ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"TELECONTROL DE INVERNADERO"

INFORME DE MATERIA DE GRADUACIÓN:

Previo a la obtención del título de:

INGENIERO EN TELEMÁTICA

Presentada por:

GUILLÉN FLORES JORGE ANDRÉS

SANTOS ALCIVAR ROSENDO ANTONIO

GUAYAQUIL – ECUADOR

AÑO: 2015

AGRADECIMIENTO

Al Ing. José Muñoz Arcentales, por su apoyo con las pruebas del prototipo y por permitirnos el ingreso al Laboratorio Pedro Carlo Paredes. Al Sr. Carlos Vaca por su valiosa colaboración en el diseño de la página Web de nuestro proyecto. Al Sr. Brick Reyes por su colaboración en la adaptación de la página Web para dispositivos móviles.

DEDICATORIAS

A Dios, a mis padres que siempre me apoyaron en el transcurso de la carrera y en cada proyecto o meta por cumplir, a mi primo el Ing. José Muñoz Arcentales por ayudar con sus conocimientos en la elaboración del proyecto, a mi tío el Ing. Willian Arcentales por estar siempre presto con su ayuda.

Jorge Andrés Guillén Flores

A Dios, que nunca me dejado vencer a mi familia y amigos que a siempre ha estado a mi lado alentando en mis proyectos de vida.

Rosendo Antonio Santos Alcívar

TRIBUNAL DE SUSTENTACIÓN

_____

MSc. Marcos Millán

PROFESOR DE LA MATERIA DE GRADUACIÓN

_____

MSc. Patricia Chávez

PROFESORA DELEGADA POR LA UNIDAD ACADEMICA

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este Informe, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la Escuela Superior Politécnica del Litoral”.

(Reglamento de Graduación de la ESPOL)

_____

GUILLÉN FLORES JORGE ANDRÉS

_____

SANTOS ALCIVAR ROSENDO ANTONIO

RESUMEN

El “Telecontrol de Invernadero” permite acceder de manera remota al invernadero automatizado, visualizar datos que envían los sensores y controlar el microclima dentro del invernadero, brindando la posibilidad de activar, de acuerdo a las condiciones ambientales que se presenten, las diferentes opciones que ayuden a mantener ese ambiente ideal donde las plantas se puedan desarrollar en su mejor esplendor. Opciones tales como: ventilación, refrigeración, riego y luz artificial, son las opciones que actuarán sobre el invernadero. Se brinda adicionalmente la posibilidad de monitorear las variables de manera local mediante una pantalla LCD y a su vez ejecutar acciones sobre el invernadero desde el mismo lugar. La metodología del árbol de problemas junto a la metodología del desarrollo de software fueron empleados para el desarrollo de este proyecto, donde su análisis y utilización se verá reflejado en los capítulos siguientes. El desarrollo e implementación del prototipo sirve de propuesta a la agricultura tradicional usando software y hardware libre, logrando con una menor inversión aumentar la producción y la calidad de las plantas sembradas dentro del invernadero.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS Y SIMBOLOGÍA

LCD	Liquid Cristal Display	Pantalla de cristal líquido.
MAC	Media Access Control	Control de acceso al medio.
RIDO	------	Rango Ideal de Desarrollo Óptimo
TCP	Transmission Control Protocol	Protocolo de control de transmisión
UDP	User Datagram Protocol	Protocolo de datagramas de usuario
USB	Universal Serial Bus	Bus serial universal

GLOSARIO

MAC Es la identificación o dirección física de un equipo, siendo esta única por cada equipo.

Fisiología Es una parte de la botánica que identifica el funcionamiento de los órganos y tejido vegetales.

Hardware libre Es el hardware que permite desarrollar proyectos a un bajo costo.

Dirección IP Etiqueta numérica o lógica a una interfaz de un dispositivo dentro de una red de protocolo IP.

Microclima Es el espacio físico y cerrado donde se lleva un control climático para el desarrollo de las plantas.

I2C Protocolo de comunicación serial y sincrónica basado en dos líneas de señal que son la de datos y la de reloj.

Software libre Software empleado para ser utilizado en el hardware libre y aplicaciones Web.

TCP Protocolo de comunicación la cual asegura la correcta comunicación entre una red y otra, además de la fiabilidad y confirmación en el envío y recepción en los datos.

Telecontrol Es el envío de indicaciones u órdenes de forma remota mediante un medio de transmisión hacia un dispositivo.

UDP Protocolo basado en el intercambio de paquetes llamados Datagramas, este permite el envío de los paquetes sin antes haber establecido una conexión entre una red y otra; además no existe confirmación ni fiabilidad en la recepción y envió de los mismos.

USB Estándar de conexión de periféricos en dispositivos electrónicos.

INTRODUCCIÓN

Debido a los diferentes cambios climáticos y la búsqueda del desarrollo tecnológico en el agro del país, la meta de este prototipo es poder brindar un lugar donde se pueda mantener las condiciones ideales ambientales de una planta para que pueda desarrollarse de la mejor manera utilizando software y hardware libre, siendo dirigido de manera automática por un controlador que analiza las diferentes variables del invernadero y activa diferentes opciones que ayuden a mantener ese clima ideal de crecimiento de la planta.

Desde cualquier ubicación y momento, mediante un computador, teléfono inteligente o tableta con servicio de internet, se puede analizar los valores enviados por los sensores del invernadero, hacia el servidor web, para que de ésta manera la persona tenga la decisión de manipular ese ambiente artificial haciendo que el microclima sea el adecuado, de acuerdo a los valores que se presenten.

Para la realización de este proyecto se utilizó una maqueta en la que se representa un invernadero a escala donde se pueda monitorear y controlar las

diferentes variables y actuadores sobre el invernadero. De esta manera la interacción hombre maquina es a través de una interfaz web, recordando que siempre se tiene la opción de actuar directamente en el mismo sitio donde se ubica el invernadero.

Este proyecto está organizado en cuatro capítulos. En el primer capítulo se indica los aspectos generales basados del proyecto, una breve introducción, los antecedentes, objetivos generales y específicos del proyecto. El capítulo 2, nos indica marco teórico en que se basa el prototipo, importancia del invernadero en la agricultura, materiales utilizados para la elaboración del prototipo y breve descripción de los mismos, que metodología se empleó para su desarrollo. El capítulo 3 presenta la descripción del proyecto, estrategia utilizada y el diagrama de bloques del sistema. Finalmente, el capítulo 4 muestra detalladamente el desarrollo de esta alternativa y como realiza cada uno de los procesos y su importante influencia dentro del invernadero. Adicionalmente en este documento se muestran gráficos, figuras y términos empleados para su fácil comprensión y entendimiento. Finalizando con la bibliografía, anexos y demás que ayudará y justificará los temas tratados en el documento.

CAPÍTULO 1

1. GENERALIDADES

En el presente capítulo revisaremos los antecedentes, justificación, metodología y resultados esperados del proyecto a fin de presentar el marco que ampara la realización del presente trabajo.

1.1 ANTECEDENTES

El aumento en las emisiones de gases, el derretimiento de glaciares, aumento de precipitaciones y los eventos meteorológicos, influyen en el acelerado cambio climático reflejándose en problemas con la ganadería y agricultura, afectando de una u otra manera la escases de alimentos en el futuro.

En la actualidad productores basándose en sus conocimientos empíricos sobre cultivo tradicional, hacen que este tipo de agricultura sea vea constantemente afectada ante las variaciones climáticas, siendo vulnerable por cambios de temperatura, humedad, luminosidad entre otros.

Las condiciones tales como cantidad de lluvia o temperatura ambiental afectan a los sembríos produciendo un deficiente crecimiento de las plantas, frutos o vegetales pequeños y la muerte de la planta. En la Figura 1.1 se muestra como afectó la sequía en la parroquia Lojana de

Zapotillo donde no pudo llegar a madurar el sembrío de maíz y por ende no produjo el fruto requerido. [1]

Figura 1.1 - Sembrío de maíz durante sequía. [1]

Entonces se pierde la inversión realizada por el agricultor en la siembra, debido a la ausencia de tecnología y esto desencadena pérdidas monetarias y escases en los alimentos y un sobreprecio por su ausencia en el mercado.

En vista de esta problemática, se propone el desarrollo del prototipo “Telecontrol de invernadero” para crear y mantener un espacio con las condiciones ambientales adecuadas para el cultivo de plantas específicas; monitoreando la luz artificial, temperatura y humedad de

forma remota en cualquier momento y lugar en el que se posea acceso a internet, además de tener la posibilidad de hacerlo de forma local.

1.2 JUSTIFICACIÓN

El desarrollo del proyecto es proponer una alternativa a la agricultura tradicional utilizando hardware y software libre de bajo costo, dando la facilidad de monitorear y controlar el desarrollo de las plantas del invernadero como si el usuario se encontrara físicamente en él, mejorando tiempos de atención al invernadero de una forma oportuna. Beneficiando a futuro el tener mayor producción en menor tiempo con una mejor calidad.

1.3 OBJETIVO GENERAL

El objetivo central es diseñar e implementar un prototipo de telecontrol de invernadero, que permita crear un microclima adecuado para mejorar el desarrollo de las plantas utilizando hardware y software libre.

