ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

TESIS DE GRADO

"IMPLEMENTACION DE LECTOR RFID Y SENSOR DE TEMPERATURA A UN ROBOT MINDSTORM NXT PARA SER UTILIZADOS EN UN SISTEMA DE ADQUISICION DE DATOS"

Previa a la obtención del titulo de:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

PRESENTADA POR:

MARCO VINICIO SOTOMAYOR SANCHEZ

CHRISTIAN XAVIER LOOR VELASQUEZ

GUAYAQUIL – ECUADOR

2008

AGRADECIMIENTO

A todos quienes

contribuyeron en el

desarrollo de este trabajo

DEDICATORIA

A mis Padres y Sandra

Marco

A mi amada esposa Tere e hijos

A mis Padres, especialmente a mi madre

Christian

TRIBUNAL DE GRADUACIÓN

PRESIDENTE

Ing. Holger Cevallos Ulloa

DIRECTOR DE TESIS

Ing. Carlos Valdivieso A.

MIEMBROS PRINCIPALES

Ing. Washington Medina M.

Ing. Maria A. Álvarez V.

DECLARACIÓN EXPRESA

"La responsabilidad del contenido de esta Tesis de Grado, nos corresponde exclusivamente; y el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Marco Vinicio Sotomayor Sánchez

Christian Xavier Loor Velásquez

RESUMEN

Realización del diseño e implementación de un módulo lector RFID y un módulo sensor de temperatura, sobre un sistema de invención robótica llamada Lego Mindstorm NXT, con el objetivo de diseñar e implementar un sistema de adquisición de datos, el cual funciona montando los módulos lector RFID y sensor de temperatura sobre un modelo de robot móvil controlado remotamente desde una PC via Bluetooh.

Para poder realizar una demostración del sistema, el robot móvil debe ser guiado hacia una muestra, la cual tiene una etiqueta RFID con un único código de identificación. Una vez posicionado el robot móvil cerca de la muestra, se procede a la identificación de la misma y posterior lectura de la temperatura de la muestra utilizando un brazo robótico, en el cual ha sido montado un sensor de temperatura digital. El muestreo de temperatura puede ser realizado de dos formas: obteniendo los datos en tiempo real y realizando una captura de datos independiente del PC, utilizando para ello un programa ejecutado en el controlador del Lego Mindstorm NXT, el cual almacena los datos obtenidos en un archivo de texto que posteriormente se puede recuperar en el lado del PC.

INDICE GENERAL

ABREVIATURAS

ARM	Advanced RISC Machines
CLR	Common Language Runtime
CSR	CAMBRIDGE SILICON RADIO
DCE	Data Communication Equipment
DTE	Equipo terminal de datos
EDR	Enhanced data rate
EIA	Electronic Industries Alliance
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
JIT	Just In Time
LCD	Liquid Cristal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
MIT	Massachussets Institute of Technology
MPL	Mozilla Public License
MSB	Most Significant Bit
MSIL	Microsoft Intermediate Language
NXC	Not eXactly C
PWM	Pulse-width modulation
RFID	Identificación por Radio Frecuencia
RIS	Robotic Invention System
RXC	Robotic Control Explorer
SCL	Serial Clock
SDA	Serial Data
SDK	software development kit
SPI BUS	Serial Peripheral Interface Bus
SPP	Serial Port profile
UART	Universal Asynchronous Receiver-Transmitter

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

La robótica es un excelente medio que permite a los estudiantes de ingeniería integrar diferentes tecnologías con el objetivo de encontrar la solución a un problema. Una forma interesante de contribuir al aprendizaje de la robótica, es brindando mejores oportunidades a los estudiantes para construir prototipos, facilitando la construcción de los mismos, sin preocuparse de la labor manual que requiere la elaboración de la estructura que soporta el prototipo, permitiéndoles enfocarse fundamentalmente en cuestiones como las técnicas de control, electrónica, programación, mecánica, etc.

En esta tesis se presentará el desarrollo de un sistema de adquisición de datos de temperatura en el cual se ha utilizado un sistema de invención robótica llamado Lego Mindstorm NXT. Esta herramienta nos permitirá construir un robot móvil por medio de piezas de lego y servo motores. Adicionalmente nos proveerá, un controlador que posee interfaz de usuario, un RTOS con máquina virtual capaz de interpretar comandos directos enviados desde una PC vía Bluetooth. Para poder identificar la muestra y tomar la temperatura se

desarrollarán dos nuevos sensores digitales (lector RFID y sensor de temperatura) no incluidos en el Kit Lego Mindstorm NXT.

Finalmente el sistema de adquisición de datos se completa desarrollando el software de PC necesario para las tareas de control del robot, captura y almacenamiento de datos de temperatura, utilizando una versión "express" de Visual Basic.

CAPITULO 1

ANALISIS DEL PROBLEMA

1.1 ANTECEDENTES

Originalmente se observa la necesidad de introducir una plataforma robótica que contribuya a un laboratorio de ingeniería para la elaboración de proyectos. Una de las opciones más interesantes fue la implementación de un sistema basado en la tecnologia de Lego ® MindStorm NXT.

Para poder dimensionar la funcionalidad de esta tecnologia decidimos crear un sistema de adquisición de datos con la creación de dos nuevos módulos, los cuales son un módulo de lector RFID y sensor de temperatura.

1.2 OBJETIVOS.-

- Desarrollar un robot móvil que monitoree una zona delimitada, con movimientos controlados remotamente a través de una PC vía Bluetooth, en busca de muestras específicas identificadas mediante una etiqueta con tecnología RFID, para la toma de temperatura, en donde los datos se almacenan y visualizan en la PC.

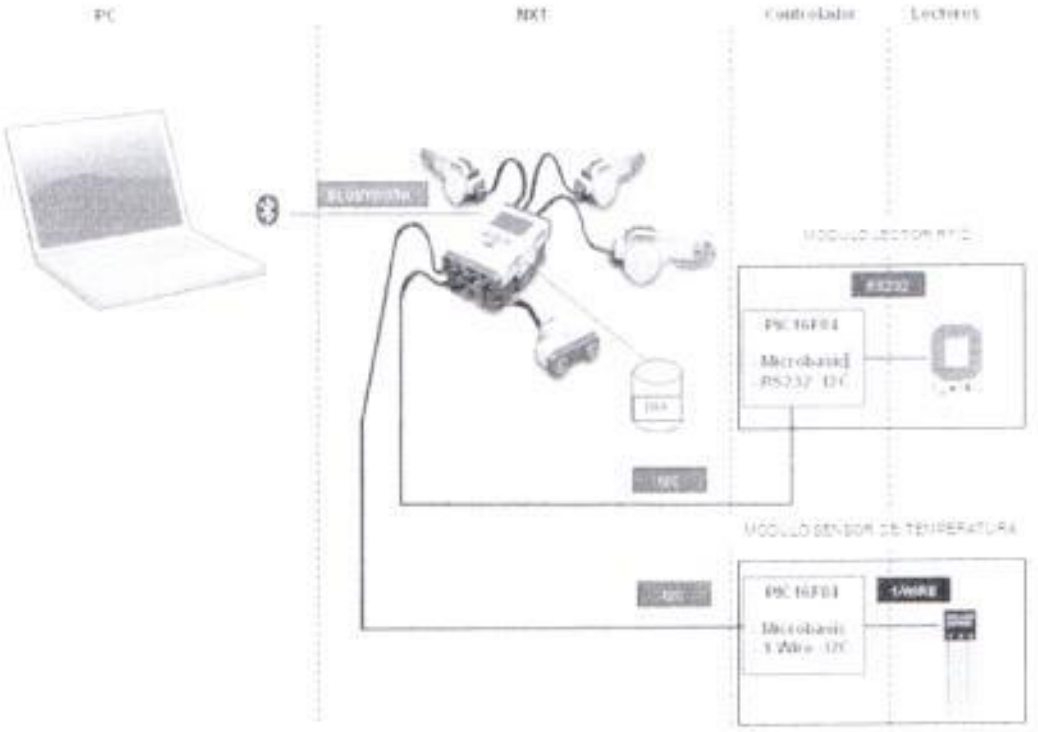


Figura 1.1 Diagrama general del sistema de adquisición de datos

- Demostrar la utilidad de la arquitectura Lego ® MindStorm NXT como medio de aprendizaje y experimentación en donde se involucran diferentes áreas de la ingenieria, como la mecánica, electrónica e informática.
- Incorporar dos nuevos sensores al sistema de invención robótica.

1.3 OBJETIVOS ESPECÍFICOS.-

- Comunicar el controlador Lego ® MindStorm NXT y la PC via bluetooth en la aplicación de esta tesis.
- Implementar un módulo de identificación de muestras con tecnologia RFID basado en un lector RFID de PARALLAX.
- Desarrollar una interfaz de comunicación RS232/I2C entre lector RFID y controlador Lego ® MindStorm NXT utilizando un microcontrolador.
- Implementar un sensor de temperatura basado en DS1820.
- Desarrollar una interfaz de comunicación 1-Wire/I2C entre sensor de temperatura y el controlador Lego ® MindStorm NXT.
- Desarrollar programa para la navegación controlada remotamente a través de una PC via Bluetooth utilizando el lenguaje de programación Visual Basic Net.

- Desarrollar programas para el controlador Lego ® MindStorm NXT para la identificación de las muestras, adquisición y almacenamiento de datos con el lenguaje de programación NXC.

1.4 BENEFICIOS DE LA APLICACIÓN DE TECNOLOGÍA LEGO MINDSTORM NXT.

Una de las ventajas más relevantes de este sistema de invención robótica es que permite crear prototipos sin necesidad de tener habilidades tales como la carpintería, trabajo de metales, etc., siendo ideal para instituciones educativas que necesitan una plataforma reusable para proyectos de laboratorio de ingeniería.

Una característica importante es la tecnología multitarea de su procesador que permite crear programas que pueden ser ejecutados en paralelo, lo cual es muy importante en robótica por que permite realizar diferentes tareas al mismo tiempo.

La gran difusión de esta tecnología, materiales de soporte técnico, protocolos estándares de comunicación permiten a los estudiantes poner en práctica conceptos de programación aprendidos, tanto en el lado de la PC, firmware de los módulos, software en el robot, etc.

CAPITULO 2

FUNDAMENTOS TEORICOS

2.1 ARQUITECTURA LEGO ® MINDSTORM NXT

Antes de analizar con más detalle este sistema de invención robótica (RIS) sobre el que se implementarán las interfaces de temperatura y RFID, es importante conocer un poco sus antecedentes y su propósito.

Lego Mindstorm NXT es fabricado por la empresa LEGO, cuyo modelo antecesor fue el Lego Mindstorm RXC (Robotic Control Explorer), el cual fue comercializado por primera vez en el año 1998 como un sistema de invención robótica. Lego Mindstorm es el resultado de la colaboración entre la empresa LEGO y el MIT (Massachussets Institute of Technology), los cuales realizaron un

acuerdo en el que la empresa LEGO financiaría investigaciones del grupo de epistemología y aprendizaje del MIT y a cambio no tendría que pagar regalías al MIT por proporcionarle nuevas ideas para sus productos.

Una de las contribuciones del MIT fue la creación del MIT Programmable Brick (Ladrillo programable) cuyo mentor del grupo de desarrollo es el matemático Seymour Papert, el cual es considerado un pionero de la inteligencia artificial y uno de los creadores del lenguaje de programación LOGO.

En la Figura 2.1. se muestra las partes básicas del Kit Lego Mindstorm NXT en la que se puede apreciar el bloque programable del NXT (Brick), tres motores y cuatro sensores.

Figura 2.1 Partes básicas del Kit Lego Mindstorm NXT.

Las especificaciones de hardware para el bloque programable NXT, se las presentan a continuación en un resumen:

Procesador principal:	ATMEL 32 bit ARM, AT91SAM7S256
	256 KB Flash
	64 KB Ram
	48 MHz
Coprocesador:	ATMEL 8 bit AVR, ATmega48
	4 Kb Flash
	512 Byte Ram
	8 MHz
Comunicación inalámbrica Bluetooth:	CSR BlueCore 4 V2.0 + EDR System
	Soporta SPP
	47Kb Ram interna
	Flash externa de 8MBit
	26 MHz
Comunicación USB 2.0:	12 Mbit/s
4 Puertos de entrada:	Interface de 6-cables que soporta

	Comunicación analógica y digital.
3 Puertos de salida:	Interface de 6-cables que adicionalmente soporta entradas para encoders.
Display:	LCD monocromático de 100 x 64 pixel
Parlante:	Sonido con 8 bits de resolución Soporta frecuencias de 2-16Khz
4 Botones de usuario:	Botones de goma
Fuente de poder:	6 pilas tipo AA Se recomienda pilas alcalinas
Conectores:	6 – Cables, RJ12 con ajuste derecho.

A continuación en la Figura 2.2 se muestra un diagrama de bloques de las unidades que componen el bloque programable del NXT.

En el Apéndice A se presenta mas referencia de este tema.

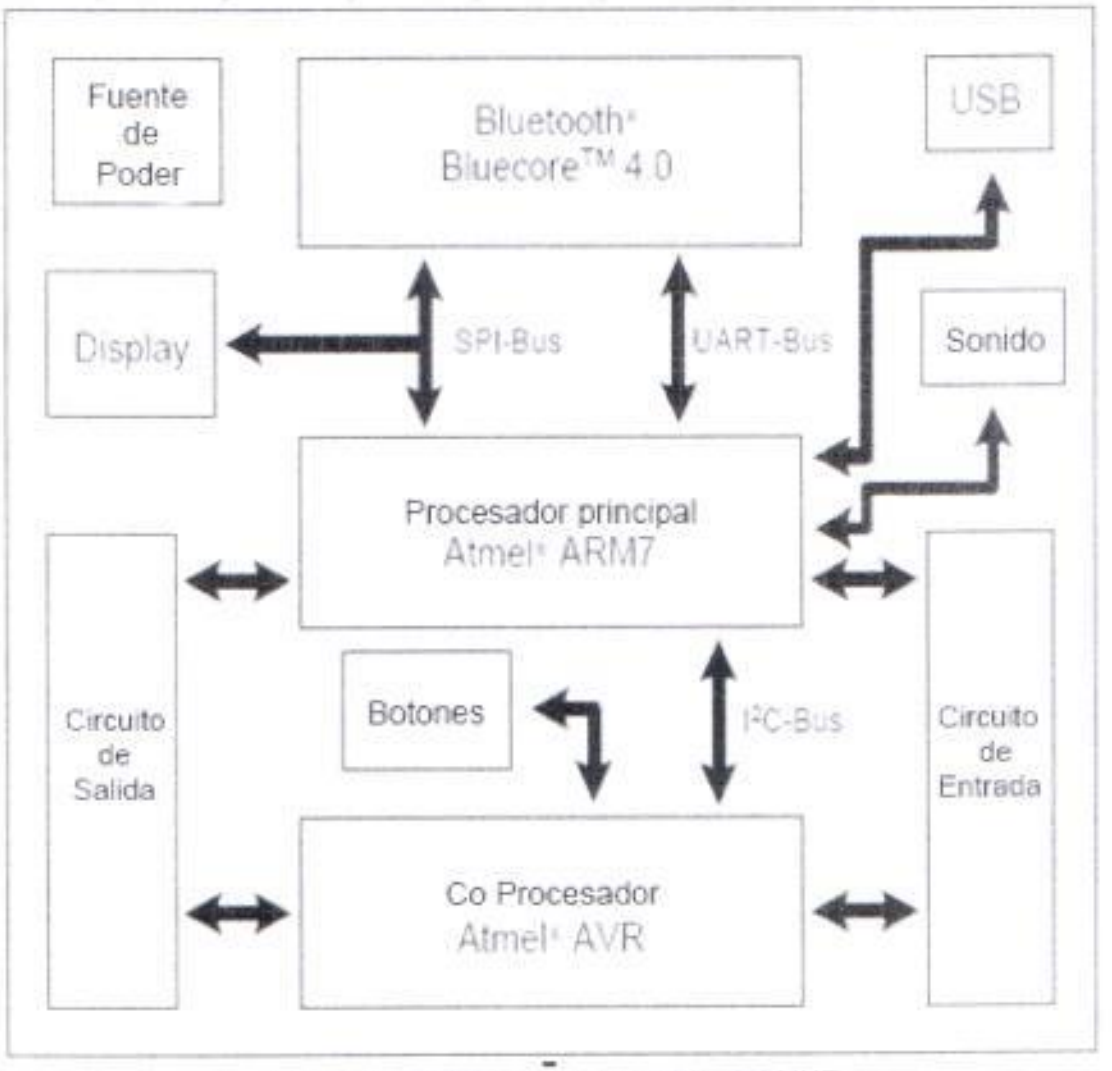


Figura 2.2 Bloque programable del NXT

2.2 PROTOCOLOS DE COMUNICACIÓN UTILIZADOS

2.2.1 I2C

El bus I2C (Inter-Integrated Circuit) desarrollado por la empresa Philips es un bus serie, formado por dos hilos, que puede conectar varios dispositivos

mediante un hardware muy simple. La versión 1.0 data del año 1992 y la versión 2.1 del año 2000.

La velocidad es de 100Kbits por segundo en el modo estándar, aunque también permite velocidades de 3.4 Mbit/s.

Los dispositivos conectados al bus I2C mantienen un protocolo de comunicaciones del tipo maestro/esclavo, tal y como se muestra en la Figura 2.3.

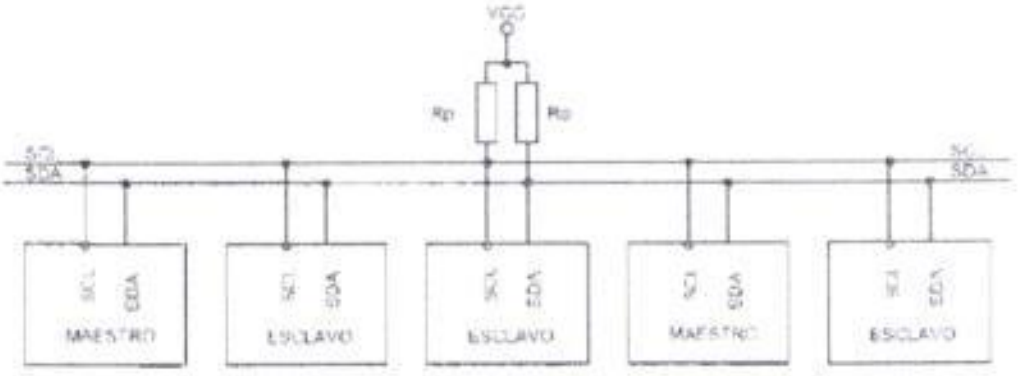


Figura 2.3 Estructura de un bus I2C

El ***maestro*** tiene como función iniciar y terminar la transferencia de información, además de generar la señal de reloj.

El ***esclavo*** es direccionado por el maestro.

Cada dispositivo conectado al bus ***I2C*** es reconocido por una única dirección que lo diferencia del resto de los dispositivos conectados.

El bus ***I2C*** puede ser multi-maestro esto significa que puede soportar más de un dispositivo capaz de controlar el bus.

Por los dos hilos se produce una comunicación serie bit a bit. Se transmiten dos señales una por cada línea:

SCL, (serial clock). Es la señal de reloj que se utiliza para la sincronización de los datos, que es siempre responsabilidad del dispositivo maestro.

SDA, (serial data). Es la línea para la transferencia serie de los datos. Esta línea es bidireccional, es decir, tanto el maestro como los esclavos pueden transmitir, dependiendo de la función del dispositivo.

El hardware del bus I2C se basa en la logica And cableada, para lo cual las etapas de salida de los dispositivos conectados al bus deben ser drenador abierto, o colector abierto. (Figura 2.4)

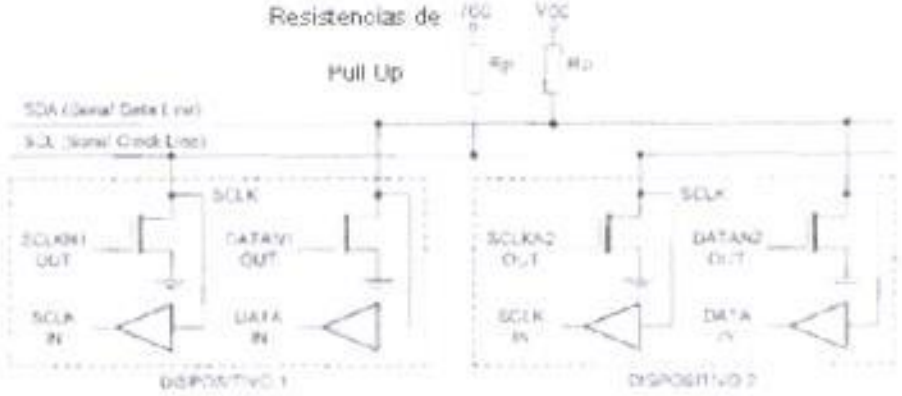


Figura 2.4 Etapa de salida de los dispositivos I2C

Las lineas SDA y SCL están conectadas a tensión positiva de alimentación a través de unas resistencias de Pull-Up (Rp). Dependiendo del estado del transistor de salida de cada dispositivo existen dos casos:

- Que el transistor esté saturado, con lo cual lleva a un nivel bajo o "0", a la linea correspondiente.
- Que el transistor esté en corte, estado de alta impedancia, con lo cual el estado de la linea depende de los otros transistores de los otros dispositivos.

En cuanto al cálculo para el valor de las resistencias de Pull-Up depende de la tensión de alimentación, de la capacidad del bus y del número de dispositivos conectados. Todo esto se tabula en las tablas que da el fabricante Philips.

El número de dispositivos conectados al bus viene limitado por la capacitancia máxima admitida, que es de 400pF.

El bus I2c en su transferencia de datos mantiene en su protocolo la siguiente secuencia de sucesos:

1. Condición de Start.
2. Transferencia de datos
3. Condición de Stop.

Para que la transferencia de datos, ilustrada en al Figura 2.5, puede ser iniciada el bus I2c debe estar libre, para esto las lineas de reloj (SCL) y datos (SDA) deben estar a un nivel alto. En este estado de bus libre cualquier dispositivo puede ocupar el bus como maestro.

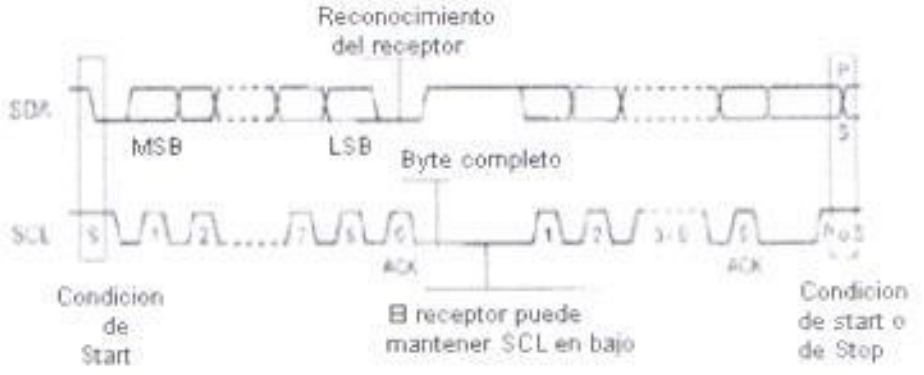


Figura 2.5 Transferencia de datos por e bus I2C.

Cuando el maestro verifica esto puede generar la condición de start (Figura 2.6), en donde SDA debe estar en flanco de bajada mientras que SCL permanece a nivel alto. Esta condición señala el comienzo de la transferencia de datos y alerta a los esclavos en espera de una transacción.

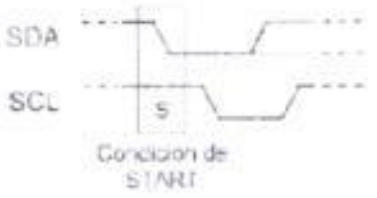


Figura 2.6 Condición de Stara

Para transferir un bit por la línea de datos SDA debe ser generado un pulso de reloj por la línea SCL. Los bits de datos transferidos por la línea SDA deben mantenerse estables mientras la línea SCL esté a nivel alto.

El estado de la línea SDA sólo puede cambiar cuando la línea SCL esta a nivel bajo. Como se muestra en la Figura 2.7.

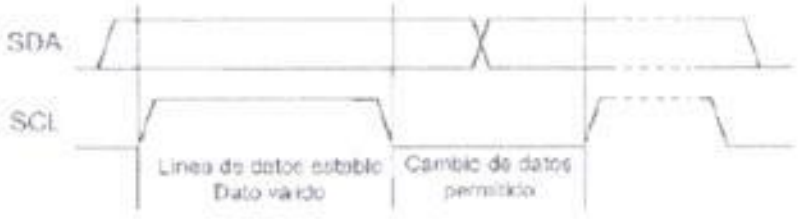


Figura 2.7 Transferencia de un bit por el bus I2C.

Cada 8 bits (un byte) enviados por al línea SDA representa un dato del protocolo I2C, todos los bytes de datos deben ser de 8 bits. El byte de datos se transfiere por bit de mayor peso denominado MSB (Most Significative Bit).

Después de la condición de Start el maestro envía el primer byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada (R/W), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo). Como se muestra en la Figura 2.8.

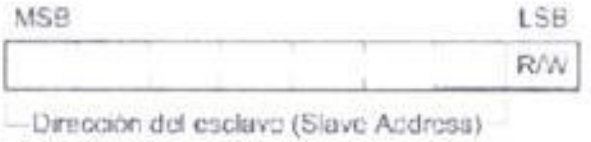


Figura 2.8 Primer byte después del bit de Start

La dirección enviada es comparada por cada esclavo del bus con su propia dirección, si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit R/W.

El esclavo responde enviando un bit de ACK que le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse (Figura 2.9).

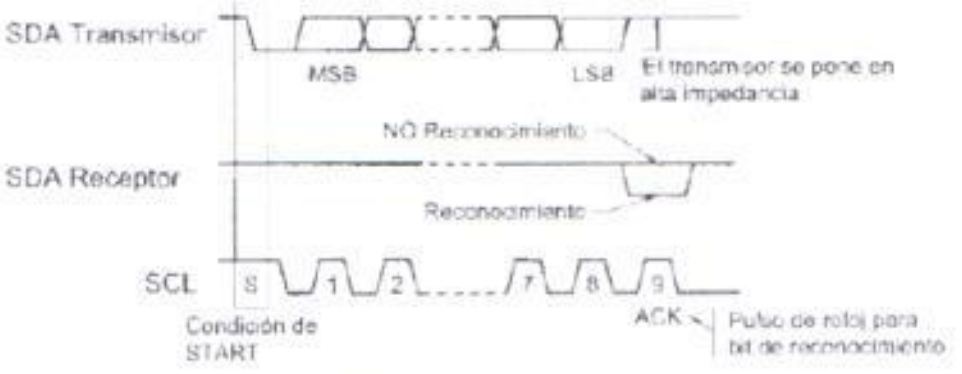


Figura 2.9 Bit de reconocimiento ACK

Seguidamente comienza el intercambio de información entre los dispositivos.

Ahora el maestro puede empezar a leer o escribir bytes de datos. El número máximo de bytes que pueden ser enviados en una transmisión no está

restringido, siendo el esclavo quien fija esta cantidad de acuerdo a sus características. Como se muestra en la Figura 2.10.

Cada byte leído/escrito por el maestro debe ser obligatoriamente reconocido por un bit de ACK por el dispositivo maestro/esclavo.

Aun cuando el maestro siempre controla el estado de la línea del reloj, un esclavo de baja velocidad o que deba detener la transferencia de datos mientras efectúa otra función, puede forzar la línea SCL a nivel bajo. Esto hace que el maestro entre en un estado de espera, durante el cual, no transmite información esperando a que el esclavo esté listo para continuar la transferencia en el punto donde había sido detenida.

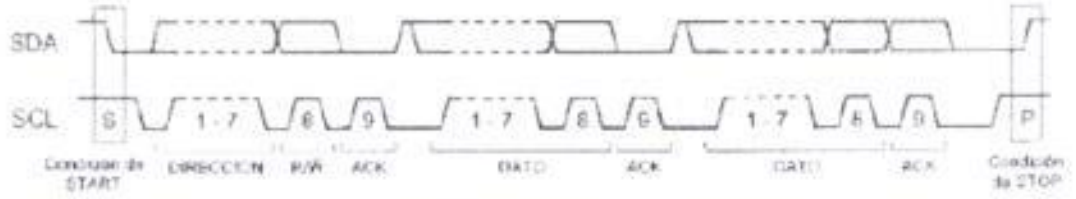


Figura 2.10 Transferencia completa de datos.

Cuando la transmisión de datos termina el maestro envía una condición de Stop, ilustrado en la Figura 2.11, en la cual SDA en flanco de subida mientras que el SCL permanece a nivel alto.

Luego de esto los buses deben estar obligatoriamente en estado inactivo por unos microsegundos.

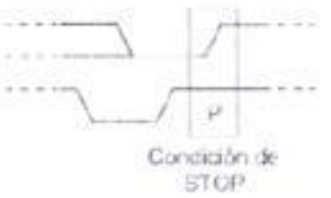


Figura 2.11 Condición de Stop.

2.2.2 1- WIRE

La tecnología 1-Wire consta básicamente de un protocolo serial implementado sobre una línea de datos más la referencia a tierra para poder establecer comunicación entre dispositivos.

Una red 1- Wire consta de un dispositivo maestro, el cual inicia y controla la comunicación con uno o mas dispositivos esclavos. Cada dispositivo esclavo consta de un número de identificación único e inalterable que viene programado de fábrica, el cual consta de 64 bits.

Lo que diferencia la tecnología 1-Wire de otros sistemas de comunicación serial como I2C o SPI, es que los dispositivos 1-Wire están diseñados para estar en contacto con el medio ambiente. Cualquier desconexión del bus o una pérdida de contacto pone a los esclavos 1- Wire dentro de un estado de reset definido. Cuando el voltaje retorna, el esclavo despierta y da señal de su presencia.

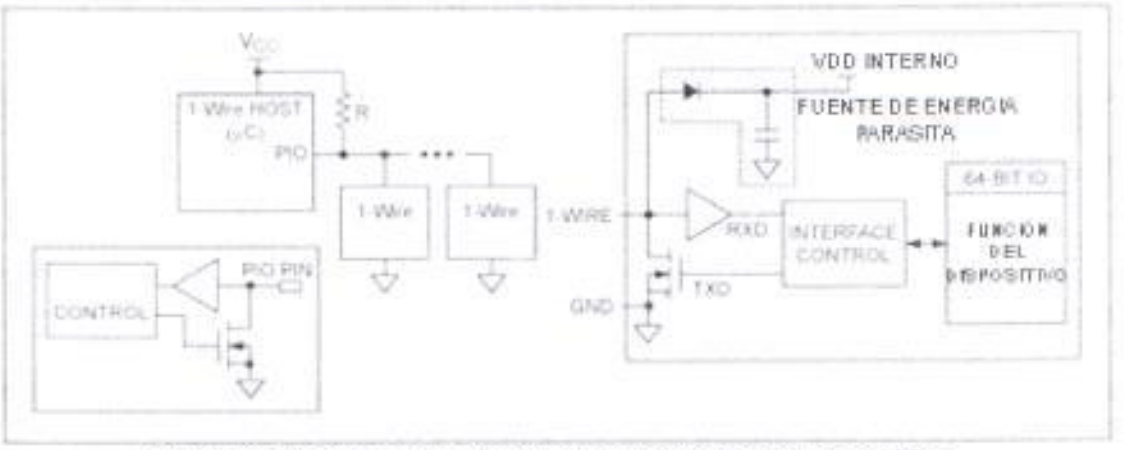


Figura 2.12 Diagrama de bloques de dispositivos 1-Wire.

Como se puede observar en el Figura 2.12 cada dispositivo de la red 1- Wire se conecta al bus a través de una linea en drenador abierto, lo que significa que cuando algún dispositivo no está transmitiendo posibilita que otros dispositivos puedan transmitir, colocándose en un estado de alta impedancia.

Se puede observar en el Figura 2.12 anteriormente presentado que el bus de una linea requiere de una resistencia de pull-up conectada a Vcc, lo cual significa que el estado de reposo del bus es el nivel alto.

El protocolo 1-Wire define las siguientes señales:

- Pulso de Presence.
- Pulso de Reset.
- Escritura de nivel lógico "0"
- Escritura de nivel lógico "1"
- Lectura de nivel lógico "0"

- Lectura de nivel lógico "1"

Todas las señales anteriormente citadas excepto "presence" son inicializadas por el microcontrolador maestro mediante un flanco de bajada y un nivel bajo de al menos 1 us.

Debe transcurrir un tiempo mínimo de 1 us entre cada señal diferente, tiempo en el cual el bus debe mantenerse en alta impedancia, dando como resultado un nivel alto en la línea debido a la resistencia de pull-up.

Los datos son transmitidos con el bit de menos peso en primer lugar.

Para la señal de reset, el microcontrolador maestro mantiene el bus en un nivel bajo por un período de tiempo mayor a 480 us, se producirá un reset a todos los dispositivos esclavos conectados al bus.

La señal presence es generada por el microcontrolador esclavo manteniendo en un nivel bajo el bus por un período de tiempo de entre 60 y 240us.

En al Figura 2.13 se muestra el proceso de inicialización de un dispositivo en el bus 1-Wire, juegan un papel importante las señales de Reset y Presence de la siguiente forma:

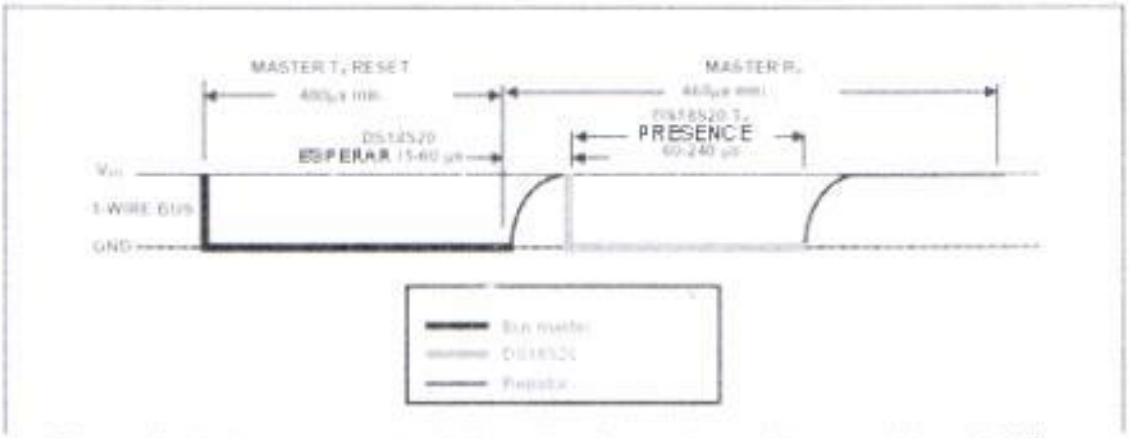


Figura 2.13 Proceso de inicialización de un dispositivo en el bus 1-Wire.

El maestro genera la señal de reset manteniendo en bajo la linea al menos 480 us.

Cuando termina de generar el pulso de reset el maestro pasa a modo de recepción, con lo que la linea se pone en un nivel alto debido a la resistencia de pull-up.

Cuando el esclavo detecta que terminó el pulso de reset, espera entre 15 y 60 us para generar el pulso de presence, que consiste en colocar en nivel bajo la linea entre 60 y 240 us. Terminado este tiempo el esclavo pasa a modo recepción, dejando el bus en alta impedancia.

