



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

**DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL
SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE**

Luis Antonio Sierra García

Asesorado por el Ing. Guillermo Antonio Puente Romero

Guatemala, septiembre de 2017

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON
EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA

POR

LUIS ANTONIO SIERRA GARCÍA

ASESORADO POR EL ING. GUILLERMO ANTONIO PUENTE ROMERO

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN ELECTRÓNICA

GUATEMALA, SEPTIEMBRE DE 2017

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 22 de marzo de 2017.



Luis Antonio Sierra García

Guatemala, 09 de agosto de 2017.

Ing. Julio César Solares Peñate
Coordinador de Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Ingeniero Solares:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: "DISEÑO DE SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE", desarrollado por el estudiante Luis Antonio Sierra García carné No. 2011-14073, ya que considero que cumple con los requisitos establecidos, por lo que el autor y mi persona somos responsables del contenido y conclusiones del mismo.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,



Ing. Guillermo Antonio Puente Romero
ASESOR
Colegiado 5898

Guillermo A. Puente R.
INGENIERO ELECTRONICO
COL. # 5898



FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.

Ciudad Universitaria, zona 12
Guatemala, Centroamérica

Guatemala, 18 de agosto de 2017

Señor Director
Ing. Otto Fernando Andrino González
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **“DISEÑO DE SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE”**, desarrollado por el estudiante **Luis Antonio Sierra García**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS


Ing. Julio César Solares Peñate
Coordinador de Electrónica





REF. EIME 41 . 2017.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; LUIS ANTONIO SIERRA GARCÍA titulado: DISEÑO DE SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE, procede a la autorización del mismo.


Ing. Otto Fernando Andriño González



GUATEMALA, 4 DE SEPTIEMBRE 2,017.

Universidad de San Carlos
de Guatemala

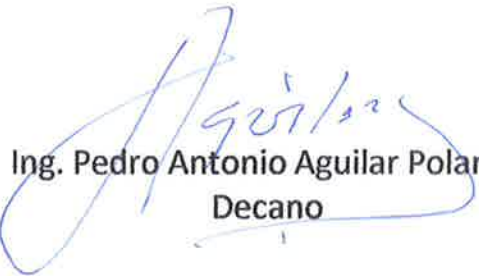


Facultad de Ingeniería
Decanato

DTG. 422.2017

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS CON EL SENSOR DS18B20 MEDIANTE EL PROTOCOLO DE TRANSMISIÓN 1-WIRE**, presentado por el estudiante universitario: **Luis Antonio Sierra García**, y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:


Ing. Pedro Antonio Aguilar Polanco
Decano

Guatemala, septiembre de 2017

/gdech



UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Jurgen Andoni Ramírez Ramírez
VOCAL V	Br. Oscar Humberto Galicia Nuñez
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADORA	Inga. Ingrid Salomé Rodríguez de Loukota
EXAMINADOR	Ing. José Aníbal Silva de los Ángeles
EXAMINADOR	Ing. Armando Alonso Rivera Carrillo
SECRETARIA	Inga. Lesbia Magalí Herrera López

ACTO QUE DEDICO A:

Dios	Por darme la vida, fuerza y sabiduría para cumplir esta meta.
Mis padres	Edgard Antonio y Mayra Rebeca, por su amor incondicional en el transcurrir de mi vida. Su ejemplo será siempre mi inspiración. Este triunfo también es de ellos.
Mis hermanos	Daniel Andrés y Diego Alejandro por todas las alegrías que compartimos.
Mi novia	Irene Urrutia, por siempre darme todo su apoyo y ánimo en las decisiones que tomo y por todo su amor.
Mi familia	Por siempre estar pendientes de mí y darme su apoyo.
Mis primos	Por todas las aventuras y momentos divertidos que hemos compartido.

Mis amigos

Por todos los momentos que compartimos y estuvimos juntos, definitivamente sin ustedes el paso por la universidad no hubiera sido la gran aventura que fue.

AGRADECIMIENTOS A:

Dios	Por su fidelidad y proveerme de perseverancia para terminar mi carrera.
Mis padres	Por apoyarme siempre y creer en mí.
Mis hermanos	Por alegrarme la vida.
Mi novia	Por creer en mí, animarme a que cumpla mis sueños y motivarme a ser mejor siempre.
Mis amigos	Por compartir triunfos y derrotas pero siempre con la cabeza en alto.
Guillermo Puente	Por el tiempo y ayuda brindada durante este proceso.
Glenda García	Por su apoyo brindado durante este proceso.
Facultad de Ingeniería	Por la formación e instrucción profesional brindada.
Universidad de San Carlos de Guatemala	Por ser una excelente <i>alma mater</i> .

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	V
LISTA DE SÍMBOLOS	IX
GLOSARIO	XI
RESUMEN	XIII
OBJETIVOS.....	XV
INTRODUCCIÓN	XVII
1. PROTOCOLO DE TRANSMISIÓN 1-WIRE	1
1.1. Características del protocolo 1-Wire.....	3
1.2. Comunicación del protocolo 1-Wire.....	4
1.2.1. Inicialización	6
1.2.2. Comandos ROM.....	7
1.2.2.1. Read ROM.....	8
1.2.2.2. Match ROM.....	8
1.2.2.3. Skip ROM	8
1.2.2.4. Search ROM [F0h].....	9
1.2.2.5. Alarm Search [ECh]	10
1.2.3. Comandos y funciones de control y memoria	11
1.2.4. Transferencia de datos	11
1.3. Energía sobre 1-Wire.....	12
1.4. Configuración de hardware.....	13
1.5. Topologías de conexión entre dispositivos	14

2.	DISPOSITIVOS ELECTRÓNICOS PARA EL DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS	17
2.1.	Sensor DS18B20.....	17
2.1.1.	Generalidades	17
2.1.1.1.	Aplicaciones	18
2.1.1.2.	Beneficios y características	18
2.1.2.	Resumen	20
2.1.3.	Operación.....	22
2.1.3.1.	Medición de temperatura.....	22
2.1.3.2.	Señales de alarma.....	24
2.1.4.	Alimentando el DS18B20	25
2.1.5.	Código ROM.....	29
2.1.6.	Memoria SPM.....	29
2.1.6.1.	Registro de configuración.....	31
2.1.7.	Generación de CRC	32
2.1.8.	Comandos de función del DS18B20	33
2.1.8.1.	Convert T [44h].....	33
2.1.8.2.	Write Scratchpad [4Eh].....	34
2.1.8.3.	Read Scratchpad [BEh].....	34
2.1.8.4.	Copy Scratchpad [48h].....	35
2.1.8.5.	Recall E ² [B8h]	35
2.1.8.6.	Read Power Supply [B4h]	35
2.2.	Arduino Uno	37
2.2.1.	Especificaciones técnicas.....	37
2.2.1.1.	Alimentación.....	38
2.2.1.2.	Memoria	39
2.2.1.3.	Entradas y salidas	40
2.2.1.4.	Comunicación.....	42
2.2.2.	Programación	42

3.	PROPUESTA DE DISEÑO DEL SISTEMA	43
3.1.	Sistema de control – Arduino IDE.....	44
3.2.	Interfaz gráfica – MATLAB R2012b	45
3.2.1.	Ventana principal	46
3.2.1.1.	Panel de conexión	47
3.2.1.2.	Panel de sensores	49
3.2.1.3.	Panel de datos.....	50
3.2.2.	Ventana de configuración de sensor	55
3.2.2.1.	Panel de sensores	56
3.2.2.2.	Panel de información	58
3.2.2.3.	Panel de alarma.....	60
3.2.2.4.	Panel de dispositivos conectados.....	61
3.2.3.	Ventana de lectura de sensores	62
	CONCLUSIONES	67
	RECOMENDACIONES	69
	BIBLIOGRAFÍA.....	71
	APÉNDICES	73

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Transmisión 1-Wire	1
2.	Red 1-Wire	2
3.	Estructura interna de un dispositivo 1-Wire	2
4.	Intervalos de tiempo o time slots	4
5.	Fases de comunicación del protocolo 1-Wire.....	5
6.	Estados del bus en las distintas fases de comunicación.....	6
7.	Fase de inicialización	7
8.	Lectura y escritura en el bus	12
9.	Circuito de alimentación del dispositivo esclavo.....	13
10.	Topologías de los dispositivos 1-Wire	15
11.	Empaquetados del sensor DS18B20	19
12.	Diagrama de bloques del sensor DS18B20	21
13.	Registro de temperatura.....	23
14.	Formato de los registros T_H y T_L	25
15.	Alimentación en parásita	27
16.	Alimentación con fuente externa	28
17.	Mapa de memoria del DS18B20	30
18.	Registro de configuración.....	31
19.	Generador de CRC	33
20.	Diagrama del Arduino Uno	38
21.	Mapeo de los puertos.....	40
22.	Iniciar comunicación serial - código.....	44
23.	Funciones del programa - código.....	45

24.	Ventana principal	46
25.	Panel de conexión	47
26.	Iniciar conexión serial - código.....	48
27.	Finalizar conexión serial - código.....	49
28.	Panel de sensores	49
29.	Panel de sensores - código.....	50
30.	Panel de datos.....	50
31.	Gráfico histórico de los sensores.....	51
32.	Gráfico histórico sensores individuales.....	52
33.	Gráficos históricos de los sensores - código.....	53
34.	Gráfico de temperaturas – tiempo real.....	54
35.	Gráfico temperaturas en tiempo real – código	55
36.	Ventana de configuración	56
37.	Panel de sensores	57
38.	Panel de sensores – código.....	57
39.	Panel de información	58
40.	Panel de información – código.....	59
41.	Panel de alarma.....	60
42.	Panel de alarma – código	61
43.	Panel de dispositivos conectados.....	62
44.	Panel de dispositivos conectados - código	62
45.	Ventana de lectura.....	63
46.	Ventana de lectura – código	64

TABLAS

I.	Comandos ROM.....	8
II.	Tabla Lookup del algoritmo Search ROM	10
III.	Descripción de los pines	21
IV.	Relación datos / temperatura	24
V.	Configuración de resolución.....	31
VI.	Comandos del sensor DS18B20	36

LISTA DE SÍMBOLOS

Símbolo	Significado
S	Bit de signo
C_{PP}	Capacitor de poder parásito
AC	Corriente alterna
DC	Corriente directa
°C	Grados Celsius
Kbps	Kilobit por segundo
KB	Kilobyte
KΩ	Kilohmio
MHz	Megahercio
m	Metro
μs	Microsegundo
mA	Miliamperio
ms	Milisegundo
T	Periodo
R_X	Pin de recepción
T_X	Pin de transmisión
T_H	Registro alto de temperatura
T_L	Registro bajo de temperatura
t_{CONV}	Tiempo de conversión
t_{WR}	Tiempo de lectura / escritura
GND	Tierra o masa
V_{IN}	Voltaje de entrada
V	Voltio

GLOSARIO

ADC	Conversión de análogo a digital.
Algoritmo	Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
Bit	Unidad mínima de información, que puede tener solamente dos valores (cero o uno).
Bus serial	Método de transmisión de un bit a la vez sobre una sola línea.
Byte	Unidad fundamental de datos en los ordenadores, son 8 bits contiguos y es la medida básica para memoria, equivalente a un carácter.
EEPROM	Memoria ROM que puede ser programada, borrada y reprogramada eléctricamente.
Half-duplex	Sistema capaz de mantener una comunicación bidireccional pero no simultáneamente.
LSB	Bit menos significativo, el cual de acuerdo a su posición, tiene el menor valor. Normalmente es el bit del extremo derecho.

Microprocesador	Circuito integrado central más complejo de un sistema informático, el cual lleva a cabo o ejecuta los programas.
MSB	Bit más significativo, el cual de acuerdo a su posición, tiene el mayor valor. Normalmente es el bit del extremo izquierdo.
Registro	Memoria de alta velocidad y poca capacidad, integrada en un microprocesador, que permite guardar transitoriamente y acceder a valores muy usados.
<i>Resistencia pull up</i>	Resistencia que se utiliza cuando se desea tener un valor lógico alto a la salida. Se conecta al voltaje de alimentación.
ROM	Memoria que solamente permite la lectura de la información.
SPM	Memoria interna de alta velocidad usada para almacenamiento temporal de cálculos, datos o algún otro trabajo que este en proceso.
SRAM	Memoria RAM basada en semiconductores.
Topología	Disposición geométrica real de la red, incluyendo sus nodos y líneas de conexión.

RESUMEN

En el presente trabajo de graduación se diseñó un sistema de medición y despliegue de temperatura con el sensor DS18B20 mediante el protocolo 1-Wire, desarrollando una interfaz gráfica para el usuario y un sistema de control para los sensores.

Inicialmente se presentó el protocolo 1-Wire con sus características y forma de comunicación, comandos y posibles configuraciones. Luego se buscaron dispositivos de control que fueran compatibles con el protocolo 1-Wire y pudiera realizarse la implementación del sistema de control; así mismo se buscó un sensor que fuera capaz de establecer una comunicación 1-Wire con el dispositivo de control que a su vez tuviera la característica de tener un rango de temperatura amplio.

Finalmente se documentó el desarrollo del sistema de temperaturas, con la adquisición de los datos de temperaturas y el despliegue de las mismas.

OBJETIVOS

General

Diseñar un sistema de medición y despliegue de temperaturas mediante el protocolo de comunicación 1-Wire.

Específicos

1. Presentar las características y forma de comunicación del protocolo 1-Wire.
2. Presentar las características de los dispositivos a utilizar en el diseño del sistema.
3. Presentar la propuesta del diseño del sistema de medición y despliegue de temperaturas.

INTRODUCCIÓN

La temperatura es una magnitud que mide el nivel térmico o el calor que un cuerpo posee y es una de las variables más usadas en los varios sectores de la industria de control de procesos. Con la búsqueda de nuevos desarrollos en el área de la energía renovable, de los nuevos combustibles y la nanotecnología existen incontables aplicaciones con la medición y control de temperatura.

La medición de temperatura es de suma importancia para cualquier situación y proceso ya sea en situaciones cotidianas como en la industria. Por ejemplo algunas situaciones cotidianas como una variación en la temperatura corporal provoca una sensación de malestar; la cocción de alimentos a una temperatura superior a la necesaria puede ocasionar que estos se quemen, entre otros.

En la industria se pueden encontrar más ejemplos donde el control preciso de la temperatura es imprescindible como las fundiciones de materiales, procesos de destilación, gestión de temperatura de confort en espacios climatizados, entre otros. En todos los procesos industriales el correcto control y gestión de la temperatura se vuelve fundamental para garantizar la calidad, seguridad e incluso el ahorro de dinero.

Los sistemas de control de temperatura son los encargados de mantener la temperatura en el margen establecido, mejorar el confort del usuario e incluso hasta incurrir en el ahorro energético, dependiendo del campo de aplicación. Un sistema básico de control de temperaturas tiene una entrada procedente de un

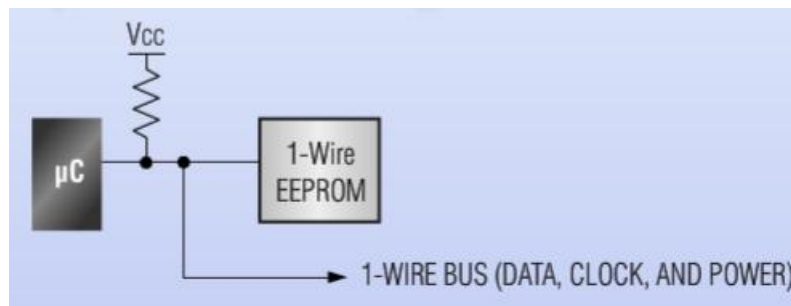
sensor de temperatura, un elemento central de control, el cual realiza todo el procesamiento del sistema y tiene una salida que está conectada a un elemento de control.

Se presenta un sistema básico de control de temperaturas para uso en distintas aplicaciones y áreas debido a la versatilidad de los sensores y el uso del protocolo 1-Wire®.

1. PROTOCOLO DE TRANSMISIÓN 1-WIRE

1-Wire[®] es un sistema de bus serial de comunicación de dispositivos bidireccional y Half-Duplex con capacidad Multidrop diseñado por Dallas Semiconductor Corp., ahora propiedad de Maxim Integrated[™], que proporciona datos a baja velocidad, señalización y potencia sobre un solo conductor más su conexión a tierra.

Figura 1. Transmisión 1-Wire

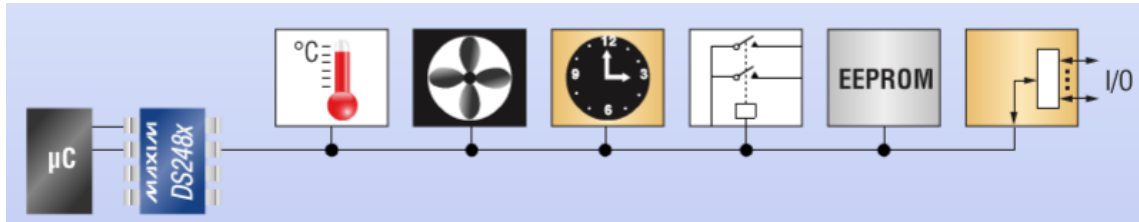


Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

Un bus serie es un vínculo para transportar datos de un bit a la vez entre un microcontrolador (*master*) y una serie de dispositivos de control (*slaves*), figura 2.

