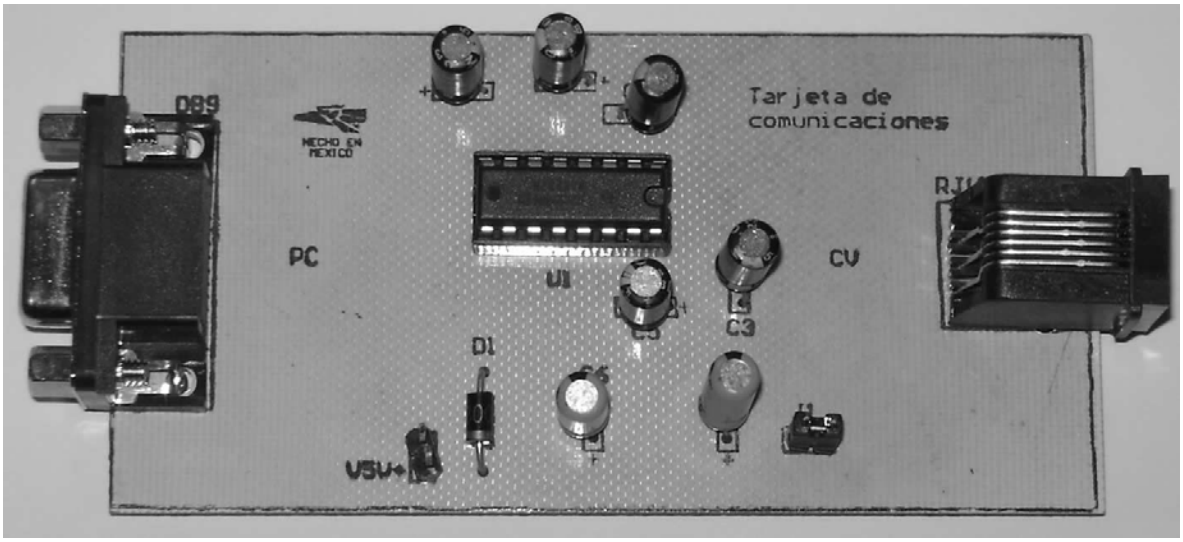


Figura 3.10: Tarjeta de comunicaciones.

3.4 Cargado de software en el SAB80C166

3.4.1 Precargador y cargador externo

Para cargar software en el SAB80C166 es necesario activar el modo de cargar programa utilizando el PIC16F876A. El modo de cargar programa (ver figuras 3.11 y 3.12) se activa cuando durante una señal de reset por hardware el pin ALE se encuentra en 1 y el pin NMI# cambia de 1 a 0. En el modo de cargar programa el SAB80C166 espera por el canal de comunicaciones 0 un byte 00H con un bit de inicio y uno de paro. Este byte 00H le sirve al SAB80C166 para calcular el baudaje del dispositivo que le va a enviar el software. Una vez recibido el byte 00H, el SAB80C166 envía como respuesta el byte 55H y espera un programa de 32 bytes que se guarda en las localidades de memoria interna 0FA40H a 0FA5FH. El programa de 32 bytes o precargador se ejecuta automáticamente después de ser recibido. El precargador (ver figura 3.13) envía el byte 01h a LabVIEW y recibe un programa de 436 bytes que se guarda a partir de la localidad de memoria interna #0FA60h. Este programa de 436 bytes llamado cargador externo (ver figura 3.16) sirve para escribir en la memoria RAM externa el programa principal de hasta 256 KBytes que es transmitido por la interfaz en LabVIEW.

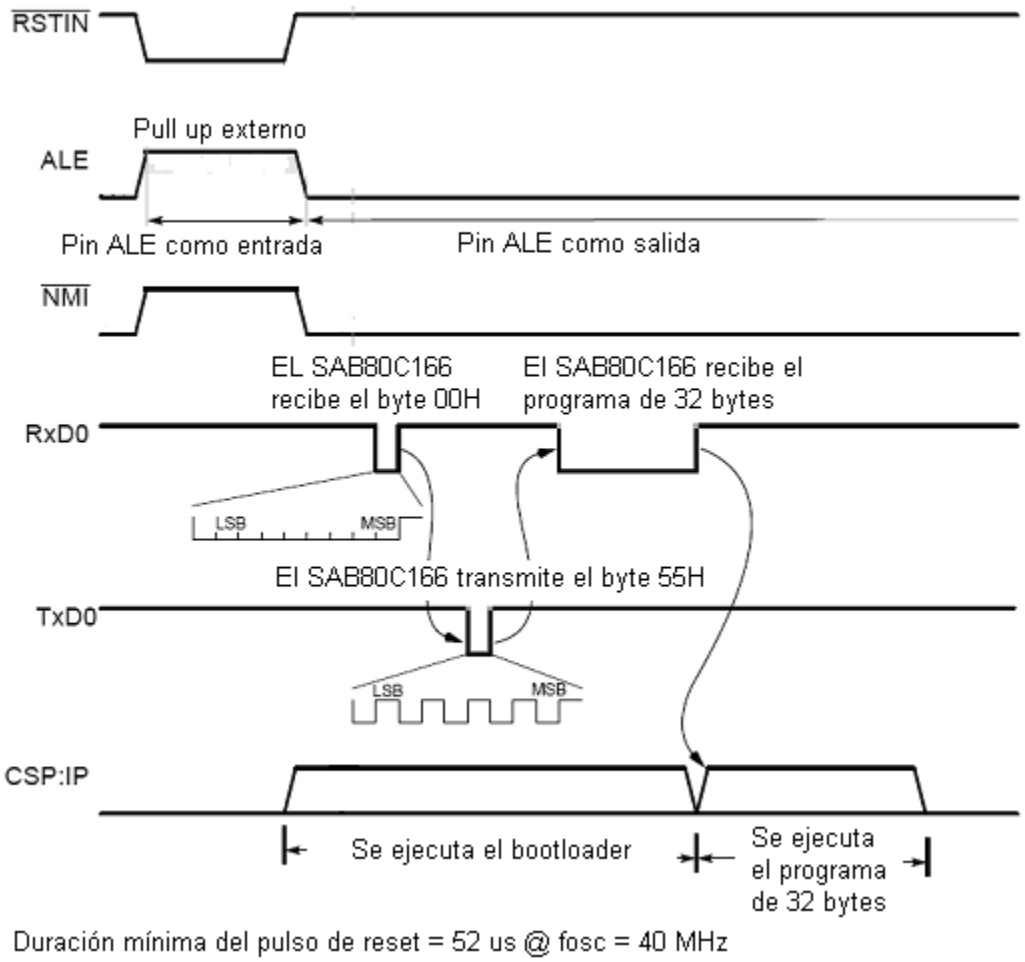


Figura 3.11: Activación del modo de cargar programa en el SAB80C166.

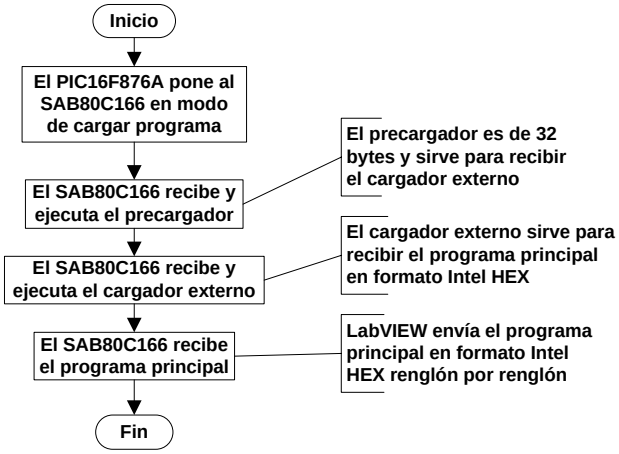


Figura 3.12: Secuencia de pasos para cargar un programa en el SAB80C166.

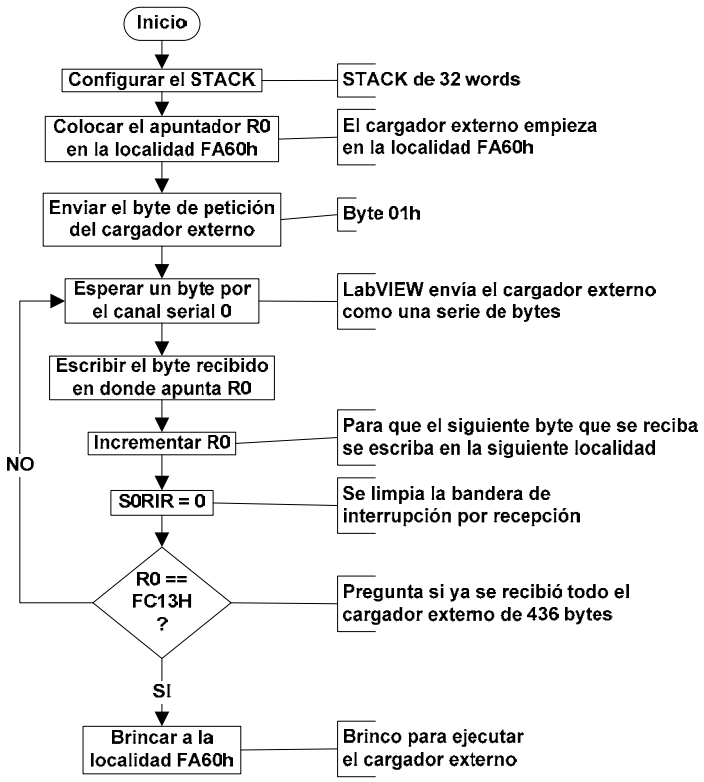


Figura 3.13: Diagrama de flujo del precargador (código en el apéndice C.1).

La interfaz en LabVIEW envía al SAB80C166 el precargador y el cargador externo como una serie de bytes con los códigos numéricos de cada instrucción. El precargador y el cargador externo están escritos en lenguaje ensamblador.

Los programas para el SAB80C166 se escribieron en lenguaje C. A partir del código en lenguaje C se obtiene un archivo en formato Intel HEX. Un archivo en formato Intel HEX (ver figura 3.14) se lee por renglones. Cada renglón (ver figura 3.15) está formado por 6 partes que son:

1. **Carácter de inicio**: ":".
2. **Número de bytes**: Byte que indica la cantidad de datos en el renglón.
3. **Dirección**: Dos bytes que indican la dirección de memoria en donde deben escribirse los datos del renglón.
4. **Record type**: Byte que indica el tipo de información contenida en el renglón.
5. **Datos**.
6. **Checksum**. Checksum de 1 byte calculado a partir de todos los bytes del renglón excepto el carácter ":" y el checksum.

Los distintos record type son:

- **Data record (00h)**. Renglón con datos para escribir en la memoria.
- **End of file record (01h)**. Último renglón del archivo.

- **Extended segment address record (02h)**. Este renglón contiene 2 bytes que se deben recorrer 4 bits a la izquierda para obtener una constante que debe sumarse a las direcciones de los siguientes renglones. Esto permite direccionar una memoria de hasta 1 MByte.
- **Start segment address record (03h)**. Sirve para indicar que los 2 primeros bytes de datos del renglón son el valor de CS (code segment) y los 2 siguientes bytes el valor de IP (instruction pointer). El valor de CS e IP indica la localidad de memoria en donde comienza la ejecución del programa.

```
:  10  0013  00  AC12AD13AE10AF1112002F8E0E8F0F22  44
:  10  0003  00  E50B250DF509E50A350CF50812001322  59
:  03  0000  00  020023                            D8
:  0C  0023  00  787FE4F6D8FD758113020003          1D
:  10  002F  00  EFF88DF0A4FFEDC5F0CEA42EFEEC88F0  16
:  04  003F  00  A42EFE22                          CB
:  00  0000  01                                    FF
```

Figura 3.14: Ejemplo de un archivo en formato Intel HEX de 7 renglones.

Caracter de inicio ':'	Número de bytes (1 byte)	Dirección (2 bytes)	Record type (1 byte)	Datos	Checksum (1 byte)

Figura 3.15: Estructura de un renglón de un archivo en formato Intel HEX.

El cargador externo (ver figura 3.16) configura el registro SYSCON del SAB80C166 y luego envía a LabVIEW el byte de petición de renglón 01h para recibir el primer renglón del archivo Intel HEX. Si el record type del renglón es 00h entonces los datos del renglón se escriben en las localidades de memoria correspondiente y se recibe el checksum. Si el record type es 01h entonces se recibe el checksum, se envía a LabVIEW el byte 00h que indica la finalización del cargado de software y se ejecuta un reset por software para comenzar la ejecución del programa recibido. Si el record type es 02h entonces se reciben 2 bytes, el checksum y se guarda la constante a sumar a las direcciones de los siguientes renglones. Si el record type es 03h entonces se reciben 4 bytes y el checksum. Si el checksum recibido es igual al calculado entonces se solicita a LabVIEW el siguiente renglón del archivo Intel HEX. Si el checksum recibido es diferente al calculado entonces se envía a LabVIEW el byte de error 02h para no continuar con el cargado de software.

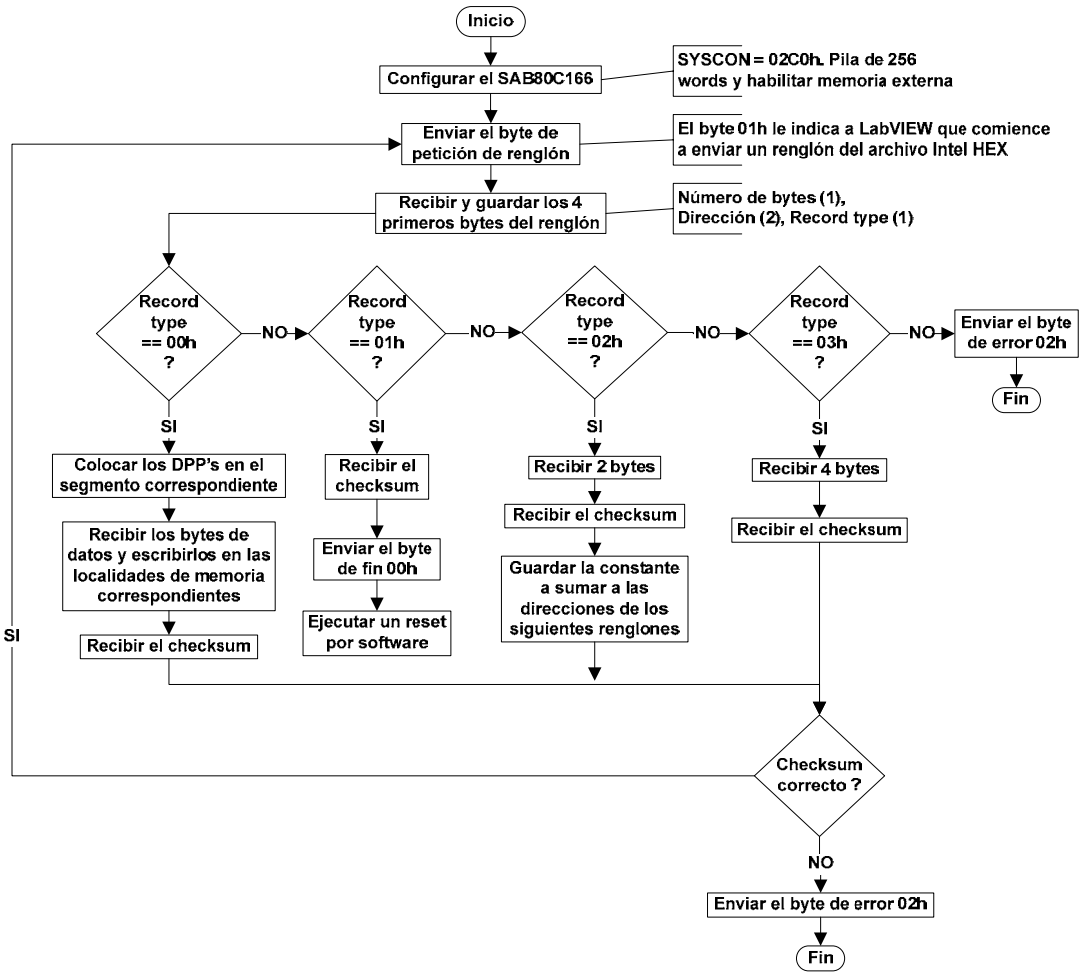


Figura 3.16: Diagrama de flujo del cargador externo (código en el apéndice C.2).

3.4.2 Interfaz en LabVIEW

La interfaz en LabVIEW necesita que el SAB80C166 esté conectado a la PC en modo de cargar programa. El PIC16F876A tiene que poner al SAB80C166 en modo de cargar programa en el encendido (ver figura 3.17).

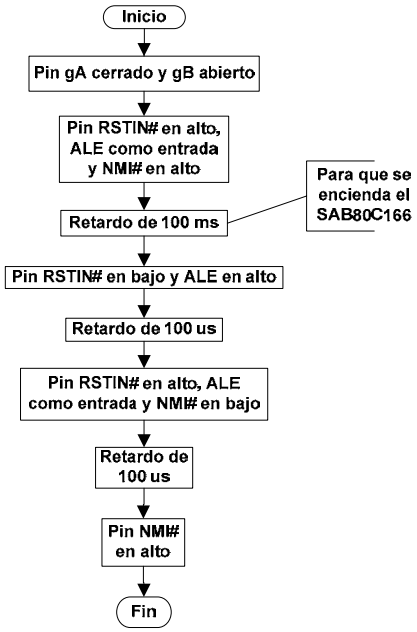


Figura 3.17: Software en el PIC16F876A para poner al SAB80C166 en modo de cargar programa en el encendido (código en el apéndice C.3).

En la interfaz de LabVIEW (ver figura 3.18) el usuario debe seleccionar un archivo Intel HEX y presionar el botón “Abrir” para que su contenido y número de bytes se despliegue en los campos correspondientes. Al presionar el botón “Programar” (ver figura 3.19) el led “Programando” se enciende y LabVIEW envía el byte 00H, recibe el byte 55H, envía el precargador de 32 Bytes, recibe el byte 01h del precargador, envía el cargador externo, se comunica con el cargador externo para enviar el archivo Intel HEX renglón por renglón y finalmente apaga el led “Programando”.

Figura 3.18: Interfaz en LabVIEW para cargar un programa en el SAB80C166.

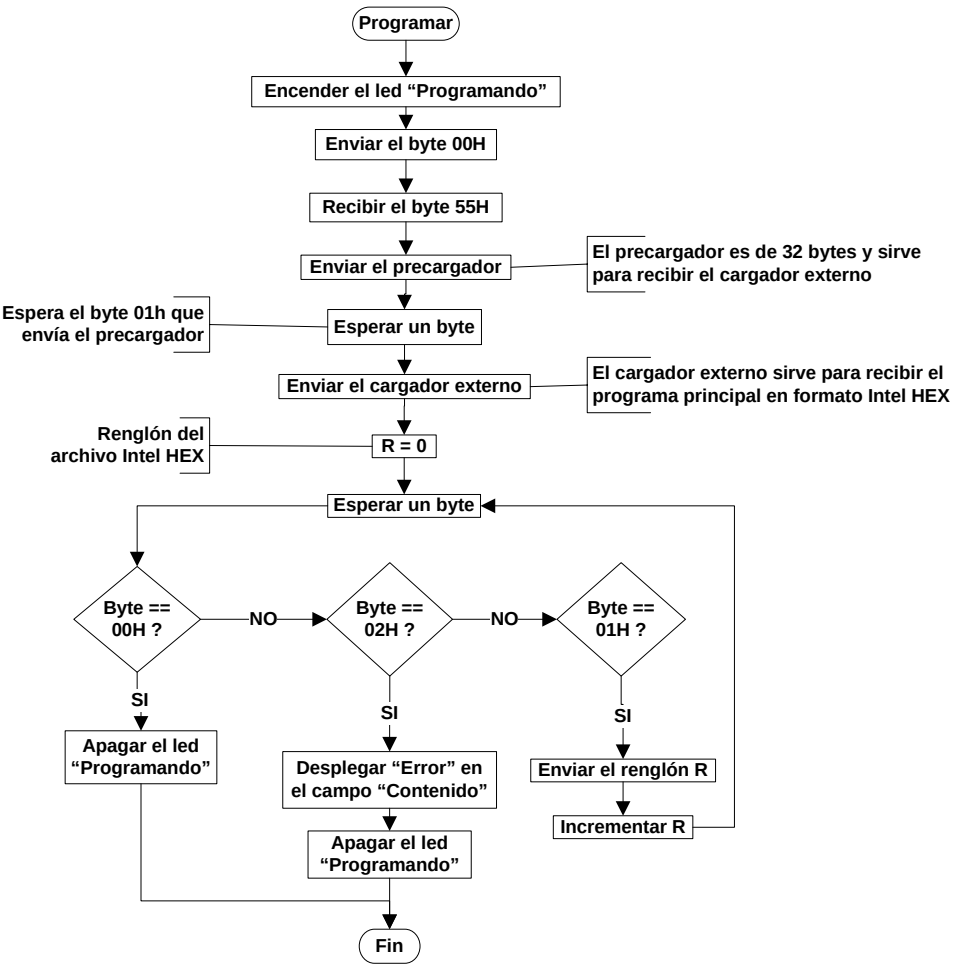


Figura 3.19: Diagrama de flujo del software en LabVIEW para cargar un programa en el SAB80C166 (código en el apéndice C.4).

El programa cargado en el SAB80C166 se ejecuta automáticamente después de ser recibido.

Capítulo 4
Sensores de temperatura 1-Wire

4.1 Introducción

La computadora de vuelo cuenta con 3 sensores de temperatura que son leídos a través de un solo pin I/O del SAB80C166 utilizando el protocolo 1-Wire. Este pin I/O está conectado a uno de los conectores tarjeta a tarjeta para permitir la lectura de sensores de temperatura 1-Wire que se encuentren en otros subsistemas. La ventaja de utilizar sensores 1-Wire es que se pueden manejar muchos utilizando un solo pin I/O del SAB80C166.

En este capítulo se describe el sensor DS18S20, se explica el protocolo 1-Wire, el software para leer el ROM code de un sensor 1-Wire y el software para leer los sensores de temperatura.

4.2 Sensor DS18S20

El circuito integrado DS18S20 es un sensor de temperatura digital tipo 1-Wire marca Dallas Semiconductor que puede tomar lecturas en el rango de - 55°C a + 125°C, precisión de ± 0.5°C y resolución de 9 bits.

La tecnología 1-Wire consiste básicamente en un dispositivo maestro que maneja a uno o varios esclavos a través de un solo pin I/O. El dispositivo maestro puede ser un microprocesador o microcontrolador y el o los esclavos pueden ser cualquiera de los dispositivos 1-Wire que fabrique Dallas Semiconductor. A un sistema con un solo esclavo se le llama single–drop y a uno con varios esclavos multi–drop. Los sistemas multi–drop pueden combinar dispositivos 1-Wire de cualquier tipo como pueden ser sensores de temperatura, convertidores A/D, memorias SRAM, etc.

4.2.1 ROM code

Cada dispositivo 1-Wire tiene un número serial único de 64 bits llamado ROM code (ver figura 4.1) que se encuentra almacenado en la memoria ROM interna. Los 8 bits menos significativos del ROM code representan la familia del dispositivo que en el caso del DS18S20 es 10h. Los siguientes 48 bits son un número serial único. Los 8 bits más significativos son el CRC calculado a partir de los 56 bits menos significativos. El CRC sirve para verificar que se ha leído correctamente el ROM code.

CRC de 8 bits		Número serial de 48 bits		Familia de 8 bits (10h)	
MSB	LSB	MSB	LSB	MSB	LSB

Figura 4.1: ROM code de 64 bits.

Para verificar el ROM code se utiliza el siguiente registro de corrimiento:

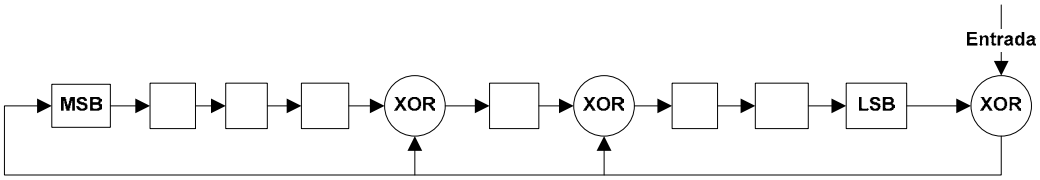


Figura 4.2: Registro de corrimiento para verificar el ROM code.

Todos los bits del registro de corrimiento comienzan con un valor de cero, después se introduce el ROM code comenzando por el bit menos significativo. Si después de introducir todos los bits del ROM code el registro de corrimiento vale cero entonces el ROM code es correcto.

4.2.2 Memoria interna

El sensor DS18S20 cuenta con una memoria SRAM interna de 9 bytes (ver figura 4.3). El byte 0 y 1 son de solo lectura y almacenan el byte menos significativo y el mas significativo de la temperatura medida. El byte 2 y 3 almacenan el límite superior e inferior de la alarma del DS18S20. El byte 4 y 5 no se utilizan y se leen como FFh. El byte 6 o count remain y el byte 7 o count per °C sirven para obtener una resolución mayor a 9 bits. El byte 8 es el CRC de los primeros 8 bytes. El CRC sirve para que el dispositivo maestro verifique que se ha leído correctamente la memoria SRAM del esclavo.

Memoria SRAM interna

Byte 0	T LSB
Byte 1	T MSB
Byte 2	TH
Byte 3	TL
Byte 4	Reservado
Byte 5	Reservado
Byte 6	COUNT REMAIN
Byte 7	COUNT PER °C
Byte 8	CRC

Figura 4.3: Memoria SRAM interna del DS18S20.

El DS18S20 cuenta con una memoria EEPROM de 2 bytes que sirve para almacenar el límite superior TH e inferior TL de la alarma y así evitar la programación de estos límites cada vez que se enciende el sensor.

La temperatura que obtiene el DS18S20 se encuentra en °C. Esta temperatura se almacena en los dos primeros bytes de la memoria SRAM. El byte más significativo es el byte de signo que vale 0 para temperaturas positivas y 1 para temperaturas negativas. El byte menos significativo es la temperatura del sensor. En la figura 4.4 se muestra el formato del par de registros en donde se almacena la temperatura.

T LSB

2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}

T MSB

S	S	S	S	S	S	S	S

Figura 4.4: Registros T LSB y T MSB de la memoria interna.

Si S vale 0 entonces la temperatura es positiva y vale:

$$T = \frac{TLSB}{2}[°C]$$

Si S vale 1 entonces la temperatura es negativa y vale:

$$T = -\frac{256 - TLSB}{2}[°C]$$

En la tabla 4.1 se muestran equivalencias de la temperatura en formato decimal, binario y hexadecimal.

Temperatura	**Binario**	**Hexadecimal**
85 °C	0000 0000 1010 1010	00AAh
25 °C	0000 0000 0011 0010	0032h
0.5 °C	0000 0000 0000 0001	0001h
0 °C	0000 0000 0000 0000	0000h
- 0.5 °C	1111 1111 1111 1111	FFFFh
- 25 °C	1111 1111 1100 1110	FFCEh
- 55 °C	1111 1111 1001 0010	FF92h

Tabla 4.1: Equivalencias de temperatura en decimal, binario y hexadecimal.

4.2.3 Comandos del sensor

Cualquier comunicación con el sensor 1-Wire ya sea para leer el ROM code, para efectuar una medición de temperatura, para leer la memoria SRAM, etc. debe de realizarse en 3 pasos:

1. Inicialización.
2. Comando de ROM.
3. Comando a ejecutar.

Inicialización. Este paso consiste en la transmisión de un pulso de reset desde el dispositivo maestro al esclavo el cual responderá con un pulso de presencia indicando que se encuentra listo y en espera del comando de ROM.

Comando de ROM. En este paso el dispositivo maestro envía un comando de 1 byte al esclavo. Los comandos de ROM son:

- READ ROM [33h]. Este comando puede utilizarse únicamente en sistemas single–drop. Cuando el dispositivo maestro envía este comando automáticamente el esclavo responde con el ROM code de 64 bits.

- MATCH ROM [55h]. El dispositivo maestro debe enviar este comando seguido del ROM code del esclavo que recibirá un comando a ejecutar.

- SKIP ROM [CCh]. El maestro utiliza este comando seguido de comandos a ejecutar que no requieren de ROM Code. Por ejemplo, puede utilizarse el comando SKIP ROM seguido de CONVERT T para que todos los sensores 1-Wire de un sistema multi–drop tomen lecturas de temperatura.

Comando a ejecutar. El maestro envía un comando de 1 byte al esclavo. El comando a ejecutar puede ser:

- CONVERT T [44h]. Después de que el dispositivo maestro envía este comando el DS18S20 toma una lectura de temperatura, realiza la conversión A/D y el resultado lo coloca en las localidades 0 y 1 de la memoria SRAM. Este proceso se lleva a cabo en aproximadamente 750 ms.

- READ SCRATCHPAD [BEh]. Después de que el maestro envía este comando el esclavo responde con el contenido de la memoria SRAM comenzando por el bit menos significativo de la localidad 0 hasta el bit más significativo de la localidad 9. El maestro puede recibir las 2 primeras localidades y después enviar un pulso de reset para no recibir el resto de la memoria SRAM.

En el protocolo 1-Wire las transferencias de datos y comandos siempre comienzan por el bit menos significativo.

4.3 Señales 1-Wire

En el protocolo 1-Wire existen 4 tipos de señales: pulso de reset, pulso de presencia, escritura de un bit, lectura de un bit. Todas estas señales, excepto el pulso de presencia, son iniciadas por el dispositivo maestro.

4.3.1 Pulso de reset y pulso de presencia

Para generar un pulso de reset (ver figura 4.5) el maestro debe de llevar a 0 el bus 1-Wire por al menos 480 µs, después el maestro debe configurarse en modo de recepción por al menos 480 µs. El esclavo responderá después de mínimo 15 µs con el pulso de presencia llevando a 0 el bus 1-Wire por mínimo 60 µs.

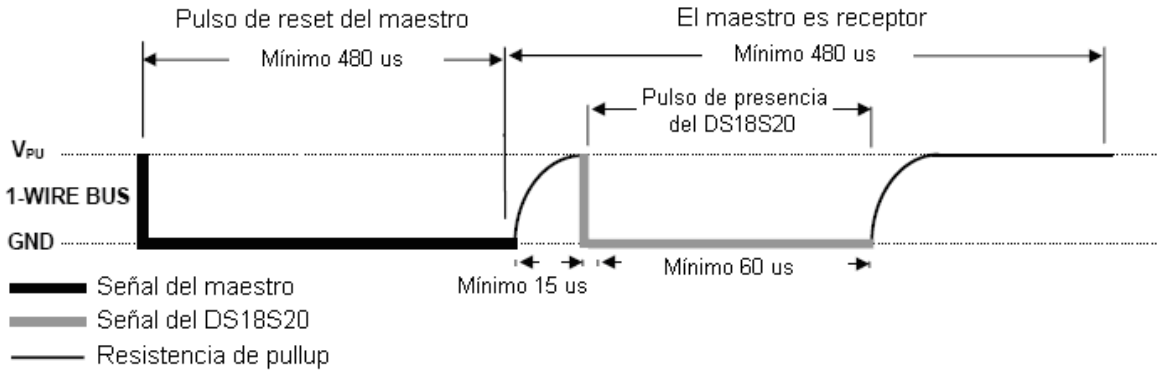


Figura 4.5: Pulso de reset y pulso de presencia.

4.3.2 Escritura de un bit

El dispositivo maestro puede escribir un bit 0 o un bit 1 en el esclavo (ver figura 4.6). Esta escritura se lleva a cabo en al menos 60 µs. Para escribir un bit 0 el maestro debe de llevar el bus 1-Wire a 0 por un tiempo de al menos 60 µs. Para escribir un bit 1 el maestro debe de llevar el bus 1-Wire a 0 por un tiempo máximo de 15 µs y después configurarse en modo de recepción por un tiempo de al menos 45µs. Entre cada bit que se escribe debe de haber un tiempo de recuperación de al menos 1 µs.

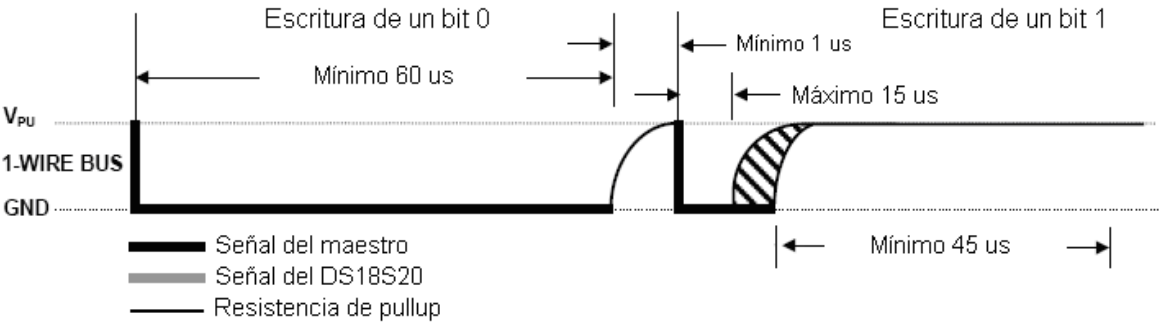


Figura 4.6: Escritura de un bit en el bus 1-Wire.

4.3.3 Lectura de un bit

La lectura de un bit (ver figura 4.7) proveniente del dispositivo esclavo comienza cuando el maestro lleva a 0 el bus 1-Wire por un tiempo de al menos 1 µs y después se configura en modo de recepción. En un tiempo máximo de 15 µs el esclavo responderá llevando el bus 1-Wire a 0 para un bit 0 o llevándolo a 1 para un bit 1. La lectura de un bit se lleva a cabo en al menos 60 µs. Entre cada bit que se lee debe de haber un tiempo de recuperación de al menos 1 µs.

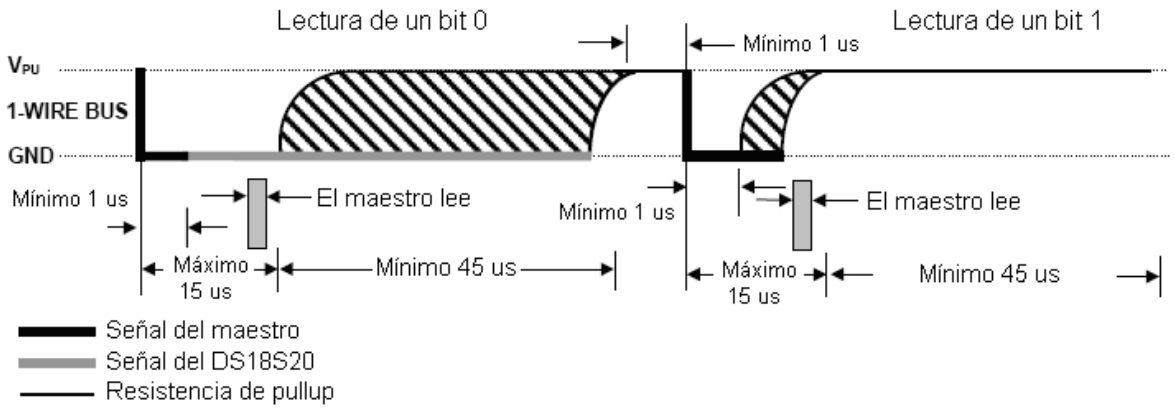


Figura 4.7: Lectura de un bit en el bus 1-Wire.

4.4 Lectura del ROM code

Antes de colocar los 3 sensores de temperatura en la computadora de vuelo hay que leer el ROM code de cada uno utilizando un microcontrolador PIC16F876A y un MAX232 colocados en un protoboard (ver figura 4.8).