La siguiente Figura 2.14 se ilustra el protocolo 1-Wire en el proceso de envio y recepción de datos por parte del maestro:

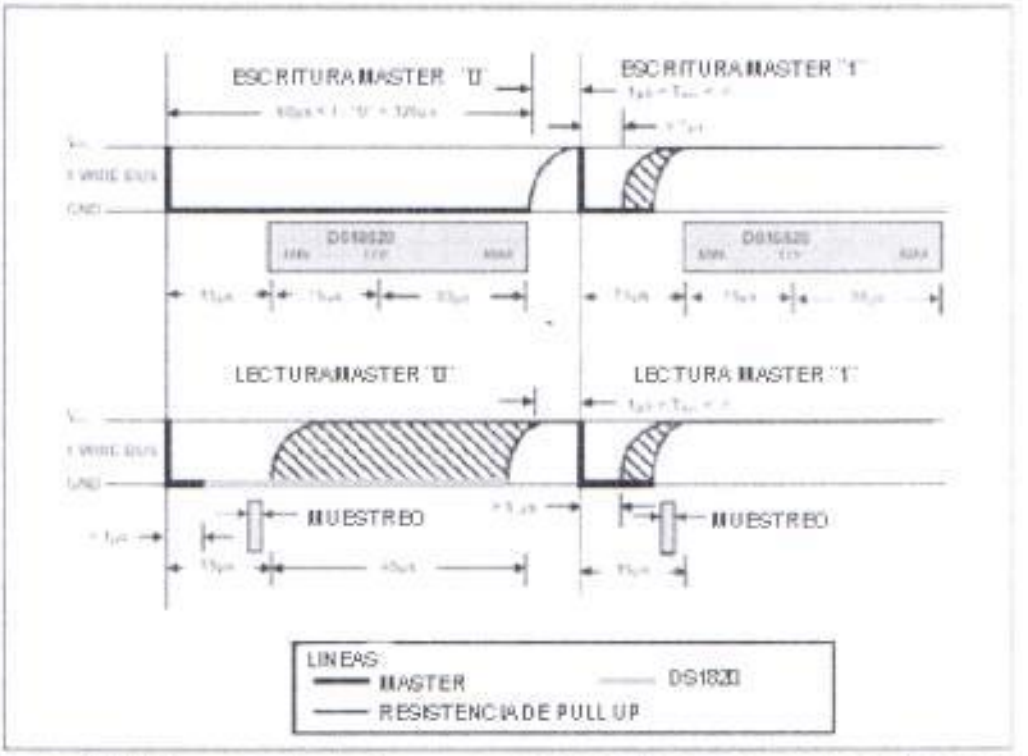


Figura 2.14 Proceso de envío y recepción de datos por parte del maestro.

Para enviar un bit "1", el maestro se lleva a 0 voltios la línea de datos durante 1-15 microsegundos. Para enviar un bit "0", el maestro se lleva a 0 voltios la línea de datos durante 60 microsegundos.

Los dispositivos esclavos leen el bit aproximadamente a los 30 microsegundos después del flanco de bajada de cada bit.

Cuando el maestro lee los datos del dispositivo esclavo el maestro pone 0 voltios entre 1-15 microsegundos en la línea de datos y a partir de ese momento el esclavo no hace nada (la señal pasa a vale 5 voltios) si quiere mandar un 1

lógico o mantiene la señal en 0 voltios hasta los 60 microsegundos si quiere mandar un 0 lógico.

2.2.3 RS-232

RS-232 (también conocido como Electronic Industries Alliance RS-232C) es una interfaz que designa una norma para el intercambio serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Data Communication Equipment, Equipo de Comunicación de datos), aunque existen otras situaciones en las que también se utiliza la interfaz RS-232.

Debido a que el estándar del puerto serie se mantiene desde hace muchos años, la institución de normalización americana (EIA) ha escrito la norma RS-232-C que regula el protocolo de la transmisión de datos, el cableado, las señales eléctricas y los conectores en los que debe basarse una conexión RS-232.

La comunicación realizada con el puerto serie es una comunicación asincrona. Para la sincronización de una comunicación se precisa siempre de una linea adicional a través de la cual el emisor y el receptor intercambian la señal del pulso. Pero en la transmisión serie a través de un cable de dos lineas esto no es posible ya que ambas están ocupadas por los datos y la masa. Por este motivo se intercalan antes y después de los datos informaciones de estado según el protocolo RS-232. Esta información es determinada por el emisor y receptor al

estructurar la conexión mediante la correspondiente programación de sus puertos serie.

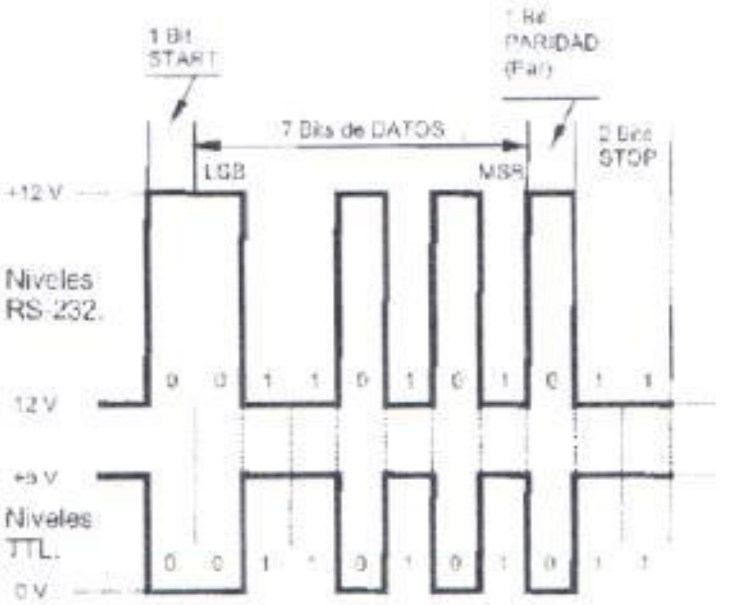


Figura 2.15 Ejemplo de envió de byte según normas RS232.

Los dispositivos deben tener una tierra en común, las tierras de los circuitos de los dos dispositivos deben estar conectadas.

Un cable de conexión de la salida del puerto emisor a la entrada del puerto receptor.

Si se quiere comunicación en los dos sentidos, otro cable de conexión del puerto que envia del receptor al receptor del transmisor.

Los niveles de los voltajes del transmisor deben ser aceptables por el receptor.

La polaridad de la señal debe ser la misma.

Las señales con las que trabaja este puerto serie son digitales, de +12V (0 lógico) y -12V (1 lógico), para la entrada y salida de datos, y a la inversa en las señales de control. El estado de reposo en la entrada y salida de datos es -12V. Dependiendo de la velocidad de transmisión empleada, es posible tener cables de hasta 15 metros.

Antes de iniciar cualquier comunicación con el puerto RS232 se debe de determinar el protocolo a seguir dado que el estándar del protocolo no permite indicar en que modo se está trabajando, es la persona que utiliza el protocolo el que debe decidir y configurar ambas partes antes de iniciar la transmisión de datos.

Siendo los parámetros a configurar los siguientes:

Protocolo serie (Bits de Datos - Bit de paridad - Bit de Stop)

Bits de Datos.-El RS-232 puede transmitir los datos en grupos de 5, 6, 7 u 8 bits

Bit de paridad.- con este bit se pueden descubrir errores en la transmisión. Se puede dar paridad par o impar.

Bit de Stop.- indica la finalización de la transmisión de una palabra de datos. El protocolo de transmisión de datos permite 1, 1.5 y 2 bits de parada.

La velocidad de transmisión, es la cantidad de información enviada por la línea de transmisión en la unidad de tiempo.

Hay distintas unidades para expresar esta medida, la mas utilizada es al baudio, que es proporcional a los bits/segundo (bps), definidos como el número de bits de información enviados por segundo.

Cualquier protocolo de control de flujo debe ser habilitado, no pasará nada hasta que se activen las lineas correspondientes y necesarias para empezar el procedimiento.

La comunicación serie RS232 asincrónica comienza enviado un bit de inicio con cada dato.

Bit de inicio.- cuando el receptor detecta el bit de inicio sabe que la transmisión ha comenzado y es a partir de entonces que debe leer las señales de la linea a distancias concretas de tiempo, en función de la velocidad determinada.

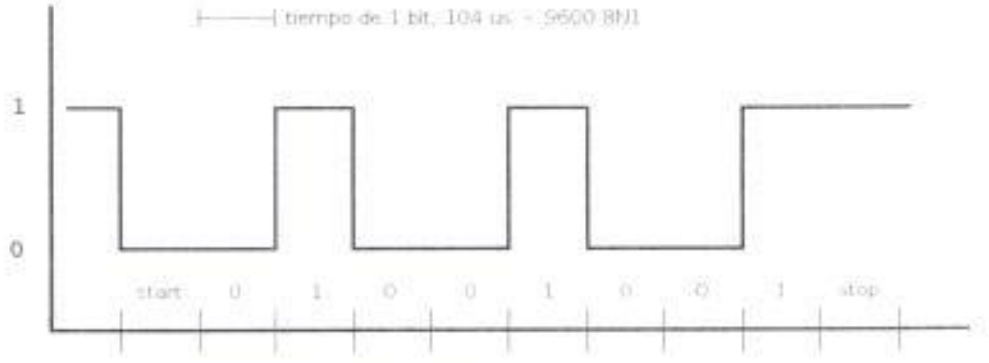


Figura 2.16 Formato de RS232 8Bits de Datos-No paridad-1bit de Stop.

En este caso se utilizará una librería UART de Mikrobasic que permitirá configurar el microcontrolador a los valores adecuado para la comunicación RS232.

Donde UART son las siglas de "Universal Asynchronous Receiver-Transmitter" (en español, "Transmisor-Receptor Asíncrono Universal"). Este controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.

Las funciones principales de chip UART son de manejar las interrupciones de los dispositivos conectados al puerto serie y de convertir los datos en formato paralelo, transmitidos al bus de sistema, a datos en formato serie, para que puedan ser transmitidos a través de los puertos y viceversa

Para el lector RFID se utilizará el formato de los bits en la comunicación serial es 8 bits de datos, no paridad, 1 BIT de stop, no invertido, bit menos significativo primero. La velocidad es 2400 bps.

2.3 LENGUAJES DE PROGRAMACION UTILIZADOS

2.3.1 VISUAL BASIC .NET

Una parte fundamental de este proyecto es la elaboración de un programa de PC que nos permita comunicarnos con el Brick del Lego Mindstorm NXT. Por lo tanto es importante elegir un lenguaje de programación que cumpla con los requisitos y facilidades para llevar a cabo las tareas necesarias para este proyecto, entre las cuales están:

- Debe poder ejecutarse en un PC

- Brindar facilidades para enviar y recibir datos a través de un puerto de comunicación.

Debido a que la mayoria de las PC actualmente vienen con el Sistema Operativo Microsoft Windows, se eligió como lenguaje de programación a Visual Basic .NET ya que cumple con los requisitos anteriormente mencionados y además ofrece una amplia fuente de información y soporte que ayudara a cumplir con nuestro objetivo de una forma segura.

Es importante establecer una base teórica del lenguaje de programación que se utilizará antes de plantear el problema especifico que se tiene que resolver para lo cual se realizará un estudio básico de este lenguaje de programación.

Visual Basic .NET es uno de los lenguajes de programación de alto nivel que funciona sobre un entorno de desarrollo multilenguaje llamado Microsoft .NET framework.

Microsoft .NET framework se compone de tres partes:

Máquina Virtual (CLR: Common Language Runtime): Procesa código escrito en un lenguaje intermedio (MSIL: Microsoft Intermediate Language).

Biblioteca de Clases: Biblioteca .NET

ASP .NET: Proporciona servicios para crear aplicaciones Web.

El entorno de desarrollo .NET incluye un programa traductor llamado compilador cuyo trabajo es tomar el código escrito en alto nivel y producir un código escrito

en un lenguaje intermedio (MSIL) común para todos los lenguajes de la plataforma .NET (C#, C++, etc.), el cual es el que entiende la máquina virtual.

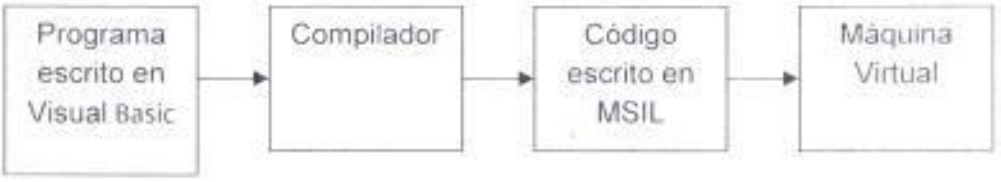


Figura 2.17 Proceso de complicacion de un programa.

El código MSIL no es el código nativo de ningún procesador; este es un lenguaje de máquina que entiende la máquina virtual de .NET que tiene funciones como tratar directamente con objetos: iniciándolos, invocando sus métodos, etc. Finalmente la máquina virtual posee un traductor de lenguaje intermedio a código máquina llamado JIT (Just In Time).

Según lo expuesto anteriormente se puede notar que para realizar un programa en Visual Basic .Net se necesitará un entorno de desarrollo que nos permita editar el programa, compilarlo, ejecutarlo y depurarlo. Microsoft proporciona uno de forma gratuita llamado .NET framework SDK, el cual se utiliza para programas que no requieran interfaz grafica.

Ya que nuestra aplicación requiere interfaz gráfica, se puede usar Microsoft Visual Basic 2008 Express Edición la cual está creada para estudiantes y programadores aficionados.

En el Apéndice B se presenta mas referencia de este tema.

2.3.2 NXC

En el proyecto se incluyen programas que debe ejecutar el bloque programable NXT autónomamente al recibir la instrucción por parte de la PC, para desarrollar estos programas existen varios lenguajes.

NXC, la abreviación de Not eXactly C, es el lenguaje escogido debido que algunas de sus caraterísticas convienen al desarrollo del proyecto:

- Puede utilizar el firmware estandar del bloque programable NXT sin necesidad de migrarlo.
- Es de libre distribucion bajo los terminos de licencia Mozilla Public License (MPL).
- El IDE llamado BricxCC (Bricx Command Center) tiene un ambiente gráfico parecido al programa Mindstorm NXT .
- Esta disponible material de soporte detallado acerca del lenguaje.

Figura 2.18 Bricx Command Center

NXC esta definido en dos partes separadas. El lenguaje NXC que describe la sintaxis a ser usada para escribir los programas y las API que describen las funciones de sistema, constantes y macros que pueden ser usadas por los programas.

En el Apéndice C se presenta más referencia de este tema.

2.3.3 MIKROBASIC

La empresa mikroElectrónica distribuye una serie de compiladores para microcontroladores, entre los que se destacan el MikroC y Mikrobasic.

Las cacterísticas mas destacadas de estos compiladores, y en particular del que nos ocupará es la inclusión de un IDE (entorno de desarrollo integrado o en

inglés Integrated Development Environment) que hace muy cómoda la programación, ya que resalta la sintaxis del lenguaje, proporciona acceso muy rápido a la excelente ayuda incluida, estadísticas sobre el uso de recursos del microcontrolador, y muchas ventajas mas.

Además, mikroElectrónica nos permite descargar una versión gratuita del compilador, que a pesar de estar limitado en la generación de código a 2Kb., es más que suficiente para muchos proyectos, y sobre todo, sirve perfectamente para que se pueda aprender el lenguaje.

Probablemente Mikrobasic sea el entorno que soporta más modelos de micros y además dispone de un enorme grupo de librerías, divididas en comunicaciones RS-232, RS-485 e I2C; teclados PS/2, conexiones USB, interfaz para LCD, y un larguísimo etc.

Respecto de la organización interna del programa, se debe saber que es necesario que las partes que componen el programa (funciones, rutinas, etc.) sigan ciertas reglas en su escritura. No es necesario que todas estén presentes.

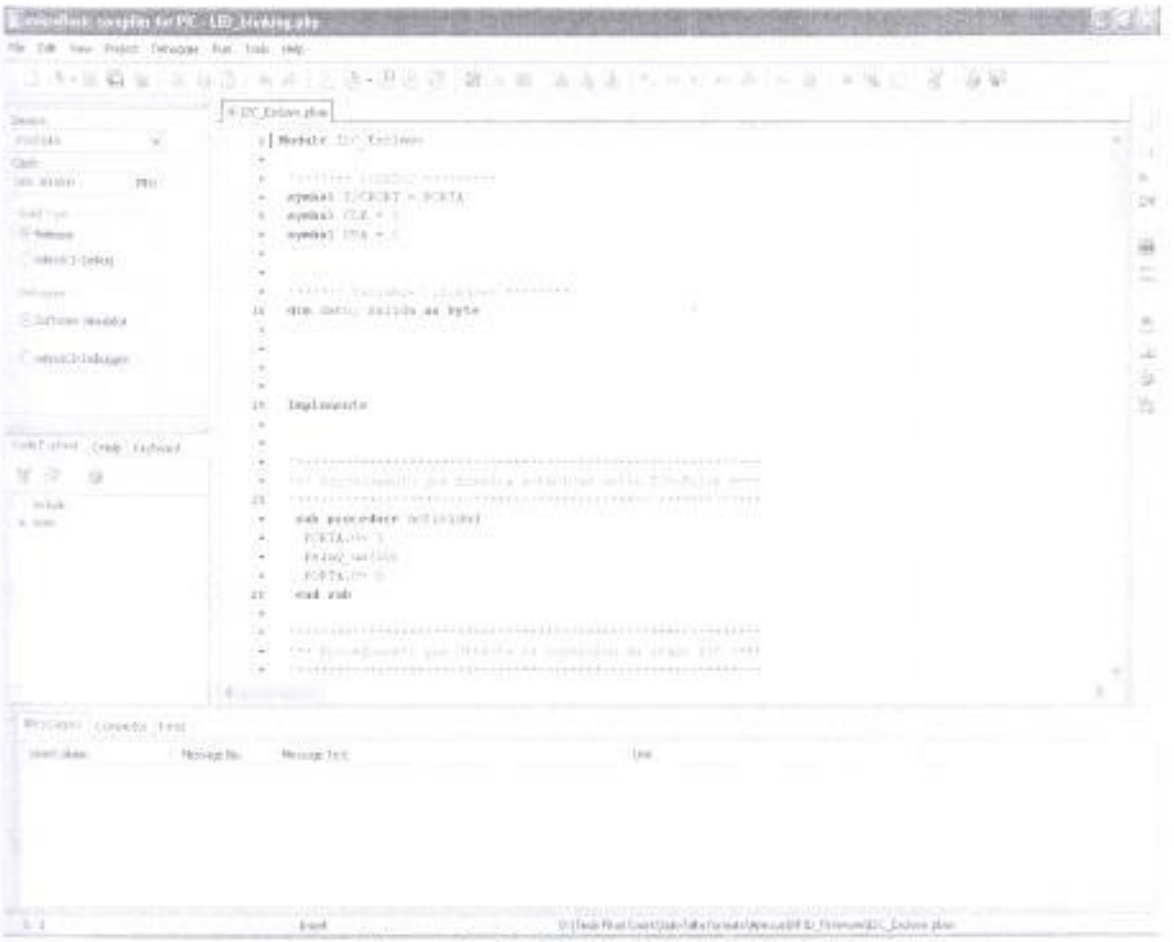


Figura.2.19 IDE de Mikrobasic.

Las secciones que son obligatorias son "program" y el bloque "main-end". Las demás, opcionales, solo se utilizaran si se las necesitan.

*program include '********** Declaraciones (globales, disponibles en todo el programa):*

*'**********

'Constantes const ...*

'variables dim ...

```
' Simbolos symbol ...
' Procedimientos  sub, procedure nombre_del_procedimiento(...)... end sub
' Funciones sub function Nombre_de_la_funcion(...) < declaraciones locales> ...
end sub
'*************************************************************
'* Programa principal: *************************************************************
main: ' Aqui escribimos nuestro código
end.
```

En el Apéndice D se presenta mas referencia de este tema.

2.4 PRINCIPALES ELEMENTOS DE HARDWARE UTILIZADOS

2.4.1 RFID

Identificación por Radio Frecuencia (RFID) es un término general para describir las tecnologías sin contacto que usan las ondas de radio para la identificación de personas y objetos. El sistema RFID se compone del Lector RFID y la etiqueta RFID. El método más común para la identificación es un número de identificación el cual es almacenado en la memoria de un chip que se encuentra en la etiqueta RFID. Existen dos tipos de etiquetas RFID las cuales son : activas y pasivas. En el caso de etiquetas pasivas, las cuales no contienen su propia fuente de energía, la forma en que trabaja este sistema consiste en que el Lector RFID contiene una antena que produce un campo magnético el cual

induce un voltaje en la etiqueta RFID el cual sirve para alimentar el chip que luego transmite via ondas de radio su número de identificación.

Las etiquetas activas poseen su propia fuente de energia, las cuales pueden transmitir a mayores distancias que las pasivas.

Los sistemas RFID usan generalmente los rangos de frecuencia siguientes: baja frecuencia (125 Khz), alta frecuencia (13.5 Mhz), ultra alta frecuencia (928 Mhz) o microondas (entre 2.45 y 5.8 Ghz).

El módulo lector RFID es una solución de bajo costo para leer etiquetas pasivas. Este módulo lector puede ser usado en una amplia variedad de aplicaciones comerciales y de Hoby incluyendo control de acceso, navegación robótica, rastreo de inventario, etc.

Entre las características importantes del lector están:

Completamente integrado, bajo costo

Interfaz de comunicación de 1 hilo, serial TTL a 2400 baudios.

Alimentación +5V DC.

Led bicolor para indicación de actividad.

0.1" de espaciado de pines para fácil integración en prototipos.

Frecuencia de operación 125 Khz.

Este módulo trabaja exclusivamente con la familia de etiquetas pasivas EM Microelectronics- Marin SA EM4100. Cada etiqueta contiene un identificador único de 2^{10} combinaciones posibles.

Este módulo electrónico tiene los siguientes pines que se muestran en la Figura 2.20.

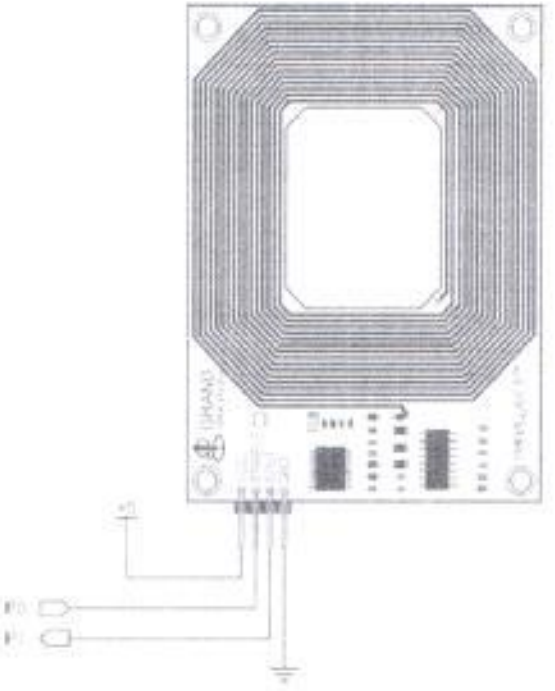


Figura 2.20 Conexiones electrónicas de lector RFID.

El módulo es controlado con un simple nivel lógico en bajo en el pin /Enable. Cuando es puesto en nivel bajo, el módulo entra en un estado activo y habilita la antena para interrogar las etiquetas. La corriente de consumo se incrementa dramáticamente cuando el módulo esta activo.

PIN	NOMBRE	TIPO	FUNCION
1	VCC	PODER	Fuente de energía +5VDC
2	/ENABLE	ENTRADA	En estado bajo habilita el lector RFID y activa la antena
3	SOUT	SALIDA	Salida serial TTL 2400 bps, 8 bits de datos, no paridad, 1 bit de parada
4	GND	TIERRA	Tierra

Tabla 2.1 Descripción de los pines del lector RFID.

Una indicación visual del estado del módulo RFID es dado por el LED. Cuando el módulo esta satisfactoriamente alimentado y esta en un estado de inactividad, el LED estará en verde. Cuando el LED esta en estado activo y la antena esta transmitiendo, el LED estará en rojo.

La lectura será más eficiente si la etiqueta se encuentra en forma paralela y en frente o posterior de la antena. El uso de más de dos etiquetas puede causar colisión y confundir al lector. La distancia aproximada a la que se debe colocar las etiquetas aproximadamente a 7,5 cm. Las condiciones pueden variar de acuerdo al tamaño de la etiqueta y las condiciones ambientales.

Cuando la etiqueta esta colocada en el rango de actividad del lector, el único ID será transmitido como una cadena ASCII 12-Byte via TTL pin SOUT con el siguiente formato.

Byte de Inicio 0X0A	ID 1	ID 2	ID 3	ID 4	ID 5	ID 6	ID 7	ID 8	ID 9	ID 10	Byte de Parada 0x0D

Figura 2.21 Trama que contiene el ID de la etiqueta RFID.

El byte de Start y Stop son usados para identificar que la cadena correcta ha sido recibida.

El formato de los bits en la comunicación serial es 8 bits de datos, no paridad, 1 BIT de stop, no invertido, bit menos significativo primero. La velocidad es 2400 bps y no puede ser cambiada.

En la Tabla 2.2 se describe las caracteristicas eléctricas del Lector RFID.

PARAMETRO	SIMBOLO	CONDICIONES DEL TEST	ESPECIFICACIONES			UNIDAD
			MIN.	NORMAL	MAX.	
FUENTE DE VOLTAJE	V_{CC}	---	4.5	5.0	5.5	V
FUENTE DE VOLTAJE, INACTIVA	I_{CCS}	---	---	10	---	mA
FUENTE DE VOLTAJE, ACTIVA	I_{CC}	---	---	50	---	mA
ENTRADA VOLTAJE NIVEL BAJO	V_{IL}	+4.5V <= V_{CC} <= +5.5V	---	---	0.8	V
ENTRADA VOLTAJE NIVEL ALTO	V_{IH}	+4.5V <= V_{CC} <= +5.5V	2.0	---	---	V
SALIDA DE VOLTAJE NIVEL BAJO	V_{OL}	V_{CC} = +4.5V	---	---	0.6	V
SALIDA DE VOLTAJE NIVEL ALTO	V_{OH}	V_{CC} = +4.5V	V_{CC} - 0.7	---	---	V

Tabla 2.2 Especificaciones eléctricas de funcionamiento del lector RFID de PARALLAX.

2.4.2 DS1820

El DS1820 es un termómetro digital de alta precisión que se comunica mediante un bus 1-Wire. Su rango de medición de temperatura esta entre -55 °C a +125 °C y tiene una precisión de ± 0,5 °C sobre el rango de -10 °C a +85 °C. Además consta de un número de identificación única e inalterable que viene programado de fábrica, el cual consta de 64 bits.

Entre las características más importantes se tiene:

Se comunica a través de un bus 1-Wire.

Número de identificación único de 64 bits.

Comunicación multipunto, lo cual simplifica el muestreo de valores de temperatura en aplicaciones distribuidas.

Puede ser alimentado desde la línea de datos. Rango de alimentación entre 3 a 5.5 V.

± 0,5 °C sobre el rango de -10 °C a +85 °C.

Termómetro de 9 bits de resolución.

Convierte temperatura en 200 ms (máx.).

Configuración de alarmas no volátiles definidas por el usuario.

Comandos para identificar dispositivos cuya temperatura esta fuera de los límites configurados por el usuario.

Es utilizada en aplicaciones de control termostático, sistemas industriales, termómetros y cualquier sistema sensible a la temperatura.

En la Figura 2.22 se muestra la configuración de los pines de los encapsulados comunes del DS1820.

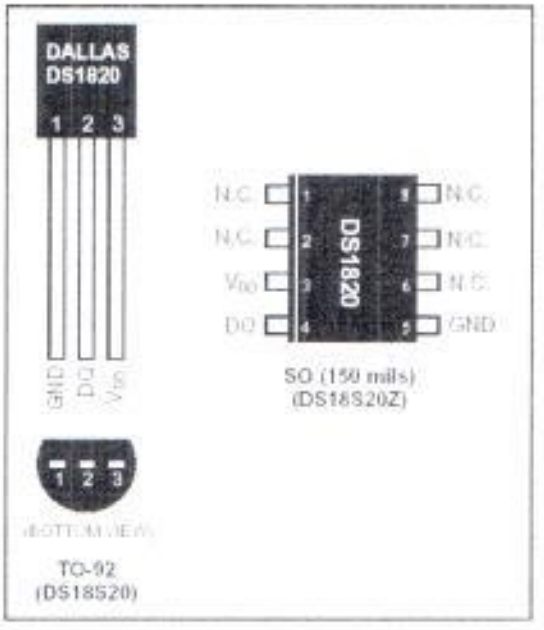


Figura 2.22 Configuración de los pines del DS1820

PIN	NOMBRE	FUNCION
1	GND	Tierra
2	DQ	E/S de datos
3	VDD	Vdd opcional. Debe estar puesta a tierra cuando opera en modo de alimentación parasita

Tabla 2.3 Descripción funcional de los pines del DS1820

En la Figura 2.23 se muestra el diagrama de bloques del DS1820.

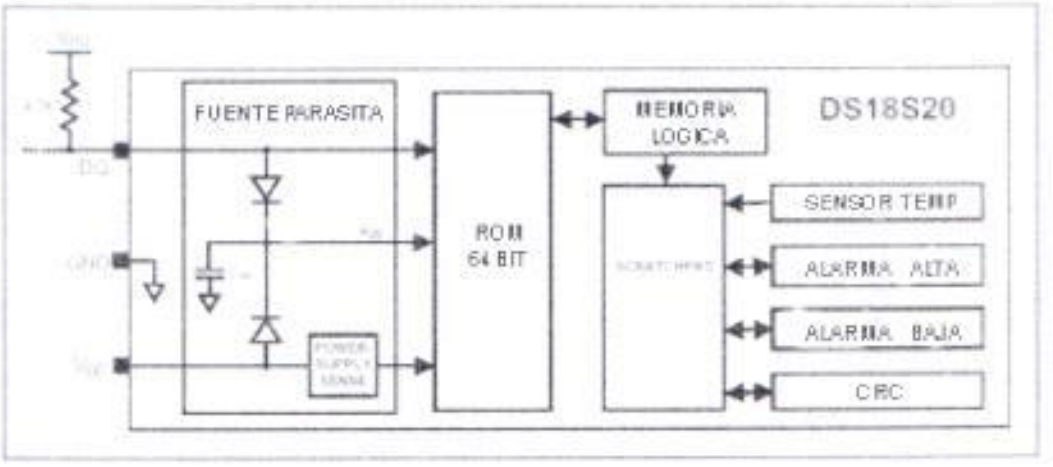


Figura 2.23 Arquitectura interna del DS1820.

La ROM de 64 bits almacena el código único que identifica al dispositivo.

La memoria Scratchpad contiene el registro de temperatura de 2 bytes que almacena la salida digital de la temperatura.

La memoria Scratchpad también provee acceso a los 2 registros que almacenan tanto el byte superior como inferior de la temperatura de alarma, los cuales son

registros no volátiles (EEPROM) y mantienen el dato a pesar que el dispositivo no este energizado.

El DS1820 utiliza el bus 1-Wire para lo cual la línea de control requiere una resistencia de pull-up, lo que adicionalmente le permite operar sin una fuente externa. La energía es suministrada a través de la resistencia de pull up vía el pin DQ cuando el bus esta en nivel alto.

Esta señal alta también carga al capacitor interno Cpp, el cual suministra energía cuando el bus está en nivel bajo. Este método de suministrar energía se conoce como "alimentación parásita".

La funcionalidad principal del DS1820 es su sensor de temperatura digital, el cual tiene una resolución de 9 bits, con pasos de 0,5 °C. El DS1820 se enciende en un estado de bajo poder de consumo llamado "idle".

Para iniciar la medición de la temperatura y la conversión A/D el maestro debe indicarlo usando el comando "Convertir [44h] ".

Después de la conversión el resultado del dato es almacenado en el registro de temperatura de 2 bytes ubicado en la memoria Scratchpad y el DS1820 retorna a su estado "idle" (inactividad).

Los resultados de salida de datos del DS1820 están en grados centigrados. El dato de temperatura se almacena en un formato de número de 16 bits, con signo y complemento a dos como se muestra en la Figura 2.24.

Los bits de signo se indican con la letra "S". Para números positivos S = 0 y para números negativos S = 1.

Figura 2.24 Formato de LS byte y MS byte.

La Tabla 2.4 muestra la relación entre la salida digital y su temperatura correspondiente.

Temperatura	Código de salida	
+125 °C	00000000	11111010
+25,5 °C	00000000	00110011
+25 °C	00000000	00110010
+0,5 ° C	00000000	00000001
+0 °C	00000000	00000000
-0,5 ° C	11111111	11111111
-25 °C	11111111	11001110
-55 °C	11111111	0010010

Tabla 2.4 Relación entre la salida digital y su temperatura correspondiente.

El valor por default cuando el DS1820 se enciende (Reset) es +85 °C (AAh).

Se puede observar que el bit de menos peso indica el decimal de la temperatura leída, es decir: xx.0 ° C este bit es "0" y para xx.5 °C este bit es "1".

El valor de la temperatura está especificado en los 7 bits de mayor peso del byte bajo por ejemplo para +125 °C se tiene los 2 bytes: 00000000 11111010 de los cuales se extraen 1111101 como valor de la temperatura lo cual es 125 en decimal.

Para las temperaturas negativas como por ejemplo -25 °C se tienen los 2 bytes: 11111111 1001110 de los cuales se extraen 1100111 el cual esta en complemento a 2, por lo tanto ya que:

$$C_2^N = 2^n - N$$

n = 7 (Digitos)

1100111 = 103 d

103 d = 128 d – N => N = 25 d

Después que el DS1820 realiza la conversión de temperatura, el valor obtenido se compara con los dos valores de alarma almacenados en los registros Th y Tl con se muestra en la Figura 2.25.

Figura 2.25 Formato de los registros TH y TL

El bit 7 es el bit de signo. Si S=0 el número es positivo y S=1 el número es negativo.

Para la comparación sólo se utilizan los bits desde el bit 8 hasta el bit 1 del registro de temperatura. Si la temperatura medida es menor que o igual a Tl o mayor que Th, se presentara una condición de alarma (Bandera de alarma) en el DS1820. La bandera de alarma es actualizada con cada medición de temperatura.

El dispositivo maestro puede chequear el status de la bandera de alarma de todos los Ds1820s en el bus usando el comando [ECh].

La memoria del DS1820 esta organizada como se muestra en la Figura 2.26.

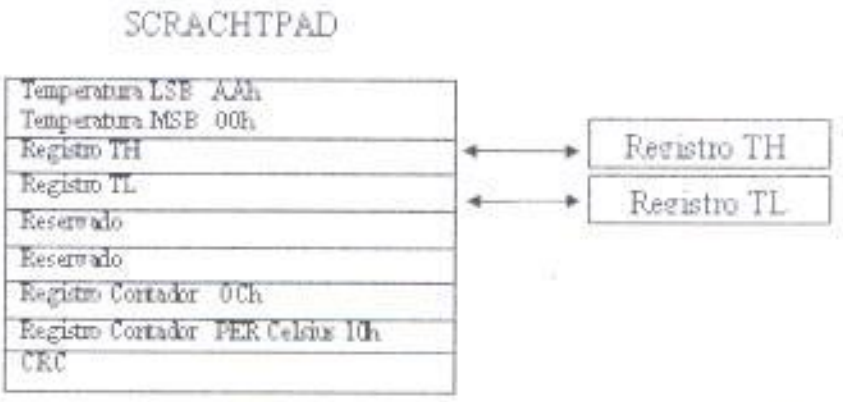


Figura 2.26 Estructura interna de la memoria Scratchpad

La memoria Scratchpad es de tipo SRAM con una porción tipo EEPROM utilizada para los registros de alarma Th y Tl.

El byte 0 y el byte 1 de la scratchpad contiene el LSB y MSB del registro de temperatura. Byte 2 y Byte 3 contienen los registros Th y Tl respectivamente. El Byte 4 y el Byte 5 están reservados para uso interno del dispositivo y no pueden ser sobrescritos.

El byte 6 y el byte 7 contienen valores que pueden ser usados para calcular resultados de resolución extendida.

El byte 8 es de sólo lectura y contiene el código CRC para bytes 0 hasta el 7 de la scratchpad.

El dato del valor de la temperatura es escrito en el byte 2 y 3 usando el comando [4Eh]; el dato debe ser transmitido al DS1820 empezando con el bit menos significativo del byte 2. Para verificar la integridad del dato, el scratchpad puede ser leído usando el comando [BEh]. Cuando se lee el scratchpad el dato es transferido al bus empezando con el bit menos significativo del byte 0. Para transferir el dato Th y Tl desde el scratchpad a la EEPROM el maestro debe enviar el comando [48h].

El dato puede ser recargado desde la EEPROM al scratchpad en cualquier momento usando el comando [B8h].

Las secuencias de transacción de acceso al DS1820 son:

Inicialización

Comandos ROM

Comandos de función del DS1820

Es muy importante seguir esta secuencia cada vez que se accede al DS1820. La excepción a la regla son los comandos Search ROM [F0h] y Alarm Search [ECh]. Después de estos comandos el maestro debe retornar al paso 1 en la secuencia.

Inicialización: Consiste de un pulso de reset del maestro del bus seguido de un pulso de presence transmitido por el esclavo.