1.4 OBJETIVOS ESPECÍFICOS

De acuerdo a nuestro esquema planteado tenemos los siguientes objetivos específicos:

- Diseñar el prototipo que nos permita actuar de manera remota y local.
- Programar el software libre del controlador del sistema.
- Configurar el servidor web y base de datos que permita establecer la comunicación con el equipo controlador.
- Implementar el hardware que nos permita interactuar con el invernadero.

1.5 RESULTADOS ESPERADOS

En la primera etapa del desarrollo del proyecto se espera lograr la correcta comunicación con el invernadero desde diferentes dispositivos móviles y poder interactuar con los actuadores del mismo a fin de adecuar su microclima para el mejor desarrollo de la planta.

Una vez concluido el proyecto, se espera que el prototipo sea del interés de pequeñas y medianas empresas, lo que podría significar su amplia aplicación en el sector agrícola con lo cual se podrá reducir costos de operación y por ende el costo de los alimentos, disminuir el índice de personas afectadas por causa de los pesticidas, mejorar la producción de plantas de todo tipo y disminuir cantidad de plantaciones perdidas.

1.6 ALCANCE

El proyecto de “Telecontrol de invernadero” tiene como alcance el envío de datos, transmisión de datos y dirigir acciones localmente y remotamente.

Envío de datos desde cada sensor al hardware libre, que este último a su vez funciona como controlador del sistema e intérprete de los datos.

Transmisión de datos desde el controlador hacia el servidor web y a la base de datos la información extraída de cada sensor.

Dirigir acciones localmente o desde la interfaz web hacia el controlador para la activación de la bomba de agua, ventilación, refrigeración e iluminación artificial.

1.7 LIMITACIONES

Para comprobar que el prototipo es funcional se utilizó como producto de cultivo al rábano (Raphanus Sativus) que es de rápido crecimiento (quince días) y es fácilmente adaptable a diferentes tipos de suelo. Por este motivo, nuestro sistema está optimizado para este tipo de cultivo pero bajo el mismo esquema se puede utilizar cualquier tipo de planta identificando los valores ideales para el desarrollo óptimo de la planta. [2] El prototipo de “Telecontrol de Invernadero” se enfoca en los factores descritos anteriormente, como lo son la temperatura, humedad y luminosidad dejando para una posterior versión del prototipo las mediciones de PH del suelo y manejo de pesticidas.

CAPÍTULO 2

2. MARCO TEÓRICO

Por definición, un invernadero es un lugar cerrado, estático y accesible a pie, fue creado debido a los constantes cambios climáticos logrando adecuar esas condiciones ambientales a un rango óptimo para el correcto desarrollo de las plantas. En la Figura 2.1 podemos apreciar el interior del invernadero de KewGardens en Londres. [3]

Figura 2.1 - Vista panorámica del interior de un invernadero en los KewGardens de Londres. [3]

2.1 IMPORTANCIA

La importancia de un invernadero es poder asegurar la producción sin que pueda ser dañada por el estado meteorológico de la zona, donde existen condiciones como viento, altas y bajas temperaturas, granizo, lluvias torrenciales y otros agentes externos como plagas o animales. Debido a ello, se ha buscado mecanismos para desarrollar y mejorar el estado actual de los invernaderos, utilizando una mejor tecnología que pueda contrarrestar los factores ambientales debido a los cambios meteorológicos que son cada vez más inestables.

Gracias a la tecnología y a los resultados de la investigación biológica en áreas como fisiología vegetal que nos brinda los parámetros y rangos que deben encontrarse estas variables para que determinada planta crezca en su mejor esplendor.

De acuerdo a lo indicado podemos establecer las siguientes ventajas: frutos fuera de época, mejora de la calidad de los productos, tener más cultivos en todo el año, control del riego y fertilizantes (ahorro), adecuada emisión de pesticidas.

Con la propuesta “Telecontrol de invernadero” se tendrá las siguientes ventajas:

- Control del invernadero las 24 horas del día.
- Manejo de actuadores de acuerdo a las diferentes variables que afecten al invernadero (microclima).
- Disminución de personal para operar el invernadero (ahorro).
- Acceso desde cualquier lugar del mundo con acceso a internet.

- Mejora en la atención a fallos de operación gracias a los sensores implementados en el proceso, lo que nos indicara una atención inmediata para verificar el estado de ese segmento o nodo dentro del invernadero.

Gracias a esto se logra acondicionar un microclima deseado controlando la temperatura, humedad y luminosidad que nos favorecen al buen crecimiento de la planta.

2.2 MATERIAL UTILIZADO

Para la realización del presente proyecto se emplearon sensores, actuadores, enrutador, tarjeta Arduino Ethernet Shield y tarjeta Arduino Uno, relés, interruptores, borneras, cables, luces piloto. La información completa de los dispositivos se encuentra en los anexos como hoja de datos.

El sensor DSB1820 permite medir temperaturas desde los -55°C a 125°C y es el encargado de indicar la temperatura interna del Invernadero que luego será enviado al controlador del sistema. En la Figura 2.2 se puede apreciar que el sensor cuenta con tres pines: V (fuente), G (tierra) y S (dato). [4]

Figura 2.2 - Sensor de temperatura DSB1820. [5]

El sensor de humedad puede trabajar entre los 3,3V a 5V su rango de medición es desde los 0 hasta 1024, donde el dato 0 representa a que está totalmente sumergido en agua y 1024 completamente seco. Posee una sonda para pinchar en el suelo y un circuito transductor para preparar la señal. En la Figura 2.3 se muestra en escala el sensor de humedad empleado en el proyecto. [6]

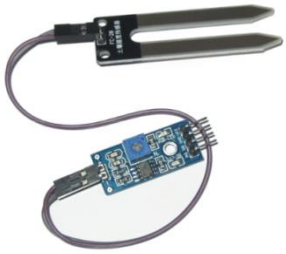


Figura 2.3 - Sensor de humedad del suelo (fc-28). [6]

El sensor de luminosidad o resistor dependiente de la luz mostrado en la Figura 2.4, es el encargado de recolectar los datos de la luminosidad que incide en el ambiente. Los datos que recolecta son valores entre 0 y 1023, donde 0 representa a totalmente oscuro y 1023 como la luminosidad más alta. [7]

Figura 2.4 - Sensor de luminosidad. [8]

El Arduino Uno es una placa electrónica la cual posee 14 pines digitales y 6 analógicos de entrada/salida, además posee un conector USB y un

conector de alimentación, los cuales se aprecian desde la vista superior de la placa como se muestra en la Figura 2.5. [9]

Figura 2.5 - Tarjeta Arduino Uno. [9]

El Arduino Ethernet Shield, es una tarjeta que se acopla al Arduino Uno, este realizando una secuencia de códigos interpretado por el Arduino, le permite conectarse al internet, la tarjeta se puede observar en la Figura 2.6. [10]

Figura 2.6 - Tarjeta Arduino Ethernet Shield. [10]

La pantalla LCD de cristal líquido y de fondo azul, la cual se puede acoplar con un módulo I2C para que la conexión sea tan solo de 4 pines que son tierra, fuente, SDA (línea de datos) y SCL (señal de reloj); y la podemos observar en la Figura 2.7, posee la capacidad de mostrar 20 caracteres, en 4 filas y tiene una alimentación de 5V. [11].

Figura 2.7 - Pantalla LCD con módulo I2C. [12]

Los Actuadores son dispositivos que interactúan con el sistema y son habilitados o deshabilitados de acuerdo a las condiciones ambientales que se encuentren presentes, y adecuando el microclima a su rango ideal, ejemplo: luz artificial, ver Figura 2.8. [13]

Figura 2.8 - Luz artificial dentro de invernadero. [13]

Un servidor web es un equipo donde se encuentra alojada una aplicación web que interactúa con datos a través de la red, guardando o extrayendo información de una base de datos Ejemplo: Apache.

Base de datos es el lugar donde se almacena toda la información necesaria de manera organizada para luego acceder a ella y depositar o extraer la información que se requiera. En este caso se utilizó MySQL por su facilidad y ambiente amigable para el programador.

El enrutador como su nombre lo indica es el equipo de comunicación que interconecta las computadoras en una red enviando paquetes y buscando las mejores rutas para establecer la comunicación. Para el prototipo se utilizó el equipo TPLINK WR841ND. [14]

Como base de conocimiento mencionamos los conceptos de comunicaciones que fueron necesarios para la elaboración del proyecto. La tarjeta Ethernet Shield de Arduino está basada en el chip Ethernet Wiznet W5100 que permite establecer comunicaciones sobre red IP mediante TCP y UDP.

Para las diferentes necesidades en el desarrollo de las aplicaciones, fueron empleadas diferentes herramientas que de una u otra manera fueron elegidas de manera personal por los integrantes del proyecto por su facilidad y mejor dominio.

- Aplicación Web, Net Framework.
- Base de datos phpmyAdmin.
- IDE Arduino para el Arduino Uno.

Cabe mencionar que para el proyecto se adicionó botones de marcha y para cuando el proceso es manipulado de forma local. Este será analizado más a fondo en el siguiente capítulo.