Figura 2. Red 1-Wire

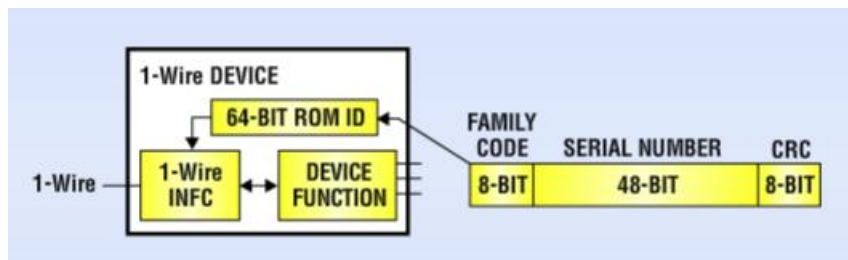


Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

El protocolo 1-Wire incluye un sistema de direccionamiento, cada dispositivo de la red debe tener un ID inalterable de 64 bits codificado en el mismo, el cual sirve como su dirección. El ID es único e incluye un código de la familia que indica el tipo de dispositivo y la funcionalidad. Los tipos de dispositivos atendidos por el 1-Wire bus incluyen sensores, en particular para los termómetros y estaciones meteorológicas y sistemas de llave electrónica, integrado en una unidad llamada iButton.

Figura 3. Estructura interna de un dispositivo 1-Wire



Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

1.1. Características del protocolo 1-Wire

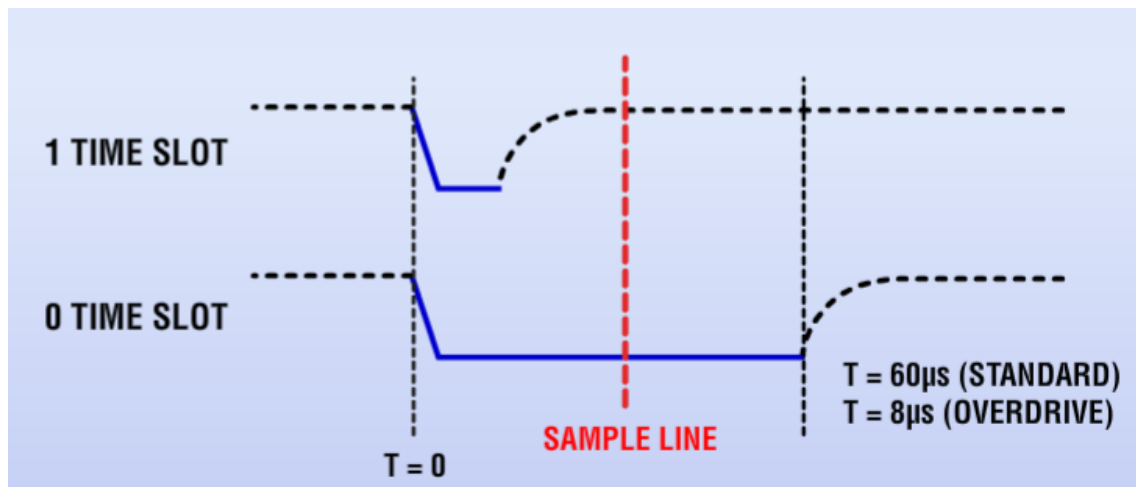
El bus 1-Wire permite realizar una comunicación serial asíncrona entre un dispositivo maestro y uno o varios dispositivos esclavos, utilizando un único pin de E/S del microcontrolador. Entre las características del bus se tiene:

- Utiliza niveles de alimentación CMOS/TTL con un rango de operación que abarca desde 2,8 V hasta 6 V.
- Ambos dispositivos, tanto maestro como esclavo, transmiten información de forma bidireccional, pero solo en una dirección a la vez, es decir *half-duplex*.
- Toda la información es leída o escrita comenzando por el bit menos significativo (LSB).
- No se requiere del uso de una señal de reloj, ya que, cada dispositivo 1-Wire posee un oscilador interno que se sincroniza con el del maestro cada vez que en la línea de datos aparezca un flanco de bajada.
- La alimentación de los esclavos se puede hacer utilizando el voltaje propio del bus.
- Las redes de dispositivos 1-Wire pueden tener fácilmente una longitud de 200 m y contener unos 100 dispositivos.
- Todas las tensiones mayores que 2,2 V son consideradas un 1 lógico mientras que como un 0 lógico se interpreta cualquier voltaje menor o igual a 0,8 V.
- La transferencia de información puede realizarse de dos modos: hasta 15,4 kbps en *standard* y hasta 125 kbps en *overdrive*.
- El dispositivo maestro inicia y controla la comunicación en la red 1-Wire.

1.2. Comunicación del protocolo 1-Wire

Se puede describir la comunicación del protocolo 1-Wire usando el concepto de intervalos de tiempo (*time slots*), un intervalo de tiempo es definido como un intervalo en el cual un 0 lógico o un 1 lógico, puede ser leído o escrito sobre el bus 1-Wire. El maestro inicia un intervalo de tiempo generando pulsos bajos controlados de corta duración para codificar los 1s y 0s. El 1 lógico es transmitido como un pulso bajo de corta duración mientras que el 0 lógico es transmitido como un pulso bajo de larga duración. El dispositivo maestro o esclavo muestra la línea durante una ventana específica de muestreo para sensor los niveles lógicos durante la secuencia de lectura o escritura. El intervalo de tiempo (T) tiene una duración de 60 μs para el modo *standard* y 8 μs para el modo *overdrive*.

Figura 4. Intervalos de tiempo o *time slots*

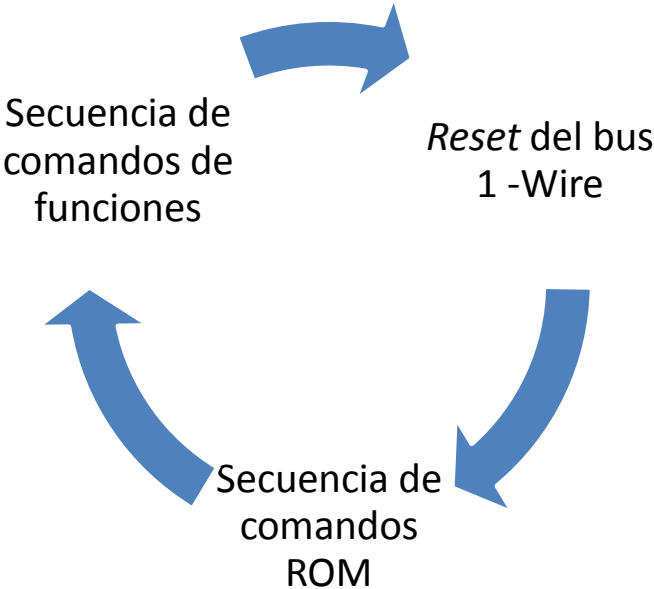


Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

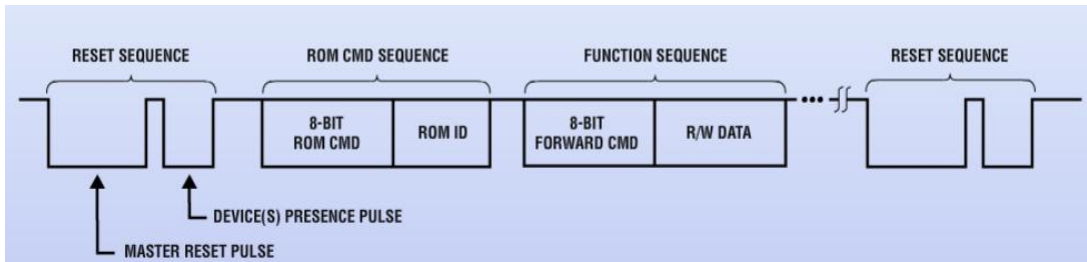
Se puede describir el protocolo 1-Wire como una secuencia de transacciones de información, la cual, se desarrolla según los siguientes fases: inicialización, comandos y funciones de ROM, comandos y funciones de control y memoria, transferencia de bytes o datos.

Figura 5. **Fases de comunicación del protocolo 1-Wire**



Fuente: elaboración propia.

Figura 6. **Estados del bus en las distintas fases de comunicación**

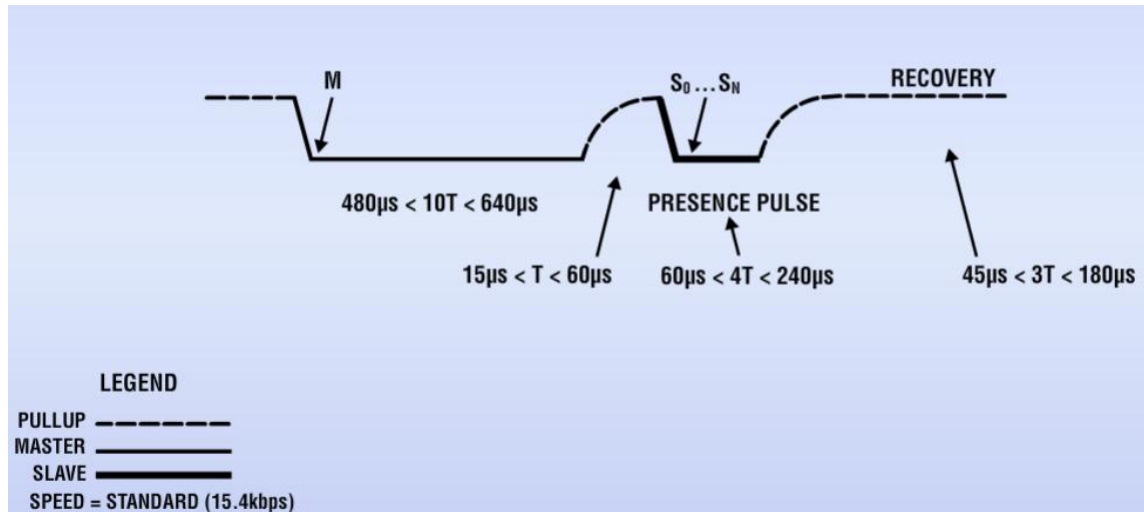


Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.
Consulta: febrero 2017.

1.2.1. Inicialización

Todas las comunicaciones en el bus 1-Wire comienzan con una secuencia de un pulso de *reset* y presencia. El pulso de *reset* provee una forma limpia de iniciar las comunicaciones, ya que, con él se sincronizan todos los dispositivos esclavos presentes en el bus. Un *reset* es un pulso que genera el maestro al colocar la línea de datos en estado lógico bajo de 480 μ s a 640 μ s, y luego de 15 μ s a 60 μ s los esclavos responderán, con un pulso de presencia, poniendo en bajo la línea durante 45 μ s a 180 μ s.

Figura 7. Fase de inicialización



Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

En un sistema con múltiples esclavos, todos bajarán la línea simultáneamente por lo que el maestro solamente podrá saber que existe más de un esclavo presente en el bus por el pulso de presencia. Para resolver el número de esclavos requiere una secuencia de detección de dispositivos.

1.2.2. Comandos ROM

Una vez que el dispositivo maestro recibe el pulso de presencia de los dispositivos esclavos, se puede enviar un comando de ROM. Los comandos de ROM, son comunes a todos los dispositivos 1-Wire y operan con el código ROM único de 64 bits de cada dispositivo esclavo y permite al maestro seleccionar un dispositivo específico si existen varios dispositivos en el bus 1-Wire. Estos comandos también permiten al maestro determinar cuántos y qué tipos de dispositivos están presentes en el bus o si algún dispositivo ha experimentado

una condición de alarma. Es decir, se relacionan con la búsqueda, lectura y utilización de la dirección de 64 bits que identifica a esclavos. La tabla 1, muestra los cinco comandos ROM utilizados con los dispositivos 1-Wire.

Tabla I. **Comandos ROM**

Comando	Código
Read ROM	33h
Match ROM	55h
Skip ROM	CCh
Search ROM	F0h
Alarm Search	ECh

Fuente: elaboración propia.

1.2.2.1. Read ROM

Permite al maestro leer el código de 8 bits de la familia, los 48 bits de número de serie y los 8 bits CRC, es decir, lee la identificación de 64 bits del dispositivo esclavo. Este comando solo funciona si existe un solo dispositivo, ya que de lo contrario ocurrirá una colisión de datos cuando todos los esclavos transmitan al mismo tiempo.

1.2.2.2. Match ROM

Seguido de la identificación de 64 bits, este comando permite al maestro direccionar a un dispositivo en específico cuando existe más de un esclavo. El dispositivo que coincida con la identificación espera por la instrucción siguiente, mientras que el resto de los esclavos esperan por el pulso de *reset*.

1.2.2.3. Skip ROM

Permite direccionar de forma directa, sin la necesidad de enviar la identificación. Este comando solo es utilizable cuando existe un solo esclavo, de lo contrario ocurrirá una colisión de datos cuando todos respondan.

1.2.2.4. Search ROM [F0h]

A través de este comando se puede leer los 64 bits de identificación de todos los dispositivos esclavos conectados. Se utiliza un algoritmo de eliminación de 3 pasos para distinguir cada dispositivo conectado: leer bit, leer bit complemento y escribir bit de dirección deseado. El algoritmo de Search ROM será explicado a continuación.

El maestro lee el primer bit LSB (1ra lectura) de todos los dispositivos conectados en el bus, luego el maestro lee el bit complemento (2da lectura) de todos los dispositivos en el bus para poder realizar una comparación entre la primera y segunda lectura. Basado en esa comparación el maestro deduce cual es el estado del valor del bit del dispositivo con base en la siguiente tabla.

Tabla II. **Tabla *lookup* del algoritmo Search ROM**

Lectura 1 = Bit (LSB)	Lectura 2 = Bit (complemento)	Información conocida
0	0	Condición 1: algunos dispositivos en el bus tienen 1 lógico, otros tienen 0 lógico en la posición del bit que se está leyendo.
0	1	Condición 2: todos los dispositivos en el bus tienen 0 lógico en la posición del bit que se está leyendo.
1	0	Condición 3: todos los dispositivos en el bus tienen 1 lógico en la posición del bit que se está leyendo.
1	1	Condición 4: no hay dispositivos presentes.

Fuente: elaboración propia.

El maestro escribe un bit de dirección en el bus y todos los dispositivos que tengan el mismo bit, que el maestro está escribiendo, en su LSB se quedan en el bus participando mientras los otros dispositivos se salen del bus hacia un estado de espera. El maestro repite 63 veces más el algoritmo para identificar el código ROM completo de un dispositivo esclavo en el bus. Luego de cada ciclo de Search ROM, el bus maestro debe regresar a la inicialización.

1.2.2.5. Alarm Search [ECh]

La operación de este comando es idéntica a la operación del comando Search ROM con la excepción que solamente los esclavos que tengan una bandera de alarma arriba responderán.

1.2.3. Comandos y funciones de control y memoria

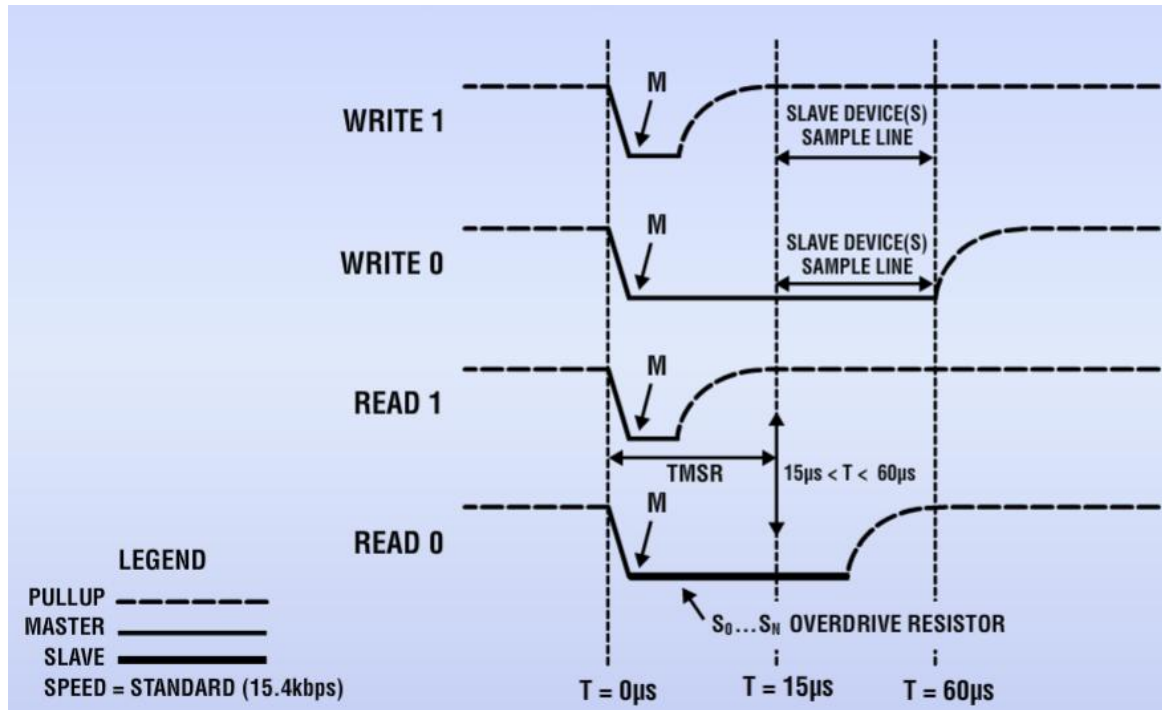
Son funciones propias del dispositivo 1-Wire. Incluyen comando para leer/escribir en localidades de memoria, leer memorias SPM, controlar el inicio de la conversión de un ADC, iniciar la medición de una temperatura o manipular el estado de un bit de salida, entre otros. Cada dispositivo define su propio conjunto de comandos. Los comandos del sensor a utilizar se verán en el capítulo 2.

