ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"Datalogger compacto, con tiempo real, para almacenaje de grandes cantidades de información provenientes de sensores externos en una memoria USB, con capacidad de comunicación serial. Fuente de energía: 4 pilas recargables AA"

TESINA DE SEMINARIO

Previa la obtención del Título de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Presentado por:

Cristhian Xavier Cercado Suárez
Victor Hugo Touriz Plua

GUAYAQUIL – ECUADOR
AÑO 2010

AGRADECIMIENTO

A Dios.

A la familia.

A todas las personas que contribuyeron en el desarrollo de este trabajo.

A todos quienes apuestan por el desarrollo tecnológico en Ecuador.

DEDICATORIA

A Dios que siempre nos ha acompañado, siendo su amor la fuente de energía para alcanzar nuestras metas.

A nuestra familia, por su comprensión y apoyo incondicional, quienes siempre nos inculcaron perseverancia con valores éticos, permitiéndonos iniciar nuestra vida profesional, y a nuestros profesores y amigos, con quienes hemos compartido el reto de culminar la educación superior.

TRIBUNAL DE SUSTENTACIÓN

Ing. Carlos Valdivieso

Profesor de Seminario de Graduación

Ing. Hugo Villavicencio V.

Delegado del Decano

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta tesina, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL".

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Cristhian Xavier Cercado Suárez

Victor Hugo Touriz Plua

RESUMEN

El principal objetivo es mejorar la capacidad de almacenamiento de un proceso tecnológico, utilizando un MEMORY STICK DATALOGGER de PARALLAX, controlado por PIC 18F4431 y una herramienta de software como MIKRO BASIC PRO for PIC de MikroElectrónica. El proyecto que se describe a continuación trata del diseño de un datalogger con tiempo real que permita recoger datos de diversos sensores para almacenarlos en una memoria USB.

El PIC controla la comunicación serial con los diversos sensores externos y la comunicación serial con el datalogger, para la comunicación con los sensores externos se crearon comandos de validación, mientras que para la comunicación con el datalogger y la memoria USB se utilizaron comandos del FIRMWARE VINCULUM, que permite el control de archivos dentro de la memoria USB.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

El objetivo de este proyecto es diseñar e implementar un datalogger compacto con tiempo real capaz de almacenar en una memoria USB datos de diversos sensores a través de comunicación serial. El MEMORY STICK DATALOGGER (MSD) que se utiliza es de PARALLAX el cual es controlado a través de un PIC 18F4431. La comunicación serial del PIC con el MSD, para almacenar datos en la memoria USB, se logra con la aplicación de comandos de archivos del FIRMWARE VINCULUM del MSD.

En el primer capítulo, se menciona una descripción general del proyecto, las partes y funciones del mismo, aplicaciones en el campo industrial y proyectos similares como el GPS USB travel logger y el Datalogger compacto de YOKOGAWA.

En el segundo capítulo, se da un detalle sobre las herramientas de hardware: el memory stick datalogger, el PIC 18F4431 con sus módulos USART y de interrupción TMR0, el programador PICKIT 2 y la tarjeta de Basic Stamp. Además de las herramientas de software: MIKRO BASIC Pro for PIC, Pbasic y los comandos de control de archivos en la memoria USB.

El tercer capítulo, trata del diseño e implementación del proyecto, empezando con una prueba del datalogger con la tarjeta Basic Stamp, la cual nos da las pautas a seguir para el desarrollo del software que se implementará en el PIC con MIKRO BASIC PRO. Se desarrolló un diagrama de bloques que detalla los elementos de nuestro proyecto, el diagrama de flojo del controlador y las funciones detalladas de la comunicación con los sensores y el memory stick datalogger.

En el cuarto y último capítulo, se muestran el esquema y simulación en PROTEUS, también se muestran las pruebas realizadas y conexiones con algunos sensores. Como no existía en PROTEUS una herramienta que simule el memory stick datalogger, se utilizó la herramienta COMPIM y el VIRTUAL TERMINAL para comprobar la comunicación serial con este dispositivo.

CAPÍTULO 1

1. DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. Antecedentes

Desde siempre todo proceso tecnológico de calidad debe ser evaluado y mejorado según las aspiraciones del sector productivo, esto se consigue con el análisis de la información recogida en un ciclo del proceso, lo que requeriría tener considerables espacios físicos para almacenar la información del mismo, para lo cual se requiere un conjunto de herramientas que permitan la observación en tiempo real y en un intervalo seleccionable de tiempo.

Mejorar la capacidad de almacenamiento de información de un proceso significa reducir su espacio físico y permitir una consulta ordenada y rápida a través de una interfaz amigable.

Este proyecto tiene como finalidad la recopilación de grandes cantidades de información en una memoria USB, recogida por un grupo de sensores a través

de un DATA LOGGER compacto en tiempo real, mejorando así, la capacidad de información que puede ser almacenada y consultada en el proceso.

1.2. Descripción del Proyecto

Para realizar el proyecto utilizamos un USB memory que va a almacenar la información a través de un MEMORY STICK DATALOGGER (# 27937) de Parallax con capacidad de comunicación serial utilizando una tarjeta BASIC STAMP para pruebas y luego un micro-controlador 18F4431.

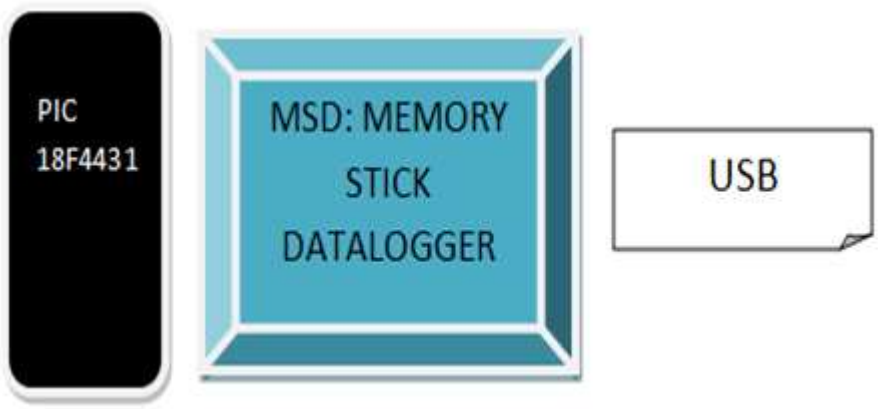


FIGURA 1-1: Descripción del proyecto

El MEMORY STICK DATALOGGER (MSD) será programado en Pbasic si utilizamos la tarjeta BASIC STAMP también de Parallax o Micro BASIC si nos conectamos con el micro-controlador.

El MSD tiene capacidad de comunicación serial simple y SPI, el cual se puede seleccionar por un JUMP ubicado en el MSD, en nuestro proyecto trabajaremos en modo de comunicación serial simple.

1.3. Aplicaciones

La aplicación del MSD es básicamente el almacenamiento de grandes cantidades de información en una memoria USB que puede ser consultada en un computador o a través de un GLCD, esta información almacenada puede provenir de datos analógicos tomados por varios sensores de algún tipo, por ejemplo sensores de temperatura (LM35), los cuales pueden entregar datos en tiempo real, en este caso el MSD es una herramienta que permite almacenar la información de todos los sensores de temperatura de una industria en un intervalo de tiempo, según los intereses del análisis del proceso, de forma que el responsable del proceso puede acceder a la información de forma sencilla y con un entorno amigable.

1.4. Proyectos similares

1.4.1. GPS USB TRAVEL LOGGER

Este interesante proyecto tiene como finalidad la fácil búsqueda de fotos, grabando las rutas de una jornada de viaje, dando la ubicación GPS

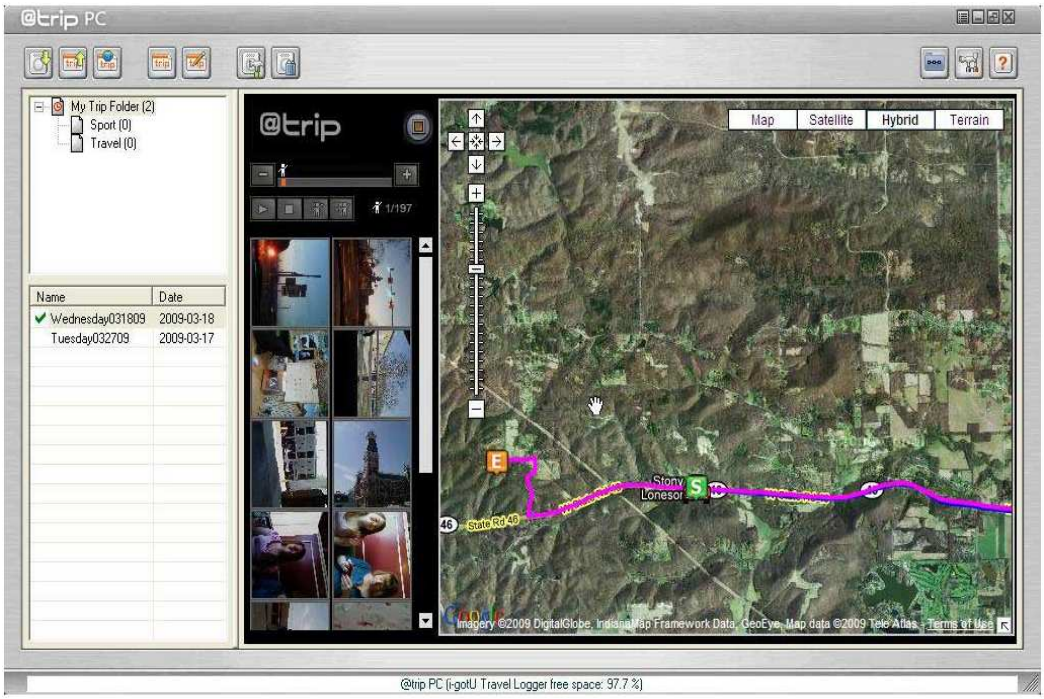


FIGURA 1-2: GPS USB TRAVEL LOGGER

1.4.2. DATALOGGER DE YOKOGAWA

Los nuevos datalogger de YOKOGAWA de la serie MV de DAQSTATION visualizan medidas en tiempo real en un display de cristal líquido de alta resolución TFT. El manejo del equipo resulta sencillo e intuitivo. Se incluye una amplia variedad de funciones de display , así como opciones de almacenamiento para una mayor flexibilidad mediante tarjetas de memoria PCMCIA ATA (hasta 160 MB) y discos Zip (100MB), además de disquetera de 3.5".

NUEVO Data Logger compacto MV100

FIGURA 1-3: Datalogger de YOKOGAWA

CAPÍTULO 2

2. FUNDAMENTO TEÓRICO

2.1. Requerimientos para aplicación del Proyecto

El desarrollo del proyecto se lo puede dividir en Software y Hardware. El software del Kit de BASIC STAMP de PARALLAX se programa en PBASIC a través del BASIC STAMP EDITOR V2.4.2.

El software del PIC 18F4431 se programa en MIKROBASIC PRO FOR PIC de MIKROELECTRONICA, en este caso se utilizó también el programa PICKIT2 V2.50, que permite quemar un PIC desde un puerto USB de la PC.

FIGURA 2-1: Requerimientos del proyecto

El DATALOGGER es un dispositivo electrónico que permite almacenar en tiempo real información proveniente de un conjunto de sensores, el circuito integrado VINCULUM IC/FIRMWARE, es el encargado de manejar los comandos del sistema de archivos, que puede ser consultado por computadora, a través de comunicación serial simple.

El PIC es otra alternativa para controlar el MSD, el PIC 18F4431 tiene 40 pines, tipo FLASH, tecnología NANO Watt y una rápida conversión ADC.

2.2. Herramientas de software

2.2.1. BASIC STAMP

Existen algunas versiones de este sencillo lenguaje de programación de PARALLAX que brinda total sencillez a sus usuarios al momento de programar alguno de sus productos, bs1, bs2 y algunas versiones del bs2

Se caracteriza por sus archivos de extensión *.BS2, que son los que se cargan directamente en el HARDWARE de PARALLAX.