2.3 METODOLOGÍA EMPLEADA

Para la resolución del problema nos basamos en la metodología de elaboración de árbol de problemas, ver Figura 2.9, donde podemos identificar cuáles son los factores que inciden y afectan en el normal desarrollo de las plantas. [15]

Figura 2.9 - Árbol de problemas del sistema.

Estas son las diferentes causas para que las condiciones ambientales afecten a las plantas:

- Suelo no fértil cuando la tierra no ha sido tratada (abono).
- Altas y bajas temperaturas.- terrible calor o frio por la capa de ozono en deterioro.
- Exceso de lluvias.- época invernal en el cual las precipitaciones varían (Ejemplo: época del niño). [16]
- Ausencia de lluvia.- atraso de la época que llegan las precipitaciones.
- Vientos huracanados condiciones meteorológicas que también afectan a los sembríos.

Todos ellos en conjunto afectando a la vida agrícola, causando efectos como la muerte de las plantas, falta de crecimiento y desarrollo pleno, frutos secos o pequeños.

2.4 ANÁLISIS DE LA SOLUCIÓN

En parte en la realización del prototipo nos basaremos en la metodología de desarrollo de software dándole el enfoque en el Modelo de Prototipos del cual podemos observar el diagrama en la Figura 2.10. [17]

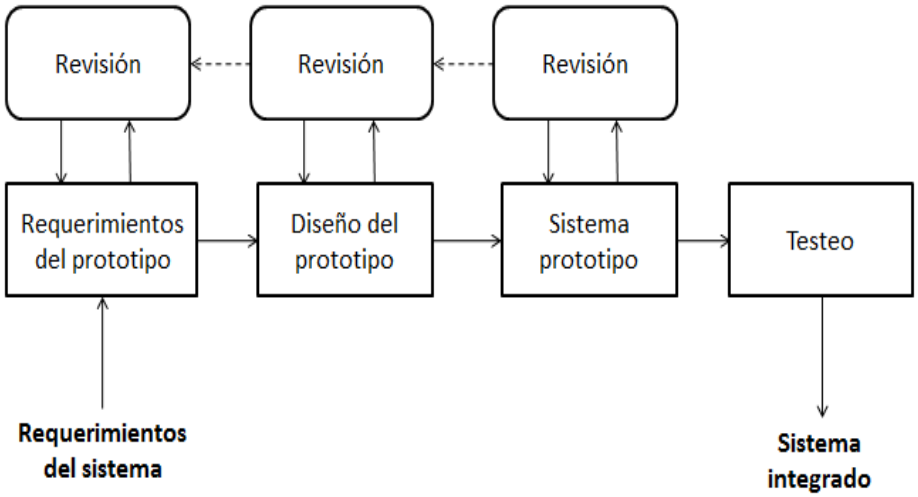


Figura 2.10 - Diagrama de modelo de prototipos

Requerimientos del sistema: medir temperatura, humedad relativa, luminosidad, pH del suelo (medida de alcalinidad del suelo) como entrada, y como salida, activaciones de calefacción refrigeración, luz artificial, ventilación, riego, abono y nutrientes manejo de pesticidas.

Requerimientos del prototipo: medir temperatura, humedad relativa, luminosidad como entrada y de salida activación de refrigeración, luz artificial, ventilación, riego.

Diseño del prototipo: Programación del Controlador, Programación de la Base de Datos en mySQL, Configuración del Servidor Web que aloja nuestra interfaz web del Invernadero, Programación de la Interfaz web donde los usuarios pueden interactuar con el Invernadero.

Sistema Prototipo: Va a contener lo antes mencionado en el diseño del prototipo con mejoras en el controlador, interfaz web y base de datos, que permitan el correcto uso y manejo del sistema.

CAPÍTULO 3

3. DESCRIPCIÓN DEL PROYECTO

Un invernadero es un espacio con un microclima adecuado para el óptimo desarrollo de las plantas, recibiendo constantemente datos de temperatura, humedad y luminosidad que hacen que ese ambiente artificial, sea controlado y ajustado a las necesidades que se requieran dentro de él.

Adecuando ese microclima permite alcanzar alta productividad, a bajo costo, en menor tiempo, sin daño ambiental, protegiéndose de las lluvias, el granizo, las heladas, o los excesos de viento que pudieran perjudicar nuestro cultivo.

Cuando una planta no tiene un alto rendimiento seguramente ha tenido exceso de humedad o carecido de ella, expuesta a altas o bajas temperaturas, carencia de luz, en definitiva un problema que no pudo ser atendido debido a las condiciones ambientales que surgieron en su momento.

El telecontrol de invernadero, brinda la opción de poder interactuar de manera local o remota con el controlador teniendo las opciones de habilitar o deshabilitar de acuerdo a las diferentes necesidades que se requieran dentro del ambiente artificial creado, provocando así un control completo del microclima del invernadero.

3.1 FUNCIONAMIENTO DEL CONTROL CLIMÁTICO

Para el control del microclima del invernadero tendremos dos modos de habilitación o des habilitación: modo automático el cual es retroalimentado y consta de cinco partes, las cuales podemos observar en la Figura 3.1; para el modo manual una persona (usuario) es quien habilite o deshabilite los actuadores de acuerdo a los valores en la página web o en la pantalla LCD. (Manera remota y local específicamente) y podemos observar en la Figura 3.3. Los Diagramas de Bloques para el modo automático y manual se muestran en las Figuras 3.2 y 3.4, respectivamente.

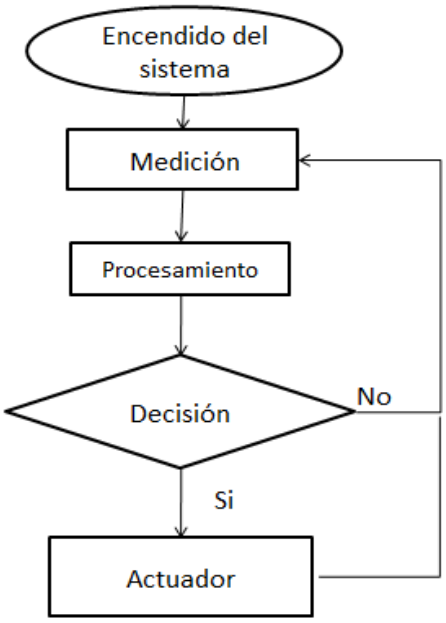


Figura 3.1 - Funcionamiento del control climático modo automático

Como primera instancia se obtiene la medición de la variable (Ejemplo: Temperatura), luego se realiza el procesamiento de los datos enviados por los sensores y recibidos en el controlador, posteriormente el controlador toma una decisión que indica la ejecución de los actuadores sobre el invernadero para finalmente ejecutar la acción tomada sobre el actuador logrando de esta manera mantener el microclima en el rango deseado. (Ejemplo: refrigeración, ventilación, riego, luz).

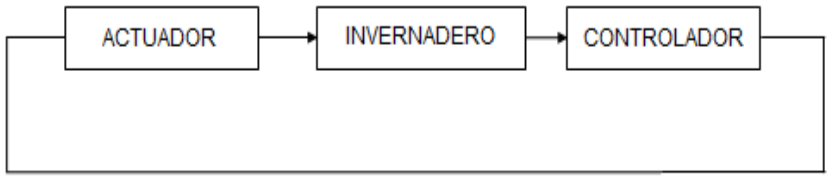


Figura 3.2 - Diagrama de bloques del sistema modo automático.

Como primer paso se realiza la medición de la variable (Ejemplo: Humedad), por consiguiente se procesan los datos enviados por los sensores y recibidos en el controlador, luego, en este modo el usuario es quien toma la decisión de acuerdo a los valores presentados para activar o desactivar los actuadores. Finalmente se ejecuta la acción sobre los actuadores para lograr el microclima deseado (Ejemplo: refrigeración, ventilación, riego, luz).

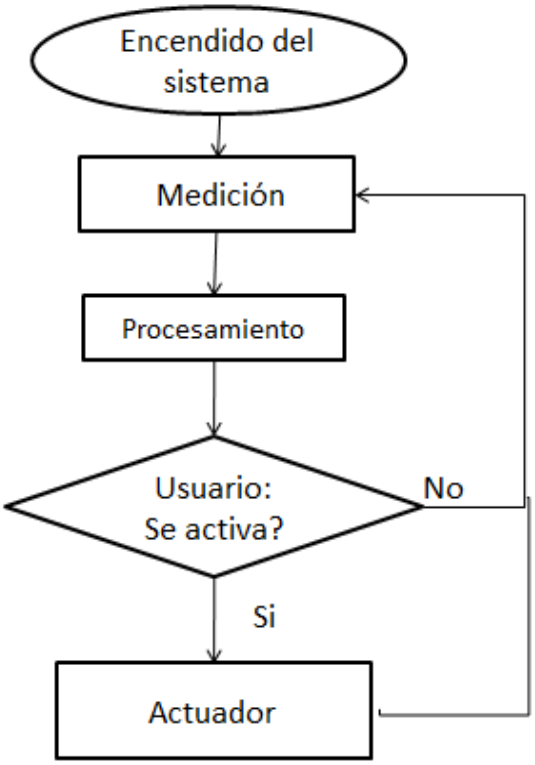


Figura 3.3 - Funcionamiento del control climático modo manual

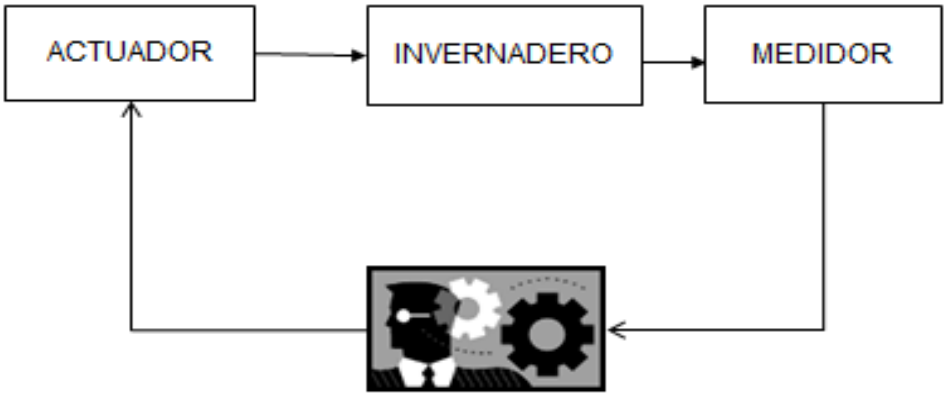


Figura 3.4 - Diagrama de bloques del sistema modo automático.