Comandos ROM: Después de que el maestro ha detectado el pulso de presence, este puede continuar con los comandos ROM los cuales operan sobre el código único de 64 bits en la ROM de cada esclavo y permiten al maestro seleccionar un dispositivo especifico de los esclavos presentes en el bus. Además permiten al maestro determinar cuántos y que tipo de dispositivos están presentes en el bus y si alguno de los dispositivos ha presentado una condición de alarma.

Search ROM [F0h]: Este comando sirve para identificar los códigos ROM de todos los dispositivos esclavos conectados al bus, permitiendo determinar el número de esclavos y sus tipos. Este comando es usado generalmente cuando hay más de un esclavo conectado al bus.

Read ROM [33h]: Este comando es usado cuando existe un solo esclavo conectado al bus. Este comando permite leer el código de 64 bits sin usar el comando Search ROM.

Match ROM [55h]: El comando Match ROM seguido por un código de 64 bits ROM permite al bus maestro direccionar a un esclavo especifico en un bus multipunto.

Skip ROM [CCh]: El maestro puede usar este comando para gestionar todos los esclavos conectados al bus simultáneamente sin enviar el código ROM. Por ejemplo cuando se quiere enviar un comando que indique que todos los dispositivos realicen la misma tarea.

Alarm Search [ECh]: Es similar al comando Search ROM excepto que sólo los esclavos con una condición de alarma responderán.

Comando de función: Después que se han usado los comandos ROM para ubicar el dispositivo esclavo con el que se desea comunicarse, el maestro puede usar los comandos de función. Estos comandos permiten al maestro escribir y leer desde la memoria ScratchPad, iniciar conversiones de temperatura y determinar el modo de la fuente de energía.

Convert T [44h]: Este comando sirve para iniciar una simple conversión de temperatura. Posterior a la conversión, el resultado del dato termal es almacenado en un registro de temperatura de 2 bytes en la memoria ScratchPad, luego de lo cual el DS1820 regresa a un estado de bajo consumo. Si el DS1820 es alimentado por una fuente externa, después del comando de conversión, el DS1820 responderá transmitiendo el 0 mientras continúa en el proceso de conversión de temperatura y 1 cuando la conversión está hecha. En el modo parásito esta técnica de notificación no puede ser usada.

Write ScratchPad [4Eh]: Este comando permite al maestro escribir los 2 bytes del dato de temperatura en la memoria ScratchPad. El primer byte es escrito en el registro Th, y el segundo byte es escrito en el registro Tl.

Read ScratchPad [BEh]: Este comando permite al maestro leer el contenido del ScratchPad.

Copy ScratchPad [48h]: Este comando copia el contenido de la memoria ScratchPad Th y Tl a la EEPROM.

Recall E2 [B8h]: Este comando invoca los valores Th y Tl desde la EEPROM y colocarlos en la memoria ScratchPad.

Read Power Suply [B4h]: Sirve para saber el modo de alimentación del DS1820.

2.4.3 PIC16F84

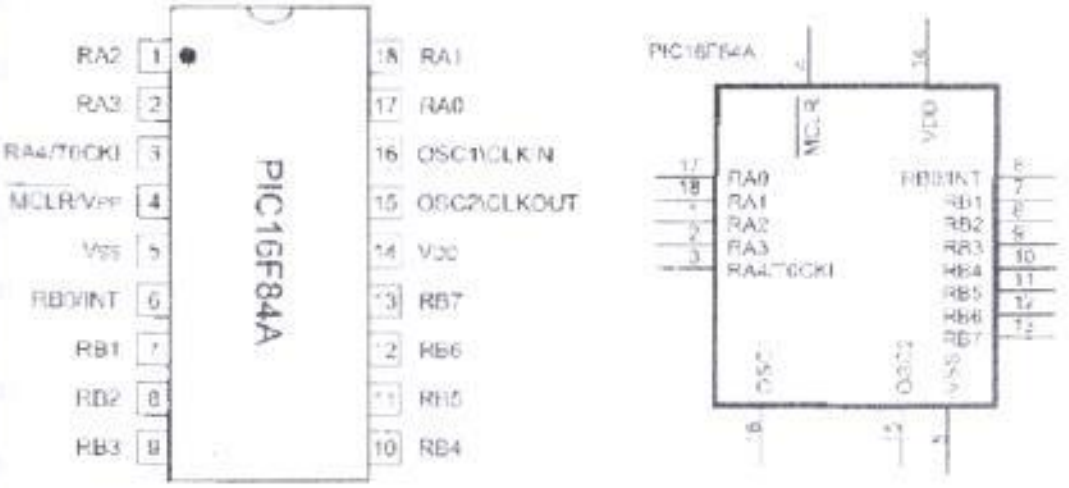
Se trata de un **microcontrolador** de 8 bits,18 pines, y un set de instrucciones RISC. Es un PIC de gama baja, cuyas caracteristicas se puede resumir en:

- **Memoria** de 1K x 14 de tipo **Flash**
- **Memoria** de datos **EEPROM** de 64 bytes
- 13 líneas de E/S con control individual
- Frecuencia de funcionamiento máxima de 10 Mhz.
- Cuatro fuentes de interrupción:

- Activación de la patita RB0/INT
- Desbordamiento del TMR0
- Cambio de estado en alguna patilla RB4-RB7
- Fin de la escritura de la **EEPROM** de datos

- Temporizador/contador TMR0 programable de 8 bits
- Perro Guardián o WatchDog

El encapsulado utilizado fue el DIP18. A continuación puede apreciarse en la Figura 2.27 dicho encapsulado y una breve descripción de cada una de las patitas.



Encapsulado DIL-18 Símbolo

Figura 2.27 Tipos de encapsulado del PIC 16F84

- VDD: alimentación

- VSS: masa
- OSC1/CLKIN-OSC2/CLKOUT: conexión del oscilador
- VPP/MCLR: tensión de programación y reset
- RA0-RA3: líneas de E/S de la puerta A
- RA4: línea de E/S de la puerta A o entrada de impulsos de reloj para TMR0
- RB0/INT: línea de E/S de la puerta B o petición de interrupción
- RB1-RB7: líneas de E/S de la puerta B

El microcontrolador PIC 16F84 puede trabajar con una frecuencia máxima de 10 MHz La versión avanzada PIC16F84A-20 puede llegar hasta los 20Mhz.

Normalmente el PIC 16F84 se alimenta con 5 voltios aplicados entre los pines Vdd y Vss que son , respectivamente, la alimentación y la masa del chip, el consumo de corriente para el funcionamiento depende de la tensión de alimentación, de la frecuencia de trabajo y de las cargas que soportan sus salidas, siendo del orden de unos pocos miliamperios.

Las líneas digitales de entrada / salida que trabajan entre 0 y 5 V. Los puertos se pueden configurar como entradas o como salidas

El PIC 16f84 tiene dos puertos:

EL puerto A con 5 líneas, pines RA0 a RA4.

El puerto B con 8 lineas, pines RB0 a RB7

Cada linea puede ser configurada como entrada o como salida, independiente unas de otras, según se programe. Las lineas son capaces de entregar niveles TTL cuando la tensión de alimentación en VDD es de 5V. La máxima capacidad de corriente de cada de ella es 25mA, cuando el pin está a nivel bajo es decir cuando consume corriente (modo Sink).

Sin embargo, la suma de las intensidades por las 5 lineas del puerto A no puede exceder de 80mA, ni la suma de 8 lineas del puerto B puede exceder de 150mA.

20 mA, cuando el pin esta a nivel alto, es decir, cuando proporciona corriente (modo source). Sin embargo, la suma de las intensidades por la 5 lineas del puerto A no pueden exceder de 50 mA, ni la suma de las 8 lineas del puerto B puede exceder de 100mA.

Todo microcontrolador requiere de un circuito que le indique la velocidad de trabajo, es el llamado oscilador o reloj. Este genera una onda cuadrada de alta frecuencia que se utiliza como señal para sincronizar todas las operaciones del sistema.

CAPITULO 3
DISEÑO DEL SOFTWARE, HARDWARE Y FIRMWARE DEL SISTEMA.

3.1 LECTOR RFID.

3.1.1 DISEÑO DEL HARDWARE DEL LECTOR RFID

Primeramente hay que identificar y definir los componentes necesarios para construir este módulo y los sistemas que van a interactuar con el mismo, lo cual se representará en el la Figura 3.1.

ETIQUETA DE RFID: Para este diseño se puede utilizar etiquetas RFID pasivas, es decir que vienen con un ID de fábrica el cual es único y sólo permiten lectura.

LECTORA RFID.- Se debe implementar una lectora RFID debido a que se utilizará etiquetas RFID pasivas, las cuales sólo permiten lectura. Adicionalmente esta lectora RFID debe poder ser montada en un sistema embebido y utilizar algún protocolo de comunicación que le permita interactuar con un microcontrolador.

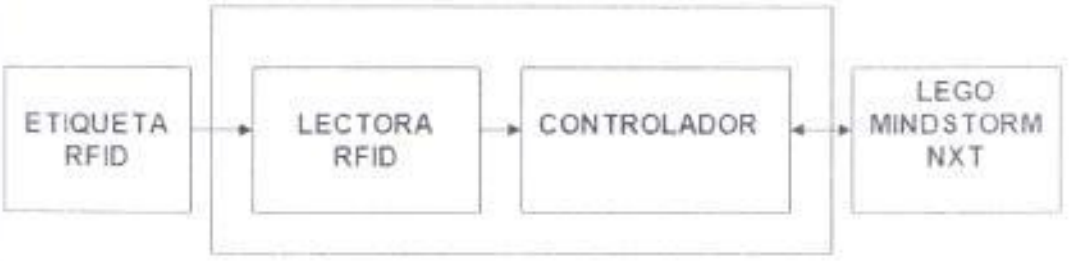


Figura 3.1 Diagrama del módulo lector RFID.

CONTROLADOR.- En términos generales el controlador debe interactuar con la lectora RFID y el controlador del Lego Mindstorm NXT, para lo cual será necesario utilizar un microcontrolador.

LEGO ® MINDSTORM NXT.- Se debe entender la forma en que se comunica el controlador del Lego Mindstorm NXT y el hardware necesario para este fin.

Para la elección de componentes se considera lo siguiente:

ELECCION DE LECTORA RFID: Después de investigar las opciones en el mercado de lectoras RFID se ha elegido una lectora RFID comercializada por

una empresa llamada Parallax (Parallax RFID Reader #28140) por los siguientes motivos:

1. Calidad /precio
2. Puede ser montada fácilmente en un sistema empotrado.
3. Consumo de energía reducido

Figura 3.2 Lector RFID Parallax #28140.

ELECCION DE ETIQUETAS RFID: La elección de las etiquetas quedó determinada por la compatibilidad con la lectora utilizada para lo cual se utilizó etiquetas pasivas de la familia EM4100, las cuales son vendidas por la misma empresa (Parallax) que vende el lector. (Figura 3.3).

ELECCION DEL CONTROLADOR: Como controlador se ha elegido el PIC16F84 ya que es un microcontrolador popular, de bajo costo y que ofrece los

recursos necesarios para la realización de esta interfase en cuanto a memoria de programa, puertos de E/S disponibles, velocidad de trabajo, etc. (Figura 3.4).

Figura 3.3 Etiqueta RFID Parallax

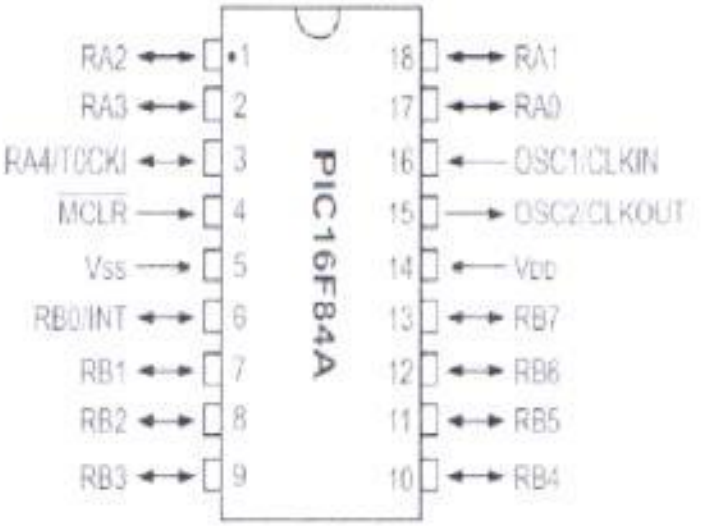


Figura 3.4 Microcontrolador PIC16F84A de Microchip.

A continuación se describe la técnica del funcionamiento del lector RFID (Figura 3.5), la primera determinación técnica sobre el lector RFID es su correspondiente patillaje y dimensiones:

- Este lector trabaja con etiquetas RFID de la familia EM4100 de solo lectura las cuales tienen un identificador único (uno de 2^{40}).
- Tiene un led bicolor para la indicación de actividad.

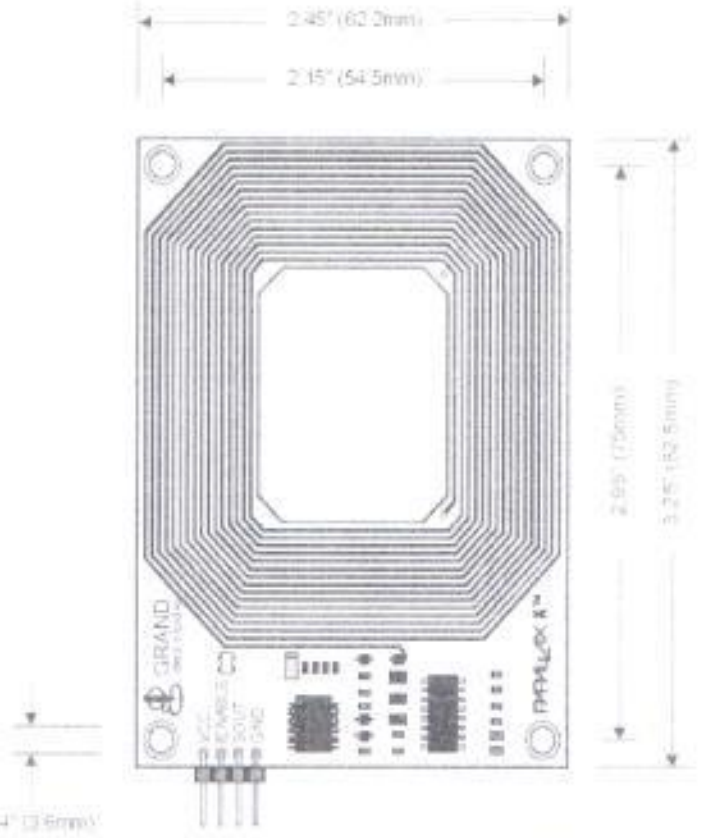


Figura 3.5 Especificaciones del hardware del lector RFID Parallax #28140.

- La descripción de los pines se la muestra en la Tabla 3.1.

PIN	NOMBRE	FUNCION
1	VCC	+5VCC
2	ENABLE	En nivel bajo habilita el lector y activa la antena
3	SOUT	Serial: 2400 bps, 8 bits de datos, No paridad, 1 bit stop
4	GND	

Tabla 3.1 Descripción funcional de los pines del lector RFID Parallax #28140.

- Cuando el nivel del pin ENABLE es bajo, el módulo entra en estado activo y habilita la antena para interrogar las etiquetas.
- Una indicación visual del estado del módulo RFID es dada por un LED, el cual cuando está satisfactoriamente energizado y está en un estado IDLE (INACTIVO), el LED será verde.
- Cuando el módulo está en estado activo y la antena está transmitiendo el LED será rojo.
- Solo un transpondedor debe leerse a la vez.
- Las etiquetas de Parallax tienen una distancia de lectura de aproximadamente 7,5 cm.
- Cuando un transpondedor válido es colocado dentro del rango de actividad (7.5 cm) del lector, El ID será transmitido como una cadena de 12 byte ASCII vía SOUT con el formato de la Figura 3.6.

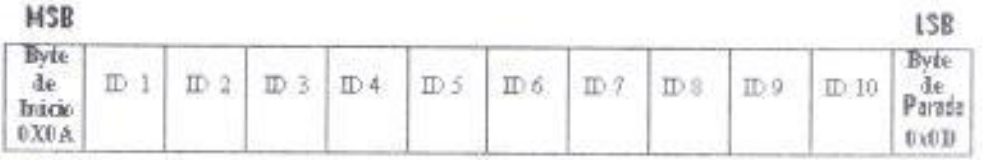


Figura 3.6 Formato del ID transmitido vía SOUT del lector RFID Parallax #28140

- 0X0A y 0X0D son usados para identificar que una cadena correcta ha sido recibida y corresponden a los caracteres LINE FEED y CARRIAGE RETURN respectivamente.
- La comunicación RS232 con datos de 8 bits, 2400 bps, no paridad, 1 bit Stop, No-invertido y bit menos significativo primero no puede ser cambiado.
- Consumo de corriente (Tabla 3.2).

IDLE:	10 mA
ACTIVE:	90 mA

Tabla 3.2 Consumo de corriente del lector RFID Parallax #28140

- El lector RFID está diseñado para baja frecuencia: 125 Khz

Para el diseño final del módulo lector RFID, primeramente se necesita familiarizarse con el funcionamiento del Lector RFID para lo cual se implementará y se analizará el funcionamiento del Lector RFID, es decir se estará en la siguiente sección del diseño final:

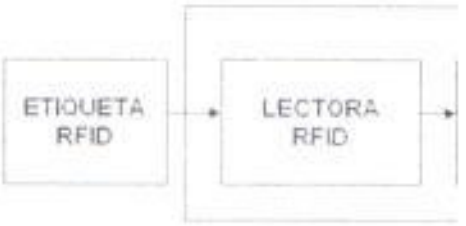


Figura 3.7 Diagrama parcial de la sección de comunicación lector RFID-controlador.

En la segunda fase del diseño, se incluirá el Lego Mindstorm NXT para lo cual se tiene que investigar la sección de hardware del Lego que va a interactuar con el controlador. (Figura 3.8).

Básicamente la sección que se debe estudiar en el Lego Mindstorm NXT, que va a interactuar con el controlador, es el puerto de entrada el cual se explicará a continuación en la Figura 3.9.

Figura 3.8 Diagrama parcial de la sección de comunicación lector controlador-NXT.

Lego Mindstorm NXT tiene 4 puertos de entrada a los cuales van conectados los sensores. Posee una interfase de 6 hilos que posibilita comunicación tanto digital como analógica.

Tanto los puerto 1, 2, 3 y 4 tienen un esquemático idéntico al que se presentará a continuación salvo que en el puerto 4 los pines DIG1A10 y DIG1A11 están conectados a una interfaz RS485.

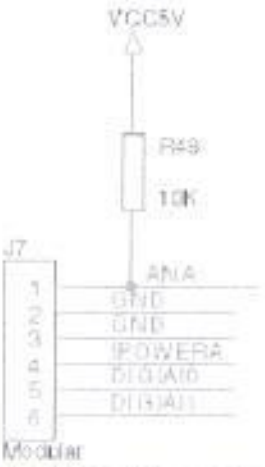


Figura 3.9 Diagrama esquemático del puerto del Lego MindStorm NXT.

ANA: El pin 1 (ANA) es un pin de entrada analógica que esta conectado a un convertidor A/D de 10 bits dentro del procesador AVR. Adicionalmente también esta conectado a un generador de corriente usado para proporcionar energía a los sensores.

IPOWERMA: Es una fuente de poder que está conectada a todos los puertos de salida y entrada, la cual proporciona un voltaje de 4.3 V y cuya máxima corriente de salida que puede manejar es 180 mA, lo cual significa que cada puerto tiene aproximadamente 20 mA.

DIG1A10 y DIG1A11: Son usados para comunicación digital implementada usando el protocolo I2C a 9600b/s. El NXT sólo puede funcionar como maestro

en la comunicación I2C y requiere que los dispositivos externos incluyan las resistencias Pull-up.

Como se puede observar en la explicación anterior los pines que se utilizará para la comunicación con el controlador son los pines 5 y 6 (DIG1A10 y DIG1A11) los cuales utilizan el protocolo I2C, como se indicaba anteriormente.

En las especificaciones hardware del producto se especifica el diagrama interno de estos circuitos, los cuales se muestran en la Figura 3.10.

Como se puede observar las resistencias de pull-up que se utilizan para el bus I2C son de 82 KΩ

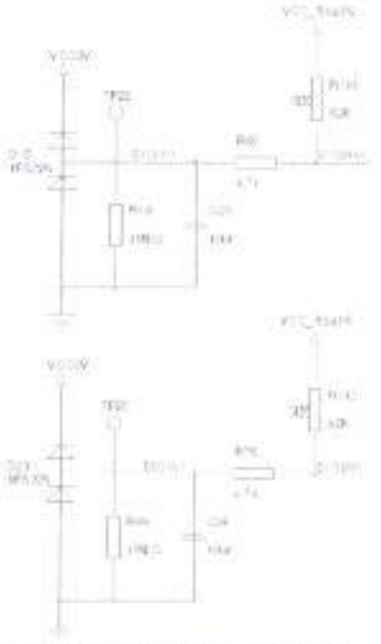


Figura 3.10 Configuración interna del puerto del Lego MindStorm NXT.

Finalmente considerando las caracteristicas de hardware, tanto del módulo de la lectora RFID y de la interfaz con el Lego Mindstorm NXT, el paso siguiente es la integración de la partes, quedando el diagrama de bloques de la forma que se muestra en la Figura 3.11.

Se puede agregar un botón de Reset y un led para indicar el estado del controlador.(Figura 3.12).

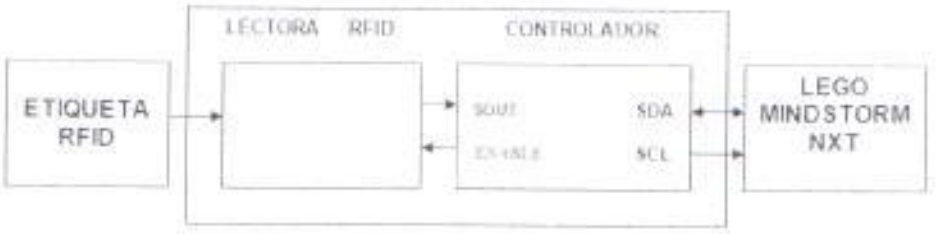


Figura 3.11 Diagrama de bloques identificando los pines de comunicación del controlador.

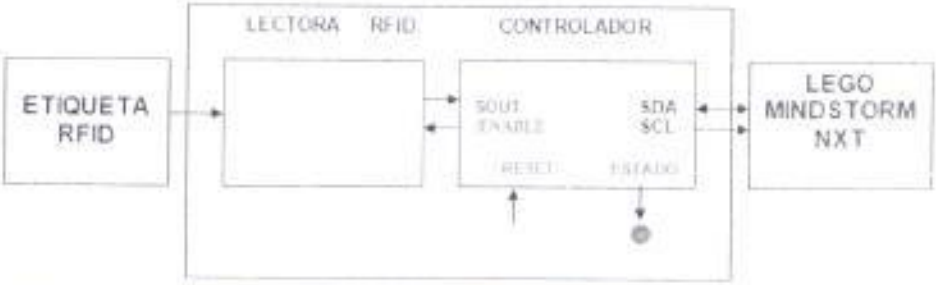


Figura 3.12 Diagrama de bloques final identificando los pines de reset y estado del controlador.

Una vez definidos los puertos que interactúan con el controlador se puede diseñar el diagrama esquemático, el cual se describe a continuación (Figura 3.13).

- Los pines que interactúan con la Lectora RFID se los colocará en la puerta B.
- Los pines para la comunicación I2C se los colocará en la puerta A.

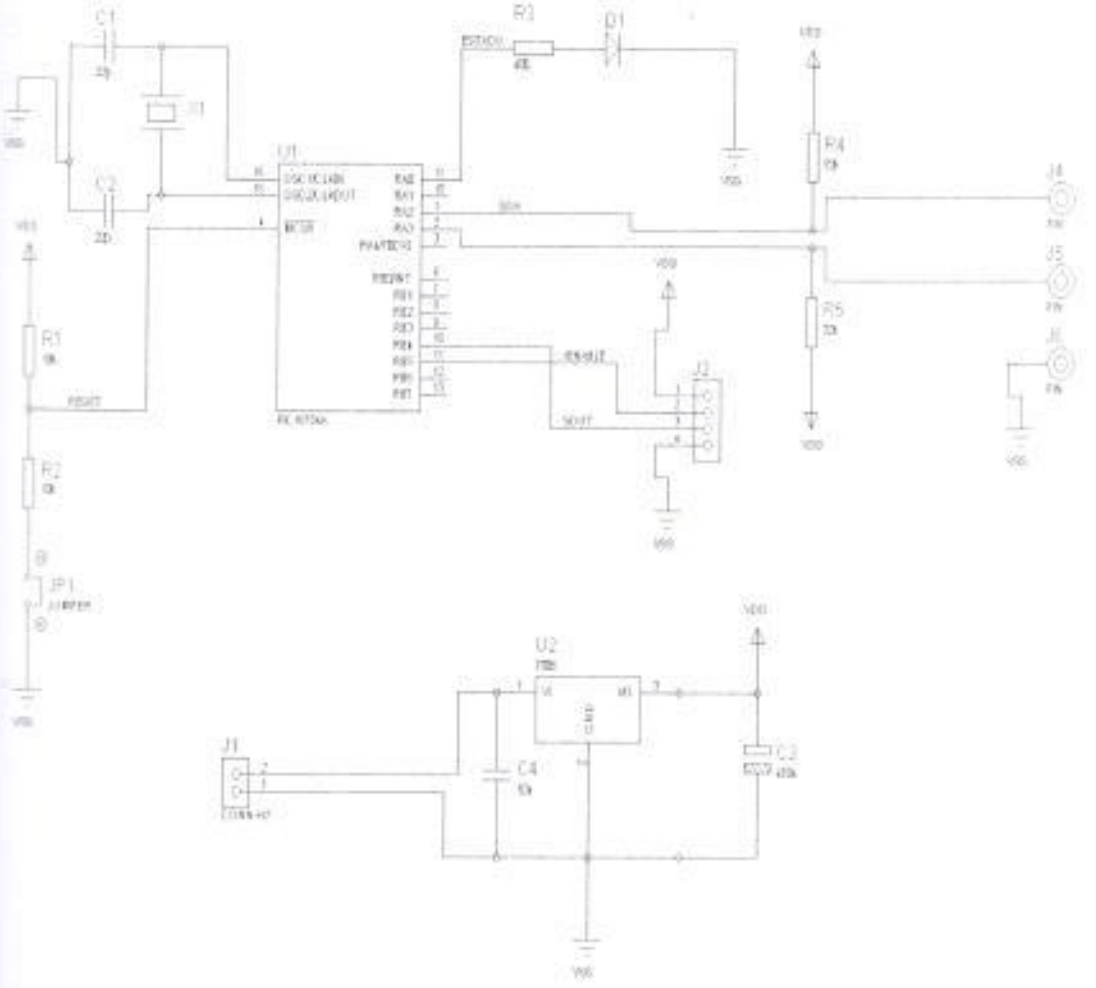


Figura 3.13 Diagrama esquemático del módulo de lector RFID.

3.1.2 DISEÑO DEL FIRMWARE DEL LECTOR RFID

Antes de empezar el diseño del firmware se debe plantear el problema y visualizar las variables que van a interactuar, para lo cual se mostrará los requerimientos generales del sistema.

Se detalla a continuación los requerimientos generales del sistema:

- El Lego Mindstorm NXT indica al controlador por medio de un comando (LEER), que lea la etiqueta RFID. (Figura 3.14).

Figura 3.14 Sistema de comunicación NXT-módulo lector RFID comando "LEER"

- El Lego Mindstorm NXT indica al controlador por medio de un comando (TRANSFERIR), que transfiera el dato (ID de la etiqueta) almacenado de la última lectura.

Figura 3.15 Sistema de comunicación NXT-módulo lector RFID comando "TRANSFERIR"

- Por defecto el sistema debe estar inactivo. En estado Inactivo la lectora RFID debe estar en modo de bajo consumo
- Los puertos SDA y SCL se utilizan para comunicar el sistema con el Lego Mindstorm NXT por medio del protocolo I2C.

Las acciones del controlador son las siguientes:

- El lector RFID puede estar en dos estados que son : Inactivo y Activo
- En estado Inactivo la lectora RFID debe estar en modo de bajo consumo, es decir IDLE, para lo cual se debe tener el pin /Enable en estado alto.
- En estado Activo el módulo debe interrogar a la etiqueta RFID, para lo cual se debe poner el pin /Enable en estado bajo.
- El dato a través de SOUT se recibe por medio del protocolo RS232.
- El controlador debe verificar el ID con un sistema de control de errores
- Los puertos SDA y SCL se utilizan para comunicar el controlador con el Lego Mindstorm NXT por medio del protocolo I2C.
- El LED indicador de estado sirve para indicar si el controlador esta listo, para recibir instrucciones por parte del Lego Mindstorm NXT.
- El dato obtenido de la lectura de la etiqueta RFID debe ser almacenado en la memoria EEPROM del controlador.

Desarrollo de una rutina para leer una etiqueta RFID y almacenar su ID en la memoria EEPROM, donde el planteamiento del problema es el siguiente:

- El controlador debe almacenar los 10 Bytes de Identificación de la etiqueta RFID, extrayéndolos de la trama de la figura 3.6

Para la resolución del problema se comienza con la espera del Byte de Start que inicia la trama, luego de lo cual se debe capturar los Bytes de Datos. Si es válido se almacenará, incrementando un contador que almacene el número de Bytes de Datos recibidos y cuando este llegue a 10 lecturas confirmar el Byte de Stop.

En la Figura 3.16 se presenta la máquina de estado finito:

- Estados:

  Esperar Inicio de Trama

  Captura de Trama

  Confirmar Byte de Stop

- Evento:

  Byte Entrante

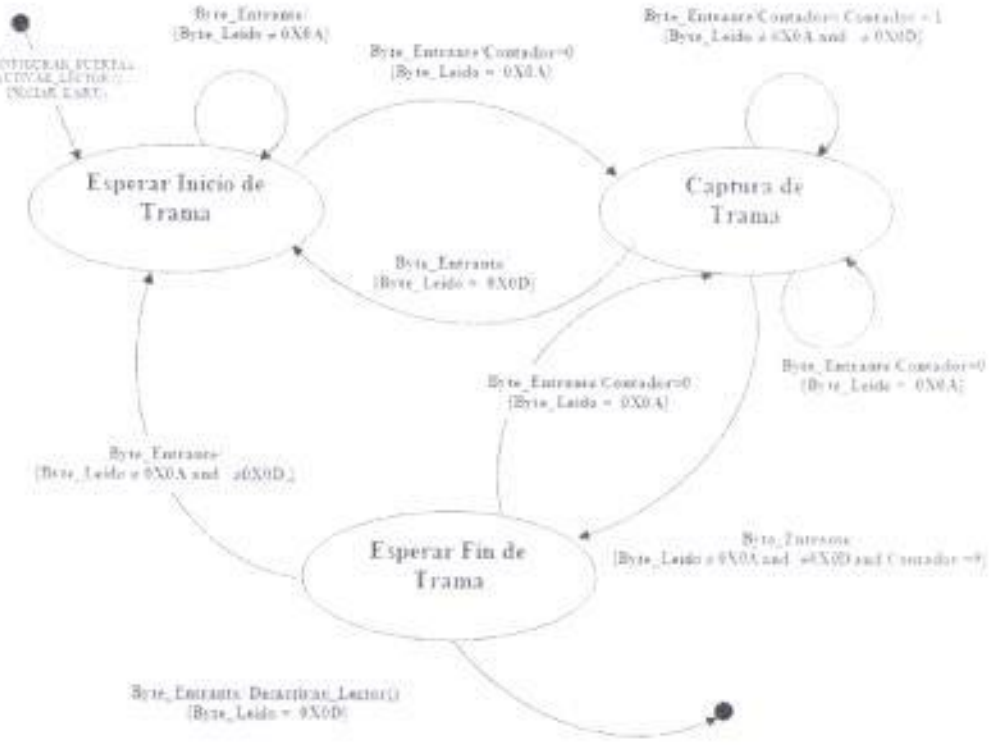


Figura 3.16 Diagrama de máquina de estado finito para la rutina de leer una etiqueta RFID y almacenar su ID en la memoria EEPROM.

En la Figura 3.17 se presenta el Diagrama de Flujo el mismo que presenta las sentencias case anidadas para representar la concepción anterior.

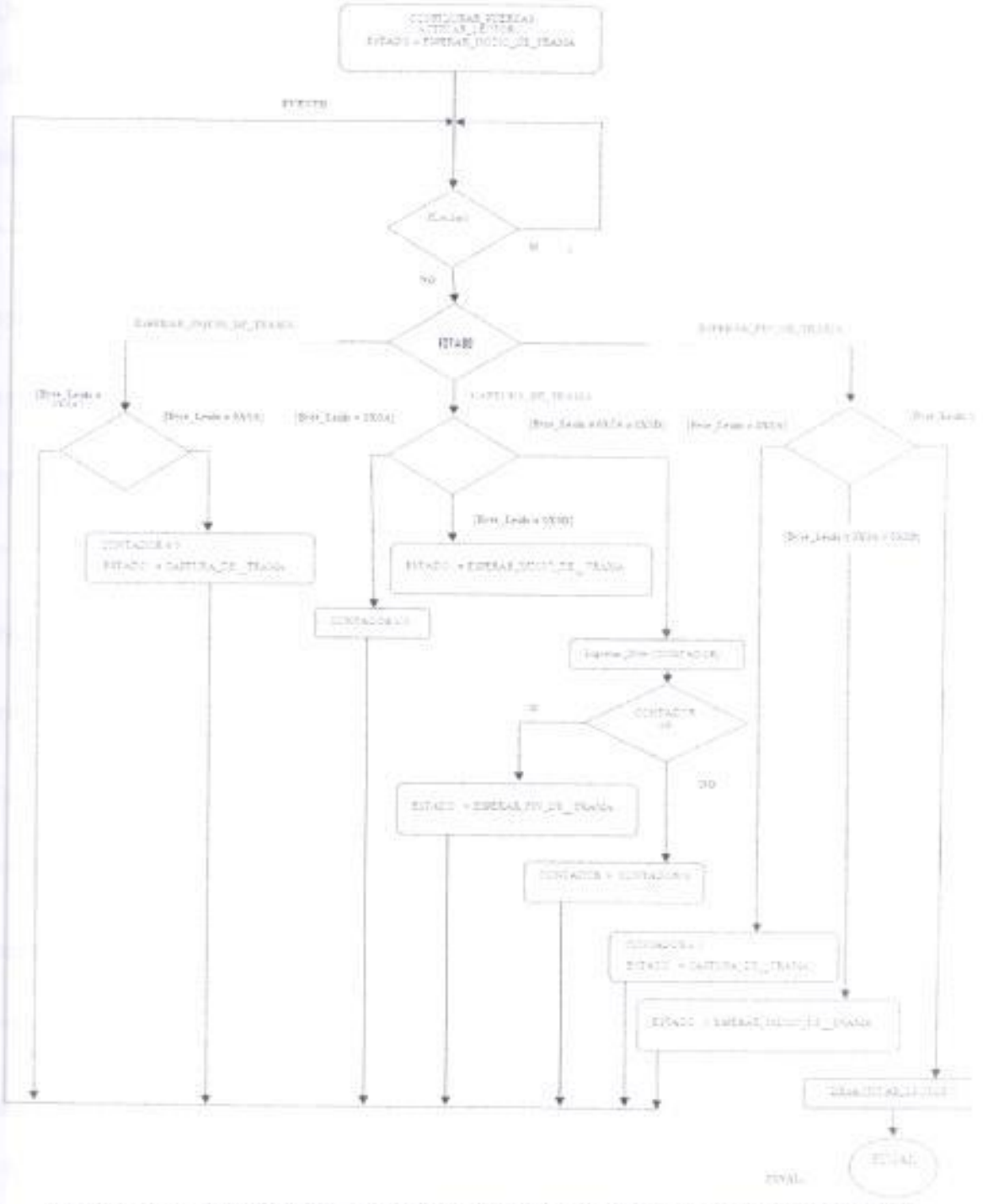


Figura 3.17 Diagrama de flujo para la rutina de leer una etiqueta RFID y almacenar su ID en la memoria EEPROM

Una vez realizado el diagrama de flujo se procede ha implementar el código utilizando el lenguaje Mikrobasic.

```
Module RFID_I2C

'****** ESTADOS ******

symbol ESPERAR_INICIO_DE_TRAMA = 0

symbol CAPTURA_DE_TRAMA        = 1

symbol ESPERAR_FIN_TRAMA       = 2

'****** PUERTOS ******

symbol Enable = PORTB.5                    'Pin Enable

symbol SOUT  = 4                           'Pin SOUT de recepción UART

symbol TX      = 6                         'Pin de transmisión UART

'****** CONSTANTES ******

const TRAMA_COMPLETA = 10                  'Longitud máxima de la trama

'****** VARIABLES ******

dim CONTADOR       as byte                 'Almacena el número de bytes
capturados
```

```
dim BYTE_LEIDO      as byte          'Almacena el Byte leido por el
Puerto SOUT

dim ESTADO          as byte          'Almacena los valores de los
estados

dim FLAG            as byte          'Bandera necesaria para la
función UART

dim TRAMA           as byte [10]     'Almacena la trama que esta
siendo capturada

Implements

******* PROCEDIMIENTOS ******

sub procedure Activar_Lector ( )     'Activa el lector RFID

Enable = 0

end sub

sub procedure Desactivar_Lector()    'Desactiva el lector RFID

Enable = 1

end sub

sub procedure Iniciar_UART()         'Inicializa el Puerto UART
```

```
Soft_Uart_Init(PortB,SOUT,TX,2400,0)

end sub

sub procedure Ingresar_Byte (dim puntero as byte) 'Ingresa los bytes capturados
en el arreglo

 TRAMA[puntero] = BYTE_LEIDO                    'Trama

end sub

sub procedure Guardar_Trama ()                  'Guarda la trama que se
encuentra en el

                                                'arreglo TRAMA en la EEPROM

dim i as byte

 for i=0 to 9

 EEPROM_Write (i, TRAMA[i])

 next i

end sub

sub procedure lecturaRFID                       'Procedimiento de lectura de
trama RFID

'************ FSM ************
```

```
'*****************************

TRISB = %00000000                    'Configura la Puerta B como
salidas

Activar_Lector()                     'Activa el Lector RFID

ESTADO =  ESPERAR_INICIO_DE_TRAMA    'Estado por Default

Iniciar_UART()                       'Inicializo parámetros UART

'***** EVENTO *****

evento:

FLAG = 1                             'Inicializo la bandera

do

 BYTE_LEIDO = Soft_Uart_Read (FLAG)  'Si recibo un byte la bandera se
pone en 0

loop until FLAG = 0

FLAG = 1

Select case  ESTADO                  'Case para selección de
Estados

    case ESPERAR_INICIO_DE_TRAMA
```

```
    Select case BYTE_LEIDO                    'Case para determinar
condiciones

        case $0A

        CONTADOR = 0

        ESTADO  = CAPTURA_DE_TRAMA

        case else

    end select

case CAPTURA_DE_TRAMA

    Select case BYTE_LEIDO

        case $0A

        CONTADOR = 0

        case $0D

        ESTADO  = ESPERAR_INICIO_DE_TRAMA

        case else

        Ingresar_Byte(CONTADOR)

        if (CONTADOR = 9) then
```

```
        ESTADO = ESPERAR_FIN_TRAMA
        else
         CONTADOR = CONTADOR + 1
        end if
        end select
        case ESPERAR_FIN_TRAMA
        Select case BYTE_LEIDO
        case $0A
         CONTADOR = 0
         ESTADO = CAPTURA_DE_TRAMA
         case $0D
         Desactivar Lector()
         goto final
         case else
         ESTADO = ESPERAR_INICIO_DE_TRAMA
    end select
```

```
end select

goto evento

final:

end sub

end.
```

Luego de codificar la rutina que captura el ID contenido en la trama de salida del Lector RFID y almacenarla en la variable Trama por medio del procedimiento lectura RFID, se puede continuar con las rutinas necesarias para controlar el sistema en general.