Las funciones más importantes en la comunicación serial son SERIN y SEROUT.

SERIN *Rpin {\Fpin}, Baudmode, {Plabel,} {Timeout, Tlabel,} [InputData]*

SEROUT *Tpin {\Fpin}, Baudmode, {Pace,} {Timeout, Tlabel,} [OutputData]*

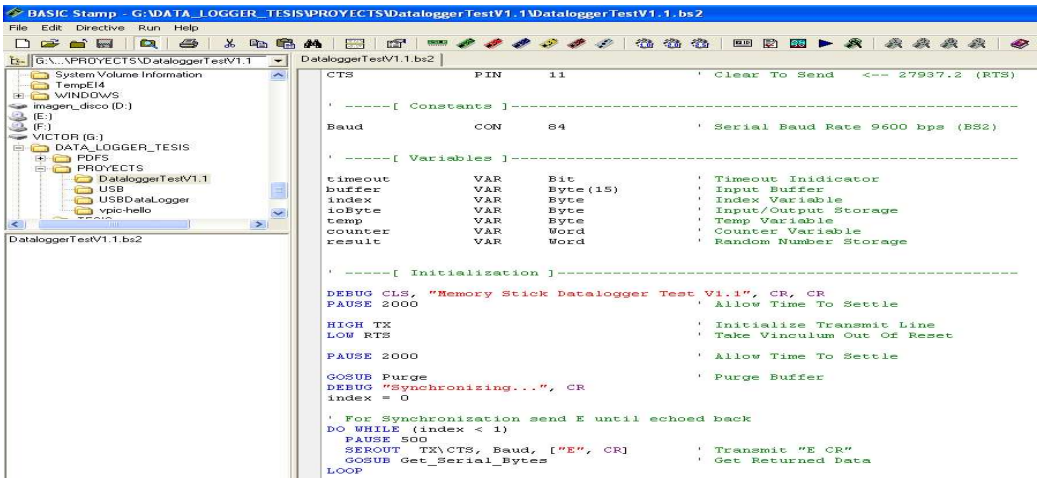


FIGURA 2-2: Entorno de BASIC STAMP EDITOR

2.2.2. MIKRO BASIC PRO for PIC

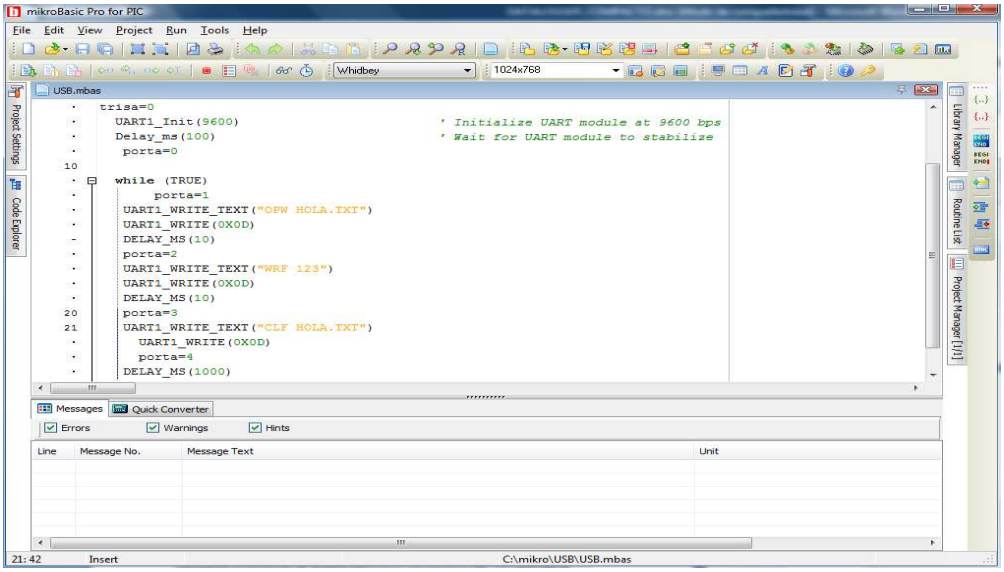


FIGURA 2-3: Entorno de MIKRO BASIC PRO

El ya conocido BASIC PRO, perteneciente a MIKROELECTRONICA, muy formal y estructurado con un entorno de trabajo más elaborado que el de BASIC STAMP, en este lenguaje podemos destacar el uso de la librería de comunicación serial UART y SOFT UART para nuestro proyecto.

2.3. Herramientas de hardware

2.3.1. MEMORY STICK DATALOGGER (MSD)

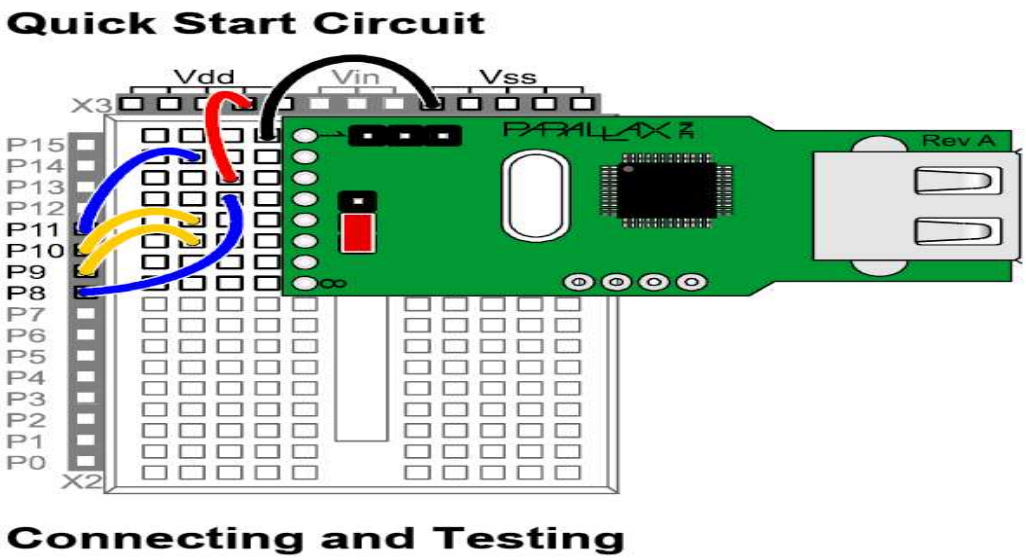


FIGURA 2-4: Diagrama del MEMORY STICK DATALOGGER

Como ya hemos mencionado la función básica del MSD es la del control de escritura y lectura sobre una memoria USB, ahora mencionaremos detalles más técnicos.

Características

- Interfaz serial simple o SPI
- Comandos de seteo/respuesta de formato corto y extendido
- Alimentación de 5vdc con I/O seguras de 3.5/5vdc
- Baja potencia de operación (25mA run / 2mA reposo)
- Fácil actualización de FIRMWARE

Operación

El corazón del MSD es el Vinculum Embedded USB Host Controller IC por FTDI, disponible en PARALLAX, este integrado permite la implementacion de las funcionalidades del controlador USB HOST, sin necesidad de tratar con el protocolo USB de bajo nivel. Tambien controla el FAT File System, lo que facilita tratar con archivos por comunicación serial sin necesidad de tratar con estructuras de archivos o USB DRIVER.

Especificaciones:

Características de alimentación y funcionamiento de MEMORY STICK DATALOGGER

Tabla 2-1: Especificaciones técnicas

Symbol	Quantity	Minimum	Typical	Maximum	Units
Vdd	Supply Voltage	4.75	5.0	5.25	V
-	Storage Temperature	-65°	-	150°	C
-	Operating Temperature	0°	-	70°	C
Ivdd	Supply Current (Running)	-	25	-	mA
Ivdd	Supply Current (Standby)	1	2	2	mA

Definición de pines en modo UART

Tabla 2-2: Configuración modo UART

Pin	Name	Description
1	Vss	Connects to System Ground
2	RTS#	Request To Send (Connects to MCU CTS)
3	Vdd	Connects to +5V (Regulated)
4	RXD	Receive Data (Connects to MCU TXD)
5	TXD	Transmit Data (Connects to MCU RXD)
6	CTS#	Clear To Send (Connects to MCU RTS)
7	NC	No Connection
8	RI#	Ring Indicator (Making this input low resumes from Suspend)

Definición de LEDS

Tabla 2-3: Señalización del funcionamiento

LED Definitions

Operation	LED Behavior
Power On	Green LED and Red LED flash alternately for 2 seconds Repeated until monitor connects
USB Disk Initialization	Green LED on, Red LED off
USB Disk Ready	Green LED off, Red LED on
USB Disk Removed	Green LED off, Red LED off
Commands From Monitor Port to USB Disk	Green LED off, Red LED flashes
Commands From Monitor Port with USB Disk Removed	Green LED off, Red LED off

La definición de LEDS nos permite observar el funcionamiento del MEMORY STICK DATALOGGER.

Dimensiones

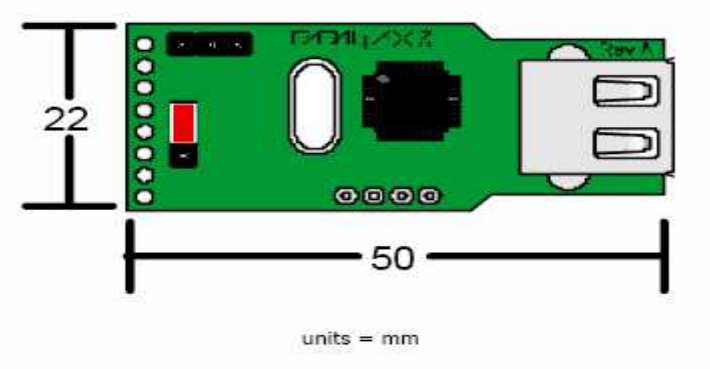


FIGURA 2-5: Dimensiones del Datalogger

VINCULUM VNC1L FIRMWARE-VDAP

La principal función de este firmware es permitir a un dispositivo, basado en VNC1L, la comunicación serial a través de VNC1L-1's UART con una memoria USB.

La interfaz VNC1L-1 UART trabaja en modo de comando o modo de datos en una forma similar a un MODEM.

FIGURA 2-6: VINCULUM FIRMWARE

Comandos para el control de archivos

Tabla 2-4: SHORT/EXTENDED COMANDS

Switching between Shortened and Extended Command sets			
'SCS'<cr>	$10,$0D	Switches to the shortened command set	This will return the prompt '>',$0D to indicate that the device is in shortened command set mode.
'ECS'<cr>	$11,$0D	Switches to the extended command set	This will return the prompt 'D:\>',$0D to indicate that the device is in extended command set mode.
'E'<cr>	'E'<cr>	Echo	This will return 'E',$0D for synchronisation purposes
'e'<cr>	'e'<cr>	Echo	This will return 'e',$0D for synchronisation purposes
'IPA'<cr>	$90,$0D	Input numbers in ASCII	<prompt>$0D
'IPH'<cr>	$91,$0D	Input numbers in HEX	<prompt>$0D

Tabla 2-5: Comandos de operaciones con archivos

File operations			
'RD'<sp> <name><cr>	$04,$20,<name> $0D	Read file <name>	This will send back the entire file in binary to the monitor. The size should first be found by using the 'DIR' <sp> <name> <cr> command so that the expected number of bytes is known. <prompt>$0D
'RDF'<sp> <size in hex(4 bytes MSB first) ><cr>	$0B,$20,size in hex(4 bytes) , $0D	Reads the data of <size in hex(4 bytes) > from the current open file.	This will send back the requested amount of data to the monitor. <prompt>$0D
'DLF'<sp> <name><cr>	$07,$20,<name> $0D	Delete file <name>	This will delete the file from the current directory and free up the FAT sectors. <prompt>$0D
'WRF'<sp> <size in hex(4 bytes MSB first) ><cr> <data bytes of size><cr>	$08,$20,size in hex(4 bytes) , $0D $data,$0D	Writes the data of <size in hex(4 bytes) > to the end of the current open file.	<prompt>$0D
'OPW'<sp> <name><cr>	$09,$20, <name>,$0D	Opens a file for writing to with 'WRF'	<prompt>$0D
'OPR'<sp> <name><cr>	$0E,$20, <name>,$0D	Opens a file for reading to with 'RDF'	<prompt>$0D
'CLF'<sp> <name><cr>	$0A,$20, <name>,$0D	Closes a file for writing.	<prompt>$0D
'REN'<sp> <orig name> <sp> <new name><cr>	$0C,$20, <orig name>,$20, <new name> <cr>	Rename a file or directory	<prompt>$0D

Los comandos de comunicación ya vienen implementados en el MEMORY STICK DATALOGGER, se puede utilizar los comandos en sus dos formatos corto o extendido para realizar operaciones con los archivos.