Dentro del invernadero se debe analizar todas las variables simultáneamente, por lo tanto, al controlador se envían los datos obtenidos por los sensores de manera periódica, en modo remoto cada 3 minutos y en modo local cada 4 segundos, este a su vez al recibir los datos, comienza la etapa de comparación de datos, con los valores máximos y mínimos establecidos para que la planta se encuentre en su RIDO. Luego de analizar el dato recibido por el sensor, se realiza una toma de decisiones donde se deberá elegir la activación o no del actuador que adecue ese indicador (luminosidad, humedad, temperatura) a los parámetros deseados.

3.2 COSTOS DE IMPLEMENTACIÓN

Se muestra la Tabla 1 con los costos de implementación en maqueta obtenidos de [18], [19], [20], [21], [22] y la Tabla 2 de los costos que conlleva la implementación del sistema en la industria para un invernadero de 4 metros de largo y 3 metros de ancho basados en [18], [20], [21], [22].

Tabla 1 - Inversión para Diseño de Maqueta.

Descripción	Cantidad	Precio unitario (USD)	Precio Total (USD)
Invernadero a Escala	1	$ 60,00	$ 60,00
Base de Cartón Prensado	1	$ 12,00	$ 12,00
Plástico cobertor	1	$ 50,00	$ 50,00
Panel de Control en Acrílico	1	$ 35,00	$ 35,00
Barra Metálica	1	$ 3,00	$ 3,00
Bomba de agua	1	$ 18,00	$ 18,00
Recipiente Contenedor de Agua	1	$ 10,00	$ 10,00
Base para Desfogue de Agua	1	$ 80,00	$ 80,00
Foco LED	1	$ 18,00	$ 18,00
Ventiladores de PC	3	$ 12,00	$ 36,00
Refrigerantes	6	$ 4,00	$ 24,00
Manguera para Riego	2 m	$ 4,00 por metro	$ 8,00
Manguera para Refrigeración	1 m	$ 10,00 por metro	$ 10,00
Chova	1 rollo	$ 7,00	$ 7,00
Tierra de Sembrado	1 funda	$ 2,00	$ 2,00
Plantas Artificiales	2 paquetes	$ 5,00 por paquete	$ 10,00
Abrazadera	1	$ 1,00	$ 1,00
Codo PVC	1	$ 1,50	$ 1,50
Sensor de Luminosidad LDR	1	$ 3,00	$ 3,00
Sensor de Temperatura DSB1820	1	$ 5,00	$ 5,00
Sensor de Humedad Relativa FC-28	1	$ 5,00	$ 5,00
Extensión	1	$ 3,00	$ 3,00
Tomacorriente	1	$ 1,00	$ 1,00
Enchufe	2	$ 0,50	$ 1,00
Módulos de Relé	2	$ 17,00	$ 34,00
Placas Electrónicas	2	$ 18,00	$ 36,00
Arduino Uno	1	$ 29,00	$ 29,00
Arduino Ethernet Shield	1	$ 20,00	$ 20,00
Pantalla LCD 20x4	1	$ 18,00	$ 18,00
Módulo I2C para LCD 20x4	1	$ 8,00	$ 8,00
Caja de Paro/Marcha	1	$ 3,00	$ 3,00
Luces Indicadoras	2	$ 2,00	$ 4,00
Interruptores	5	$ 0,35	$ 1,05
LED de 10mm	5	$ 0,80	$ 4,00

Tabla 1 - Inversión para Diseño de Maqueta (continuación).

Descripción	Cantidad	Precio unitario (USD)	Precio Total (USD)
Fuente de 12V	1	$ 10,00	$ 10,00
Canaleta	1	$ 5,00	$ 5,00
Materiales Varios (borneras, terminales, cables de varias medidas, tornillos, etc.)		$ 80,00	$ 80,00
Herramientas Varias (estilete, cautín, martillo, playo, destornilladores, cortadora, serrucho, sierra, etc.)		$ 30,00	$ 30,00
Cable directo para conexión Ethernet	5m	$ 2,00 por metro	$ 10,00
Cable USB	1	$ 5,00	$ 5,00
Computadora (Servidor)	1	$ 800	$ 800
Enrutador	1	$ 40,00	$ 40,00
Costo total de elaboración de la maqueta.			**$ 1540.55**

Tabla 2 - Inversión para Implementación de Sistema.

Descripción	Cantidad	Precio unitario (USD)	Precio Total (USD)
Materiales a comprar una sola vez por Invernadero			
Servicio de Hosting	1 año	$ 150,00	$ 150,00
Caja de Paro/Marcha	1	$ 10,00	$ 10,00
Sensor de Temperatura PT100	2	$ 15,00	$ 30,00
Sensor de Luminosidad LDR	1	$ 3,00	$ 3,00
Panel Eléctrico	1	$ 100,00	$ 100,00
Diseño e Instalación	-------	-------	$ 1000,00
Página web para telecontrol	-------	-------	$ 1200,00
Costo Número 1			**$ 2493,00**
Materiales por 4 Nodos dentro del Invernadero			
Cable flexible 22	200 m	$ 0,35 por metro	$ 70,00

Tabla 2 - Inversión para Implementación de Sistema (continuación).

Descripción	Cantidad	Precio unitario (USD)	Precio Total (USD)
Cable directo para conexión Ethernet	40 m	$ 2,00 por metro	$ 80,00
Arduino Uno	3	$ 29,00	$ 87,00
Arduino Ethernet Shield	3	$ 20,00	$ 60,00
Relés de 5V de estado sólido	7	$ 15,00	$ 105,00
Transistores bc337	7	$ 0,50	$ 3,50
Pulsadores de botón rasante	9	$ 5,00	$ 45,00
Luces Indicadoras	11	$ 5,00	$ 55,00
Riel DIN	5 m	$ 1,25 por metro	$ 6,25
Borneras de DIN	70	$ 0,30	$ 21,00
Sensor de Humedad Relativa FC-28	9	$ 5,00	$ 45,00
Pantalla LCD 20x4	1	$ 18,00	$18,00
Electroválvulas	9	$ 25,00	$ 225,00
Módulo I2C para LCD 20x4	1	$ 8,00	$ 32,00
Costo Número 2			**$ 828,75**
Costo total del sistema implementado			**$ 3321,75**

CAPÍTULO 4

4. FUNCIONAMIENTO DEL SISTEMA

Este sistema tiene cinco procesos: Proceso de encendido de luz artificial, Proceso de ventilación, Proceso de riego, Proceso de refrigeración las que serán activadas por los interruptores observados en la Figura 4.1,y el paro de emergencia controlado por los botones de paro y marcha mostrados en la Figura 4.2.

El proceso de encendido de luz artificial ayuda a la planta a realizar la fotosíntesis emitiendo los rayos azules de la luz hacia la planta que ayuda al crecimiento de la misma. El proceso de riego brinda al suelo la humedad adecuada en donde se encuentra la planta para así aportar en su crecimiento y posteriormente obtener un producto bueno y de calidad. El proceso de ventilación brinda al microclima una temperatura adecuada para la planta, en la cual se encontrara en el rango óptimo y permita el correcto desarrollo de la planta en el sembrío. El proceso de refrigeración similar a la ventilación llevando el clima a los valores ideales que pueda la planta desarrollarse en su mejor esplendor.