Otras rutinas necesarias para la comunicación entre el controlador y el Lego Mindstorm NXT es el manejo del protocolo I2C. Mikrobasic solamente tiene implementadas librerías I2C en modo maestro, y debido a que el controlador del Lego Mindstorm NXT funciona en modo maestro, es necesario que el controlador del módulo RFID funcione como esclavo, para lo cual se debe diseñar e implementar las rutinas I2C en modo esclavo.

Primeramente se debe señalar que para el intercambio de información se procede a trabajar con dos líneas que son: SDA (System Data) y SCL (System Clock).

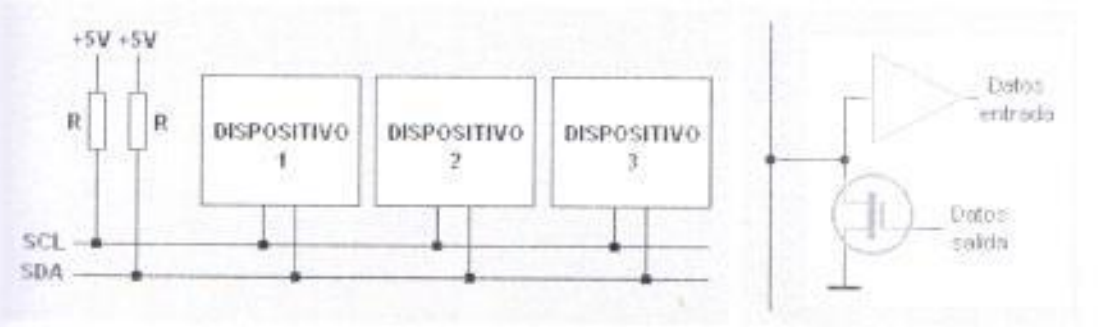


Figura 3.18 Diagrama del bus I2C

La señal SCL marca los pulsos de reloj y SDA se utiliza para la transferencia de datos.

Otro factor importante es la velocidad a la que va a trabajar el bus, la cual queda determinada por el maestro, en este caso el Lego Mindstorm NXT, la cual es 9600 bits/s.

Se debe entender la forma en que trabaja el protocolo, para lo cual se describirá la forma en que trabaja:

- Cuando las lineas están inactivas (**Bus Libre**) se encuentran en un nivel lógico alto.
- Solo los dispositivos maestros pueden iniciar la comunicación.
- El maestro debe indicar el inicio de la comunicación por medio de una señal de **Condición de Inicio,** la cual consiste en poner en bajo la línea SDA, mientras esta en alto la línea SCL como se mostrara en LA Figura 3.19.

Figura 3.19 Diagrama de secuencia de inicio

- Después de la condición de inicio se transmite el primer Byte por parte del maestro. En este primer Byte, los 7 primeros bits, componen la dirección del dispositivo que se desea seleccionar, y el octavo bit indica si se quiere leer o escribir el dispositivo esclavo.
- Después de que el maestro envía el primer Byte, indicando la dirección del dispositivo y la operación a ejecutarse (Lectura o Escritura), el esclavo requerido responde en el noveno bit, el cual es llamado **Bit de Reconocimiento (ACK)** .Si el esclavo responde en bajo, significa que este reconoce la solicitud y que esta listo para aceptar otro Byte.

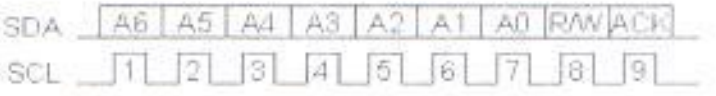


Figura 3.20 Diagrama del byte de dirección y operación R/W.

- El bit de lectura/escritura (R/W) si se encuentra en estado bajo "0" lo cual significa "escritura", el maestro enviará datos al dispositivo esclavo. Si R/W

es igual a "1" significa "lectura", el dispositivo maestro recibirá datos del esclavo.

- Cuando el maestro se encuentra recibiendo datos, luego de cada byte recibido, el maestro debe generar un pulso de reconocimiento.
- El maestro para dejar libre el bus, debe generar una Condición de Parada. Para producir una condición de parada, el maestro debe generar un flanco de subida en la linea SDA mientras la linea SCL se encuentra en estado alto.
- Si el maestro desea seguir transmitiendo, el maestro puede generar otra condición de inicio, la cual se podria emplear para direccionar un dispositivo esclavo diferente o para alterar el bit de lectura escritura.

Figura 3.21 Diagrama de la secuencia de parada.

- Los datos se transfieren en secuencias de 8 bits los cuales se transmiten empezando por el bit de más peso.

Después de presentar las señales más importantes en el Bus I2C se puede ir creando procedimientos implementados mediante Mikrobasic. Rutina que detecta la Condición de Inicio.

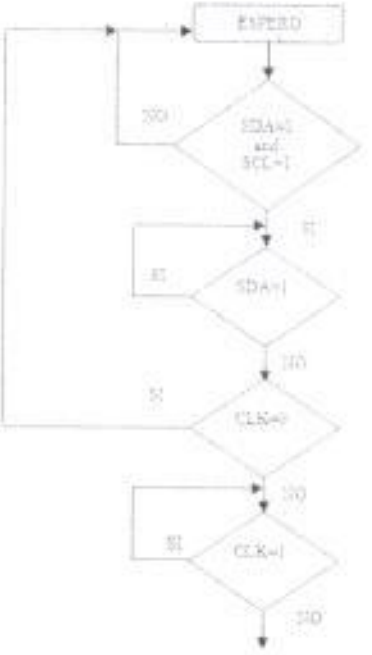


Figura 3.22 Diagrama de flujo la secuencia de inicio

```
'**********************************************

''' Procedimiento que detecta la condición de start I2C '''''

'**********************************************

 sub procedure esperostart

espero:
```

```
   while (( I2CPORT.CLK =1) and (I2CPORT.SDA =1)) = 0 'Mientras CLK y SDA
sean diferentes
   wend                                                    'al mismo tiempo a
"1" mantenerse
   while (I2CPORT.SDA =1)                                  'Mientras SDA sea
"1" mantenerse
   wend
   if ( I2CPORT.CLK =0) then                               'Si CLK esta en nivel
bajo ir a espero
   goto espero
   end if
   while ( I2CPORT.CLK =1)                                 'Mientras CLK sea
"1" mantenerse
   wend
end sub
```

Rutina que envía un Ack al maestro:

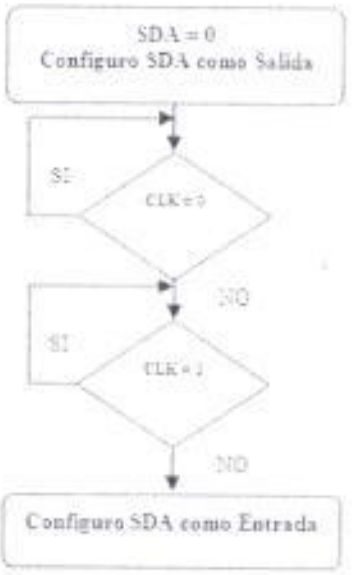


Figura 3.23 Diagrama de flujo la secuencia Ack al maestro.

```
'*****************************************************
'***     Procedimiento que envia un ack al Maestro     ****
'*****************************************************

sub procedure ponerack

  I2CPORT.SDA = 0              'Coloco SDA en bajo

  TRISA.SDA = 0                'Configuro SDA como salida

  while ( I2CPORT.CLK =0)      'Mientras CLK esta en bajo mantenerse

  wend
```

```
while ( I2CPORT.CLK =1)     'Mientras CLK está en alto mantenerse
wend
TRISA.SDA = 1               'Coloco SDA en alto
end sub
```

Rutina para capturar el dato enviado por el Maestro y enviar Ack:

Primeramente se debe tener una variable donde se almacenará el dato capturado y se puede utilizar un parámetro de entrada al procedimiento que indique el número de bits que se quieren capturar.

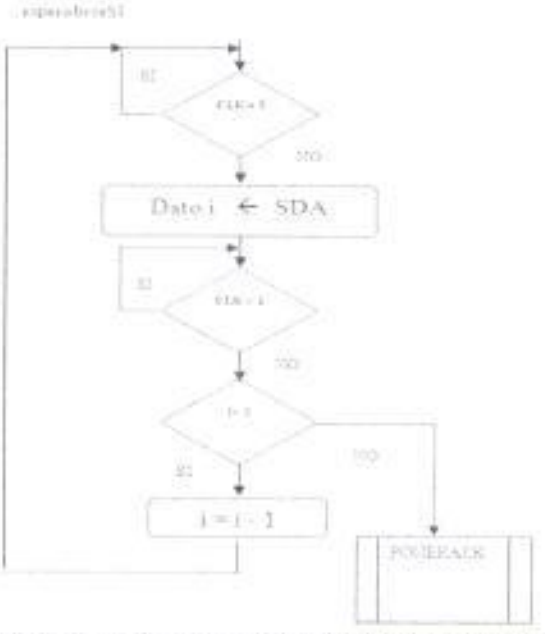


Figura 3.24 Diagrama de flujo de la secuencia capturar i-bits del dato enviado por el maestro y devolver ack

```
'*****************************************************************
'**      Procedimento que espera un dato de i+1 bits del Maestro y **
'**      cuando recupera el dato dentro de la variable "dato"      **
'**                                  envia un ack al Maestro       **
'*****************************************************************
sub procedure esperobyte(dim i as byte)
 esperobyteS1:
 while ( I2CPORT.CLK =0)                    'Mientras CLK=0 mantenerse
 wend
 dato.i = I2CPORT.SDA                       'Guardar bit "i" en dato
 while ( I2CPORT.CLK =1)                    'Mientras CLK=1 mantenerse
 wend
```

```
  if (i > 0) then                    'Si bit "i" es mayor que cero
decrementar "i"
  i = i – 1                          'e ir a etiqueta esperobyteS1
  goto esperobyteS1
  end if
  ponerack                           'Enviar un Ack
end sub
```

Rutina para leer un Bit de Reconocimiento del maestro:

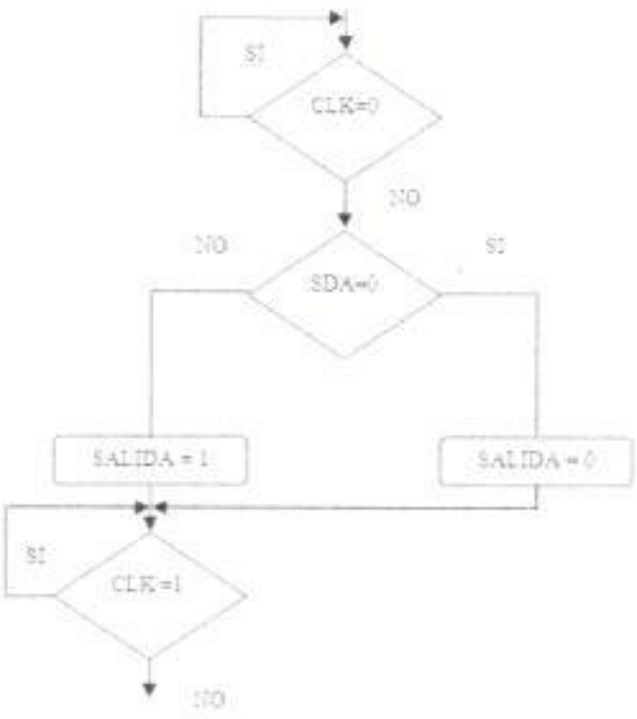


Figura 3.25 Diagrama de flujo para leer un Bit de Reconocimiento del maestro

```
'*************************************************************

'***          Procedimiento que obtiene un ack del Maestro      ****

'***          Si Maestro envia Ack entonces la variable "salida=0"  ****

'***          Si Maestro envia Nack entonces la variable "salida=1" ****

'*************************************************************
```

```
sub procedure obtenerack

  while ( I2CPORT.CLK =0)        'Mientras CLK sea cero esperar

  wend

  if (I2CPORT.SDA =0) then       'Si SDA=0 entonces salida=0 pero si SDA=1

  salida = $00                    'entonces salida=1

  else

  salida = $01

  end if

  while ( I2CPORT.CLK =1)        'Mientras CLK sea alto esperar

  wend

end sub
```

Rutina para enviar un Byte al maestro:

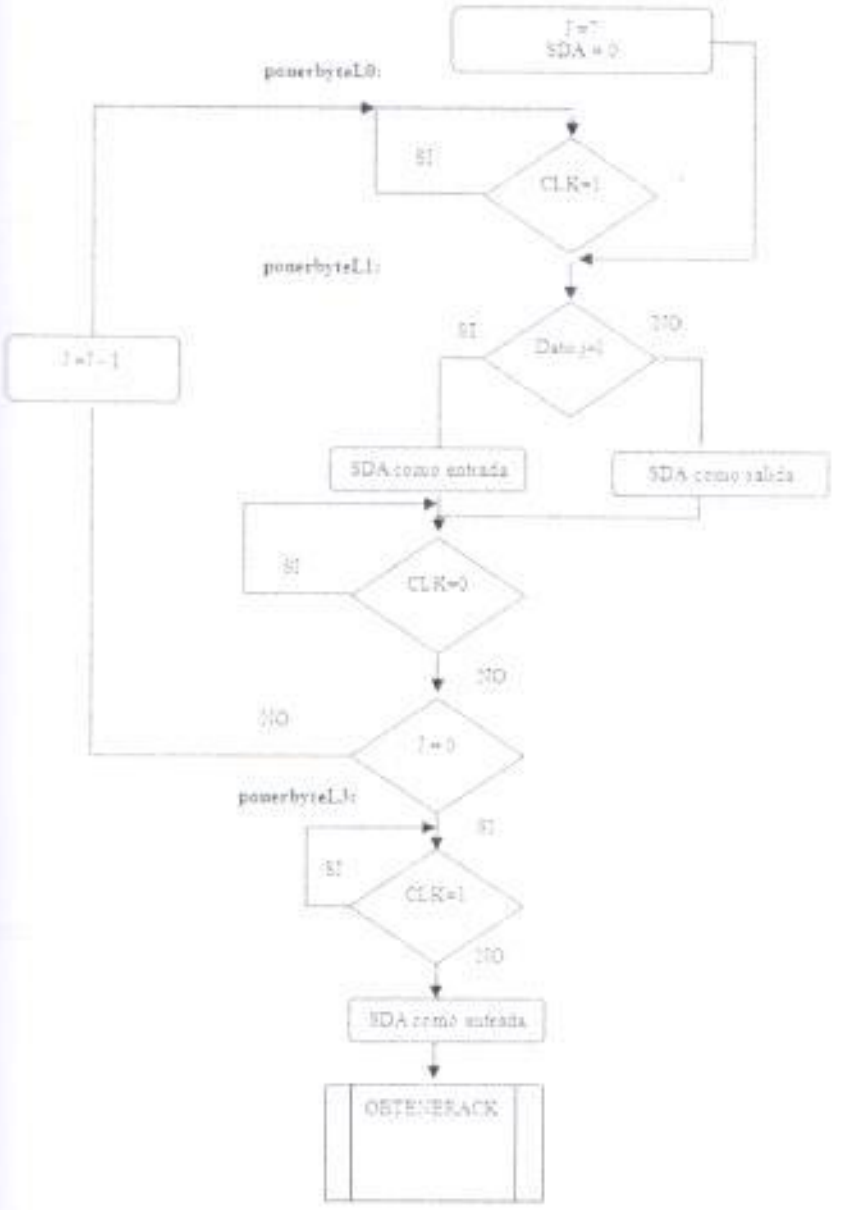


Figura 3.26 Diagrama de flujo para enviar un Byte al maestro

```
'*****************************************************
'***     Procedimiento que envia un byte al Maestro el cual      ****
'***     se  encuentra en la variable "dato"                     ****
'*****************************************************
    sub procedure ponerbyte
    dim j as byte                   'Variable utilizada para contar bits
    j =7
    I2CPORT.SDA =0                  'Prepar SDA en bajo
    goto ponerbyteL1
ponerbyteL0:
    While (I2CPORT.CLK =1)          'Mientras CLK sea alto esperar
    wend
ponerbyteL1:
```

```
    if (dato.j =1) then                 'Si valor del bit es "1" configur como entrada
SDA
    TRISA.SDA =1                        'con lo cual la resistencia de Pull-Up pone en
alto SDA
    else
    TRISA.SDA =0                        'Si valor del bit es "0" configurar como salida
SDA
    end if                              'con lo cual la linea SDA será "0" ya que
previamente fue puesto
    while ( I2CPORT.CLK =0)             'Mientras CLK sea cero esperar
    wend
    if (j=0) then                       'Si j=0 se ha terminado de enviar Byte
    goto ponerbyteL3
    else
    j=j-1
    goto ponerbyteL0
```

```
    end if
ponerbyteL3:
    while ( I2CPORT.CLK =1)        'Mientras CLK sea alto esperar
    wend
    TRISA.SDA = 1                  'Dejar SDA como salida
    Obtenerack                     'Procedimiento Obtener Ack del Maestro
end sub
```

Rutina que detecta la condición de Stop o de Start del maestro (Figura 3.27)

```
'*************************************************************************
'***          Procedimiento que detecta la condición de Stop o de Start del
Maestro     ****
'***          Si Maestro envia Stop entonces la variable "estado=3" y sale
****
'***          Si Maestro envia Start entonces la variable "estado=4" y sale
'***        Si no detecta Start ni Stop entonces espera un nuevo dato
'*************************************************************************
```

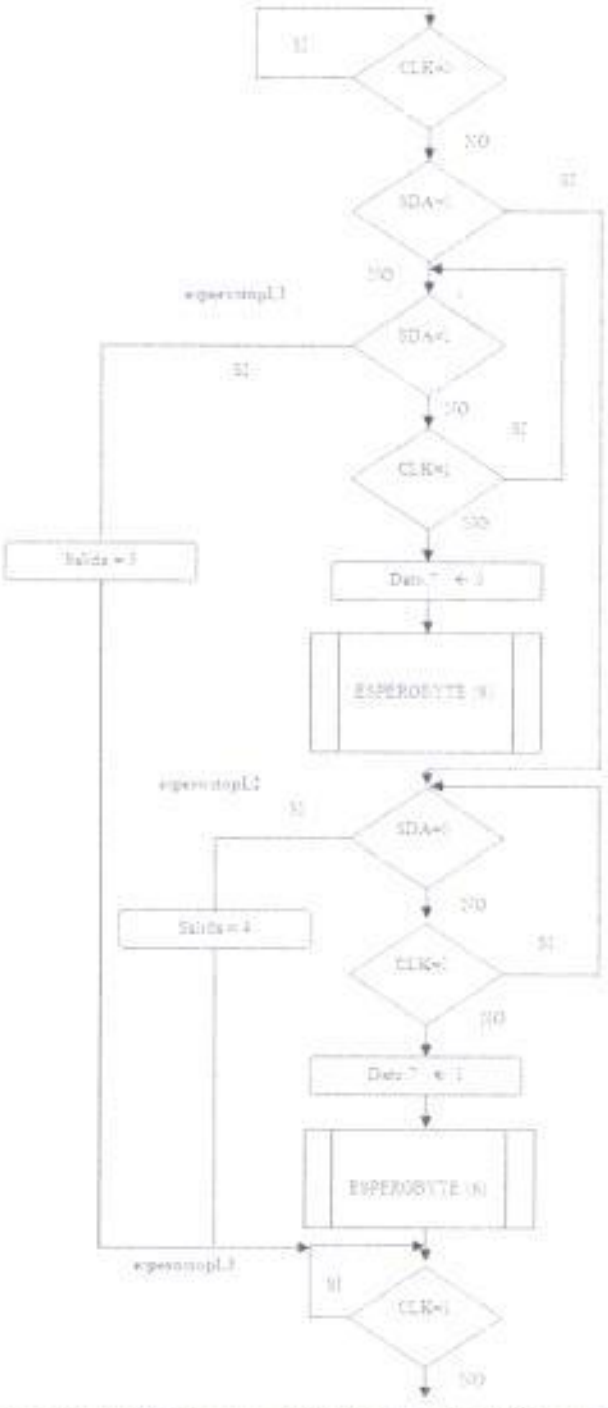


Figura 3.27 Diagrama de flujo que detecta la condición de Stop o de Start del maestro.

```
sub procedure esperostop

while ( I2CPORT.CLK =0)       'Mientras CLK sea cero esperar

wend

if (I2CPORT.SDA =1) then        'Si SDA es alto ir a rutina para byte de
start

goto esperostopL2

end if

esperostopL1:                   'Sección de Byte de Stop o Datos


if (I2CPORT.SDA =1) then      'Stop detectado entonces Salir

salida = $03

goto esperostopL3

end if

if  (I2CPORT.CLK  =1)  then        'Si  CLK  esta  alto  seguir  esperando
condición de Stop
```

```
  goto esperostopL1                'Si CLK se pone bajo entonces es un
Dato con SDA=0
  end if
  dato.7=0                         'Bit 7 fue "0"
  esperobyte (6)                   'Recibir 7 Bits restantes

esperostopL2:                      'Sección de Byte de Start o Datos

  if (I2CPORT.SDA =0) then         'Start detectado entonces Salir
  salida = $04
  goto esperostopL3
  end if
  if  (I2CPORT.CLK =1) then        'Si CLK esta alto seguir esperando
condición de Start
  goto esperostopL2                'Si CLK se pone bajo entonces es un
Dato con SDA=1
```

```
  end if
  dato.7=1                    'Bit 7 fue "1"
  esperobyte(6)               'Recibir 7 Bits restantes

esperostopL3:
  while ( I2CPORT.CLK =1)     'Mientras CLK sea "1" esperar
  wend
end sub
```

Luego de implementados estos procedimientos de manejo del protocolo I2C se puede proceder a integrarlos con las rutinas de lectura RFID.

Se podrá empezar implementando el comando "Leer", el cual es enviado desde el Maestro, es decir del Lego Mindstorm NXT .Ya que en las rutinas realizadas para el Lector RFID no existe un mecanismo de comprobación de errores, se puede implementar haciendo un lectura doble cuando se envie el comando "LEER" y comparando que sean iguales estas dos lecturas. Como resultado de la comparación de estas dos lecturas, se envia como respuesta al Lego Mindstorm NXT los siguientes mensajes:

Default = 0

Dato Valido = 1

Error = 2

Si el dato es válido, se almacena el valor leído en la EEPROM. En la Figura 3.28 se muestra el diagrama de flujo del desarrollo de esta rutina.

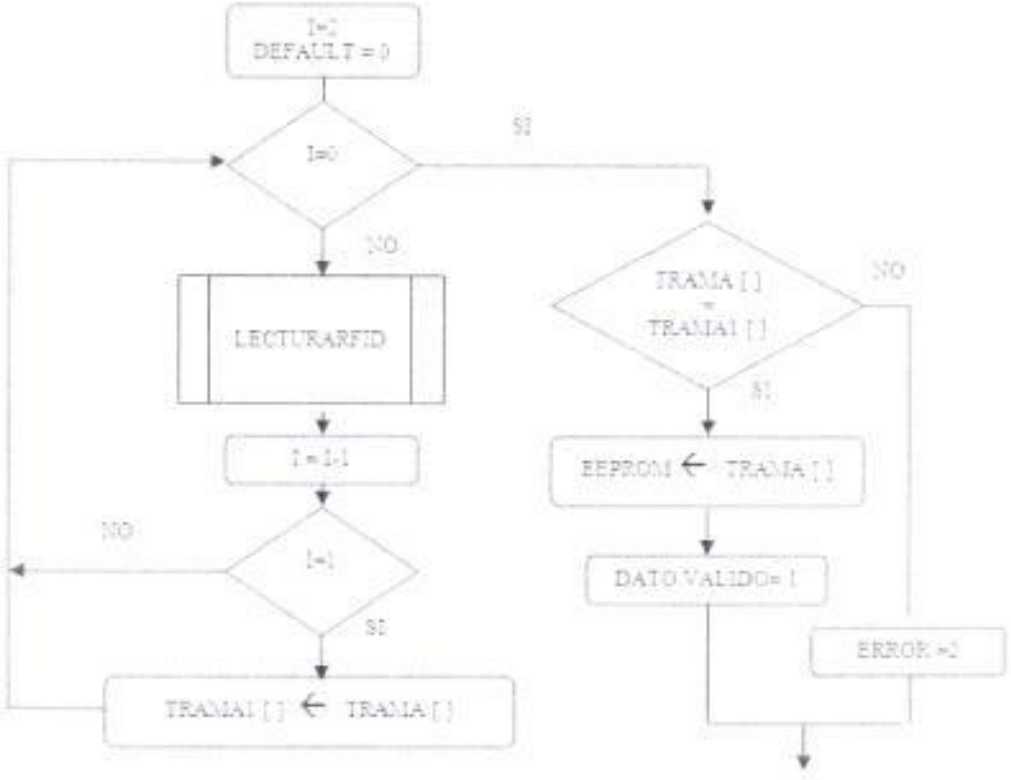


Figura 3.28 Diagrama de flujo de la rutina de comprobación de errores del valor leído.

Para realizar estas rutinas escribimos un nuevo módulo llamado "RFID_Comprobado":

```
Module RFID_Comprobado

include "RFID_Trama"

'****** VARIABLES ******

dim TRAMA1 as byte[10] 'Almacenamiento temporal para comparación

dim Resultado_RFID as byte 'Almacena resultado de la comparación

Implements

'****** PROCEDIMIENTOS ******

sub procedure compara 'Compara TRAMA1[ ] con TRAMA[ ] e indica

'el resultado de la comparacion

dim j as byte

  for j=0 to 9
```

```
    if TRAMA1[j] = TRAMA[j] then
    else
    Resultado_RFID = 2
    goto salir
    end if
  next j
    Resultado_RFID = 1
    Guardar_Trama()
salir:
end sub
sub procedure Guardar_Trama_1          'Pasa el contenido de TRAMA[ ] a
TRAMA1[ ]
    dim j as byte
 for j=0 to 9
 TRAMA1[j]=TRAMA[j]
```

```
 next j
end sub
sub procedure Lectura_RFID_Error     'Rutina que realiza la comprobacion de
                                     'error realizando 2 lecturas de la
dim i as byte                        'etiqueta RFID y comparandolas El
resultado
Resultado_RFID = 0                   'de la comparacion se almacena en
i=2                                  'Resultado_RFID
inicio:
if i=0 then
compara
else
lecturaRFID
i=i-1
  if i=1 then
```

```
Guardar_Trama_1
goto inicio
else
goto inicio
end if
end if
end sub
end.
```

3.1.3 Implementación de lector RFID

Finalmente se debe implementar el programa principal que utilizará las rutinas anteriormente implementadas, para lo cual se debe establecer un formato de comunicación entre el módulo lector RFID y el Lego Mindstorm NXT.

Básicamente, lo que se hace es enviar 3 tipos de comandos desde el Lego Mindstorm NXT, los cuales son: Comandos de acción, Comandos de Registros y Comando de información.

Comandos de Acción: Ejecuta un proceso en el módulo como "Leer etiqueta".

Comando de Registros: Devuelve el valor almacenado en la EEPROM la cual contiene ID de la etiqueta.

Comando de Información: Devuelve el valor del resultado de la última acción ejecutada.

La siguiente, Tabla 3.3, muestra los valores que se han asignado a los comandos y lo que recibirá el Lego Mindstorm NXT como respuesta:

Transmitido desde el NXT		Recibido en el NXT
Comando	Byte (d)	Byte
Valor de ID0	1	ID0
Valor de ID1	2	ID1
Valor de ID2	3	ID2
Valor de ID3	4	ID3
Valor de ID4	5	ID4
Valor de ID5	6	ID5
Valor de ID6	7	ID6
Valor de ID7	8	ID7
Valor de ID8	9	ID8
Valor de ID9	10	ID9
Registro de Validación	20	Resultado_ RFID
Leer etiqueta	30	

Tabla 3.3 Comandos soportados por el módulo Lector RFID

Ya que se ha definido la lógica de funcionamiento del controlador, se implementará la rutina principal del controlador como se indica a continuación:

```
program RFID_Firmware

include "RFID_Comprobado"
```

```
include "RFID_Trama"
include "I2C_Esclavo"
dim comando as byte
dim i as byte
main:
TRISA  = $FE
inicio:
esperostart
esperobyte(7)
comando=dato
esperostop
esperostart
esperobyte(7)
  select case  comando
     case 1
```

```
    dato=EEprom_read (0)
case 2
    dato=EEprom_read (1)
case 3
    dato=EEprom_read (2)
case 4
    dato=EEprom_read (3)
case 5
    dato=EEprom_read (4)
case 6
    dato=EEprom_read (5)
case 7
    dato=EEprom_read (6)
case 8
    dato=EEprom_read (7)
```

```
    case 9
        dato=EEprom_read (8)
    case 10
        dato=EEprom_read (9)
    case 20
        dato=Resultado_RFID
    case 30
        pausa_clock
        Lectura_RFID_Error()
        libera_clock
        goto final
  end select
respuesta:
  ponerbyte
final:
```

goto inicio

end.

3.2 SENSOR DE TEMPERATURA

3.2.1 DISEÑO DEL HARDWARE DEL SENSOR DE TEMPERATURA

El primer paso para realizar el diseño del hardware del módulo sensor de temperatura es identificar cuáles son los componentes básicos que se necesitan para llevar a cabo la toma de temperatura y llevarla hasta el PC. El siguiente diagrama de bloques describirá los componentes básicos para el sensor de temperatura. (Figura 3.29)

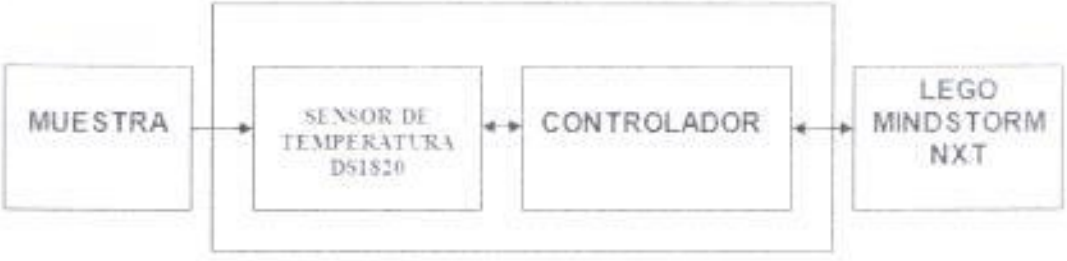


Figura 3.29 Diagrama de bloque del módulo sensor de temperatura

A continuación se describe brevemente los componentes anteriormente presentados:

MUESTRA: La muestra es el elemento sobre el que se medirá su temperatura. Este puede ser la temperatura ambiente, agua caliente o fria, etc.

DS1820: El DS1820 es un termómetro digital de alta precisión

CONTROLADOR: En términos generales el controlador debe interactuar con el sensor de temperatura DS1820 y el brick del Lego Mindstorm NXT, para lo cual será necesario utilizar un microcontrolador.

LEGO ® MINDSTORM NXT: se debe entender la forma en que se comunica el brick del Lego Mindstorm NXT y el hardware necesario para este fin.

A continuación se procede a la elección de los componentes:

DS1820: Se ha elegido este termómetro digital por los siguientes motivos:

- Por su encapsulado TO-92, apropiado para nuestro propósito, ya que es del tamaño de un transistor que se puede adaptar fácilmente como una punta de prueba.
- Por su tecnología y protocolo 1-Wire, ya que están diseñados para estar en contacto con el medio ambiente, el cual es apropiado para mediciones en ambientes externos.
- Rango de medición de temperatura entre -55 °C a +125 °C y tiene una precisión de ± 0,5 °C sobre el rango de -10 °C a +85 °C.

Es necesario entender el hardware del DS1820 para poder implementar el diseño del módulo. El siguiente gráfico muestra los pines del DS1820 y su función.

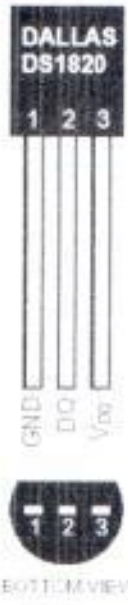


Figura 3.30 Encapsulado del DS1820

PIN	NOMBRE	FUNCION
1	GND	Tierra
2	DQ	E/S de datos.
3	VDD	Vdd opcional.Debe estar puesta a tierra cuando opera en modo de alimentación parásita.

Tabla 3.4 Descripción de los pines del DS1820

Como se puede observar de la Figura 2.30, se presenta una resistencia de pull-up conectada a la línea de transmisión de datos. La conexión de Vdd es opcional, pero ya que en nuestro diseño el sensor se encuentra cerca del módulo y no se requiere de una restricción en lo que se refiere al cableado se

utilizará la línea de alimentación Vdd. Se puede alimentar con una fuente de +5V.

CONTROLADOR: Como controlador se ha elegido el PIC16F84 ya que es un microcontrolador popular, de bajo costo y ofrece los recursos necesarios para la realización de esta interfase en cuanto a memoria de programa, puertos de E/S disponibles, velocidad de trabajo, etc.