Tabla 2-6: Comandos de operación con la memoria USB

Responses to indicate if disk is online			
<cr>	$0D	Check if online	This will return the appropriate prompt or 'no disk' message for the current command set.

Este comando permite verificar la presencia de una memoria USB.

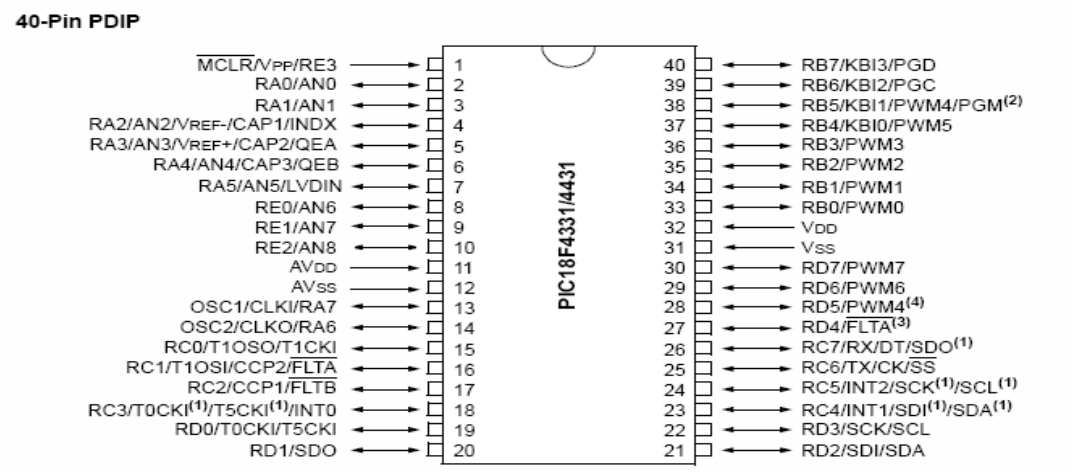
2.3.2. PIC 18F4431

2.3.2.1. Características

- Convertidor analógico/digital de 10 bits, alta velocidad.
- Estructura de oscilador flexible.
- Modos de ahorro de energía.
- Voltaje en cualquier pin con respecto a VSS (excepto MCLR, VDD, RA4): 0.3V a 0.3V+VDD
- Voltaje en VDD con respecto a VSS: 0.3V a 7.5V
- Voltaje en el MCLR con respecto a VSS: 0V a 13.25V
- Corriente máxima en VDD: 250mA

2.3.2.2. Diagrama de pines

FIGURA 2-7: Diagrama de pines del PIC 18F4431



2.3.3. BASIC STAMP HOMEWORK BOARD

En la tarjeta de BASIC STAMP, procederemos a probar las funciones del datalogger en PBASIC, esta tarjeta se comunica por puerto serial a la PC, donde está instalada la aplicación del BASIC STAMP EDITOR.

FIGURA 2-8: Módulo de Basic Stamp 2

2.3.3.1. Partes de la tarjeta BASIC STAMP

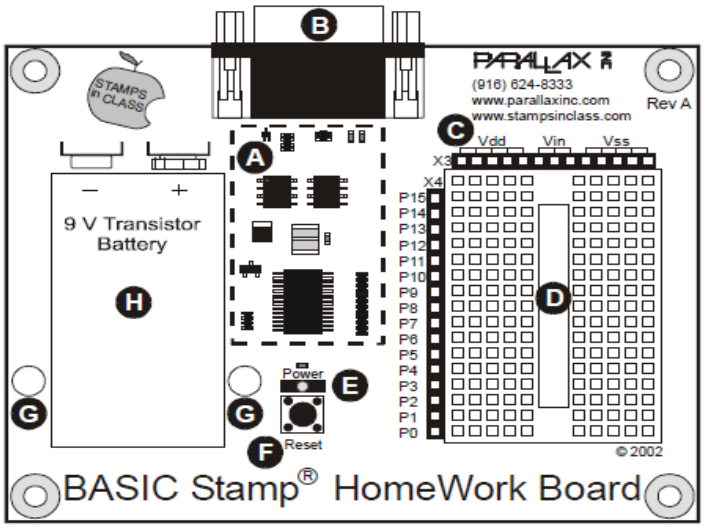


FIGURA 2-9: tarjeta Basic Stamp

- A) Módulo de BASIC STAMP 2

- B) Puerto DB-9 hembra
- C) Conexiones de alimentación
- D) Proto Board
- E)Botón de encendido
- F) Botón de reset
- G) Sujetador de baterías
- H) Batería de 9V

2.3.3.2. Características

Tabla 2-3: Especificaciones de la tarjeta BASIC STAMP

Microcontroller:	PIC16C57 surface mount
Speed:	20 MHz / ~4,000 instructions per second
EEPROM:	2K bytes (program and data)
Program Length:	500 lines of PBASIC
RAM (variables):	32 bytes (6 for I/Os and 26 for variables)
Input / Outputs:	16 (up to 17 RS-232 communication ports)
Source / Sink Current:	50 mA / 50 mA
Serial Communication:	300-50K baud I/O
Current Requirements:	7 mA running, 50 uA in sleep
PC Interface:	Serial port
Power Supply:	5 V through an LM2936 regulator from a 9V transistor battery
Environment:	32 to 158° F (0 to 70° C), 70% non-condensing humidity
HomeWork Board Size:	3" x 4"
Project Area:	Built-in breadboard on StampLab
Microcontroller:	PIC16C57 surface mount
Speed:	20 MHz / about 4,000 instructions per second
EEPROM:	2K bytes (program and data)
Program Length:	~500 to 600 lines of PBASIC

La tarjeta de pruebas de BASIC STAMP nos ayudó a dar los primeros pasos en la programación de MIKRO BASIC PRO, ya que nos permitió entender el algoritmo de escritura/lectura del MEMORY STICK DATALOGGER

2.3.4. PIC KIT 2

El PICkit 2 es un programador fabricado por Microchip© para programar toda su línea de micro-controladores PIC's® desde los PIC10, PIC12, PIC14, PIC16, PIC18, PIC24, dsPIC30 y dsPIC33 (todos los micro-controladores con memoria Flash sin excepción). El programador fue diseñado para programar los microcontroladores en circuito (ICSP) lo que significa que puede programar los microcontroladores montados directamente en tu aplicación y/o protoboard sin necesidad de tener que sacarlo y meterlo cada vez que se modifica el programa.

CAPÍTULO 3

3. DISEÑO E IMPLEMENTACIÓN DEL PROYECTO

A continuación se detalla el proceso del diseño y la implementación del proyecto. Nuestra prueba inicial nos ayudó a entender el funcionamiento de las funciones de comunicación entre el controlador y el memory stick datalogger.

Ya con las bases del proyecto diseñamos el diagrama de bloques, donde el PIC 18F4431 es el controlador de la comunicación serial entre el datalogger y los sensores. Se desarrolló el algoritmo del microcontrolaodr, con las funciones principales que se muestran en el diagrama de flojo y la programación principal.

Antes de desarrollar las funciones del programa principal se estableció comandos de comunicación con los diversos sensores.

3.1. Prueba inicial

Como primer paso realizamos pruebas con el MEMORY STICK DATALOGGER utilizamos la tarjeta BASIC STAMP DE PARALLAX, realizamos pruebas de escritura y lectura con un USB MEMORY en PBASIC.

El programa lleva por nombre DATALOGGERTESTV1.1.BS2, y escribe un archivo llamado 'SEEDFILE.TXT' guardado en un archivo 'DATADIR', luego este podra ser leido al conectar la memoria USB a la PC.