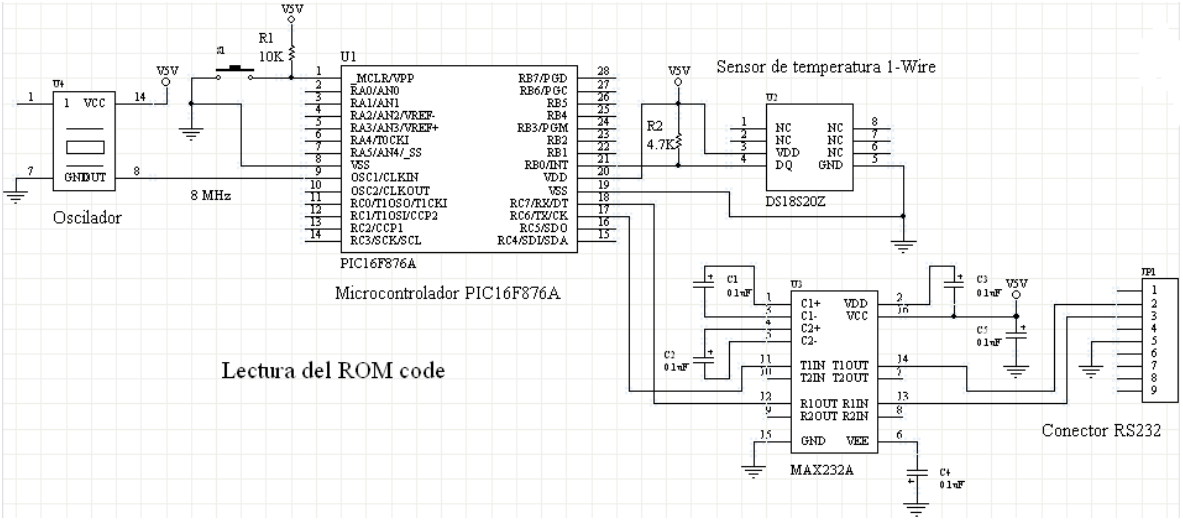


Figura 4.8: Diagrama del circuito para leer el ROM code de un sensor 1-Wire.

4.4.1 Software para el PIC16F876A

Los pasos para leer el ROM code de un sensor 1-Wire (ver figura 4.9) son: inicializar el sensor, enviarle el comando Read Rom [33h], recibir el ROM code de 64 bits y enviarlo por el puerto serial hacia la PC para poder desplegar y verificar el valor del ROM code.

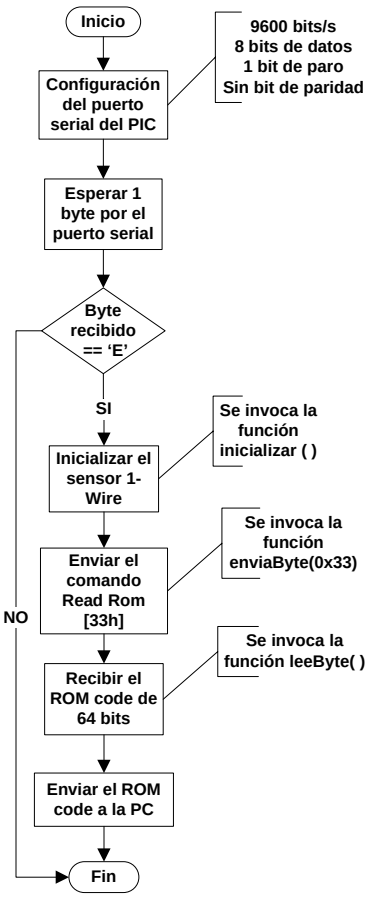


Figura 4.9: Diagrama de flujo del programa en el PIC16F876A para leer el ROM code (código en el apéndice D.1).

4.4.2 Interfaz en LabVIEW

La interfaz en LabVIEW (ver figura 4.10) envía por el puerto serial de la PC el caracter 'E', espera el ROM code de 8 bytes, despliega el ROM code y enciende el led si el valor del ROM code es correcto.

Figura 4.10: Interfaz en LabVIEW para leer el ROM code.

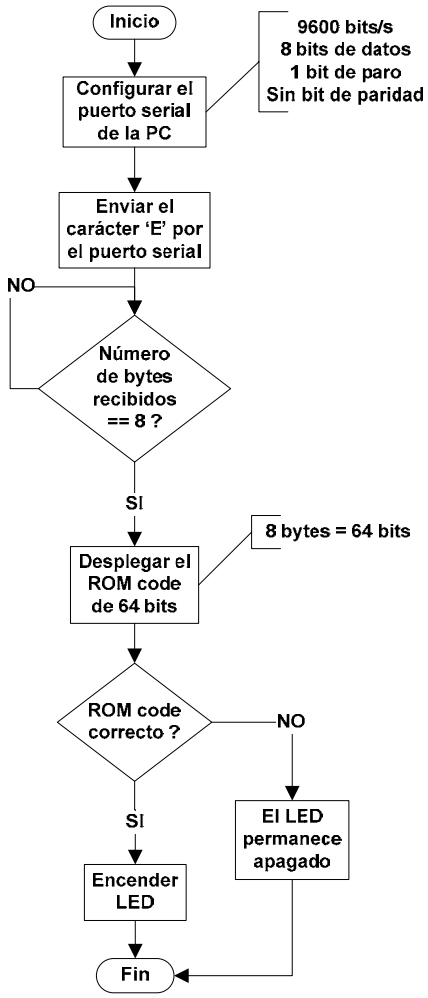


Figura 4.11: Diagrama de flujo del software en LabVIEW para desplegar y verificar el ROM code (código en el apéndice D.2).

El valor del ROM code se verifica utilizando el registro de corrimiento de la figura 4.2.

4.5 Lectura de los sensores de temperatura de la computadora de vuelo

4.5.1 Software en el SAB80C166

El software para leer los 3 sensores de temperatura de la computadora de vuelo (ver figura 4.12) espera un byte por el puerto serial 0, si el byte recibido es 00h entonces no se realiza ninguna operación, pero si es 01h entonces se inicializan los sensores, se envía el comando Convert T [44h] para que los 3 sensores tomen lecturas de temperatura, se envía el comando Match Rom [55h] y Read Scratchpad [BEh] para leer la temperatura en cada sensor y finalmente se envían las 3 temperaturas por el puerto serial hacia la PC para que sean desplegadas. Este software envía la lectura de los sensores de temperatura cada vez que recibe un byte 01h desde LabVIEW.

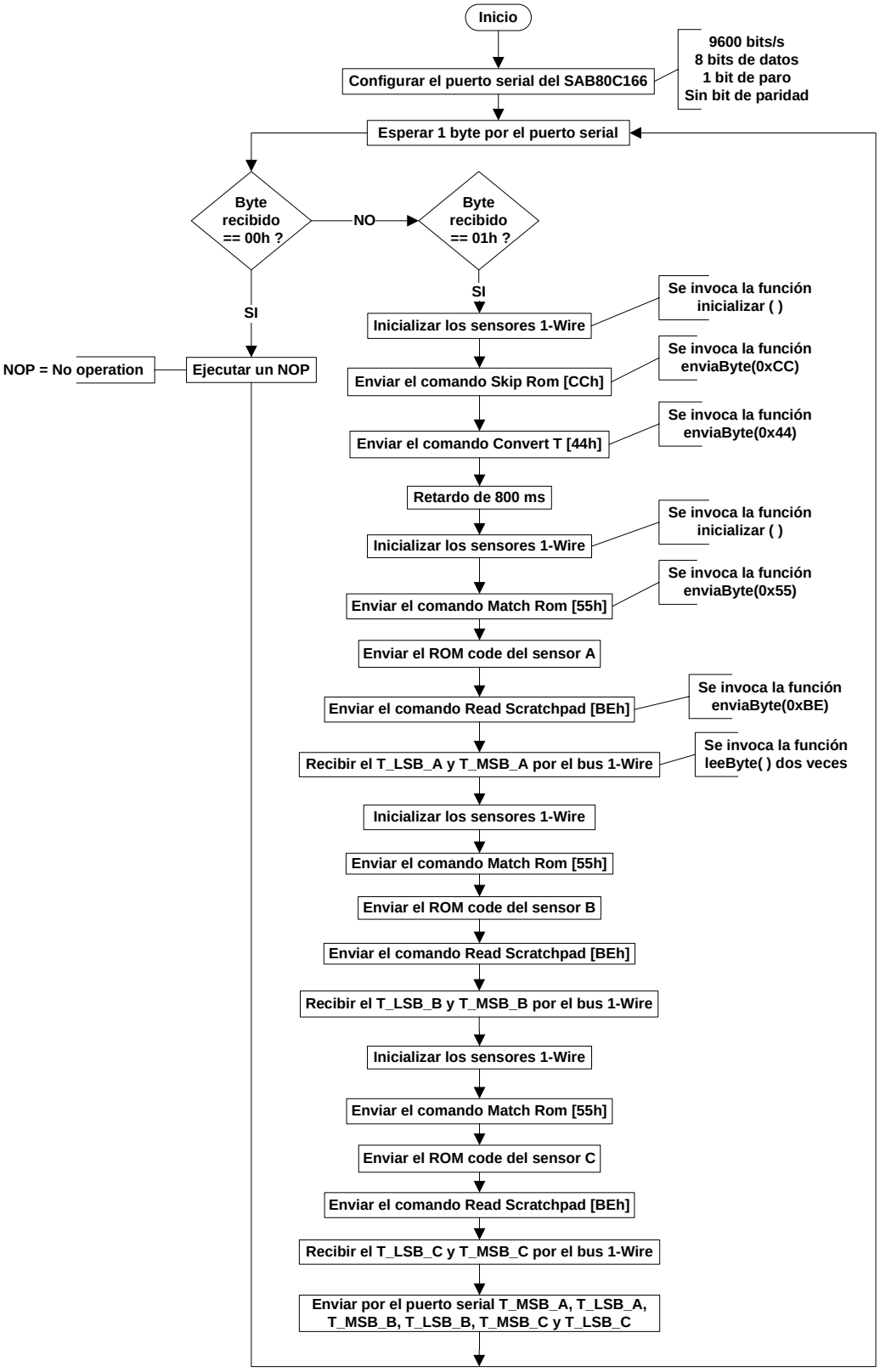


Figura 4.12: Diagrama de flujo del software para leer los sensores de temperatura de la computadora de vuelo (código en el apéndice D.3).

4.5.2 Interfaz en LabVIEW

La interfaz en LabVIEW (ver figura 4.13) se encuentra cíclicamente monitoreando el estado del interruptor que aparece en la pantalla. Cuando el usuario pone el interruptor en la posición “ON” se envía el byte 01h y se reciben 6 bytes que representan la temperatura en °C de los 3 sensores. Si el interruptor esta en la posición “OFF” se envía el byte “00h” y se ponen en 0 todos los indicadores.

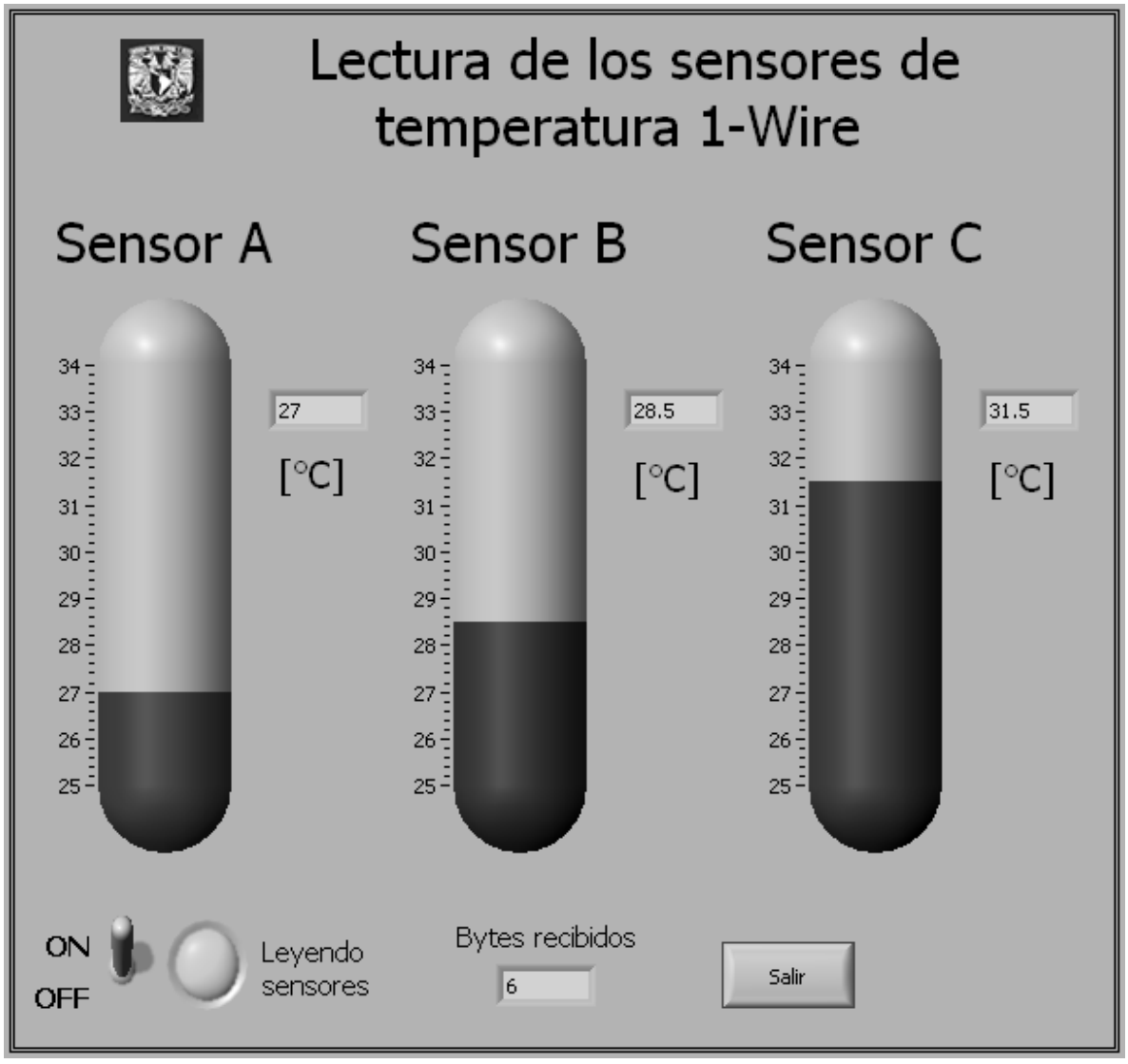


Figura 4.13: Interfaz en LabVIEW para desplegar las lecturas de los sensores de temperatura.

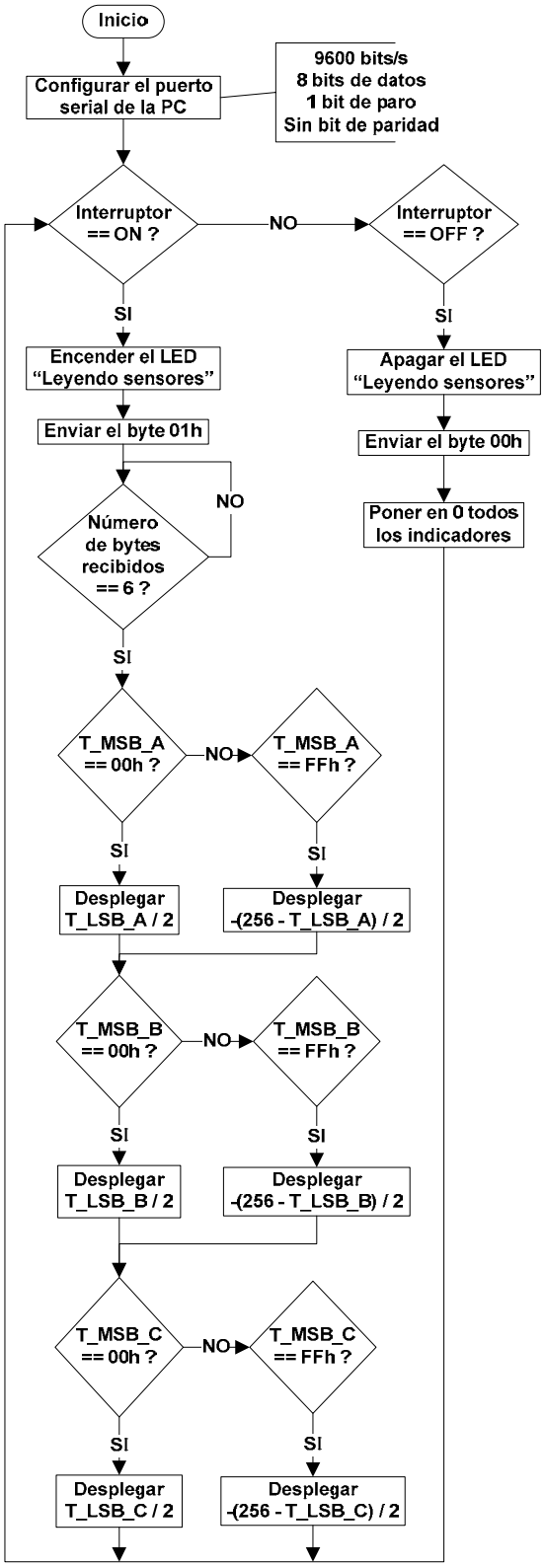


Figura 4.14: Diagrama de flujo del software en LabVIEW para visualizar las lecturas de temperatura (código en el apéndice D.4).

Capítulo 5 Memoria flash

5.1 Introducción

La computadora de vuelo cuenta con 32 MBytes de memoria flash. Esta memoria sirve para el almacenamiento de telemetría de todos los sensores del sistema, para guardar imágenes de una cámara digital y para guardar el programa principal del SAB80C166.

En este capítulo se describe la memoria flash AT45DB642D y se explica el software para la escritura, lectura y borrado de la memoria.

5.2 Memoria flash AT45DB642D

5.2.1 Organización de la memoria

El chip de memoria AT45DB642D cuenta con 8 MBytes de memoria principal tipo flash los cuales están organizados en páginas, bloques o sectores (ver figura 5.1). Una página consta de 1056 bytes de memoria, un bloque se forma con 8 páginas y un sector con 32 bloques (256 páginas). El sector 0 se divide en sector 0a y sector 0b. El sector 0a es de 8 páginas y el 0b de 248 paginas.

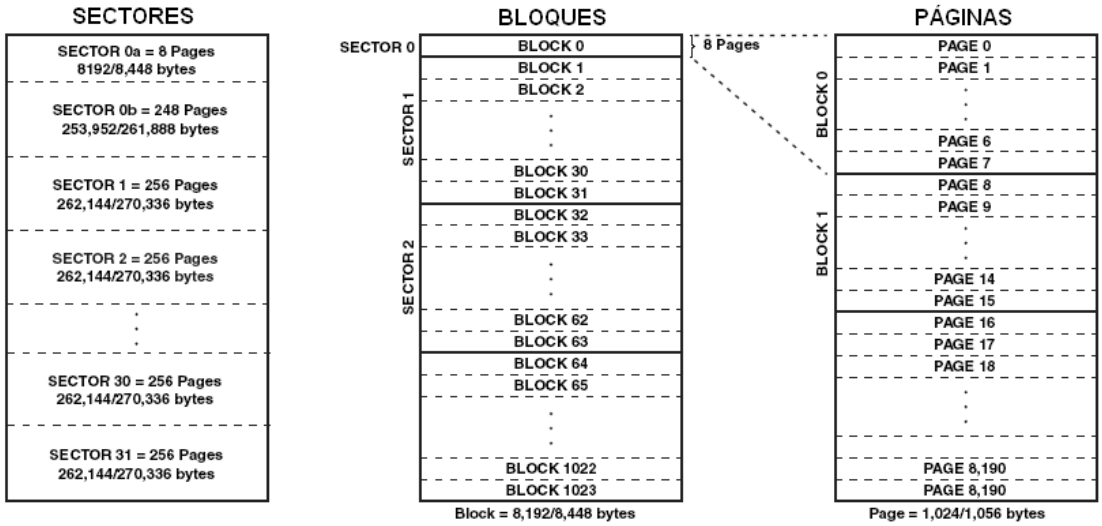


Figura 5.1: Organización de la memoria flash AT45DB642D.

Este chip cuenta con dos buffers de memoria SRAM de 1056 Bytes cada uno, los cuales son utilizados por el propio dispositivo en operaciones de escritura.

5.2.2 Descripción del funcionamiento

Para controlar la memoria flash se necesita una interfaz SPI de 4 pines: CS#, SCK, SI y SO (ver figura 5.2). La memoria AT45DB642D es un dispositivo SPI esclavo y por lo tanto recibe una señal de reloj, comandos y datos provenientes de un dispositivo maestro. En la computadora de vuelo el dispositivo maestro es el SAB80C166.

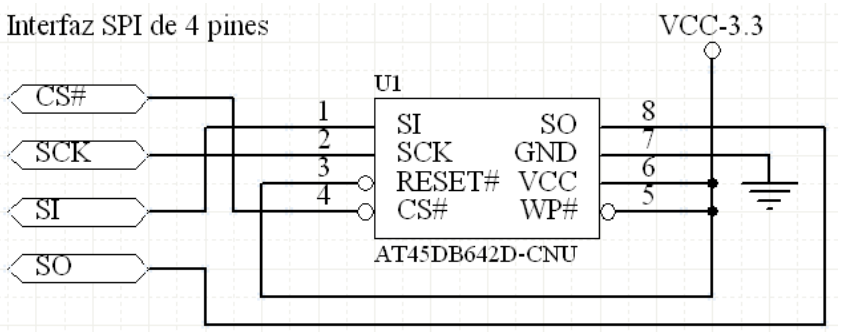


Figura 5.2: Interfaz SPI de 4 pines de la memoria flash AT45DB642D.

El pin WP# en alto sirve para que la memoria no este protegida contra escritura. El pin RESET# sirve para interrumpir externamente cualquier proceso de lectura, escritura o borrado que se este llevando acabo en la memoria. Para manejar la memoria el pin RESET# debe estar en alto.

La memoria flash estará inactiva siempre y cuando el dispositivo maestro mantenga el pin CS# en alto. Cuando el pin CS# se encuentra en alto el pin SO está en estado de alta impedancia y cualquier comando que se introduzca en el pin SI será ignorado. El maestro activa la memoria flash llevando el pin CS# a bajo y generando una señal de reloj de hasta 66 MHz en el pin SCK. A través del pin SI el dispositivo maestro introduce comandos y datos en la memoria flash. Por el pin SO el dispositivo maestro recibe datos desde la memoria flash. Los bits de información entran y salen de la memoria flash en sincronía con la señal de reloj SCK.

En la computadora de vuelo se tienen 4 memorias flash conectadas al SAB80C166 de la siguiente manera:

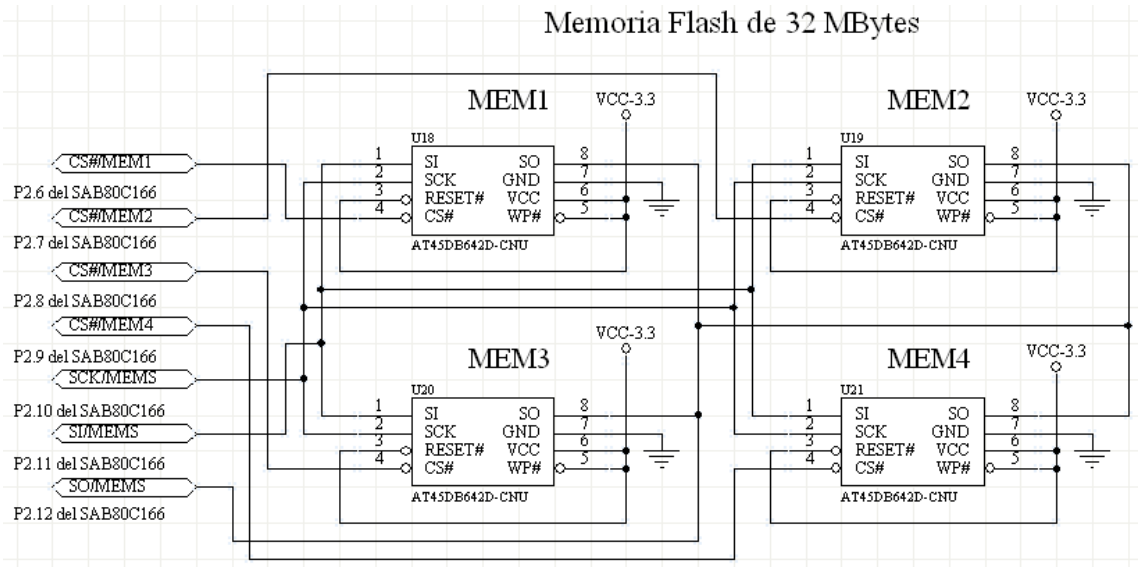


Figura 5.3: Conexión de 32 MBytes de memoria flash al SAB80C166.

Esta conexión permite que el SAB80C166 pueda comunicarse con cualquier memoria flash llevando su pin CS# a bajo y dejando los pines CS# de las demás memorias en alto.

5.2.3 Comandos de la memoria flash

La memoria flash cuenta con comandos de escritura, lectura y borrado. Todas las transferencias de comandos y datos comienzan por el bit más significativo (MSB).

- **Comandos de escritura**

Main memory page program through buffer. Este comando sirve para escribir bytes de información en cualquiera de los buffers SRAM y después automáticamente estos bytes se escriben en la página especificada de la memoria principal. El maestro debe enviar el comando 82h para utilizar el buffer 1 o 85h para el buffer 2. Después el maestro envía 3 bytes de dirección. Los primeros 13 bits (PA12 – PA0) especifican la página en la que se desea escribir y los últimos 11 bits (BFA10 – BFA0) especifican el byte en donde se comienza a escribir dentro del buffer. Finalmente el maestro envía los bytes a escribir en el buffer. Si la escritura de bytes llega al final del buffer la memoria continua escribiendo desde el principio del buffer. Cuando en el pin CS# hay una transición de bajo a alto la memoria automáticamente borra la pagina seleccionada (escribe 1s) y después escribe el contenido del buffer en la pagina. El tiempo máximo de borrado y escritura de una pagina es tEP = 40 ms.

- **Comandos de lectura**

Main memory page read. Este comando sirve para leer alguna de las 8,192 páginas de la memoria principal. El maestro debe enviar el comando D2h. Después el maestro envía 3

bytes de dirección. Los primeros 13 bits (PA12 – PA0) especifican la pagina que se quiere leer y los últimos 11 bits (BA10 – BA0) el byte que se comienza a leer dentro de la pagina. A continuación el maestro envía 4 bytes tipo don't care (cualquier valor). Finalmente el maestro recibe los bytes de información por el pin SO. Cuando la lectura llega al final de la página, la memoria continúa leyendo desde el principio de la misma pagina. La lectura se finaliza con una transición de bajo a alto en el pin CS#.

Continuous array read. Este comando sirve para leer todo el contenido de la memoria principal a partir de la dirección especificada. El maestro debe enviar el comando E8h. Después el maestro envía 3 bytes de dirección. Los primeros 13 bits (PA12 – PA0) especifican la pagina que se quiere leer y los últimos 11 bits (BA10 – BA0) el byte que se comienza a leer dentro de la pagina. A continuación el maestro envía 4 bytes tipo don't care (cualquier valor). Finalmente el maestro recibe los bytes de información por el pin SO. Cuando la lectura llega al final de la memoria principal, la memoria continúa leyendo desde el principio de la memoria principal. La lectura se finaliza con una transición de bajo a alto en el pin CS#.

- **Comandos de borrado**

Page erase. Este comando sirve para borrar cualquier página de la memoria principal. El maestro debe enviar el comando 81h. Después el maestro envía 3 bytes de dirección. Los primeros 13 bits (PA12 – PA0) especifican la pagina que se quiere borrar y los últimos 11 bits son tipo don't care (cualquier valor). El borrado de la pagina comienza con una transición de bajo a alto en el pin CS#. La pagina borrada se lee como 1s. El tiempo máximo de borrado es tPE = 35 ms.

Block erase. Este comando sirve para borrar cualquier bloque de la memoria principal. El maestro debe enviar el comando 50h. Después el maestro envía 3 bytes de dirección (A23 – A0). Los bits A12 – A3 especifican el bloque a borrar y los 14 bits restantes son tipo don't care (cualquier valor). El borrado del bloque comienza con una transición de bajo a alto en el pin CS#. El tiempo máximo de borrado es tBE = 100ms.

5.3 Interfaz SPI

La interfaz SPI de la memoria flash AT45DB642D consta de 4 pines: CS#, SCK, SI y SO. Mientras el pin CS# este en alto cualquier comando que sea introducido por el pin SI es ignorado y el pin SO estará en estado de alta impedancia. Cuando en el pin CS# hay una transición de alto a bajo el maestro debe de generar en el pin SCK una señal de reloj con un ciclo de trabajo del 50 % y frecuencia máxima de 66 MHz. En cada flanco de bajada de la señal de reloj el maestro debe de introducir un bit por el pin SI. En cada flanco de subida el maestro debe de leer los bits por el pin SO.

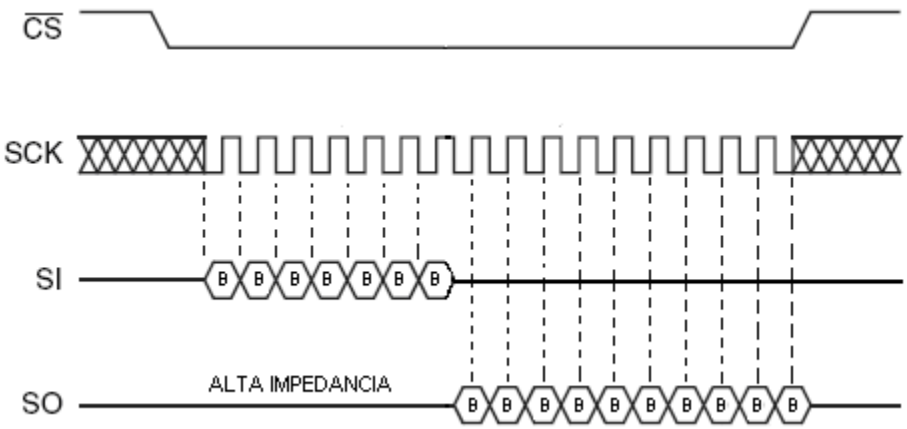


Figura 5.4: Trafico de datos en una interfaz SPI.

El SAB80C166 no cuenta con hardware para una interfaz SPI de 4 pines por lo que se tuvo que programar una utilizando pines I/O.

5.4 Escritura, lectura y borrado de la memoria flash

5.4.1 Software para el SAB80C166

El software para manejar la memoria flash con el SAB80C166 (ver figura 5.8) recibe comandos de escritura, lectura y borrado provenientes de LabVIEW.

El comando para escribir en la memoria flash (ver figura 5.5) consta de 9 bytes que son: la letra ‘E’, la memoria en la que se desea escribir, el buffer SRAM a utilizar, la pagina a escribir, la localidad dentro de la pagina en donde se comienza a escribir y el numero de bytes a escribir.

‘E’ (1 Byte)	Memoria (1 Byte)	Buffer SRAM (1 Byte)	Pagina (2 Bytes)	Localidad (2 Bytes)	Numero de bytes (2 Bytes)

Figura 5.5: Comando para escribir en la memoria flash.

Una vez recibido el comando, el SAB80C166 envía la letra ‘K’ a LabVIEW para después recibir los bytes a escribir. La escritura se realiza utilizando el comando *Main memory page program through buffer* (ver figura 5.9). Cuando la escritura finaliza el SAB80C166 envía la letra ‘K’ a LabVIEW. Utilizando este comando se pueden escribir bytes solamente en una página (1056 Bytes). Si se quieren escribir mas de 1056 Bytes entonces el comando de escritura debe enviarse varias veces.

El comando para leer la memoria flash (ver figura 5.6) consta de 8 bytes que son: la letra ‘L’, la memoria que se quiere leer, la página a leer, la localidad dentro de la página en donde se comienza a leer y el número de bytes a leer.

‘L’ (1 Byte)	Memoria (1 Byte)	Página (2 Bytes)	Localidad (2 Bytes)	Numero de bytes (2 Bytes)

Figura 5.6: Comando para leer la memoria flash.

La lectura se realiza utilizando el comando *Main memory page read* (ver figura 5.10). Una vez finalizada la lectura, el SAB80C166 envía los bytes leídos a LabVIEW. Utilizando este comando se pueden leer bytes de una sola página (1056 Bytes). Si se quieren leer mas de 1056 Bytes entonces el comando de lectura debe enviarse varias veces.

El comando para borrar la memoria flash (ver figura 5.7) consta de 5 bytes que son: la letra ‘B’, un byte 01h para borrar una página o 02h para borrar un bloque, la memoria a borrar y una dirección de 2 bytes que puede ser el número de página o bloque.

‘B’ (1 Byte)	01h: Pagina ó 02h: Bloque (1 Byte)	Memoria (1 Byte)	Dirección (2 Bytes)

Figura 5.7: Comando para borrar la memoria flash.

El borrado de página se realiza utilizando el comando *Page erase* y el de bloque con *Block erase* (ver figura 5.11). Cuando el borrado de la memoria flash finaliza, el SAB80C166 envía la letra ‘K’ a LabVIEW.

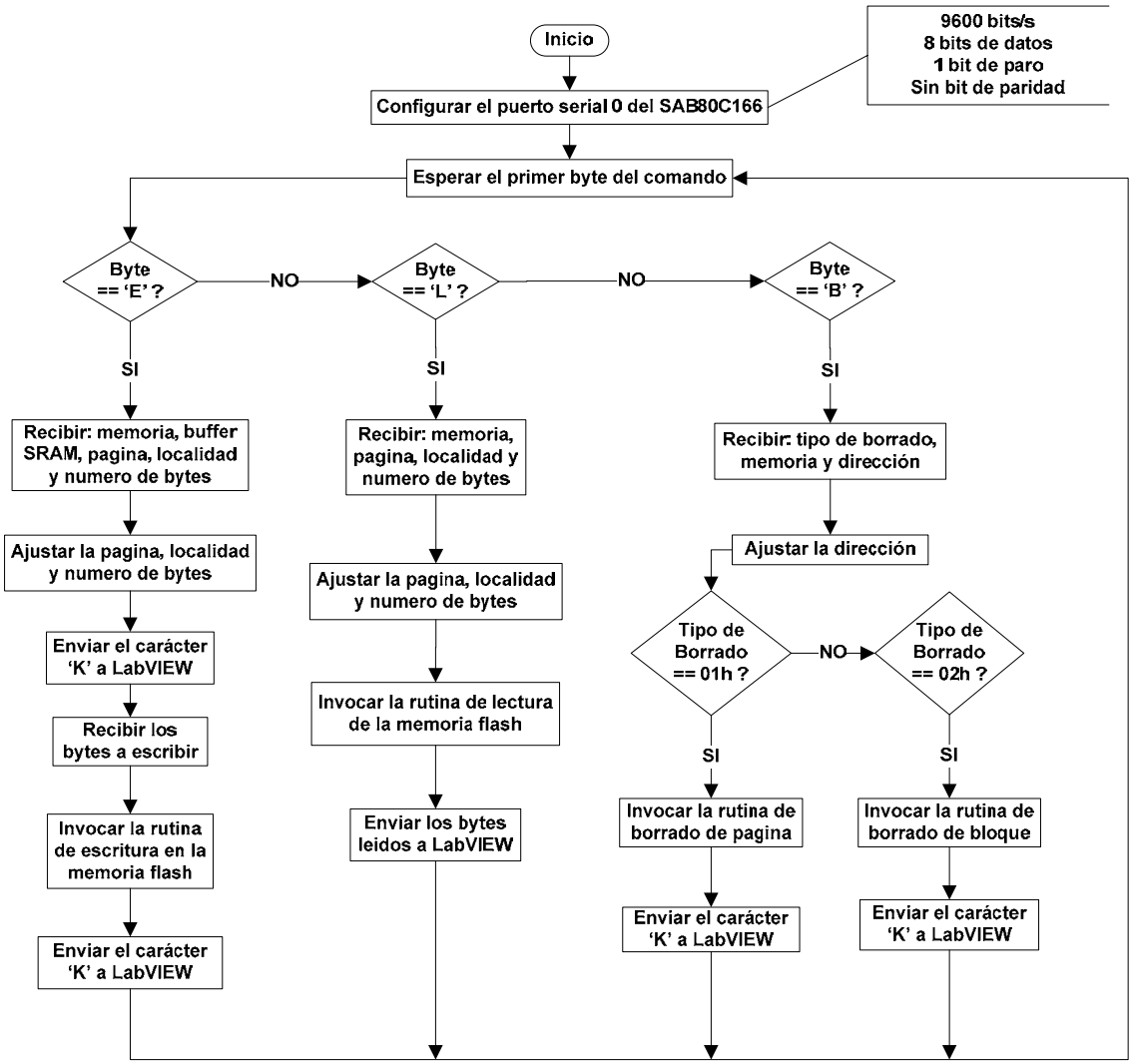


Figura 5.8: Diagrama de flujo del software en el SAB80C166 para la escritura, lectura y borrado de la memoria flash (código en el apéndice E.1).