3.2.2 DISEÑO DEL FIRMWARE DEL SENSOR DE TEMPERATURA

Para desarrollar el firmware se debe tener una visión general del problema para poder implementar las rutinas necesarias para el funcionamiento del sistema.

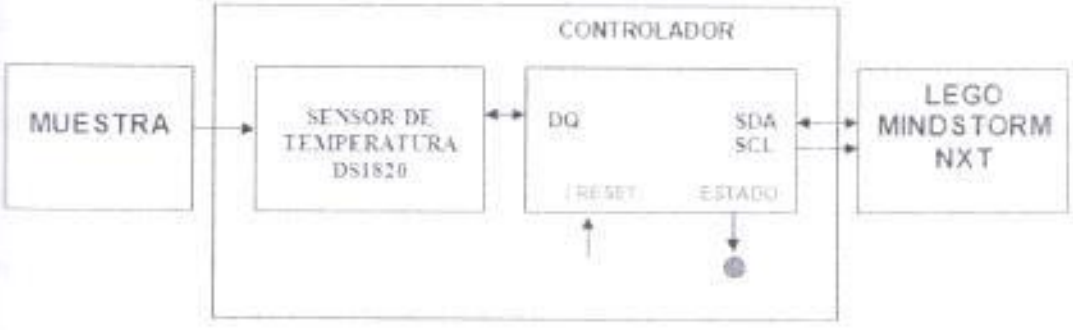


Figura 3.31 Diagrama de bloques final del módulo de sensor de temperatura

Se debe identificar los protocolos que se utilizará para la comunicación entre el controlador, el DS1820 y el Lego MindStorm NXT.

Entre el controlador y el sensor de temperatura DS1820 se utilizará el protocolo 1-Wire ya que este es el protocolo manejado por el DS1820. En el controlador se debe implementar el protocolo 1-Wire en modo Maestro, para lo que se utilizará rutinas incluidas en las librerías del lenguaje Mikrobasic.

Para la comunicación entre el controlador y el Lego MindStorm NXT se debe implementar el protocolo I2C. En el lado del controlador se debe implementar el protocolo I2C en modo esclavo, para lo cual se utilizará las rutinas que fueron diseñadas en el Módulo Lector RFID para este propósito.

En términos generales lo que el sistema realizará es que cuando reciba un comando "Leer" el sistema debe tomar una lectura de la temperatura en ese momento y retornar el valor leido como se muestra en la siguiente gráfica.

Figura 3.32 Diagrama de bloques del Comando "LEER"

Para llevar a cabo esta función se tiene que entender la forma en que trabaja el termómetro digital DS1820.

Adicionalmente, para nuestro propósito se debe tener presente que solamente se tiene un dispositivo esclavo en el bus 1-Wire y que básicamente se realizará una rutina de lectura de la temperatura.

Para acceder al DS1820 se tiene que realizar una secuencia de transacciones las cuales son:

Inicialización

Comandos ROM

Comandos de función del DS1820

En primer lugar para la inicialización, el Maestro emitirá un pulso de reset seguido por un pulso de "presence" transmitido por el esclavo. Cuando el Maestro a detectado el pulso de presence, este debe continuar con los comandos ROM los cuales se utilizan para seleccionar un esclavo especifico de los presentes en el bus. Finalmente cuando se ha ubicado al dispositivo esclavo, con el que se establecerá la comunicación, el Maestro puede usar los comandos de función como por ejemplo iniciar conversiones de temperatura, leer desde la memoria ScratchPad, etc.

Ahora que se entiende el modo de operación del DS1820 se revisará las librerías de Mikrobasic que se tienen disponibles para manejar el protocolo 1-Wire.

Para la inicialización, Mikrobasic dispone de la función "OW_Reset" la cual tiene el siguiente formato:

function OW_Reset (Dim byref Puerto as byte, Dim Pin as byte) as byte

Esta función retorna "0" si el DS1820 está presente y "1" si no lo está.

Para escribir un comando en el bus 1-Wire se utiliza el procedimiento "OW_Write", el cual tiene el siguiente formato:

Procedure OW_Write (dim byref Puerto as byte, dim PIN, comando as byte)

Para este procedimiento, se indica el puerto y el pin del microcontrolador por el que se quiere transmitir el comando hacia el DS1820.

Para leer un valor transmitido desde el DS1820 se tiene la función "OW_Read" con el siguiente formato:

function OW_Read (Dim byref Puerto as byte, Dim Pin as byte) as byte

Esta función retorna el valor leído desde el DS1820.

Con estas funciones básicas, se puede implementar una rutina que lea la temperatura en el DS1820 siguiendo los siguientes pasos.

1. Inicializar el DS1820 con el siguiente comando indicando el Puerto y el Pin del microcontrolador en el que está conectado el Pin DQ del DS1820.

   Ow_Reset (PORTDS, DQ)

2. Ya que se tiene sólo un esclavo conectado al bus, no es necesario que se identifique con su ID, por lo que se puede enviar un comando "Skip ROM" el cual en hexadecimal es CCh.

   Ow_Write (PORTDS, DQ, $CC)

3. Luego se debe aplicar un comando de función para la conversión de la temperatura. Posterior a la conversión, el resultado del dato termal es almacenado en un registro de temperatura de 2 bytes en la memoria ScratchPad, luego de lo cual el DS1820 regresa a un estado de bajo consumo.

   Ow_Write (PORTDS, DQ, $44)

4. Una vez realizada la conversión de temperatura y almacenado su valor en la memoria ScratchPad, nuevamente se realizará una secuencia de transacción para ahora leer el dato almacenado en la memoria.

   Ow_reset (PORTDS, DQ)

Ow_Write (PORTDS, DQ, $CC)

5. Ahora se tiene que enviar el comando para leer la memoria ScratchPad.

Ow_Write (PORTDS, DQ, $BE)

6. Finalmente se aplica el comando para transferir el dato leido de la memoria

Ow_Read (PORTDS, DQ)

Este proceso básico es el necesario para leer el dato de temperatura del DS1820 por parte del controlador.

Luego de obtenido el dato de la temperatura se debe transferir al Lego MindStorm NXT para lo cual se puede utilizar las rutinas I2C en modo esclavo que fueron desarrolladas e implementadas en el Módulo Lector RFID.

3.2.3 IMPLEMENTACION DEL MODULO SENSOR DE TEMPERATURA

Primeramente se diseñara la sección para la comunicación entre el sensor de temperatura DS1820 y el controlador.

Figura 3.33 Sección de comunicación entre el sensor de temperatura DS1820 y el controlador.

Esta sección puede ser implementada usando el siguiente conexionado en forma general. (Figura 3.34).

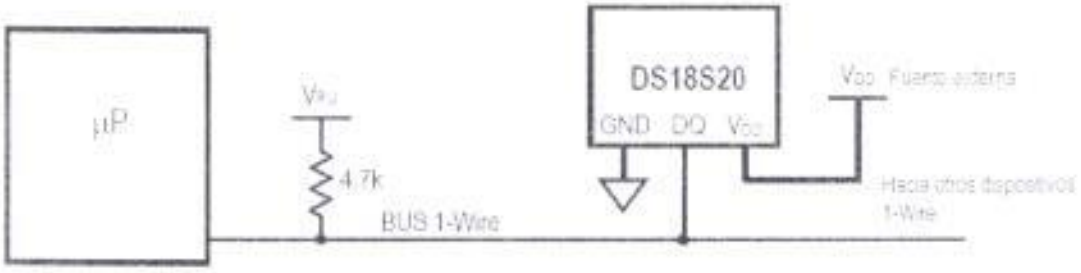


Figura 3.34 Forma general de la conexión al Bus 1-wire

La segunda parte del diseño corresponde a los bloques que se muestran en la Figura 3.35.

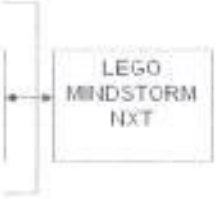


Figura 3.35 Sección de comunicación entre el controlador/NXT

Esta sección es exactamente igual a la implementada para el módulo sensor RFID.

Se puede agregar al módulo un botón de reset, el cual resulta útil para realizar pruebas del firmware y por si el controlador entra en un estado de inhibición, y se agrega también un led indicador, el cual resulta útil para informar visualmente algún estado del sistema.

Finalmente, se va a realizar un esquemático para la implementación del módulo como se muestra en la Figura 3.36.

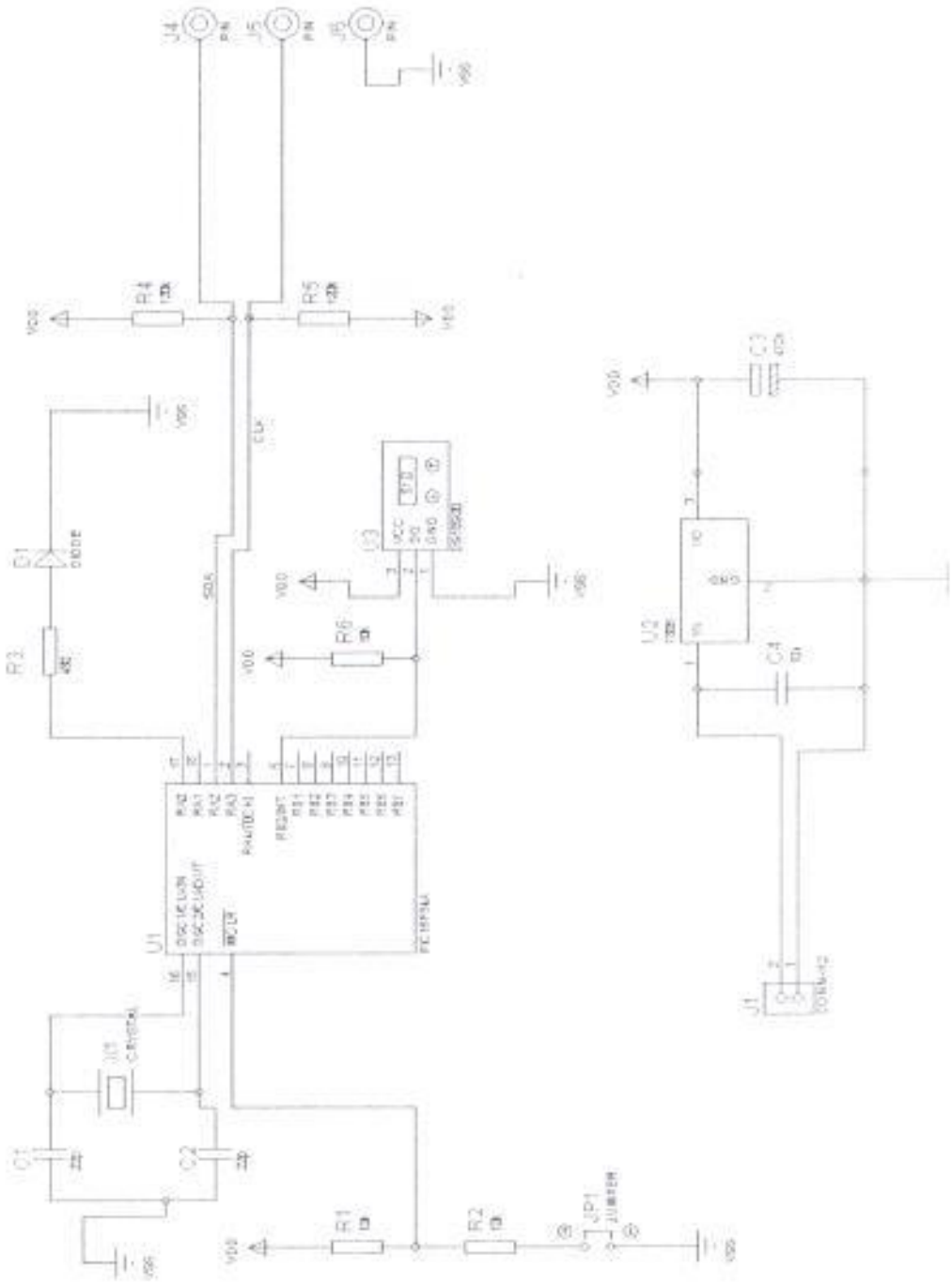


Figura 3.36 Diagrama esquemático del módulo sensor de temperatura

3.3 DESARROLLO DEL SOFTWARE EN EL PC

3.3.1 DISEÑO DEL SOFTWARE

Antes de empezar a diseñar un software de comunicación entre el PC y el Lego MindStorm NXT se debe entender el protocolo de comunicación del Lego MindStorm NXT, el cual se lo describirá a continuación.

El LEGO MINDSTORM NXT incluye las siguientes posibilidades de comunicación:

Comunicación Bluetooth, V2 con EDR, soportando el Serial Port Profile (SPP).

Comunicación USB, V2.0.

Tanto en la PC como en el bloque programable NXT, el sistema de comunicación BLUETOOTH y USB se encuentran divididos en sus respectivas capas de comunicación que se ilustran en la Figura 3.37.

Esto posibilita el acceso a la comunicación con el NXT ya sea utilizando el protocolo de LEGO MINDSTORM NXT, o escribiendo y leyendo directamente de los buffer de comunicación. Cuando se lee o se escribe directamente en los buffer, no hay manera de comprobar los datos que se envían o leer nuevamente desde la unidad NXT.

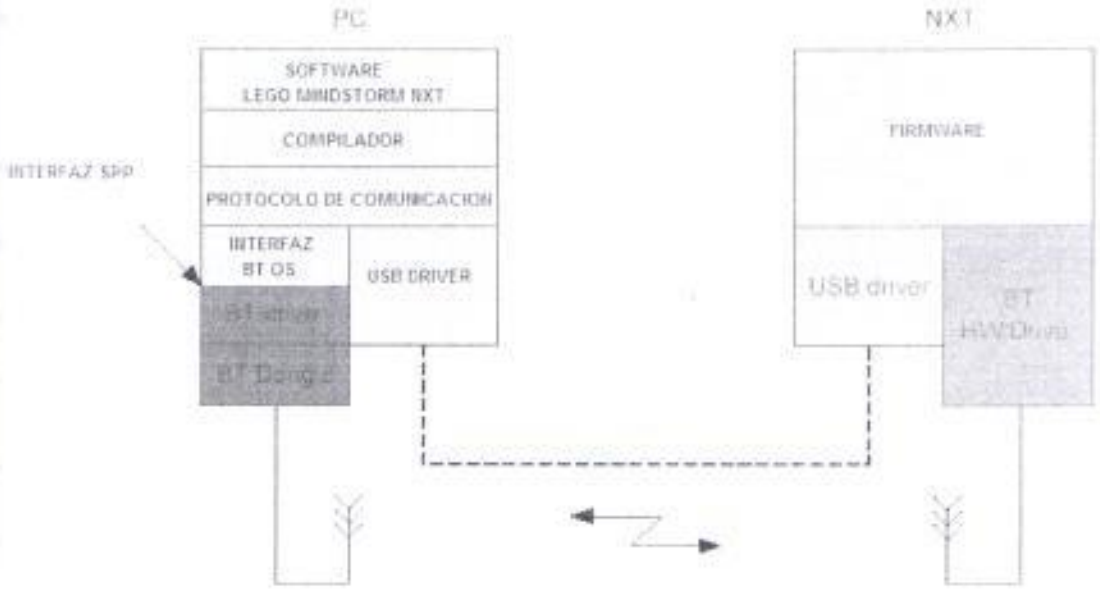


Figura 3.37 Capas de comunicación entre el PC y el Lego Mindstorm NXT

La capa de aplicación, ilustrada en la Figura 3.37 Lego Mindstorm NXT Software "GUI", está arriba de los puertos de comunicación Bluetooth y USB, porque esta capa maneja el dato que será cargado dentro de las capas de comunicación USB y Bluetooth.

Todos los cálculos de CheckSum y el manejo de los números de paquetes serán manejados por las capas individuales de comunicación USB o Bluetooth.

El nuevo protocolo de comunicación maneja los siguientes propósitos:

Bajar archivos (Escritura de archivos):

- Firmware *.rfw
- Programas definidos por el usuario *.rxe

- Programación en el controlador (Archivos generados en el controlador) *.rpg
- Programas Try – Me (Archivos preprogramados) *.rtm
- Sonido *.rso
- Gráficos *.ric

Subir archivos (Lectura de archivos):

- Subir programas definidos por el usuario *.rxe
- Gráficos *.ric
- Sonidos *.rso
- Archivos del data logger *.rdt

Borrar datos definidos por el usuario en el sistema embebido:

- Programas definidos por el usuario *.rxe
- Programación en el controlador (Archivos generados en el controlador) *.rpg
- Programas Try – Me (Archivos preprogramados) *.rtm
- Sonido *.rso
- Gráficos *.ric
- Archivos de data logger *.rdt

Comunicación directa con el sistema NXT:

- Obtener la lista de archivos dentro del NXT.

- Comandos directos a la máquina virtual.
- Comandos mensaje a Mailboxes.
- Datos directos desde el puerto de alta velocidad.

En la Figura 3.38 se muestra la vista general del protocolo.

Figura 3.38 Vista general de la trama.

Byte 0: Tipo de comando. El 7º bit más bajo de este byte es usado para identificar el tipo de comando. Actualmente el sistema tiene los siguientes tipos de comando que necesitan ser identificados. El bit 7 (MSB) es usado para identificar si el mensaje debe dar un mensaje de réplica o no.

0X00: Comando directo. Réplica requerida

0X01: Comando de sistema. Réplica requerida.

0X02: Comando respuesta.

0X80: Comando directo. Réplica no requerida.

0X81: Comando de sistema. Réplica no requerida.

Byte 1: Comando. Usado para identificar que debe pasar con el dato. Abrir, Leer, Escribir, Borrar dato, Comunicación directa con el NXT.

Byte 2 – N: Estos bytes ofrecen información adicional.

El protocolo de comunicación estándar NXT descrito es el que será envuelto dentro de los protocolos Bluetooth serial profile o el USB.

Es posible bajar 64 Bytes (USB HW tamaño del Buffer) antes que el microcontrolador salve los datos en la memoria FLASH. Esto tomará aproximadamente 6 ms durante lo cual el microcontrolador no puede recibir datos adicionales. Actualmente el sistema almacena hasta 256 bytes de datos los bytes de los datos en el buffer, lo cual se relaciona con una página de datos dentro de la memoria FLASH del procesador ATMEL. Cuando los 256 bytes han sido recibidos, ellos son escritos en la memoria flash del procesador ATMEL.

Todos los nombres de archivos en los comandos usarán 19 bytes (15.3 Char). Caracteres no usados deberán ser llenados con terminadores null.

Todos los comandos, por defecto, tendrán un paquete de retorno. Lo que significa, que si se ha decidido utilizar un comando con no retorno, debe ser especificado explicitamente dentro del comando.

Los siguientes comandos son usados para obtener el nivel de la bateria dentro del módulo controlador.

GET BATTERY LEVEL:

Paquete enviado al NXT desde el PC:

BYTE 0: 0X00 O 0X80

BYTE 1: 0X0B

Paquete de Retorno del NXT al PC

BYTE 0: 0X02

BYTE 1: 0X0B

BYTE 2: Status: 0 es igual a éxito. Mayor que cero indica un error.

BYTE 3 - 4: Voltaje en milivoltios

La funcionalidad del Bluetooth dentro del Lego Mindstorm NXT es usando el Serial Port Profile. Todo el manejo en banda base es hecho dentro del chip Bluetooth Lego Mindstorm NXT, lo cual significa que el dato enviado a y desde el chip Bluetooth es el a mismo dato que es enviado y desde la capa del protocolo de comunicación Lego Mindstorm NXT en el lado del PC.

En vista de que la trama es variable se tiene que manejar el problema de determinar el tamaño del dato que está siendo recibido, los datos de longitud deben ser agregados en frente del paquete del protocolo de comunicación. Estos bytes describen la longitud total del paquete de datos que deben ser recibidos menos los bytes de longitud en si mismos. Habrá 2 bytes que describen la longitud del paquete.

Long LSB	Long MSB	Tipo de Comando	Comando	Byte 5	Byte 6	Etc

Figura 3.39 Paquetes de datos enviado a través de bluetooth

Para la implementación en Visual basic 2008 de la rutina "GET BATTERY LEVEL" primeramente se debe establecer comunicación Bluetooth entre el PC y el Lego Mindstorm NXT, para lo cual se ha utilizado un adaptador Bluetooth marca Dlink modelo DBT-122 instalándolo con los drivers que vienen con el producto.

Posteriormente se enciende el Brick NXT y se activa la opción de comunicación Bluetooth, adicionalmente nos pedirá una clave tanto en el Brick como en el PC. Una vez conectados los dispositivos se debe observar en el PC el puerto COM utilizado por el adaptador Bluetooth para comunicarse. (Figura 3.40).

Figura 3.40 Visualización de puerto virtual serial del bluetooth

En nuestro caso es el COM5.

De la información dada en el manual Lego Mindstorm Communication protocol para obtener el nivel de la bateria del controlador.

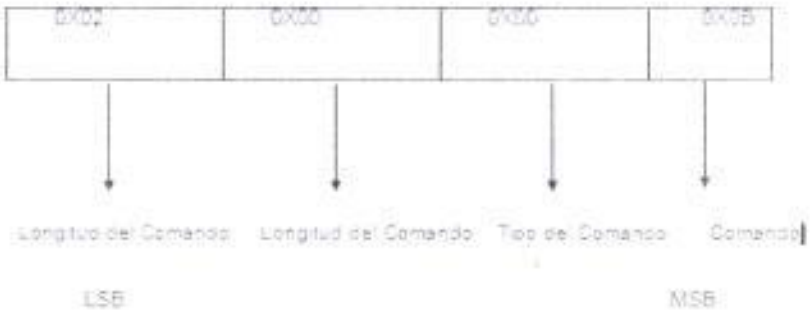


Figura 3.41 Formato del comando para obtener el nivel de bateria.

Para poder enviar el paquete via Bluetooth, se lo tiene que hacer por el Puerto "COM5" (Puerto Virtual Bluetooth), para lo cual se utilizará el control SerialPort que viene en el Visual Basic .NET. La siguiente rutina obtiene el nivel de la bateria del Lego MindStorm NXT. El formato del comando se lo puede observar en la Figura 3.41.

```
Public Function bateria(ByRef puertocom As System.IO.Ports.SerialPort) As
Integer

Dim comando(5) As Byte

Dim byteIn(6) As Byte

Dim Voltaje, i As Integer

    Try

comando(0) = &H2 'Tamaño de mensaje 2 bytes LSB
```

```
comando(1) = &H0  'Tamaño de mensaje 2 bytes MSB
comando(2) = &H0  '&H0 = Espero respuesta
comando(3) = &HB  '$HB = comando para leer bateria
puertocom.Write(comando, 0, 4) '0 = offset  y 4 = número de bytes
'Recibo dato
byteIn(0) = puertocom.ReadByte ' número de bytes en el mensaje
byteIn(1) = puertocom.ReadByte ' Debe ser 0 para NXT
        For i = 2 To 1 + byteIn(0) ' resto del mensaje
            byteIn(i) = puertocom.ReadByte()
        Next
Voltaje = byteIn(5) + byteIn(6) * 256 ' Voltaje tiene el byte                    bajo
en 5 y alto en 6
Return Voltaje
Catch ex As Exception
        MsgBox(ex.Message + " Verificar energia del Robot")
```

End Try

El resto de comandos directos, se implementan con el mismo formato anteriormente mostrado. Si requerimos una respuesta por parte del Lego MindStorm NXT, se implementará una función, pero si no es requerida una respuesta se implementa un procedimiento. Este diseño de las rutinas de comunicación con el controlador Lego MindStorm NXT se basan en el formato anteriormente presentado.

Se creará clases de los comandos de comunicación directa para utilizarlos en un programa principal. A continuación se describe las clases que son utilizadas en este proyecto.

Clase Brick

Esta clase contiene métodos relacionados con el estatus del controlador MindStorm NXT.

Figura 3.42 Clase Brick

Batería: Devuelve el valor de el voltaje de alimentación del controlador MindStorm NXT. Este método está sobrecargado, es decir se puede pasarle como parámetro el puerto serial o un PictureBox que sirve para indicar una falla de energia en el controlador MindStorm NXT.

Firmware: Obtiene la versión del firmware del controlador MindStorm NXT.

ponernombre: Sirve para poner un nombre al controlador del Lego MindStorm NXT, el cual aparece en el display del controlador cuando se enciende.

Clase Archivos

Esta clase tiene métodos relacionados con el manejo de archivos en el controlador Lego MindStorm NXT.

Figura 3.43 Clase Archivos

Abrirarchivo: este método abre un archivo en el controlador del Lego MindStormNXT.

Archivo_de_Sonido: este método abre un archivo de sonido en el controlador del Lego MindStormNXT.

Obtener_Nombre_Programa: Obtiene el nombre del programa que está en ejecución en el controlador del Lego MindStormNXT.

Subir: este método baja al PC un archivo que está en el controlador del Lego MindStormNXT llamado "datos.txt" y lo almacena en el PC en la siguiente ruta ""c:\datos.txt".

Clase I2C

Esta clase tiene los métodos relacionados con los comandos que trabajan con los puertos del Lego MindStormNXT y manejan el protocolo de comunicación I2C.

Figura 3.44 Clase I2C

I2C_Escribir: Envia un mensaje a un determinado puerto del Lego MindStorm NXT e indica cuantos bytes quiere recibir como respuesta cuando se requiera leer.

I2C_Leer: Lee los bytes que fueron requeridos cuando se envió el comando escribir.

I2C_Status: Devuelve el Status del puerto del Lego MindStorm NXT que se desea conocer.

Sonar: Sirve para enviar un comando al sensor de distancia que indica que se desea leer la distancia de un objeto colocado al frente del mismo.

Tipo _ sensor: Sirve para indicar al controlador del Lego MindStorm NXT, con que tipo de sensor se desea configurar en este puerto.

Clase Motor

Contiene métodos relacionados con el manejo de los motores del Lego MindStorm NXT.

Figura 3.45 Clase Motor

Getouputstate: Obtiene datos relacionados con el estatus de los motores del Lego MindStorm NXT.

Setouputstate: Configura los motores del Lego MindStorm NXT con los diferentes parámetros necesarios para su funcionamiento.

Clase Mailbox

Contiene los métodos necesarios para trabajar con el MailBox del Lego MindStorm NXT.

Figura 3.46 Clase Mailbox

Escribir: escribe un mensaje que se quiere enviar al Lego MindStorm NXT.

Leer: Lee un mensaje que esta en el Lego MindStorm NXT.

Las clases anteriormente presentadas son las necesarias para realizar las funciones de movimiento del robot, obtención de datos del robot, trabajar con los sensores conectados en los puertos del robot, trabajar con archivos y transferencia de mensajes.

Una vez presentadas las clases necesarias para la comunicación con el controlador del Lego MindStorm NXT, se dividirá el software del sistema en 3 secciones que son: Sección general, Sección de captura de datos de temperatura y Sección control y detección de muestras.

A continuación se describe brevemente sus funciones.

Sección general: Esta sección se mantendrá a lo largo de toda la ejecución del programa y nos permitirá conectarnos con el robot, presentará una consola de

mensajes general y los botones que nos permitirán cambiar entre la sección de captura de datos de temperatura y la sección de control y detección de muestras.

Sección de captura de datos de temperatura: Esta sección contiene los controles y las funciones necesarias para la captura de datos de temperatura.

Sección control y detección de muestras: Esta sección contiene los controles y las funciones necesarias para el control de movimiento del robot y la detección de muestras con etiqueta RFID.

3.3.2 FASE DE IMPLEMENTACION DEL SOFTWARE

Como punto de partida se ha implementado la interfaz gráfica de la aplicación la cual se muestra en la Figura 3.47.

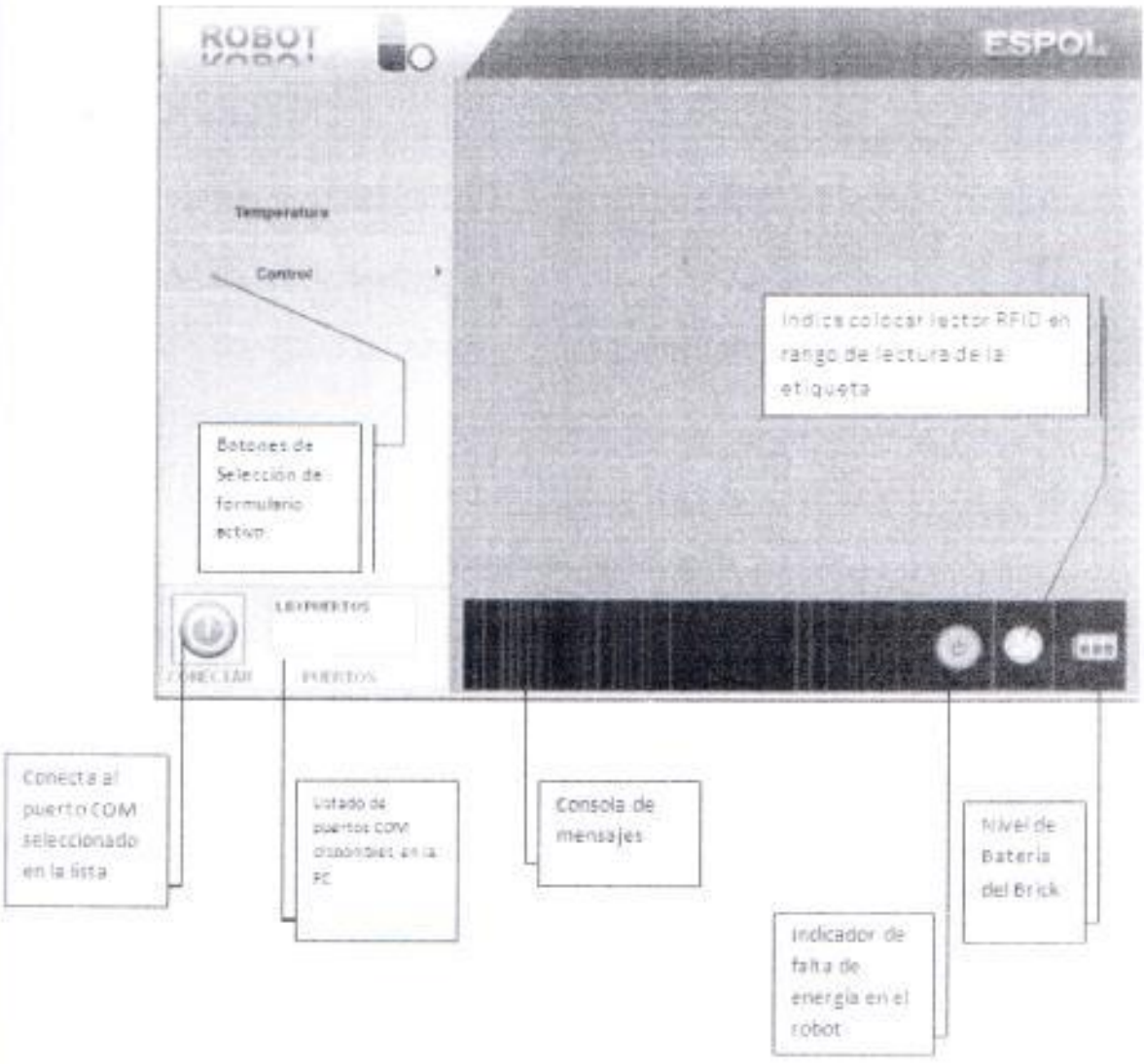


Figura 3.47 Formulario principal del programa.

Primeramente se ha enfocado el diseño de los formularios utilizando la técnica MDI (Multiple Document Interface) en la cual varios formularios pueden residir bajo un formulario padre.

Cuando se cargue el formulario se debe mostrar un listado de los puertos COM que están disponibles en la PC, para lo cual se utilizará la propiedad My.Computer.Ports.SerialPortNames la cual obtiene una colección de elementos tipo string con los nombres de los puertos COM. Luego los valores recuperados se almacenan en un ListBox.

```
For Each s As String In My.Computer.Ports.SerialPortNames
LBXPUERTOS.Items.Add(s)
Next
```

Luego se debe inicializar los botones del formulario.

Se procede analizar la forma de conectarse a un puerto COM. En el evento click del botón "CONECTAR" primeramente se debe asegurar que el usuario seleccione un puerto del ListBox ya que puede provocar un error por referencia Null.

```
If LBXPUERTOS.SelectedItem = Nothing Then
MsgBox("Debe seleccionar un Puerto")
Exit Sub
End If
```

Para abrir el puerto seleccionado en el ListBox se crea primeramente un objeto del control Serial Port con los siguientes valores en sus propiedades, las cuales permitirán una comunicación con el Brick.

Figura 3.48 Propiedades del puerto serial en el programa

En el evento click del botón "CONECTAR" primeramente se cierra el puerto ya que así se asegura que esté cerrado para poder abrirlo

```
SerialPort1.Close()
TBConsola.AppendText("Puerto " & SerialPort1.PortName & " cerrado" & vbCrLf)
SerialPort1.PortName = LBXPUERTOS.SelectedItem
TBConsola.AppendText("Abriendo el puerto: " & LBXPUERTOS.SelectedItem & vbCrLf)
```

SerialPort1.Open()

TBConsola.AppendText("Puerto: " & SerialPort1.PortName & " abierto " & vbCrLf)

Luego se consulto la versión de firmware y finalmente activo el Timer que monitorea la energia del robot y los controles si no se producen errores.

3.4 DESARROLLO DE LOS SOFTWARE EN EL LEGO MINDSTORM NXT

3.4.1 DISEÑO DEL SOFTWARE DE CAPTURA DE DATOS DE TEMPERATURA.

Inicialmente se describirá los controles del formulario que fue diseñado para la captura de datos de temperatura, para lo cual se presenta la Figura 3.49.

El software de captura de datos de temperatura tiene dos modos de funcionamiento, los cuales son:

Modo de captura de datos de temperatura en tiempo real.

Modo de captura de datos de temperatura independiente del PC.

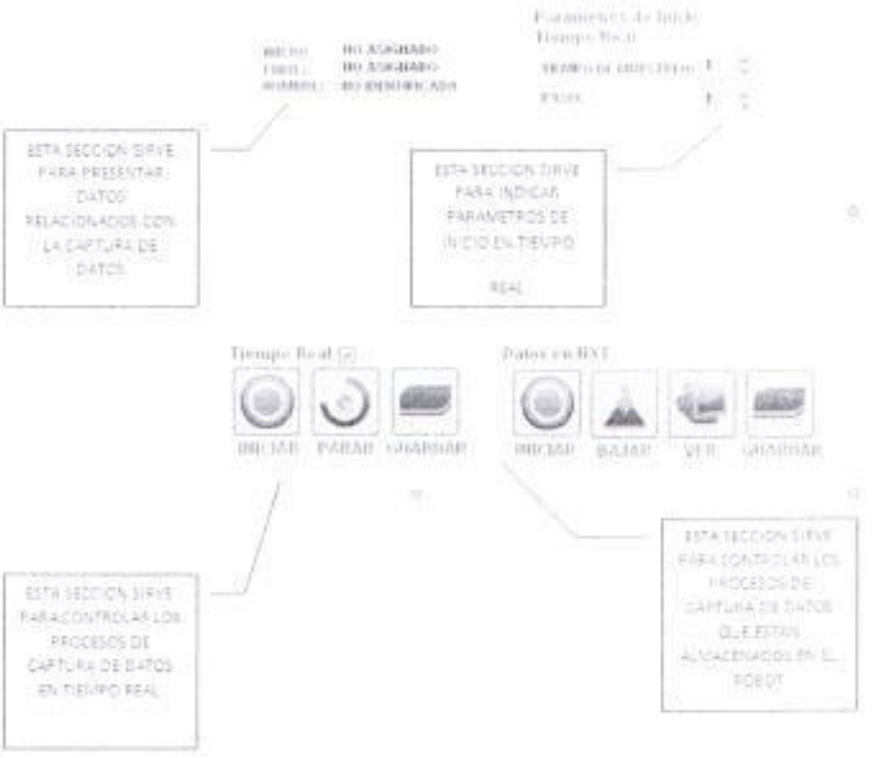


Figura 3.49 Formulario de captura de datos de temperatura

Tanto en el modo de captura en tiempo real y en el modo independiente del PC se necesita visualizar el dato de temperatura capturado en un plano cartesiano, el cual se necesita diseñar.

Para realizar el gráfico del plano cartesiano, primeramente se debe tomar en cuenta el lienzo sobre el que se va a dibujar, en nuestro caso es un PictureBox.

Los ejes de coordenadas que se utilizará tienen su origen en el extremo superior izquierdo, los valores +X aumentan hacia la derecha y los valores +Y aumentan hacia abajo (En pixeles).