1.2.4. Transferencia de datos

La lectura y escritura de datos en el bus 1-Wire se hace por medio de ranuras, la generación de estos es responsabilidad del maestro. Cuando el maestro lee información del bus, debe forzar la línea de datos a un estado bajo durante al menos $1\mu\text{s}$ y esperar unos $15\mu\text{s}$ para entonces leer el estado de la misma. El estado lógico de la línea en ese momento, está determinado por el dispositivo esclavo.

Al momento de efectuar la escritura del bit en el bus ocurre algo similar, el maestro produce un pulso entre $1\mu\text{s}$ y $15\mu\text{s}$ de duración, para luego colocar en el bus al bit que se desea transmitir. Este bit deberá permanecer en el bus al menos $60\mu\text{s}$.

Figura 8. Lectura y escritura en el bus



Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

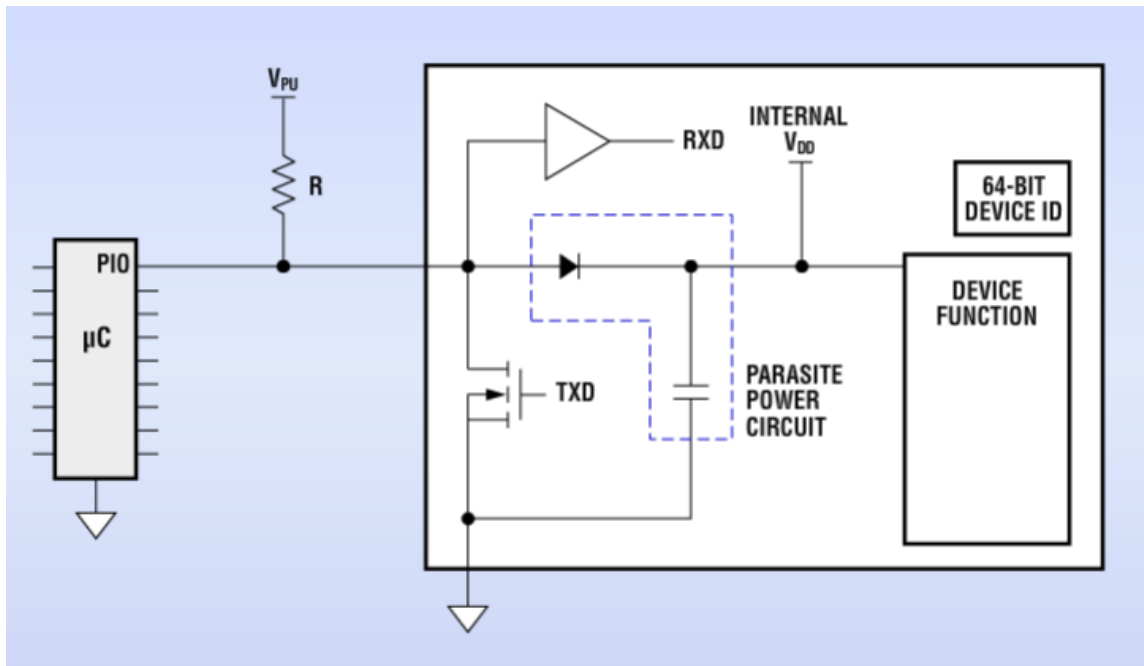
Consulta: febrero 2017.

1.3. Energía sobre 1-Wire

Se puede alimentar los dispositivos esclavos mediante el bus 1-Wire en un modo llamado alimentación parásita. Para ello, es necesario que cada circuito esclavo tenga circuito rectificador de media onda y un condensador, los cuales proveen el circuito de alimentación parásita. Durante los períodos en los cuales no se realiza ninguna comunicación, el bus se encuentra en estado alto debido a la resistencia de *pullup*, en esa condición, el diodo entra en conducción y carga el condensador. Cuando el voltaje cae por debajo del voltaje del condensador, el diodo se polariza en inverso evitando que el condensador se

descargue. La carga que almacena el condensador provee energía para alimentar el circuito esclavo hasta que el bus vuelva a su estado alto.

Figura 9. **Circuito de alimentación del dispositivo esclavo**



Fuente: <https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>.

Consulta: febrero 2017.

1.4. Configuración de hardware

El bus 1-Wire tiene por definición solamente una sola línea de datos. Cada dispositivo (maestro o esclavo) se conecta a la línea de datos a través de un drenaje abierto (*open drain*) o puerto de 3 estados. Esto permite a cada dispositivo liberar la línea de datos cuando el dispositivo no está transmitiendo datos por lo que el bus está disponible para el uso de otro dispositivo.

El bus 1-Wire requiere una resistencia externa de *pullup* de aproximadamente 5 k Ω , por lo tanto, el estado inactivo para el bus 1-Wire es alto. Si por alguna razón una transacción necesita ser suspendida, el bus debe dejarse en su estado inactivo si la transacción se reanuda. El tiempo infinito de recuperación puede ocurrir entre los bits siempre y cuando el bus 1-Wire se encuentre en un estado inactivo (alto) durante el periodo de recuperación. Si el bus es mantenido bajo durante más de 480 μ s, todos los componentes en el bus se van a resetear.

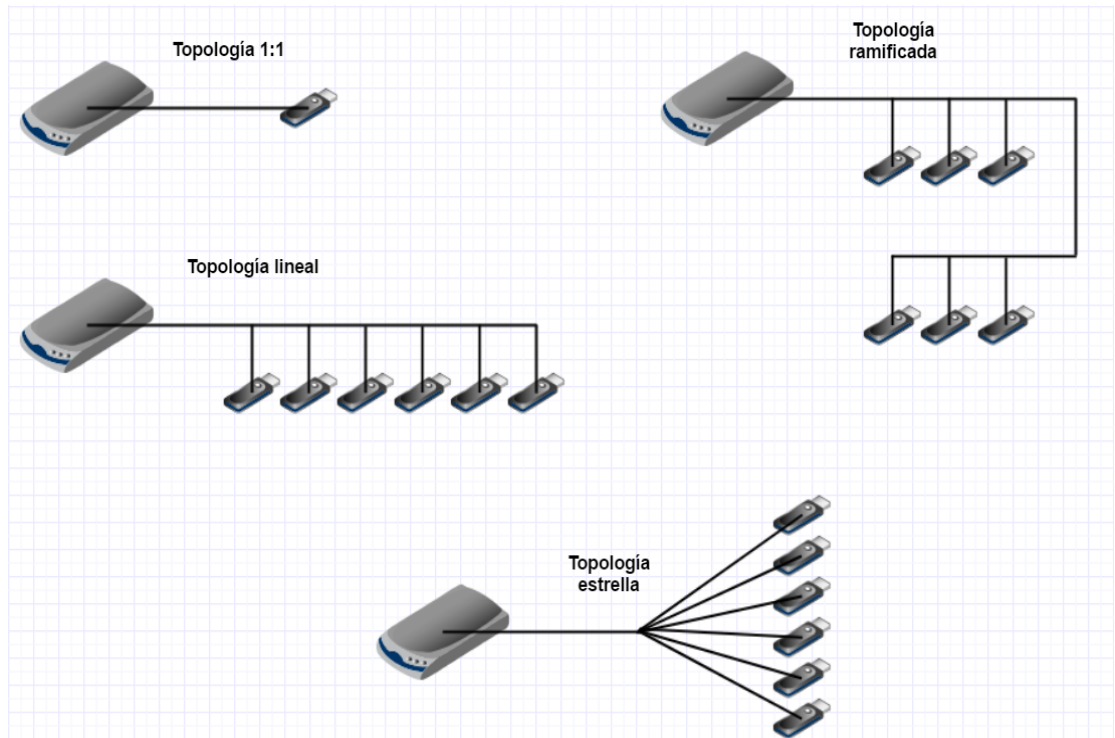
1.5. Topologías de conexión entre dispositivos

La figura 10 muestra las diferentes topologías de conexión entre dispositivos en una red 1-Wire. La topología exclusiva también denominada 1:1, es la más simple de todas, se permite en este tipo de topología la conexión de un dispositivo maestro con un dispositivo esclavo. Es muy utilizada para la medición de parámetros en dispositivos esclavos tipo *stand alone* como por ejemplo: los dispositivos iButton de Dallas Semiconductor.

Las topologías lineal y ramificada extienden el alcance de la red 1-Wire hasta una distancia aproximada de 200 metros. En ellas, los dispositivos esclavos pueden interconectarse de forma secuencial o a través de ramificaciones.

Por último, la topología tipo estrella, la cual, en la práctica es la más utilizada, permite la conexión de ramas a través de un punto común denominado nodo de conexión, sin embargo, esta topología limita la cantidad de dispositivos esclavos en comparación con las anteriores, ya que, incrementa la capacidad equivalente en el punto central de conexión al estar las ramas conectadas en paralelo.

Figura 10. **Topologías de los dispositivos 1-Wire**



Fuente: elaboración propia.

2. DISPOSITIVOS ELECTRÓNICOS PARA EL DISEÑO DEL SISTEMA DE MEDICIÓN Y DESPLIEGUE DE TEMPERATURAS

A continuación se presentan los dispositivos más importantes para el diseño del sistema de medición y despliegue de temperaturas.

2.1. Sensor DS18B20

Los sensores de temperatura son dispositivos que transforman los cambios de temperatura en cambios en señales eléctricas que son procesadas por equipo electrónico. Típicamente suele estar formado por el elemento sensor, la vaina que lo envuelve y que está rellena de un material conductor de la temperatura, para que los cambios se transmitan rápidamente al elemento sensor y del cable al que se le conectará el equipo electrónico.

2.1.1. Generalidades

El DS18B20 es un termómetro digital que provee medidas de temperatura Celsius desde 9-bits a 12-bits y posee una función de alarma con puntos de umbral superior e inferior no volátiles programados por el usuario. Se comunica sobre un bus 1-Wire que por definición requiere solo una línea de datos (y tierra) para comunicarse con el microprocesador central. En adicción, el DS18B20 puede obtener el poder directamente de la línea de datos (poder parásito), eliminando la necesidad de una fuente de poder externa.

Cada DS18B20 tiene un código serial único de 64-bits, que permite el funcionamiento en el mismo bus 1-Wire de múltiples sensores DS18B20. Por lo

tanto, es simple de usar un solo microprocesador que controle varios DS18B20 distribuidos en un área amplia.

2.1.1.1. Aplicaciones

El sensor DS18B20 presenta una gran versatilidad por lo que posee un gran número de aplicaciones, entre las cuales destacan:

- Controles termostáticos
- Sistemas industriales
- Termómetros
- Sistemas térmicos sensibles

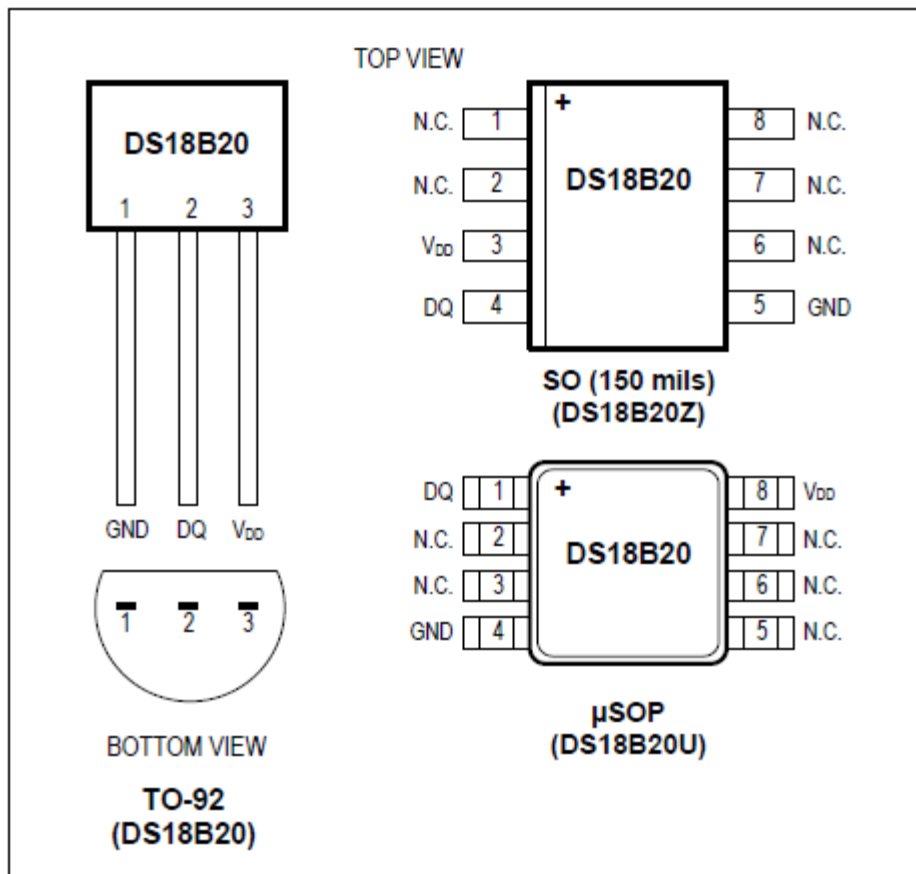
2.1.1.2. Beneficios y características

Algunas de las características más importantes del sensor DS18B20, que lo convierten en un equipo versátil, se presentan a continuación:

- Interfaz 1-Wire® única, solamente requiere un puerto pin para comunicación.
- Sensor de temperatura integrado y EEPROM.
 - Mide temperaturas desde -55 °C hasta 125 °C.
 - $\pm 0,5$ °C de exactitud desde -10 °C hasta +85 °C.
 - Resolución de 9 bits a 12 bits programable.
 - No requiere componentes externos.
- Modo de poder parásito, requiere solamente 2 pines para operar (DQ y GND).
- Distribución que simplifica la toma la temperatura.

- Aplicaciones con capacidad *multidrop*.
 - Cada dispositivo tiene un código serial único de 64 bits guardado en la ROM.
- Disponible en 8-Pin SO, 8-Pin μ SOP y 3-Pin empaquetado TO-92

Figura 11. **Empaquetados del sensor DS18B20**



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.2. Resumen

La figura 12 muestra el diagrama de bloques del DS18B20 junto con la descripción de sus pines en la tabla III.

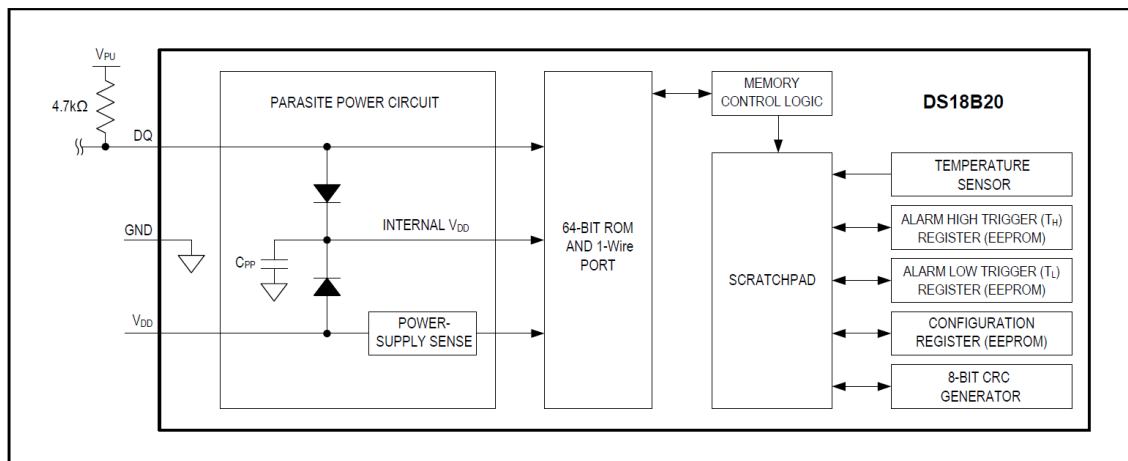
La memoria ROM de 64 bits almacena el código único serial del dispositivo. La memoria SPM contiene el registro de temperatura de 2 bytes que almacena la salida digital del sensor de temperatura. Adicional, la SPM provee acceso a los registros de alarma de umbral alto y bajo (T_H y T_L) de 1 byte y al registro de configuración de 1 byte. El registro de configuración permite al usuario configurar una resolución para la conversión de temperatura a digital a 9, 10, 11 o 12 bits. Los registros T_H , T_L y el de configuración son no volátiles (EEPROM), por lo que van a retener sus datos una vez el dispositivo sea apagado.

El DS18B20 usa el protocolo bus 1-Wire exclusivo de Maxim, que implementa la comunicación del bus usando solamente una señal de control. La línea de control requiere un resistor *pullup* débil, ya que todos los dispositivos están conectados al bus vía 3 estados o puerto drenaje abierto (*open-drain port*). En este sistema de bus, el microprocesador identifica y direcciona los dispositivos presentes en el bus usando el código único de 64-bits de cada dispositivo. Debido a que cada dispositivo tiene un código único, el número de dispositivos que pueden direccionarse en un bus es virtualmente ilimitado.

Otra característica del DS18B20 es la habilidad de operar sin necesidad de una fuente de poder externa. La energía es suministrada a través del resistor *pullup* del 1-Wire a través del pin DQ cuando el bus está en estado alto. La señal en estado alto del bus también carga el capacitor interno (C_{pp}), que suministra energía al dispositivo cuando el estado del bus es bajo. Este método

de entregar energía del bus 1-Wire es referido como alimentación parásita (*parasite power*). Como alternativa, el DS18B20 puede ser también energizado con una fuente externa en V_{DD} .

Figura 12. Diagrama de bloques del sensor DS18B20



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.