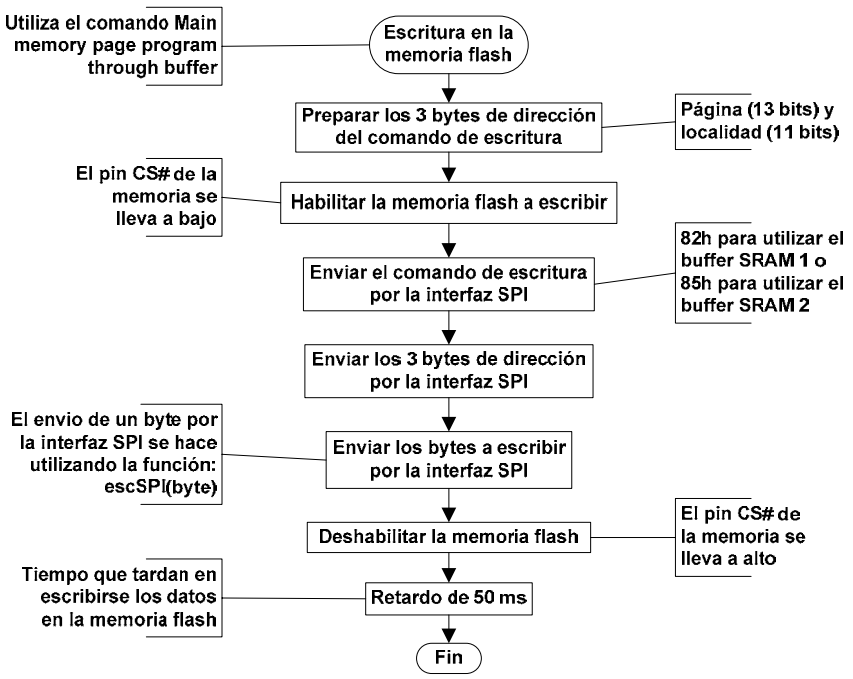


Figura 5.9: Diagrama de flujo de la rutina de escritura en la memoria flash (código en el apéndice E.1).

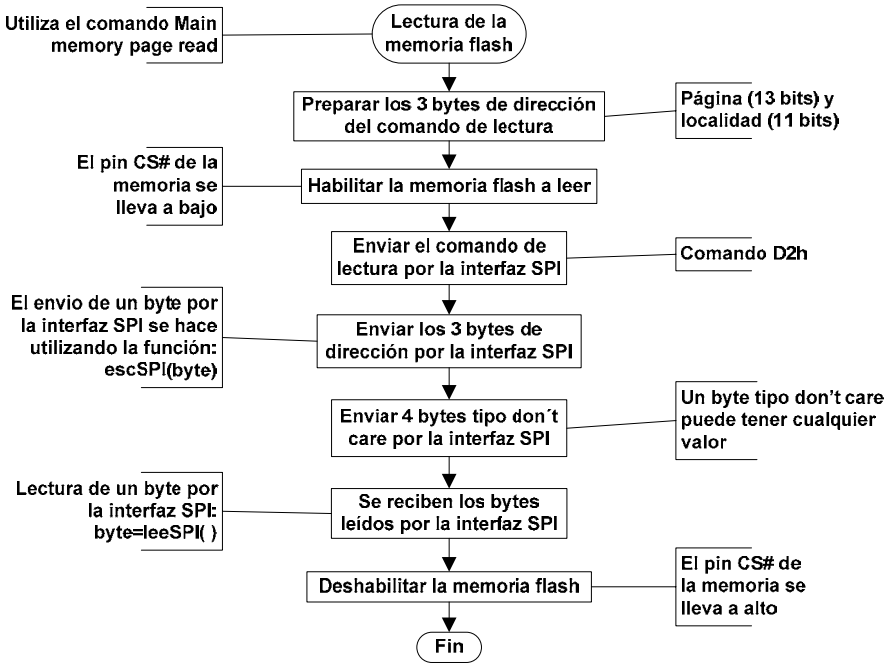


Figura 5.10: Diagrama de flujo de la rutina de lectura de la memoria flash (código en el apéndice E.1).

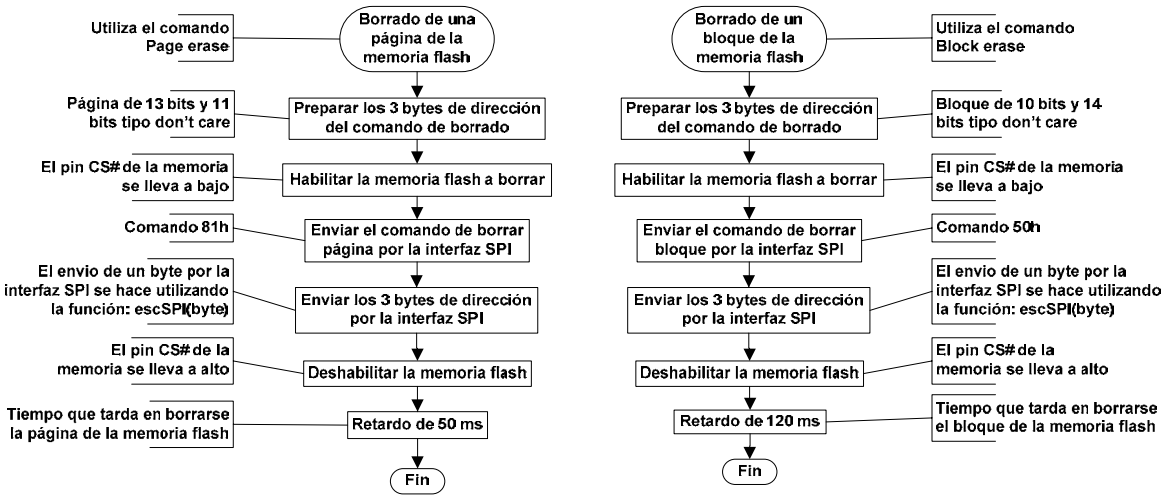


Figura 5.11: Diagrama de flujo de las rutinas de borrado de página y de bloque de la memoria flash (código en el apéndice E.1).

5.4.2 Interfaz en LabVIEW

Para escribir en la memoria flash (ver figura 5.12) el usuario selecciona la memoria, el buffer SRAM a utilizar, la página y la localidad en donde se comienza a escribir. Después el usuario introduce los bytes a escribir o abre un archivo Intel HEX. Los bytes introducidos en el campo "Bytes a escribir" pueden separarse con un espacio. Si el usuario abre un archivo Intel HEX este se despliega en el campo "Bytes a escribir" sin caracteres ':' ni finales de línea y los bytes aparecen separados con un espacio. Al presionar el botón "Escribir" (ver figura 5.13) el led "Escribiendo" se enciende y LabVIEW envía comandos de escritura y datos las veces que sea necesario para que el SAB80C166 escriba en la memoria flash todos los bytes. El led "Escribiendo" se apaga cuando finaliza la escritura de la memoria flash.

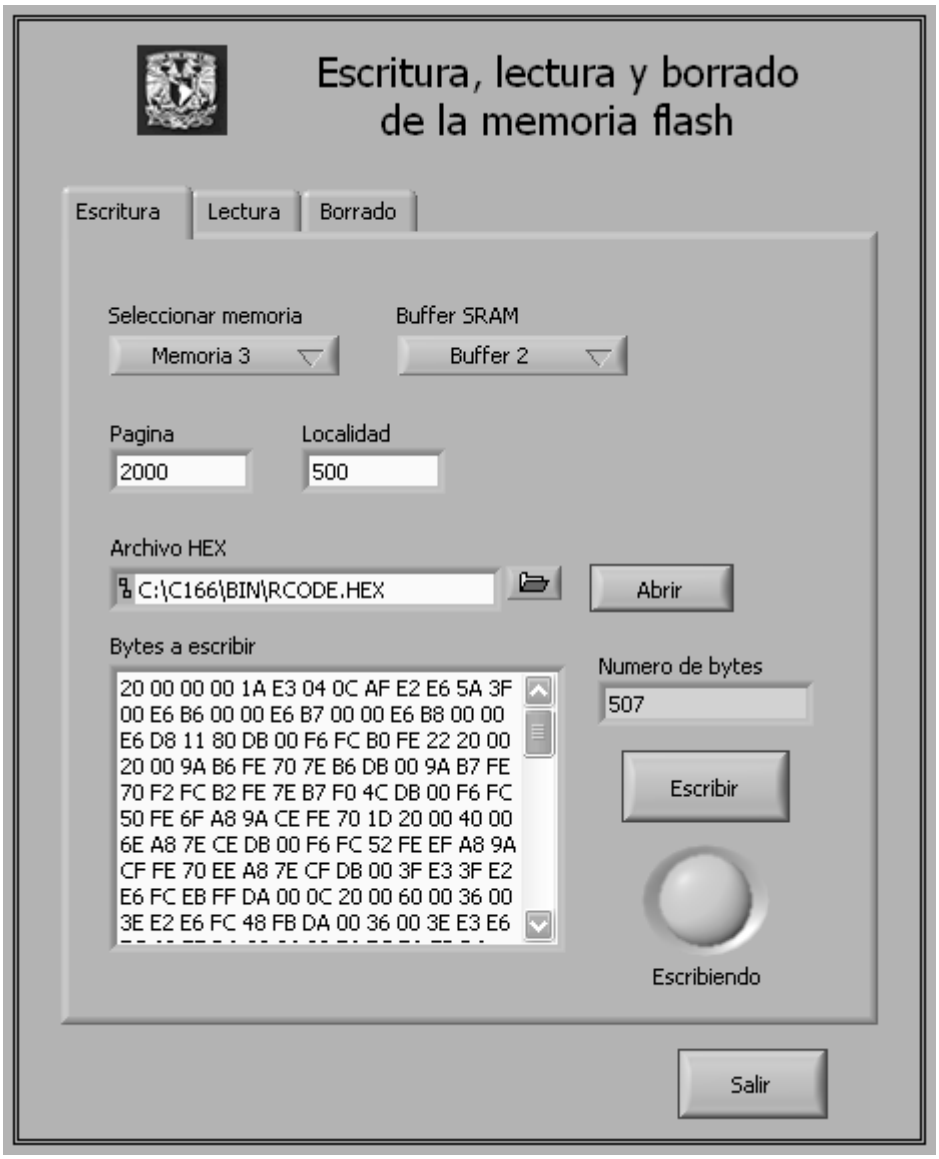


Figura 5.12: Interfaz en LabVIEW para escribir en la memoria flash.

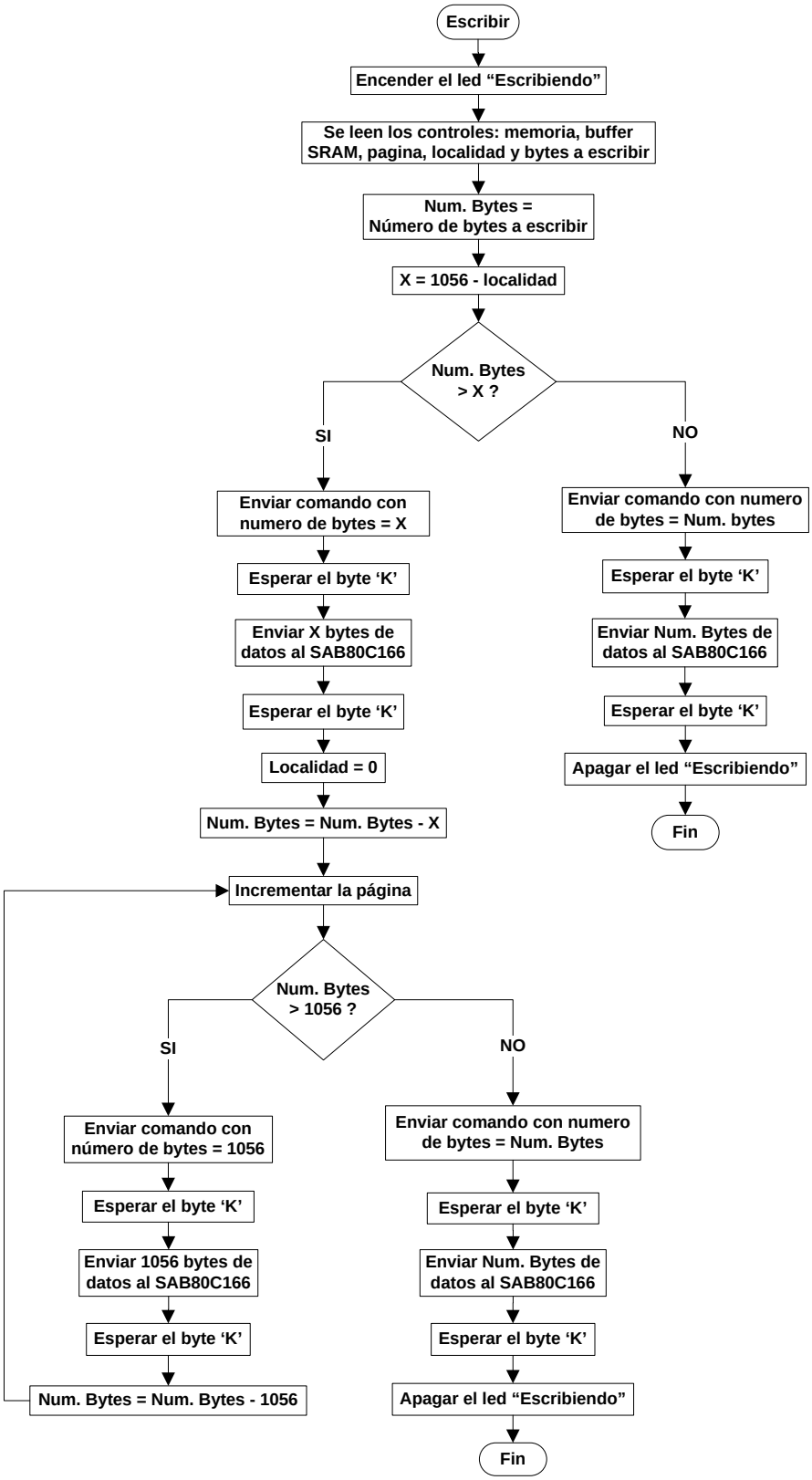


Figura 5.13: Diagrama de flujo del software en LabVIEW para escribir datos en la memoria flash (código en el apéndice E.2).

Para leer la memoria flash (ver figura 5.14) el usuario selecciona la memoria, la página, la localidad de inicio y la cantidad de bytes a leer. Al presionar el botón “Leer” (ver figura 5.15) el led “Leyendo” se enciende y LabVIEW envía comandos de lectura y recibe datos las veces necesarias para leer la cantidad de bytes indicada. El led “Leyendo” se apaga cuando finaliza la lectura de la memoria flash y en el campo “Bytes leídos” se muestra el resultado de la lectura. El usuario puede guardar los bytes leídos en un archivo presionando el botón “Guardar”.

Figura 5.14: Interfaz en LabVIEW para leer la memoria flash.

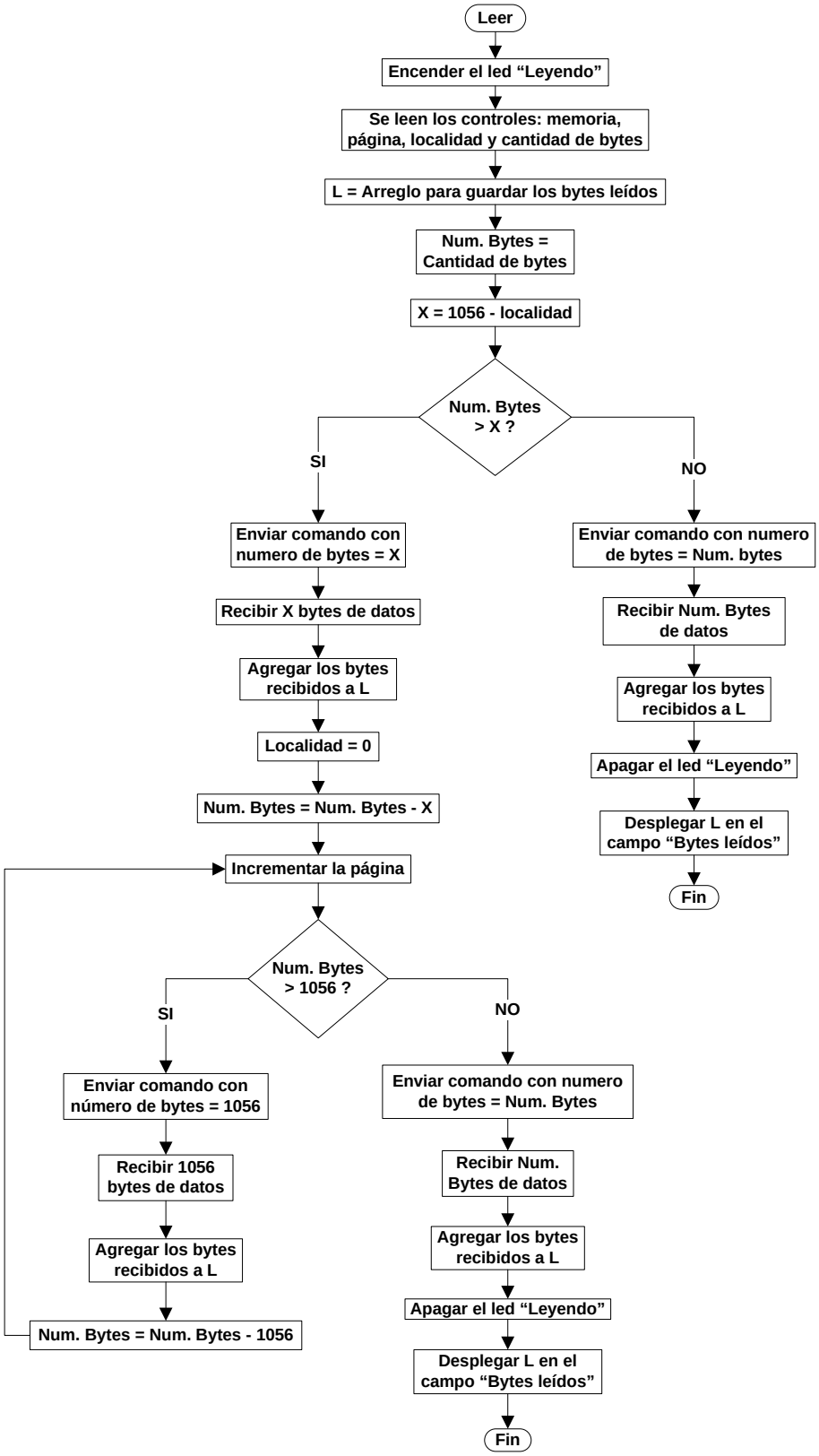


Figura 5.15: Diagrama de flujo del software en LabVIEW para leer la memoria flash (código en el apéndice E.2).

Para borrar la memoria flash (ver figura 5.16) el usuario selecciona la memoria, el tipo de borrado e introduce la página o bloque a borrar. Al presionar el botón “Borrar” (ver figura 5.17) el led “Borrando” se enciende y LabVIEW envía el comando de borrado. El led “Borrando” se apaga cuando LabVIEW recibe la letra ‘K’ lo que significa que ha finalizado el borrado de la memoria flash.

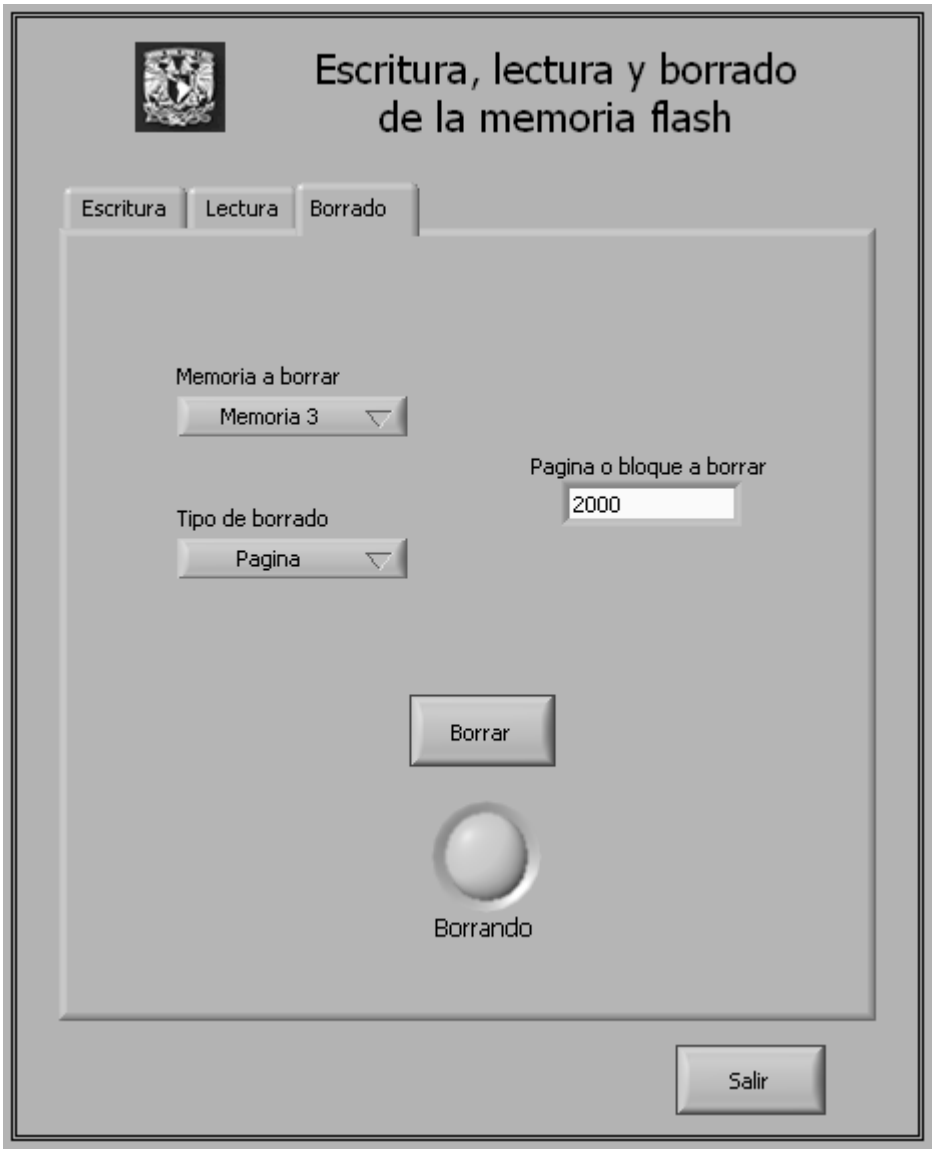


Figura 5.16: Interfaz en LabVIEW para borrar la memoria flash.

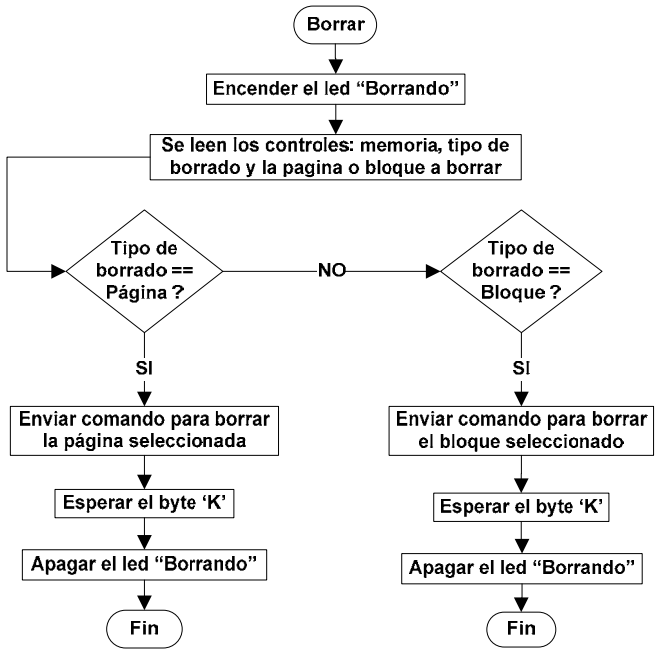


Figura 5.17: Diagrama de flujo del software en LabVIEW para borrar la memoria flash (código en el apéndice E.2).

Capítulo 6
Protecciones contra efecto latch-up

6.1 Introducción

Cuando los semiconductores son expuestos a la radiación ionizante, como la del espacio, presentan el efecto latch-up que se manifiesta con la elevación en el consumo de corriente eléctrica. Entre mayor sea el nivel de integración el efecto latch-up es mayor. Si esta elevación en el consumo de la corriente eléctrica no se detiene el dispositivo puede dañarse permanentemente. Una forma de detener el efecto latch-up es desenergizando el dispositivo. En la computadora de vuelo la electrónica para detectar el efecto latch-up consiste principalmente en un circuito integrado MAX4071 y un comparador de voltaje LM6511 los cuales generan la señal eléctrica para que la tarjeta de potencia desenergize los dispositivos electrónicos. En la computadora el SAB80C166, el PIC16F876A y la memoria RAM están protegidos contra efecto latch-up. La computadora cuenta con un convertidor A/D 1-Wire DS2450 que sirve para medir la corriente eléctrica consumida por los dispositivos protegidos contra efecto latch-up.

En este capítulo se explica el procedimiento para diseñar las protecciones contra efecto latch-up y el software para leer el convertidor A/D 1-Wire DS2450.

6.2 El circuito integrado MAX4071

El circuito integrado MAX4071 se utiliza para el monitoreo de corriente eléctrica (ver figura 6.1).

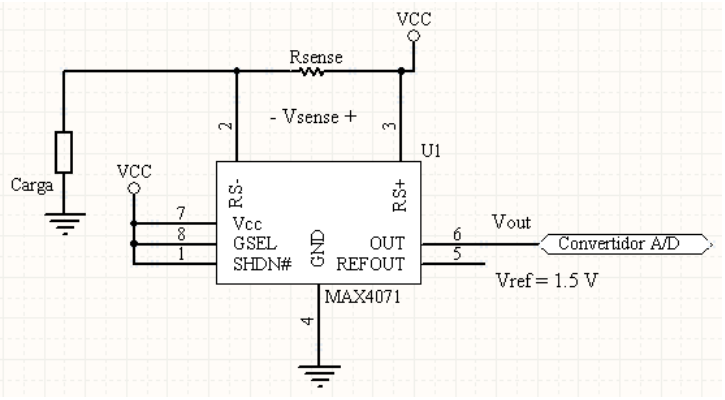


Figura 6.1: Monitoreo de corriente eléctrica utilizando el MAX4071.

El MAX4071 necesita de una resistencia externa llamada Rsense por la cual debe fluir la corriente eléctrica que se desea monitorear. Si en el pin GSEL se aplica un voltaje alto se tendrá una ganancia de 100 V/V y con un voltaje bajo 50 V/V. En el pin REFOUT hay un voltaje de referencia constante de 1.5 V. El voltaje respecto a tierra en el pin Vout es

directamente proporcional a la corriente eléctrica que fluye a través de Rsense y está dado por la siguiente ecuación:

$$V_{out} = G_{sel} R_{sense} I_{Rsense} + 1.5 \quad \text{---------- (1)}$$

En la tabla 6.1 se muestran los valores recomendados por el fabricante para Rsense. Al seleccionar el valor de Rsense hay que tomar en cuenta que se debe elegir el mínimo valor posible para que la caída de voltaje en Rsense sea mínima. La resistencia Rsense debe tener una inductancia pequeña si la corriente a monitorear varía mucho.

Corriente máxima [A]	**Rsense [mΩ]**	**Ganancia [V/V]**	**Vsense [mV]**	**Vout – Vref [V]**
0.075	1000	50	75	3.75
0.05	1000	100	50	5.0
0.75	100	50	75	3.75
0.5	100	100	50	5.0
3.75	20	50	75	3.75
2.5	20	100	50	5.0
7.5	10	50	75	3.75
5.0	10	100	50	5.0
15.0	5	50	75	3.75
10.0	5	100	50	5.0

Tabla 6.1: Valores recomendados de Rsense.

6.3 El comparador LM6511

El circuito integrado LM6511 es un comparador de voltaje con salida tipo colector abierto. Su voltaje de alimentación es de 2.7 V a 36 V y tiene un tiempo de respuesta de $180\eta s$. En la figura 6.2 se muestra al LM6511 conectado como un comparador de lazo abierto.

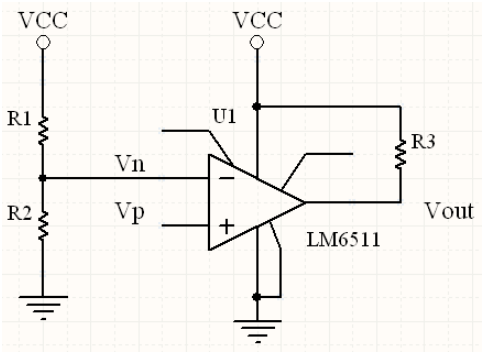


Figura 6.2: Circuito integrado LM6511 conectado como comparador de lazo abierto.

El voltaje Vn esta dado por la siguiente ecuación:

$$V_n = R_2 \frac{V_{cc}}{R_1 + R_2}$$

De la ecuación:

$$V_{out} = A_v (V_p - V_n)$$

Donde Av es la ganancia de voltaje de lazo abierto y tiende a infinito, se tiene que mientras Vp sea menor que Vn Vout valdrá cero y para que Vout valga Vcc se necesita que Vp sea mayor a Vn. Sea Vref un voltaje de referencia aplicado en el pin Vp entonces el valor de R1 y R2 para que Vn sea igual a Vref es:

$$V_{ref} = V_n$$
$$V_{ref} = R_2 \frac{V_{cc}}{R_1 + R_2}$$
$$V_{ref}(R_1 + R_2) = R_2 V_{cc}$$
$$R_2(V_{ref} - V_{cc}) = -V_{ref} R_1$$

$$R_1 = R_2\left(\frac{V_{cc}}{V_{ref}} - 1\right) \quad \text{---------- (2)}$$

Dado un valor de R2 se obtiene un valor para R1. Si Vp es menor a Vref entonces Vout valdrá cero y para que la salida cambie a Vcc Vp debe de ser mayor a Vref.

6.4 Diseño de las protecciones contra efecto latch-up

6.4.1 Protección del SAB80C166

La corriente máxima de alimentación del SAB80C166 es de 150mA = 0.15A. De la tabla 6.1 se tiene que el valor de Rsense más conveniente es el de 100 mΩ = 0.1Ω y ganancia de 100 V/V para así poder medir una corriente máxima de hasta 0.5A. De la ecuación (1) tenemos que el valor de Vout para I = 0.14A es:

$$V_{out} = (100)(0.1)(0.14) + 1.5[V] = 2.9[V]$$

Sustituyendo en la ecuación (2) Vref = 2.9 V y R2 = 2.7 kΩ tenemos que R1 vale:

$$R_1 = (2.7k\Omega)\left(\frac{5}{2.9} - 1\right) = 1.9551[k\Omega] \approx 2[k\Omega]$$

La protección contra efecto latch-up del SAB80C166 queda de la siguiente manera:

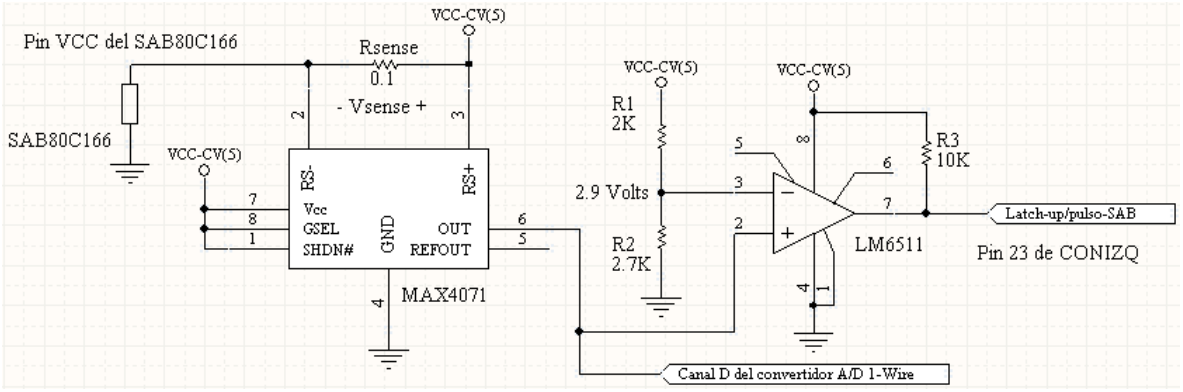


Figura 6.3: Protección contra efecto latch-up del SAB80C166.

6.4.2 Protección del PIC16F876A

La corriente máxima de alimentación del PIC16F876A es de 250mA = 0.25A. De la tabla 6.1 se tiene que el valor de Rsense mas conveniente es el de 100 mΩ = 0.1Ω y ganancia de 100 V/V para así poder medir una corriente máxima de hasta 0.5A. De la ecuación (1) tenemos que el valor de Vout para I = 0.24A es:

$$V_{out} = (100)(0.1)(0.24) + 1.5[V] = 3.9[V]$$

Sustituyendo en la ecuación (2) Vref = 3.9 V y R2 = 3.9 kΩ tenemos que R1 vale:

$$R_1 = (3.9k\Omega)\left(\frac{5}{3.9} - 1\right) = 1.1[k\Omega]$$

La protección contra efecto latch-up del PIC16F876A queda de la siguiente manera:

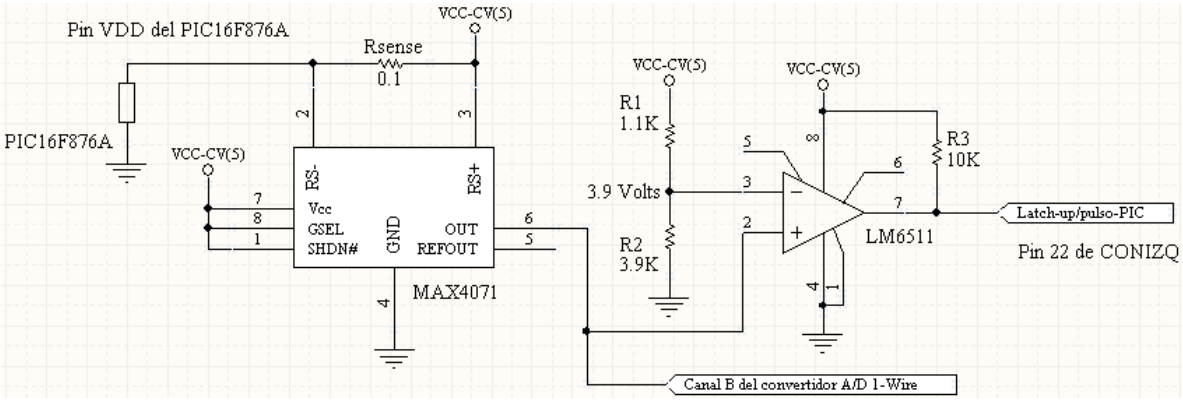


Figura 6.4: Protección contra efecto latch-up del PIC16F876A.