El PictureBox tiene las siguientes dimensiones: 490x200 y sobre el mismo se dibujara un eje de coordenadas con las siguientes caracteristicas.

Figura 3.50 Eje de coordenadas

Para realizar el diseño del modo de captura de datos de temperatura en tiempo real se necesita utilizar variables que almacenen los datos a medida que se van capturando, para lo cual se utilizará un control ListBox. Además una variable "h", servirá para almacenar el valor del tiempo acumulado. Los valores para el tiempo que se desea muestrear y el paso o intervalo entre muestras se los

tomará de los controles de los parámetros de inicio que se encuentran en el formulario de captura de datos de temperatura.

3.4.2 IMPLEMENTACION DEL SOFTWARE DE CAPTURA DE DATOS DE TEMPERATURA.

Se empezará por describir la implementación del modo de captura de datos de temperatura en tiempo real. En el evento clic del botón "INICIAR" que se encuentra en el GroupBox de Tiempo Real se realiza la siguiente secuencia de acciones.

- Deshabilita los controles de tiempo de muestreo y paso.
- Habilitar Botón "PARAR" y deshabilitar botón "INICIAR".
- Deshabilitar el chequeo para modo de muestreo.
- Colocar la hora de sistema para indicar el inicio de la captura.
- Asignar las propiedades del PictureBox sobre el que se procederá a graficar.
- Conectar el evento Paint del PictureBox a el manejador del evento.
- Inicializar Timer para la toma de muestras, colocando la propiedad "interval" con el valor que indica el control donde se coloca el paso del muestreo multiplicado por mil ya que esta propiedad esta en milisegundos.

Ya que se habilita el timer, cada vez que se cumpla el intervalo asignado se ingresa en el procedimiento "Timer1_Tick", en el cual se realizan las siguientes acciones.

- Declarar los objetos necesarios para graficar los puntos de la temperatura en el plano.
- Preparar configuraciones de puerto I2C.
- Enviar comando que indica lectura en el Módulo de Temperatura.
- Capturar el dato de temperatura recibido.
- Agregar dato a la lista declarada a nivel de formulario.
- Graficar el dato.
- Acumular el tiempo de muestreo en la variable "h" a nivel de formulario.
- Revisar si estoy en el limite del muestreo, determinado por el valor definido en tiempo de muestreo.

Si se ha alcanzado el limite de muestreo se realizará las siguientes acciones:

- Deshabilitar botón "PARAR" y habilitar botón "GUARDAR".
- Colocar la hora de sistema que indica el tiempo final de la captura de datos.

Cada vez que se desea repintar el formulario, se ingresa en el evento "PictureBox1_Paint", en el cual se realizan las siguientes acciones:

- Primeramente se define el lienzo sobre el que se va a dibujar
- Puntos de referencia para dibujar el plano cartesiano
- Variables necesarias para implementar el manejo de ficheros
- Color de fondo del lienzo
- Definir los lapiceros que se va utilizar

- Definir la fuente y la brocha
- Graficar los ejes
- Graficar las divisiones en el eje x.
- Graficar las divisiones en el eje y.
- Graficar las lineas de referencia
- Colocar los titulos de los ejes

Luego de estas acciones preliminares verificar el checkbox de selección de modo, y si se encuentra en modo tiempo real se realizará las siguientes acciones:

- Solo se grafica si el tiempo acumulado de muestreo es diferente de cero.
- Si tiempo acumulado es diferente de cero, se grafica los valores que están almacenados en la lista declarada a nivel de formulario.

Al dar click en el botón "PARAR" se realiza las siguientes acciones:

- Deshabilitar el botón "PARAR" y habilitar el botón "GUARDAR".
- Detener el Timer de muestreo.
- Colocar la hora de sistema que indica el tiempo final de la captura de datos.

Al presionar click en el botón "GUARDAR", se realizan las siguientes acciones:

- Habilitar el chequeo en para modo de muestreo.
- Agregar datos finales a la lista.

- Rutinas para guardar en archivo. El cual se guarda en "C:\DatosNXT\" ,con la hora y la fecha del sistema como nombre con extensión "txt".
- Limpiar Valores de etiquetas.
- Borrar contenido de lista y valor de variable h.
- Artificio para invocar el evento paint para limpiar pantalla.
- Deshabilitar botón de "GUARDAR" y habilitar botón "INICIAR".
- Habilitar los controles de tiempo de muestreo y paso.

Para implementar el modo de captura de datos de temperatura independiente del PC, se trabaja con los siguientes controles.

Si al realizar click en el botón "INICIAR" que se encuentra en la sección de captura de datos de temperatura independiente del PC, se ejecutan las siguientes acciones:

- Ejecutar remotamente el archivo en el controlador del Lego MindStorm NXT.
- Iniciar Timer2 que controla si termino de ejecutarse programa en el controlador del Lego MindStorm NXT.

Al dar click en el botón "BAJAR", se ejecuta la siguiente acción:

Form1.abrir.subir(Form1.SerialPort1)

Este método sirve para subir el archivo "datos.txt" almacenado en el Lego MindStorm NXT y luego de recuperado lo almacena en la PC en "c:\datos.txt "

Al dar click en el botón "VER", se ejecutan las siguientes acciones:

- Artificio para Invocar el evento paint para limpiar el gráfico anterior en la pantalla.
- Se asigna las propiedades del PictureBox que se utilizara para graficar la temperatura.
- Se conecta el evento Paint del PictureBox a el manejador del evento.

Al realizar las acciones anteriores se ingresa en en el evento "PictureBox1_Paint" ,en el cual primeramente se grafica el eje de coordenadas y luego se verifica el checkbox de selección de modo . Si este checkbox no esta seleccionado se ejecutan las siguientes acciones:

- Se crea un flujo desde el fichero datos.txt
- Abrir el archivo datos.txt
- Leer una linea que abarca todo el fichero datos.txt
- Cerrar el archivo.
- Crear un objeto string con el texto leído
- Se extrae los valores de la cadena, y se los convierte a tipo enteros y se los muestran.

Programa independiente del PC utilizando NXC

Se puede utilizar un programa desarrollado en el lenguaje NXC el cual se ejecutará en el Controlador del Lego Mindstorm NXT, con el propósito de realizar la captura de datos de temperatura independiente del PC.

Debido a que los puertos del Lego Mindstorm NXT son capaces de comunicarse con una variedad de diferentes tipos de sensores, se debe preparar el puerto del controlador del Lego Mindstorm NXT, indicando el tipo de sensor con el que se comunicará. El módulo sensor de temperatura es un sensor de tipo digital el cual se comunicará utilizando el protocolo I2C, por lo tanto se debe utilizar el comando SetSensorType.

El comando SetSensorType tiene el siguiente formato:

SetSensor (port, const configuration)

Como parámetros se tiene el puerto y el tipo de sensor, los cuales ya se encuentran predefinidos por el NXC. El comando puede quedar de la siguiente forma:

```
#define PUERTO_SENSOR_TEMPERATURA      IN_1
SetSensorType(PUERTO_SENSOR_TEMPERATURA, IN_TYPE_LOWSPEED);
```

En el comando anterior IN_1 es el nombre con que el NXC reconoce al Puerto 1, el cual se lo identifica con una etiqueta.

IN_TYPE_LOWSPEED es una constante predefinida en NXC, la cual indica que es un sensor de tipo digital el cual se comunicará utilizando el protocolo I2C.

Adicionalmente el NXT permite a los sensores ser configurados en diferentes modos, lo cual determina como los valores de los datos son procesados (Boolean, porcentajes, grados Celsius, etc).

Para la configuración anterior se utiliza el comando:

SetSensorMode (PUERTO_SENSOR_TEMPERATURA, IN_MODE_RAW);

IN_MODE_RAW indica valores desde 0 a 1023.

Luego de configurar el tipo y el modo del sensor, se debe resetear el valor de un sensor para asegurarnos de que este valor es valido usando el siguiente comando:

ResetSensor (PUERTO_SENSOR_TEMPERATURA);

Finalmente se debe dar un tiempo para que se apliquen los cambios, con un comando:

Wait (tiempo);

Todos estos comandos anteriormente mencionados se puede agruparlos en un bloque de inicialización, de la siguiente forma:

```
void inicio_puerto (){
 SetSensorType(PUERTO_SENSOR_TEMPERATURA,
IN_TYPE_LOWSPEED); //Configurar el tipo de sensor
SetSensorMode(PUERTO_SENSOR_TEMPERATURA, IN_MODE_RAW);    //
Configurar el modo del sensor
ResetSensor(PUERTO_SENSOR_TEMPERATURA); // Resetear el sensor
Wait(1000);     //Dar un tiempo para que se estabilicen los valores y
configuraciones
```

Luego de inicializado el puerto se debe implementar un comando para enviar un dato y recibir un dato como respuesta para lo cual se puede utilizar un comando de alto nivel que realice esta tarea.

El comando I2CBytes puede ser usado para este propósito, para lo cual se describirá a continuación.

El comando tiene el siguiente formato:

I2CBytes (port, inbuf, in/out count, out outbuf)

Este método escribe los bytes contenidos en el buffer de entrada "inbuf" al dispositivo I2C en el puerto especificado, y luego se prepara para leer el número de bytes especificados y los almacena en el buffer de salida "outbuf". Retorna verdadero o falso indicando si el proceso de lectura I2C fue exitoso.

Si se utiliza para diseñar una función que tenga como parámetro de entrada el comando que se desea enviar y como retorno se obtiene el valor que envía el módulo sensor de temperatura quedaría de la siguiente forma:

```
byte escribir_leer_i2c(byte comando){
    byte bytes_a_leer=1;                    // Bytes que se espera leer como
respuesta
    byte respuesta[1];                      // Variable que almacena la
Respuesta
```

```
  byte mensaje[2];                          // Variable que almacena el
mensaje
  ArrayInit(mensaje, 0, 2);                 // Inicializo arreglo mensaje con
ceros
  ArrayInit(respuesta, 0, 1);               // Inicializo arreglo respuesta con
ceros
  mensaje[0] = DIRECCION_SENSOR_TEMPERATURA;  // Direccion I2C de
módulo sensor de temperatura
  mensaje[1] = comando;                     // Comando que se quiere
enviar
  I2CBytes(PUERTO_SENSOR_TEMPERATURA, mensaje, bytes_a_leer ,
respuesta);
  return respuesta[0];                      // Lo que devuelve la funcion
}
```

Estas rutinas básicas nos pueden servir para la comunicación entre el Módulo Sensor de temperatura y el controlador del Lego Mindstorm NXT.

El paso siguiente es crear un archivo en el controlador del Lego Mindstorm NXT en el cual se va almacenar los datos de temperatura capturados, se lo llamará "datos.txt".; pero previamente se borrará un archivo con el mismo nombre del

que se creará ya que de esta forma se asegura de borrar la información anteriormente almacenada, para lo cual se utilizará el siguiente comando.

```
DeleteFile ("datos.txt");
```

Luego de asegurarse de borrar los datos anteriores, se procede a crear el archivo para lo cual se usa la función CreateFile, en la cual se debe poner como parámetros el nombre del archivo que se quiere crear, el tamaño del archivo en bytes y un manejador de archivo, de la siguiente forma.

```
#define LONG_MAX        500
byte handle  ;
CreateFile("datos.txt", LONG_MAX , handle)
```

Ya que CreateFile es una función que retorna códigos de resultado, se implementará una estructura de control que detecte si ocurrió un error al abrir el archivo.

```
if (CreateFile("datos.txt", LONG_MAX , handle) == NO_ERR) {  //Si se creo el archivo sin retornar error entonces
   TextOut( 0, LCD_LINE1, "Archivo Abierto." );  //Mostrar el mensaje "Archivo Abierto."
   Wait( 2000 );
 } else {
```

```
  TextOut( 0, LCD_LINE1, "Error al abrir archivo!!" ); //Mostrar el mensaje "Error
al abrir archivo!!"
  Wait( 10000 );
  Stop(true); // Detener ejecucion de el Programa
 }
```

Si ocurre un error al crear el archivo entonces muestro un mensaje en la pantalla del controlador del Lego MindStorm NXT y detengo la ejecución del programa.

Si no ocurre un error al crear el archivo, muestro un mensaje de éxito al crear el archivo y se continúa con limpiar la pantalla del controlador del Lego MindStorm NXT con el comando:

```
ClearScreen();
```

3.4.3 DISEÑO DEL SOFTWARE DE DETECCION DE MUESTRA CON ETIQUETA RFID.

Inicialmente se debe tener un conjunto de ID de muestras con las que se va a comparar la ID de la etiqueta RFID que se lee con el módulo lector RFID, para lo cual se crea una lista con 3 ID como se puede observar en la Tabla 3.5.

0F00B321F8	Muestra 1
0E00B93B52	Muestra 2
0F00AED9E9	Muestra 3

Tabla 3.5 Base de datos de ID de las etiquetas RFID.

Como se puede observar a cada ID de 10 bytes, se representará con nombres como: "Muestra 1", "Muestra 2" y "Muestra 3".

A continuación se mostrará la interfaz de usuario en la que se incluyen los controles que se van a utilizar para la detección de muestra con etiqueta RFID.

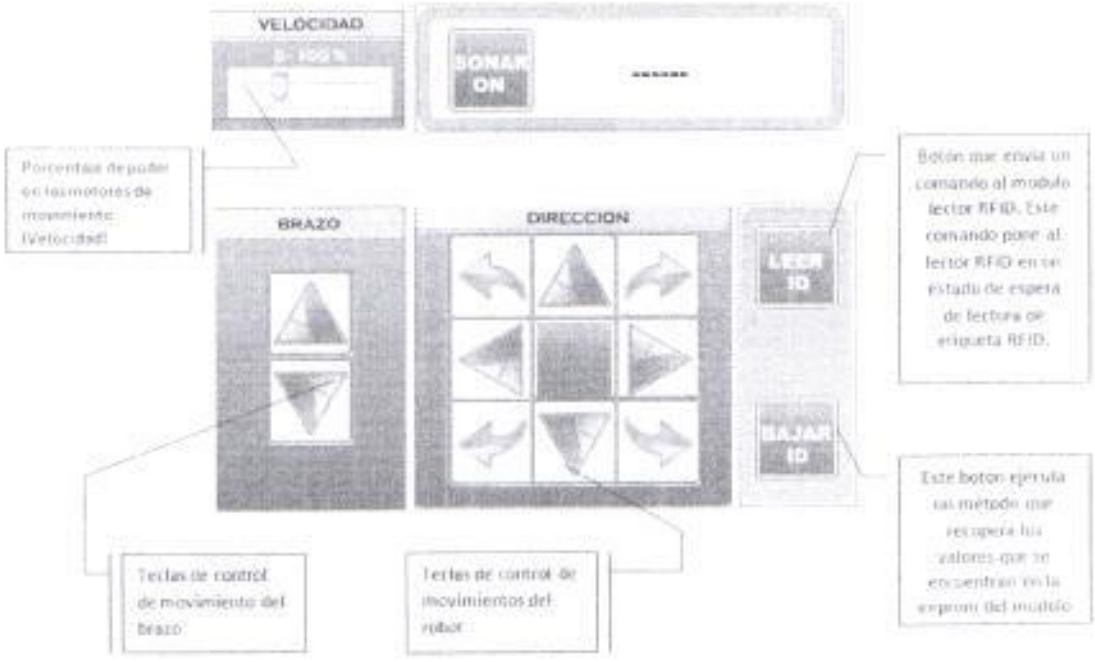


Figura 3.51 Formulario lectura de etiquetas RFID y de control.

Como se puede observar en la Figura 3.51, solamente dos botones nos servirán para poder realizar la detección de la muestra con etiqueta RFID.

Con el botón "Leer ID" lo que se hace es enviar al módulo lector RFID el comando "30", que indica "Leer etiqueta RFID" con lo que se pone al Lector RFID en un estado de alerta, activando la antena en espera de aproximar la

etiqueta RFID. Cuando la etiqueta RFID se encuentra en el rango de lectura del lector, la etiqueta es leida y su valor se almacena en la memoria EEPROM del controlador del módulo lector RFID. Adicionalmente cuando se presiona este botón se muestra un icono que nos indica que se debe aproximar la etiqueta RFID a el Lector RFID.

A continuación se muestra en la Figura 3.52 el siguiente icono:

Figura 3.52 Icono de "Leer etiqueta RFID"

Finalmente se tiene el botón "Bajar ID" el cual ejecuta un método que primeramente recupera las 10 ubicaciones de la memoria EEPROM del controlador del módulo lector RFID, las cuales almacenan el valor de la última etiqueta leida e implementan un mecanismo de detección de errores, filtrando caracteres no validos. Luego de recuperar los 10 bytes de ID, se compara este ID leído con los 3 ID que están almacenados en la lista y según su orden corresponderán a las muestras respectivas, y se mostrará en la consola de mensajes si ha sido identificada la muestra y su valor.

3.4.4 IMPLEMENTACION DEL SOFTWARE DE DETECCION DE MUESTRA CON ETIQUETA RFID.

Inicialmente se crea una base de datos con un conjunto de ID de muestras con las que se comparará la ID de la etiqueta RFID que se lee con el módulo lector RFID de la siguiente forma:

```
Me.LBoxID.Items.Add("0F00B321F8")      'Muestra 1
Me.LBoxID.Items.Add("0E00B93B52")      'Muestra 2
Me.LBoxID.Items.Add("0F00AED9E9")      'Muestra 3
```

Como se puede observar se utiliza un ListBox llamado "LBoxID" y los índices en los que están ubicados los ID que corresponden con las muestras.

Cuando se da click en el botón "Leer ID" se ejecuta lo siguiente:

- Se muestra el PictureBox que contiene el icono que indica que se tiene que aproximar Lector RFID a la etiqueta RFID.

```
Form1.PictureBox4.Visible = True
```

- Se envia el comando "30" el cual indica al módulo lector RFID "Leer etiqueta RFID

```
I2CPuerto.I2C_Escribir(Form1.SerialPort1, I2C.PuertosNXT.Puerto2, 30, 1)
```

- Si se recibe como respuesta un dato inválido, se vuelve a enviar el comando, para lo cual se implementó el siguiente código:

```
While dato = 1
I2CPuerto.I2C_Escribir(Form1.SerialPort1, I2C.PuertosNXT.Puerto2, 30, 1)
dato = I2CPuerto.I2C_Leer(Form1.SerialPort1, I2C.PuertosNXT.Puerto2)
End While
```

Ahora se describirá el código que se implementará cuando se da click en el botón "Bajar ID". Ya que el objetivo es leer las 10 ubicaciones de memoria EEPROM del módulo lector RFID, enviando los comandos "1","2","3"....."10" y recibiendo como respuesta los valores de ID0 hasta ID10, se implementará con un ciclo FOR de la siguiente forma:

```
For i = 1 To 10
I2CPuerto.I2C_Escribir(Form1.SerialPort1, I2C.PuertosNXT.Puerto2, i, 1)
dato = I2CPuerto.I2C_Leer(Form1.SerialPort1, I2C.PuertosNXT.Puerto2)
'Mecanismo de filtrado de caracteres inválidos
 While dato = 1
I2CPuerto.I2C_Escribir(Form1.SerialPort1, I2C.PuertosNXT.Puerto2, i, 1)
```

```
dato = I2CPuerto.I2C_Leer(Form1.SerialPort1, I2C.PuertosNXT.Puerto2)
End While
While dato = 0
If contador = 10 Then
Exit While
Else
I2CPuerto.I2C_Escribir(Form1.SerialPort1, I2C.PuertosNXT.Puerto2, i, 1)
dato = I2CPuerto.I2C_Leer(Form1.SerialPort1, I2C.PuertosNXT.Puerto2)
contador = contador + 1    'Es necesario para que no se cuelgue cuando no hay
respuesta
'Verifico si el dato nuevamente es cero
If dato = 0 Then
Form1.TBConsola.AppendText("Puerto 2 no responde. Verificar si módulo esta
encendido" & vbCrLf)
End If
End If
```

```
End While
```

Se puede observar que se implementó tambien un mecanismo de filtrado de caracteres no válidos. Luego de haber recuperado el ID, se debe compararlo con la base de datos de IDs de la siguiente forma:

```
'Comparo con los elementos del ListB
For j = 0 To Me.LBoxID.Items.Count - 1
If mensaje = Me.LBoxID.Items.Item(j) Then
  Select Case j
  Case 0
  Form1.valor_muestra = "MUESTRA 1"
                Archivos_Control.Archivo_de_Sonido(Form1.SerialPort1,
"muestra1.rso")
   Form1.TBConsola.AppendText("Muestra 1 encontrada" & vbCrLf)
  Case 1
  Form1.valor_muestra = "MUESTRA 2"
  Form1.TBConsola.AppendText("Muestra 2 encontrada" & vbCrLf)
```

```
Case 2

Form1.valor_muestra = "MUESTRA 3"

Form1.TBConsola.AppendText("Muestra 3 encontrada" & vbCrLf)

End Select

End If

Next
```

Cabe mencionar que fue necesario implementar el control de los movimientos del robot en el formulario utilizado para la detección de muestras con etiquetas RFID ya que permite realizar la accion de acercar el lector RFID a la muestra y además permite manipular el brazo en el que se encuentra el sensor de temperatura.

CAPITULO 4

PRUEBAS REALIZADAS

4.1 PRUEBAS DE LOS MÓDULOS

4.1.1 PRUEBAS DEL LECTOR RFID.

Para la realizar las pruebas es necesario establecer la correcta polarización del módulo RFID, una descripción grafica más detallada seria la que se muestra en la Figura 4.1.

Luego de polarizar el lector y aproximar una etiqueta RFID, se observará lo siguiente:

- Al polarizar el lector RFID con Vcc =+5V y el pin /Enable = 1. El led indicador de estado permanece en color verde lo cual indica que se encuentra en estado IDLE, es decir inactivo.

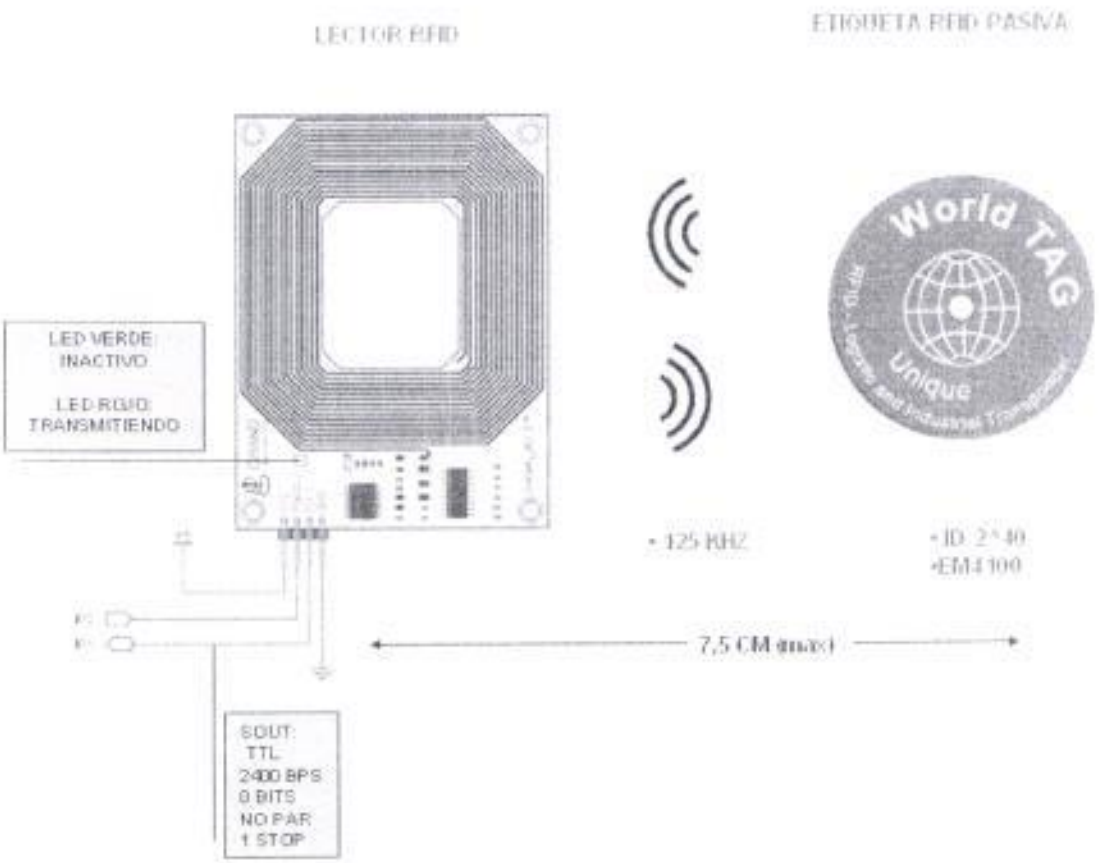


Figura 4.1 Grafica de polarización del lector RFID

- En este estado IDLE si se acerca la etiqueta RFID, el led continua en color verde, debido a que el pin /Enable = 1.
- Si se hace que el pin /Enable = 0, entonces el led indicador de estado se torna de color rojo lo cual indica que lector está en estado Activo.

Luego de observar el comportamiento del lector por medio del led indicador de estado, se procede a observar la trama que se genera en la salida SOUT, como

se muestra en la Figura 4.2, (Niveles de voltaje TTL) cuando se acerca una etiqueta RFID, por medio de un analizador lógico.

Figura 4.2 Trama generada en la salida SOUT.

Se analizará más detalladamente la trama:

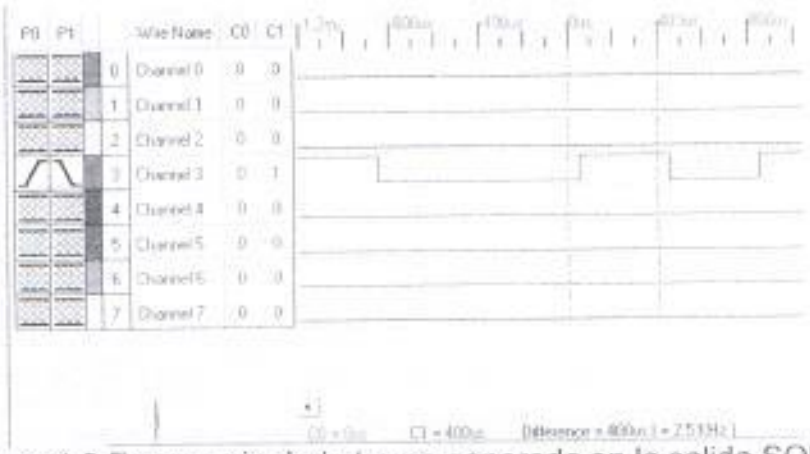


Figura 4.3 Frecuencia de la trama generada en la salida SOUT.

De la Figura 4.3 se puede observar la frecuencia aproximada de la transmisión es una frecuencia de 2,5 Khz (2500 bps) la cual es aproximada a la indicada en

el manual del Lector RFID de 2400 bps. Posteriormente se puede analizar un byte completo.

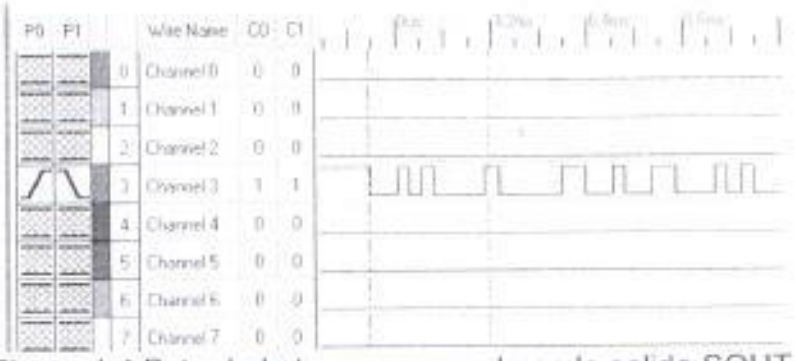


Figura 4.4 Byte de la trama generada en la salida SOUT.

En la Figura 4.4 se muestra el primer byte 01010000. Debido a que se envía el bit menos significativo primero el valor del byte es 00001010 lo cual es igual a 0x0A, es decir el byte de Start.

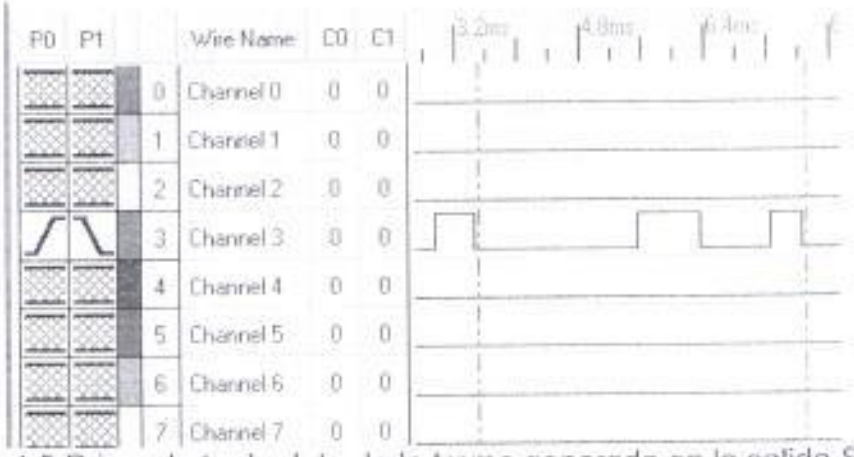


Figura 4.5 Primer byte de dato de la trama generada en la salida SOUT.

El byte que sigue al byte de Start, que se muestra en la Figura 4.5, es el primer byte 00001100. . Debido a que se envía el bit menos significativo primero el

valor del byte es 00110000 lo cual es igual a 0x30. Debido a que los bytes de la trama están en código ASCII ,el valor 0x30 representa al número 0.

Observar el último byte de la trama:

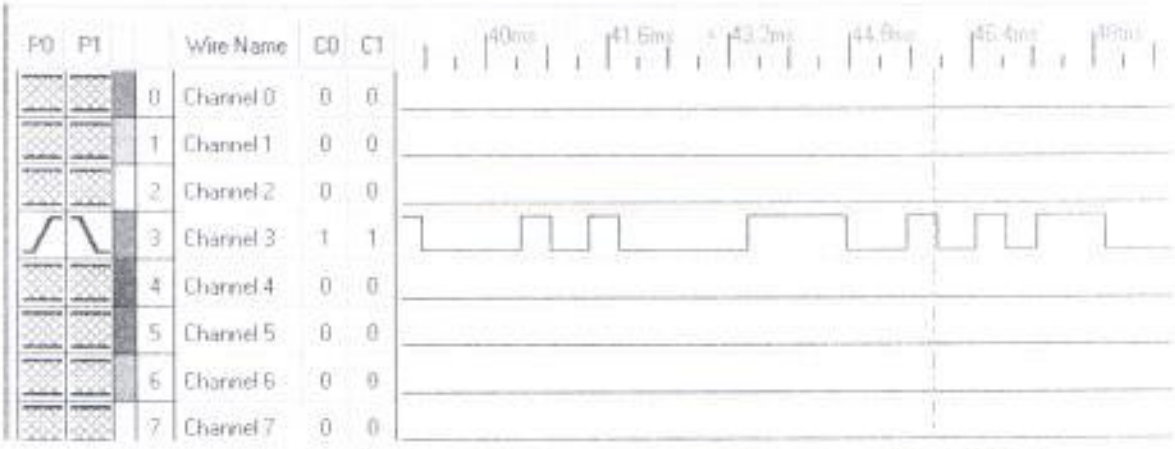


Figura 4.6 Byte de stop de la trama generada en la salida SOUT.

Como se muestra en la Figura 4.6 que el ultimo byte es 10110000 .Debido a que se envia el bit menos significativo primero el valor del byte es 00001101 lo cual es igual a 0x0D es decir el byte de Stop. Observando byte por byte en la trama como se muestra en la Figura 4.7.

BYTE DE INICIO	DIG	DIG	DIG	DIG	DIG	DIG	DIG	DIG	DIG	DIG	BYTE DE PARADA
0X0A	0	F	0	0	B	3	2	1	F	8	0X0D

Figura 4.7 Valores hexadecimales de la trama generada en la salida SOUT.

Se obtiene que el ID de la etiqueta eliminando el byte de Start y Stop:

0F00B321F8

En las pruebas realizadas entre el NXT y el controlador se puede observar la siguiente Figura 4.8, en donde muestra el resultado de enviar el comando para lectura del registro de ID9

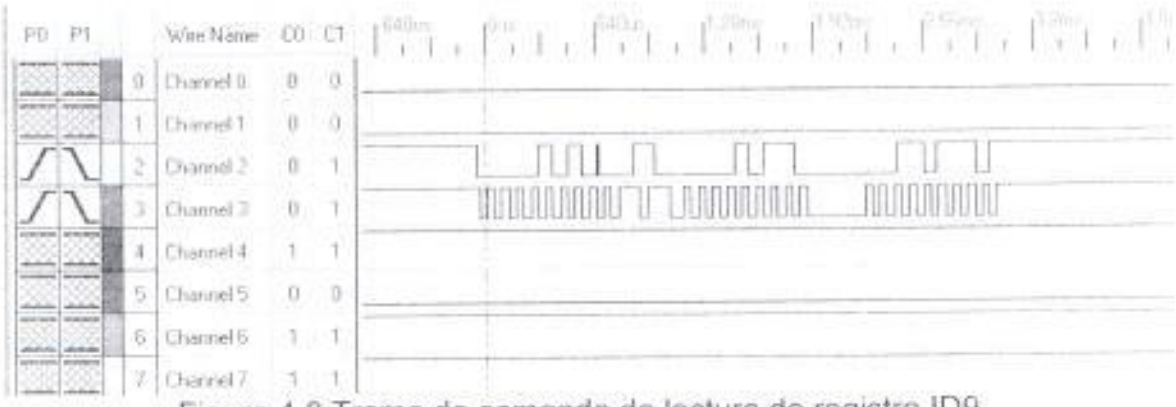


Figura 4.8 Trama de comando de lectura de registro ID9.

Como se puede observar primeramente se envia el comando, luego viene un byte que indica que el maestro quiere leer y el último byte es la respuesta del esclavo, es decir el módulo lector RFID.

Enviando sucesivamente este comando se observara una situación en la que no se produce la respuesta esperada como en la siguiente Figura 4.9.

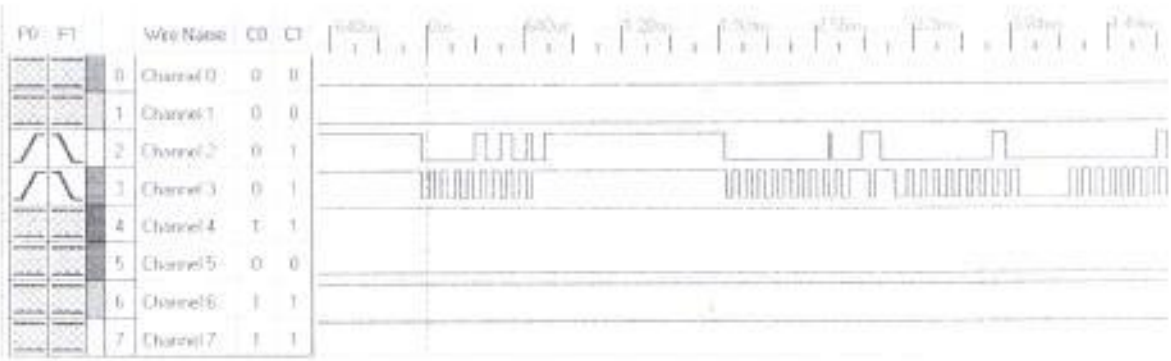


Figura 4.9 Trama que muestra una condición de error

Se puede observar que en el primer byte, no se está enviando el noveno pulso de reloj en el cual se envía un ack al maestro, por lo que se pierde la secuencia esperada.

Este problema introduce errores como enviar caracteres "null" o "1". Se puede manejar este problema implementando algún mecanismo de detección de errores en el receptor.

En la Figura 4.9 se puede observar que cuando se produce el error se esta enviando el dato "1" y ya que este no pertenece a un carácter de ID válido se puede filtrarlo; es decir descartar esta lectura y volver a leer.