<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

Tabla III. Descripción de los pines

PIN	Nombre	Función
1	GND	Tierra
2	DQ	Entrada / salida de datos. Pin de la interfaz 1-Wire <i>open-drain</i> . Así mismo suministra la energía al dispositivo cuando está operando en modo alimentación parásita.
3	V_{DD}	V_{DD} opcional. V_{DD} debe de conectarse a tierra cuando está operando en el modo alimentación parásita.

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.

<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.3. Operación

Dentro de las operaciones del sensor la más importante es la medición de temperatura, sin embargo, se tiene la opción de configurar temperaturas de alarma mediante valores umbrales. Ambas operaciones se detallan a continuación.

2.1.3.1. Medición de temperatura

La funcionalidad principal del DS18B20 es su sensor de temperatura digital directo. La resolución del sensor de temperatura es configurable al usuario a 9, 10, 11 o 12 bits, correspondientes a incrementos de 0,5 °C, 0,25 °C, 0,125 °C y 0,0625 °C, respectivamente. La resolución por defecto al encenderlo es de 12 bits. El DS18B20 se enciende en un estado inactivo de baja potencia (*low-power IDLE state*). Para iniciar una medición de temperatura y la conversión analógica a digital (ADC), el maestro debe emitir un comando Convert T [44h]. Seguidamente de la conversión, los datos de temperatura resultantes son almacenados en el registro de temperatura de 2 bytes en la memoria SPM y el DS18B20 regresa a su estado inactivo (*IDLE state*). Si el DS18B20 está alimentada por una fuente externa, el maestro puede emitir la lectura por intervalos de tiempo) después del comando Convert T y el DS18B20 responderá transmitiendo un 0 mientras la conversión de temperatura está en proceso y un 1 cuando la conversión está terminada. Si el DS18B20 está alimentado en el modo parásito, la técnica de notificación mencionada anteriormente no puede ser usada ya que el bus debe de ser llevado a un estado alto mediante un *pullup* fuerte durante la conversión completa de temperatura. Los requerimientos del bus para el modo parásito están explicados a detalle en la sección 2.1.4.

La salida de temperatura del DS18B20 está calibrada en grados centígrados, para aplicaciones de grados Fahrenheit, una rutina de conversión debe ser usada. Los datos de la temperatura son almacenados como dos números complementos de 16 bits tipo *sign-extended* en el registro de temperatura (figura 13). Los bits de signo (S) indican si la temperatura es positiva o negativa: para números positivos $S = 0$ y para números negativos $S = 1$. Si el DS18B20 es configurado para una resolución de 12 bits, todos los bits en el registro de temperatura contendrán datos válidos. Para una resolución de 11 bits, el bit 0 es indefinido. Para una resolución de 10 bits, el bit 1 y el 0 son indefinidos y para una resolución de 9 bits, los bits 2, 1 y 0 son indefinidos. La tabla IV da un ejemplo de la salida de datos digital y la lectura de temperatura correspondiente para conversiones con resolución de 12 bits.

Figura 13. **Registro de temperatura**

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

Tabla IV. **Relación datos / temperatura**

Temperatura (°C)	Salida digital (binario)	Salida digital (HEX)
+125	0000 0111 1101 0000	07D0h
+85	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	Fc90h

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.3.2. Señales de alarma

Luego que el DS18B20 efectúa una conversión de temperatura, el valor de la temperatura es comparado con los valores lumbrales de alarma que el usuario definió, almacenados en los registros de 1 byte T_H y T_L (figura 13). El bit de signo (S) indica si el valor es positivo o negativo: para números positivos $S = 0$ y para número negativos $S = 1$. Los registros T_H y T_L son no volátiles (EEPROM) por lo que van a retener los datos cuando el dispositivo sea apagado. Los registros T_H y T_L pueden ser accedidos a través de los bytes 2 y 3 de la memoria SPM.

Solamente los bits 11 al 4 del registro de temperatura son usados en la comparación de los registros T_H y T_L , ya que registros T_H y T_L son registros de 8 bits. Si la temperatura medida es menor o igual a T_L o mayor o igual a T_H , existe una condición de alarma y una bandera de alarma es activada dentro del DS18B20. La bandera es actualizada luego de cada medición de temperatura,

por lo tanto, si la condición de alarma se quita, la bandera será desactivada luego de la siguiente medición de temperatura.

El dispositivo maestro puede revisar el estado de la bandera de alarma de todos los DS18B20s que están en el bus enviando el comando Alarm Search [ECh]. Cualquier DS18B20 con la bandera de alarma activada va a responder al comando, por lo que el maestro podrá determinar exactamente cual DS18B20 experimentó la condición de alarma. Si la condición de alarma existe y los registros T_H y T_L han sido cambiados, otra conversión de temperatura debe realizarse para validar la condición de alarma.

Figura 14. **Formato de los registros T_H y T_L**

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
S	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

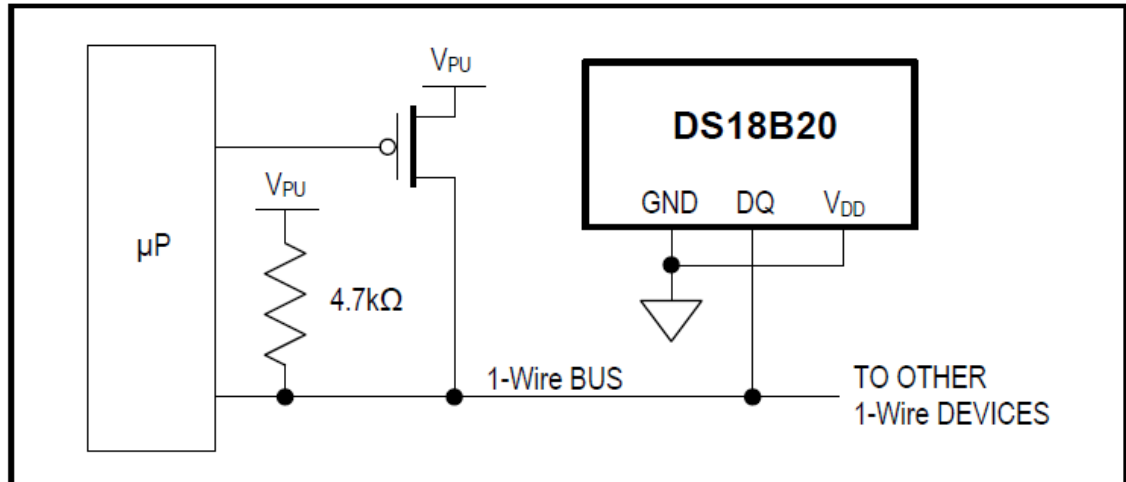
2.1.4. Alimentando el DS18B20

El DS18B20 puede ser alimentado por una fuente externa en el pin V_{DD} , o puede operar en el modo parásito, lo que permite al DS18B20 funcionar sin una fuente local externa. El modo parásito es muy útil para aplicaciones que requieran la medición remota de temperaturas o que poseen un espacio reducido. En la figura 15 se muestra el circuito de control del modo parásito del DS18B20, el cual obtiene la energía del bus 1-Wire a través del pin DQ cuando el bus está en estado alto. La carga obtenida alimenta el DS18B20 mientras el bus está en alto y parte de la carga es almacenada en el capacitor parásito

(C_{PP}) para proveer energía cuando el bus está en estado bajo. Cuando el DS18B20 es usado en el modo parásito, el pin V_{DD} debe conectarse a tierra.

En el modo de parásito, el bus 1-Wire y el C_{PP} pueden proporcionar suficiente corriente al DS18B20 para la mayoría de operaciones siempre y cuando los requerimientos de tiempo y voltaje sean cumplidos. Sin embargo, cuando el DS18B20 está ejecutando una conversión de temperatura o copiando datos desde la SPM a la EEPROM, la corriente operativa puede ser tan alta como 1,5 mA. Esta corriente puede causar una caída inaceptable de voltaje a través de la débil resistencia *pullup* del 1-Wire y es más corriente de lo que puede proporcionar C_{PP} . Para asegurar que el DS18B20 tenga suficiente suministro de corriente, es necesario proveer un *pullup* más fuerte en el bus 1-Wire siempre que las conversiones de temperatura están efectuándose o los datos están siendo copiados desde la SPM a la EEPROM. Esto puede lograrse utilizando un MOSFET para jalar el bus directamente al riel como lo muestra la figura 15. El bus 1-Wire debe de ser cambiado a un *pullup* más fuerte dentro de 10 μ s (máximo) luego de que se emita el comando Convert T [44h] o Copy Scratchpad [48h] y el bus debe mantenerse en alto por el *pullup* durante la duración de la conversión (t_{CONV}) o transferencia de datos ($t_{WR} = 10$ ms). Ninguna otra actividad puede tomar lugar en el bus 1-Wire mientras el *pullup* este habilitado.

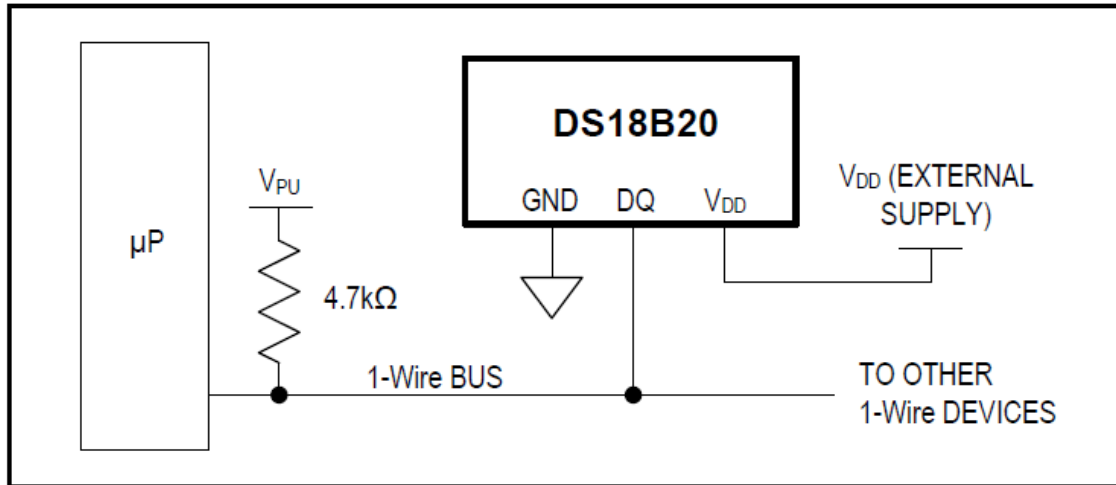
Figura 15. Alimentación en parásita



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

El DS18B20 también puede ser alimentado por el método convencional de conectarlo a una fuente de alimentación externa al pin V_{DD}, como se muestra en la figura 16. La ventaja de este método es que el uso del MOSFET como *pullup* no es requerido y el bus 1-Wire es libre de transportar otro tráfico durante el tiempo de conversión de temperatura.

Figura 16. Alimentación con fuente externa



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

El uso de la alimentación parásita no es recomendable para temperaturas por encima de los +100 °C ya que el DS18B20 puede que no sea capaz de sostener las comunicaciones debido a las grandes corrientes de fuga que pueden existir a esas temperaturas. Para aplicaciones en las que esas temperaturas son probables, es fuertemente recomendado que el DS18B20 sea alimentado por una fuente externa.

En algunas situaciones el bus maestro puede no saber si los DS18B20 en el bus son alimentados de forma parásita o alimentados por fuentes externas. El maestro necesita esa información para determinar si el uso del *pullup* fuerte en el bus será usado durante la conversión de temperatura. Para obtener esa información el maestro puede enviar el comando Skip ROM [CCh], seguido del comando Read Power Supply [B4h], seguido por una lectura por intervalos de tiempo. Durante la lectura por intervalos de tiempo, la alimentación parásita

pondrá el bus en estado bajo y la alimentación externa dejará el bus en estado alto. Si el bus es llevado a un estado bajo, el maestro sabe que se debe suministrar un *pullup* fuerte en el bus 1-Wire durante la conversión de temperatura.

2.1.5. Código ROM

Cada DS18B20 contiene un código de 64 bits único almacenado en la ROM. Los 8 bits menos significativos (LSB) del código de la ROM contienen el código 1-Wire de la familia de los sensores DS18B20: 28h. Los siguientes 48 bits contienen un número serial único. Los 8 bits más significativos (MSB) contienen un chequeo de redundancia (CRC) que es calculado de los primeros 56 bits del código ROM. El código ROM de 64 bits y una función ROM asociada al control lógico permiten que el DS18B20 opere como un dispositivo 1-Wire.

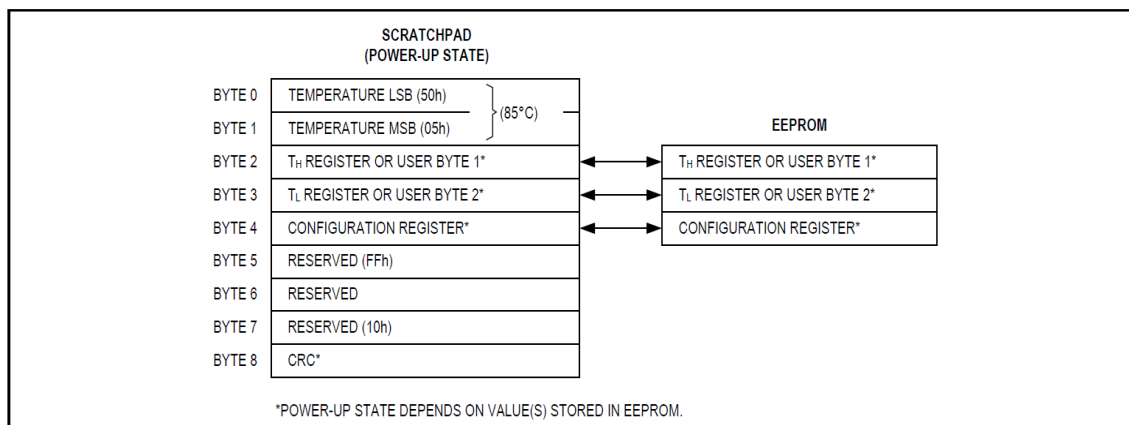
2.1.6. Memoria SPM

La memoria del DS18B20 está organizada como se muestra en la figura 17. La memoria consiste de una SRAM *scratchpad* con almacenamiento no volátil EEPROM para los registros de umbral de alarma alto y bajo (T_H y T_L) y registro de configuración. Nótese que si la función de alarma del DS18B20 no se utiliza, los registros T_H y T_L pueden servir como memoria de propósito general. El byte 0 y el byte 1 de la SPM contienen el LSB y el MSB de los registros de temperatura, respectivamente. Estos bytes son solamente de lectura. El byte 2 y el 3 proveen acceso a los registros T_H y T_L . El byte 4 contiene el registro de configuración. Los bytes 5, 6 y 7 están reservados para uso interno del dispositivo y no pueden ser sobrescritos. El byte 8 es solo lectura y contiene el código CRC para los bytes del 0 al 7 de la SPM. El DS18B20 genera ese CRC usando el método de generación de CRC.

Los datos de los bytes 2, 3 y 4 son escritos en la memoria usando el comando Write Scratchpad [4Eh], los datos deben ser transmitidos al DS18B20 empezando con el LSB del byte 2. Para verificar la integridad de los datos, la memoria puede ser leída utilizando el comando Read Scratchpad [BEh], luego que los datos son escritos. Cuando se lee la memoria, los datos son transferidos en el bus 1-Wire comenzando con el LSB del byte 0. Para transferir los datos de los registros T_H , T_L y configuración de la memoria a la EEPROM, el maestro debe enviar el comando Copy Scratchpad [48h].

Los datos en los registros de la EEPROM son retenidos cuando el dispositivo es apagado, al momento de encender el dispositivo, los datos de la EEPROM son cargados nuevamente a las ubicaciones correspondientes dentro de la SPM. Los datos pueden ser reenviados de la EEPROM a la SPM en cualquier momento usando el comando Recall E² [B8h].

Figura 17. **Mapa de memoria del DS18B20**



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.6.1. Registro de configuración

El byte 4 de la SPM contiene el registro de configuración, organizado como se muestra en la figura 18. El usuario puede configurar la resolución de la conversión del DS18B20 usando los bits R0 y R1 del registro como se muestra en la tabla V. La configuración por defecto de esos bits al encender el dispositivo es R0 = 1 y R1 = 1 (resolución de 12 bits). Nótese que existe una compensación directa entre la resolución y el tiempo de conversión. El bit 7 y los bits del 0 al 4 en el registro de configuración están reservados para uso interno del dispositivo y no pueden ser sobrescritos.

Figura 18. Registro de configuración

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

Tabla V. Configuración de resolución

R1	R0	Resolución (bits)	Tiempo máximo de conversión	
0	0	9	93.75 ms	($t_{CONV} / 8$)
0	1	10	187.5 ms	($t_{CONV} / 4$)
1	0	11	375 ms	($t_{CONV} / 2$)
1	1	12	750 ms	(t_{CONV})

Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.7. Generación de CRC

Los bytes CRC están incluidos como parte del código ROM de 64 bits del DS18B20 y en el noveno byte de la SPM. El código CRC de la ROM es calculado de los primeros 56 bits del código ROM y está contenido en los MSB de la ROM. El CRC de la SPM es calculado de los datos almacenados en la SPM y por lo tanto, cambia cuando los datos en la SPM cambian. El CRC proporcional al bus maestro con un método de validación de datos cuando los datos son leídos del DS18B20. Para verificar que los datos han sido leídos correctamente, el bus maestro recalcula el CRC de los datos recibidos y luego compara ese valor ya sea con el código ROM (la ROM lee) o con la SPM (la SPM lee). Si el CRC calculado concuerda con el CRC leído, los datos han sido recibidos sin error. La comparación de los valores CRC y la decisión de continuar con la operación son determinadas completamente por el bus maestro. No hay circuitería dentro del DS18B20 que prevenga que se inicie una secuencia de comandos si el CRC del DS18B20 (ROM o SPM) no concuerde con el valor generado por el bus maestro.