3.1.1. Código de prueba en PBASIC

```
'   {$STAMP BS2}
'   {$PBASIC 2.5}

' -----[ I/O Definitions ]-------------------------------------------------
TX            PIN     8             ' Transmit Data   --> 27937.4 (RXD)
RTS           PIN     9             ' Request To Send --> 27937.6 (CTS)
RX            PIN     10            ' Receive Data    <-- 27937.5 (TXD)
CTS           PIN     11            ' Clear To Send   <-- 27937.2 (RTS)

' -----[ Constants ]-------------------------------------------------------
Baud          CON     84            ' Serial Baud Rate 9600 bps (BS2)

' -----[ Variables ]-------------------------------------------------------
timeout       VAR     Bit           ' Timeout Inidicator
buffer        VAR     Byte(15)      ' Input Buffer
index         VAR     Byte          ' Index Variable
ioByte        VAR     Byte          ' Input/Output Storage
temp          VAR     Byte          ' Temp Variable
counter       VAR     Word          ' Counter Variable
result        VAR     Word          ' Random Number Storage
```

```
' -----[ Initialization ]----------------------------------------------
DEBUG CLS, "Memory Stick Datalogger Test V1.1", CR, CR
PAUSE 2000                          ' Allow Time To Settle

HIGH TX                             ' Initialize Transmit Line
LOW RTS                             ' Take Vinculum Out Of Reset

PAUSE 2000                          ' Allow Time To Settle

GOSUB Purge                         ' Purge Buffer
DEBUG "Synchronizing...", CR
index = 0

' For Synchronization send E until echoed back
DO WHILE (index < 1)
  PAUSE 500
  SEROUT  TX\CTS, Baud, ["E", CR]       ' Transmit "E CR"
  GOSUB Get_Serial_Bytes                ' Get Returned Data
LOOP
GOSUB Purge                         ' Purge Buffer

' Send e to complete synchronization
PAUSE 500
SEROUT  TX\CTS, Baud, ["e", CR]         ' Transmit "e CR"
GOSUB Get_Serial_Bytes

' Send CR to see if drive is present
PAUSE 500
GOSUB Purge                         ' Purge Buffer
DEBUG "Checking for USB Flash drive...", CR
SEROUT TX\CTS, Baud, [CR]               ' Send Carriage Return
GOSUB Get_Serial_Bytes                ' Wait for D:\>

IF (buffer(0) = "N") THEN
  DEBUG "No drive present!"
  STOP
ELSE
  DEBUG "Drive D: Ready...", CR
ENDIF
GOSUB Purge                         ' Purge Buffer

' -----[ Program Code ]-------------------------------------------------

Main:
```

```
result = 11000                  ' Initial "seed" Value For Random
RANDOM result                   ' Generate Random Number
DEBUG "Random Number (Seed) = ", DEC result, CR

' Delete SEEDFILE.TXT if one exists, ignore error if no file exists
PAUSE 500
SEROUT TX\CTS, Baud, ["DLF seedfile.txt", CR]
GOSUB Get_Serial_Bytes
GOSUB Purge                     ' Purge Buffer

' Open SEEDFILE.TXT for output (write)
PAUSE 500
SEROUT TX\CTS, Baud, ["OPW seedfile.txt", CR]
GOSUB Get_Serial_Bytes

' Write random seed value to SEEDFILE.TXT
PAUSE 250
SEROUT TX\CTS, Baud, ["WRF ", $00, $00, $00, $05, CR, DEC5 result, CR]
GOSUB Get_Serial_Bytes

' Close SEEDFILE.TXT
PAUSE 250
SEROUT TX\CTS, Baud, ["CLF seedfile.txt", CR]
GOSUB Get_Serial_Bytes
GOSUB Purge                     ' Purge Buffer

' Create folder DATADIR and ignore error if folder exists
PAUSE 200
SEROUT TX\CTS, Baud, ["MKD datadir", CR]
GOSUB Get_Serial_Bytes
GOSUB Purge                     ' Purge Buffer

counter = 0                     ' Loop Counter

DO
' Open SEEDFILE.TXT for input (read)
PAUSE 200
SEROUT TX\CTS, Baud, ["OPR seedfile.txt", CR]
GOSUB Get_Serial_Bytes

' Read random seed into variable
PAUSE 200
SEROUT TX\CTS, Baud, ["RDF ", $00, $00, $00, $05, CR]
SERIN RX\RTS, Baud, 800, No_Data2, [DEC5 result]
```

```
DEBUG "Seed:", DEC result, CR          ' Display stored seed value

' Close SEEDFILE.TXT
PAUSE 120
SEROUT TX\CTS, Baud, ["CLF seedfile.txt", CR]
GOSUB Get_Serial_Bytes

' Change to DATA folder
PAUSE 120
SEROUT TX\CTS, Baud, ["CD datadir", CR]
GOSUB Get_Serial_Bytes

' Open DATAFILE.TXT for output (write)
PAUSE 120
SEROUT TX\CTS, Baud, ["OPW datafile.txt", CR]
GOSUB Get_Serial_Bytes

counter = counter + 1                  ' Increment Counter

' Write loop counter value and data separator to DATAFILE.TXT
PAUSE 120
DEBUG "Writing data separator now... ", CR
SEROUT TX\CTS, Baud, ["WRF ", $00, $00, $00, $09, CR,
                      DEC5 counter, "**", CR, LF, CR]
GOSUB Get_Serial_Bytes

' Write 5 random numbers to DATAFILE.TXT
PAUSE 120
DEBUG "Writing 5 random numbers now... ", CR
FOR temp = 1 TO 5
  RANDOM result
  DEBUG "Random Number = ", DEC result, CR
  SEROUT TX\CTS, Baud, ["WRF ", $00, $00, $00, $07, CR,
                        DEC5 result, CR, LF, CR]
  GOSUB Get_Serial_Bytes
  PAUSE 120
NEXT

' Close DATAFILE.TXT
PAUSE 120
SEROUT TX\CTS, Baud, ["CLF datafile.txt", CR]
GOSUB Get_Serial_Bytes

' Change to root folder
```

```
  PAUSE 120
  SEROUT TX\CTS, Baud, ["CD ..", CR]
  GOSUB Get_Serial_Bytes

  ' Delete SEEDFILE.TXT
  PAUSE 120
  SEROUT TX\CTS, Baud, ["DLF seedfile.txt", CR]
  GOSUB Get_Serial_Bytes

  ' Open SEEDFILE.TXT for output (write)
  PAUSE 120
  SEROUT TX\CTS, Baud, ["OPW seedfile.txt", CR]
  GOSUB Get_Serial_Bytes

  ' Write new random seed value
  PAUSE 120
  SEROUT TX\CTS, Baud, ["WRF ", $00, $00, $00, $05, CR, DEC5 result, CR]
  GOSUB Get_Serial_Bytes

  ' Close SEEDFILE.TXT
  PAUSE 120
  SEROUT TX\CTS, Baud, ["CLF seedfile.txt", CR]
  GOSUB Get_Serial_Bytes

  ' USB Suspend Mode (Power Saving Mode)
  PAUSE 120
  DEBUG "Putting drive to sleep...", CR
  SEROUT TX\CTS, Baud, ["SUD", CR]
  GOSUB Get_Serial_Bytes

  PAUSE 10000

  ' Wake Drive (Full Power)
  PAUSE 120
  DEBUG "Waking drive now", CR
  SEROUT TX\CTS, Baud, ["WKD", CR]
  GOSUB Get_Serial_Bytes
LOOP

' -----[ Subroutines ]------------------------------------------------

Get_Serial_Bytes:
  timeout = 1                    ' Set Timeout Indicator Flag
  index = 0                      ' Initialize Index
```

```
  DO WHILE (timeout > 0)              ' While Timeout Has Not Occurred
    ioByte = 0                      ' Clear Temporary Storage
    SERIN RX\RTS, Baud, 100, No_Data, [ioByte]
    buffer(index) = ioByte            ' Save Byte Received To Array
    index = index + 1               ' Increment Index
    IF (index > 14) THEN              ' Check For Overflow
      index = 14                    ' Prevent Overflow
    ENDIF
  LOOP
RETURN

Purge:
  timeout = 1                       ' Set Timeout Indicator Flag
  DO WHILE (timeout > 0)              ' While Timeout Has Not Occurred
    PAUSE 50
    SERIN RX\RTS, Baud, 500, No_Data, [ioByte]
  LOOP
  RETURN

No_Data:
  timeout = 0                       ' Timeout, Clear Flag
RETURN

No_Data2:
  DEBUG CR, "Error reading the seed file -- Halting execution!", CR
  PAUSE 1000
  GOSUB Purge                       ' Purge Buffer
  STOP
RETURN
```

3.2. Descripción del proyecto final

3.2.1. Diagrama de bloques

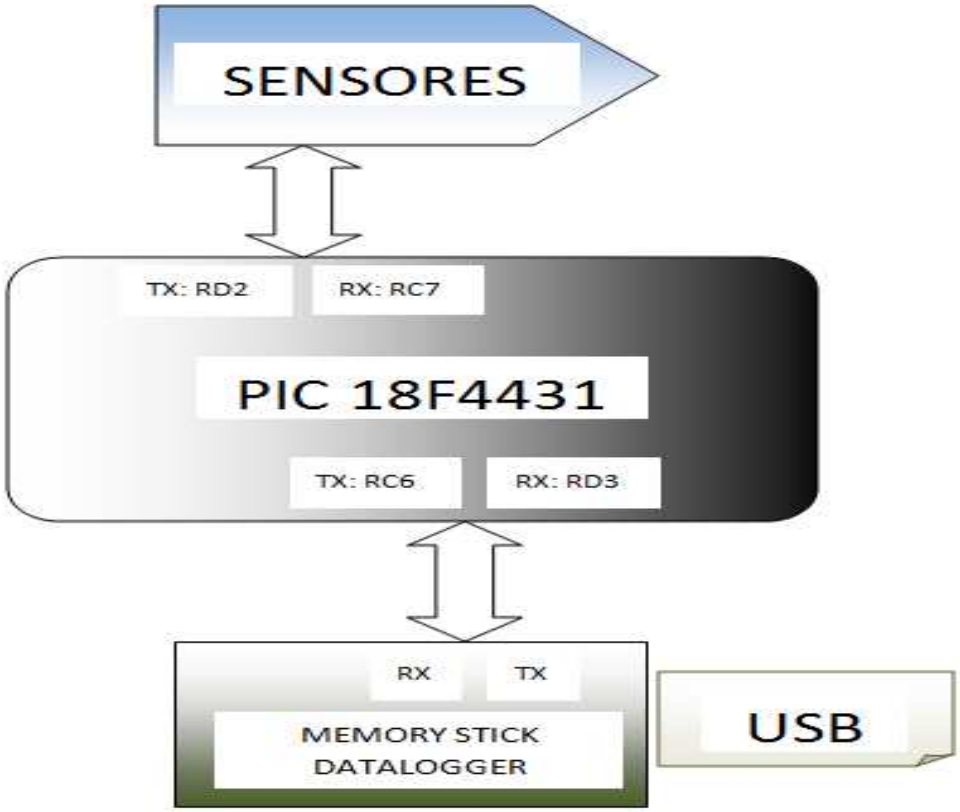


FIGURA 3-1: Diagrama de bloques del proyecto

Podemos observar que el PIC 18F4431 controla la comunicación entre los sensores externos y el MEMORY STICK DATALOGGER utilizando dos módulos de comunicación serial.

3.3. Algoritmo del microcontrolador

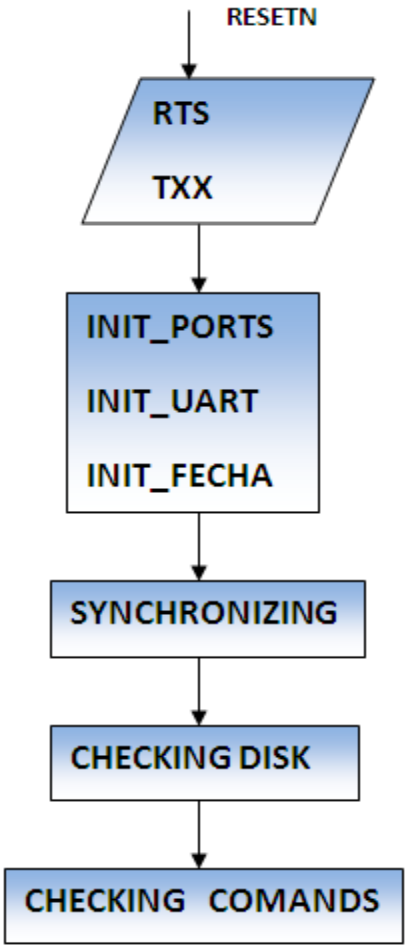


FIGURA 3-2: Algoritmo del controlador

En el algoritmo del controlador se inicializa primero los puertos, luego el módulo de comunicación serial y la fecha del tiempo real del DATALOGGER COMPACTO, La sincronización es importante para la comunicación con la memoria USB.

El bloque de comandos establece la comunicación entre los sensores externos y el MEMORY STICK DATALOGGER.

3.4. Programa principal del microcontrolador

```
program PROYECTO

main:
    'INICIALIZACION
    INIT_PORTS()
    INIT_FECHA()
    SOFT_UART_INIT(PORTD,3,2,9600,0)'RX->3,TX->2
    UART1_INIT(9600)
    STRCPY(LASTFILE,"TMP")
    SOFT_WR("CLR")

    RTS=0   TXX=1
    DELAY_MS(5000)
    RTS=0  PORTA.0=0

    'SINCRONIZANDO
    WHILE CADENA[0]<>"E"
        DELAY_MS(500)
        UART1_WRITE("E")    RTS=1
        UART1_WRITE(0X0D)
        SOFT_RD()
    WEND
    RTS=1

    'CHECKING DISC
    DO
      RTS=1 UART1_WRITE(0X0D)
      SOFT_RD() RTS=1
    LOOP UNTIL CADENA[0]<>"N"
    DELAY_MS(50)

    PORTA.0=0
    ST[0]=0X0D  ST[1]=0
    SOFT_WR("  ")
    SOFT_WR("CC")
    'CHECKING COMMANDS

    WHILE 1
          IF UART1_DATA_READY() THEN
             UART1_READ_TEXT(CADENA,ST,40)
             DELAY_MS(52)
             COMANDOS(CADENA)
             ST[0]=0X0D ST[1]=0
          END IF
          PORTA.0=1
    WEND

end.
```

3.5. Funciones para la comunicación con los sensores

Para la comunicación con los sensores seguimos el estándar de los siguientes comandos:

Tabla 3-1: Comandos de comunicación con los sensores

COMANDOS RECIBIDOS		COMANDOS RESPUESTA		
COMANDOS	FUNCIÓN	COMANDOS		FUNCIÓN
"CN"0X0D	Cambiar nombre	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"MYFILE.TXT"0X0D	Nombre de archivo	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"WR"0X0D	Escribe archivo	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"65535"0X0D	(dato a guardar)	"32535"0X0D"OK"0X0D		Dato guardado
	"NNNNN" CADENA TIPO WORD SOLO NÚMEROS			
"32535"0X0D	(siguiente dato)	"32535"0X0D"OK"0X0D		Dato guardado
"END"0X0D	Fin de envio de datos	"EF"0X0D		Archivo cerrado
"WRI"0X0D	Escribe archivo desde el inicio	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"RD"0X0D	Leer el archivo	"12345"0X0D		Primer dato
"R"0X0D	Leer el siguiente dato	"00002"XD		Siguiente dato
"R"0X0D	Leer el siguiente dato	"EF"0X0D		Si ya no hay datos
"F"0X0D	Fin de lectura	"EF"0X0D		fin de lectura
"RDD"0X0D	Leer con offset	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"3"0X0D	Offset a leer	"00252"XD"OK"0XD	"EF"0XD	Lee el cuarto dato/fin de lectura
"RDC"0X0D	Leer desde el offset	"OK"0X0D	"CF"0X0D	Com correcto/com fallido
"2"0X0D	Offset a leer	"00252"XD"OK"0XD	"EF"0XD	Lee desde el 3 dato/fin de lectura
"R"0X0D	Leer siguiente dato	"00523"XD		Siguiente dato
"F"0X0D	Fin de lectura	"EF"0X0D		fin de lectura

Los comandos son usados en las funciones de lectura y escritura del SOFT_UART, para la comunicación con los sensores.