4.1.2 PRUEBAS DEL SENSOR DE TEMPERATURA.

Pruebas realizadas entre el DS1820 y controlador. Esta prueba realizada nos permite ver la trama del protocolo 1 wire

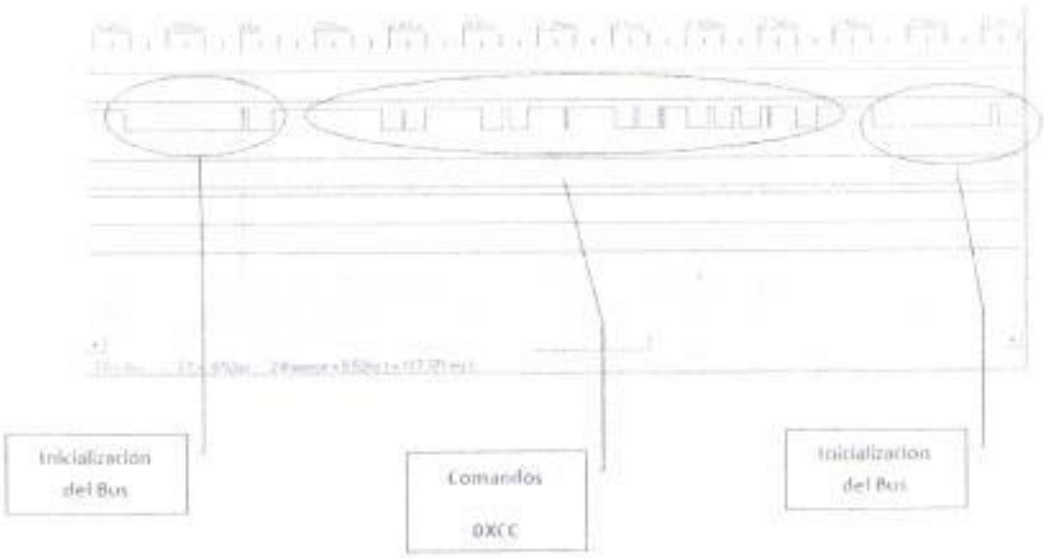


Figura 4.10 Trama 1-wire

Observando la trama se verifica el protocolo 1-wire el mismo que comienza inicializando el bus, enviando luego los comandos, inicializando otra vez, enviando comando, y luego recibiendo el dato de temperatura.

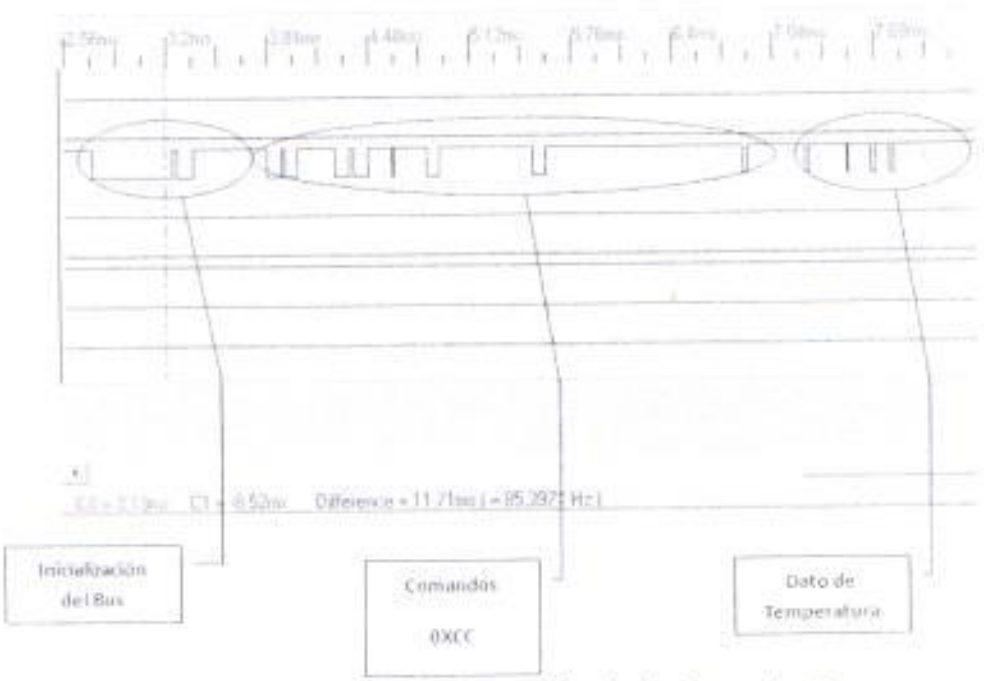


Figura 4.11 Continuación de la trama 1-wire.

Como observación adicional se verifica que el tiempo de la trama 1-wire es de aproximadamente 8ms.

Pruebas realizadas entre el NXT y controlador. En la Figura 4.12 es la prueba que servirá para observar la transferencia de datos entre el controlador del Lego MindStorm NXT y el controlador del módulo sensor de temperatura.

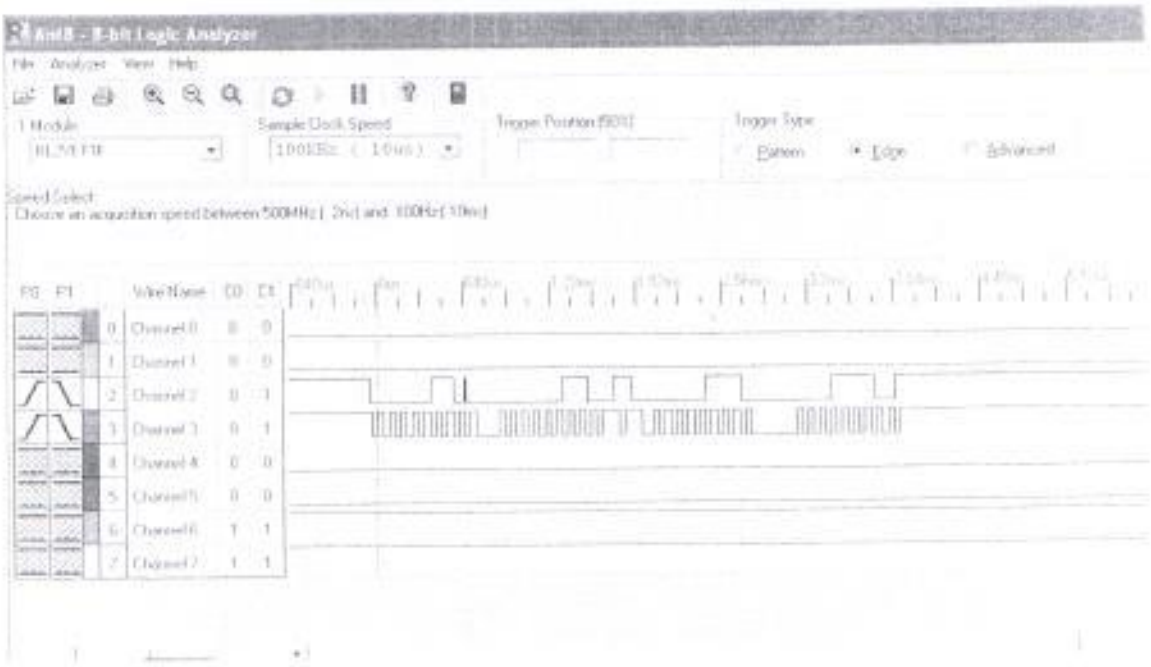


Figura 4.12 Trama general del protocolo I2C

La trama que se observa se trasmite cada 1 segundo. Inicialmente se envia la dirección del esclavo junto con la acción que va a realizar el maestro en el último BIT, que en el caso de la figura muestra escritura, seguido se presenta el comando 3 que es enviado al esclavo, luego el maestro envia la dirección del esclavo y la acción de lectura, observando la repuesta del esclavo que es el último byte, que es el valor de la temperatura.

4.2 PRUEBAS DEL SISTEMA

La primera prueba realizada del sistema es en modo de captura de datos de temperatura en tiempo real. Para entrar en este modo de funcionamiento se

tiene que marcar la casilla de chequeo etiquetada como "Tiempo Real", luego de lo cual se mostrará el siguiente formulario.

Figura 4.13 Pantalla modo de captura de datos de temperatura en tiempo real

Se puede observar la casilla de chequeo etiquetada como "Tiempo Real" esta seleccionada. Además se tiene el tiempo de muestreo el cual se mide en segundos y tiene un máximo de 450 s (7,5 minutos) ya que este es el máximo tiempo que se puede representar en el plano sobre el que esta dibujando.

Además en esta sección también se tiene el paso o frecuencia con la que se toma las lecturas de temperatura, la misma que esta en segundos.

También se tiene la sección que nos indica la hora en la que se inicia la captura, la que indica la hora en la que finaliza la captura y el nombre de la muestra que

se esta tomando los datos de temperatura. Se puede observar que antes de iniciar la captura estas etiquetas no tienen asignados los valores.

Adicional se tiene disponible el botón que nos permite iniciar la captura de datos en tiempo real. La siguiente Figura 4.14 muestra los datos que están siendo capturados después de presionar el botón "INICIAR".

Figura 4.14 Inicio de captura de datos en modo tiempo real

Se muestra que después de iniciar la captura, se activa el botón "PARAR" si se desea detener el muestreo en cualquier momento, o dejar que cumpla con los

tiempos establecidos previamente en los parámetros de inicio. Cuando termina el muestreo, se presenta en la Figura 4.15.

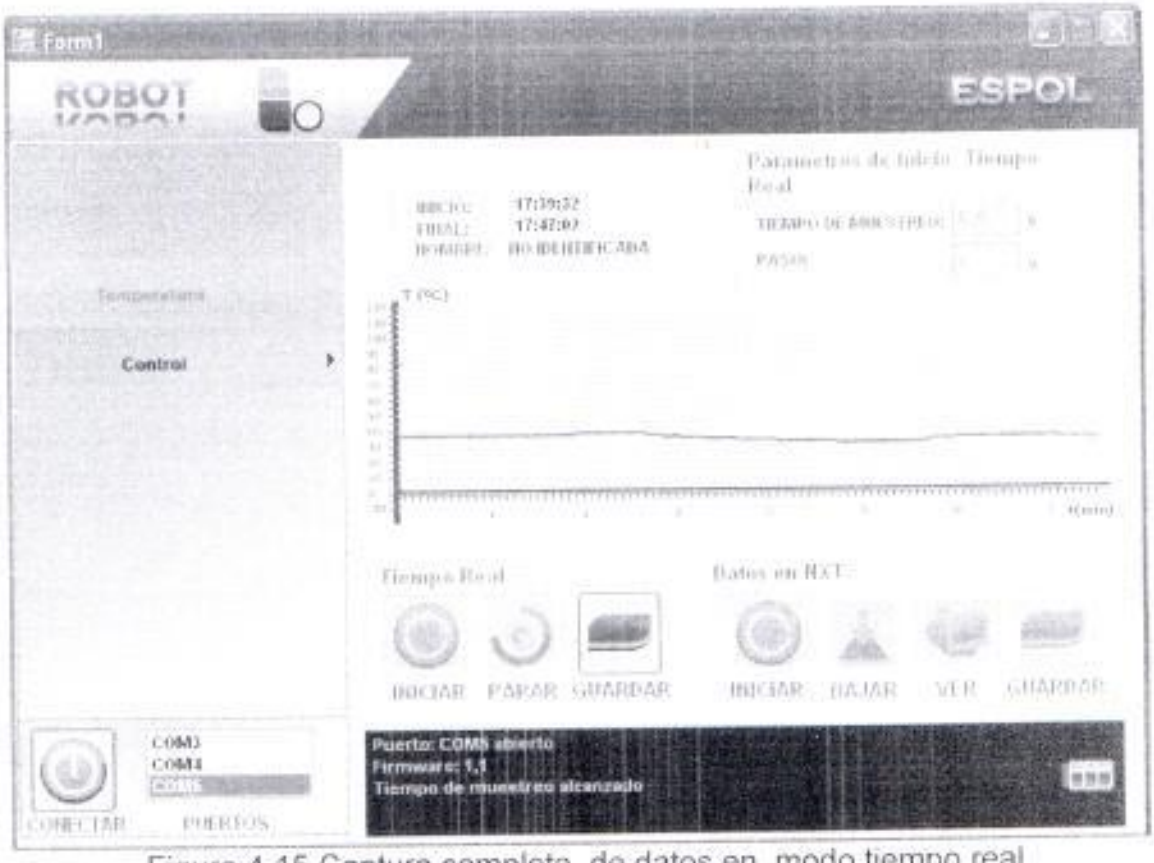


Figura 4.15 Captura completa de datos en modo tiempo real

Al finalizar se muestra el botón "GUARDAR" y los datos de los tiempos de inicio y fin de la captura de datos y el nombre de la muestra. Después de presionar el botón "GUARDAR", los datos se almacenan en un archivo en la unidad

C:\DatosNXT\ con el nombre "174747-08-22-2008.txt", en el cual se muestra la hora del día y la fecha en el momento de guardar.

Modo de captura de datos de temperatura independiente del PC.

Inicialmente cuando se ingresa en este modo, se muestra en el siguiente gráfico.

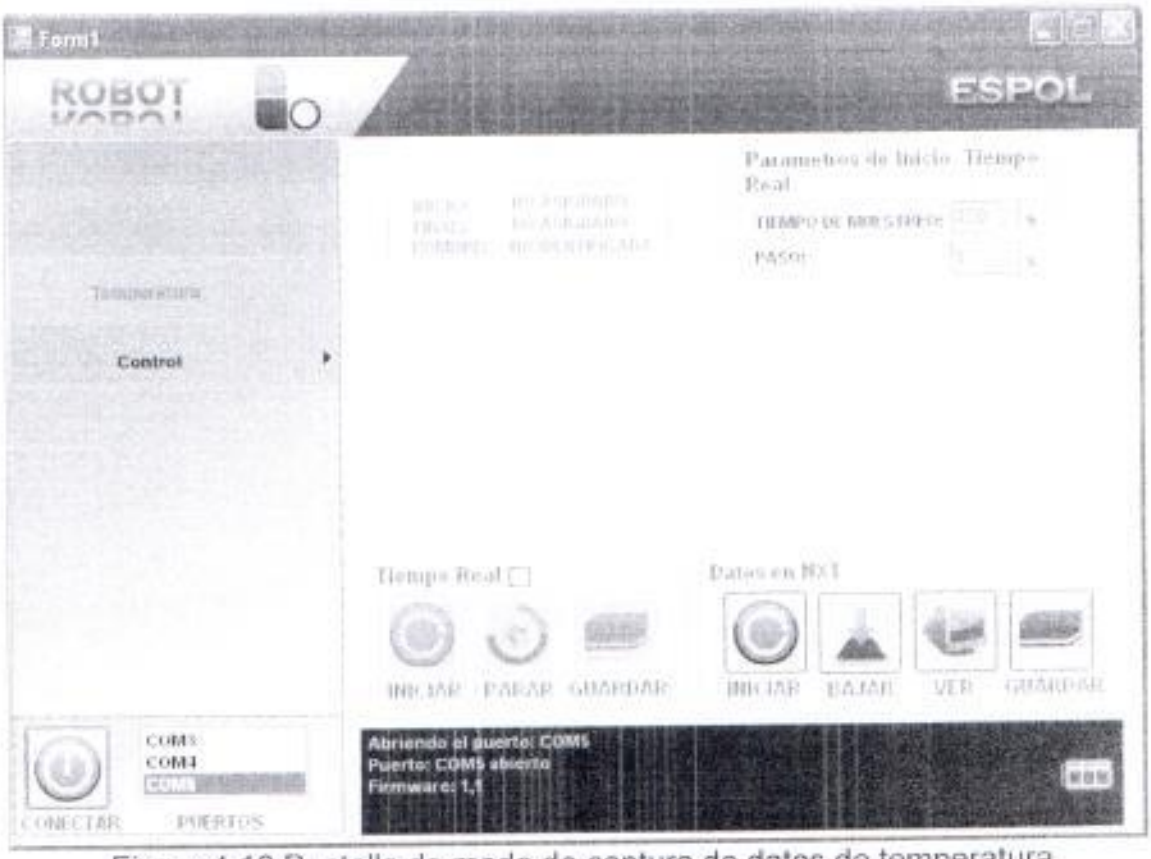


Figura 4.16 Pantalla de modo de captura de datos de temperatura independiente del PC

Cuando no esta seleccionada la casilla de chequeo etiquetada como "Tiempo Real", se entra en el modo de funcionamiento en el modo de captura de datos de temperatura independiente del PC.

En este modo se activa el botón "INICIAR" y cuando se lo presiona se activa un evento en el que se ordena que un programa "ds1820.rxe" que se encuentra en el robot se inicie, luego de lo cual se monitorea si este programa sigue en ejecución como se muestra a continuación .

Figura 4.17 Captura de datos de temperatura independiente del PC en progreso

Cuando finaliza la captura autónoma se muestra en la Figura 4.18.

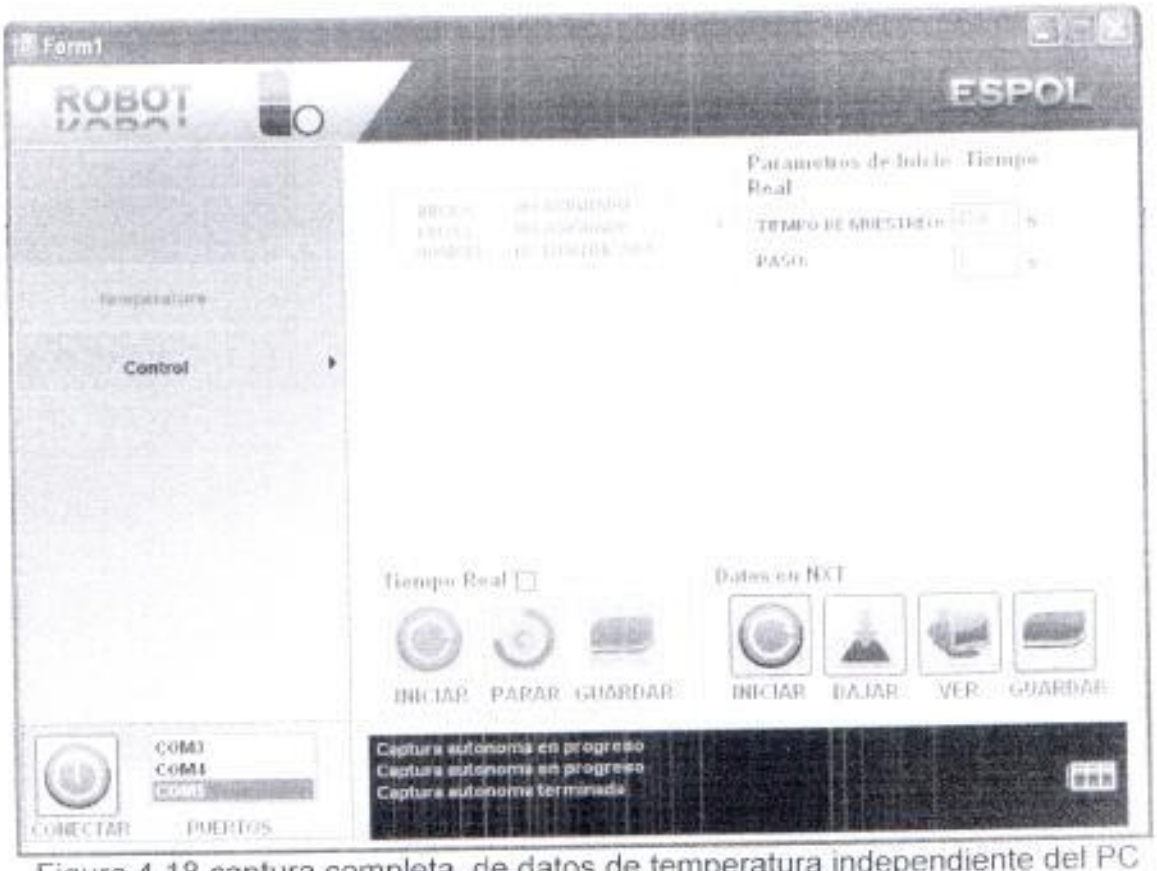


Figura 4.18 captura completa de datos de temperatura independiente del PC

Las opciones que se tiene disponibles luego de terminada la captura remota son:

Bajar: Lo que realiza esta acción es que baja un archivo que se encuentra en el robot llamado "datos.txt" el mismo que contiene los datos de la ultima captura autónoma realizada y los almacena en la PC en "c:\datos.txt".

Ver. Este botón lo que realiza es graficar los datos que se encuentran en el archivo que se encuentra en "c:\datos.txt".

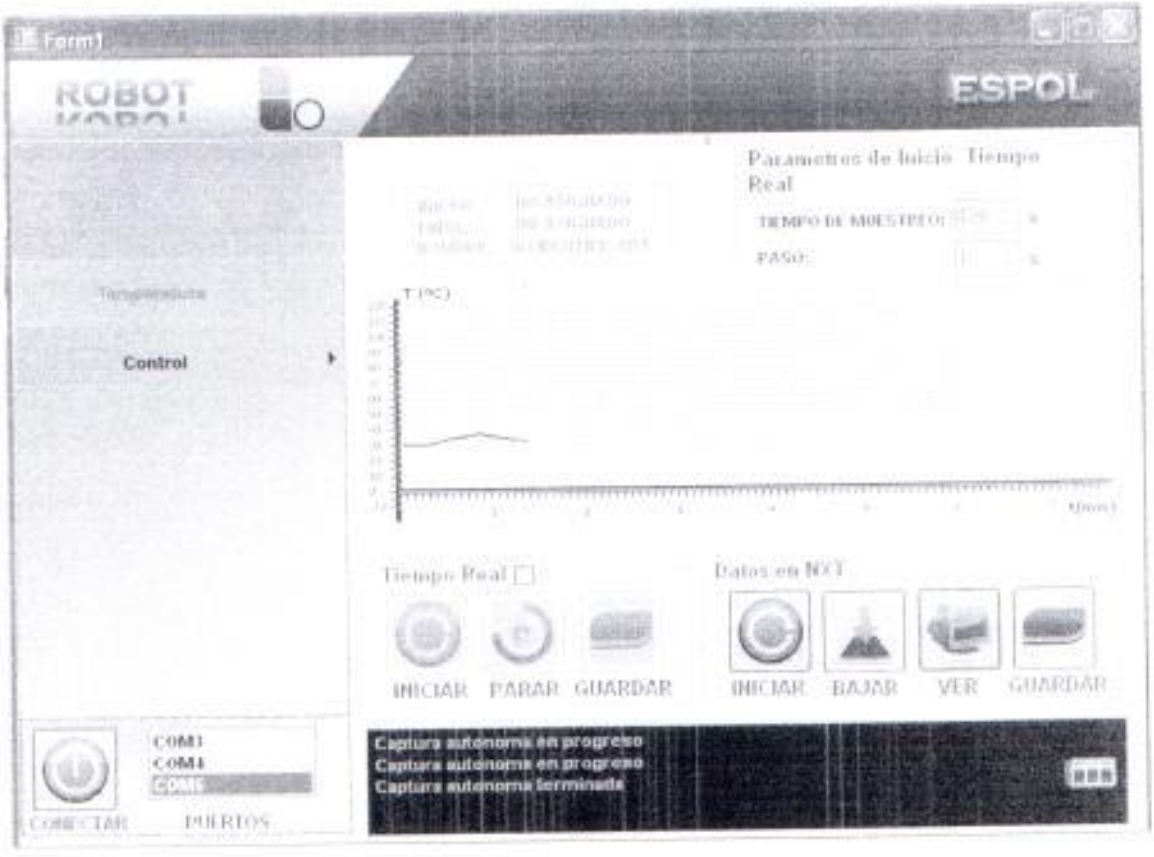


Figura 4.19 Visualización de la captura completa independiente del PC

Guardar: este botón permite guardar el archivo que se encuentra en "c:\ datos.txt" en otra ubicación.

Modo de lectura de la etiqueta RFID

Los controles para proceder a la lectura de la etiqueta RFID se encuentran en el formulario de control, en donde se observa el botón "LEER ID".

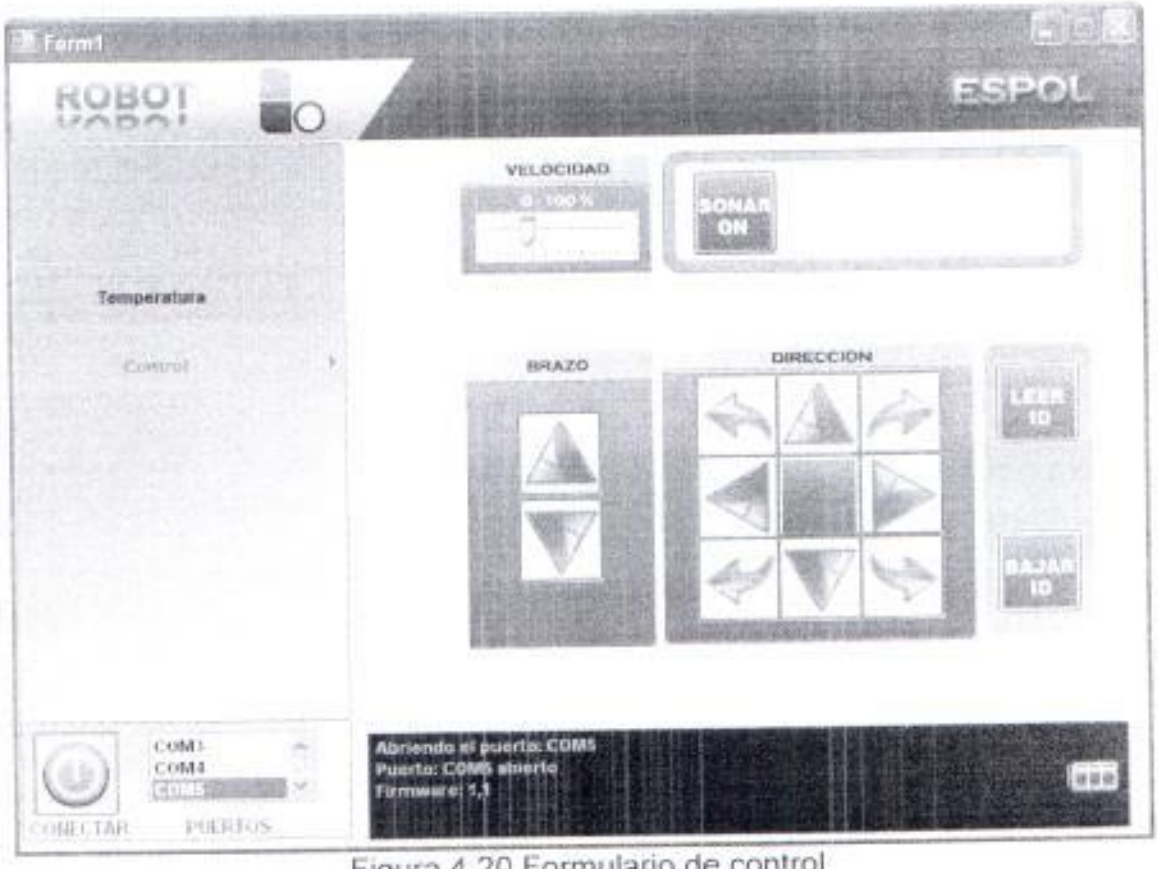


Figura 4.20 Formulario de control

Al presionar el botón "LEER ID" se visualiza en la consola de mensajes el icono naranja de lectura RFID. El lector RFID entra en un estado de espera de la etiqueta, esto se lo evidencia observando el led del lector RFID cambia a color rojo.

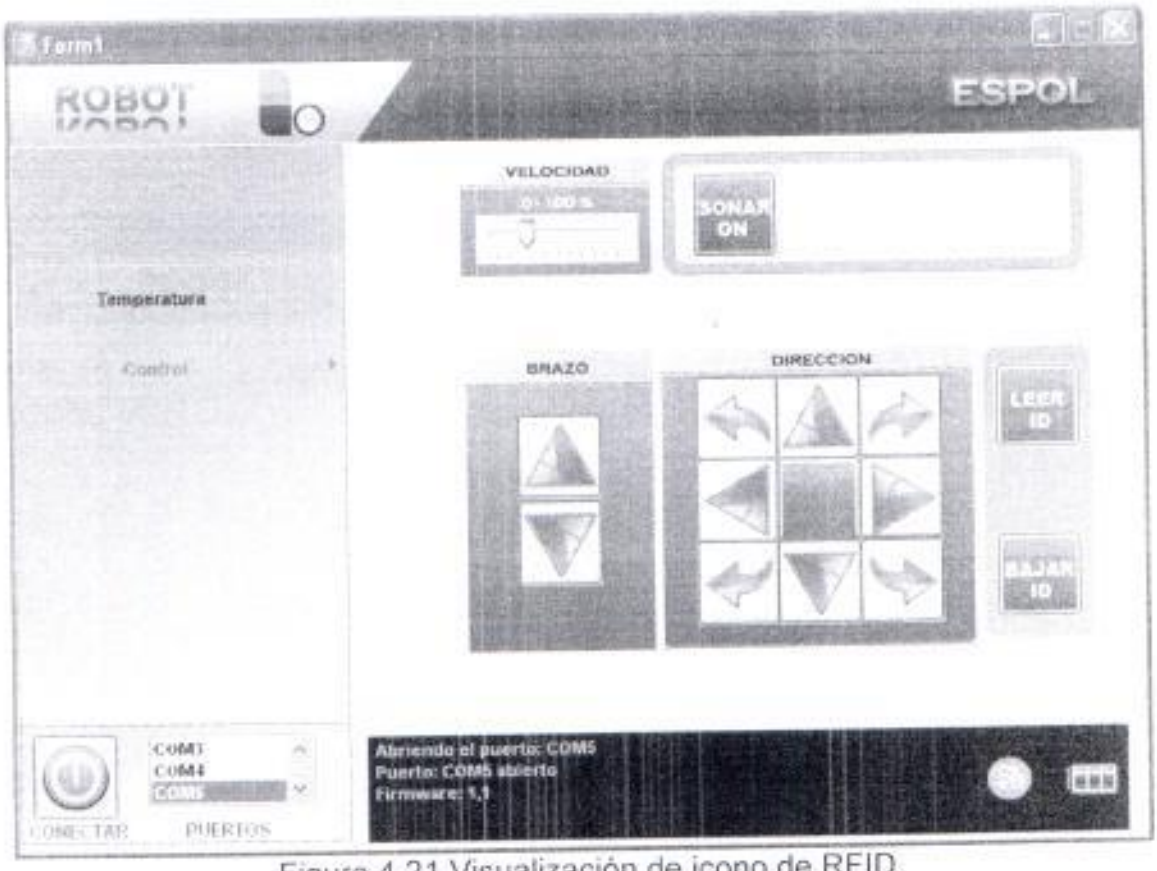


Figura 4.21 Visualización de icono de RFID

Cuando se aproxima la etiqueta en el rango de lectura lee dicha etiqueta y almacena los datos en la eeprom del controlador RFID, a su vez el led cambia a color a verde.

Luego de esto se procede ha recuperar el valor del ID de la etiqueta RFID para lo cual se presiona el botón "BAJAR ID" el mismo que mostrará en la consola de mensajes el valor del ID de la etiqueta almacenada en la eeprom, si la muestra

pertenece a las almacenadas en la base de datos también mostrara su identificación.

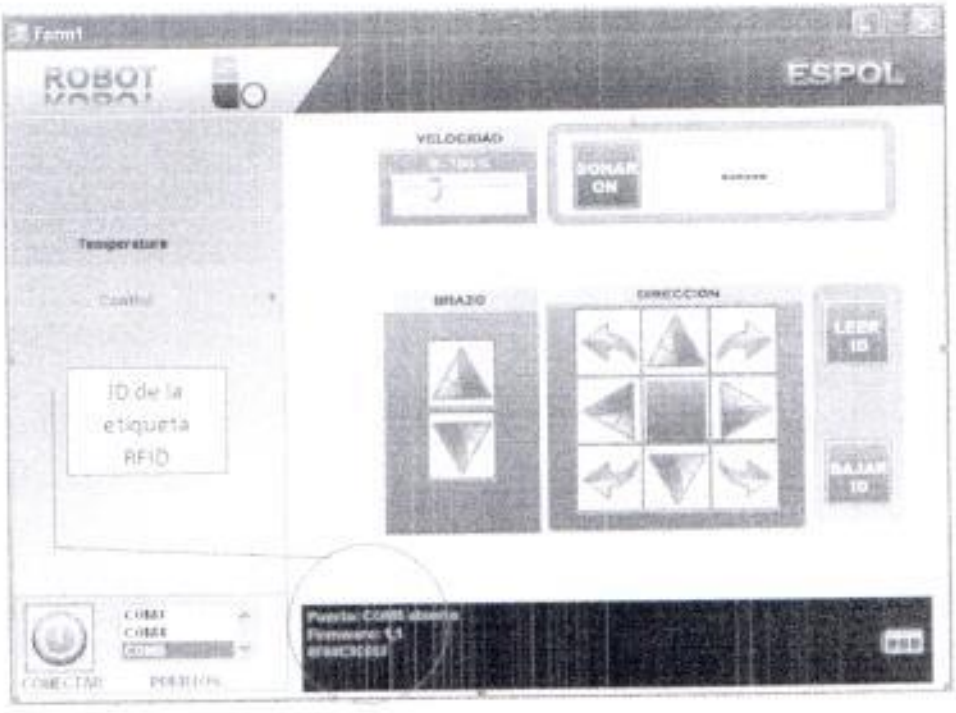


Figura 4.22 Valor del ID de la etiqueta RFID

En caso de falla de la alimentación de energía del módulo RFID o alguna inhibición del Módulo lector RFID el sistema nos mostrara un mensaje de alerta en la consola de mensajes.

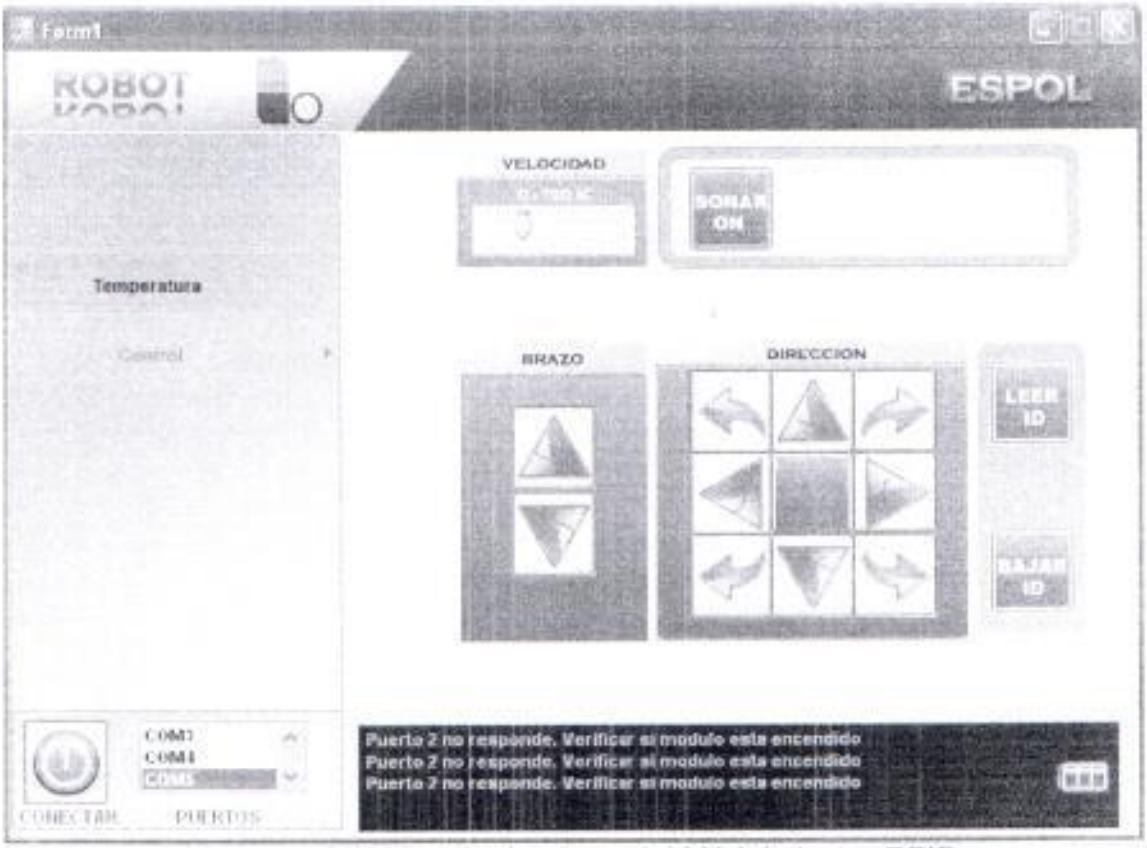


Figura 4.23 Mensaje de alerta del Módulo lector RFID

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES.-

Una vez que el sistema de adquisición de datos fue implementado junto con el lector RFID y sensor de temperatura a un robot Mindstorm NXT se puede obtener las siguientes conclusiones:

1. El sistema de invención robótica nos permite construir prototipos de una manera sencilla y reusable, lo cual nos parece muy importante para su aprovechamiento en los laboratorios de ingenieria.