6.4.3 Protección de la memoria RAM

La corriente máxima de alimentación de cada memoria RAM CY7C109B-25VC es de 70 mA = 0.07A. Las 2 memorias RAM juntas tienen una corriente máxima de alimentación de 140 mA = 0.14A. De la tabla 6.1 se tiene que el valor de Rsense mas conveniente es el de 100 mΩ = 0.1Ω y ganancia de 100 V/V para así poder medir una corriente máxima de hasta 0.5A. De la ecuación (1) tenemos que el valor de Vout para I = 0.13A es:

$$V_{out} = (100)(0.1)(0.13) + 1.5[V] = 2.8[V]$$

Sustituyendo en la ecuación (2) Vref = 2.8 V y R2 = 1.5 kΩ tenemos que R1 vale:

$$R_1 = (1.5k\Omega)\left(\frac{5}{2.8} - 1\right) = 1.1785[k\Omega] \approx 1.2[k\Omega]$$

La protección contra efecto latch-up de la memoria RAM queda de la siguiente manera:

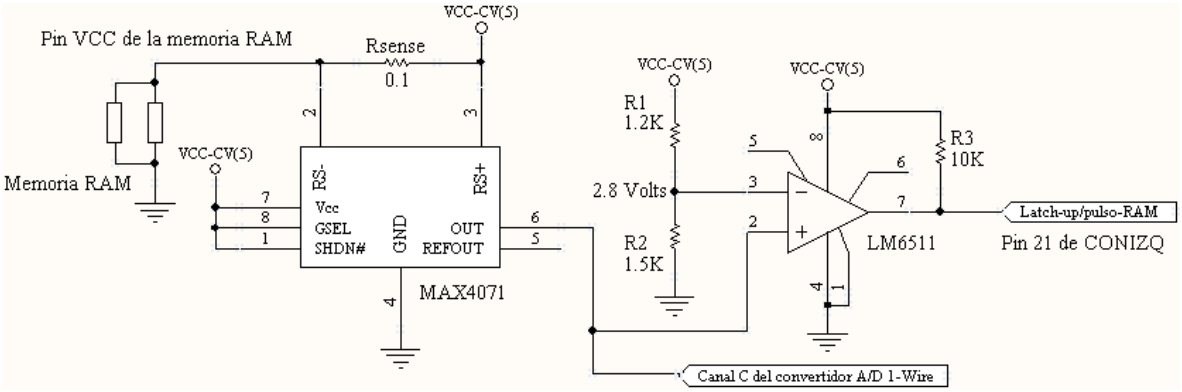


Figura 6.5: Protección contra efecto latch-up de la memoria RAM.

6.5 Convertidor A/D 1-Wire DS2450

Los canales del convertidor A/D 1-Wire DS2450 están conectados al pin de salida de los circuitos MAX4071. De esta manera el SAB80C166 puede tomar lecturas del voltaje de salida de cada circuito MAX4071 y determinar la corriente eléctrica consumida por el SAB80C166, el PIC16F876A y la memoria RAM.

6.5.1 ROM code

Cada dispositivo 1-Wire tiene un número serial único de 64 bits llamado ROM code (ver figura 6.6) que se encuentra almacenado en la memoria ROM interna. Los 8 bits menos significativos del ROM code representan la familia del dispositivo que en el caso del DS2450 es 20h. Los siguientes 48 bits son un número serial único. Los 8 bits más

significativos son el CRC calculado a partir de los 56 bits menos significativos. El CRC sirve para verificar que se ha leído correctamente el ROM code.

CRC de 8 bits	Número serial de 48 bits	Familia de 8 bits (20h)
MSB LSB	MSB LSB	MSB LSB

Figura 6.6: ROM code de 64 bits.

El ROM code se verifica utilizando el siguiente registro de corrimiento:

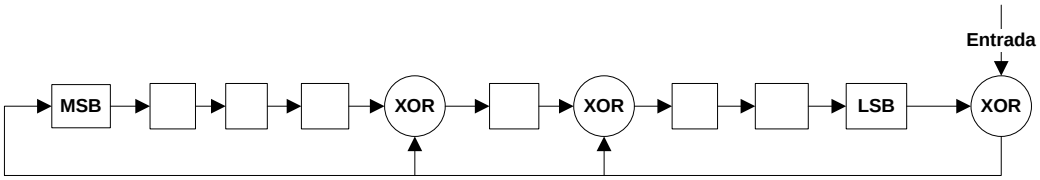


Figura 6.7: Registro de corrimiento para verificar el ROM code.

Todos los bits del registro de corrimiento comienzan con un valor de cero, después se introduce el ROM code comenzando por el bit menos significativo. Si después de introducir todos los bits del ROM code el registro de corrimiento vale cero entonces el ROM code es correcto.

6.5.2 Memoria interna

El DS2450 tiene una memoria SRAM interna de 32 bytes que está organizada en 4 páginas de 8 bytes cada una.

Página 0. La página 0 (ver figura 6.8) es de solo lectura y se utiliza para almacenar los resultados de las conversiones A/D. La máxima resolución de una conversión A/D es de 16 bits. Si se utiliza una resolución menor a 16 bits entonces el bit más significativo del registro de memoria SRAM será el bit más significativo del resultado de la conversión A/D y los bits menos significativos del registro se llenarán con ceros. En aplicaciones que requieren menos de 4 canales se debe comenzar a utilizar por el canal D.

Dirección	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
00	A	A	A	A	A	A	A	LSBIT A
01	MSBIT A	A	A	A	A	A	A	A
02	B	B	B	B	B	B	B	LSBIT B
03	MSBIT B	B	B	B	B	B	B	B
04	C	C	C	C	C	C	C	LSBIT C
05	MSBIT C	C	C	C	C	C	C	C
06	D	D	D	D	D	D	D	LSBIT D
07	MSBIT D	D	D	D	D	D	D	D

Figura 6.8: Página 0 de la memoria SRAM interna del DS2450.

Página 1. La página 1 (ver figura 6.9) se utiliza para configurar y controlar cada canal del convertidor A/D. Se tienen 2 bytes asignados para cada canal. Los bits RC3 – RC0 sirven

para especificar la resolución del convertidor A/D. Un valor de 1111 dará una resolución de 15 bits, 0000 dará una resolución de 16 bits. Para activar el convertidor A/D el bit OE (output enable) debe valer 0. Si el bit OE vale 1 entonces hay que conectar una resistencia de pull up a un voltaje positivo en el pin del canal para que al escribir un 1 en el bit OC (output control) la salida sea alto y al escribir un cero sea bajo. Si el bit IR vale 0 entonces el rango del voltaje de entrada es de 0V – 2.55V y si vale 1 es de 0V – 5.10V. Los siguientes dos bits AEL (alarm enable low) y AEH (alarm enable high) sirven para habilitar el umbral bajo y/o alto de la alarma. Los bits AFL (alarm flag low) y AFH (alarm flag high) son las banderas de la alarma. El bit POR (power on reset) vale 1 mientras el DS2450 se encuentra en una secuencia de power on reset. Después de que se enciende el dispositivo, el maestro debe escribir un 0 en el bit POR.

Dirección	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
08	OE - A	OC - A	0	0	RC3 - A	RC2 - A	RC1 - A	RC0 - A
09	POR	0	AFH - A	AFL - A	AEH - A	AEL - A	0	IR - A
0A	OE - B	OC - B	0	0	RC3 - B	RC2 - B	RC1 - B	RC0 - B
0B	POR	0	AFH - B	AFL - B	AEH - B	AEL - B	0	IR - B
0C	OE - C	OC - C	0	0	RC3 - C	RC2 - C	RC1 - C	RC0 - C
0D	POR	0	AFH - C	AFL - C	AEH - C	AEL - C	0	IR - C
0E	OE - D	OC - D	0	0	RC3 - D	RC2 - D	RC1 - D	RC0 - D
0F	POR	0	AFH - D	AFL - D	AEH - D	AEL - D	0	IR - D

Figura 6.9: Página 1 de la memoria SRAM interna del DS2450.

Página 2. La página 2 (ver figura 6.10) se utiliza para configurar el umbral alto y el umbral bajo de la alarma. Al encenderse el dispositivo el umbral bajo es 00h y el umbral alto es FFh.

Dirección	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
10	MSBL – A	A	A	A	A	A	A	LSBL – A
11	MSBH – A	A	A	A	A	A	A	LSBH – A
12	MSBL – B	B	B	B	B	B	B	LSBL – B
13	MSBH – B	B	B	B	B	B	B	LSBH – B
14	MSBL – C	C	C	C	C	C	C	LSBL – C
15	MSBH – C	C	C	C	C	C	C	LSBH – C
16	MSBL – D	D	D	D	D	D	D	LSBL – D
17	MSBH – D	D	D	D	D	D	D	LSBH – D

Figura 6.10: Página 2 de la memoria SRAM interna del DS2450.

Página 3. La página 3 (ver figura 6.11) contiene los valores de calibración de fábrica del convertidor A/D. Es importante no modificar estos valores de calibración. Si el convertidor A/D se alimenta con VCC entonces hay que escribir 40h en la localidad 1Ch después de encender el dispositivo.

Dirección	
18	Valor de calibración.
19	Valor de calibración.
1A	Valor de calibración.
1B	Valor de calibración.
1C	Escribir 40hex si la alimentación es externa.
1D	Valor de calibración.
1E	Valor de calibración.
1F	Valor de calibración.

Figura 6.11: Página 3 de la memoria SRAM interna del DS2450.

6.5.3 Comandos del convertidor A/D

Cualquier comunicación con el convertidor A/D 1-Wire ya sea para leer el ROM code, para efectuar una conversión A/D, para leer la memoria SRAM interna, etc. debe realizarse en 3 pasos:

1. Inicialización.
2. Comando de ROM.
3. Comando a ejecutar.

Inicialización. Este paso consiste en la transmisión de un pulso de reset desde el dispositivo maestro al esclavo el cuál responderá con un pulso de presencia indicando que se encuentra listo y en espera del comando de ROM.

Comando de ROM. En este paso el dispositivo maestro envía un comando de 1 byte al esclavo. Los comandos de ROM son:

- READ ROM [33h]. Este comando puede utilizarse únicamente en sistemas single–drop. Cuando el dispositivo maestro envía este comando automáticamente el esclavo responde con el ROM code de 64 bits.

- MATCH ROM [55h]. El dispositivo maestro debe enviar este comando seguido del ROM code del esclavo que recibirá un comando a ejecutar.

- SKIP ROM [CCh]. El maestro utiliza este comando seguido de comandos a ejecutar que no requieren de ROM Code.

Comando a ejecutar. El maestro envía un comando de 1 byte al esclavo. El comando a ejecutar puede ser:

- READ MEMORY [AAH]. Este comando se utiliza para leer la memoria SRAM interna, se envía seguido de dos bytes de dirección (TA1 = (T7:T0), TA2 = (T15:T8)) que indican la localidad de inicio de la lectura. Los bits T15:T5 valen cero. Después de enviar el comando, el maestro recibe bit por bit a partir de la

localidad indicada hasta el final de la página y recibe un CRC de dos bytes del comando, dirección y bytes leídos. La lectura continúa en la siguiente página y al final se recibe el CRC de dos bytes calculado a partir de los datos leídos.

- WRITE MEMORY [55H]. Este comando se utiliza para escribir en las páginas 1, 2 y 3, se envía seguido de dos bytes de dirección (TA1 = (T7:T0), TA2 = (T15:T8)) y el dato a escribir (D7:D0), el maestro recibe un CRC de dos bytes calculado a partir del comando, la dirección y el dato, el DS2450 escribe el dato en la localidad especificada y el maestro recibe una copia del dato para verificar la escritura. Si no se ha llegado al final de la página, el DS2450 incrementa automáticamente la dirección y el maestro debe enviar el siguiente dato a escribir, el maestro recibe un CRC de dos bytes calculado a partir de la dirección y el dato escrito. Después el maestro recibe una copia del byte escrito.

- CONVERT [3CH]. Este comando se utiliza para realizar una conversión A/D en 1 o más canales del DS2450. La conversión A/D lleva de 60µs a 80 µs por bit. Mientras el DS2450 realiza su conversión A/D el maestro puede comunicarse con otros dispositivos del bus 1-Wire. La conversión A/D se realiza en el orden A, B, C, D. El maestro puede leer el resultado de la conversión de un canal antes de que finalicen las conversiones de todos los canales. Después de enviar el comando el maestro envía un byte llamado input select mask (ver figura 6.12) que sirve para seleccionar los canales en los que se desea realizar la conversión A/D. Para seleccionar un canal hay que escribir un 1 en el campo correspondiente. Después el maestro envía un byte llamado read-out control (ver figura 6.13) que sirve para escribir 1's o 0's en los registros de la pagina 0 y así tener la seguridad de que se ha leído correctamente el resultado de la conversión A/D. En aplicaciones en las que el maestro puede esperar a que se realice la conversión A/D de todos los canales no importa como se configure el byte read out control. Una vez enviado el comando, el byte input select mask y el byte read-out control el maestro recibe el CRC de dos bytes del comando, el input select mask y el read-out control y la conversión A/D comienza en 10 µs.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Ø	Ø	Ø	Ø	D	C	B	A

Figura 6.12: Byte input select mask.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Set D	Clear D	Set C	Clear C	Set B	Clear B	Set A	Clear A

Set	**Clear**	**Explicación**
0	0	Sin modificación
0	1	Escribir 0's
1	0	Escribir 1's
1	1	No se utiliza

Figura 6.13: Byte read-out control.

En el protocolo 1-Wire las transferencias de datos y comandos comienzan por el bit menos significativo.

6.6 Lectura del convertidor A/D 1-Wire

6.6.1 Software en el SAB80C166

El software para leer el convertidor A/D (ver figura 6.14) configura los canales B, C y D con una resolución de 8 bits y un rango de entrada de 5.1V. Después recibe 1 byte por el canal serial 0, si el byte recibido es 00h entonces no se ejecuta ninguna operación, pero si es 01h entonces se realiza una conversión A/D en los canales B, C y D mediante el comando Convert y una vez finalizada la conversión se lee el resultado mediante el comando Read Memory y se envía a LabVIEW para su despliegue. Los resultados de la conversión A/D se envían por el puerto serial cada vez que se recibe un byte 01h desde LabVIEW. El tiempo calculado para esta conversión es:

$$t_{conv} = (3)(8)(80\mu s) = 1.92\text{ms}$$

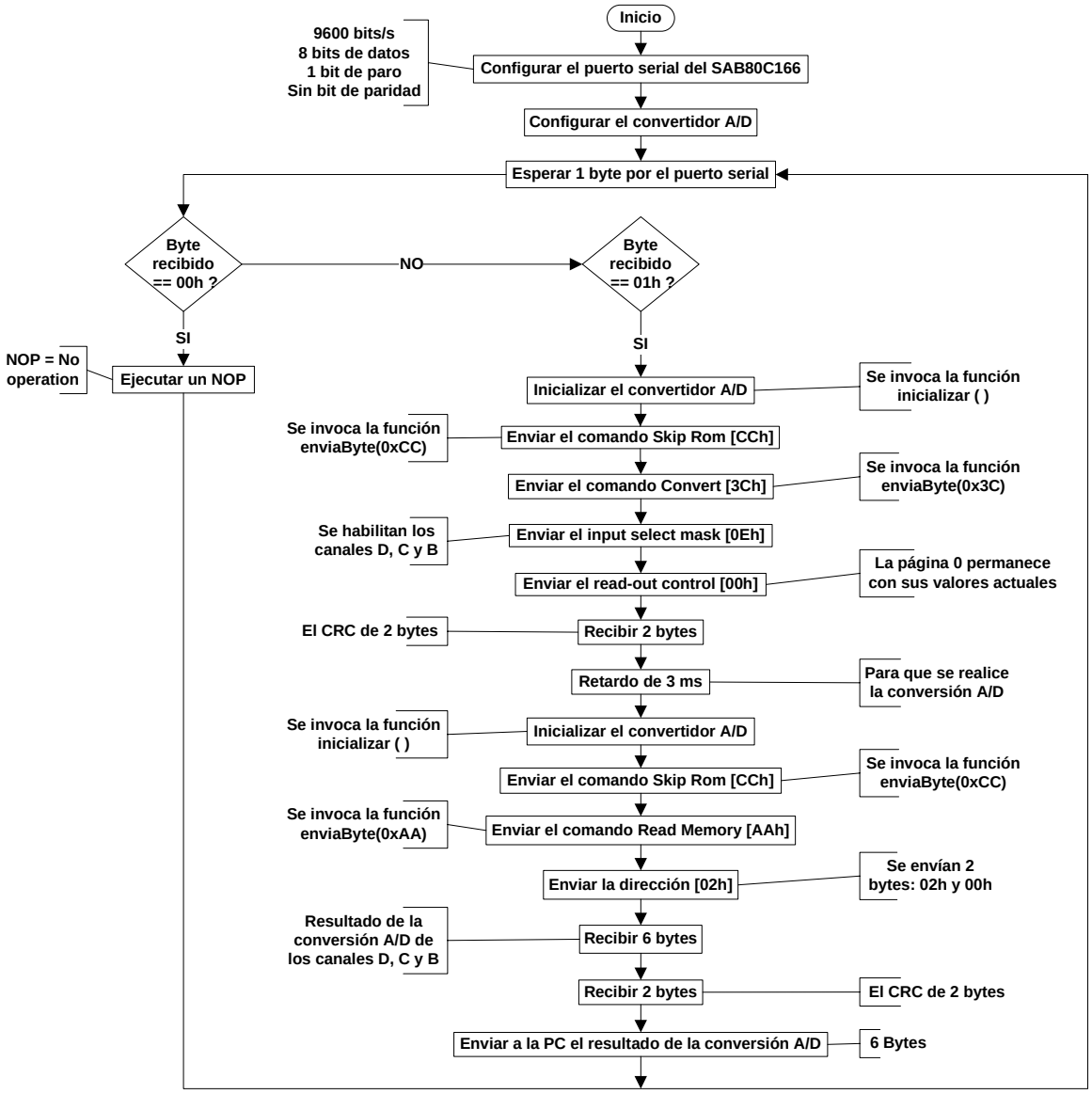


Figura 6.14: Diagrama de flujo del software en el SAB80C166 para leer el convertidor A/D DS2450 (código en el apéndice F.1).

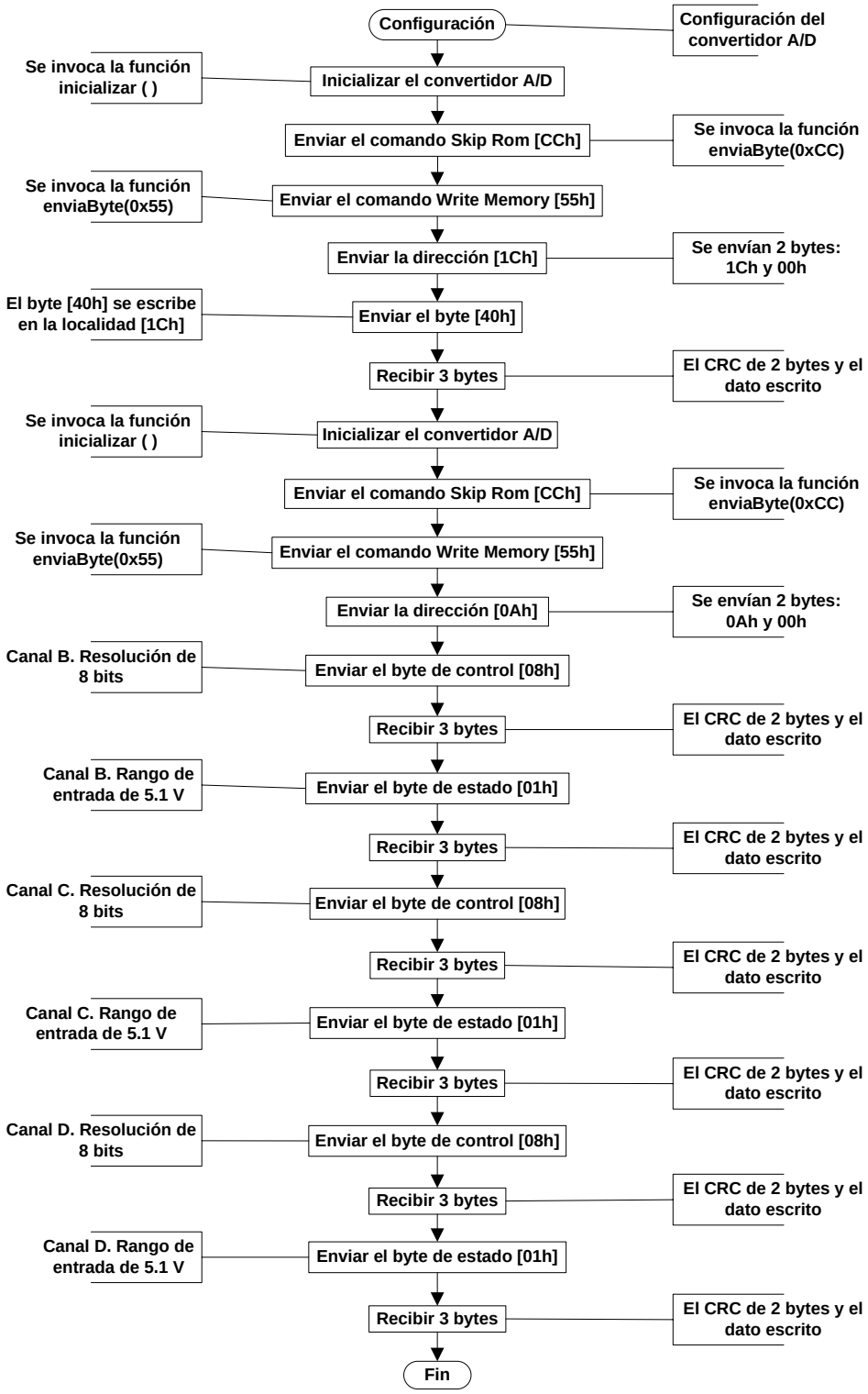


Figura 6.15: Diagrama de flujo de la configuración del convertidor A/D (código en el apéndice F.1).

6.6.2 Interfaz en LabVIEW

La interfaz en LabVIEW (ver figura 6.16) se encuentra cíclicamente monitoreando el estado del interruptor. Si el interruptor está en la posición “ON” entonces se envía el byte 01h por el puerto serial y después se reciben 6 bytes que son el resultado de la conversión A/D el cual se convierte en Amperes y se despliega. Si el interruptor está en la posición “OFF” entonces se envía el byte 00h y se ponen en 0 todos los indicadores. Como la resolución del convertidor A/D es de 8 bits los bytes 1, 3 y 5 recibidos contienen el resultado de la conversión A/D y los bytes 0, 2 y 4 valen cero. Con una resolución de 8 bits y un rango de 5.1 V se tienen 20 mV/bit y por lo tanto los bytes recibidos se multiplican por 0.02 para tener una lectura en Volts. Despejando I de la ecuación (1) y sustituyendo Rsense = 0.1Ω y Gsel = 100 queda:

$$V_{out} = G_{sel} R_{sense} I + 1.5$$

$$I = \frac{V_{out} - 1.5}{G_{sel} R_{sense}}$$

$$I = \frac{V_{out}}{10} - 0.15$$

Esta última ecuación se aplica a cada lectura de voltaje para obtener la corriente eléctrica consumida.

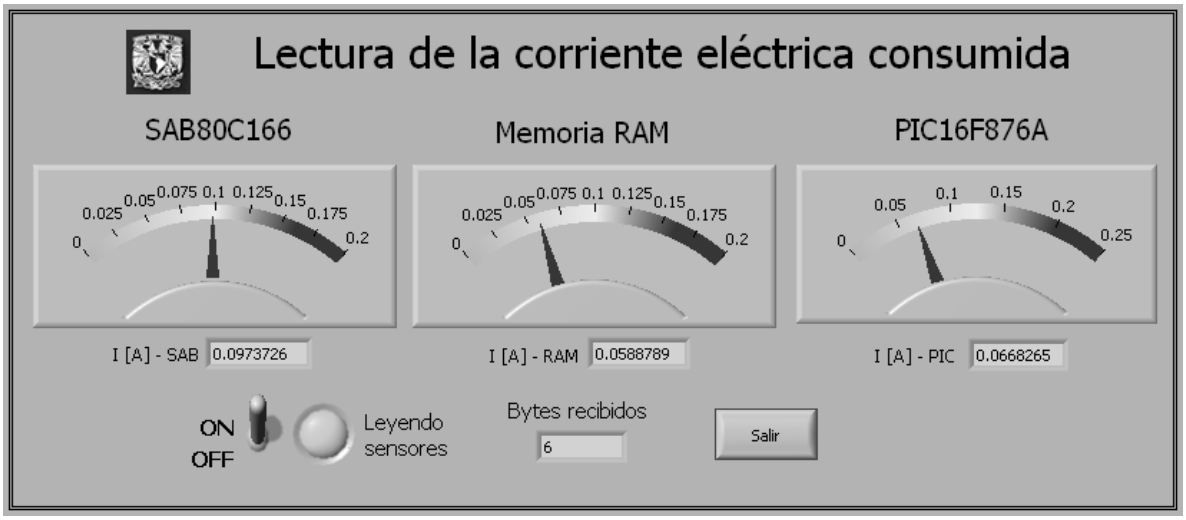


Figura 6.16: Interfaz en LabVIEW para desplegar la corriente eléctrica consumida por el SAB80C166, la memoria RAM y el PIC16F876A.

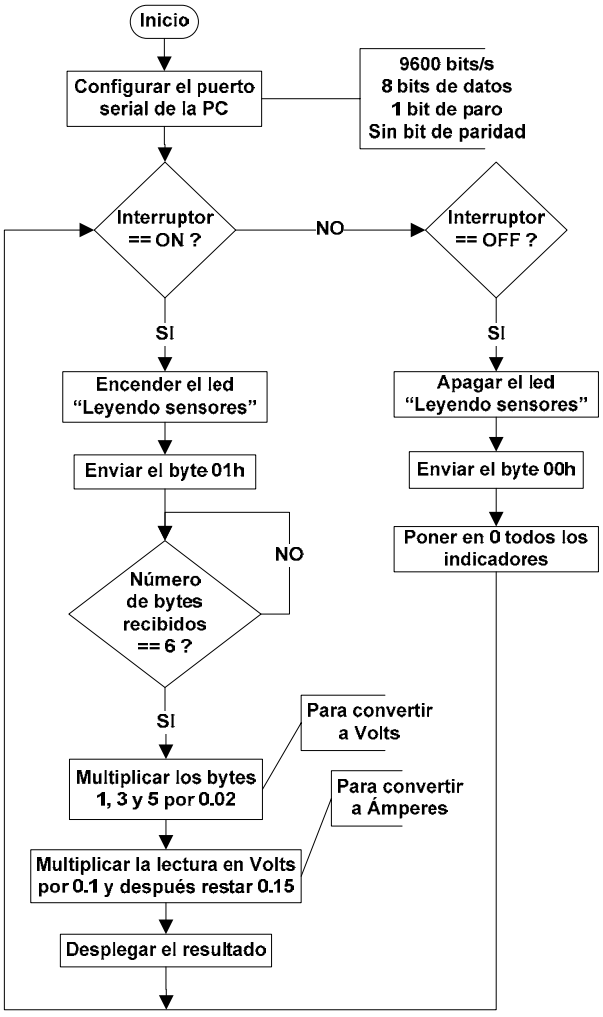


Figura 6.17: Diagrama de flujo del software en LabVIEW para desplegar la corriente eléctrica consumida por el SAB80C166, la memoria RAM y el PIC16F876A (código en el apéndice F.2).

Capítulo 7
Desarrollo de la tarjeta electrónica

7.1 Introducción

El PCB de la computadora de vuelo tiene forma cuadrada de 8.9cm x 8.9cm con componentes en ambos lados. Casi todos los componentes son de montaje superficial. El diseño de la computadora de vuelo se hizo en Protel DXP.

En este capítulo se explica el procedimiento para diseñar la computadora de vuelo.

7.2 Diagrama electrónico de la computadora de vuelo

En el diagrama de la computadora (ver figura 7.1) se muestra el SAB80C166 conectado a 256 KBytes de memoria RAM para programas, el PIC16F876A que sirve para activar el modo de cargar programa en el SAB80C166, los 3 sensores de temperatura 1-Wire, 32 MBytes de memoria flash, 2 multiplexores para conectar la computadora de vuelo hasta con 8 subsistemas y las protecciones contra efecto latch-up en el SAB80C166, el PIC16F876A y la memoria RAM.

La señal de reloj del SAB80C166 se obtiene de un oscilador TTL a 40MHz. Esta señal a 40MHz entra a 2 Flip-Flop's JK configurados en modo "toogle" cuya salida es la señal de reloj del PIC16F876A a 10MHz.

7.3 Diseño del PCB de la computadora de vuelo

El PCB de la computadora de vuelo (ver figuras 7.2 y 7.4) tiene una forma cuadrada de 8.9cm x 8.9cm con componentes en ambos lados. Las pistas tienen un grosor de 8 mil, separación entre pistas de 8 mil y vías de 38 mil x 20 mil. Para rutear el PCB se colocaron los componentes electrónicos de tal forma que las conexiones entre ellos sea lo más directa posible (ver figuras 7.3 y 7.5). Después se conectaron manualmente las señales de más alta frecuencia que son las de 40MHz, 20MHz y 10MHz cuidando que la pista sea lo más recta y corta posible. Una vez hecho esto se utilizó el autoruteador configurado para que el *top layer* sea ruteado horizontalmente y el *bottom layer* verticalmente. Finalmente se agregó un polígono de tierra en el *top layer* y uno en el *bottom layer*.

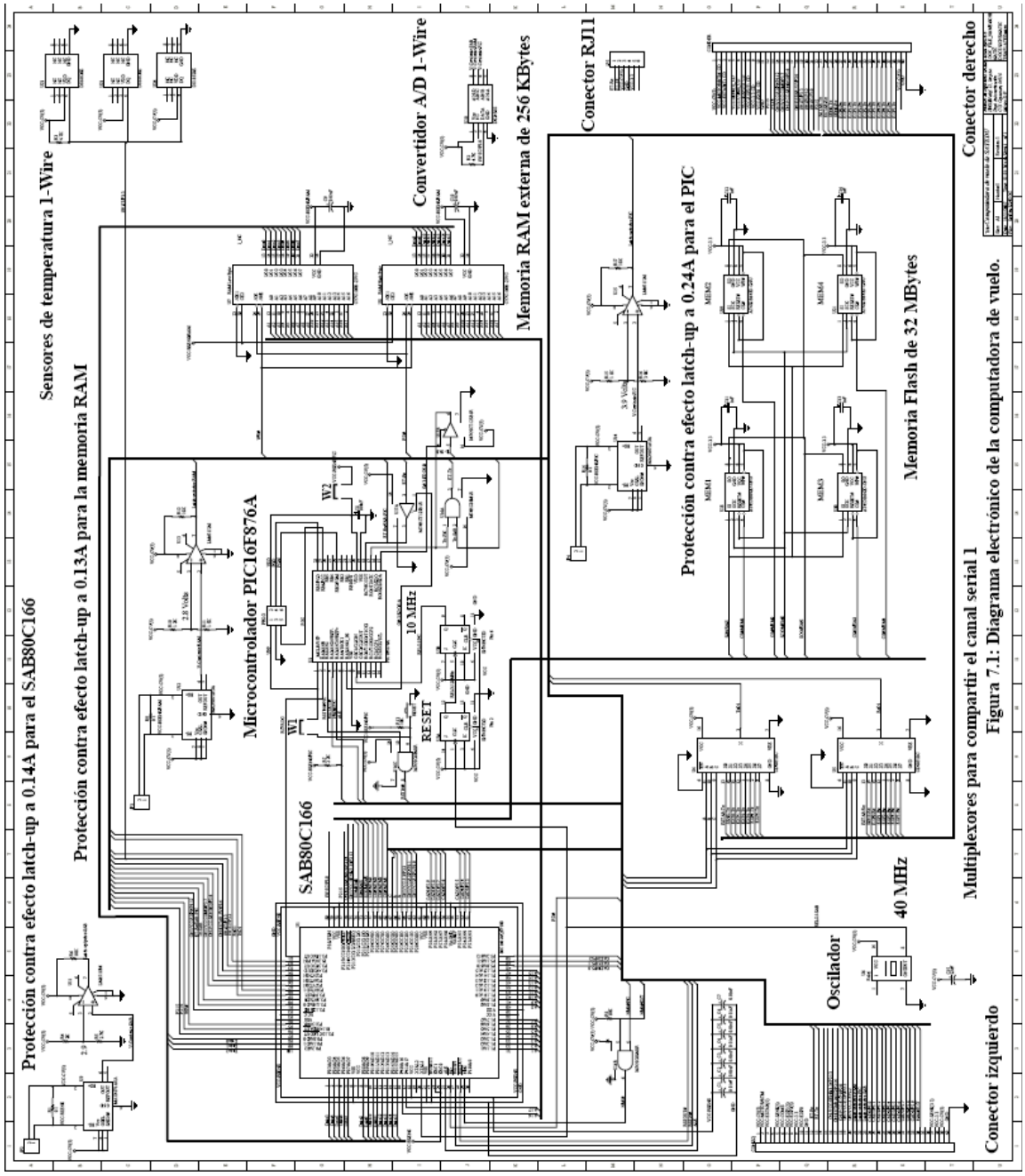


Figura 7.1: Diagrama electrónico de la computadora de vuelo.

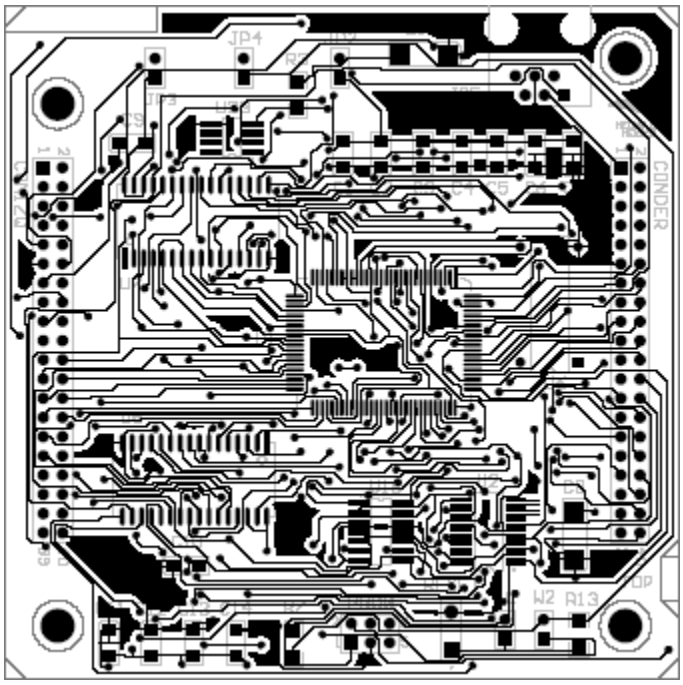


Figura 7.2: Top layer de la computadora de vuelo.