3.6. Funciones implementadas en el microcontrolador

3.6.1. Inicialización

Primero se especifican las líneas deinicialización de los módulos UART y SOFT_UART, también la fecha y hora del sistema con su módulo de interrupción y los puertos.

```
SUB PROCEDURE INIT_FECHA()
    YEA=2010-1980  MON=2
    DAY=30         HOU=9
    MIN=0          SEC=16/2

    TMR0L  = 0          ' Timer0 initial value
    counter = 0         ' Initialize counter
END SUB

SUB PROCEDURE INIT_PORTS()
      ADRESH=0  ADCON0=0
      ADRESL=0  OSCCON=2
      ANSEL0=0  PWMCON0=0
      TRISB=2   TRISA=0
      PORTD=0   PORTC=0
      TRISD=0   TRISC=0X80
      RTS=1     TXX=0
      ADCON1 = 0x3F       ' Set AN pins to Digital I/O
      INTCON = 0xA0       ' Enable TMR0 interrupt
      T0CON  = 0xC4       ' Set TMR0 in 8bit mode, assign prescaler t
END SUB
```

3.6.2. Verificación de los comandos

Esta función controla la comunicación con los sensores a través de la validación de varios comandos de control

```
SUB PROCEDURE COMANDOS(DIM BYREF CADENA AS STRING[40])
    PORTA.0=0
    IF STRLEN(CADENA)=2 THEN

       PORTA.0=0
         IF STRCMP(CADENA,"CN")=0 THEN  'cambia el nombre
            SOFT_WR("OK")
            CHANGENAME()
         ELSE
             'graba datos desd la ultima posicion
             'por cada dato retorna el dato guardado y "ok"
             IF STRCMP(CADENA,"WR")=0 THEN
                GRABAR()
             ELSE
                 'lee datos desd el inicio(cada vez que recibe un datc
                 'si se quiere seguir leyendo hay que enviar R<enter>
                 'para finalizar F<enter>, si no hay datos recibe EF
                 IF STRCMP(CADENA,"RD")=0 THEN
                    'SOFT_WR("OK")
                   LEER()
                 ELSE
                     IF STRCMP(CADENA,"VW")=0 THEN
                        SOFT_WR("OK")
                        SOFT_WR(LASTFILE)
                     ELSE
                         SOFT_WR("CF")
                     END IF
                 END IF
             END IF
         END IF
    ELSE
```

```
        IF STRLEN(CADENA)=3 THEN
            PORTA.0=0
            IF STRCMP(CADENA,"FEC")=0 THEN
                SOFT_WR("OK")
                OBTENERFECHA()
            ELSE
                IF STRCMP(CADENA,"RDD")=0 THEN
                    SOFT_WR("OK")
                    LEER2()
                ELSE
                    IF STRCMP(CADENA,"RDC")=0 THEN
                        SOFT_WR("OK")
                        LEER3()
                    ELSE
                        IF STRCMP(CADENA,"WRI")=0 THEN
                            GRABARINI()
                        ELSE
                            IF STRCMP(CADENA,"RSV")=0 THEN
                                SOFT_WR("OK")
                                RESERVAR()
                                SOFT_WR("EF")
                            ELSE
                                    SOFT_WR("CF")
                            END IF
                        END IF
                    END IF
                END IF
            ELSE
                SOFT_WR("CF")
            END IF
    END IF
END SUB
```

3.6.3. Comandos de interacción con los sensores externos. Funciones CHANGENAME, GRABAR y LEER.

CHANGENAME: Si el sensor externo envía el comando “CN” , entonces se setea el nombre del archivo del cual vamos a tomar los datos y en el cual guardaremos nuevos datos.

```
SUB PROCEDURE CHANGENAME()
    ST[0]=13  ST[1]=0
    PORTA.0=1 RTS=0
    WHILE 1
        IF UART1_DATA_READY() THEN
            UART1_READ_TEXT(CADENA,ST,40)
            STRCPY(LASTFILE,CADENA)
            GOTO WSALIR
        END IF
    WEND
    WSALIR:
    SOFT_WR("OK")
END SUB
```

GRABAR: Si el sensor envía el comando “WR”, entonces se abre el archivo en modo escritura, y escribe en el archivo hasta que recibe el comando de fin de escritura.

```
SUB PROCEDURE GRABAR()
    DARFECHAHEX()
    PORTA.2=0  RTS=0  I=0

    'ABRIENDO EL ARCHIVO EN MODO ESCRITURA
    OPENW(LON)
    SOFT_WR("OK")
    ST[0]=13  ST[1]=0

    PORTA.2=1
    WHILE 1
        IF UART1_DATA_READY() THEN
           UART1_READ_TEXT(CADENA,ST,40) 'DATOS O FIN DE RECEPCION DE
           IF STRLEN(CADENA)>0 THEN
               IF STRCMP(CADENA,"END")=0 THEN
                   'CERRANDO ARCHIVO
                   CLOSEF()
                   GOTO SALIDA1
               ELSE
                   PORTA.1=1
                   'GUARDANDO EL DATO (SOLO SE ADMITEN WORDS)
                    VARI=STRTOWORD(CADENA)
                    'SOFT_WR(CADENA)
                    WORDTOSTR(VARI,CADENA)
                    STRCAT(CADENA,"-")
                    WRITEN(6,CADENA)
                   SOFT_WR("OK")
                   PORTA.1=0
               END IF
           END IF
           CADENA[0]=0
        END IF
    WEND
    SALIDA1:
    PORTA.2=0
    SOFT_WR("EF")
END SUB
```

GRABARINI: Es otra versión de la función GRABAR que realiza la misma acción pero con OFFSET predeterminado.

```
SUB PROCEDURE GRABARINI()
    DARFECHAHEX()
    PORTA.2=0  RTS=0  I=0
    'ABRIENDO EL ARCHIVO EN MODO ESCRITURA
    OPENW(LON)
    OFFSET(0)
    SOFT_WR("OK")
    ST[0]=13  ST[1]=0

    PORTA.2=1
    WHILE 1
        IF UART1_DATA_READY() THEN
           UART1_READ_TEXT(CADENA,ST,40) 'DATOS O FIN DE RECEPCION DE DATOS
           IF STRCMP(CADENA,"END")=0 THEN
               'CERRANDO ARCHIVO
               CLOSEF()
               GOTO SALIDA1
           ELSE
               PORTA.1=1
               'GUARDANDO EL DATO (SOLO SE ADMITEN WORDS)
                VARI=STRTOWORD(CADENA)
                WORDTOSTR(VARI,CADENA)
                STRCAT(CADENA,"-")
                WRITEN(6,CADENA)
               SOFT_WR("OK")
               PORTA.1=0
           END IF
           CADENA[0]=0
        END IF
    WEND
    SALIDA1:
    PORTA.2=0
    SOFT_WR("EF")
END SUB
```

WRITEN: Escribe los datos por el módulo UART1, ejecuta la acción principal de las funciones GRABAR.

```
SUB FUNCTION WRITEN(DIM LON AS WORD,DIM BYREF CADENA AS STRING[40])
    RTS=0                                                    AS BYTE
    UART1_WRITE_TEXT("WRF ")
    UART1_WRITE(LON>>24) UART1_WRITE(LON>>16)
    UART1_WRITE(LON>>8)  UART1_WRITE(LON)
    RTS=0                UART1_WRITE(13)
    WHILE STRLEN(CADENA)<LON
        strappendpre(" ",CADENA)
    WEND
    CADENA[LON]=0
    UART1_WRITE_TEXT(CADENA)
    IF LON>0 THEN CADENA[LON-1]=0
    ELSE CADENA[0]=0 END IF
    SOFT_WR(CADENA)
    RTS=1   UART1_WRITE(13)
    SOFT_RD()
    IF STRCMP(CADENA,"D:\>")=0 THEN
       RESULT=1
    ELSE
       RESULT=0
    END IF
END SUB
```

LEER: Si se recibe el comando “RD” , entonces los datos previamente grabados son leídos con esta función hasta encontrar el comando de fin de lectura o hasta llegar al final del archivo.

```
SUB PROCEDURE LEER()
        IF OPENR()THEN
          I=0     ST[0]=0
          WHILE 1
              IF LEERN(6,CADENA) =0 THEN
                 GOTO SWEN
              ELSE
                 CADENA[5]=0
                 SOFT_WR(CADENA)
                 ST[0]=0X0D I=1
                 ST[1]=0
              END IF
              WHILE I=1
                    IF UART1_DATA_READY() THEN
                       UART1_READ_TEXT(CADENA,ST,40)
                       IF STRCMP(CADENA,"R")=0 THEN
                          I=0 ST[0]=0
                       END IF
                       IF STRCMP(CADENA,"F")=0 THEN
                          GOTO SWEN
                       END IF
                    END IF
              WEND
          WEND
        SWEN:
        CLOSEF()
        SOFT_WR("EF")
    ELSE
        SOFT_WR("CF")
    END IF
END SUB
```

LEER2 : Es una versión de la función leer pero lo realizan con distintos offset.

```
SUB PROCEDURE LEER2()
    PORTA.2=1
    CADENA[0]=0      RTS=0
    ST[0]=0X0D       ST[1]=0
    WHILE STRLEN(CADENA)=0
          IF UART1_DATA_READY() THEN
             UART1_READ_TEXT(CADENA,ST,40) 'LEYENDO EL OFFSET EN WORD
             LON=STRTOWORD(CADENA)         'CONVIRTIENDO OFFSET EN WORD
             LON=LON*6                     'SIEMPRE HAY 6 DATOS Y POR ESO SE MULT
             IF LON<>0 THEN LON=LON+1 END IF
             IF OPENR() THEN
                I=0
                IF OFFSET(LON) THEN
                   IF LON<>0 THEN LON=LON-1 END IF
                   OFFSET(LON)
                      IF LEERN(6,CADENA) =0 THEN
                         GOTO SWEN
                      ELSE
                          CADENA[5]=0
                          SOFT_WR(CADENA)
                          ST[0]=0X0D  I=1
                          ST[1]=0
                      END IF
                ELSE
                    SOFT_WR("CF0")
                END IF
             END IF
          END IF
    WEND
    SWEN:
    CLOSEF()
    SOFT_WR("EF")
END SUB
```

LEERN: Ejecuta la acción princpipal de las funciones LEER

```
SUB FUNCTION LEERN(DIM LON AS WORD,DIM BYREF CADENA AS STRING[40]) AS BYTE
    RTS=0
    UART1_WRITE_TEXT("RDF ")
    UART1_WRITE(LON>>24) UART1_WRITE(LON>>16)
    UART1_WRITE(LON>>8)  UART1_WRITE(LON)
    RTS=1                UART1_WRITE(13)
    SOFT_RD()
    IF CADENA[LON]="C" THEN
        RESULT=0
    ELSE
        RESULT=1
    END IF
END SUB
```