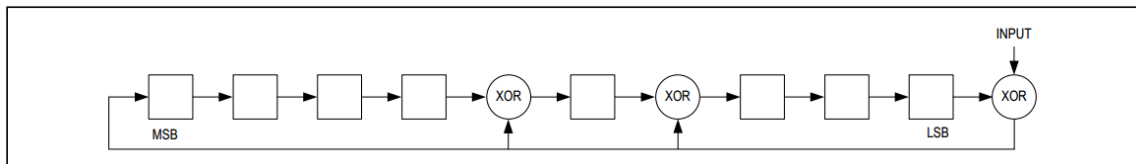
La función polinomial equivalente del CRC (ROM o SPM) es:

$$CRC = X^8 + X^5 + X^4 + 1$$

El bus maestro puede recalcular el CRC y compararlo con el valor CRC del DS18B20 usando un generador polinomial mostrado en la figura 19. El circuito consiste de un registro de desplazamiento y compuertas XOR y los bits del registro de desplazamiento son inicializados en 0. Empezando con el LSB del código de la ROM o con el LSB del byte 0 de la SPM, un bit a la vez debe desplazarse al registro de desplazamiento. Luego del desplazamiento del bit 56 de la ROM o el MSB del byte 7 de la SPM, el generador polinomial contendrá el CRC recalculado. Seguido, el CRC del código de 8 bits de la ROM o la SPM

debe ser insertado al circuito. En este punto, si el CRC recalculado fue correcto, el registro de desplazamiento contendrá solamente ceros.

Figura 19. **Generador de CRC**



Fuente: DS18B20 Programmable Resolution 1-Wire Digital Thermometer.
<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>. Consulta: febrero 2017.

2.1.8. Comandos de función del DS18B20

Luego que el bus maestro ha usado un comando ROM para direccionar el DS18B20 con quien el desee comunicarse, el maestro puede enviar un comando de función del DS18B20. Estos comandos permiten al maestro escribirle y leer la SPM del DS18B20, iniciar la conversión de temperatura y determinar el modo de alimentación. Los comandos de función del DS18B20, están descritos a continuación, los cuales están resumidos en la tabla VI.

2.1.8.1. Convert T [44h]

Este comando inicia una conversión de temperatura. Seguido de esta conversión, los datos de temperatura resultantes son almacenados en los registros de temperatura de 2 bytes de la SPM y el DS18B20 regresa a su estado inactivo de bajo consumo. Si el dispositivo está siendo usado en el modo de poder parásito, dentro de los siguientes 10 μ s (máximo) luego que el comando fue emitido, el maestro debe habilitar un *pullup* fuerte en el bus 1-Wire

para la duración de la conversión (t_{CONV}). Si el DS18B20 está alimentado por una fuente externa, el maestro puede emitir una lectura por espacios de tiempo luego del comando Convert T y el DS18B20 responderá transmitiendo un 0 mientras la conversión de temperatura está en proceso y un 1 cuando la conversión se finaliza. En el modo de poder parásito, esta técnica de notificación no puede ser usada, ya que el bus es llevado a un estado alto por un *pullup* fuerte durante la conversión.

2.1.8.2. Write Scratchpad [4Eh]

Este comando permite al maestro escribir 3 bytes de datos a la SPM del DS18B20. El primer byte de data es escrito en el registro T_H (byte 2 de la SPM), el segundo byte es escrito en el registro T_L (byte 3 de la SPM) y el tercer byte es escrito en el registro de configuración (byte 4 de la SPM). Los datos deben ser transmitidos por el LSB primero. Los 3 bytes deben ser escritos antes de que el maestro emita un *reset* o los datos pueden ser corrompidos.

2.1.8.3. Read Scratchpad [BEh]

Este comando permite al maestro leer el contenido de la SPM. Los datos transferidos comienzan con el LSB del byte 0 y continua a través de la SPM hasta que el byte 9 (byte 8 – CRC) es leído. El maestro puede emitir un *reset* para terminar la lectura en cualquier momento, si solamente una parte de los datos de la SPM son necesarios.

2.1.8.4. Copy Scratchpad [48h]

Este comando copia el contenido de los registros T_H , T_L y de configuración de la SPM (byte 2, 3 y 4) a la EEPROM. Si el dispositivo está siendo usado en el modo de poder parásito, dentro de los siguientes 10 μ s (máximo) luego que el comando fue emitido, el maestro debe habilitar un *pullup* fuerte en el bus 1-Wire por al menos 10 ms.

2.1.8.5. Recall E^2 [B8h]

Este comando lee nuevamente los valores umbrales de alarma (T_H y T_L) y los datos de configuración de la EEPROM y coloca los datos en los bytes 2, 3 y 4, respectivamente, en la SPM. El dispositivo maestro puede emitir una lectura por intervalos de tiempo seguido del comando Recall E^2 y el DS18B20 indicará el estado de lectura transmitiendo 0 mientras la lectura está en progreso y 1 cuando la lectura está terminada. La operación de lectura ocurre automáticamente al encender, por lo que los datos validados están disponibles en la SPM tan pronto la energía es aplicada al dispositivo.

2.1.8.6. Read Power Supply [B4h]

El dispositivo maestro emite este comando seguido por una lectura por intervalos de tiempo para determinar si algún DS18B20 en el bus está usando poder parásito. Durante la lectura por intervalos de tiempo, los DS18B20 con poder parásito pondrán en bajo el estado de bus y los DS18B20 con fuente externa, dejarán en alto el estado del bus.

Tabla VI. **Comandos del sensor DS18B20**

Comando	Descripción	Protocolo	Actividad del bus 1-Wire luego de la emisión del comando
Comandos de conversión de temperatura			
Convert T	Inicia una conversión de temperatura	44h	DS18B20 transmite el estado de la conversión al maestro.
Comandos de memoria			
Read Scratchpad	Lee la SPM completa incluyendo el byte CRC	BEh	DS18B20 transmite hasta 9 bytes de datos al maestro.
Write Scratchpad	Escribe los datos en los bytes 2, 3 y 4 de la SPM (registros T_H , T_L y de configuración)	4Eh	El maestro transmite 3 bytes de data al DS18B20.
Copy Scratchpad	Copia los datos de los registros T_H , T_L y de configuración de la SPM a la EEPROM.	48h	Ninguna
Recall E ²	Lee nuevamente los datos de los registros T_H , T_L y de configuración de la EEPROM a la SPM.	B8h	DS18B20 transmite el estado de la lectura al maestro.
Read Power Supply	Señala el modo de alimentación de los DS18B20 al maestro.	B4h	DS18B20 transmite el estado de la alimentación al maestro.

Fuente: elaboración propia.

2.2. Arduino Uno

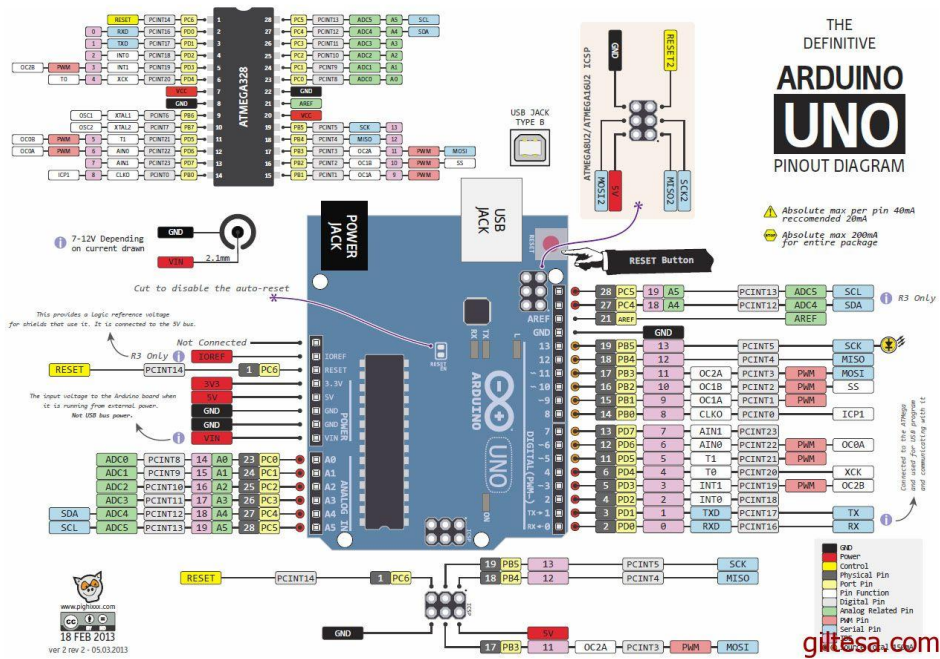
El Arduino Uno es una tarjeta microcontroladora basada en el ATmega328P. Tiene 14 pines digitales entrada/salida (de los cuales 6 pueden ser usados como salidas PWM), 6 entradas analógicas, un cristal de cuarzo de 16 MHz, una conexión USB, un conector de poder, un encabezado ICSP y un botón de reinicio.

2.2.1. Especificaciones técnicas

A continuación se presentan las especificaciones de hardware presentadas por el Arduino Uno.

• Microcontrolador	ATmega328P
• Voltaje de operación	5 V
• Voltaje de entrada (recomendado)	7 - 12 V
• Voltaje de entrada (límites)	6 - 20 V
• Pines E/S digitales	14
• Pines entrada análogos	6
• Corriente DC por pin E/S	20 mA
• Corriente DC por pin 3,3 V	50 mA
• Memoria flash	32 KB
• SRAM	2 KB
• EEPROM	1 KB
• Velocidad de reloj	16 MHz

Figura 20. Diagrama del Arduino Uno



Fuente: <https://www.arduino.cc/en/Hacking/PinMapping168>. Consulta: abril 2017.

2.2.1.1. Alimentación

La tarjeta del Arduino Uno puede ser alimentada a través de una conexión USB o con una fuente externa. La fuente de poder es seleccionada automáticamente. La alimentación externa puede provenir ya sea de un adaptador AC a DC o de baterías. La tarjeta puede operar con una fuente externa de 6 a 20 voltios. Si se suministra con menos de 7 V, el pin de 5 V puede suministrar menos de cinco voltios y la tarjeta puede volverse inestable. Si se usa más de 12 V, el regulador de voltaje se puede sobrecalentar y dañar la tarjeta. El rango recomendado es de 7 a 12 voltios.

Los pines de energía son descritos a continuación:

- V_{IN} : el voltaje de entrada a la tarjeta Arduino cuando está usando una fuente de poder externa (ya sea los 5 voltios de la conexión USB u otra fuente de poder regulada). Se puede suministrar voltaje a través de este pin, o, si se está suministrando voltaje por el conector de poder, acceder al voltaje por este pin.
- 5 V: este pin provee 5 V regulados del regulador en la tarjeta. La tarjeta puede proveer con energía ya sea con el conector DC, el conector USB o el pin V_{IN} de la tarjeta. Suministrar voltaje a través de los pines de 5V o de 3,3 V pasa por el regulador y puede dañar la tarjeta. Por lo que no se recomienda suministrar voltaje en esos pines.
- 3V3: este pin suministra 3,3 voltios por el regulador que posee la tarjeta. El máximo consumo de corriente en el pin es de 50 mA.
- GND: pin de tierra.
- IOREF: este pin en la tarjeta Arduino provee un voltaje de referencia con el cual opera el microcontrolador. Un escudo correctamente configurado puede leer el voltaje del pin IOREF y seleccionar la fuente de alimentación apropiada o habilitar los traductores de voltaje en las salidas para trabajar con 5 V o 3,3 V.

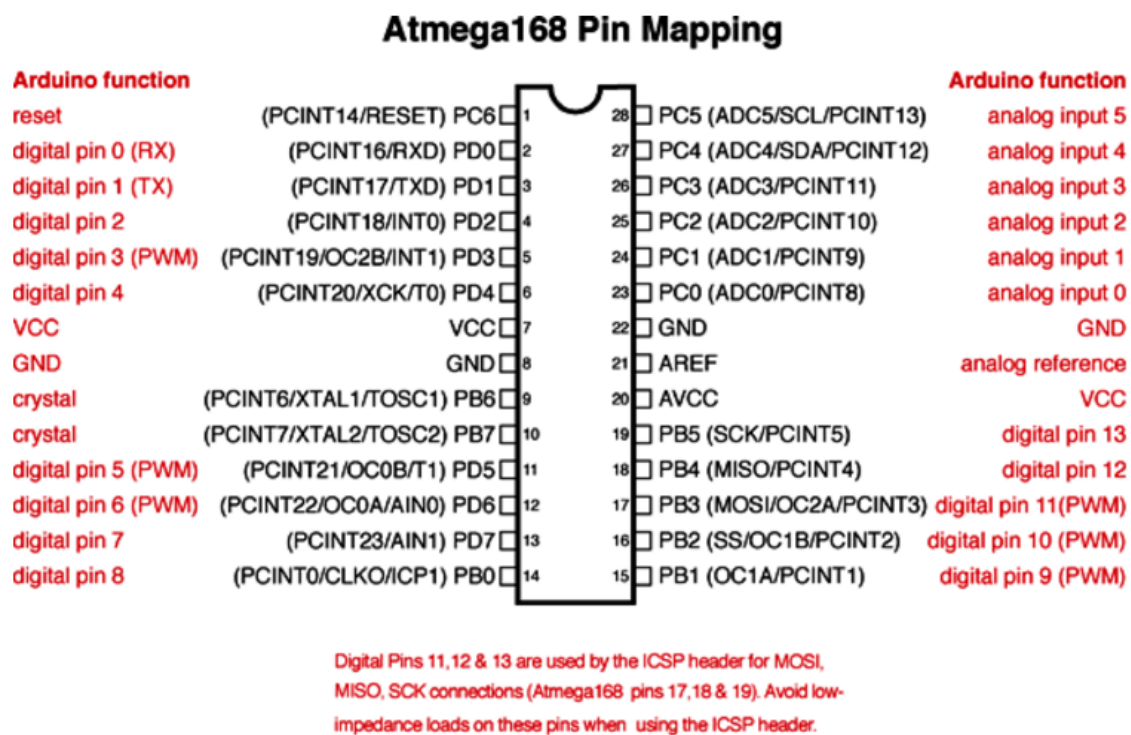
2.2.1.2. Memoria

El ATmega328 tiene 32 KB de memoria, de los cuales 0,5 KB son ocupados por el sistema de arranque. Así mismo tiene 2 KB de SRAM y 1 KB de EEPROM, la cual puede ser leída o escrita con la librería EEPROM.

2.2.1.3. Entradas y salidas

El mapeo entre los pines del Arduino y los puertos del ATmega328P se puede ver en la figura 21.

Figura 21. Mapeo de los puertos



Fuente: <https://www.arduino.cc/en/Hacking/PinMapping168>. Consulta: abril 2017.

Cada uno de los 14 pines digitales del Uno puede ser usada como entrada o salida, usando las funciones `pinMode()`, `digitalWrite()` y `digitalRead()` y operan a 5 voltios. Cada pin puede proveer o recibir 20 mA como condición de operación recomendada y tienen un resistor interno de *pullup*, desconectado por defecto, de 20 – 50 KΩ. Un valor máximo de 40 mA es lo que no debe

exceder cualquiera de los pines E / S para evitar daños permanentes al microcontrolador.

Algunos pines tienen funciones especializadas, presentados a continuación:

- Serial: estos pines son usados para transmitir (T_x) y recibir (R_x) datos seriales TTL. Están conectados a los pines correspondientes del chip serial ATmega8U2 USB a TTL. El pin 0 es R_x y el pin 1 es T_x .
- Interrupciones externas: éstas interrupciones están en los pines 2 y 3. Estos pines pueden ser configurados para disparar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en un valor.
- PWM: están ubicados en los pines 3, 5, 6, 9, 10 y 11. Estos pines proveen una salida PWM de 8 bits con la función `analogWrite()`.
- SPI: estos pines soportan comunicación SPI usando la librería SPI. La configuración de los pines se presenta a continuación: pin 10 (SS), pin 11 (MOSI), pin 12 (MISO) y pin 13 (SCK).
- Led: en la tarjeta se encuentra un led manejado por el pin digital 13. Cuando el pin tiene un valor alto, el led está encendido, cuando el pin está en un valor bajo, el led está apagado.
- AREF: este pin provee un voltaje de referencia para las entradas análogas.

El Arduino Uno tiene 6 entradas análogas, etiquetadas desde A0 hasta A5, cada una provee una resolución de 10 bits (1 024 valores diferentes). Por defecto miden del valor de tierra a 5 voltios, sin embargo, es posible cambiar el límite superior del rango usando el pin AREF y la función `analogReference()`.

2.2.1.4. Comunicación

El Arduino Uno tiene varias facilidades para comunicarse con una computadora, otra tarjeta Arduino u otro tipo de microcontroladores. El ATmega328 proporciona comunicación serial UART TTL (5V), la cual está disponible en los pines digitales 0 y 1. Un ATmega16U2 en la tarjeta conduce esa comunicación serial a través del USB y aparece como un puerto COM virtual al software en la computadora. El *firmware* del 16U2 usa los *drivers* estándares del USB COM y no se necesitan *drivers* externos. El software de Arduino (IDE) incluye un monitor serial que permite que los datos de texto sean enviados hacia y desde la tarjeta.