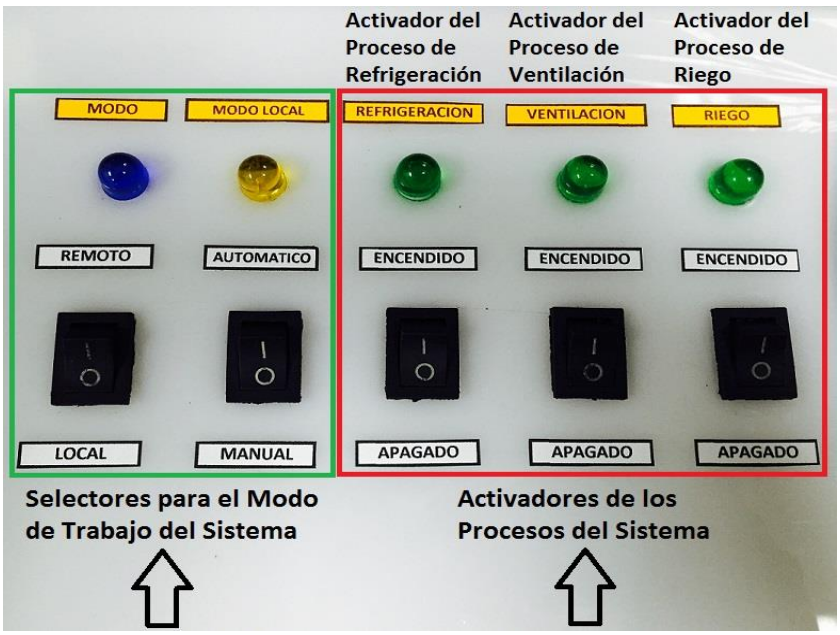


Figura 4.1 - Interruptores para activación/desactivación de los Procesos del Sistema.

Paro de emergencia.- Si bien este último no entra dentro de los procesos pero no deja de ser importante ya que en un sistema automatizado es necesario que exista un proceso que detenga completamente la marcha ante cualquier adversidad o problema en los procesos descritos anteriormente y así mismo uno que reanude la marcha del proceso en el sistema.

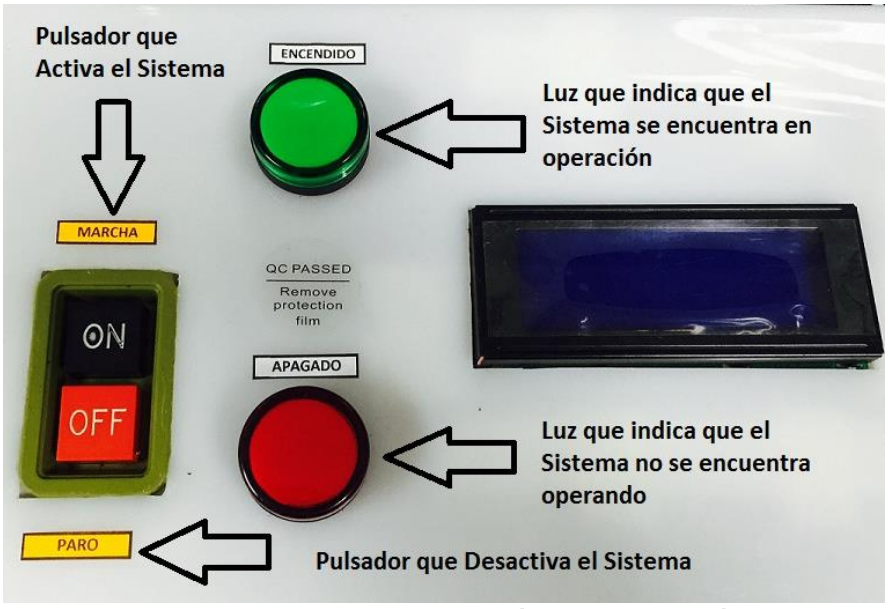


Figura 4.2 - Interruptores para activación/desactivación del Sistema.

En el Anexo A se encuentra un diagrama del flujo donde se evidencia la secuencia de activación o desactivación de cada uno de los actuadores de acuerdo a las variables climáticas que se encuentren presentes.

CONCLUSIONES YRECOMENDACIONES

CONCLUSIONES

1. De acuerdo a la Tabla 2, el prototipo de “Telecontrol de Invernadero” es factible implementarlo en el campo obteniendo beneficios, además de ganancias a mediano o largo plazo. En primera instancia el valor supera la inversión de un sembrado tradicional, pero los beneficios que se obtendrá harán que se recupere la inversión en aproximadamente 5 años sin tomar en consideración que el producto final de un invernadero presenta una mejor calidad en comparación con un sembrado tradicional que implica usos de pesticidas.

RECOMENDACIONES

1. Para el correcto funcionamiento del sistema se debería adicionar una batería alterna o un generador eléctrico que permita continuar el proceso sin ningún inconveniente.

2. Se indica que este proyecto en una próxima versión puede contener más variables que puedan ser analizadas como fue comentado inicialmente, para tener una mejor seguridad y calidad de los productos sembrados. Tal como el nivel de acidez del suelo, y una correcta fumigación por plagas que se presenten en el invernadero.

ANEXOS

ANEXO A

FLUJO DE ACTIVACIÓN O DESACTIVACIÓN DE LOS ACTUADORES DEL SISTEMA

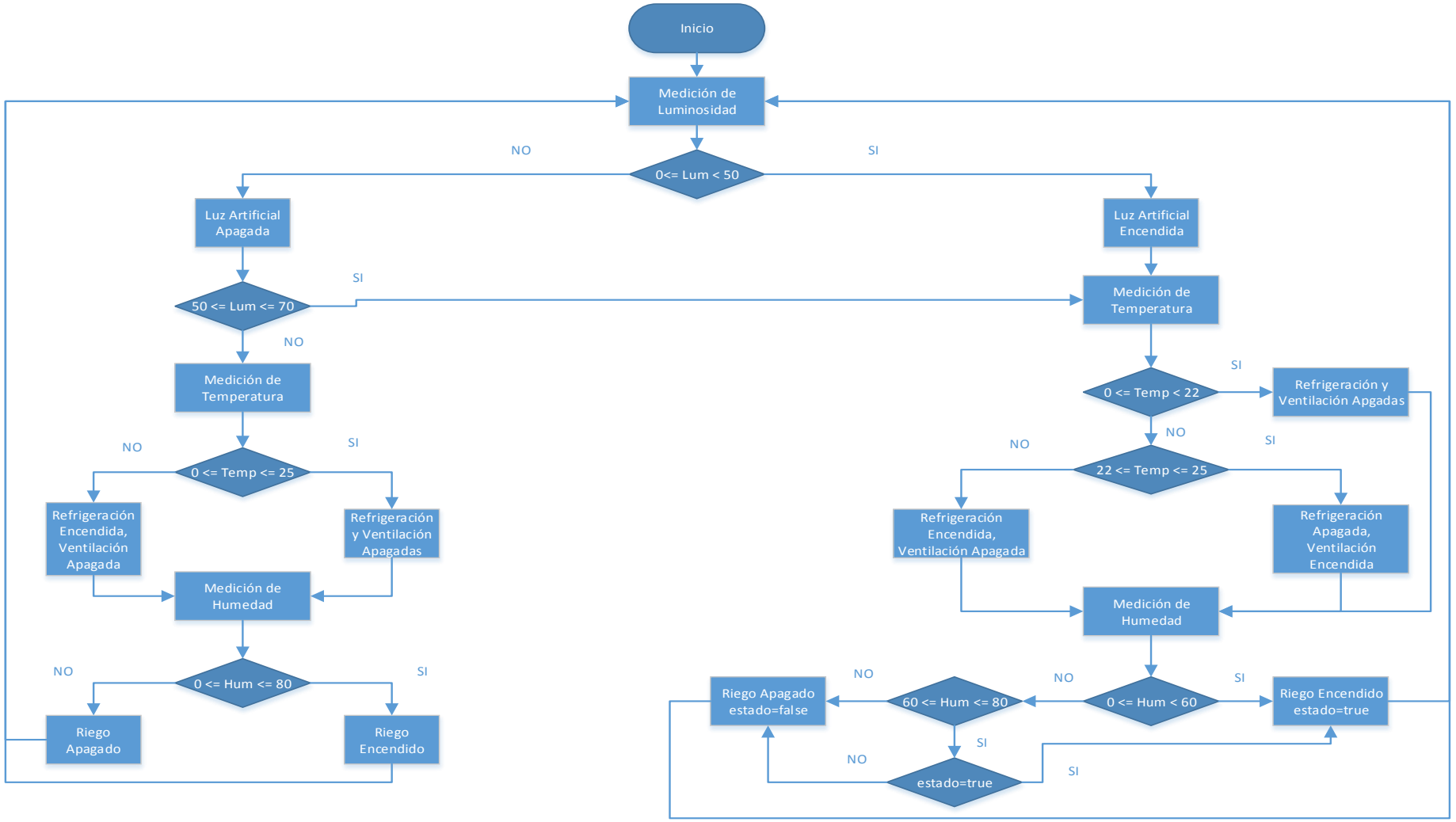


Figura A.1 – Flujo del Proceso de Activación o Desactivación de los actuadores del Sistema.

ANEXO B

MANUAL DE USUARIOS

Manual de Usuario

Al ingresar a la interfaz Web del Invernadero se muestra la página principal en la cual se realiza la autenticación de los usuarios, como se muestra en la Figura A.1.

Figura A.1.- Página principal, en la cual se realiza la autenticación de usuario.

En primer lugar mostraremos el Rol de Usuario Administrador. En la Figura A.2 podemos observar la página Principal del Usuario con privilegio de Administrador, con su Respectivo menú.