2. Las señales de los puertos del NXT dan la apertura de comunicación con diversos dispositivos que manejen los protocolos requeridos, lo mismo que permite el diseño de nuevos módulos.

3. La utilización de comandos directos dan un beneficio de interacción con una máquina virtual que se encarga de manejar los recursos del hardware del NXT.

4. La implementación de las nuevas interfaces, la lectora de etiqueta RFID y el módulo del sensor de temperatura fueron exitosas y nos permitieron proveer nuevos servicios al Lego Mindstorm NXT.

5. La tecnologia multi-tarea que posee NXT nos da la posibilidad de manejar varios recursos del robot al mismo tiempo debido al sistema de ejecución de tareas en paralelo, sin la preocupación de mecanismos de interrupciones o similares.

6. Se demuestra que la toma de datos en tiempo real con los retardos en las comunicaciones utilizados no afectan en la precisión de los datos adquiridos en tiempo de aproximadamente 1 segundo.

7. La lectura de la etiqueta por parte del lector RFID fue bastante sencilla y llegaron a considerarse fácil y fiable.

RECOMENDACIÓN

1. Para el correcto funcionamiento del sistema es requerido, utilizar etiquetas compatibles con el lector RFID, las baterías y/o fuente de alimentación deben tener la carga de energía requerida, las temperaturas a ser muestreadas deben encontrase el rango de medición de temperatura.

2. Para futuros proyectos se podría reemplazar el sensor (ds1820) por otro sensor de temperatura con características específicas de acuerdo a las necesidades del proyecto.

3. Implementar funciones que permitan que la base de datos de las etiquetas RFID sea de una dimensión variable con el objetivo de ingresar más datos o en su defecto eliminarlos.

4. En trabajos posteriores se podría incorporar un lector de RFID de etiquetas activas para ampliar el rango de lectura de las etiquetas RFID.

5. Desarrollar un módulo de memoria externa con el objetivo de almacenar mayor número de datos en la operación de medir la temperatura en forma independiente del PC.

APENDICES

APENDICE A

DETALLE DE LOS COMPONENTES DEL LEGO MINDSTORM NXT

El Lego Mindstorm NXT tiene 3 puertos de salida que sirven para conectar los motores. La interfaz para interactuar con los motores consta de 6 hilos, cuya descripción se muestra a continuación.

Figura A.1. Puerto de salida

MA0 y MA1: Son las señales de salida que controlan los actuadores (motores). Estas señales internamente son controladas por un driver que puede suministrar 700 mA para cada puerto de salida, con una corriente pico de 1A.

La señal de salida del driver es PWM y adicionalmente posee una protección térmica, que en caso de sobrecarga la controla ajustando la corriente de salida.

POWERMA: Es una fuente de poder que está conectada a todos los puertos de salida y entrada, la cual proporciona un voltaje de 4.3 V y cuya máxima corriente de salida que puede manejar es 180 mA, lo cual significa que cada puerto tiene aproximadamente 20 mA.

TACHOA0 y TACHOA1: Son puertos de entrada conectados al procesador ARM7 los cuales posibilitan tener un detector de cuadratura el cual es usado para contar los pulsos del tacòmetro del motor y detectar el sentido de giro del motor.

Lego Mindstorm NXT tiene 4 puertos de entrada a los cuales van conectados los sensores. Posee una interfaz de 6 hilos que posibilita comunicación tanto digital como analógica.

Tanto los puerto 1, 2, 3 y 4 tienen un esquemático idèntico al que se presenta a continuación salvo que en el puerto 4 los pines DIG1A10 y DIG1A11 están conectados a un controlador RS485.

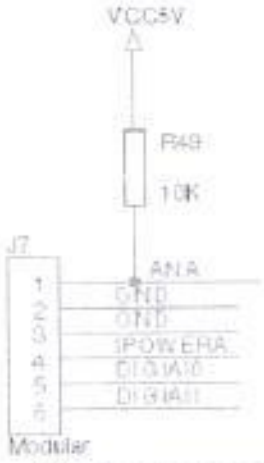


Figura A.2. Puerto de entrada

ANA: El pin 1 (ANA) es un pin de entrada analógica que está conectado a un convertidor A/D de 10 bits dentro del procesador AVR. Adicionalmente también

tiene una doble función ya que está conectado a un generador de corriente usado para proporcionar energía a los sensores.

POWERMA: Es una fuente de poder que está conectada a todos los puertos de salida y entrada, la cual proporciona un voltaje de 4.3 V y cuya máxima corriente de salida que puede manejar es 180 mA, lo cual significa que cada puerto tiene aproximadamente 20 mA.

DIG1A10 y DIG1A11: Son usados para comunicación digital implementada usando el protocolo I2C a 9600b/s. El NXT solo puede funcionar como maestro en la comunicación I2C y requiere que los dispositivos externos incluyan las resistencias Pull-up.

Los sensores en un robot son el mecanismo que permite al robot medir cantidades físicas como distancia, sonido, luz, temperatura, etc. Dependiendo en gran medida de estos para percibir el entorno que lo rodea.

Además del bloque programable NXT este kit de robótica viene con 5 tipos de sensores que son: sensor de tacto, sensor de sonido, sensor de luz, sensor ultrasónico y sensor de rotación. Cabe notar que el sensor de rotación se encuentra dentro del motor con el propósito de medir la posición del eje y la velocidad del motor.

Una forma de clasificar los sensores es en: sensores activos, sensores pasivos y sensores digitales.

Los sensores activos son aquellos que producen un estimulo y miden una respuesta de su entorno. para lo cual necesitan de la alimentación proporcionada por un generador de corriente incorporado al bloque programable del NXT el cual provee de 18mA de corriente de salida cada 3 ms y mide el valor análogo de la respuesta durante 0,1 ms .

Los siguientes son los sensores activos:

- Sensor de luz
- Sensor de rotación

Los sensores pasivos son aquellos que simplemente miden las señales del entorno y no necesitan de alimentación especial. Estos sensores son muestreados cada 3ms.

Los siguientes son los sensores pasivos:

- Sensor de tacto
- Sensor de sonido

Todos los sensores que utilizan comunicación I2C son llamados sensores digitales ya que utilizan un microcontrolador para procesar los datos obtenidos del entorno físico.

Un ejemplo de sensor digital:

- Sensor ultrasónico

Con el objetivo de facilitar la interfaz de usuario del bloque programable del NXT (Brick), este tiene incorporado una pantalla grafica (LCD) monocromática con una resolución de 100X64 pixeles. Otra caracteristica importante del LCD es el que es continuamente actualizado linea a linea, requiriendo 17 ms para actualizar la pantalla completa.

El la Figura A.3 se muestra el sistema de coordenadas utilizado como referencia en el LCD.

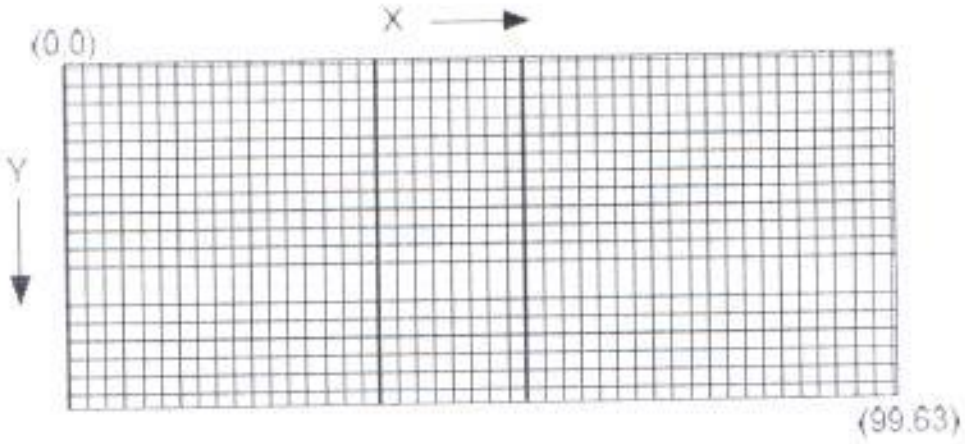


Figura A.3 Ejes de referencia del LCD

Lego Mindstorm NXT tiene incorporado el chip CSR BlueCore 4 versión 2 que maneja la pila de comunicación Bluetooth. Este chip permite que el NXT pueda conectarse inalámbricamente a tres dispositivos al mismo tiempo pero sólo puede comunicarse con un dispositivo a la vez, utilizando para ello un tipo de comunicación llamada SPP (Serial Port Profile). Con el objetivo de reducir el

consumo de energía se ha implementado la tecnología Bluetooth Class II lo que significa que pueden comunicarse a una distancia aproximada de 10 metros.

El bloque programable del NXT puede funcionar como maestro/esclavo en la comunicación Bluetooth. En una red Bluetooth un bloque programable debe desempeñar el papel de maestro y los otros bloques programables dentro de la red deben comunicarse a través de el.

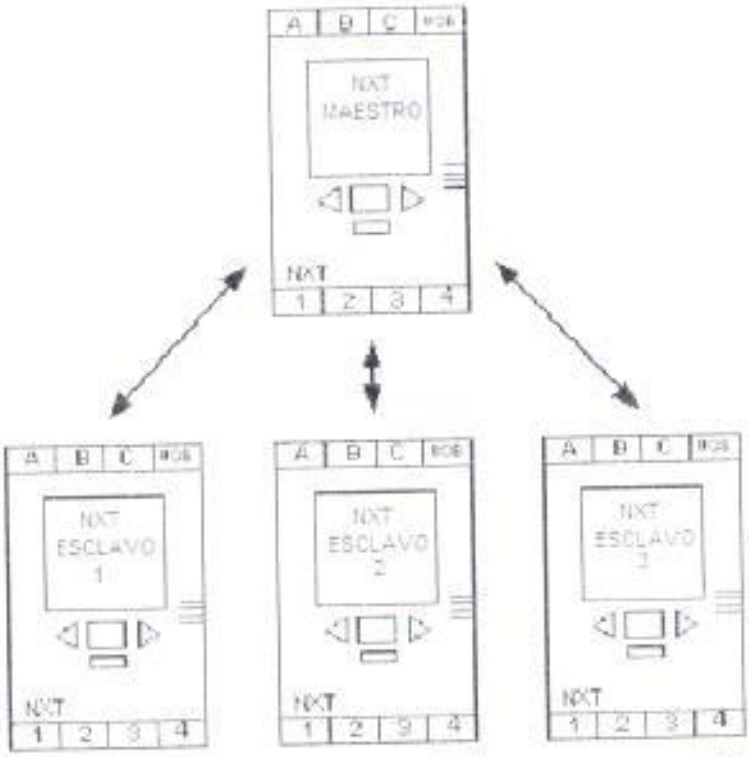


Figura A.4 Red Bluetooth entre 4 bloques programables.

La comunicación Bluetooth entre los dispositivos se realizan a través de canales. El NXT tiene 4 canales de comunicación. El canal 0 es usado por los

dispositivos NXT esclavos para enviar información hacia el maestro NXT, mientras los canales 1, 2 ,3 son usados para enviar información desde el maestro hacia los dispositivos esclavos.

Para la función del sonido el bloque programable del NXT incluye un parlante el cual es alimentado por un chip amplificador de sonido. La señal de sonido es PWM y es controlada por el procesador ARM7. La Tabla A.1 muestra el consumo de energía del parlante.

FRECUENCIA	CORRIENTE mA	Potencia mW
440 Hz	102	169
4 KHz	78	97

Tabla A.1 Tabla de consumo de energía del parlante

En la Figura A.5. se muestra el diagrama de circuitería interna del parlante.

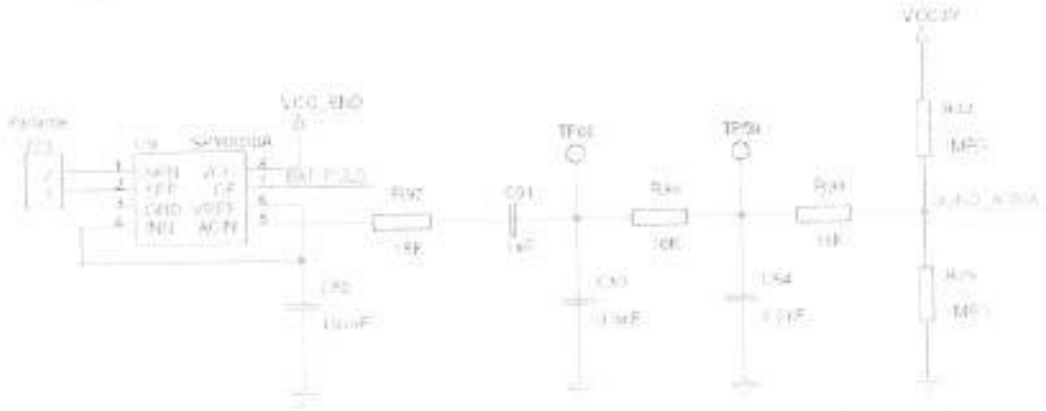


Figura A.5. Diagrama esquemático del sistema de sonido.

APENDICE B

OPERACIONES BÁSICAS DE VISUAL BASIC NET.

Las variables representan espacios de memoria para almacenar valores de tipos determinados, los cuales pueden ser modificados en los bloques donde las variables son accesibles. El formato básico de la declaración de una variable es el siguiente:

Dim nombre de variable As Tipo de la Variable

Generalmente una variable declarada dentro de un bloque es accesible sólo dentro de ese bloque, entendiendo por bloque el código que finaliza con End, Loop o Next. Existen también modificadores de acceso que indican el tipo de acceso de la variable.

Las variables son iniciadas por el compilador al valor por omisión según su tipo o pueden ser iniciadas explícitamente con un valor.

A los tipos de las variables se los puede clasificar en:

- Tipos enteros: Byte, Short, Integer, Long y Char.
- Tipos Reales: Single, Double y Decimal.
- Tipo Boolean.

La Tabla B.1 muestra los tipos de datos en Visual Basic .NET

TIPO DE VISUAL BASIC	ESTRUCTURA DE TIPO COMMON LANGUAGE RUNTIME	ASIGNACIÓN DE ALMACENAMIENTO NOMINAL	INTERVALO DE VALORES
Boolean	Boolean	En función de la plataforma de implementación	True o False
Byte	Byte	1 byte	0 a 255 (sin signo)
Char (carácter individual)	Char	2 bytes	0 a 65535 (sin signo)
Date	DateTime	8 bytes	0:00:00 (medianoche) del 1 de enero de 0001 a 11:59:59 p.m. del 31 de diciembre de 9999.
Decimal	Decimal	16 bytes	0 a +/- 79.228.162.514.264.337.593.543.950.335 (+/-7,9... E+28) † sin separador decimal; 0 a +/- 7,9228162514264337593543950335 con 28 posiciones a la derecha del decimal; el número distinto de cero más pequeño es +/- 0,0000000000000000000000000001 (+/-1E-28) †
Double (punto flotante de precisión doble)	Double	8 bytes	-1,79769313486231570E+308 a -4,94065645841246544E-324 † para los valores negativos; 4,94065645841246544E-324 a 1,79769313486231570E+308 † para los valores positivos
Integer	Int32	4 bytes	-2.147.483.648 a 2.147.483.647 (con signo)
Long (entero largo)	Int64	8 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 (9,2... E+18 †) (con signo)
Object	Object (clase)	4 bytes en plataforma de 32 bits 8 bytes en plataforma de 64 bits	Cualquier tipo puede almacenarse en una variable de tipo Object

SByte	SByte	1 byte	-128 a 127 (con signo)
Short (entero corto)	Int16	2 bytes	-32.768 a 32.767 (con signo)
Single (punto flotante de precisión simple)	Single	4 bytes	-3,4028235E+38 a -1,401298E-45 † para los valores negativos; 1,401298E-45 a 3,4028235E+38 † para los valores positivos
String (longitud variable)	String (clase)	En función de la plataforma de implementación	0 a 2.000 millones de caracteres Unicode aprox.
UInteger	UInt32	4 bytes	0 a 4.294.967.295 (sin signo)
ULong	UInt64	8 bytes	0 a 18.446.744.073.709.551.615 (1,8...E+19 †) (sin signo)

TIPO DE VISUAL BASIC	ESTRUCTURA DE TIPO COMMON LANGUAGE RUNTIME	ASIGNACIÓN DE ALMACENAMIENTO NOMINAL	INTERVALO DE VALORES
User-Defined (estructura)	(hereda de ValueType)	En función de la plataforma de implementación	Cada miembro de la estructura tiene un intervalo de valores determinado por su tipo de datos y es independiente de los intervalos de valores correspondientes a los demás miembros.
UShort	UInt16	2 bytes	0 a 65.535 (sin signo)

TABLA B.1 TIPOS DE DATOS EN VISUAL BASIC .NET

Otro aspecto importante a tomar en cuenta es la forma en que son almacenados los valores en las variables, para lo cual se utiliza el siguiente formato:

Variable operador de asignación Valor

En la cual se evalúa lo que esta a la derecha del operador de asignación y el resultado se asigna a la variable en la izquierda.

Cuando se le asigna un valor numérico a una variable se esta colocando ese valor en una localidad de memoria asociada con esa variable, y si esta variable ya tenia asignado un valor este es destruido por el valor nuevo. En cambio las cadenas de caracteres, por ejemplo, son almacenadas en objetos y en la variable lo que se almacena es una referencia (Dirección de memoria) a ese objeto.

Los comentarios son mensajes que utilizan para documentar el código fuente los cuales no son compilados.

Un comentario comienza con una comilla simple (').

Es un conjunto de operandos unidos por operadores para expresar una operación aritmética o lógica.

Cuando se tiene en una operación aritmética operandos de diferentes tipos, los operandos de precisión más baja se convierten a los de más alta.

El resultado obtenido de una operación aritmética es convertido implícita o explícitamente al tipo de variable que almacena dicho resultado.

En Visual Basic .NET se tiene los siguientes operadores aritméticos y lógicos que se muestran en la Tabla B.2

VISUAL BASIC .NET
And
AndAlso
Or
OrElse
Xor
Not
=
<>
& (concatenación de cadenas)
\ (división de números enteros)
\=
Mod
Is Nothing

Tabla B.2 Operadores aritméticos y lógicos:

Cuando en una expresión aritmética participan varios operadores, se ejecutan de izquierda a derecha y de mayor a menor prioridad.

Se puede decir que programación orientada a objetos es un paradigma de programación (Enfoque para la construcción de software) que usa objetos y sus interacciones para diseñar programas de computadora.

Una clase es una generalización de un tipo específico de objetos, su estructura, la definición de los elementos que conforman el objeto. Un objeto es una instancia de una clase.

Para crear un objeto de una determinada clase se utiliza el operador New, por Ej.

Dim cuenta01 As CCuenta = New CCuenta( )

Lo que hace la línea anterior es crear un objeto de la clase CCuenta y asigna a cuenta01 una referencia al mismo. En la programación orientada a objetos un mensaje está relacionado con un método, ya que cuando un objeto recibe un mensaje, la respuesta de ese mensaje es ejecutar el método asociado.

Una clase se compone de dos partes:

Atributos: Que son los datos que se refieren al estado del objeto

Métodos: Es la forma de operar sobre los atributos.

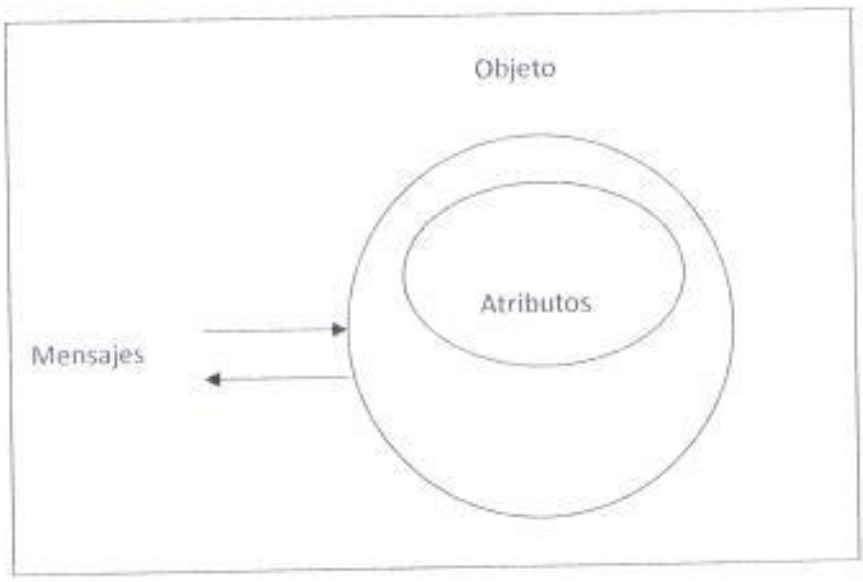


Figura B.1 Partes de un objeto

Normalmente los atributos se ocultan a los usuarios de los objetos, siendo los mensajes los que se comunican con los métodos, los que a su vez operan sobre los atributos.

Para declarar una clase se utiliza la palabra reservada Class seguida del nombre de la clase, luego viene el cuerpo de la clase el cual está entre la palabra Class y End Class. Ej.

```
Class CCuenta
'Cuerpo de la clase: Atributos y métodos
End Class
```

Los atributos que tienen el modificador Private (Dim se supone Private) son accesibles solamente por los métodos de su propia clase .Ej.

```
Class CCuenta
Private nombre as string
End Class
```

El otro componente de la clase son los métodos, los cuales son rutinas de código, que se ejecutan cuando es invocado por algún mensaje. Ej.

```
Class CCuenta
Private nombre as string
Public Sub ponerNombre (nom as String)
```

Nombre = nom

El método consta de su nombre precedido por Sub si no devuelve un valor y por Function si devuelve valor.

End Sub

End Class

Si el método es declarado publico (Public), este es accesible para cualquier otra clase y por omisión un método se supone Public.

En términos generales para acceder a un miembro de un objeto (atributo o método) se utiliza la siguiente sintaxis:

nombre_objeto.nombre_miembro

El constructor es un método especial de una clase que es llamado automáticamente cuando se crea un objeto de una clase y cuya función es inicializar un objeto.

El constructor se distingue porque tiene el nombre New

Public Sub New

End Sub

Cuando en una clase se define varias veces un mismo método con diferente número de parámetros, o con el mismo número de parámetros con al menos un tipo diferente se dice que el método esta sobrecargado.

La herencia permite que una clase herede los atributos y métodos de otra clase (los constructores no se heredan).

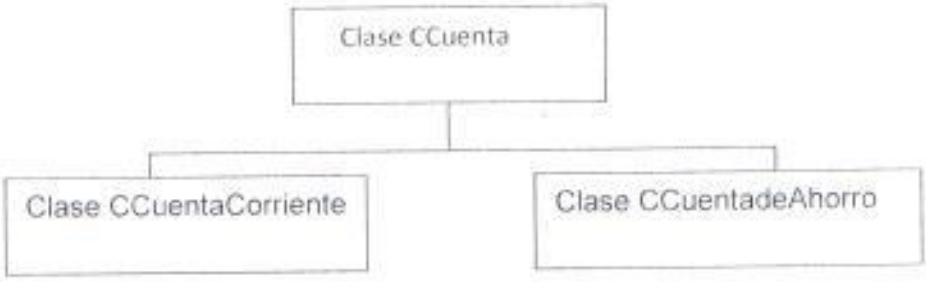


Figura B.2 Herencia de clases

La herencia se implementa de la siguiente forma:

Class CCuentaAhorro

Inherits CCuenta

End Class

Una subclase puede redefinir cualquier método heredado de su clase padre. Redefinir un método heredado significa volverlo a escribir en la subclase con el mismo nombre, la misma lista de parámetros y el mismo tipo de valor retornado que tenia en la superclase.

Para referirse a un miembro oculto de la clase base se utiliza:

MyBase.miembro_oculto

Por ejemplo si se quiere invocar al constructor de la clase base se utiliza:

MyBase.New (p1, p2...)

Otras consideraciones con respecto a la herencia son:

Una subclase no tiene acceso directo a los miembros privados (Private) de su superclase.

Una subclase puede añadir sus propios atributos y métodos. Si el nombre de alguno de estos coincide con el de un miembro heredado, este ultimo queda oculto para la subclase.

APENDICE C

NXC

NXC es sensible a las mayúsculas.

Dos tipos de comentarios son soportados por NXC.

Multilinea: /* Hola */

Una sola linea: // Hola

Las constantes pueden ser escritas en Decimal o Hexadecimal. Por ejemplo:

X=10

X= 0x10

Un programa en NXC esta compuesto de bloques de código y variables. Hay dos tipos de bloques de código: tareas y funciones.

NXT soporta Multi-threading es decir múltiples tareas se pueden ejecutar al mismo tiempo. Las tareas se definen usando la palabra "task" Ejm:

```
task name ()
{
}
```

Un programa debe tener al menos una tarea llamada "main" la cual es el punto de entrada de un programa. Puede haber hasta 256 tareas.

Existen funciones API para manejar tareas como: precedes, follow, start, stop ,etc.

NXC soporta funciones con argumentos y retornan valores. Se definen utilizando la siguiente sintaxis:

[safecall] [inline] return_type name (argument_list)

{

}

Las funciones que no retornan valor son especificadas con "void".

Las listas de argumentos pueden estar vacias o contener una o más definiciones de argumentos, separados por comas.

NXC soporta pasar argumentos por valor, valor constante, por referencia, referencia constante.

Para pasar valores por referencia se usa: arg_type & Ejm:

void suma (int &x)

{

}

Una función definida como "inline" lo que hace es que cada vez que se invoca la función se realiza una copia del código en el programa.

En una función definida como "safecall" si un segundo hilo intenta llamar a esta función y otro hilo la esta ejecutando, este segundo hilo tendrá que esperar que termine la ejecución en el primero.

En NXC se tiene variables con los siguientes tipo como se muestra en la Tabla C.1.

Nombre de Tipo	Información
Booleano	8 Bit sin signo
Byte,caracter sin signo	8 Bit sin signo
Caracter	8 Bit con signo
Entero, sin signo	16 Bit sin signo
Entero, short	16 Bit con signo
Long, sin signo	32 Bit sin signo
Long	32 Bit con signo
Mutex	Tipo especial usado para uso exclusivo de codigos de acceso
Cadena	Arreglo de Byte
Estructura	Tipo de estructura definido por usuario
Arreglos	Arreglo de cualquier tipo

Tabla C.1 Tipos de variables NXC

Para declarar una variable primero se define el tipo deseado, luego el nombre y finalmente se puede ";" Ejm

int x;

Las variables globales son declaradas a nivel de programa (fuera de cualquier bloque). Su visibilidad empieza en la declaración y termina al final del programa. Las variables locales pueden ser declaradas dentro de funciones y tareas. Estas variables son accedidas solamente dentro de los bloques donde fueron definidas.

Una vez declarados las variables deben ser asignadas con los siguientes operadores.

OPERADOR	ACCION
=	ASIGNA VARIABLE A LA EXPRESION
+=	SUMA EXPRESION A LA VARIABLE
-=	RESTA EXPRESION DE LA VARIABLE
*=	MULTIPLICA VARIABLE POR EXPRESION
/=	DIVIDE VARIABLE POR LA EXPRESION
%=	ASIGNA VARIABLE AL REMANENTE DESPUES DE DIVIDIR POR LA EXPRESION
&=	COMPARACION AND DE LA EXPRESION A LA VARIABLE
|=	COMPARACION OR DE LA EXPRESION A LA VARIABLE
^=	COMPARACION EXCLUSIVA OR A LA VARIABLE
abs=	ASIGNA VARIABLE AL VALOR ABSOLUTO DE LA EXPRESION
sign=	ASIGNA VARIABLE AL SIGNO (-1,+1,0) DE LA EXPRESION
>>=	DESPLAZAMIENTO A LA DERECHA DE LA VARIABLE A LA EXPRESION
<<=	DESPLAZAMIENTO A LA IZQUIERDA DE LA VARIABLE A LA EXPRESION

Tabla C.2 Operadores aritméticos y lógicos NXC

NXC soporta tipos definidos por el usuario denominados estructuras.

```
struct person
{
string name;
int age;
}
```

Después de definirlas se puede usarlas de la siguiente forma:

```
person myperson;
myperson.age = 40;
```

Los arreglos se declaran de la misma forma que las variables pero con corchetes Ejm:

```
int my_Array [ ];
```

Para declarar arreglos de más de una dimensión simplemente se agrega corchetes Ejm:

```
int my_Array [ ] [ ];
```

Se puede inicializar un arreglo de la siguiente forma:

```
int x[ ] = {1,2,3,4};
```

También se puede inicializar arreglos con la función ArrayInit Ejm:

```
int myVector[ ];
ArrayInit (myVector, 0, 10);   // 10 ceros en myVector
```

La estructura de control más simple son las sentencias compuestas las cuales van encerradas entre llaves Ejm.

```
{
    x=1;
    y=2;
}
```

SENTENCIA	DESCRIPCION	SINTAXIS
IF	La sentencia "if" evalua una condición la cual si es verdadera ejecuta el cuerpo de la sentencia	if (condición) consecuencia; if (condición) consecuencia else alternativa;
WHILE	Es evaluada y si es verdadera el cuerpo es ejecutado	while (x<10) { y=x+1 }
DO – WHILE	La diferencia con while es que ejecuta el cuerpo al menos una vez	do cuerpo while (condición)
FOR	Siempre ejecuta la sentencia 1 , verifica la condición y ejecuta la sentencia 2	for (smt1; condición ; smt2) cuerpo
REPEAT	Ejecuta un Loop un numero determinado de veces	repeat (expresión) cuerpo
SWITCH	Es usado para ejecutar uno o varios bloques de código dependiendo del valor de la expresión	switch(x) {case 1 break;case 2: break;default break;}
GOTO	Fuerza al programa a saltar a la localidad especificada	my_loop: x++;goto my_loop;
UNTIL	Continúa el lazo hasta que la condición es verdadera	#define until(c) while (!c)) until (SENSOR1=1);

Tabla C.3 Estructuras de control

APENDICE D

MIKROBASIC

Sus instrucciones más elementales son las siguintes: Comentarios: se puede comentar el código usando el apostrofe "'". Los comentarios pueden ir solos en una linea, o a continuación de una instrucción cualquiera. No se permiten comentarios de más de una linea:

' Esto es un comentario A = 10

' y esto también....

Estos comentarios son los que nos permitirán realizar modificaciones o mejoras en el código sin necesidad de perder horas intentando comprender lo que se escribio tiempo atrás.

Identificadores: Los llamados "identificadores" son en realidad los nombres de nuestras variables, procedimientos, funciones, constantes, etc. En Mikrobasic los nombres de identificadores pueden contener letras desde la "a" hasta "z" y desde la "A" hasta "Z", el guión bajo ("_") y los digitos del "0" al "9".

El primer carácter no puede ser un digito. Además, los nombres de los identificadores no son "case-sensitive", es decir que "Total", "total" y "TOTAL" son nombres de la misma variable, y se puede usar cualquiera de ellos por igual.

Al igual que en otros dialectos Basic, las variables son objetos cuyo valor puede

cambiar con la ejecución del programa. Cada variable tiene un nombre único, que se ajuste a las reglas definidas mas arriba para los identificadores.

Deben declararse con dim antes de ser usadas, y su ámbito de validez varía de acuerdo a la sección en que se declaran

La sintaxis de dim es: dim lista de variables as tipo.

La lista de tipos disponibles incluyen Byte (8 bits, con valores de 0 a 255), Word (2 bytes, o 16 bits, con valores de 0 a 65535) y Longint (con valores desde -2147483648 a 2147483647).

No son los únicos, la lista completa de tipos se puede consultar en la documentación del compilador.

Algunos ejemplos validos de dim son:

- dim i, j, k as byte
- dim contar, temp as word
- dim cadena as longint[100]

Figura D.1 Codee Explorer Mikrobasic

Bucles: Se tiene 3 formas de lograr que un grupo de instrucciones se repitan:

1) FOR – NEXT : Se repite mientras variable sea menor que valor_final. En cada ciclo variable se incrementa en incremento o en 1 si step no esta presente.

for variable = valor_inicial to valor_final [step incremento]

Instrucciones a repetir

next variable

2) WHILE – END: Mientras que la expresión sea verdadera, las instrucciones a repetir se ejecutaran. Si dentro de esas instrucciones no existe alguna que modifique el resultado de expresión, el ciclo se repetirá eternamente. Como la expresión se evalúa al comenzar el ciclo, puede ocurrir que si al comenzar el

programa la expresión sea falsa las instrucciones a repetir no se ejecuten nunca.

while expresión

Instrucciones a repetir

Wend

3) DO – LOOP: En este caso, las instrucciones a repetir se repiten mientras la expresión sea verdadera. Al evaluarse al final del ciclo, el grupo de instrucciones a repetir se ejecutaran al menos una vez.

do

Instrucciones a repetir s

loop until expresión

Cualquiera es estas tres formas de crear un bucle puede ser interrumpida si dentro de las instrucciones a repetir se coloca la instrucción break. Al ejecutarla, el control del programa salta a la sentencia escrita inmediatamente a continuación de next, wend o loop until, según el caso.

Toma de decisiones: Existen dos instrucciones para la toma de decisiones dentro de un programa escrito en Mikrobasic:

1) IF – THEN – ELSE – ENDIF: Las instrucciones a continuación del then se ejecutan si la expresión es verdadera, y las que después del else si es falsa.

if expresión then

instrucciones

[elseinstrucciones]

end if

2) SELECT – CASE: Se utiliza para transferir el control a una rama del programa, basándose en una determinada condición. Consiste en una expresión y una lista de sus posibles valores. Las instrucciones a continuación del case else, si existen, se ejecutan en caso de que selector tome un valor para el que no exista una rama case disponible.

select case selector

case valor_1

instrucciones_1

...

case valor_n

instrucciones_n

[case else

Instrucciones por defecto]

end select

Saltos: También se tiene dos instrucciones que permiten saltos incondicionales.

1) GOTO: Generalmente desaconsejada, esta instrucción nos permite transferir el control a cualquier punto dentro del programa. A pesar de estar demostrado que es posible construir cualquier programa sin utilizar nunca esta instrucción, se utiliza con cierta frecuencia. El control del programa se transfiere a la linea que tenga la etiqueta utilizada en el goto:

for i = 0 to n

for j = 0 to m

...

if desastre

goto Error

end if

...

next j

next i

Error: ' código para manejar el error. . .

2) GOSUB – RETURN: Funciona de manera similar a GOTO, pero con la ventaja de que el control del programa regresa a la instrucción posterior a GOSUB luego de ser ejecutado el return. Esto la hace ideal para crear subrutinas que pueden ser llamadas varias veces y/o desde distintos puntos del programa

gosub mi_rutina

instrucciones

...

mi_rutina: 'código de la subrutina

...

Return

Es muy recomendable por cuestiones de claridad del código y sobre todo para facilitar su mantenimiento, que aquellos grupos de instrucciones que se utilicen mas de una vez dentro del programa, se escriban como funciones o procedimientos. Mikrobasic permite el uso de ambas, y esta es la sintaxis que se debe respetar:

Funciones:

sub function nombre_de_la_funcion(lista_de_parametros) as tipo

[declaraciones locales]

Cuerpo de la funcion

end sub

Procedimientos:

sub procedure nombre_del_procedimiento(lista_de_parametros

[declaraciones locales]

Cuerpo del procedimiento

end sub

BIBLIOGRAFIA

1. "Microcontrolador Pic16F84, Desarrollo de proyectos" ,Autor Enrique Palacios, Fernando Remiro, Lucas Lopez, Editorial Alfa Omega, , Edición Original, Año 2004

2. "Lenguaje de programación Visual Basic.NET", Autor Fco Javier Ceballos, Editorial Alfa Omega, Edición Original, Año 2003

3. "LEGO MINDSTORMS NXT Hardware Developer Kit", Autor The Lego Group, Revision. 1.00 Año 2006

4. "Datasheet DS1820", Autor Dallas Semiconductor Corporation, Revision 042208, Año 2008

5. http://www.maxim-ic.com/appnotes.cfm/appnote_number/1796/

6. http://es.wikipedia.org/wiki/1-Wire

7. http://es.wikipedia.org/wiki/Lego_Mindstorms

8. http://msdn.microsoft.com/es-es/library/47zceaw7(VS.80).aspx