2.2.2. Programación

El Arduino Uno puede ser programado con el software Arduino IDE. El ATmega328 en el Arduino Uno viene preprogramada con un sistema de arranque que permite que un nuevo código pueda ser subido al Arduino sin el uso de un programador externo, utilizando el protocolo STK500 original.

También se puede omitir el sistema de arranque y programar el microcontrolador a través del encabezado ICSP (In-Circuit Serial Programming) usando el Arduino ISP o algún similar.

3. PROPUESTA DE DISEÑO DEL SISTEMA

El diseño propuesto será un sistema versátil para que pueda ser implementado con facilidad en distintas aplicaciones de control y manejo de temperaturas, ya que el sensor DS18B20 presenta un encapsulado metálico que permitirá la medición de temperaturas para líquidos como también para gases que estén en un rango entre -55 °C y 125 °C. Algunas aplicaciones donde podría ser implementado el sistema son:

- Domótica (control de temperatura para un edificio o casa)
- Intercambiadores de calor
- Líneas de producción
- Acuicultura (control de temperatura de los estanques)

El sistema propuesto se basará en dos partes, la primera parte será la encargada del sistema de control del sensor y la segunda parte será la interfaz gráfica del usuario la cual se va a comunicar con el sistema de control para poder presentar los datos deseados. El sistema de control de temperatura contará con una interfaz gráfica para el usuario desde la cual podrá interactuar con la aplicación creada, mediante la interfaz podrá iniciar / terminar la conexión hacia la terminal de control, leer la temperatura de los sensores, configurar algunos parámetros de los sensores (temperaturas de alarma), consultar la información de los sensores (ID, modo de proveer la energía, temperaturas de alarma), también podrá graficar un histórico de las temperaturas que se obtuvieron en un determinado tiempo y exportar los datos en un archivo de Excel para un manejo más detallado de los mismos.

El programa establece una comunicación tanto con la PC como con los sensores. La conexión entre la computadora (interfaz gráfica) y el microprocesador (sistema de control) se realiza mediante un bus serial y la conexión entre el microprocesador (sistema de control) y los sensores de temperatura se realiza mediante un bus 1-Wire.

La programación de la parte del sistema de control se desarrolla en el programa Arduino IDE y la programación de la parte de la interfaz gráfica se desarrolla en MATLAB R2012b.

3.1. Sistema de control – Arduino IDE

El sistema de control desarrollado para los sensores DS18B20 abarca la mayoría de funcionalidades descritas en la sección 2.1.

El programa comienza estableciendo una comunicación con la computadora, la cual queda a la espera hasta que se inicie la conexión serial. El código de esta función se detalla en la siguiente figura.

Figura 22. Iniciar comunicación serial - código

```
// Establecer comunicación Serial
Serial.println('a'); //enviar en el bus serial la letra 'a'
char a = 'b'; //declarar y asignar letra 'b' a la variable a
while (a != 'a'){ //ciclo para esperar la conexión de la PC
  a = Serial.read(); //lee el bus serial
}
```

Fuente: elaboración propia, empleando Arduino IDE.

Una vez iniciada la conexión, el Arduino queda a la espera que le envíen un comando (en este caso es el uso solamente de una letra) dependiendo de la

acción se quiera realizar el usuario. Una vez recibido el comando el Arduino realiza el subprograma con la función especificada, ya sea la obtención o envío de información a los sensores mediante el bus 1-Wire. El código de estas funciones se detalla en la siguiente figura.

Figura 23. **Funciones del programa - código**

```
void loop(void)
{
  if (Serial.available() >0){ //queda a la espera de que la PC
  envíe por el bus serial
    mode = Serial.read(); //la lectura del puerto se asigna en la
    variable mode
    switch (mode){
      case 'R': //condición si se recibe la letra 'R'
        getTempCent(); //función de lectura de temperatura
        break;
      case 'I': //condición si se recibe la letra 'I'
        getInfo(); //función de información de sensores
        break;
      case 'C': //condición si se recibe la letra 'C'
        getSensors(); //función de verificación de sensores
        break;
    }}
}
```

Fuente: elaboración propia, empleando Arduino IDE.

La información obtenida es enviada a la computadora por el bus serial para ser mostrado al usuario. Una vez terminada la función regresa al ciclo de espera de un comando por parte del usuario.

Para ver a detalle las funciones de cada subprograma del sistema y el programa completo, consultar la sección de apéndices.

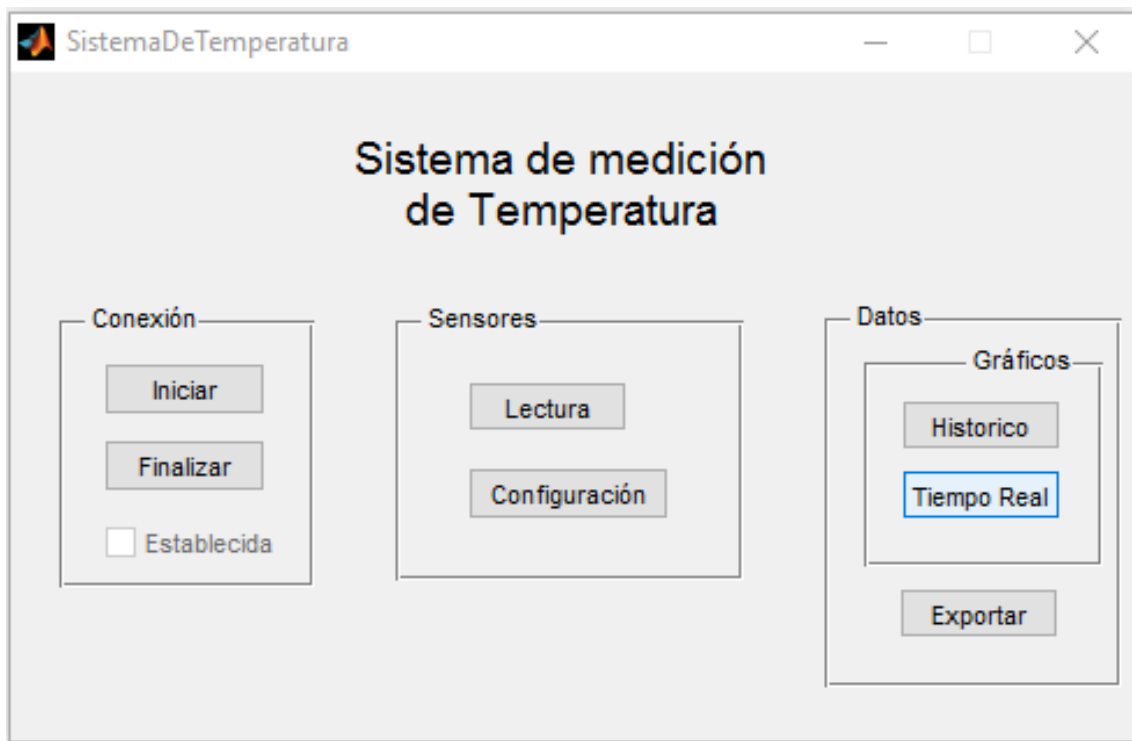
3.2. Interfaz gráfica – MATLAB R2012b

La interfaz gráfica cuenta con varias ventanas donde se pueden acceder a las distintas funcionalidades programadas para el usuario final.

3.2.1. Ventana principal

La ventana principal, mostrada en la figura 24, contiene todo el control de la interfaz del usuario.

Figura 24. Ventana principal



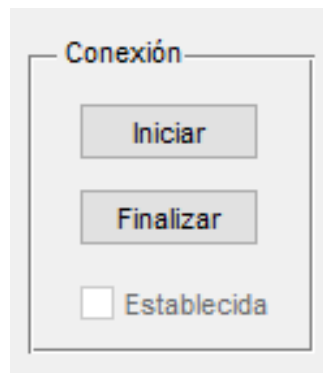
Fuente: elaboración propia, empleando MATLAB R2012b.

En la ventana principal se pueden encontrar tres paneles descritos a continuación: el panel de conexión, el panel de sensores y el panel de datos.

3.2.1.1. Panel de conexión

El panel de conexión, mostrado en la figura 25, contiene dos botones, los cuales permiten al usuario iniciar y finalizar la conexión hacia el microprocesador y un *check box* para indicar si la conexión ha sido establecida correctamente o hubo algún problema.

Figura 25. Panel de conexión



Fuente: elaboración propia, empleando MATLAB R2012b.

El código del botón que inicia la conexión serial con el microprocesador se detalla en la siguiente figura.

Figura 26. Iniciar conexión serial - código

```
function openSerial_Callback(hObject, eventdata, handles)
global s; %variable global s para el uso del puerto serial
s = serial('COM3'); %se establece el puerto serial en 'COM3' y se le asigna
la variable s
set(s, 'DataBits', 8); %configuraciones del puerto serial
set(s, 'StopBits', 1);
set(s, 'BaudRate', 9600);
set(s, 'Parity', 'none');
fopen(s) %se abre la comunicación con el Arduino
a = 'b'; %asigna a la variable "a" una letra distinta de a
while (a ~= 'a') %se crea un ciclo que mientras no se lea la letra a,
mantenga leyendo
    a = fread(s, 1, 'uchar'); %lectura del puerto serial
end
if (a == 'a') %condición de igualdad
    disp('Comunicación exitosa'); %si la letra recibida es a despliega un
mensaje
end
fprintf(s, '%c', 'a'); %luego de recibir la letra a, se envia la misma
letra al arduino
mbox = msgbox('Comunicación iniciada'); %despliega mensaje en ventana
set(handles.serialStatus, 'Value', 1); %se habilita el check box
uiwait(mbox); %esperamos para que el usuario pueda serar la ventana
fclose(s, 'u'); %queda abierto el puerto para recibir siguiente comando
```

Fuente: elaboración propia, empleando MATLAB R2012b.

La función es establecer una conexión serial correcta mediante el envío de un carácter específico, en este caso se utilizó el carácter 'a', que al momento de estar presente tanto en la computadora como en el microprocesador, la comunicación se estableció correctamente.

El código del botón que finaliza la conexión serial con el microprocesador se detalla en la figura 27.

Figura 27. **Finalizar conexión serial - código**

```
function closeSerial_Callback(hObject, eventdata, handles)
global s;
fclose(s); %cerrando el puerto serial
delete(s); %borrando la variable s
clear s; %limpiando la variable s
disp('Comunicación terminada'); %mensaje de que el puerto fue cerrado
correctamente
mbox = msgbox('Comunicación terminada');
set(handles.serialStatus, 'Value', 0); %removiendo el estado del check box
uiwait(mbox);
```

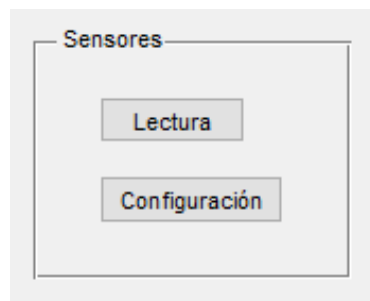
Fuente: elaboración propia, empleando MATLAB R2012b.

La función es cerrar el puerto serial con el microprocesador.

3.2.1.2. **Panel de sensores**

El panel de sensores, mostrado en la figura 28, contiene la interacción con los sensores tanto la parte de lectura como de configuración. Cuenta con dos botones, los cuales permiten al usuario acceder a una ventana para la lectura de los sensores, también permite acceder a otra ventana con las configuraciones de los mismos

Figura 28. **Panel de sensores**



Fuente: elaboración propia, empleando MATLAB R2012b.

Debido a que los botones presentes en el panel de sensores despliegan otra ventana, el código fuente no es más que el despliegue de las ventanas de lectura y configuración respectivamente.

Figura 29. **Panel de sensores - código**

```
% --- Executes on button press in openRead.  
function openRead_Callback(hObject, eventdata, handles)  
Temperaturas; %despliega la ventana de lectura de temperaturas  
% --- Executes on button press in config.  
function config_Callback(hObject, eventdata, handles)  
Configuracion; %despliega la ventana de configuración de sensores
```

Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.1.3. **Panel de datos**

El panel de datos, mostrado en la figura 30, contiene el despliegue de los datos capturados por el sensor en las lecturas previas. Puede desplegar, ya sea una gráfica en tiempo real como también una gráfica del histórico de las temperaturas, también pueden exportarse los datos capturados a un libro de Excel.

Figura 30. **Panel de datos**

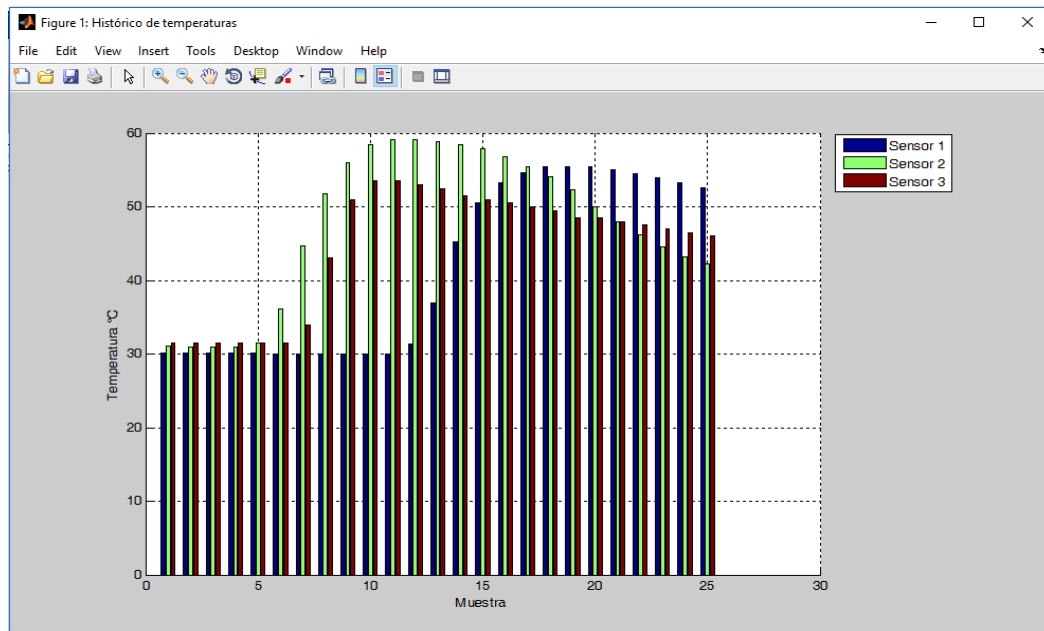


Fuente: elaboración propia, empleando MATLAB R2012b.

En el panel de gráficos se presentan dos opciones, la gráfica de un histórico de temperaturas guardadas o una gráfica de las temperaturas en tiempo real. En el caso de los gráficos históricos, se presenta un diagrama de barras verticales con las temperaturas de los datos que se guardaron de los sensores que estaban en línea. Adicional al gráfico en conjunto de los sensores, se despliega un gráfico donde se muestra el comportamiento individual de los sensores presentes en el bus mediante un análisis de un gráfico de barras donde se despliegan los datos almacenados.

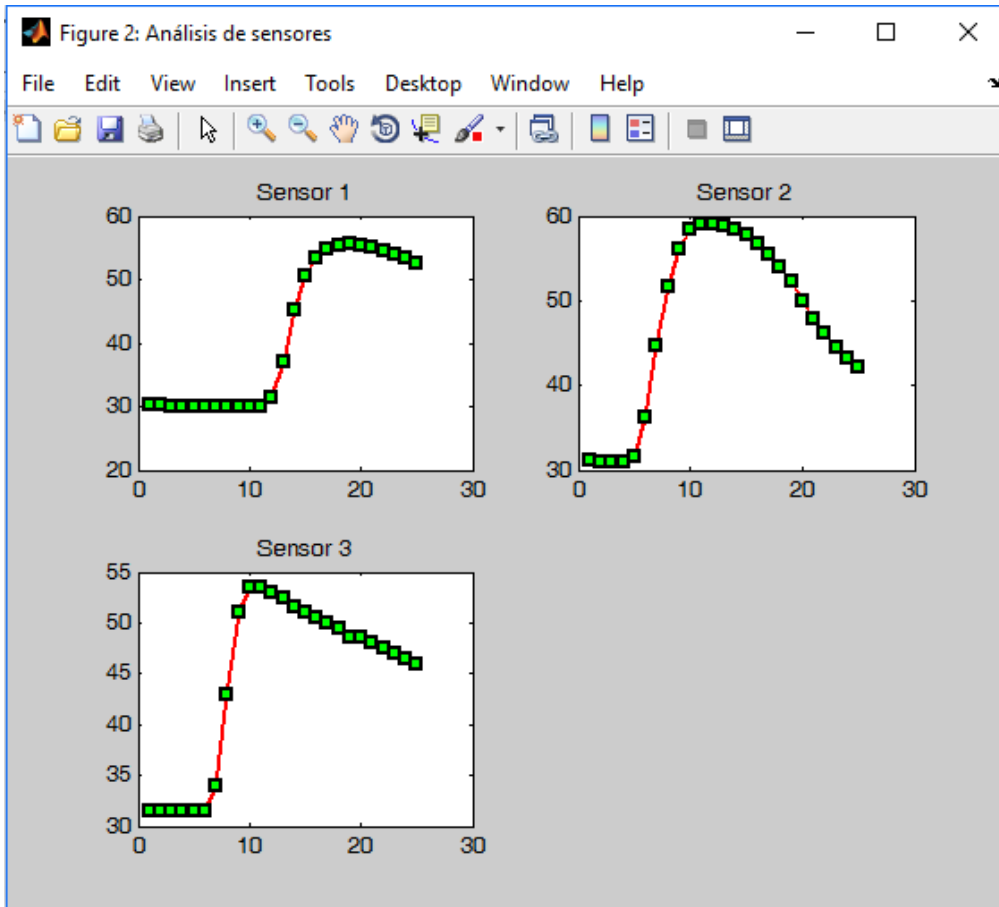
Los datos presentados en el gráfico individual son los mismos datos que se presentan en el gráfico grupal, la diferencia es el tipo de análisis que se puede efectuar en cada uno de los casos. El gráfico en conjunto es mostrado en la figura 31 y los gráficos individuales en la figura 32.