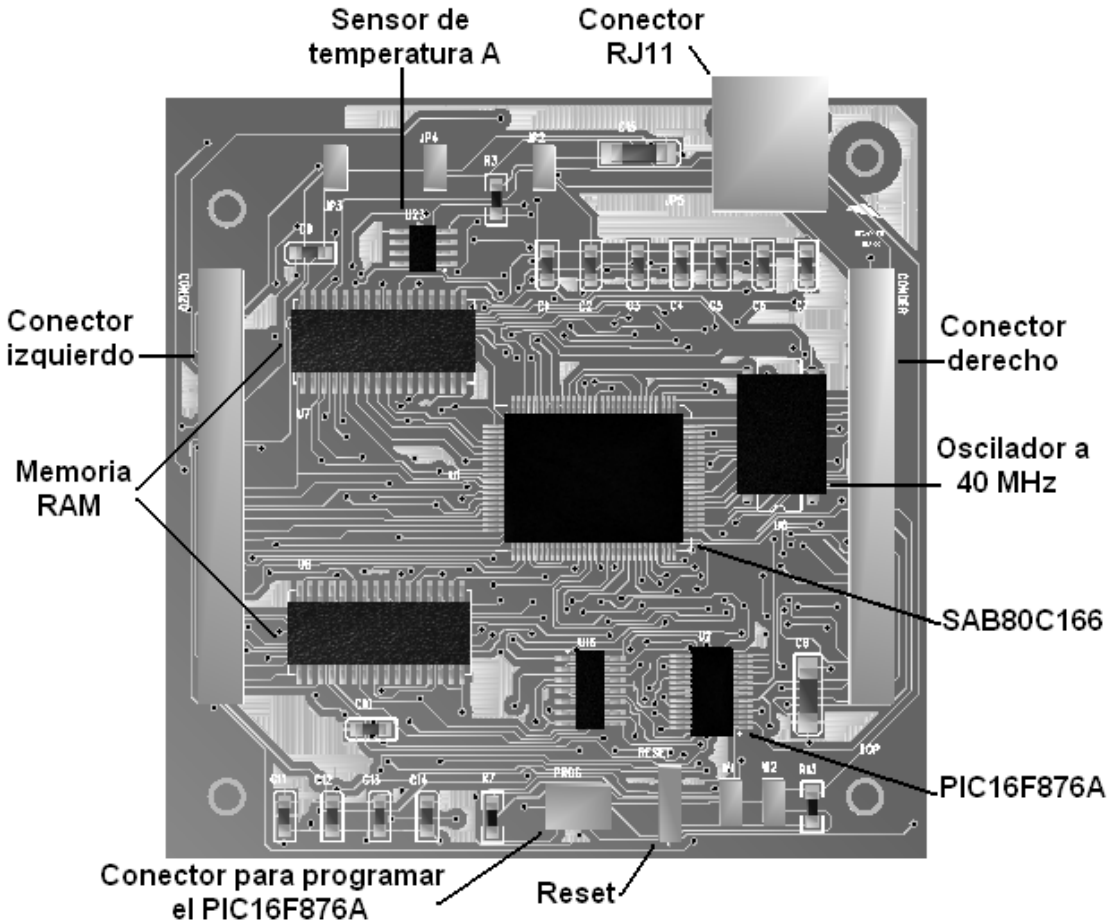


Figura 7.3: Ubicación de los componentes en el top layer de la computadora de vuelo.

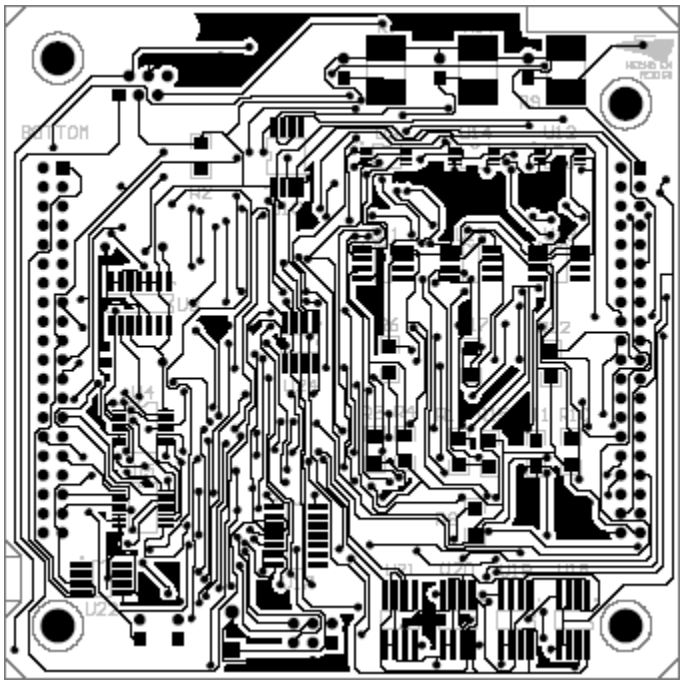


Figura 7.4: Bottom layer de la computadora de vuelo.

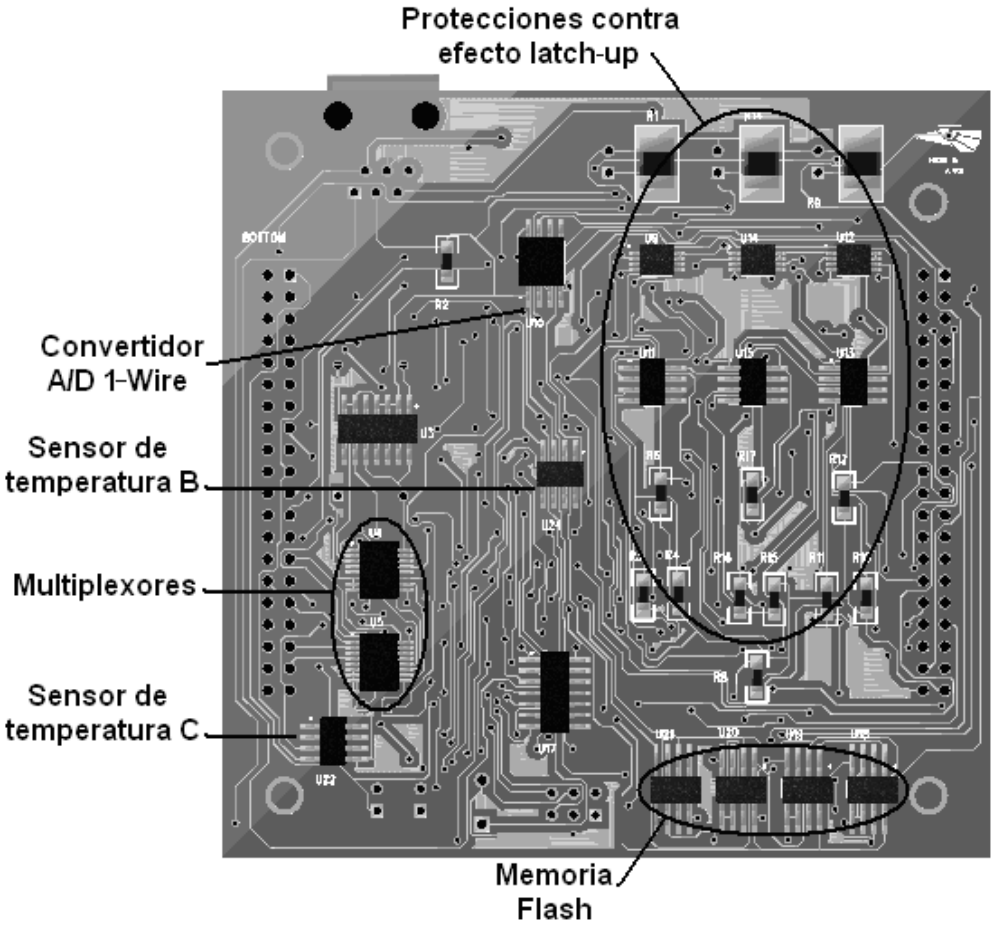


Figura 7.5: Ubicación de los componentes en el bottom layer de la computadora de vuelo.

Capítulo 8
Conclusiones y recomendaciones

8.1 Conclusiones

Del trabajo desarrollado se concluye lo siguiente:

- Se efectuó el diseño de la computadora de vuelo para el sistema de capacitación de recursos humanos en el manejo de satélites. Esta computadora cuenta con un microcontrolador SAB80C166, 256 KBytes de memoria RAM para el almacenamiento de programas, 32 MBytes de memoria flash, 3 sensores de temperatura 1-Wire, protecciones contra efecto latch-up en el SAB80C166, en el PIC16F876A y en la memoria RAM y electrónica para medir la corriente eléctrica consumida por los componentes protegidos contra efecto latch-up.

- Se diseñó el software para el cargado de programas en la computadora de vuelo, para la lectura de los sensores de temperatura 1-Wire, para el manejo de la memoria flash y para la lectura de la corriente eléctrica consumida por los componentes protegidos contra efecto latch-up. A partir de este software se podrá desarrollar la aplicación completa para la computadora de vuelo.

8.2 Recomendaciones

Algunas recomendaciones para la mejora de la computadora de vuelo son las siguientes:

- Implementar *code shadowing* el cual consiste en que una vez encendida la computadora se transfiera automáticamente el software de la computadora de vuelo de la memoria flash a la memoria RAM para su ejecución. Para hacer esto, al encenderse la computadora el PIC16F876A debe poner al SAB80C166 en modo de cargar programa, enviarle el precargador y después el cargador externo. En este caso el cargador externo debe ser capaz de leer el programa en formato Intel HEX almacenado en la memoria flash utilizando el comando *Continuous array read* y transferirlo a la memoria RAM para su ejecución. El software de la computadora de vuelo puede grabarse en la memoria flash utilizando el software para el manejo de la memoria flash desarrollado en esta tesis.

- La computadora de vuelo se puede modernizar sustituyendo el microcontrolador SAB80C166 por un microcontrolador de la familia C166 de Infineon. Dentro de la familia C166 de Infineon se encuentran los microcontroladores C161, C164, C165 y C167. El modo de cargar programa y el set de instrucciones de los microcontroladores de la familia C166 de Infineon es muy similar al SAB80C166.

BIBLIOGRAFÍA

Spasov Peter, **Microcontroller technology**, Prentice Hall, U.S.A., 1993.

Hall Douglas V., **Microprocessors and interfacing programming and hardware**, McGraw – Hill, U.S.A., 1992.

Johnson Gary W., Jennings Richard, **LabVIEW Graphical Programming**, McGraw – Hill, U.S.A., 2001.

Bitter Rick, Taqi Mohiuddin, Nawrocki Matthew, **LabVIEW Advanced Programming Techniques**, CRC Press, U.S.A., 2000.

Gerald L. Ginsberg, **Printed Circuit Design**, McGraw - Hill, U.S.A., 1991.

Torres Portero Manuel, **Diseño e Ingeniería Electrónica asistida con Protel DXP**, Alfaomega, México, 2005.

Morton John, **The PIC microcontroller**, Newnes, U.S.A., 2005.

Nigel Gardner, **An introduction to programming The Microchip PIC in CCS C**, Bluebird Electronics, U.S.A., 2002.

MANUALES

Siemens AG, **Microcomputer Components SAB 80C166/83C166 16 - Bit CMOS Single -Chip Microcontrollers for Embedded Applications**, Siemens AG, 1997.

Siemens AG, **Bootstrap Loader 8xC166**, Siemens AG, 1993.

Infineon Technologies AG, **Instruction Set Manual for the C166 Family of Infineon 16–Bit Single–Chip Microcontrollers**, Infineon Technologies AG, 2001.

Boston Systems Office/Tasking, **80166 Cross–Assembler User's Guide**, Boston Systems Office/Tasking, 1993.

Boston Systems Office/Tasking, **80166 C Cross–Compiler User's Guide**, Boston Systems Office/Tasking, 1993.

Microchip Technology Incorporated, **PIC16F87XA Data Sheet**, Microchip Technology Incorporated, 2003.

Custom Computer Services Incorporated, **C Compiler Reference Manual**, Custom Computer Services Incorporated, 2002.

Cypress Semiconductor Corporation, **CY7C109B, CY7C1009B 128K x 8 Static RAM**, Cypress Semiconductor Corporation, 2002.

Dallas Semiconductor–Maxim, **DS18S20 High–Precision 1–Wire Digital Thermometer**, Dallas Semiconductor–Maxim, 2006.

Dallas Semiconductor–Maxim, **DS2450 1–Wire Quad A/D Converter**, Dallas Semiconductor–Maxim, 2006.

Atmel Corporation, **ATMEL 64–Megabit, 2.7–Volt, Dual–interface DataFlash AT45DB642D**, Atmel Corporation, 2006.
Maxim Integrated Products, **MAXIM Bidirectional, High–Side, Current–Sense Amplifiers with Reference MAX4069–MAX4072**, Maxim Integrated Products, 2003.

National Semiconductor Corporation, **LM6511 180 ns 3V Comparator**, National Semiconductor Corporation, 2003.

Texas Instruments Incorporated, **CD4051B-Q1, CD4052B-Q1, CD4053B-Q1 CMOS analog multiplexers/demultiplexers with logic–level conversion**, Texas Instruments Incorporated, 2004.

Texas Instruments Incorporated, **SN54HC08, SN74HC08 Quadruple 2–input positive–AND gates**, Texas Instruments Incorporated, 2003.

SGS–THOMSON Microelectronics, **M54/74HC125, M54/74HC126 Quad bus buffers (3–state), SGS–THOMSON Microelectronics**, 1993.

Texas Instruments Incorporated, **CD54HC73, CD74HC73, CD74HCT73 Dual J–K Flip–Flop with Reset Negative–Edge Trigger**, Texas Instruments Incorporated, 2003.

National Semiconductor Corporation, **LP38690/LP38692 1A Low dropout CMOS linear regulators stable with ceramic output capacitors**, National Semiconductor Corporation, 2005.

Maxim Integrated Products, **MAXIM +5V–Powered, Multichannel RS–232 Drivers/Receivers MAX220–MAX249**, Maxim Integrated Products, 2006.

REFERENCIAS WEB

Programador de PIC's Multi PIC Programmer http://feng3.cool.ne.jp/en/pg5v2.html

Software IC–Prog http://www.ic-prog.com/

PCB track width calculator http://www.desmith.net/NMdS/Electronics/TraceWidth.html

Apéndice A

A.1 Señales en el conector izquierdo de la computadora de vuelo

Pin	Nombre	Descripción
1	VCC-CV(5)	Alimentación de la CV
2	VCC-MOTOR/BTM	Alimentación del motor de C.D. y las bobinas de torque magnético
3	VCC-ESTAB(5)	Alimentación de la tarjeta de estabilización
4	Pin libre	Pin libre
5	VCC-SENS(5)	Alimentación de la tarjeta de sensores
6	VCC-COMS	Alimentación de la tarjeta de comunicaciones
7	VCC-EXP1(5)	Alimentación de la expansión 1
8	VCC-EXP2(5)	Alimentación de la expansión 2
9	VCC-3.3	Alimentación de la memoria flash
10	VCC-EXP0	Alimentación de la expansión 0
11	GND	Tierra
12	ET-Rx	Receptor de la computadora de vuelo
13	ET-Tx	Transmisor de la computadora de vuelo
14	Pin libre	Pin libre
15	Pin libre	Pin libre
16	ON-VCC-SENS/P2.13	Apagado de la tarjeta de sensores
17	ON-ESTAB-MOT/P2.14	Apagado de la tarjeta de estabilización
18	ON-VCC-EXP1/P2.0	Apagado de la expansión 1
19	ON-VCC-EXP2/P2.1	Apagado de la expansión 2
20	ON-VCC-3.3/P2.2	Apagado de la memoria flash
21	Latch-up/pulso-RAM	Efecto latch-up en la memoria RAM
22	Latch-up/pulso-PIC	Efecto latch-up en el PIC16F876A
23	Latch-up/pulso-SAB	Efecto latch-up en el SAB80C166
24	CAD0/P5.0	Canal 0 del convertidor A/D del SAB80C166
25	CAD1/P5.1	Canal 1 del convertidor A/D del SAB80C166
26	CAD2/P5.2	Canal 2 del convertidor A/D del SAB80C166
27	CAD3/P5.3	Canal 3 del convertidor A/D del SAB80C166
28	CAD4/P5.4	Canal 4 del convertidor A/D del SAB80C166
29	CAD5/P5.5	Canal 5 del convertidor A/D del SAB80C166
30	CAD6/P5.6	Canal 6 del convertidor A/D del SAB80C166
31	CAD7/P5.7	Canal 7 del convertidor A/D del SAB80C166
32	CAD8/P5.8	Canal 8 del convertidor A/D del SAB80C166
33	CAD9/P5.9	Canal 9 del convertidor A/D del SAB80C166
34	NMI#/EXT	Pin de interrupción no mascarable del SAB80C166
35	Pin libre	Pin libre
36	Pin libre	Pin libre
37	Pin libre	Pin libre
38	VCC-3.3	Alimentación de la memoria flash
39	VCC-CV(5)	Alimentación de la CV
40	GND	Tierra

A.2 Señales en el conector derecho de la computadora de vuelo

Pin	Nombre	Descripción
1	VCC-CV(5)	Alimentación de la CV
2	VCC-MOTOR/BTM	Alimentación del motor de C.D. y las bobinas de torque magnético
3	VCC-ESTAB(5)	Alimentación de la tarjeta de estabilización

4	Pin libre	Pin libre
5	VCC-SENS(5)	Alimentación de la tarjeta de sensores
6	VCC-COMS	Alimentación de la tarjeta de comunicaciones
7	VCC-EXP1(5)	Alimentación de la expansión 1
8	VCC-EXP2(5)	Alimentación de la expansión 2
9	VCC-3.3	Alimentación de la memoria flash
10	VCC-EXP0	Alimentación de la expansión 0
11	GND	Tierra
12	P3.15	P3.15 del SAB80C166
13	P3.14	P3.14 del SAB80C166
14	ON-VCC-EXP0/P3.12	Apagado de la expansión 0
15	ON-VCC-COMS/P3.7	Apagado de la tarjeta de comunicaciones
16	Pin libre	Pin libre
17	Pin libre	Pin libre
18	ON-ESTAB-5V/P3.4	Apagado de la tarjeta de estabilización
19	1W-ST/P3.3	Sensores 1-Wire de temperatura
20	1W-SV/P3.2	Convertidores A/D 1-Wire
21	1W-SC/P3.0	Lecturas de corriente eléctrica consumida a través de convertidores A/D 1-Wire
22	P2.15	P2.15 del SAB80C166
23	ESTAB-Tx	Transmisor hacia la tarjeta de estabilización
24	ESTAB-Rx	Receptor para la tarjeta de estabilización
25	SENS-Tx	Transmisor hacia la tarjeta de sensores
26	SENS-Rx	Receptor para la tarjeta de sensores
27	EXP0-Rx	Receptor para la expansión 0
28	EXP0-Tx	Transmisor hacia la expansión 0
29	EXP1-Rx	Receptor para la expansión 1
30	EXP1-Tx	Transmisor hacia la expansión 1
31	EXP2-Rx	Receptor para la expansión 2
32	EXP2-Tx	Transmisor hacia la expansión 2
33	EXP3-Rx	Receptor para la expansión 3
34	EXP3-Tx	Transmisor hacia la expansión 3
35	EXP4-Rx	Receptor para la expansión 4
36	EXP4-Tx	Transmisor hacia la expansión 4
37	EXP5-Rx	Receptor para la expansión 5
38	EXP5-Tx	Transmisor hacia la expansión 5
39	VCC-CV(5)	Alimentación de la CV
40	GND	Tierra

A.3 Señales en los pines del SAB80C166

Puerto	Pin	Señal	Descripción
0	**0 a 15**	Bus de datos	Bus de datos de la memoria RAM
1	**0 a 15**	Bus de direcciones	Bus de direcciones de la memoria RAM
2	**0**	ON-VCC-EXP1	Apagado de la tarjeta de expansión 1
2	**1**	ON-VCC-EXP2	Apagado de la tarjeta de expansión 2
2	**2**	ON-VCC-3.3	Apagado de la memoria flash
2	**3**	A	Señal de control de los multiplexores
2	**4**	B	Señal de control de los multiplexores
2	**5**	C	Señal de control de los multiplexores
2	**6**	CS#/MEM1	CS# de la memoria flash 1

2	**7**	CS#/MEM2	CS# de la memoria flash 2
2	**8**	CS#/MEM3	CS# de la memoria flash 3
2	**9**	CS#/MEM4	CS# de la memoria flash 4
2	**10**	SCK/MEMS	Señal de reloj de las memorias flash
2	**11**	SI/MEMS	Señal de entrada de las memorias flash
2	**12**	SO/MEMS	Señal de salida de las memorias flash
2	**13**	ON-VCC-SENS	Apagado de la tarjeta de sensores
2	**14**	ON-ESTAB-MOT	Apagado del motor de la tarjeta de estabilización
2	**15**	Pin libre	Pin libre
3	**0**	1W-SC	Bus 1-Wire de los convertidores A/D que miden la corriente eléctrica consumida
3	**1**	RCPROG	Pin de señalización para el PIC16F876A
3	**2**	1W-SV	Bus 1-Wire para los convertidores A/D
3	**3**	1W-ST	Bus 1-Wire para los sensores de temperatura
3	**4**	ON-ESTAB-5V	Apagado de la tarjeta de estabilización
3	**5**	Pin libre	Pin libre
3	**6**	Pin libre	Pin libre
3	**7**	ON-VCC-COMS	Apagado de la tarjeta de comunicaciones
3	**8**	TxD1	Transmisor de red interna
3	**9**	RxD1	Receptor de red interna
3	**10**	Tx-SAB	Transmisor hacia el software de PC
3	**11**	ET-Rx/SAB-PIC	Receptor desde el software de PC
3	**12**	ON-VCC-EXP0	Apagado de la expansión 0
3	**13**	WR#	Señal Write enable de la memoria RAM
3	**14**	Pin libre	Pin libre
3	**15**	Pin libre	Pin libre
4	**0**	A16	Bus de direcciones extendido
4	**1**	A17	Bus de direcciones extendido
5	**0**	CAD0	Canal 0 del convertidor A/D
5	**1**	CAD1	Canal 1 del convertidor A/D
5	**2**	CAD2	Canal 2 del convertidor A/D
5	**3**	CAD3	Canal 3 del convertidor A/D
5	**4**	CAD4	Canal 4 del convertidor A/D
5	**5**	CAD5	Canal 5 del convertidor A/D
5	**6**	CAD6	Canal 6 del convertidor A/D
5	**7**	CAD7	Canal 7 del convertidor A/D
5	**8**	CAD8	Canal 8 del convertidor A/D
5	**9**	CAD9	Canal 9 del convertidor A/D

Apéndice B. Ejemplo del software en el PIC16F876A

```
#include <16f876a.h>
#fuses HS,NOWDT,NOPUT,NOPROTECT,NODEBUG,NOBROWNOUT,NOLVP,NOCPD,NOWRT
#use delay(clock=10000000) // 10 MHz
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7,parity=n,bits=8,stream=normal)
#use rs232(baud=9600,xmit=PIN_C7,rcv=PIN_C6,parity=n,bits=8,stream=encendido)
#define rcprog pin_a0
#define rstin pin_a1
#define rstout pin_a2
#define nmi pin_a3
#define ale pin_a4
#define gA pin_a5
#define gB pin_c4

int j;
int cmd[13];
long num=0;
int tiempo;

int chksum (void) {
int x=0;
for (j=0;j<12;j++)
x=x+cmd[j]; // Se suman el numero de comando y los parametros
x=255-x; // Calcula el
x++; // complemento a2
if (x == cmd[12])
return 1; // Si el checkcum calculado es igual al recibido
else
return 0; // Si el checkcum calculado es diferente al recibido
}

void modoCargar (void) {
output_low(rstin);
output_high(ale);
delay_us(100);
output_high(rstin);
output_float(ale);
output_low(nmi);
delay_us(100);
output_high(nmi);
}

void cmd0 (void) {
modoCargar( ); // El comando 0 pone al SAB80C166 en modo de cargar programa
while(!input(rcprog)); // Espera a que se termine de recibir el programa
}

void cmd1 (void) {
output_low(rstin);
delay_us(100); // El comando 1 le da un reset por hardware al SAB80C166
output_high(rstin);
while(!input(rstout)); // Espera a que termine la secuencia de reset
}

void cmd2 (void) {
output_low(nmi);
delay_us(100); // El comando 2 activa la interrupcion NMI del SAB80C166
output_high(nmi);
while(!input(rcprog)); // Espera a que se termine de ejecutar la rutina de interrupción NMI
}

#INT_RDA
recibCmd ( ) {
for (j=0;j<13;j++)
cmd[j]=fgetc(normal); // Recibe el comando de 13 bytes
if (chksum( )==1) { // Verifica el checksum
switch (cmd[0]) {
case 0:
```

```
cmd0( ); // Invoca el comando 0
break;
case 1:
cmd1( ); // Invoca el comando 1
break;
case 2:
cmd2( ); // Invoca el comando 2
break;
}
}
else
fputc('E',normal); // Si el checksum es incorrecto envia el byte de error 'E' al software de PC
}

#INT_TIMER1
rutina ( ) { // Retardo de 1 minuto utilizando el timer 1
num++;
if (num==299) {
num=0;
tiempo=1;
}
}

int espRecib (void) {
setup_timer_1(T1_INTERNAL | T1_DIV_BY_8); // Configura el timer 1 con
set_timer0(0);                                    // desbordamiento cada 0.2s
enable_interrupts(GLOBAL);
enable_interrupts(INT_TIMER1); // Habilita la interrupcion por desbordamiento del timer 1
tiempo=0;
while( (kbhit( )==0) || (tiempo==0) ); // Esperar 1 byte por 1 minuto
if (kbhit( ))
return 1;
else
return 0;
}

main ( ) {
int i;
int prog32[32]={0xE6,0x0B,0x40,0xFA,0xE6,0x0A,0x20,0xFA,0xE6,0x86,0xC0,
0x62,0xE6,0xE1,0xFF,0xEF,0xE6,0xE3,0xFF,0xF5,0xE6,0xE0,0xC0,0x03,0xE6,
0xE2,0x80,0x25,0x1F,0xE2,0x0D,0xFF}; // Programa de 32 bytes para el SAB80C166
int msg[5]={'E','N','C','E','N'};
int resp[4];
output_high(gA); // Abierto
output_low(gB); // Cerrado
output_high(rstin); // Alto
output_float(ale); // Entrada
output_high(nmi); // Alto
delay_ms(100); // El SAB80C166 se enciende
modoCargar( ); // El SAB80C166 se pone en modo cargar nuevo programa
fputc(0x00,encendido); // Envia el byte 0x00
fgetc(encendido); // Recibe el byte 0x55
for (i=0;i<32;i++)
fputc(prog32[i],encendido); // Envia el programa de 32 bytes al SAB80C166
while(!input(rcprog)); // Espera a que se termine de ejecutar el programa de 32 bytes
output_float(pin_c7); // Entrada
output_low(gA); // Cerrado
output_high(gB); // Abierto
do {
for (i=0;i<5;i++)
fputc(msg[i],normal); // Envia un mensaje al software de PC cada 1min
} while (!espRecib( )); // hasta que reciba un byte de respuesta
for (i=0;i<4;i++)
resp[i]=fgetc(normal); // Recibe el mensaje RECI del software de PC
enable_interrupts(GLOBAL);
enable_interrupts(INT_RDA); // Habilita la interrupcion por recepcion en el puerto serial
while(1); // Espera comandos
}
```

Apéndice C. Cargado de software en la computadora de vuelo

C.1 Precargador

```
$LISTALL
NAME precar
SSKDEF 03                       ;Pila de 32 words
REGDEF R0
SEC1 SECTION CODE AT 0FA40h     ;Direccion de inicio de programa: 0FA40h
MAIN PROC TASK INTNO=0
MOV R0,#0FA60h          ;Direccion donde comienza el cargador externo

MOV S0TBUF,#00001h              ;Envia el byte 01h para comenzar
ETRANS: JNB S0TIC.7,ETRANS      ;a recibir el cargador externo

ESP: JNB S0RIC.7,ESP    ;Espera un byte por el canal serial 0
MOVB [R0],S0RBUF        ;El byte recibido lo escribe en el cargador externo
BCLR S0RIC.7            ;Pone en 0 el bit S0RIR
CMPI1 R0,#0FC13h  ;Si ya se recibio todo el cargador externo
JMPR CC_NZ,ESP          ;brinca a la siguiente instruccion
JMPA CAREXT             ;Brinco al cargador externo de 436 bytes
RETV
MAIN ENDP
SEC1 ENDS
SEC2 SECTION CODE AT 0FA60h     ;Cargador externo
CAREXT:
SEC2 ENDS
END
```

C.2 Cargador externo

```
$LISTALL
NAME carext
REGDEF R0       ;Numero de bytes
REGDEF R1       ;DireccionH, Direccion
REGDEF R2       ;DireccionL
REGDEF R3       ;Record type
REGDEF R4       ;Checksum
REGDEF R5       ;ByteH, Extended segment address
REGDEF R6       ;ByteL
REGDEF R7       ;R7
REGDEF R8       ;Checksum calculado
REGDEF R9       ;Segmento

SEC1 SECTION CODE AT 0FA60h     ;Direccion de inicio del cargador externo
MAIN PROC TASK INTNO=0
MOV SYSCON,#062C0h      ;Habilitar la memoria externa, Stack de 32 words
JMPS SEG SEC1,siglin    ;Dummy jump
siglin: far
MOV DPP0,#00000h ;Data Page Pointer's en el segmento 0
MOV DPP1,#00001h
MOV DPP2,#00002h
MOV DPP3,#00003h
BSET P3.13      ;pin WR# en alto
BSET DP3.13     ;pin WR# como salida
MOV SYSCON,#02C0h       ;Stack de 256 words, memoria externa habilitada
MOV STKUN ,#0FB20h      ;Configura el Stack Underflow
MOV STKOV ,#0FA20h      ;Configura el Stack Overflow
BCLR S0TIC.7            ;Pone en 0 el bit S0TIR

OTRO: MOV S0TBUF,#00001h        ;Envia el byte 01h para pedir un renglon
ESPUN: JNB S0TIC.7,ESPUN  ;Espera a que se envie el byte 01h
BCLR S0TIC.7                    ;Pone en 0 el bit S0TIR
MOV R8,#00000                   ;Escribe 0 en el Checksum calculado

ESPNB: JNB S0RIC.7,ESPNB  ;Recibe el Numero de bytes
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
```

```
MOV R0,S0RBUF                ;Numero de bytes se guarda en R0
ADD R8,R0                    ;La suma queda en R8

ESPDH: JNB S0RIC.7,ESPDH  ;Recibe DireccionH
BCLR S0RIC.7                 ;Pone en 0 el bit S0RIR
MOV R1,S0RBUF                ;DireccionH se guarda en R1
ADD R8,R1                    ;La suma queda en R8

ESPDL: JNB S0RIC.7,ESPDL   ;Recibe DireccionL
BCLR S0RIC.7                 ;Pone en 0 el bit S0RIR
MOV R2,S0RBUF                ;DireccionL se guarda en R2
ADD R8,R2                    ;La suma queda en R8

ESPRT: JNB S0RIC.7,ESPRT   ;Recibe el Record type
BCLR S0RIC.7                 ;Pone en 0 el bit S0RIR
MOV R3,S0RBUF                ;El Record type se guarda en R3
ADD R8,R3                    ;La suma queda en R8

SHL R1,#00008h
OR R1,R2                     ;En R1 se guarda la Direccion de 2 bytes

CMP R3,#00000h               ;Record type de datos ?
JMPR CC_NZ,SIG1          ;Si no es, entonces brinca a SIG1
ADD R1,R5                    ;En R1 se guarda la Direccion
JMPR cc_NC,BSEG1         ;Si no hay carry brinca a BSEG1
ADD R9,#00001h               ;Suma el carry
BSEG1: CMP R9,#00000h        ;Segmento 0 ?
JMPR CC_NZ,BSEG2             ;Si no es, entonces brinca a BSEG2
MOV DPP0,#00000h         ;Data Page Pointer's
MOV DPP1,#00001h         ;en el segmento 0
MOV DPP2,#00002h
MOV DPP3,#00003h
JMPA DAT                     ;Brinca a DAT
BSEG2: CMP R9,#00001h        ;Segmento 1 ?
JMPR CC_NZ,BSEG3             ;Si no es, entonces brinca a BSEG3
MOV DPP0,#00004h         ;Data Page Pointer's
MOV DPP1,#00005h         ;en el segmento 1
MOV DPP2,#00006h
MOV DPP3,#00007h
JMPA DAT                     ;Brinca a DAT
BSEG3: CMP R9,#00002h        ;Segmento 2 ?
JMPR CC_NZ,BSEG4             ;Si no es, entonces brinca a BSEG4
MOV DPP0,#00008h         ;Data Page Pointer's
MOV DPP1,#00009h         ;en el segmento 2
MOV DPP2,#0000Ah
MOV DPP3,#0000Bh
JMPA DAT                     ;Brinca a DAT
BSEG4:                       ;Segmento 3
MOV DPP0,#0000Ch         ;Data Page Pointer's
MOV DPP1,#0000Dh             ;en el segmento 3
MOV DPP2,#0000Eh
MOV DPP3,#0000Fh
DAT: JNB S0RIC.7,DAT         ;Recibe un byte
MOVB [R1],S0RBUF             ;El byte recibido lo escribe en la localidad correspondiente
ADD R8,S0RBUF                ;La suma de bytes se guarda en R8
BCLR S0RIC.7                 ;Pone en 0 el bit S0RIR
ADD R1,#00001h               ;Incrementa la Direccion
SUB R0,#00001h               ;Decrementa el Numero de bytes
CMP R0,#00000h               ;Recibio todos los bytes ?
JMPR CC_NZ,DAT               ;Si no, entonces continua recibiendo
CHKS1: JNB S0RIC.7,CHKS1  ;Recibe el Checksum
BCLR S0RIC.7                 ;Pone en 0 el bit S0RIR
MOV R4,S0RBUF                ;El checksum se guarda en R4
SUB R8,#00100h               ;En R8 se guarda el Checksum calculado
AND R8,#000FFh               ;Toma el byte menos significativo
CMP R4,R9                    ;Compara el checksum recibido con el calculado
JMPR CC_NZ,ERR               ;Si son diferentes, entonces envia el byte de error
JMPA OTRO                    ;Brinco para recibir otro renglon
```