3.6.4. Funciones para el control de los archivos

OBTENER FECHA: Fija la fecha del archivo(AÑO, MES, DÍA, HORA, MINUTO Y SEGUNDO)

```
SUB PROCEDURE OBTENERFECHA()
        I=0 ST[0]=0X0D ST[1]=0
        WHILE I<4          'LEYENDO ANIO
              IF UART1_DATA_READY() THEN
                 PORTA.2=1
                 CADENA[I]=UART1_READ()
                 I=I+1
              END IF
        WEND
        CADENA[4]=0        YEA=STRTOWORD(CADENA)-1980
        PORTA.2=0    I=0
        WHILE I<2   'LEYENDO MES
              IF UART1_DATA_READY() THEN
                 PORTA.2=1
                 CADENA[I]=UART1_READ()
                 I=I+1
              END IF
        WEND
        CADENA[2]=0        MON=STRTOWORD(CADENA)
        PORTA.0=0    I=0
```

```
        WHILE I<2   'LEYENDO DIA
              IF UART1_DATA_READY() THEN
                 PORTA.2=1
                 CADENA[I]=UART1_READ()
                 I=I+1
              END IF
        WEND
        CADENA[2]=0       DAY=STRTOWORD(CADENA)
        PORTA.2=0   I=0
        WHILE I<2   'LEYENDO HORA
              IF UART1_DATA_READY() THEN
                 PORTA.2=1
                 CADENA[I]=UART1_READ()
                 I=I+1
              END IF
        WEND
        CADENA[2]=0        HOU=STRTOWORD(CADENA)
        PORTA.2=0   I=0
        WHILE I<2      'LEYENDO MINUTOS
              IF UART1_DATA_READY() THEN
                 PORTA.2=1
                 CADENA[I]=UART1_READ()
                 I=I+1
              END IF
        WEND
        CADENA[2]=0      MIN=STRTOWORD(CADENA)
        PORTA.2=0   I=0

          WHILE I<2   'LEYENDO SEGUNDOS
                IF UART1_DATA_READY() THEN
                   PORTA.2=1
                   CADENA[I]=UART1_READ()
                   I=I+1
                END IF
          WEND
          CADENA[2]=0       SEC=STRTOWORD(CADENA)/2
          SOFT_WR("OK")        CADENA[1]=0
          PORTA.2=0
END SUB
```

Tabla 3-2: CONFIGURACIÓN DE LA FECHA

32 Bit Values	16 Bit Values	Description	Allowable Values	Meaning
25:31	9:15	Year	0 – 127	0 = 1980 127 = 2107
21:24	5:8	Months	1 – 12	1 = January 12 = December
16:20	0:4	Days	1 – 31	1 = first day of month
11:15	N/A	Hours	0 – 23	24 hour clock
5:10	N/A	Minutes	0 – 59	
0:4	N/A	Seconds/2	0 – 29	0 = 0 seconds 29 = 58 seconds

Esta tabla muestra la configuración de la fecha, se puede observar que el año esta seteado desde 1980, los meses del 1 al 12, los días del 1 al 31, las horas de 0 a 23, los minutos de 0 a 59 y los segundos de 0 a 29, se entienden que se toman de dos en dos.

CLOSEF: Verifica que los archivos queden cerrados despues del acceso.

```
SUB FUNCTION CLOSEF() AS BYTE
    UART1_WRITE_TEXT("CLF ")
    UART1_WRITE_TEXT(LASTFILE)
    RTS=1 UART1_WRITE(13)
    SOFT_RD()
    IF STRCMP(CADENA,"D:\>")=0 THEN
       RESULT=1
    ELSE
       RESULT=0
    END IF
END SUB
```

OPENR: Verifica si el archivo está listo para ser leido

```
SUB FUNCTION OPENR() AS BYTE
    RTS=0
    STRCPY(CADENA,"OPR ") STRCAT(CADENA,LASTFILE)
    UART1_WRITE_TEXT(CADENA)
    RTS=1  UART1_WRITE(13)
    SOFT_RD()
    IF STRCMP(CADENA,"D:\>")=0 THEN
       RESULT=1
    ELSE
       RESULT=0
    END IF
END SUB
```

OPENW: Verifica si el archivo está listo para escrito

```
SUB FUNCTION OPENW(DIM LON AS LONGWORD) AS BYTE
    STRCPY(CADENA,"OPW ") STRCAT(CADENA,LASTFILE)
    STRCAT(CADENA," ")    UART1_WRITE_TEXT(CADENA)
    UART1_WRITE(LON>>24)  UART1_WRITE(LON>>16) 'BYTES DE FECHA DE CREACION
    UART1_WRITE(LON>>8)   UART1_WRITE(LON)
    RTS=1  UART1_WRITE(13)
    SOFT_RD()
    IF STRCMP(CADENA,"D:\>")=0 THEN
       RESULT=1
    ELSE
       RESULT=0
    END IF
END SUB
```

3.6.5. Función de interrupción para el reloj de tiempo real

Utilizando la interrupción del TIMER0, controlamos el reloj de tiempo real

```
sub procedure Interrupt()
    if counter=800 then
        inc(sec)
        counter=0
    else
        if sec=30 then
            inc(min)
            sec=0
        else
            if min=60 then
                inc(hou)
                min=0
            else
                if hou=24 then
                   inc(day)
                   hou=0
                else
                    if day=31 then
                       inc(mon)
                       day=1
                    else
                        if mon=13 then
                           inc(yea)
                           mon=1
                        end if
                    end if
                end if
            end if
        end if
    end if

    Inc(counter)         ' Increment value of counter on every interru
    TMR0L  = 96
    INTCON = 0x20        ' Set T0IE, clear T0IF
  end sub
```

3.6.6. Funciones de lectura/ESCRITURA A TRAVÉS DEL UART/SOFT_UART

SOFT_WR: Escribe a través de la función SOFT_UART_WRITE una cadena de bits en el archivo.

```
SUB PROCEDURE SOFT_WR(DIM BYREF CADENA AS STRING[40])
    DIM I AS BYTE
    FOR I=0 TO STRLEN(CADENA)-1
        SOFT_UART_WRITE(CADENA[I])
    NEXT I
    SOFT_UART_WRITE(0X0D)
END SUB
```

SOFT_RD: Lee a través de la función SOFT_UART_READ una cadena de bits dentro del archivo

```
SUB PROCEDURE SOFT_RD()
    DIM I AS BYTE
    I=0 RTS=0
    DO
       CADENA[I]=SOFT_UART_READ(TIMEOUT)
       I=I+1
    LOOP UNTIL CADENA[I-1]=0X0D
    CADENA[I-1]=0
END SUB
```

CAPÍTULO 4

4. SIMULACIÓN Y PRUEBAS

A continuación describiremos las simulaciones y pruebas desarrolladas en nuestro proyecto, cabe recalcar que no existe una herramienta en PROTEUS que simule el funcionamiento del memory stick datalogger, pero esto lo solucionamos utilizando las herramientas COMPIM y VIRTUAL TERMINAL.

Mostraremos primero las pruebas que realizamos utilizando las herramientas como el virtual Terminal, para ver cómo se desarrolla la comunicación entre el datalogger y los sensores, también se anexan fotos del proyecto implementado en protoboard y además la prueba realizada con el proyecto que controlaba el sensor de Orientación magnética.

4.1. Simulación en Proteus

A través de la herramienta COMPIM de Proteus podemos observar como se establece la comunicación entre el DATALOGGER y el controlador del sensor, se utilizó un puerto serial de la PC y un convertidor DB9-USB para el otro puerto, aquí observamos ambos terminales virtuales.

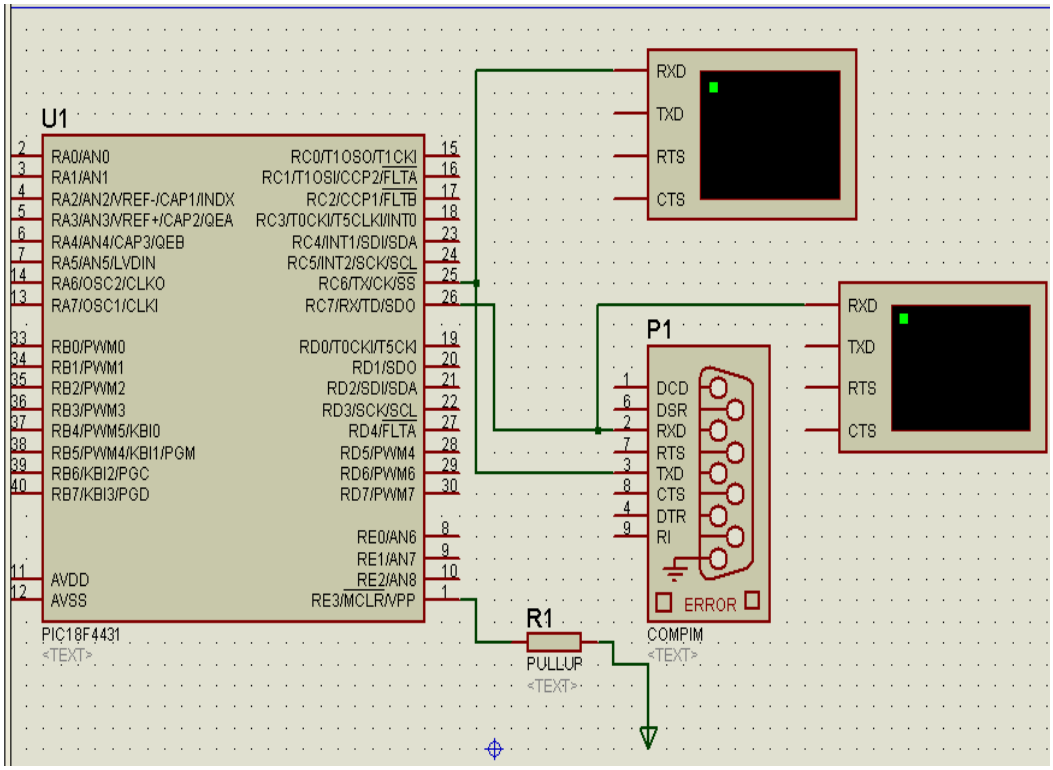


FIGURA 4-1: Simulación en PROTEUS

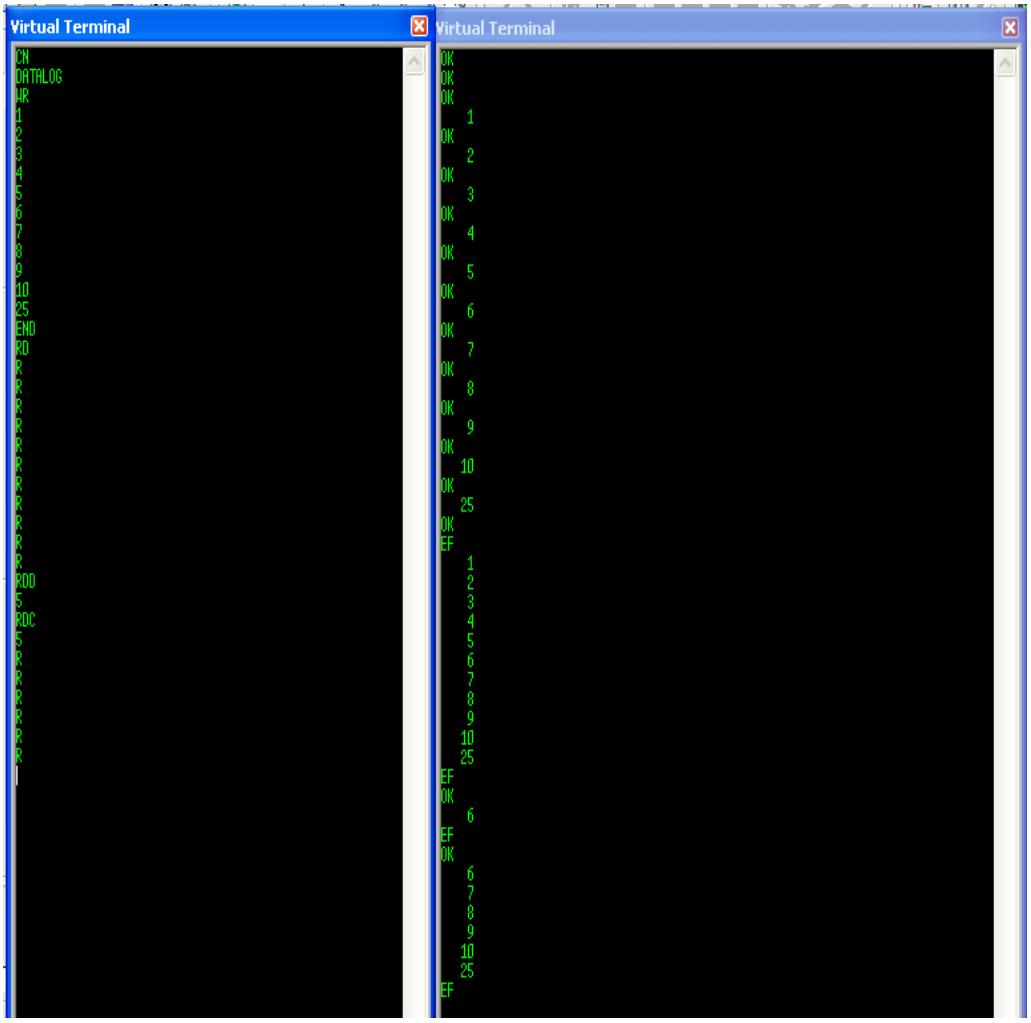


FIGURA 4-2: Simulación de comunicación entre el Datalogger y un controlador de sensor

Podemos observar una prueba del DATALOGGER COMPACTO, utilizando la herramienta del VIRTUAL TERMINAL de Proteus, se aprecian los comandos de comunicación CN(cambio de nombre del archivo) WR(Escribir datos en la memoria USB) y su debida respuesta de sincronización.