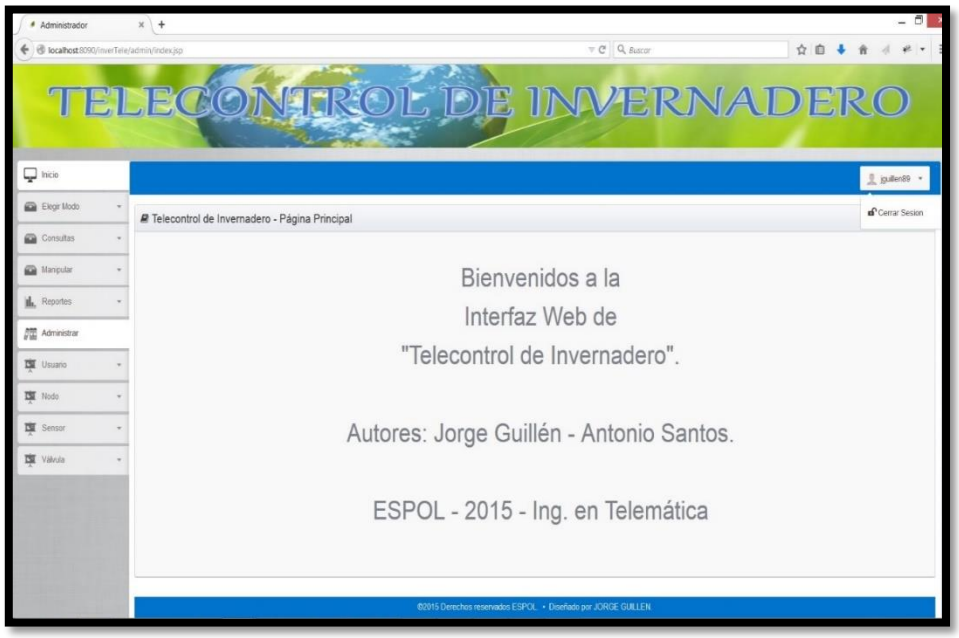


Figura A.2.- Página principal de Usuarios con privilegio de administrador.

Al hacer clic en el menú “Elegir Modo” (resaltado en la Figura A.3) se desplegará la opción “Elegir Modo de Trabajo” en la cual podrá elegir entre Automático y Manual.

Figura A.3.- Página para elegir el modo de trabajo, automático o manual.

Haciendo clic en el menú “Consultas” y luego en “Valor de Sensores” se mostrará la página donde podremos elegir el Nodo, Tipo de sensor y el Sensor del que deseamos consultar su valor actual, tal como se muestra en la Figura A.4.

Figura A.4.- Página de consulta de valor de los sensores por nodo.

Luego al hacer clic en el botón “Consultar” nos redirigirá a la página donde mostraremos los valores: actual, mínimo y máximo del sensor antes consultado, como se muestra en la Figura A.5.

Figura A.5.- Página donde se muestra el valor actual del sensor consultado.

Si el usuario dentro del menú “Consultas” selecciona “Estado de Refrigeración” podremos ver la página que se muestra en la Figura A.6; desde la cual, luego al hacer clic en el botón “Consultar”, podremos visualizar el estado de la Refrigeración.

Figura A.6.- Página para consultar el estado de la refrigeración en el nodo seleccionado.

Si el usuario dentro del menú “Consultas” hace clic en la pestaña de “Estado de Ventilación” podremos ver la página que se muestra en la Figura A.7, luego al hacer clic en el botón “Consultar” nos redirigiremos a la página que muestra el estado de la Ventilación.

Figura A.7.- Página para consultar el estado de la ventilación en el nodo seleccionado.

Si el usuario dentro del menú “Consultas” selecciona “Estado de Válvulas” podremos ver la página que se muestra en la Figura A.8, luego al hacer clic en el botón “Consultar” nos redirigiremos a la página que muestra el estado de la Válvula.

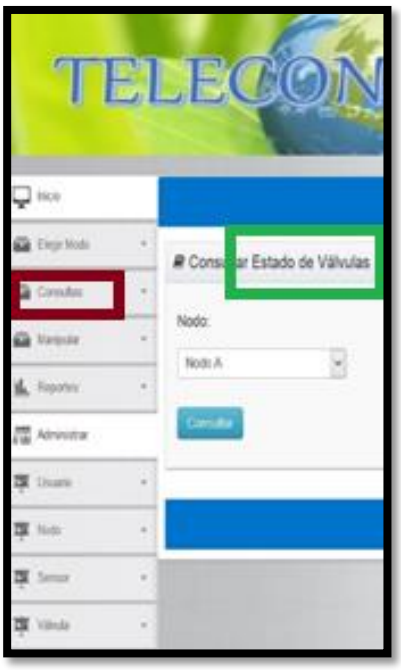


Figura A.8.- Página para consultar el estado de la válvula en el nodo seleccionado.

Si el usuario dentro del menú “Consultas” hace clic en la pestaña de “Estado de Luz Artificial” podremos ver la página que se muestra en la Figura A.9, luego al hacer clic en el botón “Consultar” nos redirigiremos a la página correspondiente.

Figura A.9.- Página para consultar el estado de la luz artificial en el nodo seleccionado.

Al hacer clic en el menú “Manipular” y seleccionar “Válvulas” se presenta la página para consultar el Estado de la Válvula (ver Figura A.10); si el Modo de Trabajo es Automático nos da la opción de observar el estado como se observa en la Figura A.11, pero si el Modo de trabajo es Manual se da la opción de cambiar el estado de la Válvula como se muestra en la Figura A.12.

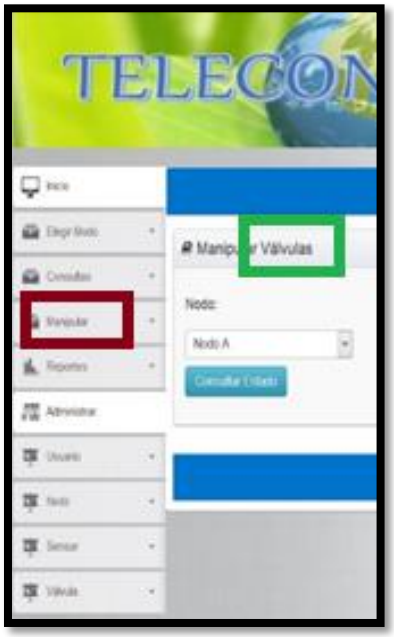


Figura A.10.- Página para consultar el estado de la válvula antes de ser manipulada.

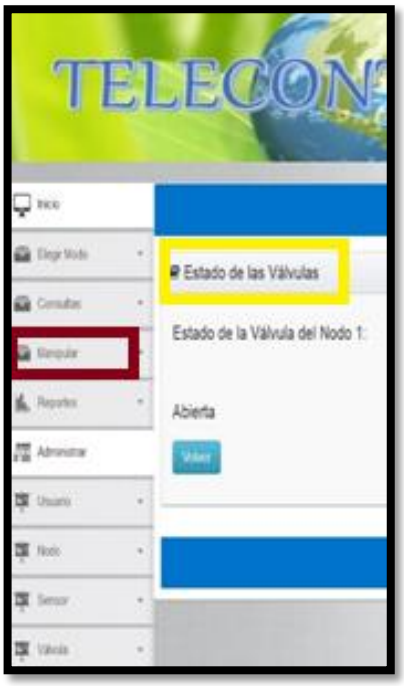


Figura A.11.- Página donde se muestra el estado de la válvula cuando el modo de trabajo es automático.

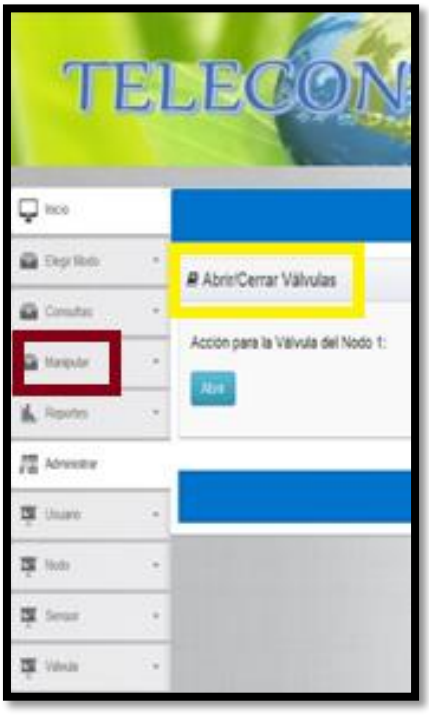


Figura A.12.- Página donde se muestra la acción a realizar en la válvula, posible realizar solo en modo manual.

Al hacer clic en el menú “Manipular” y seleccionar “Ventilación” se presenta la página para consultar el Estado de la Ventilación, como se muestra en la Figura A.13; si el Modo de Trabajo es Automático nos da la opción de observar el estado en que se encuentra como se observa en la Figura A.14, pero si el Modo de trabajo es Manual se da la opción de cambiar el estado de la Ventilación como se muestra en la Figura A.15.

Figura A.13.- Página para consultar el estado de la ventilación.