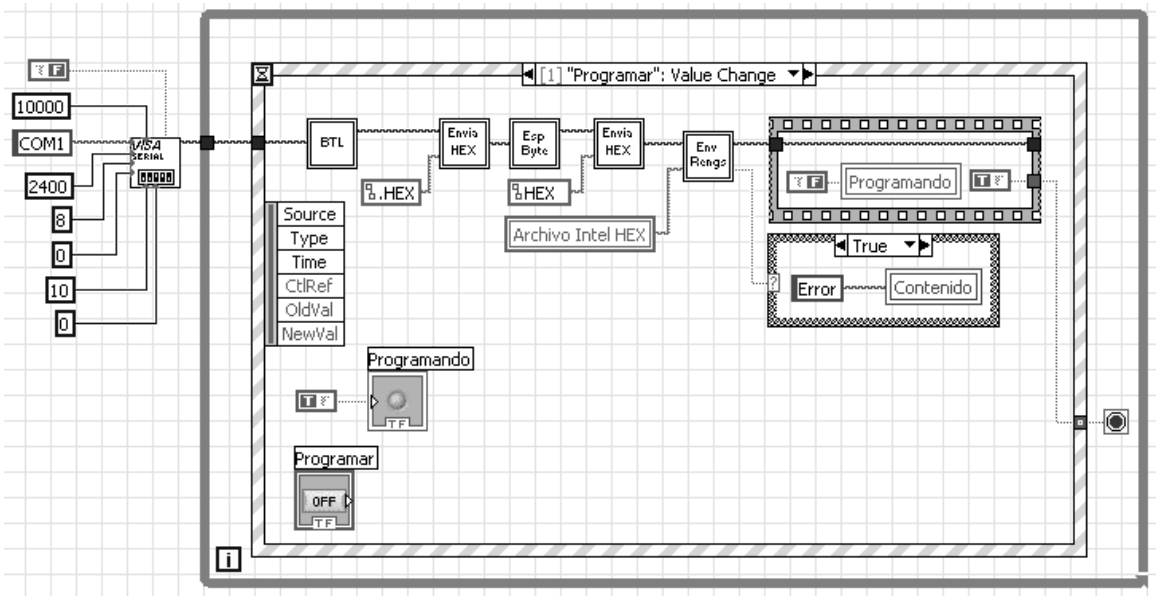
```
SIG1: CMP R3,#00001h            ;Record type de final de archivo ?
JMPR CC_NZ,SIG2          ;Si no es, entonces brinca a SIG2
FINCHK: JNB S0RIC.7,FINCHK      ;Recibe el Checksum
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R4,S0RBUF                   ;El Checksum se guarda en R4
MOV S0TBUF,#00000h              ;Envia el byte 00h para finalizar el cargado de software
ESPCER: JNB S0TIC.7,ESPCER      ;Espera a que se envie el byte 00h
BCLR S0TIC.7                    ;Pone en 0 el bit S0TIR
SRST                            ;Reset por software

SIG2: CMP R3,#00002h            ;Record type 2 ?
JMPR CC_NZ,SIG3          ;Si no es, entonces brinca a SIG3
ESPBH: JNB S0RIC.7,ESPBH ;Recibe el ByteH
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R5,S0RBUF                   ;El byteH se guarda en R5
ADD R8,R5                       ;La suma queda en R8
ESPBL: JNB S0RIC.7,ESPBL ;Recibe el ByteL
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R6,S0RBUF                   ;El ByteL se guarda en R6
ADD R8,R6                       ;La suma queda en R8
CHKS2: JNB S0RIC.7,CHKS2 ;Recibe el Checksum
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R4,S0RBUF                   ;El Checksum se guarda en R4
SUB R8,#00100h                  ;En R8 se guarda el Checksum calculado
AND R8,#000FFh                  ;Toma el byte menos significativo
CMP R4,R8                       ;Compara el Checksum recibido con el calculado
JMPR CC_NZ,ERR                  ;Si son diferentes, entonces envia el byte de error
SHL R5,#00008h
OR R5,R6                        ;En R5 se guarda la Extended segment address
MOV R9,R5                       ;En R9 se copea R5
SHL R5,#00004h                  ;Corrimiento de 4 bits a la izquierda
SHR R9,#0000Ch                  ;Corrimiento de 12 bits a la derecha. En R9 se guarda el segmento
JMPA OTRO                       ;Brinco para recibir otro renglon

SIG3: CMP R3,#00003h            ;Record type 3 ?
JMPR CC_NZ,ERR                  ;Si no es, entonces brinca a ERR
ESPPB: JNB S0RIC.7,ESPPB ;Recibe el 1er byte
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R7,S0RBUF                   ;El 1er byte se guarda en R7
ADD R8,R7                       ;La suma se guarda en R8
ESPSB: JNB S0RIC.7,ESPSB ;Recibe el 2do byte
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R7,S0RBUF                   ;El 2do byte se guarda en R7
ADD R8,R7                       ;La suma se guarda en R8
ESPTB: JNB S0RIC.7,ESPTB ;Recibe el 3er byte
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R7,S0RBUF                   ;El 3er byte se guarda en R7
ADD R8,R7                       ;La suma se guarda en R8
ESPCB: JNB S0RIC.7,ESPCB ;Recibe el 4to byte
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R7,S0RBUF                   ;El 4to byte se guarda en R7
ADD R8,R7                       ;La suma se guarda en R8
CHKS3: JNB S0RIC.7,CHKS3 ;Recibe el Checksum
BCLR S0RIC.7                    ;Pone en 0 el bit S0RIR
MOV R4,S0RBUF                   ;El Checksum se guarda en R4
SUB R8,#00100h                  ;En R8 se guarda el Checksum calculado
AND R8,#000FFh                  ;Toma el byte menos significativo
CMP R4,R8                       ;Compara el checksum recibido con el calculado
JMPR CC_NZ,ERR                  ;Si son diferentes, entonces envia el byte de error
JMPA OTRO                       ;Brinco para recibir otro renglon

ERR: MOV S0TBUF,#00002h  ;Envia el byte 02h para indicar que hubo un error
ESPERR: JNB S0TIC.7,ESPERR      ;Espera a que se envie el byte 02h
BCLR S0TIC.7                    ;Pone en 0 el bit S0TIR
FIN: JMPA FIN
RETV
MAIN ENDP
SEC1 ENDS
END
```

C.3 Software en el PIC16F876A para poner al SAB80C166 en modo de cargar programa en el encendido

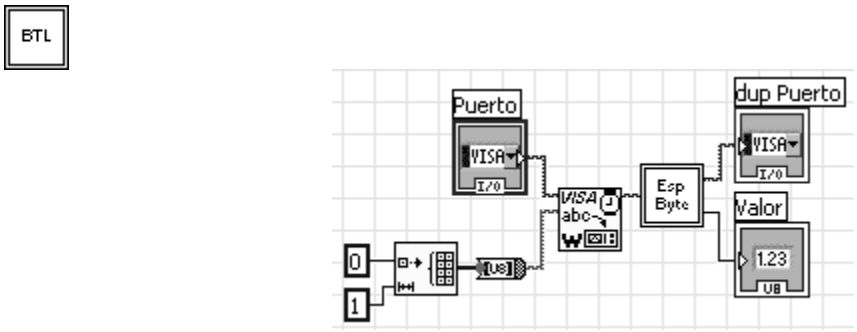
```
#include <16f876a.h>
#fuses HS,NOWDT,NOPUT,NOPROTECT,NODEBUG,NOBROWNOUT,NOLVP,NOCPD,NOWRT
#use delay(clock=10000000) // 10 MHz

#define rcprog pin_a0
#define rstin pin_a1
#define rstout pin_a2
#define nmi pin_a3
#define ale pin_a4
#define gA pin_a5
#define gB pin_c4

void modoCargar (void) {
output_low(rstin);
output_high(ale);
delay_us(100);
output_high(rstin);
output_float(ale);
output_low(nmi);
delay_us(100);
output_high(nmi);
}

main ( ) {
output_low(gA); // Cerrado
output_high(gB); // Abierto
output_high(rstin); // Alto
output_float(ale); // Entrada
output_high(nmi); // Alto
delay_ms(100); // El SAB80C166 se enciende
modoCargar( ); // El SAB80C166 se pone en modo de cargar programa
}
```

C.4 Software en LabVIEW para cargar un programa en el SAB80C166

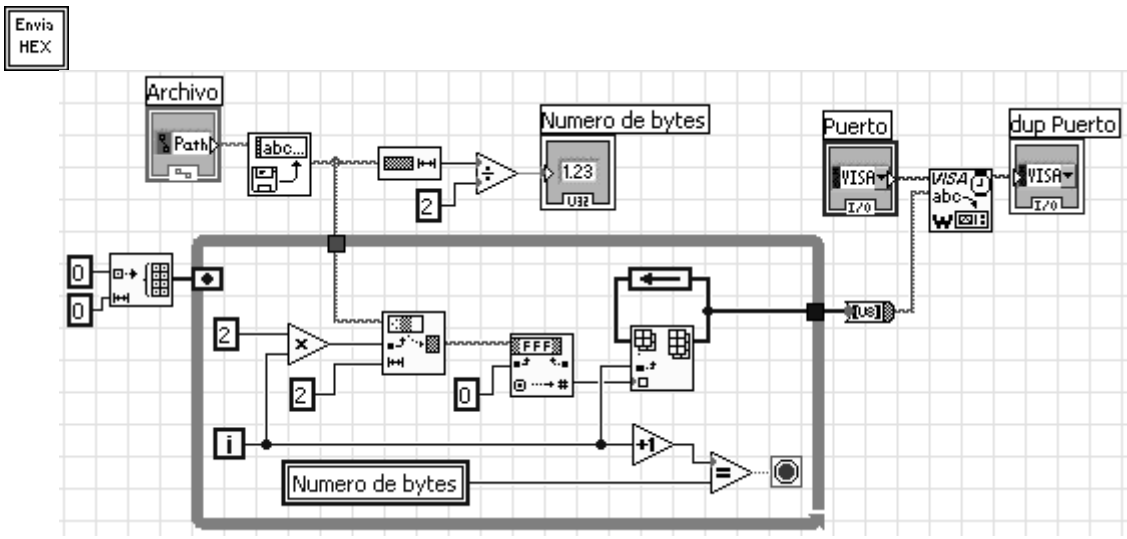
Evento "Programar". Este evento envía el byte 00h, el precargador, el cargador externo y el programa en formato Intel HEX al SAB80C166.



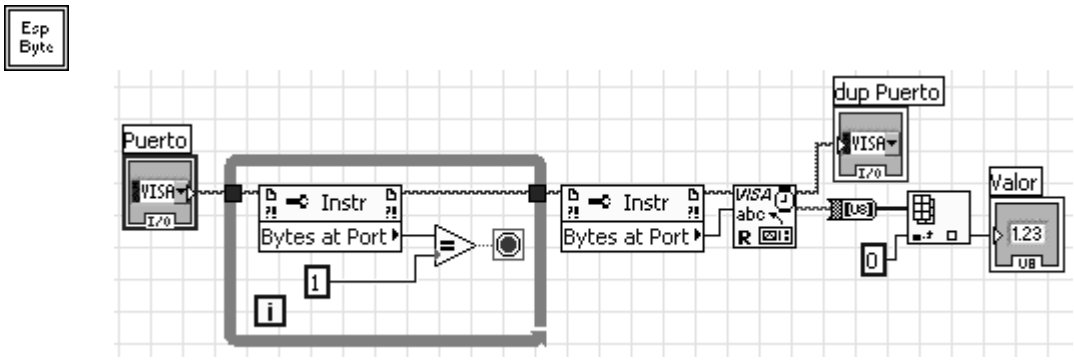
SubVI "BTL". Este subVI envía el byte 00h y recibe el byte 55h del SAB80C166.



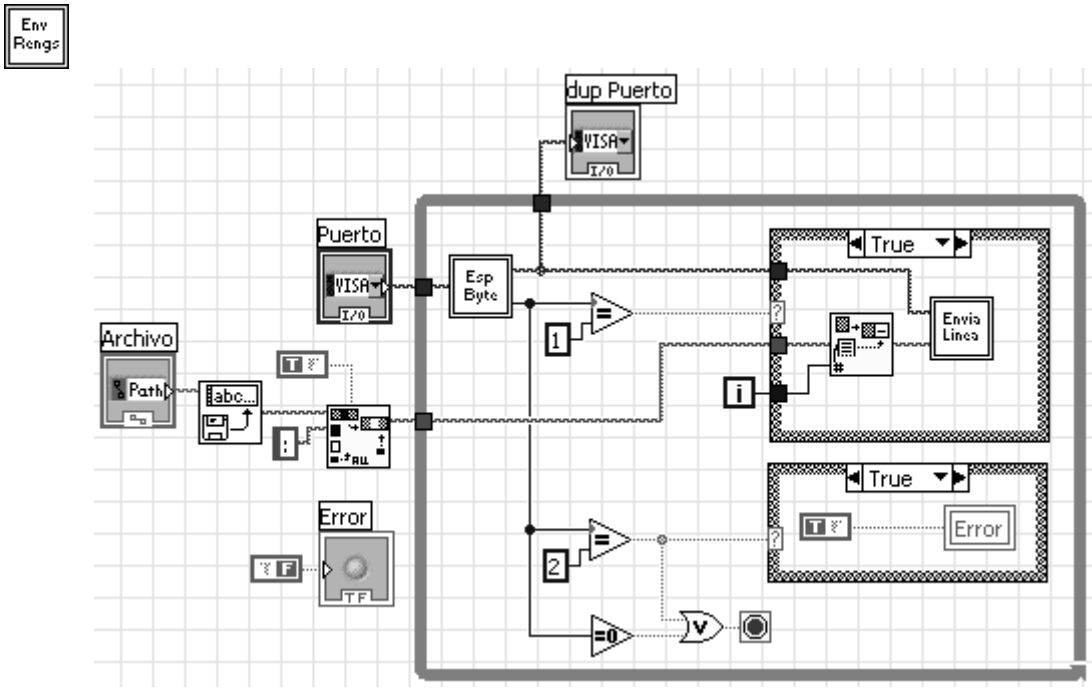
SubVI "Envía HEX". Este subVI toma los archivos en donde se encuentran el precargador y el cargador externo, los convierte a bytes y los envía al SAB80C166.



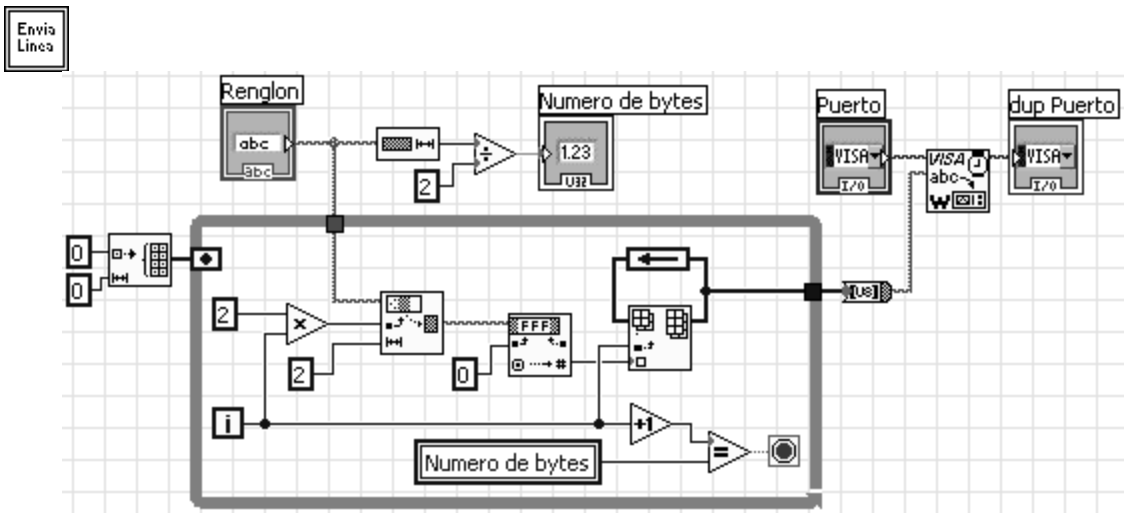
SubVI "Esp Byte". Este subVI recibe un byte por el puerto serial de la PC.



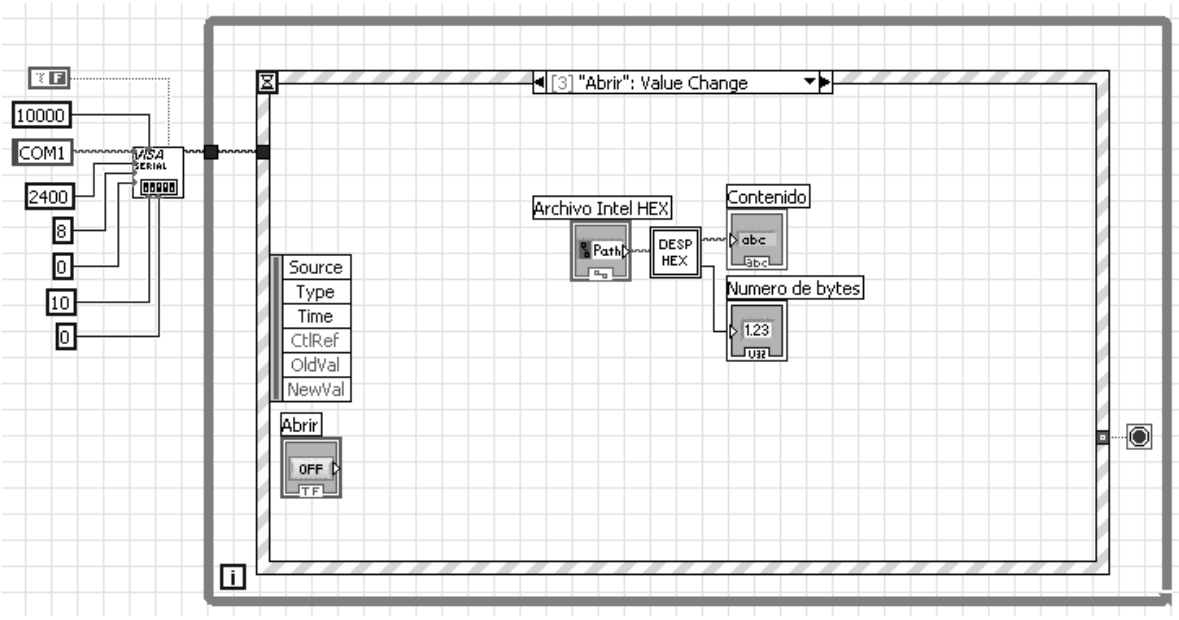
SubVI "Env Rengs". Este subVI envía renglón por renglón del programa en formato Intel HEX al SAB80C166.



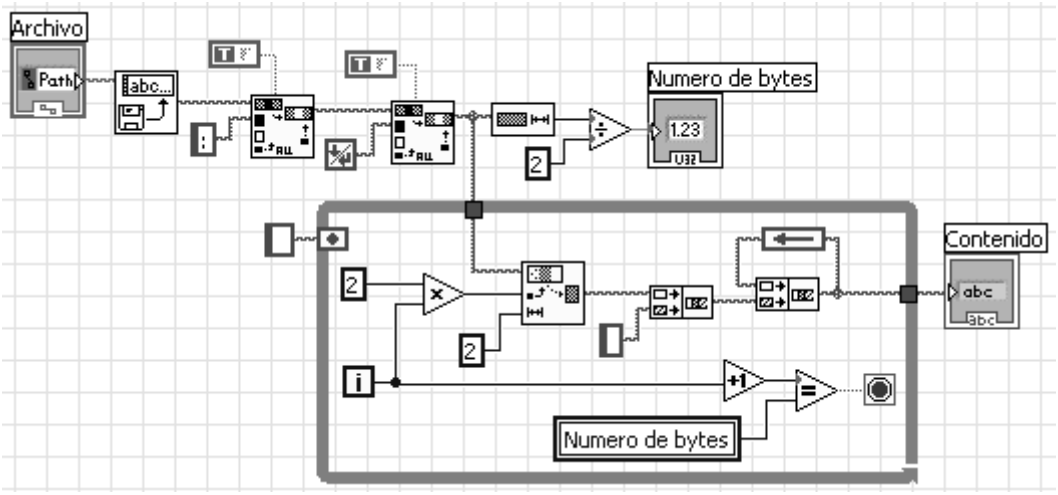
SubVI "Envía Línea". Este subVI convierte el renglón de caracteres del archivo Intel HEX en bytes y los envía por el puerto serial de la PC.



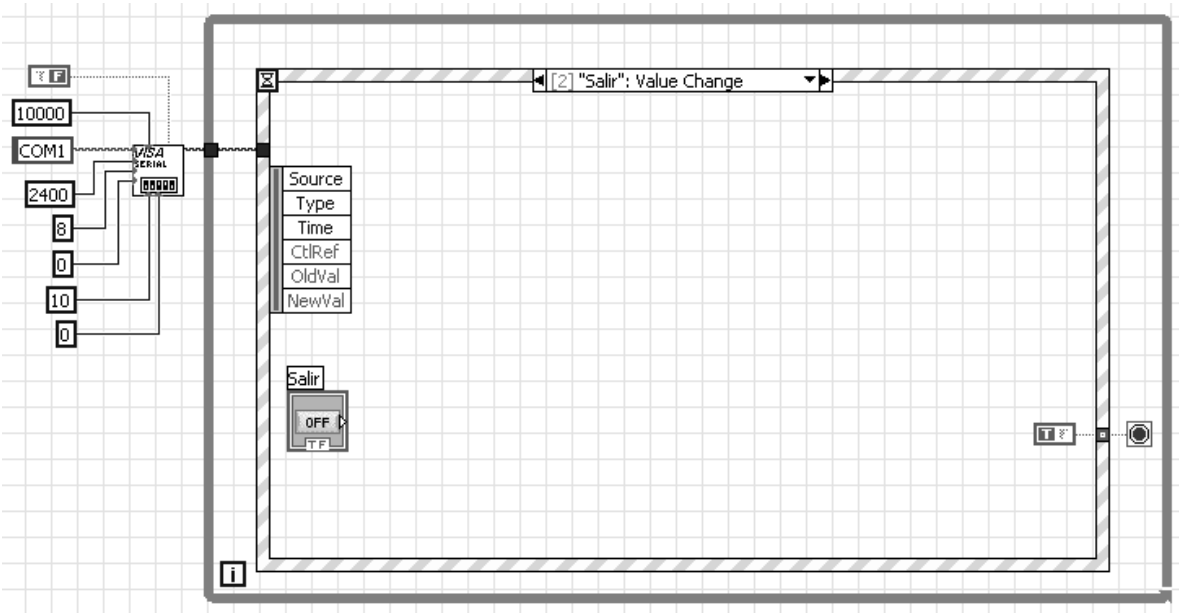
Evento "Abrir". Este evento abre un programa en formato Intel HEX y lo despliega en el campo "Contenido".



SubVI "DESP HEX". Este subVI separa cada byte de un programa en formato Intel HEX con un espacio.



Evento “Salir”. Este evento finaliza la ejecución del software en LabVIEW.



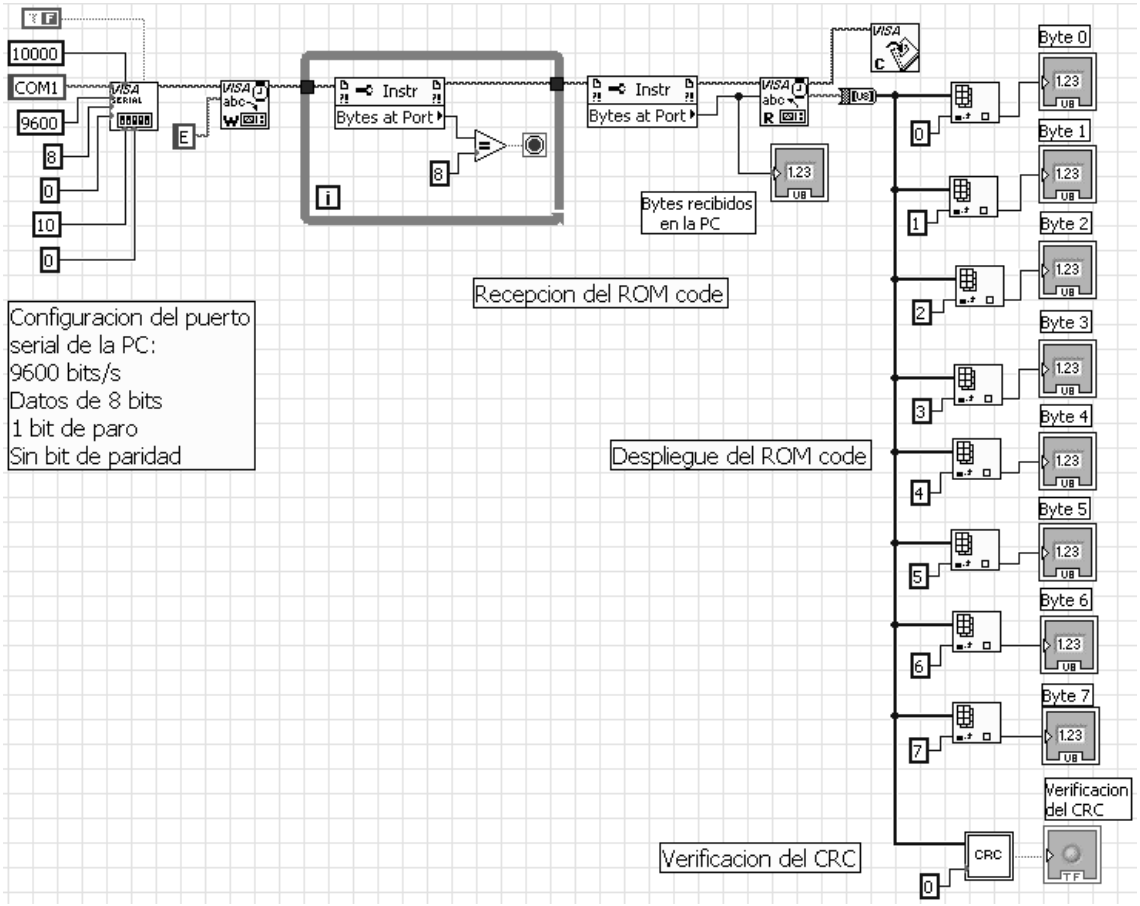
Apéndice D. Software para los sensores de temperatura 1-Wire

D.1 Software en el PIC16F876A para leer el ROM code

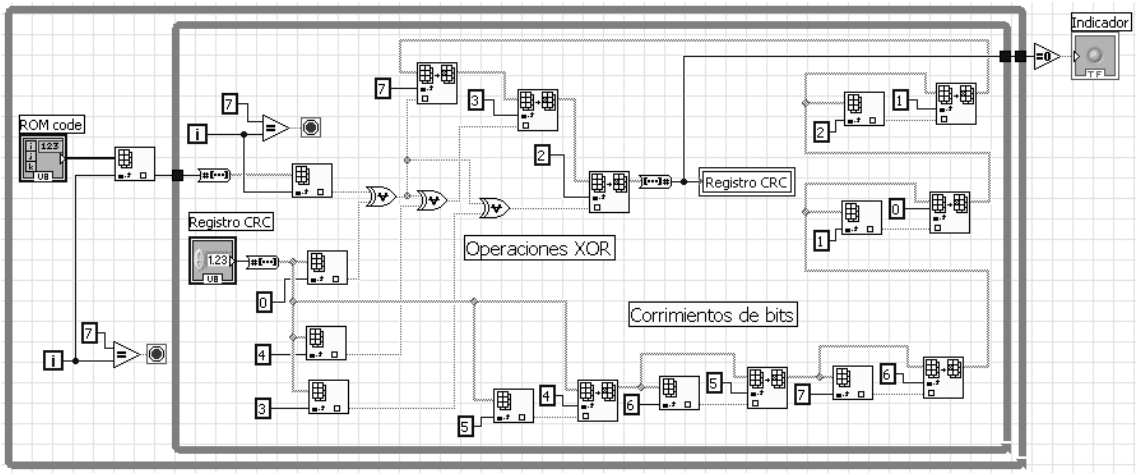
```
#include <16f876a.h>
#fuses HS,NOWDT,NOPUT,NOPROTECT,NODEBUG,NOBROWNOUT,NOLVP,NOCPD,NOWRT
#use delay(clock=8000000) // 8 MHz
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7,parity=n,bits=8)

#define DQ pin_b0

void inicializar ( ) {    // Inicializa el sensor 1-Wire
output_low(DQ);
delay_us(485);
output_float(DQ);
delay_us(65);
// En DQ debe de haber cero.
delay_us(420);
}

void enviaUno ( ) {    // Envia el bit 1 por el bus 1-Wire
delay_us(10);
output_low(DQ);
delay_us(10);
output_float(DQ);
delay_us(50);
}

void enviaCero ( ) {    // Envia el bit 0 por el bus 1-Wire
delay_us(10);
output_low(DQ);
delay_us(80);
output_float(DQ);
}

void enviaByte (byte dato) {       // Envia 1 byte por el bus 1-Wire
byte a;
for (a=0;a<8;a++) {
if(shift_right(&dato,1,0))
enviaUno();
else
enviaCero();
}
}

byte leeByte ( ) {       // Lee 1 byte por el bus 1-Wire
byte a,dato;
for(a=0;a<8;a++) {
delay_us(10);
output_low(DQ);
delay_us(2);
output_float(DQ);
delay_us(5);
shift_right(&dato,1,input(DQ));
delay_us(45);
}
return dato;
}

main ( ) {
int i, recib;
byte ROM_Code[8];
recib=getc( );
if (recib == 'E') {
inicializar( );             // Inicializar el sensor 1-Wire
enviaByte(0x33);       // Comando Read Rom
for(i=0;i<8;i++)
```

```
ROM_Code[i]=leeByte( );          // Guarda el ROM Code de 64 bits (8 bytes)
delay_ms(100);
for(i=0;i<8;i++)
putc(ROM_Code[i]); // Envia el ROM Code a la PC
}
}
```

D.2 Software en LabVIEW para desplegar y verificar el ROM code



SubVI "CRC". Este subVI verifica el valor del ROM code.



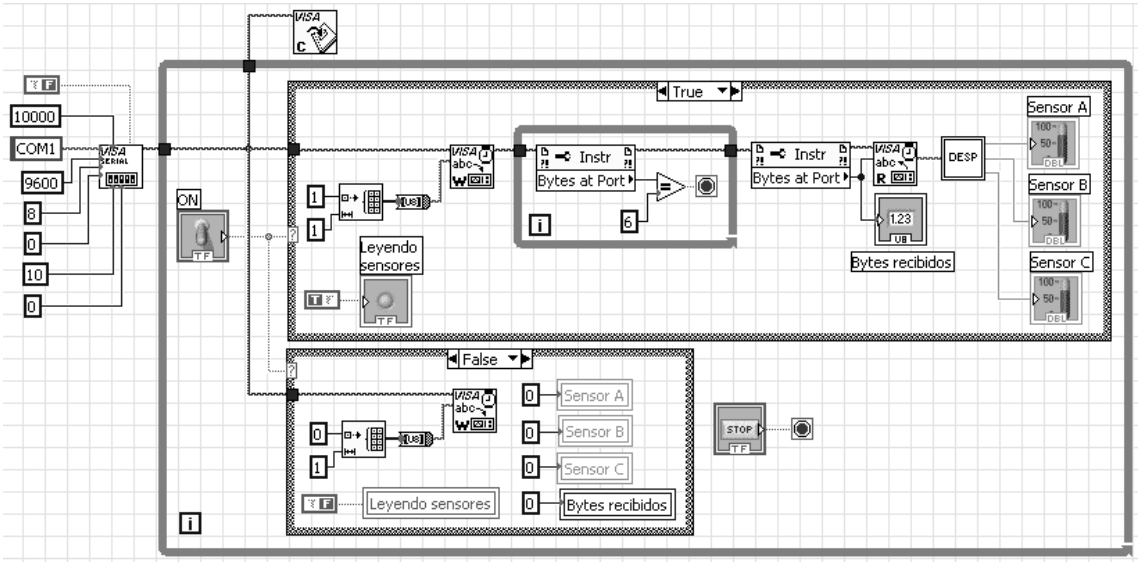
D.3 Software para leer los 3 sensores de temperatura de la computadora de vuelo

```
#include <stdio.h>
#include <c166.h>
#include <reg166.h>

/* P3.3 es DQ */

void iniSer0 (void) { /* Configura el canal serial 0 */
_bfld(DP3,0x0C00,0x0400); /* P3.11 = RX0, P3.10 = TX0 */
_putbit(1,P3,10);
S0BG=0x003F; /* 9600 bauds */
S0TIC=0x0000; /* S0TIR, S0TIE, ILVL, GLVL */
S0RIC=0x0000; /* S0RIR, S0RIE, ILVL, GLVL */
S0EIC=0x0000; /* S0EIR, S0EIE, ILVL, GLVL */
S0CON=0x8011; /* Rx habilitado, 8 bits, 1 bit de paro, BG habilitado */
}

void txByte0 (unsigned int c) { /* Envia 1 byte por el canal serial 0 */
S0TBUF=c;
while(!S0TIR);
S0TIR=0;
}

unsigned int rxByte0 (void) { /* Recibe 1 byte por el canal serial 0 */
unsigned int c;
while(!S0RIR);
c=S0RBUF;
S0RIR=0;
return c;
}

void retC (unsigned int a) { /* Retardo de tiempo de hasta 26 ms */
T0=a;                        /* a=65535-(tiempo/0.4us) */
T0R=1;
while(!T0IR);
T0R=0;
T0IR=0;
}
```

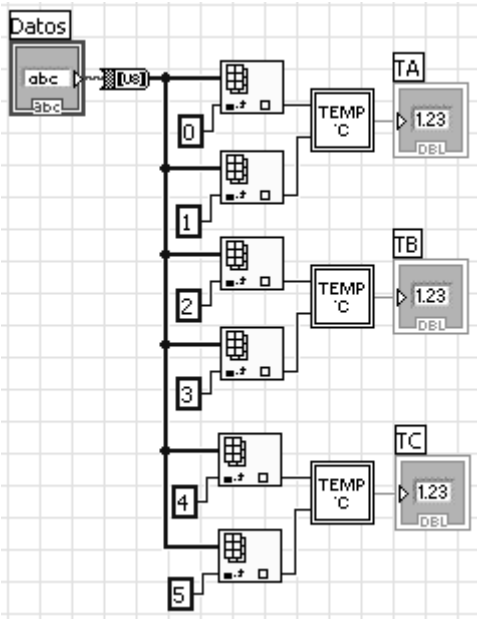
```
void retL (unsigned int a) { /* Retardo de tiempo de hasta 3.36 s */
T1=a;                        /* a=65535-(tiempo/51.2us) */
T1R=1;
while(!T1IR);
T1R=0;
T1IR=0;
}

void inicializar (void) { /* Inicializa los sensores 1-Wire */
_putbit(1,DP3,3); /* P3.3 es salida */
_putbit(0,P3,3);
retC(64328); /* 485us */
_putbit(0,DP3,3); /* P3.3 es entrada */
retC(65378); /* 65us */
/* En DQ debe de haber cero */
retC(64490); /* 420us */
}

void enviaUno (void) { /* Envia el bit 1 por el bus 1-Wire */
retC(65515); /* 10us */
_putbit(1,DP3,3); /* P3.3 es salida */
_putbit(0,P3,3);
retC(65515); /* 10us */
_putbit(0,DP3,3); /* P3.3 es entrada */
retC(65415); /* 50us */
}

void enviaCero (void) { /* Envia el bit 0 por el bus 1-Wire */
retC(65515); /* 10us */
_putbit(1,DP3,3); /* P3.3 es salida */
_putbit(0,P3,3);
retC(65340); /* 80us */
_putbit(0,DP3,3); /* P3.3 es entrada */
}

void enviaByte (unsigned int dato) { /* Envia 1 byte por el bus 1-Wire */
unsigned int a,bitEnv;
for(a=0;a<8;a++){
bitEnv=_ror(dato,a);
bitEnv=bitEnv&0x0001;
if(bitEnv==0x0001)
enviaUno();
else
enviaCero();
}
}

unsigned int leeByte (void) { /* Lee 1 byte por el bus 1-Wire */
unsigned int a,dato=0x0000,bitC;
for(a=0;a<8;a++){
bitC=_rol(0x0001,a);
retC(65515); /* 10us */
_putbit(1,DP3,3); /* P3.3 es salida */
_putbit(0,P3,3);
retC(65535); /* 2us */
_putbit(0,DP3,3); /* P3.3 es entrada */
retC(65528); /* 5us */
if(_getbit(P3,3))
dato=dato|bitC;
retC(65428); /* 45us */
}
return dato;
}

main () {
unsigned int ROM_CODE_A[8]={0x10,0xE0,0x02,0xFF,0x00,0x08,0x00,0xCC};
unsigned int ROM_CODE_B[8]={0x10,0x51,0x96,0x24,0x01,0x08,0x00,0x6A};
unsigned int ROM_CODE_C[8]={0x10,0x87,0xFF,0x23,0x01,0x08,0x00,0x78};
unsigned int recib,LSB_A,MSB_A,LSB_B,MSB_B,LSB_C,MSB_C,x;
```

```
T01CON=0x0700; /* Configura timers con */
T0IR=0;        /* desbordamientos T0=26ms y T1=3.36s */
T1IR=0;
iniSer0(); /* Configura el canal serial 0 */
while(1) {
recib=rxByte0();
switch (recib){
case 0:
_nop();
break;
case 1:
inicializar();
enviaByte(0xCC); /* Comando Skip Rom */
enviaByte(0x44); /* Comando Convert T */
retL(49910); /* 800ms */
/* Sensor A */
inicializar();
enviaByte(0x55); /* Comando Match Rom */
for(x=0;x<8;x++)
enviaByte(ROM_CODE_A[x]); /* Envia el Rom Code del sensor A */
enviaByte(0xBE); /* Comando Read Scratchpad */
LSB_A=leeByte();
MSB_A=leeByte(); /* Byte de signo */
/* Sensor B */
inicializar();
enviaByte(0x55); /* Comando Match Rom */
for(x=0;x<8;x++)
enviaByte(ROM_CODE_B[x]); /* Envia el Rom Code del sensor B */
enviaByte(0xBE); /* Comando Read Scratchpad */
LSB_B=leeByte();
MSB_B=leeByte(); /* Byte de signo */
/* Sensor C */
inicializar();
enviaByte(0x55); /* Comando Match Rom */
for(x=0;x<8;x++)
enviaByte(ROM_CODE_C[x]); /* Envia el Rom Code del sensor C */
enviaByte(0xBE); /* Comando Read Scratchpad */
LSB_C=leeByte();
MSB_C=leeByte(); /* Byte de signo */
txByte0(MSB_A); /* Envia las lecturas de */
txByte0(LSB_A); /* temperatura a la PC */
txByte0(MSB_B);
txByte0(LSB_B);
txByte0(MSB_C);
txByte0(LSB_C);
break;
}
}
}
```

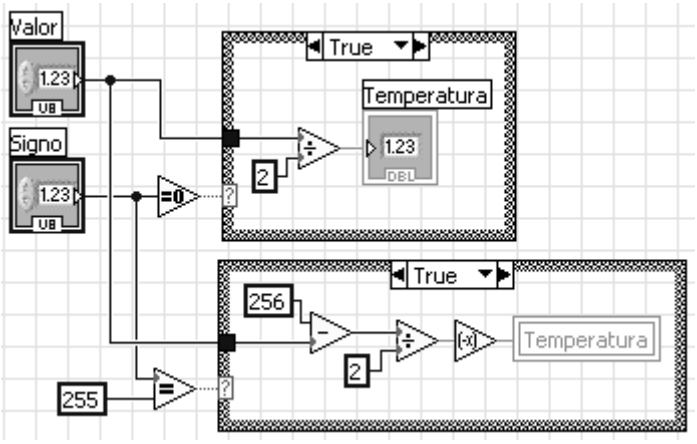
D.4 Software en LabVIEW para desplegar las lecturas de los sensores de temperatura



SubVI "DESP". Este subVI despliega las lecturas de los 3 sensores de temperatura.