4.2. Implementación en protoboard

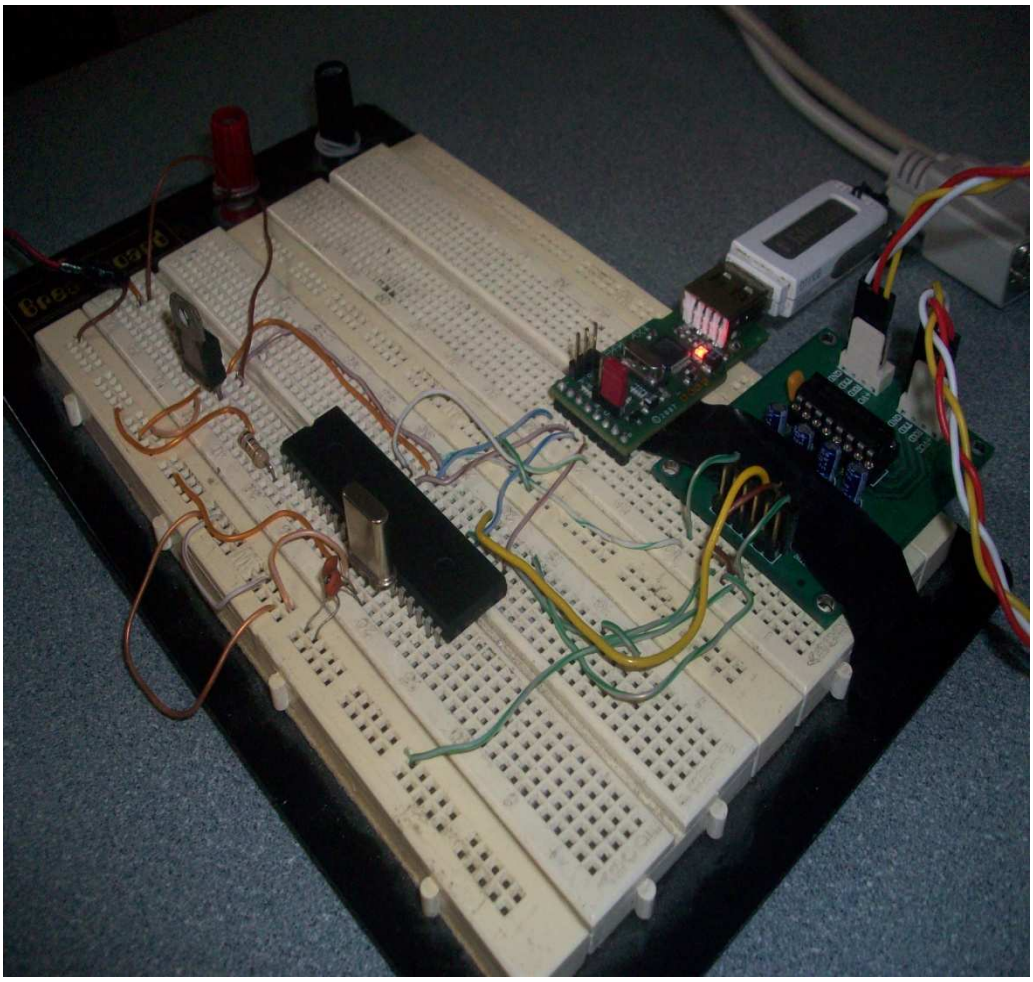


FIGURA 4-3: Datalogger compacto

En esta fotografía mostramos el DATALOGGER COMPACTO en funcionamiento con una memoria USB, se utilizó una tarjeta con un integrado MAX 232 para poder observar en la computadora la comunicación controlada por el PIC 18F4431.

4.3. Comunicación con un sensor

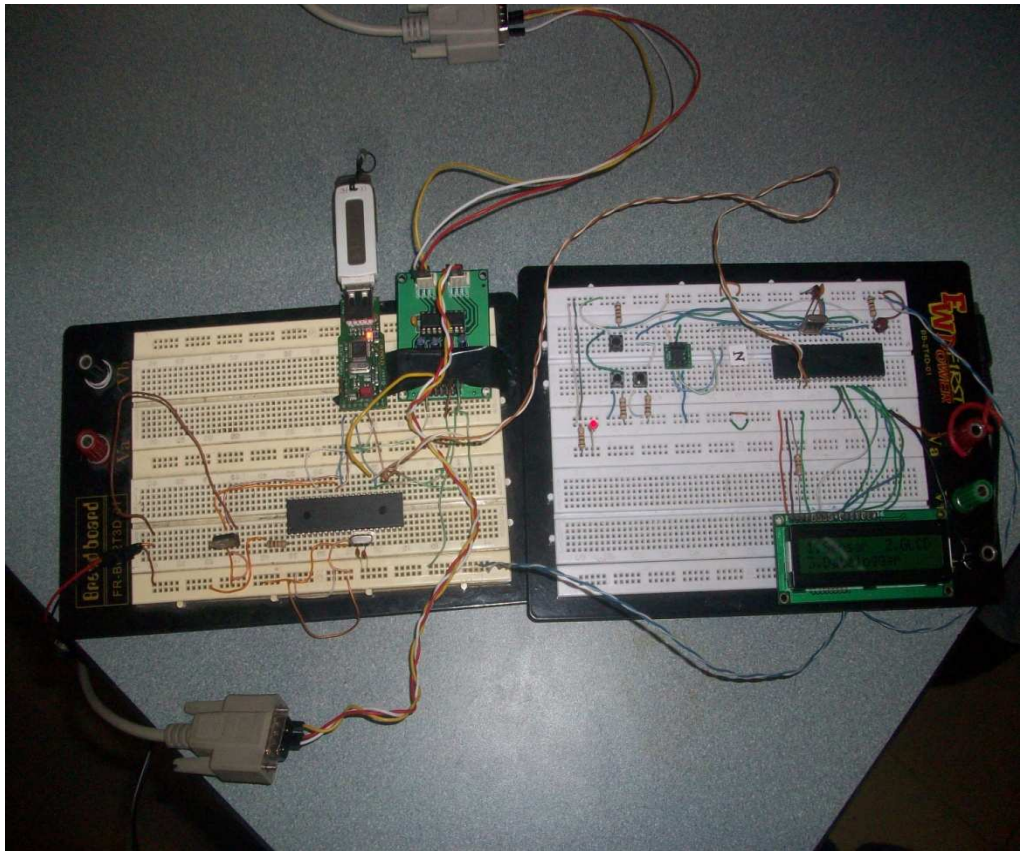
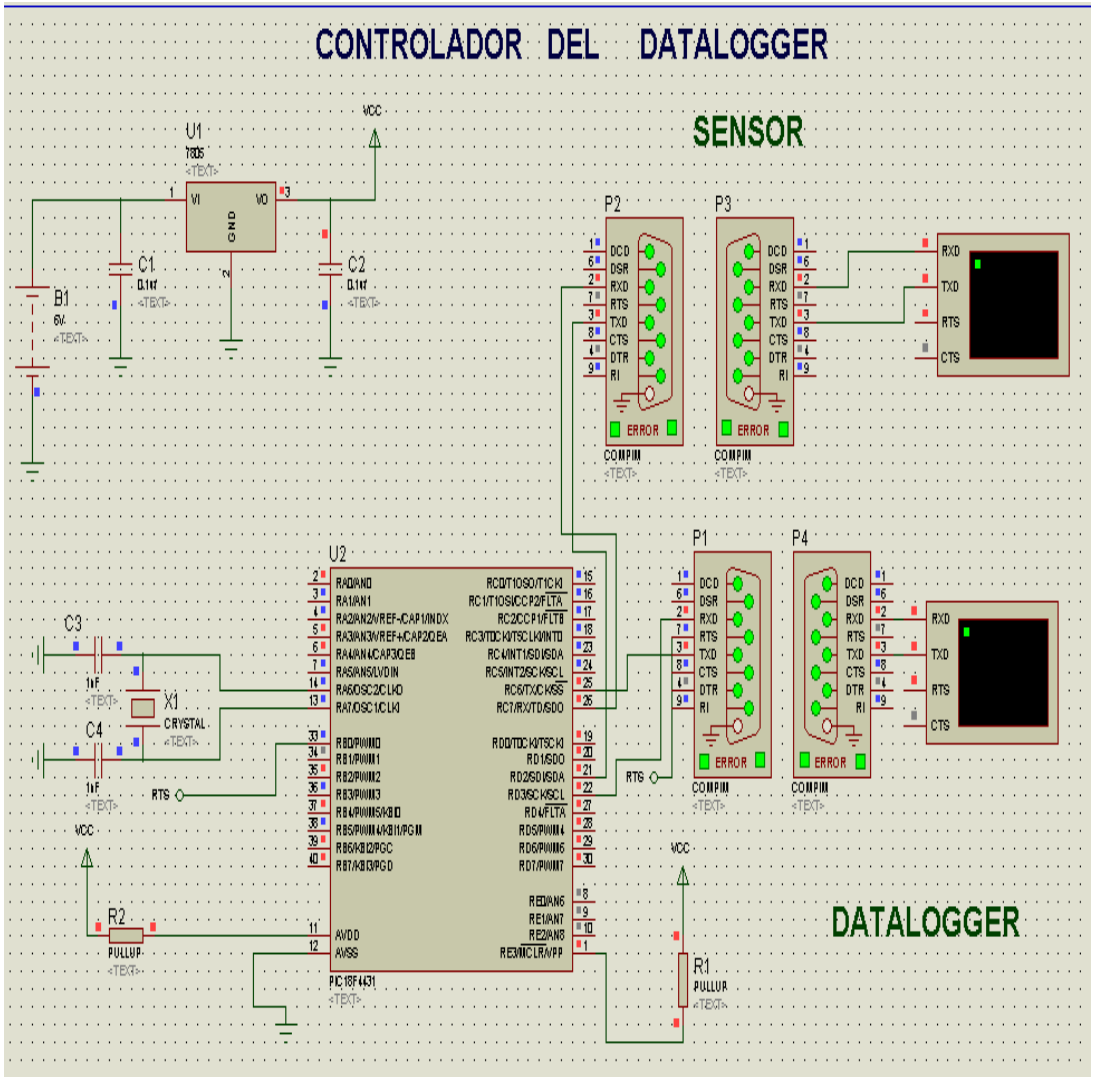


FIGURA 4-4: Comunicación con un sensor

En esta fotografía podemos apreciar al DATALOGGER COMPACTO ya en funcionamiento con el sensor externo ORIENTADOR MAGNÉTICO, este guarda datos de posición con respecto a la orientación magnética de la tierra. Igual podemos verificar la comunicación en la computadora gracias al MAX 232.