Figura 31. **Gráfico histórico de los sensores**



Fuente: elaboración propia, empleando MATLAB R2012b.

Figura 32. **Gráfico histórico sensores individuales**



Fuente: elaboración propia, empleando MATLAB R2012b.

El despliegue de estos gráficos se realiza con el código que se muestra en la siguiente figura.

Figura 33. Gráficos históricos de los sensores - código

```
function historyGraph_Callback(hObject, eventdata, handles)
global datos; %se utiliza la variable global 'datos' que contiene las
temp. guardadas
x = 1:1:25; %se crea un vector con el número de muestras que se tengan
guardadas
temp1 = zeros(25,1); %se crea un vector para almacenar los datos del
sensor 1
temp2 = zeros(25,1); %se crea un vector para almacenar los datos del
sensor 2
temp3 = zeros(25,1); %se crea un vector para almacenar los datos del
sensor 3

temp1 = datos(:,1); %se guardan las temp. guardadas del sensor 1 en el
vector anterior
temp2 = datos(:,2); %se guardan las temp. guardadas del sensor 2 en el
vector anterior
temp3 = datos(:,3); %se guardan las temp. guardadas del sensor 3 en el
vector anterior

%Gráfica con sensores en conjunto
figure('Name','Histórico de temperaturas'); %nombre de la ventana donde
se despliega la figura
xlabel('Muestra'); %nombre del eje de las x
ylabel('Temperatura °C'); %nombre del eje de las y
grid on; %se coloca un cuadrículado de fondo en la gráfica
hold on; %se deja la ventana hasta que el usuario decida cerrarla

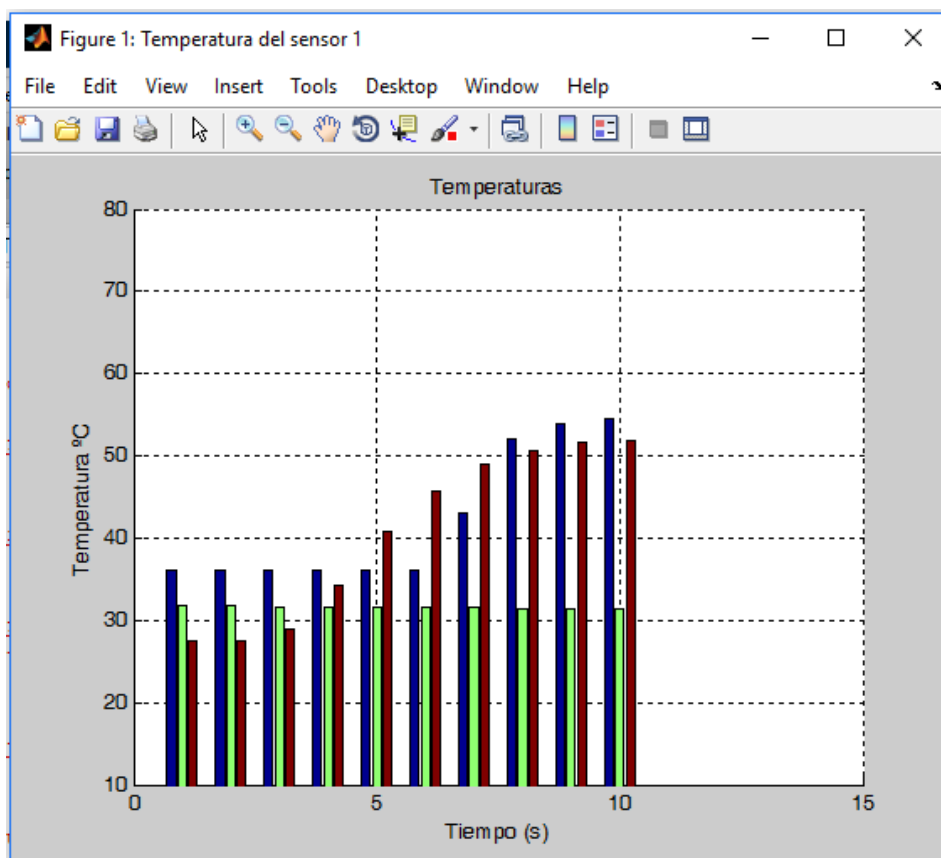
bar(datos, 'grouped'); %se hace la gráfica de las temp. guardadas en
barras verticales
legend('Sensor 1', 'Sensor 2', 'Sensor 3','Location','NorthEastOutside')
%se habilita la leyenda del gráfico y se indica la ubicación de la misma

%Gráfica con sensores individuales
figure('Name','Análisis de sensores'); %nombre de la ventana donde se
despliega la figura
subplot(2,2,1); %ubicación del sensor 1 dentro de la ventana
plot(x, temp1, '-
rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize
',5)
title('Sensor 1') %nombre de la gráfica 1
subplot(2,2,2); %ubicación del sensor 2 dentro de la ventana
plot(x, temp2, '-
rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize
',5)
title('Sensor 2') %nombre de la gráfica 2
subplot(2,2,3); %ubicación del sensor 3 dentro de la ventana
plot(x, temp3, '-
rs','LineWidth',2,'MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize
',5)
title('Sensor 3') %nombre de la gráfica 3
```

Fuente: elaboración propia, empleando MATLAB R2012b.

La otra opción dentro del panel de gráfico es la de tiempo real, esta gráfica se presenta como barras verticales con los sensores que están en línea al momento de realizar la medición. Cada uno de los sensores es representado por un color diferente. La gráfica se muestra en la siguiente figura.

Figura 34. **Gráfico de temperaturas – tiempo real**



Fuente: elaboración propia, empleando MATLAB R2012b.

El código que permite la gráfica de temperaturas en tiempo real se presenta en la siguiente figura.

Figura 35. **Gráfico temperaturas en tiempo real – código**

```
% --- Executes on button press in realtimeGraph.
function realtimeGraph_Callback(hObject, eventdata, handles)
global s; %se importa la variable de la conexión serial

y = zeros(20,3); %se crea un arreglo para guardar las temp.
contador_muestras = 1; %variable de control
numero_muestras = 20; %variable de control

figure('Name','Temperaturas en tiempo real'); %nombre de la figura
xlabel('Muestra'); %nombre del eje de las x
ylabel('Temperatura °C'); %nombre del eje de las y
grid on; %se coloca un cuadrículado en el fondo de la figura
hold on;

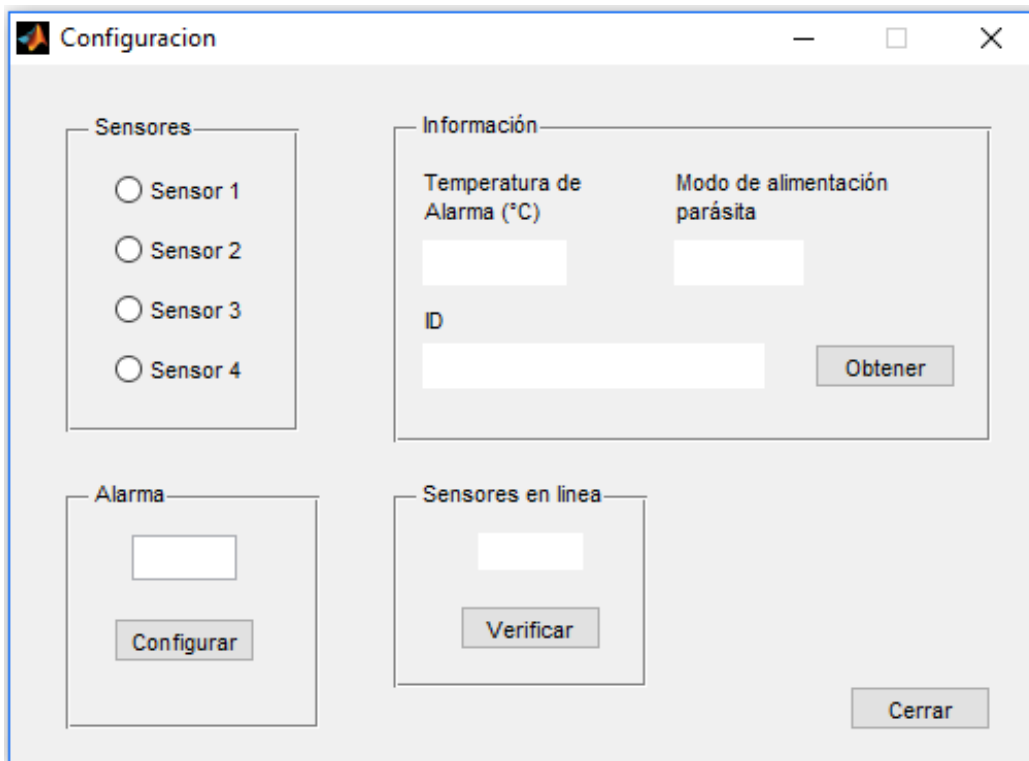
while contador_muestras <= numero_muestras %ciclo para graficar en tiempo
real
    ylim([10 80]); %limites del eje de las y
    xlim([contador_muestras-1 contador_muestras+5]); %límite del eje de
las x
    fprintf(s, 'R'); %lectura de los sensores, envía letra 'R' al
microcontrolador para efectuar lectura
    temp1 = fscanf(s, '%f'); %lee temperatura del sensor 1
    pause(0.5);
    temp2 = fscanf(s, '%f'); %lee temperatura del sensor 2
    pause(0.5);
    temp3 = fscanf(s, '%f'); %lee temperatura del sensor 3
    pause(0.5);
    y(contador_muestras, 1) = temp1; %asignación de las temperaturas
y(contador_muestras, 2) = temp2; %obtenidas al arreglo el cual se
y(contador_muestras, 3) = temp3; %va a graficar
    bar(y, 'grouped'); %gráfica de barras de las temperaturas
    drawnow %actualiza la gráfica cada ciclo completado
    contador_muestras = contador_muestras+1; %aumenta la variable de
control
end
```

Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.2. Ventana de configuración de sensor

La ventana de configuración, mostrada en la figura 36, contiene las opciones para leer y escribir algunas configuraciones del sensor.

Figura 36. **Ventana de configuración**



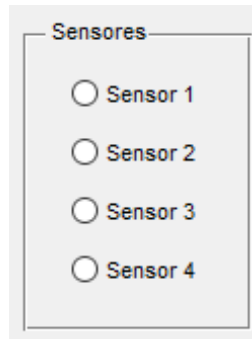
Fuente: elaboración propia, empleando MATLAB R2012b.

En la ventana de configuración se pueden encontrar tres paneles descritos a continuación: sensores, información, alarma y sensores en línea.

3.2.2.1. **Panel de sensores**

El panel de sensores, mostrado en la figura 37, permite realizar la selección de los sensores que están en línea para configurar ya sea una alarma de temperatura u obtener la información correspondiente al panel de información.

Figura 37. Panel de sensores



Fuente: elaboración propia, empleando MATLAB R2012b.

La función del panel descrito y mostrado anteriormente es la selección de solamente un sensor para el despliegue de su información o su configuración de alarma. Esto se realiza mediante las funciones de *callback*, al momento de seleccionar cualquiera de los sensores, automáticamente se elimina la sección anterior. El código del panel se presenta en la siguiente figura.

Figura 38. Panel de sensores – código

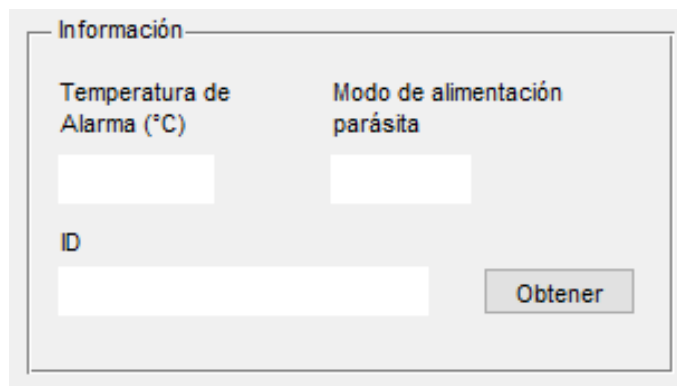
```
% --- Executes on button press in sensor1.
function sensor1_Callback(hObject, eventdata, handles)
set(handles.sensor2, 'Value', 0); %deselecciona el sensor 2
set(handles.sensor3, 'Value', 0); %deselecciona el sensor 3
set(handles.sensor4, 'Value', 0); %deselecciona el sensor 4
function sensor2_Callback(hObject, eventdata, handles)
set(handles.sensor1, 'Value', 0); %deselecciona el sensor 1
set(handles.sensor3, 'Value', 0); %deselecciona el sensor 3
set(handles.sensor4, 'Value', 0); %deselecciona el sensor 4
function sensor3_Callback(hObject, eventdata, handles)
set(handles.sensor1, 'Value', 0); %deselecciona el sensor 1
set(handles.sensor2, 'Value', 0); %deselecciona el sensor 2
set(handles.sensor4, 'Value', 0); %deselecciona el sensor 4
function sensor4_Callback(hObject, eventdata, handles)
set(handles.sensor1, 'Value', 0); %deselecciona el sensor 1
set(handles.sensor2, 'Value', 0); %deselecciona el sensor 2
set(handles.sensor3, 'Value', 0); %deselecciona el sensor 3
```

Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.2.2. Panel de información

El panel de información, mostrado en la figura 39, muestra la información del sensor como su ID único, el modo de alimentación al que está conectado y la alarma configurada.

Figura 39. Panel de información



El panel de información, titulado "Información", contiene los siguientes campos y botones:

Temperatura de Alarma (°C)	Modo de alimentación parásita
<input type="text"/>	<input type="text"/>
ID	<input type="text"/>
	<input type="button" value="Obtener"/>

Fuente: elaboración propia, empleando MATLAB R2012b.

El uso de este panel es mediante la selección de uno de los sensores del panel anterior, una vez seleccionado uno de los sensores se puede obtener la información disponible en el panel. Esta información es solicitada al microcontrolador que es el encargado del manejo de los sensores. El código utilizado se describe a continuación:

Figura 40. Panel de información – código

```
% --- Executes on button press in getInfo.
function getInfo_Callback(hObject, eventdata, handles)
%declaración de variables globales
global s; %variable de conexión serial
global alarmS1; %variable de alarma sensor 1
global alarmS2; %variable de alarma sensor 2
global alarmS3; %variable de alarma sensor 3
%selección de sensores
selS1 = get(handles.sensor1, 'Value'); %selección de sensor 1
selS2 = get(handles.sensor2, 'Value'); %selección de sensor 2
selS3 = get(handles.sensor3, 'Value'); %selección de sensor 3

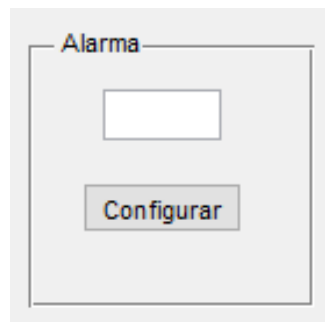
%condiciones de despliegue de información
if selS1 == 1 %condición de selección de sensor 1
    set(handles.dispAlarm, 'String', num2str(alarmS1));
    fprintf(s, 'I'); %envío de letra 'I' al microcontrolador para
    esperar información
    fprintf(s, '1'); %envío de número de sensor seleccionado
    power = fscanf(s, '%s'); %lectura del modo de energía del sensor
    seleccionado
    set(handles.dispPower, 'String', power); %despliegue del dato
    ID = fscanf(s, '%s'); %lectura del ID del sensor seleccionado
    set(handles.dispID, 'String', ID) %despliegue del dato
elseif selS2 == 1 & selS1 == 0
    set(handles.dispAlarm, 'String', num2str(alarmS2));
    fprintf(s, 'I'); %envío de letra 'I' al microcontrolador para
    esperar información
    fprintf(s, '2'); %envío de número de sensor seleccionado
    power = fscanf(s, '%s'); %lectura del modo de energía del sensor
    seleccionado
    set(handles.dispPower, 'String', power); %despliegue del dato
    ID = fscanf(s, '%s'); %lectura del ID del sensor seleccionado
    set(handles.dispID, 'String', ID) %despliegue del dato
elseif selS3 == 1 & selS2 == 0 & selS1 == 0
    set(handles.dispAlarm, 'String', num2str(alarmS3));
    fprintf(s, 'I'); %envío de letra 'I' al microcontrolador para
    esperar información
    fprintf(s, '3'); %envío de número de sensor seleccionado
    power = fscanf(s, '%s'); %lectura del modo de energía del sensor
    seleccionado
    set(handles.dispPower, 'String', power); %despliegue del dato
    ID = fscanf(s, '%s'); %lectura del ID del sensor seleccionado
    set(handles.dispID, 'String', ID) %despliegue del dato
else
    mbox = msgbox('Selección incorrecta');
end
```

Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.2.3. Panel de alarma

El panel de alarma, mostrada en la figura 41, permite al usuario programar una temperatura de alarma para los sensores en línea, se debe seleccionar el sensor en el que se desea programar la alarma y presionar el botón de configurar para que se guarde la temperatura.

Figura 41. Panel de alarma



Fuente: elaboración propia, empleando MATLAB R2012b.