Figura A.14.- Página donde se muestra el estado de la ventilación cuando el modo de trabajo es automático

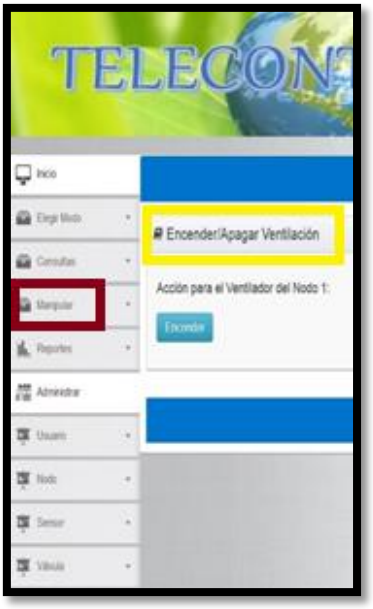


Figura A.15.- Página donde se muestra la acción a realizar en la ventilación**.**

Dentro del menú “Manipular” seleccione “Refrigeración” y podrá ver la página para consultar el Estado de la Refrigeración (ver Figura A.16); si el Modo de Trabajo es Automático nos da la opción de observar el estado (ver la Figura A.17), pero si el Modo de trabajo es Manual se da la opción de cambiar el estado de la Refrigeración como se muestra en la Figura A.18.

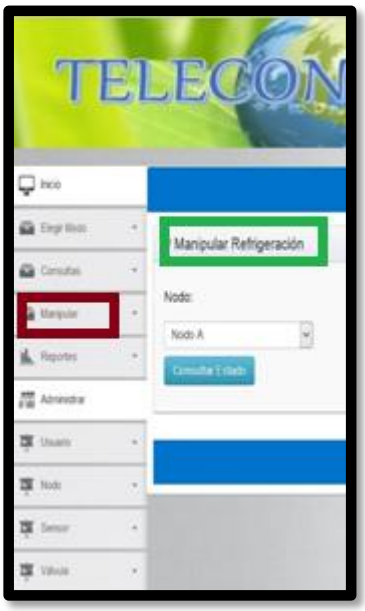


Figura A.16.- Página para consultar el estado de la refrigeración

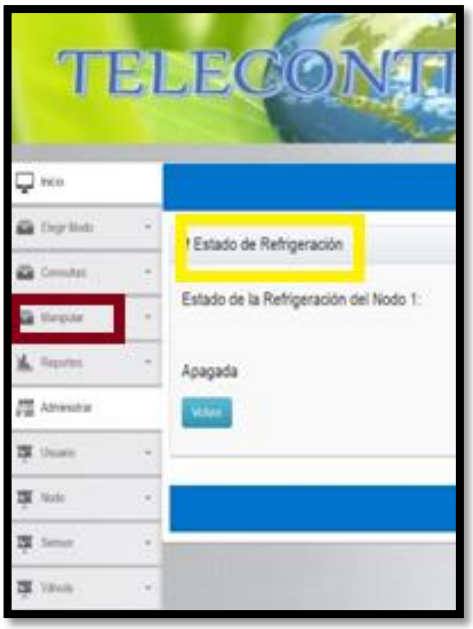


Figura A.17.- Página donde se muestra el estado de la refrigeración cuando el modo de trabajo es automático.

Figura A.18.- Página donde se muestra la acción a realizar en la refrigeración, posible realizar solo en modo manual**.**

Si hacer clic en el menú “Manipular” se selecciona “Luz Artificial” se presenta la página mostrada en la Figura A.19. Si el Modo de Trabajo es Automático nos da la opción de observar el estado en que se encuentra como se observa en la Figura A.20, pero si el Modo de trabajo es Manual se da la opción de cambiar el estado de la Luz Artificial como se muestra en la Figura A.21.

Figura A.19.- Página para consultar el estado de la luz artificial antes de ser manipulada.

Figura A.20.- Página donde se muestra el estado de la luz artificial cuando el modo de trabajo es automático

Figura A.21.- Página donde se muestra la acción a realizar en la luz artificial, posible realizar solo en modo manual**.**

Al hacer clic en el menú “Reportes” y luego en la opción “Medición de Sensores” se muestra la página para escoger la Fecha en la que se desea consultar las mediciones de los sensores por Nodo y Tipo de Sensor, como se muestra en la Figura A.22.

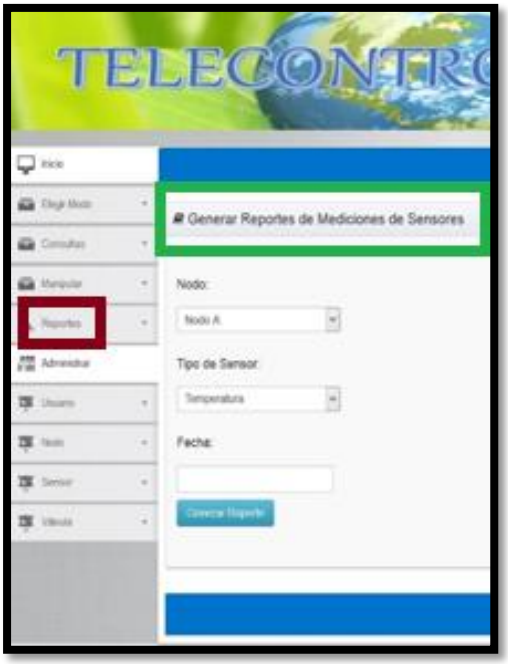


Figura A.22.- Página para seleccionar la fecha en la que se quiere generar reporte de medición de un sensor por nodo y tipo de sensor.

Al hacer clic en el botón “Generar Reporte” nos redirige a la página donde nos muestra la tabla con los campos necesarios para el reporte el cual se lo puede descargar en diferentes formatos, esto se puede observar en la Figura A.23.

Figura A.23.- Reporte de medición del sensor seleccionado en la fecha seleccionada

Dentro del menú “Reportes” existe la opción “Medición de Sensores por Rango de Fecha” que muestra una página para escoger las Fechas de consulta de las mediciones de los sensores por Nodo y Tipo de Sensor, como se muestra en la Figura A.24.

Figura A.24.- Página para seleccionar un rango de fechas

Al hacer clic en el botón “Generar Reporte” nos redirige a la página donde nos muestra la tabla con los campos necesarios para el reporte el cual se lo puede descargar en diferentes formatos, esto se puede observar en la Figura A.25.

Figura A.25.- Página donde se muestra el reporte de medición del sensor seleccionado

Al hacer clic en el menú “Reportes” y luego en la opción “Conexiones de Válvula” se mostrará la página para escoger la Fecha de consulta, como se muestra en la Figura A.26, al hacer clic en el botón “Consultar” nos redirige a la página donde nos muestra la tabla con los campos necesarios para el reporte el cual se lo puede descargar en diferentes formatos, esto se puede observar en la Figura A.27.

Figura A.26.- Página para seleccionar la fecha de consulta.

Figura A.27.- Página donde se muestra el reporte de conexión/desconexión de la válvula.

Al hacer clic en el menú “Reportes” y luego en la opción “Conexiones de Válvulas por Rango de Fecha” se mostrará la página para escoger las Fechas en la que se desea consultar las conexiones/desconexiones de la válvula por Nodo, como se muestra en la Figura A.28, al hacer clic en el botón “Consultar” nos redirige a la página donde nos muestra la tabla con los campos necesarios para el reporte el cual se lo puede descargar en diferentes formatos, esto se puede observar en la Figura A.29.

Figura A.28.- Página para seleccionar un rango de fechas en la que se quiere generar reporte de conexión/desconexión de válvula por nodo.

Figura A.29.- Página donde se muestra el reporte de conexión/desconexión de la válvula.

El administrador puede manejar la información de los usuarios. Hacer clic en el menú "Usuario" donde tendra la opción de "Listar Usuarios" como se muestra en la Figura A.30. En esta página el administrador puede editar la información de los usuarios dando clic en "Editar" (ver Figura A.31) o eliminar al usuario con la opción "Eliminar".

Figura A.30.- Página donde se muestran los usuarios registrados en la base de datos, con opción a descargar la lista en diferentes formatos.

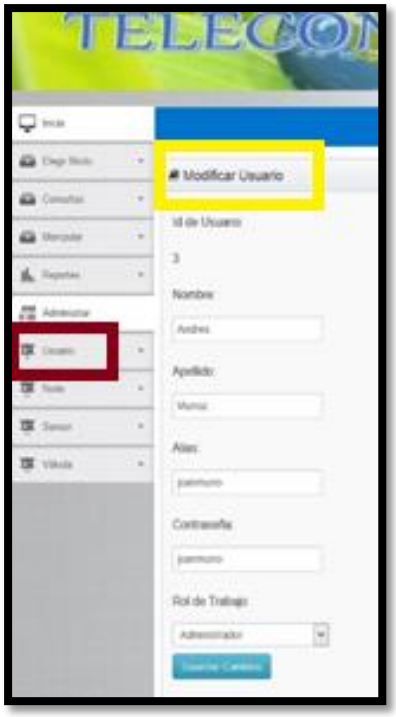


Figura A.31.- Página donde se puede modificar la información de los usuarios y registrar el cambio en la base de datos.

Además en el menú despeglable tendremos la opción de "Crear Usuario" como mostramos en la Figura A.32 y la opción de "Asociar Nodo a Usuario" como lo podemos notar en la Figura A.33.

Figura A.32.- Página para crear un nuevo usuario y registrarlo en la base de datos.