4.4. ESQUEMA DE CONEXIONES DEL CONTROLADOR

A continuación mostramos la simulación en Proteus con la ayuda de las herramientas COMPIM y VIRTUAL TERMINAL podemos suplir la ausencia de un MEMORY STICK DATALOGGER en Proteus.



CONCLUSIONES

1. Logramos construir un sistema que permite el almacenamiento de gran cantidad de información en una memoria USB, obtenida de diversos sensores externos, a través de comunicación serial asincrónica y la utilización de diversos comandos de comunicación entre el DATALOGGER y los sensores.
2. Debido a que resonador interno que posee el Microcontrolador no permitía una comunicación UART sin fallas, se tuvo que utilizar un oscilador externo, en este caso se utilizó un cristal de cuarzo a una frecuencia de 11.0592MHz para una mayor exactitud, ya que este cristal fue diseñado para mejorar la comunicación serial.
3. Como el Memory Stick que utilizamos para implementar el Datalogger utiliza comunicación serial, y además necesitábamos comunicarnos de forma serial con diversos sistemas externos, se utilizó dos puertos de comunicación serial UART asíncrono a una frecuencia de 9600 baudios.
4. Debido a que el Microcontrolador 18F4431 que utilizamos en nuestro proyecto sólo posee un par de pines para la comunicación UART por medio de Hardware, y necesitábamos dos, se tuvo que implementar una comunicación UART por medio de Software, la cual ya se encuentra

implementada en el Software que utilizamos, el MIKRO BASIC PRO for PIC.

5. Como no existe un Elemento que represente al Memory Stick en el Simulador, se tuvo que conectar el Microcontrolador Virtual al puerto de comunicación serial del computador hacia el Memory Stick, adaptando los niveles de voltaje que maneja el computador a los niveles que maneja el Memory Stick por medio de la utilización del componente MAX232.

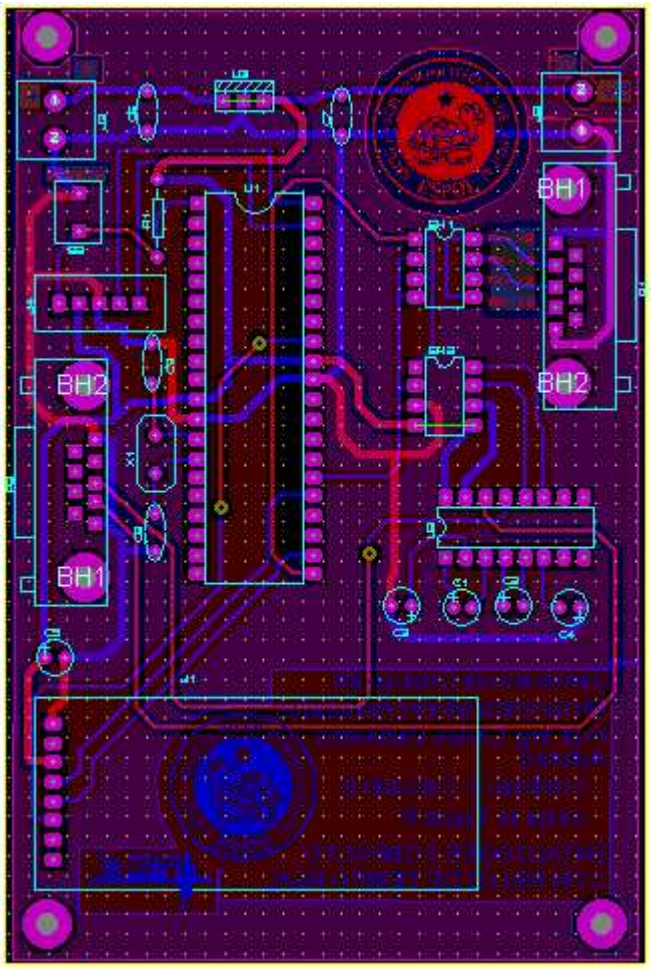
RECOMENDACIONES

1. Cuando se está transmitiendo comandos hacia el Memory Stick y se espera su respuesta, asegurar el envío de un nivel de voltaje bajo al pin CTS del Memory Stick, ya que este elemento dispone de comunicación serial con control de flujo por Hardware.
2. Verificar que el Microcontrolador envíe y reciba los datos correctamente de forma serial con el oscilador que se esté utilizando, por medio de un osciloscopio o un computador que posea comunicación serial.
3. Crear un buen modelo de comandos para que la comunicación entre el datalogger y los sensores sea fácil, y muy transparente.
4. Es necesario para una buena comunicación que las referencias a tierra entre el controlador del datalogger y el sensor estén conectados entre sí.

ANEXOS

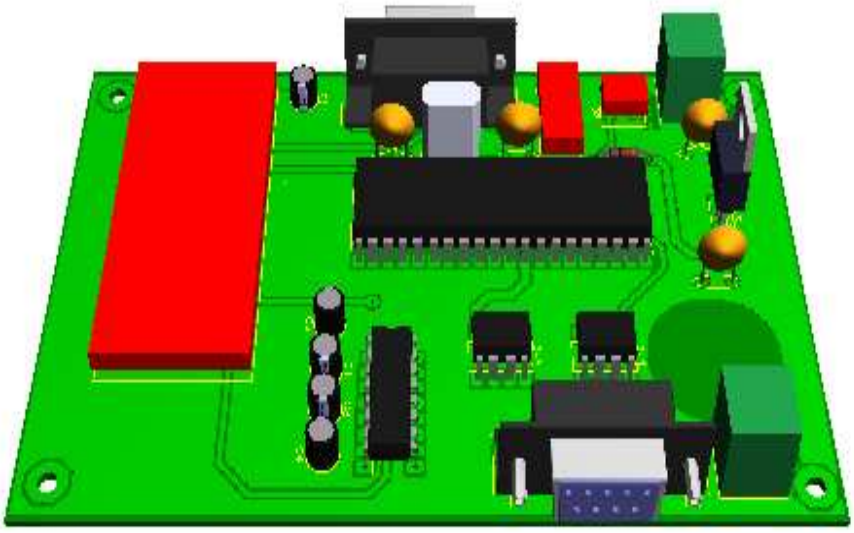
ANEXO A: DISEÑO DE LA TARJETA ELECTRÓNICA

El diseño se realizó utlizando las herramientas de PROTEUS en conjunto con ARES, cabe mencionar que se diseño una tarjeta con doble recubrimiento de cobre.



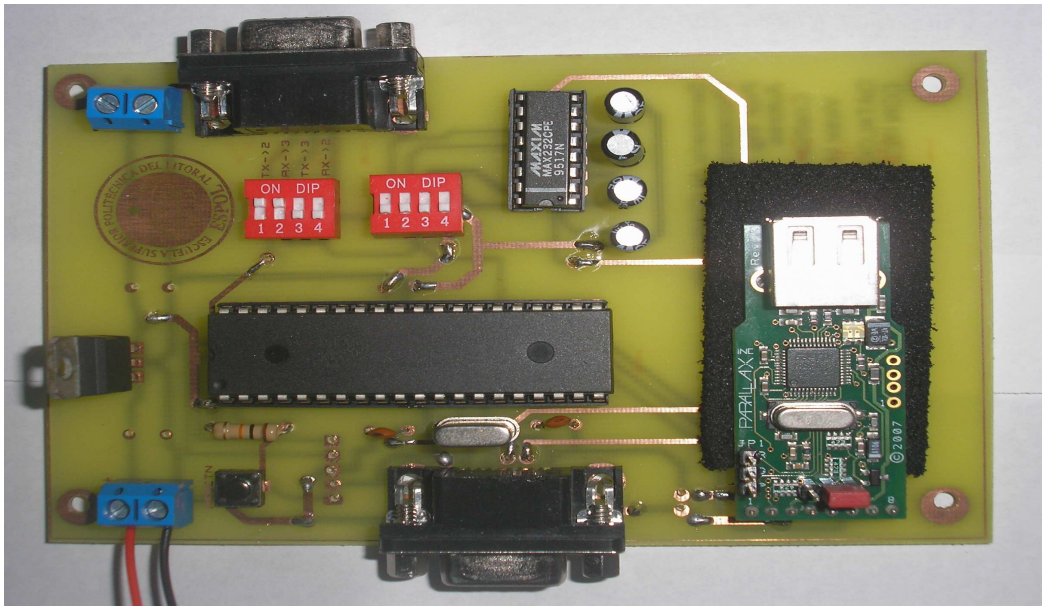
ANEXO B: VISTA 3D DEL DISEÑO.

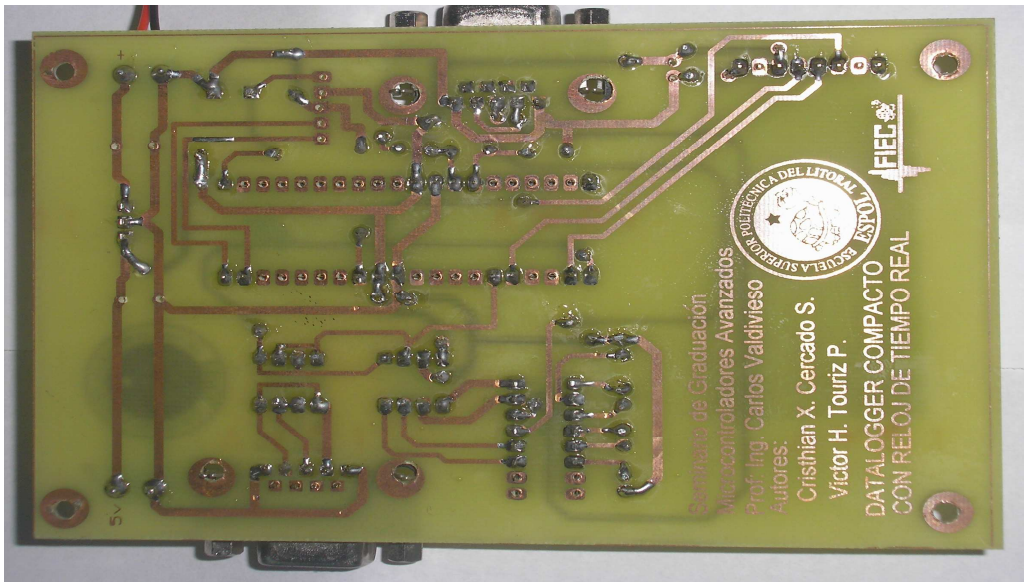
Esta herramienta de ARES nos permite visualizar como quedaría el diseño de la tarjeta



ANEXO C: FOTOGRAFÍAS DE LA TARJETA ELECTRÓNICA.

Estas fotografías muestran el resultado de un eficaz uso de las herramientas de PROTEUS Y ARES además de la paciencia y destreza que requiere soldar lo elementos.





BIBLIOGRAFÍA

1. MANUEL GIL RODRIGUEZ, Introducción rápida a Matlab y Simulink, Ediciones Díaz Santos, Madrid , 2003.

2. ZONA GPS, GPS USB TRAVEL LOGGER, www.6300-i-gotu-gt-100-usb-gps-travel-logger.htm , 23/04/2010

3. DATAQ INSTRUMENTS, DATALOGGER SYSTEM, www. di710.htm, 23/04/2010

4. INFOEPE, DATALOGGER COMPACTO DE DAQSTATION, http://www.Yokogawa-Iberia-presenta-el-nuevo-Data-Logger-compacto-MV-de-DAQSTATION-n1220.htm, 23/04/2010

5. MIKROELECTRÓNICA, PIC Microcontrollers - Programming in BASIC Mikroelektronika, http://www.mikroe.com/en/books/pic-books/mikrobasic/, 23/04/2010

6. MICROCHIP, Hoja de Datos PIC18F4431, http://www.datasheetcatalog.net/es/, 24/04/2010

7. PARALLAX, MEMORY STICK DATALOGGER, ww.parallax.com,24/04/2010