El panel consta de un cuadro editable de texto donde se debe colocar la temperatura de alarma que se desea configurar en el sensor seleccionado en el panel anterior y un botón el cual permite guardar el valor de temperatura. El código utilizado se muestra en la siguiente figura.

Figura 42. Panel de alarma – código

```
% --- Executes on button press in setAlarm.
function setAlarm_Callback(hObject, eventdata, handles)
%declaración de variables globales
global s; %variable de conexión serial
global alarmS1; %variable de alarma sensor 1
global alarmS2; %variable de alarma sensor 2
global alarmS3; %variable de alarma sensor 3
%selección de sensores
S1 = get(handles.sensor1, 'Value'); %selección de sensor 1
S2 = get(handles.sensor2, 'Value'); %selección de sensor 2
S3 = get(handles.sensor3, 'Value'); %selección de sensor 3
%condiciones de establecer alarma
if S1 == 1
    alarmS1 = str2double(get(handles.alarmTemp, 'String'));
    %almacenamos el valor del cuadro de texto en la variable del sensor 1
    set(handles.alarmTemp, 'string', 0); %limpia el cuadro de texto
    mbox = msgbox('Temperatura configurada'); %despliegue de ventana de
    aviso
    uiwait(mbox);
elseif S2 == 1
    alarmS2 = str2double(get(handles.alarmTemp, 'String'));
    %almacenamos el valor del cuadro de texto en la variable del sensor 2
    set(handles.alarmTemp, 'string', 0); %limpia el cuadro de texto
    mbox = msgbox('Temperatura configurada'); %despliegue de ventana de
    aviso
    uiwait(mbox);
elseif S3 == 1
    alarmS3 = str2double(get(handles.alarmTemp, 'String'));
    %almacenamos el valor del cuadro de texto en la variable del sensor 3
    set(handles.alarmTemp, 'string', 0); %limpia el cuadro de texto
    mbox = msgbox('Temperatura configurada'); %despliegue de venatana de
    aviso
    uiwait(mbox);
else
    mbox = msgbox('No selecciono ningun sensor'); %mensaje de error
    uiwait(mbox);
end
```

Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.2.4. Panel de dispositivos conectados

El panel de dispositivos conectados, mostrado en la figura 43, permite al usuario revisar cuántos dispositivos están conectados en el bus.

Figura 43. **Panel de dispositivos conectados**



Fuente: elaboración propia, empleando MATLAB R2012b.

El panel consta de un cuadro de texto no editable donde se muestra el número de sensores presentes en el bus luego de presionar el botón de 'Verificar'. Esta verificación la realiza el microcontrolador (Arduino Uno) ya que es quien tiene el control sobre los sensores, se envía un código para que efectúe el subprograma que reporta el número de sensores. El código utilizado se muestra en la siguiente figura.

Figura 44. **Panel de dispositivos conectados - código**

```
% --- Executes on button press in countSensors.  
function countSensors_Callback(hObject, eventdata, handles)  
global s; %variable de conexión serial  
fprintf(s, 'C'); %envío de letra 'C' al microcontrolador para esperar  
información  
num = fscanf(s, '%s'); %lectura de los datos enviados  
set(handles.onlineSensors, 'String', num); %establecer el dato recibido  
en el cuadro de texto
```

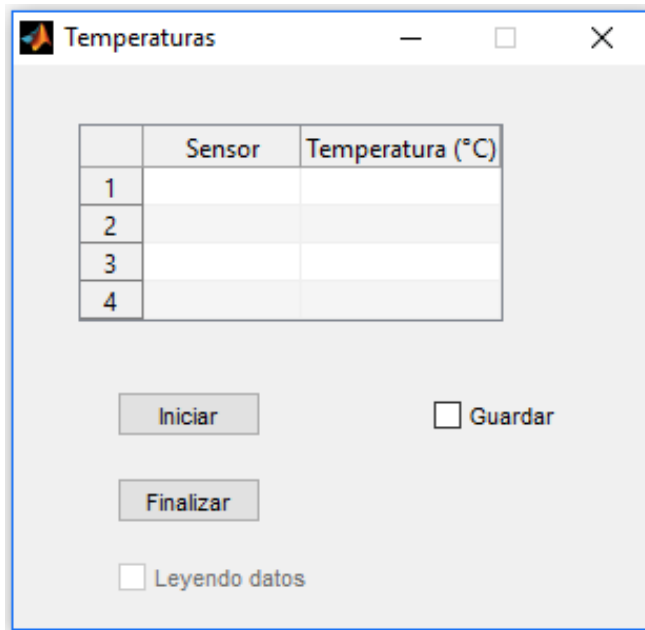
Fuente: elaboración propia, empleando MATLAB R2012b.

3.2.3. **Ventana de lectura de sensores**

La ventana de lectura de sensores, mostrada en la figura 45, permite al usuario interactuar con la lectura de las temperaturas de los sensores, puede

iniciar o detener la lectura, también tiene la opción de almacenar las lecturas de las temperaturas que se están realizando.

Figura 45. **Ventana de lectura**



Fuente: elaboración propia, empleando MATLAB R2012b.

La ventana de lectura es la más importante, ya que es en donde se efectúa la lectura y despliegue de las temperaturas obtenidas de los sensores. La ventana consta de una tabla, dos botones y un *checkbox* para la interacción con el usuario. Al momento de presionar el botón de 'Iniciar' se habilita un *checkbox* – no editable – que indica que se están leyendo los datos y las temperaturas que se leen de los sensores son desplegadas en la tabla y son actualizadas cada determinado tiempo. Mientras se está efectuando la lectura de las temperaturas se puede activar el *checkbox* – editable – con el nombre 'Guardar'. Esta acción permite que las temperaturas que se están recibiendo en los sensores sean almacenadas para luego realizar gráficas o exportar los

datos para realizar otro tipo de análisis en un documento de Excel. El botón 'Finalizar' detiene la lectura de los sensores. El código utilizado se muestra en la siguiente figura.

Figura 46. Ventana de lectura – código

```
% --- Executes on button press in startRead.
function startRead_Callback(hObject, eventdata, handles)
%declaracion de variables globales
global s; %varibale de conexión serial
global flag; %variable de control
global temp1; %variable para obtener temp del sensor 1
global temp2; %variable para obtener temp del sensor 2
global temp3; %variable para obtener temp del sensor 3
set(handles.flag, 'Value', 1); %configuración de variable de control
flag = get(handles.flag, 'Value');
while flag == 1 %ciclo para lectura de sensores
    fprintf(s, 'R'); %envío de letra para comenzar la lectura de los
    sensores
    temp1 = fscanf(s, '%f'); %recepción dato del sensor 1
    pause(0.5);
    temp2 = fscanf(s, '%f'); %recepción dato del sensor 2
    pause(0.5);
    temp3 = fscanf(s, '%f'); %recepción dato del sensor 3
    pause(0.5);
    infoS1 = [1 temp1]; %asignación de los datos
    infoS2 = [2 temp2];
    infoS3 = [3 temp3];
    set(handles.dataTemp, 'data', vertcat(infoS1, infoS2, infoS3));
    %despliegue de lectura de temperaturas en la tabla de la ventana
    alarm(); %subprograma para revisar alarmas de temperatura
end
% --- Executes on button press in endRead.
function endRead_Callback(hObject, eventdata, handles)
global flag; %variable de control

set(handles.flag, 'Value' , 0); %configuración de variable de control
flag = get(handles.flag, 'Value');
% --- Executes on button press in saveTemp.
function saveTemp_Callback(hObject, eventdata, handles)
%declaracion de variables globales
global s; %varibale de conexión serial
global datos; %variable para almacenar datos
global temp1; %variable para obtener temp del sensor 1
global temp2; %variable para obtener temp del sensor 2
global temp3; %variable para obtener temp del sensor 3
%declaración de variables locales
contador_muestras = 1; %variable de control
numero_muestras = 25; %variable de control
```

Continuación figura 46.

```
datos = zeros(numero_muestras, 3); %declaración de matriz de datos
save = get(handles.saveTemp, 'Value');

if save == 1; %condición para almacenar datos
    while contador_muestras <= numero_muestras %ciclo para almacenar
temp.
        fprintf(s, 'R'); %envío de letra para comenzar la lectura de
los sensores
        temp1 = fscanf(s, '%f'); %recepción dato del sensor 1
        datos(contador_muestras, 1) = temp1; %almacenando dato del sensor
1
        pause(0.5);
        temp2 = fscanf(s, '%f'); %recepción dato del sensor 2
        datos(contador_muestras, 2) = temp2; %almacenando dato del sensor
2
        pause(0.5);
        temp3 = fscanf(s, '%f'); %recepción dato del sensor 3
        datos(contador_muestras, 3) = temp3; %almacenando dato del sensor
3
        pause(0.5);
        contador_muestras = contador_muestras + 1; %aumenta la variable
de control
        infoS1 = [1 temp1]; %asignación de datos
        infoS2 = [2 temp2];
        infoS3 = [3 temp3];
        set(handles.dataTemp, 'data', vertcat(infoS1, infoS2, infoS3));
        %despliegue de lectura de temperaturas en la tabla de la ventana
    end
end
set(handles.saveTemp, 'Value', 0); %configuración variable de control
mbox = msgbox('Datos guardados'); %ventana de aviso
uiwait(mbox);

%subprograma de verificación de alarmas
function alarm()
%declaración de variables globales
global alarmS1; %variable de valor de temp de alarma sensor 1
global alarmS2; %variable de valor de temp de alarma sensor 2
global alarmS3; %variable de valor de temp de alarma sensor 3
global temp1; %variable para obtener temp del sensor 1
global temp2; %variable para obtener temp del sensor 2
global temp3; %variable para obtener temp del sensor 3

%comparación de temperaturas leida vs alarma
if temp1 > alarmS1 %comparación para sensor 1
    msgbox('Sensor 1: temperatura por encima del límite');
elseif temp2 > alarmS2 %comparación para sensor 2
    msgbox('Sensor 2: temperatura por encima del límite');
elseif temp3 > alarmS3 %comparación para sensor 3
    msgbox('Sensor 3: Temperatura por encima del límite');
end
end
```

Fuente: elaboración propia, empleando MATLAB R2012b.

El sistema desarrollado permite la interacción del usuario final con el sistema que controla los sensores para realizar las mediciones. En resumen, la aplicación debe iniciar una conexión con el maestro para luego realizar las funciones establecidas, dentro de estas funciones está la lectura y configuración de los sensores y el manejo de los datos. El manejo de los datos puede ser mediante gráficos o la exportación de un archivo con los datos tomados por el usuario.

CONCLUSIONES

1. Se presenta un diseño de sistema versátil para la medición y despliegue de temperaturas con la finalidad que pueda ser implementado en distintas aplicaciones donde se requiera el monitoreo de la temperatura.
2. Las características de comunicación asíncrona y bidireccional mediante el uso de un solo bus serial y la lectura de un código único presente en cada dispositivo, convierten al protocolo 1-Wire en un protocolo útil al momento de realizar integraciones con varios dispositivos.
3. Las características de medición de temperatura, configuración de alarmas y resolución, modo de alimentación y compatibilidad 1-Wire que presenta el sensor DS18B20, permite ser utilizado en distintas configuraciones y aplicaciones.
4. El sistema propuesto permite al usuario el manejo de la lectura y despliegue de temperaturas para un fluido entre $-55\text{ }^{\circ}\text{C}$ y $125\text{ }^{\circ}\text{C}$.

RECOMENDACIONES

1. Considerar el desarrollo de proyectos de domótica, el uso del protocolo 1-Wire para su desarrollo por la versatilidad presentada en la forma de comunicación, ya sea para controles de acceso, sistemas de alarma o controles de temperatura.
2. Tomar en cuenta que es factible una conexión inalámbrica entre la aplicación y el sistema de control, ya sea mediante módulos RF o en segundo caso módulos *bluetooth*.
3. Definir cuáles son los requerimientos y funciones que va a utilizar el usuario previo a realizar la interfaz gráfica del sistema.

BIBLIOGRAFÍA

1. Datasheet: *Arduino UNO*. [en línea]. <<https://www.arduino.cc/en/Main/ArduinoBoardUno>>. [Consulta: febrero 2017].
2. Datasheet: *DS18B20, Programmable Resolution 1-Wire Digital Thermometer*. [en línea]. <<https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>>. [Consulta: febrero 2017].
3. GILAT, Amos. *Matlab Una introducción con ejemplos prácticos*. 2a ed. España: Reverté, S.A., 2006. 331 p.
4. Manual: *1-Wire® Tutorial*. [en línea]. <<https://www.maximintegrated.com/en/products/1-wire/flash/overview/index.cfm>> [Consulta: octubre 2016].
5. MOORE, Holly. *MATLAB® para ingenieros*. 1a ed. México: Pearson Educación, 2007. 624 p.

APÉNDICES

Apéndice 1. Programa del Arduino Uno

```
// Librerías
#include <OneWire.h> //Librería de protocolo 1-Wire
#include <DallasTemperature.h> //Librería del sensor DS18B20

// Cable de datos en el puerto 2 del Arduino
#define ONE_WIRE_BUS 2

// Creando una instancia 1-Wire para comunicar los dispositivos
OneWire oneWire(ONE_WIRE_BUS);

// Pasando la referencia 1-Wire a la librería del sensor
DallasTemperature sensors(&oneWire);

// Arreglos para almacenar las direcciones de los dispositivos
DeviceAddress Termometro1, Termometro2, Termometro3;

// Declaración de variables
int mode = -1; //variable para la comunicación Serial
int num; //variable para el número de sensores en línea
float tempC; //variable para almacenar la temperatura

// Configuración del Arduino
void setup(void)
{
  // Comenzar la comunicación Serial
  Serial.begin(9600);

  // Verificar comunicación Serial
  Serial.println('a'); //enviar en el bus serial la letra 'a'
  char a = 'b'; //declarar y asignar letra 'b' a la variable a
  while (a != 'a'){ //ciclo para esperar la conexión de la PC
```

Continuación apéndice 1.

```
    a = Serial.read(); //lee el bus serial
  }

  // Inicialización de los sensores
  sensors.begin();

  // Configurando la resolución de los sensores a 9 bits
  sensors.setResolution(Termometro1, 9);
  sensors.setResolution(Termometro2, 9);
  sensors.setResolution(Termometro3, 9);

  // Configurando alarmas a los sensores
  sensors.setHighAlarmTemp(Termometro1, 75);
  sensors.setHighAlarmTemp(Termometro2, 80);
  sensors.setLowAlarmTemp(Termometro3, 5);
}

void loop(void)
{
  if (Serial.available() >0){ //queda a la espera de que la PC envíe por el bus serial
    mode = Serial.read(); //la lectura del puerto se asigna en la variable mode
    switch (mode){
      case 'R': //condición si se recibe la letra 'R'
        getTempCent(); //función de lectura de temperatura
        break;
      case 'I': //condición si se recibe la letra 'I'
        getInfo(); //función de información de sensores
        break;
      case 'C': //condición si se recibe la letra 'C'
        getSensors(); //función de verificación de sensores
        break;
    }
  }
}
```

Continuación apéndice 1.

```
// Función de lectura de temperaturas
void getTempCent(){
    sensors.requestTemperatures(); //solicitar temperatura a los sensores en linea
    tempC = sensors.getTempCByIndex(0); //leyendo temperatura del sensor 0
    Serial.println(tempC); //enviando temperatura a la PC
    delay(500);
    tempC = sensors.getTempCByIndex(1); //leyendo temperatura del sensor 1
    Serial.println(tempC); //enviando temperatura a la PCC
    delay(500);
    tempC = sensors.getTempCByIndex(2); //leyendo temperatura del sensor 2
    Serial.println(tempC); //enviando temperatura a la PC
    delay(500);
}

// Función para imprimir el ID del sensor
void printAddress(DeviceAddress deviceAddress)
{
    for (uint8_t i = 0; i < 8; i++)
    {
        if (deviceAddress[i] < 16) Serial.print("0");
        Serial.print(deviceAddress[i], HEX);
    }
}

// Función para obtener la información del sensor
void getInfo(){
    while (1){
        if (Serial.available() >0){ //queda a la espera de que la PC envíe por el bus serial
            char s = Serial.read(); //la lectura del puerto se asigna en la variable s
            switch (s){
                case '1': //envío de información del sensor 1 cuando se reciba el número '1'
                    if (sensors.isParasitePowerMode()){ //revisión si está en poder parásito
                        Serial.println("ON");
                    }
                else Serial.println("OFF");
            }

            if (!sensors.getAddress(Termometro1, 0)){
```


Continuación apéndice 1.

```
        Serial.println("Unable to find address for Device 0");
    }
    printAddress(Termometro1);
    break;

case '2': //envío de información del sensor 2 cuando se reciba el número '2'
    if (sensors.isParasitePowerMode()){ //revisión si está en poder parásito
        Serial.println("ON");
    }
    else Serial.println("OFF");

    if (!sensors.getAddress(Termometro2, 1)){
        Serial.println("Unable to find address for Device 0");
    }
    printAddress(Termometro2);
    break;

case '3': //envío de información del sensor 3 cuando se reciba el número '3'
    if (sensors.isParasitePowerMode()){ //revisión si está en poder parásito
        Serial.println("ON");
    }
    else Serial.println("OFF");

    if (!sensors.getAddress(Termometro3, 1)){
        Serial.println("Unable to find address for Device 0");
    }
    printAddress(Termometro3);
    break;
}
}}}

// Función para verificar número de sensores conectados.
void getSensors(){
    num = Serial.println(sensors.getDeviceCount(), DEC);
}
```

Fuente: elaboración propia, empleando Arduino IDE.

Apéndice 2. **Arduino Uno**



Fuente: elaboración propia.

Apéndice 3. **Sensor DS18B20**



Fuente: elaboración propia.