SubVI "TEMP °C". Este subVI calcula la temperatura en °C de cada sensor a partir de los bytes recibidos por el puerto serial.



Apéndice E. Software para la escritura, lectura y borrado de la memoria flash

E.1 Software en el SAB80C166

```
#include <stdio.h>
#include <c166.h>
#include <reg166.h>

/* CS#/MEM1 : P2.6 */
/* CS#/MEM2 : P2.7 */
/* CS#/MEM3 : P2.8 */
/* CS#/MEM4 : P2.9 */
/* SCK/MEMS : P2.10 */
/* SI/MEMS : P2.11 */
/* SO/MEMS : P2.12 */

unsigned int escMemFlash[1056]; /* Bytes a escribir en la memoria flash */
unsigned int leeMemFlash[1056]; /* Bytes leidos de la memoria flash */

unsigned int tiempo=65415; /* retC(tiempo) producira un retardo de 50us */
                           /* y una interfaz SPI de 10KHz */

void iniSer0 (void) { /* Configura el canal serial 0 */
_bfld(DP3,0x0C00,0x0400); /* P3.11 = RX0, P3.10 = TX0 */
_putbit(1,P3,10);
S0BG=0x003F; /* 9600 bauds */
S0TIC=0x0000; /* S0TIR, S0TIE, ILVL, GLVL */
S0RIC=0x0000; /* S0RIR, S0RIE, ILVL, GLVL */
S0EIC=0x0000; /* S0EIR, S0EIE, ILVL, GLVL */
S0CON=0x8011; /* Rx habilitado, 8 bits, 1 bit de paro, BG habilitado */
}

void txByte0 (unsigned int c) { /* Envia 1 byte por el canal serial 0 */
S0TBUF=c;
while(!S0TIR);
S0TIR=0;
}

unsigned int rxByte0 (void) { /* Recibe 1 byte por el canal serial 0 */
unsigned int c;
while(!S0RIR);
c=S0RBUF;
S0RIR=0;
return c;
}

void retC (unsigned int a) { /* Retardo de tiempo de hasta 26 ms */
T0=a;                        /* a=65535-(tiempo/0.4us) */
T0R=1;
while(!T0IR);
T0R=0;
T0IR=0;
}

void retL (unsigned int a) { /* Retardo de tiempo de hasta 3.36 s */
T1=a;                        /* a=65535-(tiempo/51.2us) */
T1R=1;
while(!T1IR);
T1R=0;
T1IR=0;
}

void escSPI (unsigned int enviar) { /* Escribe 1 byte por el bus SPI */
unsigned int i,bitC,bitEnv;
for (i=0;i<8;i++) {
_putbit(0,P2,10); /* SCK/MEMS */
```

```
bitC=_ror(0x0080,i);
bitEnv=enviar&bitC;
if (bitEnv == 0)
_putbit(0,P2,11); /* SI/MEMS */
else
_putbit(1,P2,11); /* SI/MEMS */
retC(tiempo); /* retardo de tiempo */
_putbit(1,P2,10); /* SCK/MEMS */
retC(tiempo); /* retardo de tiempo */
}
}

unsigned int leeSPI (void) { /* Lee 1 byte por el bus SPI */
unsigned int i,bitC,x=0x0000;
for (i=0;i<8;i++) {
_putbit(0,P2,10); /* SCK/MEMS */
retC(tiempo); /* retardo de tiempo */
bitC=_ror(0x0080,i);
if(_getbit(P2,12)) /* SO/MEMS */
x=x|bitC;
_putbit(1,P2,10); /* SCK/MEMS */
retC(tiempo); /* retardo de tiempo */
}
return x;
}

void habilit (unsigned int nMem) { /* Habilita la memoria flash */
switch (nMem) {
case 1:
_putbit(0,P2,6); /* CS#/MEM1 */
break;
case 2:
_putbit(0,P2,7); /* CS#/MEM2 */
break;
case 3:
_putbit(0,P2,8); /* CS#/MEM3 */
break;
case 4:
_putbit(0,P2,9); /* CS#/MEM4 */
break;
}
}

void deshabilit (unsigned int nMem) { /* Deshabilita la memoria flash */
switch (nMem) {
case 1:
_putbit(1,P2,6); /* CS#/MEM1 */
break;
case 2:
_putbit(1,P2,7); /* CS#/MEM2 */
break;
case 3:
_putbit(1,P2,8); /* CS#/MEM3 */
break;
case 4:
_putbit(1,P2,9); /* CS#/MEM4 */
break;
}
}

void escFlash (unsigned int nMem, unsigned int nBuff, unsigned int pa, unsigned int bfa, unsigned int nBytes) {
unsigned int cmdEsc,B1,B2,B3,a,b,i;
switch (nBuff) {
case 1:
cmdEsc=0x0082; /* Comando de escritura usando el buffer SRAM 1 */
break;
case 2:
cmdEsc=0x0085; /* Comando de escritura usando el buffer SRAM 2 */
break;
```

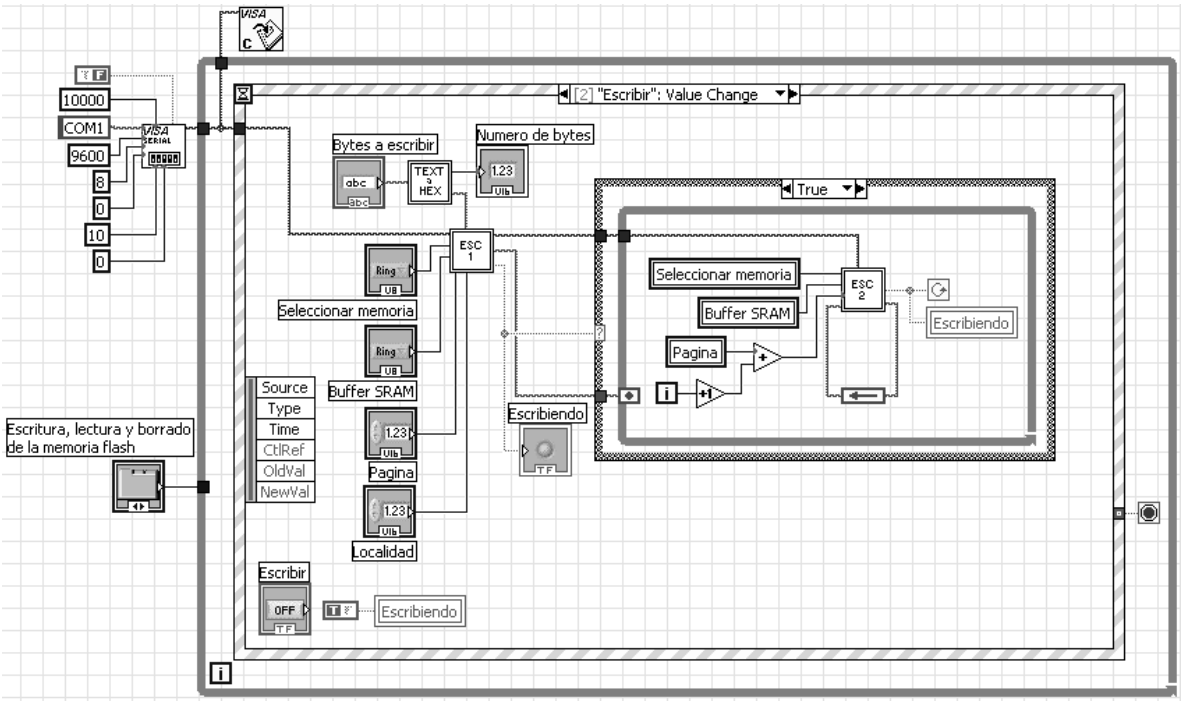
```
}
B1=_ror(pa,5);
B1=0x00FF&B1; /* Byte de direccion 1 */
a=_rol(pa,3);
a=0x00F8&a;
b=_ror(bfa,8);
b=0x0007&b;
B2=a|b;
B2=0x00FF&B2; /* Byte de direccion 2 */
B3=0x00FF&bfa; /* Byte de direccion 3 */
habilit(nMem); /* Habilita la memoria flash */
/* Enviar el comando de escritura [82h] para el buffer 1 o [85h] para el buffer 2 */
/* Comando Main memory page program through buffer */
escSPI(cmdEsc);
/* Enviar 3 bytes de direccion. Pagina: PA12 - PA0. Byte: BFA10 - BFA0 */
/* Pagina pa. Byte bfa. */
escSPI(B1);
escSPI(B2);
escSPI(B3);
/* Enviar los bytes a escribir */
for (i=0;i<nBytes;i++) {
escSPI(escMemFlash[i]);
}
deshabilit(nMem); /* Deshabilita la memoria flash */
_putbit(1,P2,10); /* SCK/MEMS */
_putbit(0,P2,11); /* SI/MEMS */
retL(64558); /* 50 ms para que se escriban los bytes en la memoria flash */
}

void leeFlash (unsigned int nMem, unsigned int pa, unsigned int ba, unsigned int nBytes) {
unsigned int B1,B2,B3,a,b,i;
B1=_ror(pa,5);
B1=0x00FF&B1; /* Byte de direccion 1 */
a=_rol(pa,3);
a=0x00F8&a;
b=_ror(ba,8);
b=0x0007&b;
B2=a|b;
B2=0x00FF&B2; /* Byte de direccion 2 */
B3=0x00FF&ba; /* Byte de direccion 3 */
habilit(nMem); /* Habilita la memoria flash */
/* Enviar el comando de lectura [D2h] */
/* Comando Main memory page read */
escSPI(0x00D2);
/* Enviar 3 bytes de direccion. Pagina: PA12 - PA0. Byte: BA10 - BA0 */
/* Pagina pa. Byte ba. */
escSPI(B1);
escSPI(B2);
escSPI(B3);
/* Enviar 4 bytes tipo don´t care */
escSPI(0x0055);
escSPI(0x0055);
escSPI(0x0055);
escSPI(0x0055);
/* Recibir los bytes por el pin SO */
for (i=0;i<nBytes;i++) /* Los bytes leidos de la memoria flash */
leeMemFlash[i]=leeSPI(); /* se guardan en el arreglo leeMemFlash */
deshabilit(nMem); /* Deshabilita la memoria flash */
_putbit(1,P2,10); /* SCK/MEMS */
_putbit(0,P2,11); /* SI/MEMS */
}

void borraPag (unsigned int nMem, unsigned int dir) {
unsigned int B1,B2;
B1=_ror(dir,5);
B1=0x00FF&B1;
B2=_rol(dir,3);
B2=0x00F8&B2;
habilit(nMem); /* Habilita la memoria flash */
```

```
/* Comando Page erase */
escSPI(0x0081);
/* Enviar los 3 bytes de direccion */
escSPI(B1);
escSPI(B2);
escSPI(0x0000); /* Byte tipo don´t care */
deshabilit(nMem); /* Deshabilita la memoria flash */
_putbit(1,P2,10); /* SCK/MEMS */
_putbit(0,P2,11); /* SI/MEMS */
retL(64558); /* 50 ms para que se borre la pagina (DS=35ms) */
}

void borraBloq (unsigned int nMem, unsigned int dir) {
unsigned int B1,B2;
B1=_ror(dir,5);
B1=0x001F&B1;
B2=_rol(dir,3);
B2=0x00F8&B2;
habilit(nMem); /* Habilita la memoria flash */
/* Comando Block erase */
escSPI(0x0050);
/* Enviar los 3 bytes de direccion */
escSPI(0x0000); /* Byte tipo don´t care */
escSPI(B1);
escSPI(B2);
deshabilit(nMem); /* Deshabilita la memoria flash */
_putbit(1,P2,10); /* SCK/MEMS */
_putbit(0,P2,11); /* SI/MEMS */
retL(63191); /* 120 ms para que se borre el bloque (DS=100ms) */
}

/* CS#/MEM1 : P2.6 */
/* CS#/MEM2 : P2.7 */
/* CS#/MEM3 : P2.8 */
/* CS#/MEM4 : P2.9 */
/* SCK/MEMS : P2.10 */
/* SI/MEMS : P2.11 */
/* SO/MEMS : P2.12 */

main ( ) {
unsigned int recib,nMem,nBuff,paH,paL,bfaH,bfaL,nBytesH,nBytesL,nBytes,pa,bfa,baH,baL,ba,i,tipoB,dirH,dirL,dir;
T01CON=0x0700; /* Configura timers con desbordamientos T0=26ms y T1=3.36s */
T0IR=0; /* Limpia la bandera de interrupcion de */
T1IR=0; /* timer */
iniSer0(); /* Configura el puerto serial 0 */
_bfld(DP2,0x1FC0,0x0FC0); /* Memorias flash */
_bfld(P2,0x0FC0,0x07C0); /* deshabilitadas */
while (1) {
recib=rxByte0(); /* Espera el primer byte del comando */
switch (recib) {
case 'E': /* Comando para escribir en la memoria flash */
nMem=rxByte0(); /* Numero de memoria */
nBuff=rxByte0(); /* Numero de buffer de datos SRAM */
paH=rxByte0(); /* Numero de pagina, */
paL=rxByte0(); /* 13 bits */
bfaH=rxByte0(); /* Localidad de inicio dentro */
bfaL=rxByte0(); /* de la pagina, 11 bits */
nBytesH=rxByte0(); /* Bit 15 - Bit 8 */
nBytesL=rxByte0(); /* Bit 7 - Bit 0 */
paH=_rol(paH,8);
paH=0xFF00&paH;
paL=0x00FF&paL;
pa=paH|paL; /* Numero de pagina, 13 bits */
bfaH=_rol(bfaH,8);
bfaH=0xFF00&bfaH;
bfaL=0x00FF&bfaL;
bfa=bfaH|bfaL; /* Localidad de inicio dentro de la pagina, 11 bits */
nBytesH=_rol(nBytesH,8);
nBytesH=0xFF00&nBytesH;
```

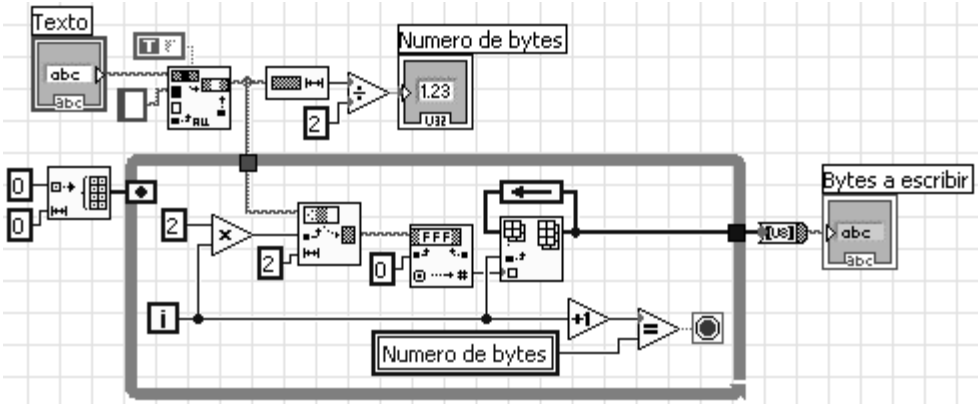
```
nBytesL=0x00FF&nBytesL;
nBytes=nBytesH|nBytesL; /* Numero de bytes a escribir */
txByte0('k');
for (i=0;i<nBytes;i++)
escMemFlash[i]=rxByte0(); /* Recibe los bytes */
escFlash(nMem,nBuff,pa,bfa,nBytes); /* Escribe los bytes */
txByte0('k');
break;
case 'L': /* Comando para leer la memoria flash */
nMem=rxByte0(); /* Numero de memoria */
paH=rxByte0(); /* Numero de pagina, */
paL=rxByte0(); /* 13 bits */
baH=rxByte0(); /* Localidad de inicio dentro */
baL=rxByte0(); /* de la pagina, 11 bits */
nBytesH=rxByte0(); /* Bit 15 - Bit 8 */
nBytesL=rxByte0(); /* Bit 7 - Bit 0 */
paH=_rol(paH,8);
paH=0xFF00&paH;
paL=0x00FF&paL;
pa=paH|paL; /* Numero de pagina, 13 bits */
baH=_rol(baH,8);
baH=0xFF00&baH;
baL=0x00FF&baL;
ba=baH|baL; /* Localidad de inicio dentro de la pagina, 11 bits */
nBytesH=_rol(nBytesH,8);
nBytesH=0xFF00&nBytesH;
nBytesL=0x00FF&nBytesL;
nBytes=nBytesH|nBytesL; /* Numero de bytes a leer */
leeFlash(nMem,pa,ba,nBytes); /* Lee los bytes */
for (i=0;i<nBytes;i++)
txByte0(leeMemFlash[i]); /* Transmite los bytes a la PC */
break;
case 'B': /* Comando para borrar la memoria flash */
tipoB=rxByte0(); /* Borrado de pagina (01h) o bloque (02h) */
nMem=rxByte0(); /* Numero de memoria */
dirH=rxByte0(); /* Byte de direccion */
dirL=rxByte0(); /* Byte de direccion */
dirH=_rol(dirH,8);
dirH=0xFF00&dirH;
dirL=0x00FF&dirL;
dir=dirH|dirL; /* Direccion */
switch (tipoB) {
case 1:
borraPag(nMem,dir); /* Borra una pagina */
break;
case 2:
borraBloq(nMem,dir); /* Borra un bloque */
break;
}
txByte0('k');
break;
}
}
}
```

E.2 Software en LabVIEW

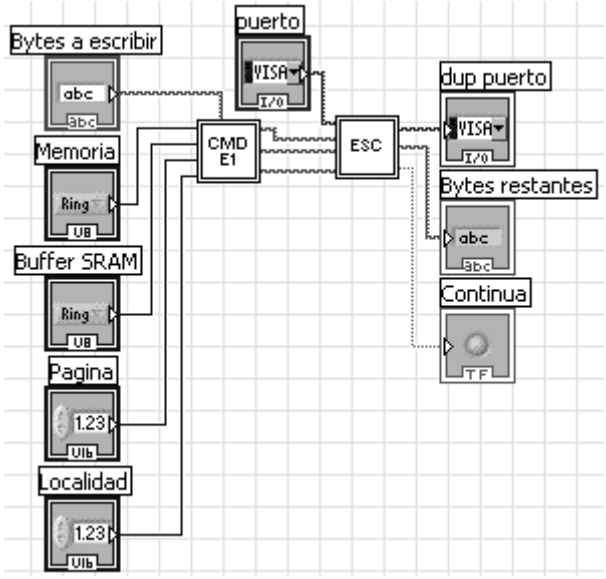
Evento "Escribir". Este evento envía al SAB80C166 el o los comandos de escritura y los bytes que se quieren escribir en la memoria flash.



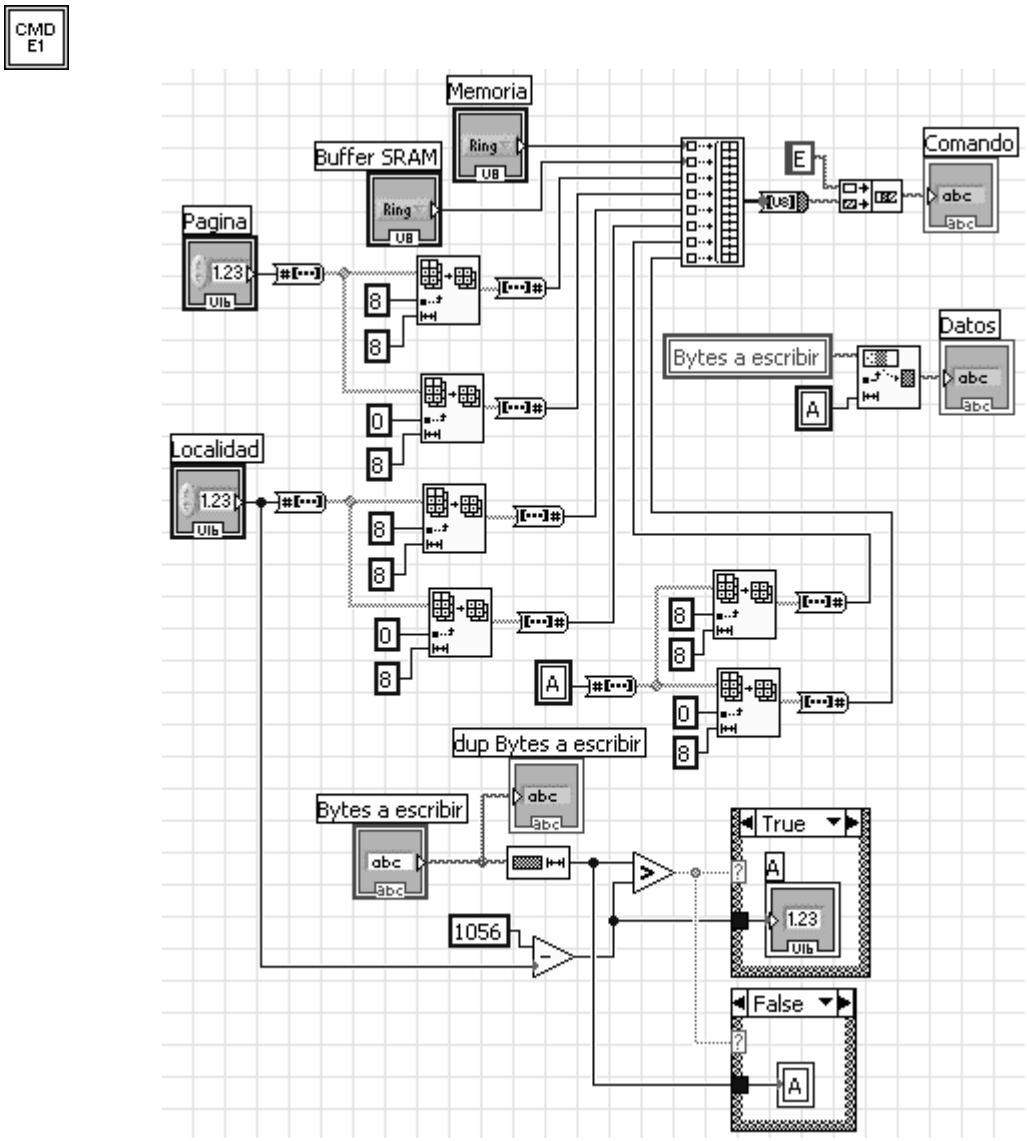
SubVI "TEXT a HEX". Este subVI convierte la cadena de caracteres que contiene los bytes a escribir en un arreglo de bytes.



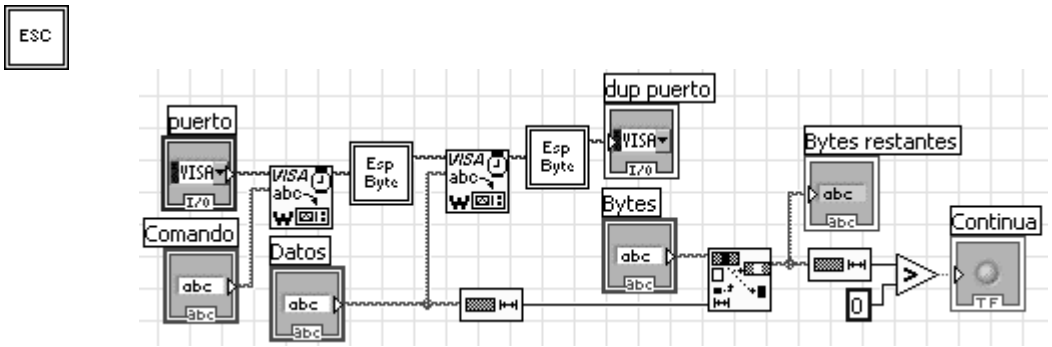
SubVI "ESC 1". Este subVI envía al SAB80C166 el primer comando de escritura y los bytes a escribir.



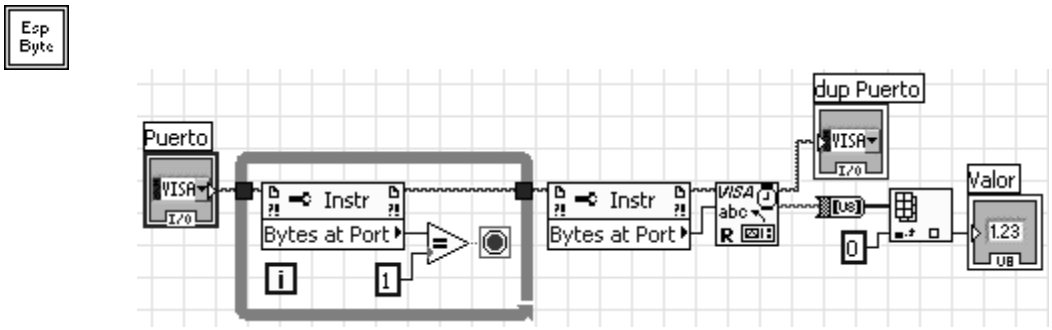
SubVI "CMD E1". Este subVI crea el primer comando de escritura de la memoria flash.



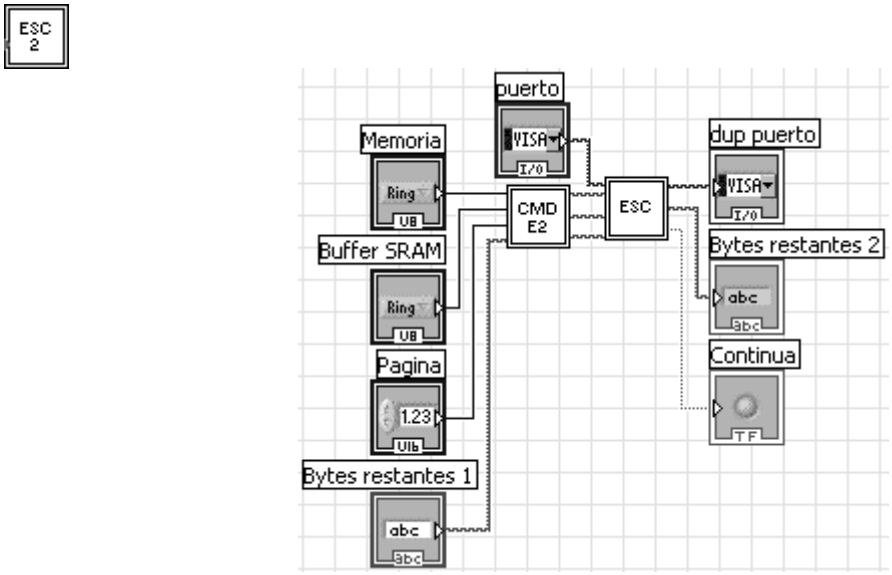
SubVI "ESC". Este subVI envía al SAB80C166 un comando de escritura y los bytes a escribir en la memoria flash.



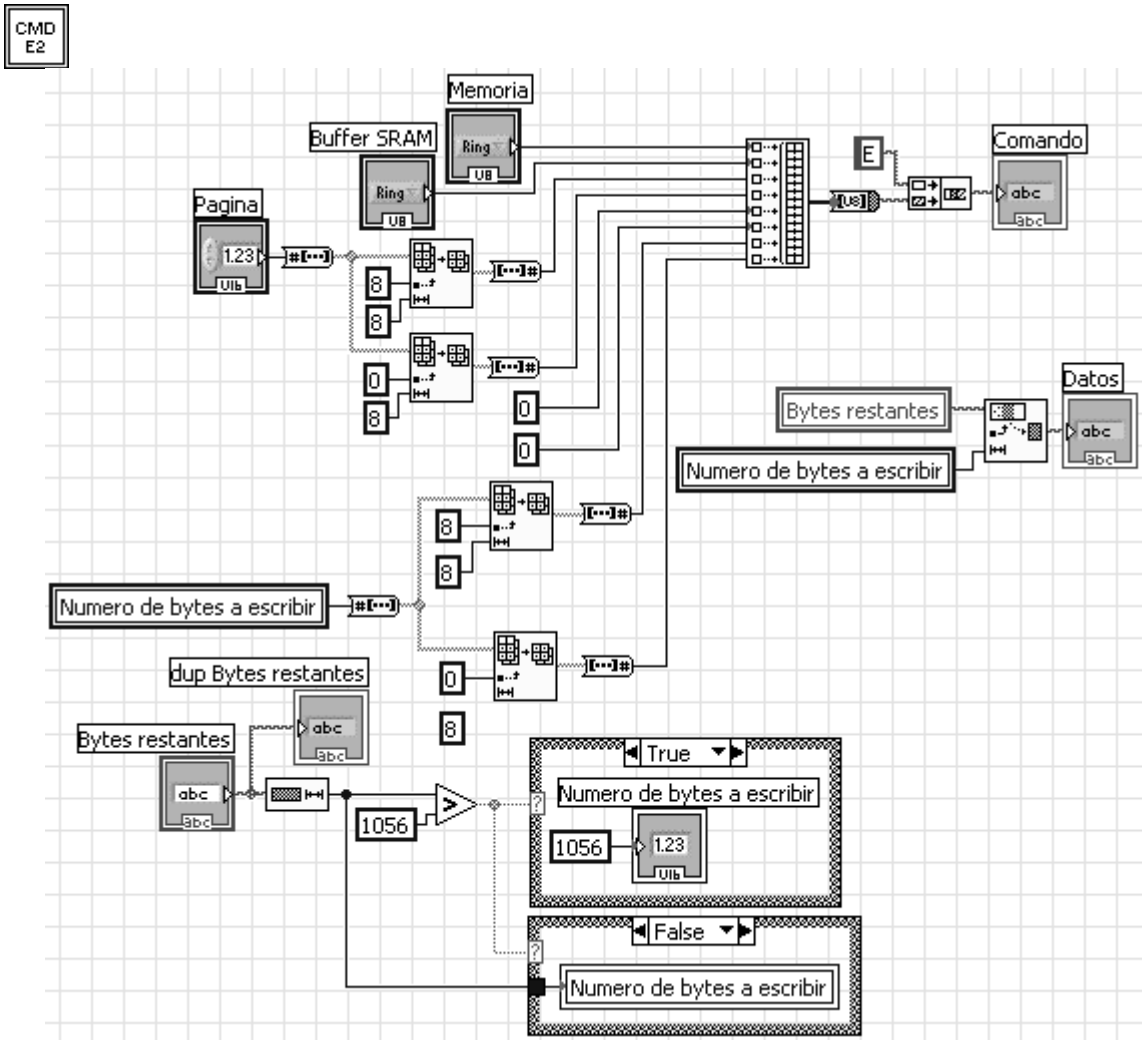
SubVI "Esp Byte". Este subVI recibe un byte por el puerto serial de la PC.



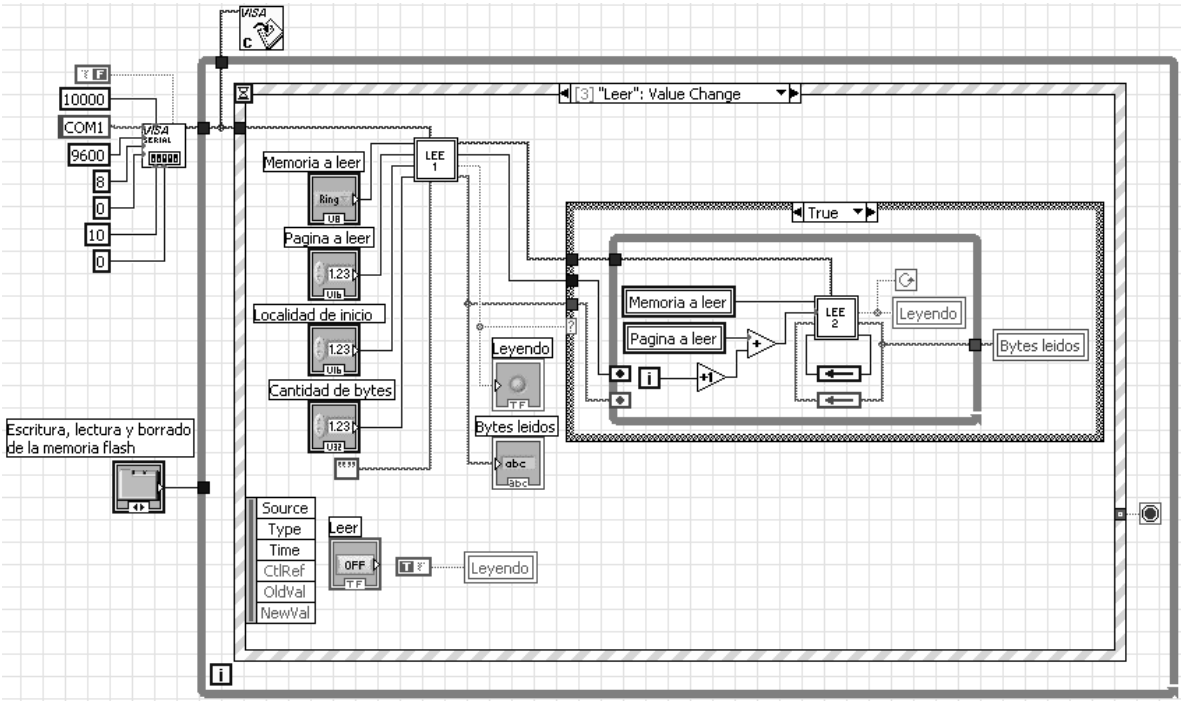
SubVI "ESC 2". Este subVI envía al SAB80C166 comandos de escritura y bytes a escribir cuando se requiere más de un comando para escribir los datos en la memoria flash.



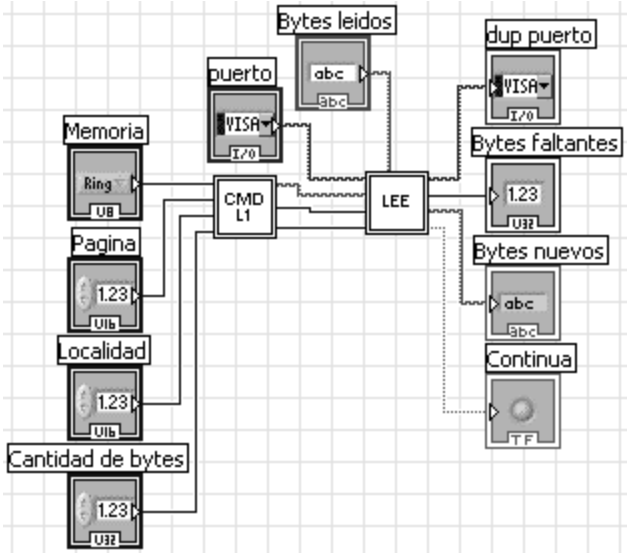
SubVI "CMD E2". Este subVI crea comandos de escritura cuando se requiere más de uno para escribir los datos en la memoria flash.



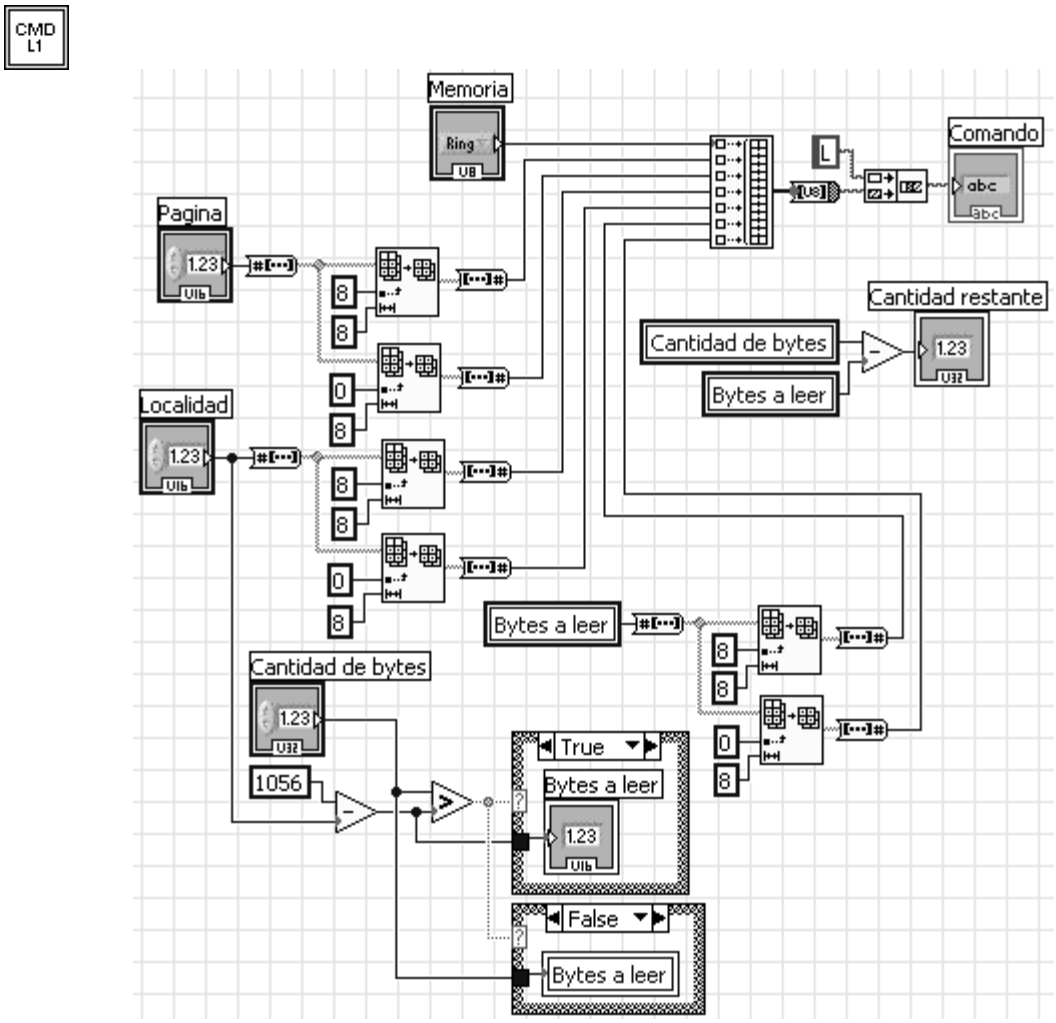
Evento "Leer". Este evento envía el o los comandos de lectura al SAB80C166 y recibe los bytes leídos de la memoria flash para desplegarlos.



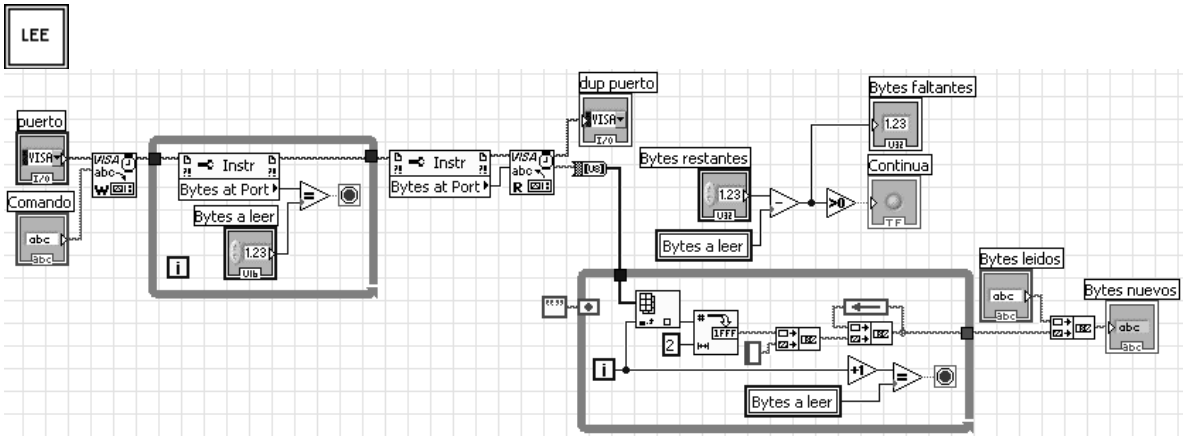
SubVI "LEE 1". Este subVI envía al SAB80C166 el primer comando de lectura de la memoria flash y recibe los bytes leídos.



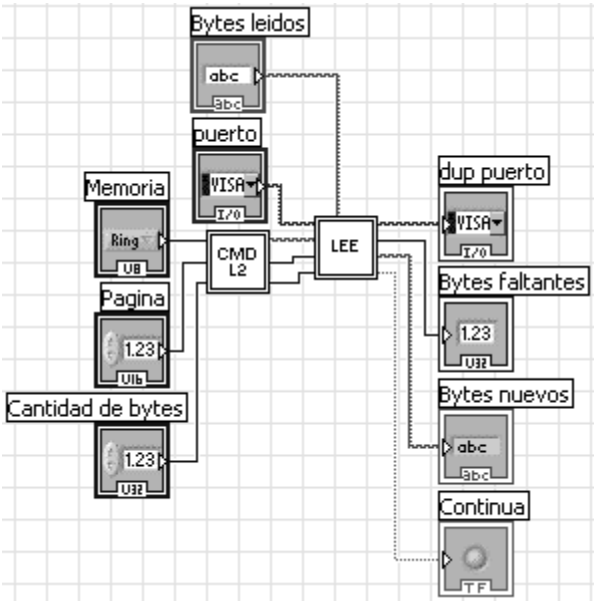
SubVI "CMD L1". Este subVI crea el primer comando de lectura de la memoria flash.



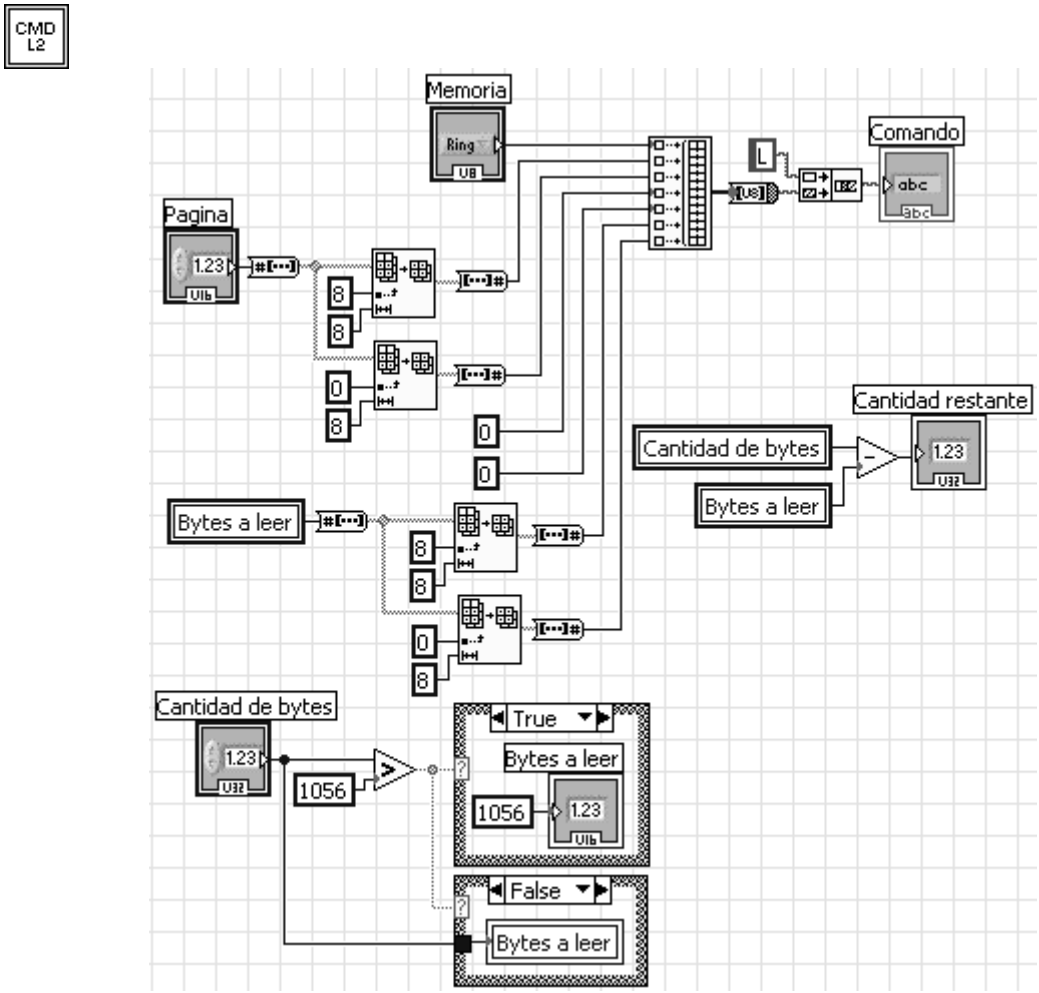
SubVI "LEE". Este subVI envía al SAB80C166 comandos de lectura de la memoria flash y recibe los bytes leídos. Cada byte leído se separa con un espacio para desplegarlo en la interfaz de LabVIEW.



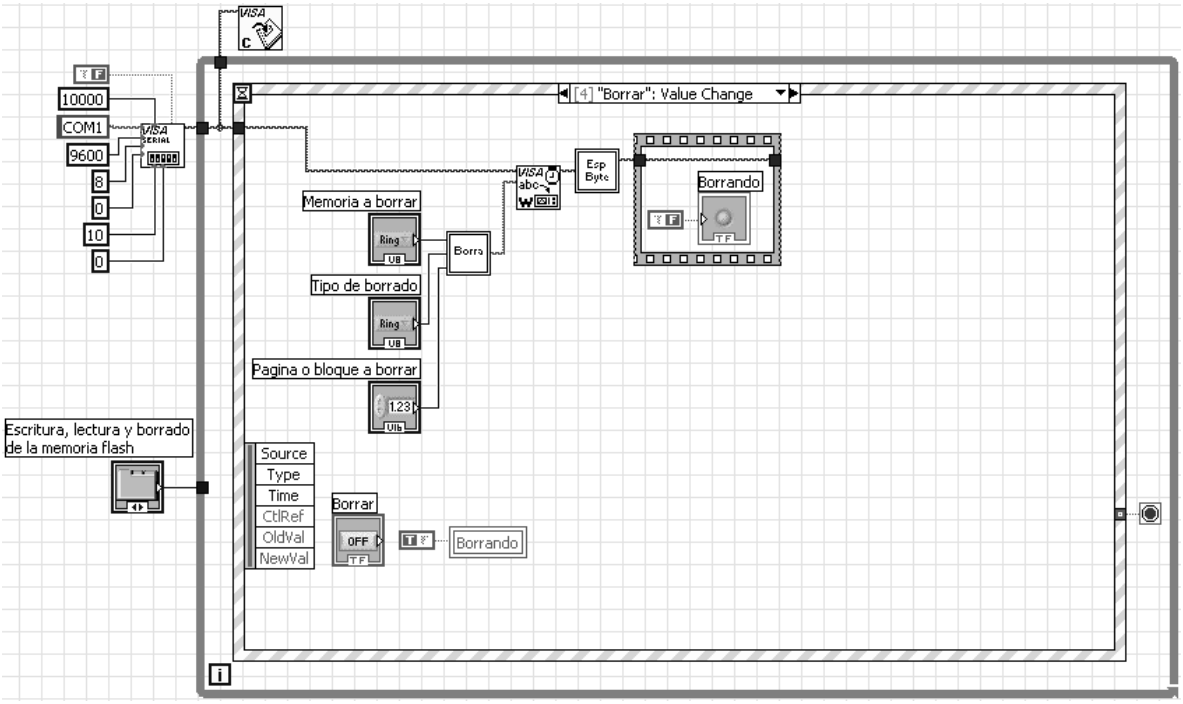
SubVI "LEE 2". Este subVI envía al SAB80C166 comandos de lectura y recibe los bytes leídos de la memoria flash cuando se requiere más de un comando de lectura.



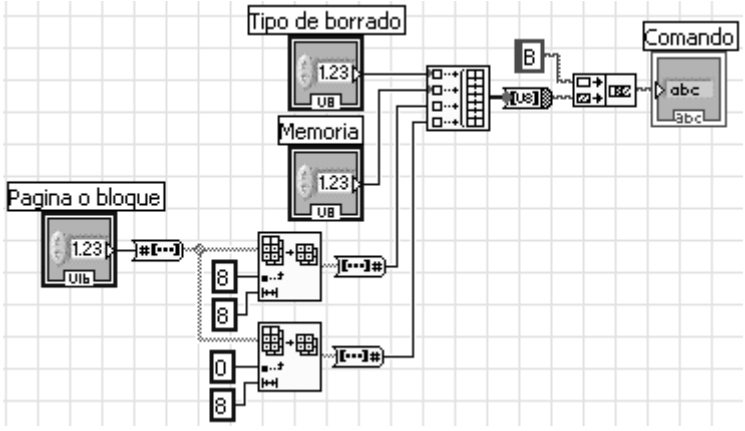
SubVI "CMD L2". Este subVI crea un comando de lectura de la memoria flash cuando se requiere más de un comando de lectura para completar la operación.



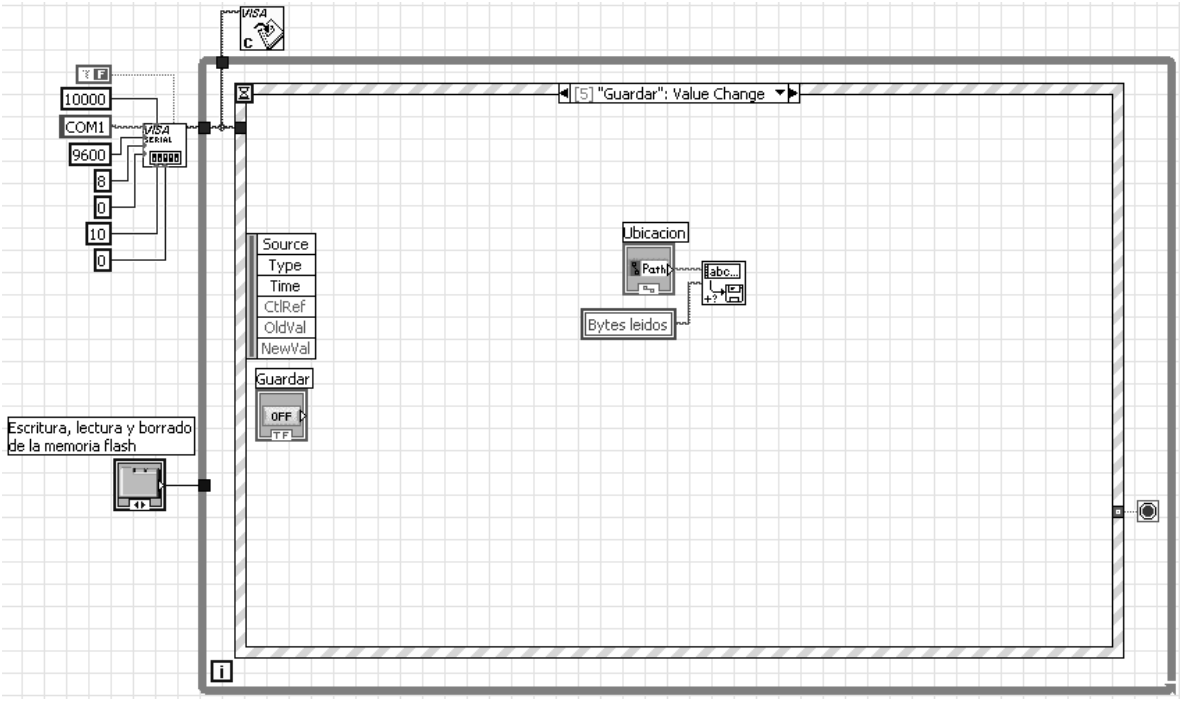
Evento "Borrar". Este evento envía al SAB80C166 un comando de borrado de la memoria flash.



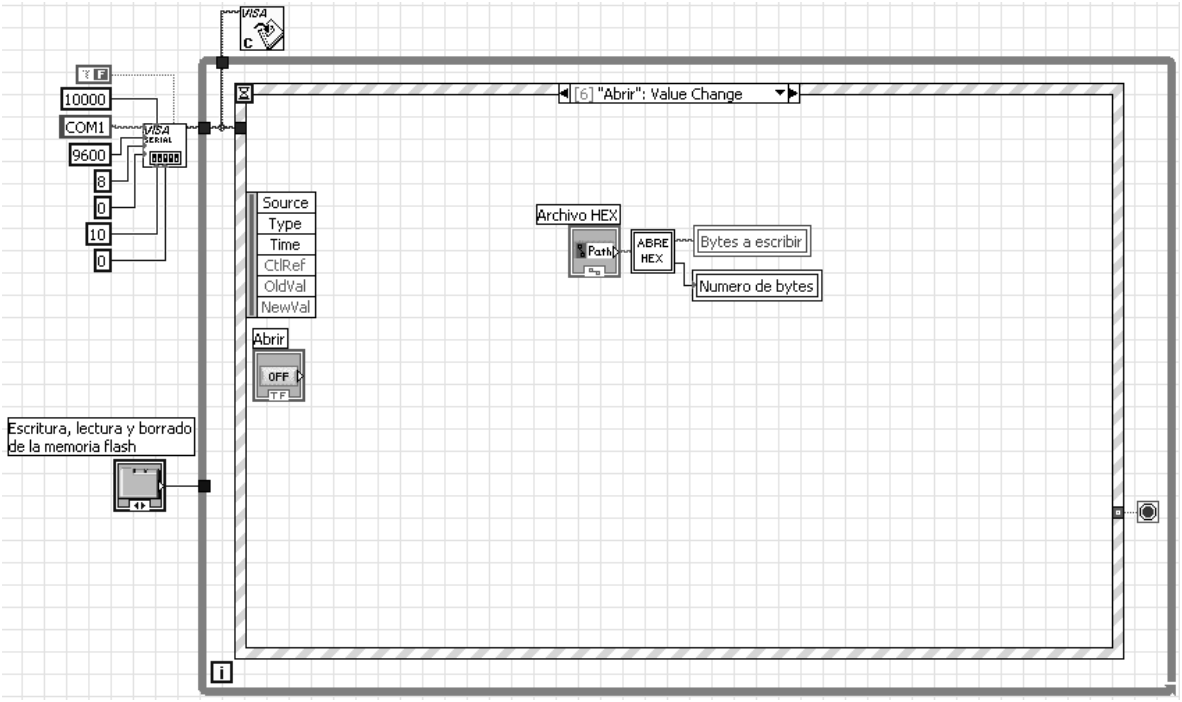
SubVI "Borra". Este subVI crea un comando de borrado de la memoria flash.



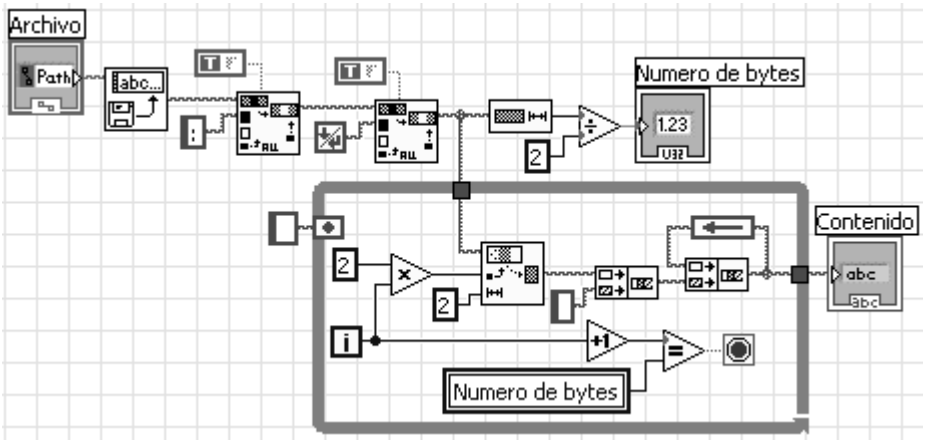
Evento "Guardar". Este evento guarda los bytes leídos de la memoria flash en un archivo HEX.



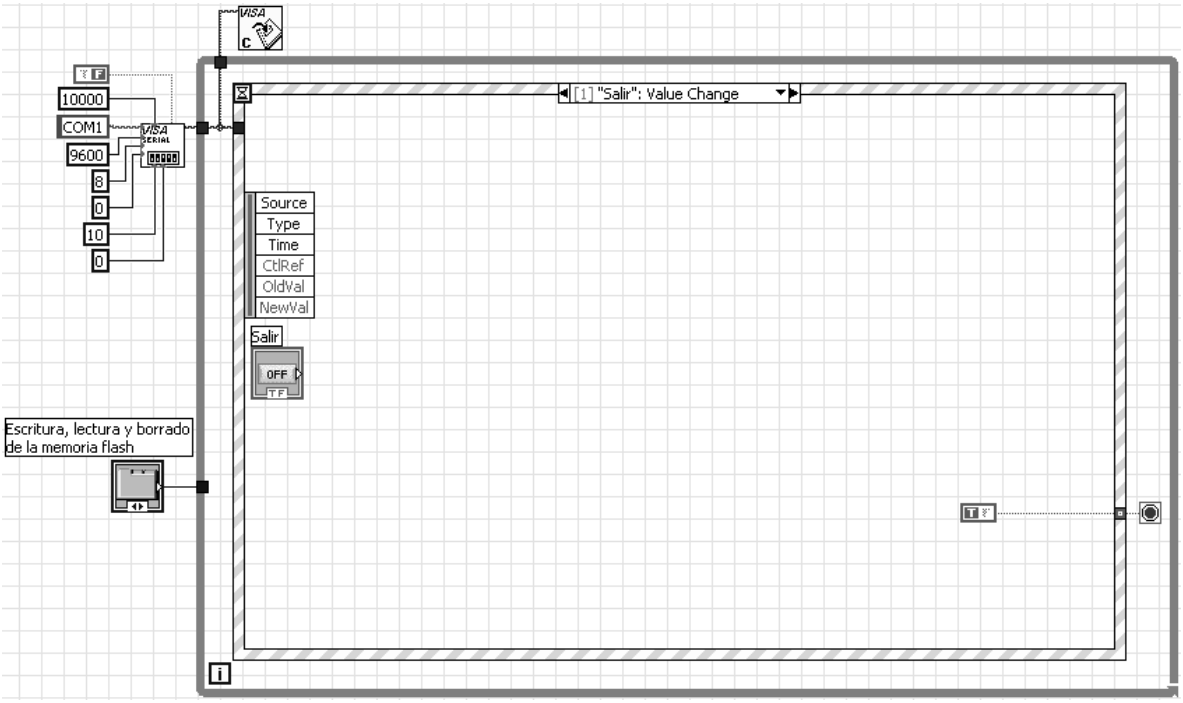
Evento "Abrir". Este evento abre un archivo Intel HEX para escribirlo en la memoria flash.



SubVI "ABRE HEX". Este subVI abre un archivo en formato Intel HEX, separa cada byte con un espacio y despliega su contenido en la interfaz de LabVIEW.



Evento "Salir". Este evento finaliza la aplicación de LabVIEW.



Apéndice F. Software para leer el convertidor A/D 1-Wire

F.1 Software en el SAB80C166

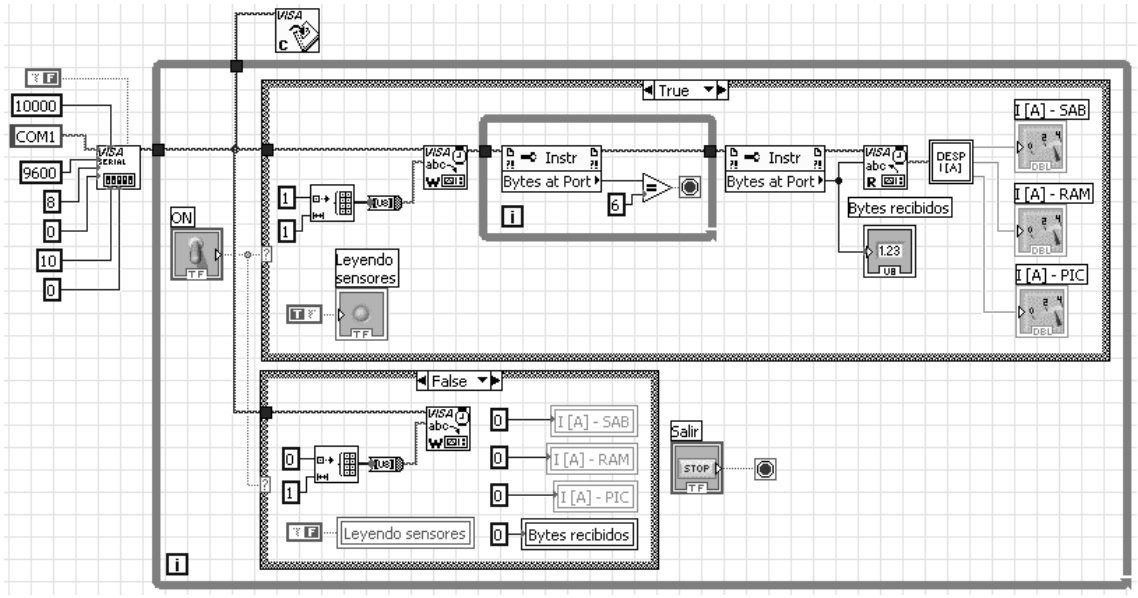
```
#include <stdio.h>
#include <c166.h>
#include <reg166.h>

/* P3.0 es DQ */

void iniSer0 (void) { /* Configura el canal serial 0 */
_bfld(DP3,0x0C00,0x0400); /* P3.11 = RX0, P3.10 = TX0 */
_putbit(1,P3,10);
S0BG=0x003F; /* 9600 bauds */
S0TIC=0x0000; /* S0TIR, S0TIE, ILVL, GLVL */
S0RIC=0x0000; /* S0RIR, S0RIE, ILVL, GLVL */
S0EIC=0x0000; /* S0EIR, S0EIE, ILVL, GLVL */
S0CON=0x8011; /* Rx habilitado, 8 bits, 1 bit de paro, BG habilitado */
}

void txByte0 (unsigned int c) { /* Envia 1 byte por el canal serial 0 */
S0TBUF=c;
while(!S0TIR);
S0TIR=0;
}

unsigned int rxByte0 (void) { /* Recibe 1 byte por el canal serial 0 */
unsigned int c;
while(!S0RIR);
c=S0RBUF;
S0RIR=0;
return c;
}

void retC (unsigned int a) { /* Retardo de tiempo de hasta 26 ms */
T0=a;                        /* a=65535-(tiempo/0.4us) */
T0R=1;
while(!T0IR);
T0R=0;
T0IR=0;
}

void retL (unsigned int a) { /* Retardo de tiempo de hasta 3.36 s */
T1=a;                       /* a=65535-(tiempo/51.2us) */
T1R=1;
while(!T1IR);
T1R=0;
T1IR=0;
}

void inicializar (void) { /* Inicializa el convertidor A/D 1-Wire */
_putbit(1,DP3,0); /* P3.0 es salida */
_putbit(0,P3,0);
retC(64328); /* 485us */
_putbit(0,DP3,0); /* P3.0 es entrada */
retC(65378); /* 65us */
/* En DQ debe de haber cero */
retC(64490); /* 420us */
}

void enviaUno (void) { /* Envia el bit 1 por el bus 1-Wire */
retC(65515); /* 10us */
_putbit(1,DP3,0); /* P3.0 es salida */
_putbit(0,P3,0);
retC(65515); /* 10us */
_putbit(0,DP3,0); /* P3.0 es entrada */
retC(65415); /* 50us */
```

```
}
void enviaCero (void) { /* Envia el bit 0 por el bus 1-Wire */
retC(65515); /* 10us */
_putbit(1,DP3,0); /* P3.0 es salida */
_putbit(0,P3,0);
retC(65340); /* 80us */
_putbit(0,DP3,0); /* P3.0 es entrada */
}

void enviaByte (unsigned int dato) { /* Envia 1 byte por el bus 1-Wire */
unsigned int a,bitEnv;
for(a=0;a<8;a++){
bitEnv=_ror(dato,a);
bitEnv=bitEnv&0x0001;
if(bitEnv==0x0001)
enviaUno();
else
enviaCero();
}
}

unsigned int leeByte (void) { /* Lee 1 byte por el bus 1-Wire */
unsigned int a,dato=0x0000,bitC;
for(a=0;a<8;a++){
bitC=_rol(0x0001,a);
retC(65515); /* 10us */
_putbit(1,DP3,0); /* P3.0 es salida */
_putbit(0,P3,0);
retC(65535); /* 2us */
_putbit(0,DP3,0); /* P3.0 es entrada */
retC(65528); /* 5us */
if(_getbit(P3,0))
dato=dato|bitC;
retC(65428); /* 45us */
}
return dato;
}

main () {
unsigned int recib,LSB_B,MSB_B,LSB_C,MSB_C,LSB_D,MSB_D,x,i;
T01CON=0x0700; /* Configura timers con */
T0IR=0; /* desbordamientos T0=26ms y T1=3.36s */
T1IR=0;
iniSer0(); /* Configura el canal serial 0 */
inicializar();
enviaByte(0x00CC); /* Comando Skip Rom */
enviaByte(0x0055); /* Comando Write Memory */
enviaByte(0x001C);
enviaByte(0x0000);
enviaByte(0x0040);
x=leeByte(); /* CRC 16 */
x=leeByte();
x=leeByte(); /* Dato */
inicializar();
enviaByte(0x00CC); /* Comando Skip Rom */
enviaByte(0x0055); /* Comando Write Memory */
enviaByte(0x000A);
enviaByte(0x0000);
for (i=0;i<3;i++) {
enviaByte(0x0008); /* Byte de control */
x=leeByte(); /* CRC 16 */
x=leeByte();
x=leeByte(); /* Dato */
enviaByte(0x0001); /* Byte de estado */
x=leeByte(); /* CRC 16 */
x=leeByte();
x=leeByte(); /* Dato */
}
while (1) {
```

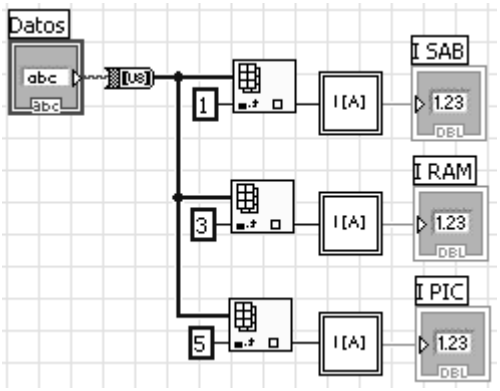
```
recib=rxByte0();
switch (recib) {
case 0:
_nop();
break;
case 1:
inicializar();
enviaByte(0x00CC); /* Comando Skip Rom */
enviaByte(0x003C); /* Comando Convert */
enviaByte(0x000E); /* Input select mask */
enviaByte(0x0000); /* Read-out control (0x0054) */
x=leeByte(); /* CRC 16 */
x=leeByte();
retC(58035); /* 3 ms */
inicializar();
enviaByte(0x00CC); /* Comando Skip Rom */
enviaByte(0x00AA); /* Comando Read Memory */
enviaByte(0x0002);
enviaByte(0x0000);
LSB_B=leeByte(); /* Se leen los resultados */
MSB_B=leeByte(); /* de la conversion A/D */
LSB_C=leeByte();
MSB_C=leeByte();
LSB_D=leeByte();
MSB_D=leeByte();
x=leeByte(); /* CRC 16 */
x=leeByte();
txByte0(LSB_B); /* Se envia a la PC los */
txByte0(MSB_B); /* resultados de la conversion A/D */
txByte0(LSB_C);
txByte0(MSB_C);
txByte0(LSB_D);
txByte0(MSB_D);
break;
}
}
}
```

F.2 Software en LabVIEW

Este software despliega la corriente eléctrica en Amperes consumida por el SAB80C166, la memoria RAM y el PIC16F876A.



SubVI "DESP I [A]". Este subVI despliega la corriente eléctrica consumida en Amperes a partir de los bytes recibidos por el puerto serial de la PC.



SubVI "I [A]". Este subVI calcula la corriente eléctrica consumida en Amperes a partir de los bytes recibidos por el puerto serial de la PC.