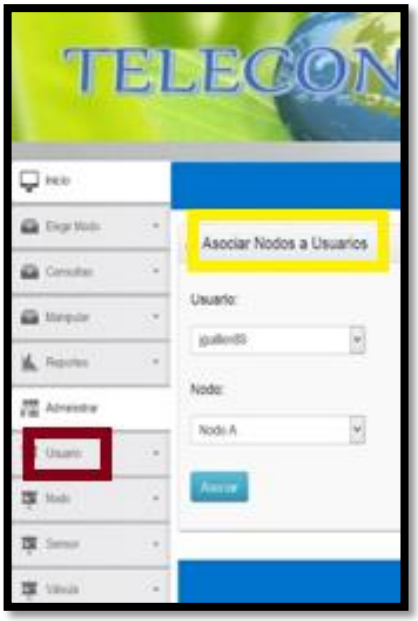


Figura A.33.- Página para asociar un usuario a un nodo del invernadero y registrar en la base de datos.

El administrador también puede manejar la información de los nodos, al hacer clic en el menú “Nodo” donde tendra la opción de “Listar Nodos” como se muestra en la Figura A.34. En esta página el administrador puede editar la información de los nodos dando clic en “Editar” (ver Figura A.35) o eliminar el nodo con la opción “Eliminar”.

Figura A.34.- Página donde se listan los nodos existentes en el invernadero y registrados en la base de datos, con opción a descargar la lista en diferentes formatos.

Figura A.35.- Página para modificar la información de los nodos del invernadero y registrar el cambio en la base de datos.

Además en el menú despeglable tendremos la opción de “Crear Nodo” como mostramos en la Figura A.36.

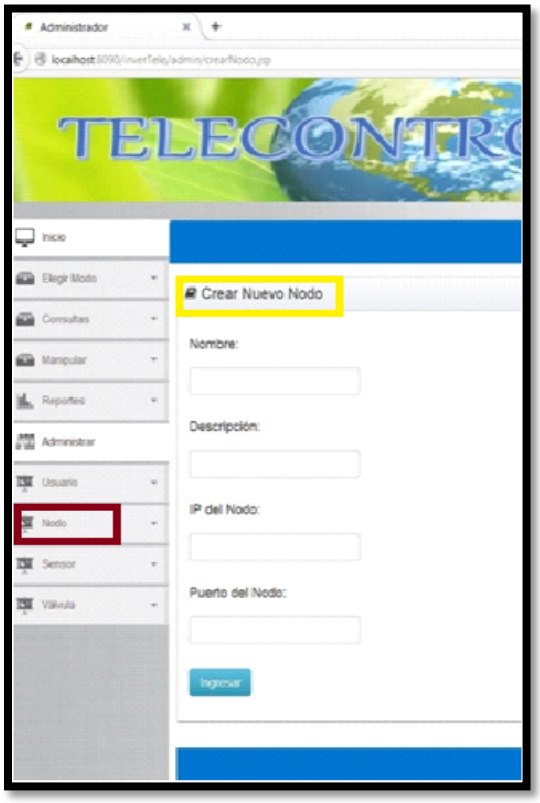


Figura A.36.- Página para crear un nuevo nodo y registrarlo en la base de datos.

Si el administrador desea manejar la información de los sensores, haciendo clic en el menú “Sensor” tendra la opción de “Listar Sensores” como se muestra en la Figura A.37. En esta página el administrador puede editar la información de los sensores dando clic en “Editar” (ver Figura A.38) o eliminar el sensor con la opción “Eliminar”.

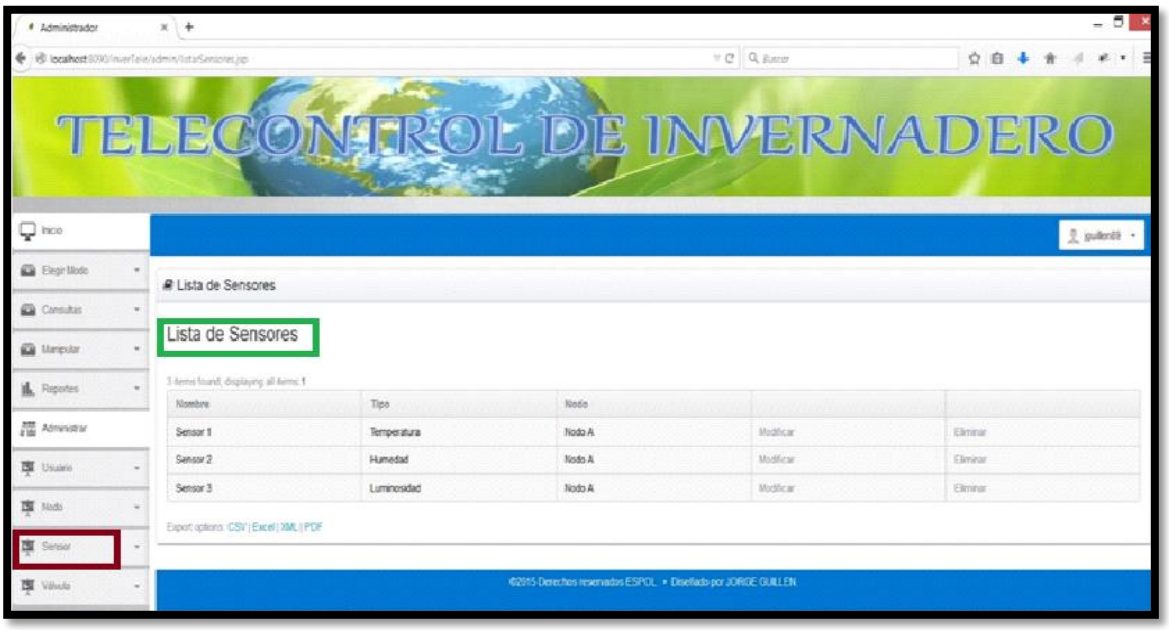


Figura A.37.- Página para listar los sensores existentes en el invernadero y registrados en la base de datos, con opción a descargar la lista en diferentes formatos.

Figura A. 38.- Página para modificar la información de los sensores del invernadero y registrar el cambio en la base de datos.

En el menú despeglable el usuario tendrá la opción de “Crear Sensor” como mostramos en la Figura A.39.

Figura A.39.- Página para crear un nuevo sensor en un nodo y registrarlo en la base de datos.

El administrador podrá manejar la información de las válulas, haciendo clic en el menú “Válvula” tendra la opción de “Listar Válvulas” como se muestra en la Figura A.40. En esta página el administrador puede editar la información de las válvulas dando clic en “Editar” (ver Figura A.41) o eliminar la válvula con la opción “Eliminar”.

Figura A.40.- Página para listar las válvulas existentes en el invernadero y registradas en la base de datos, con opción a descargar la lista en diferentes formatos.

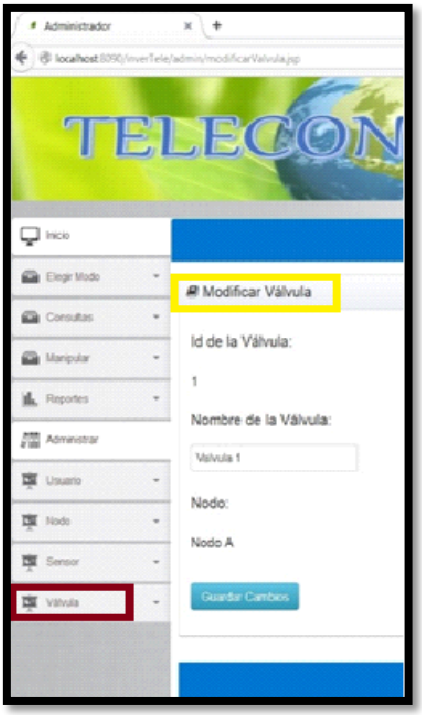


Figura A.41.- Página para modificar la información de las válvulas del invernadero y registrar el cambio en la base de datos.

Además en el menú despeglable el usuario tendrá la opción de “Crear Válvula” como mostramos en la Figura A.42.

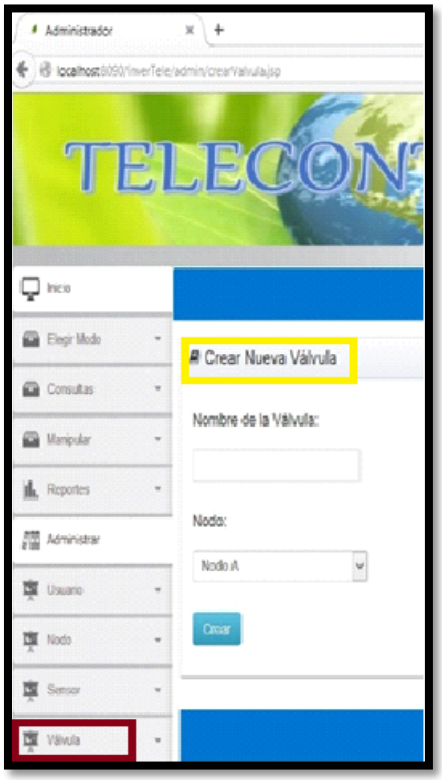


Figura A.42.- Página para crear una nueva válvula en un nodo del invernadero y registrar en la base de datos.

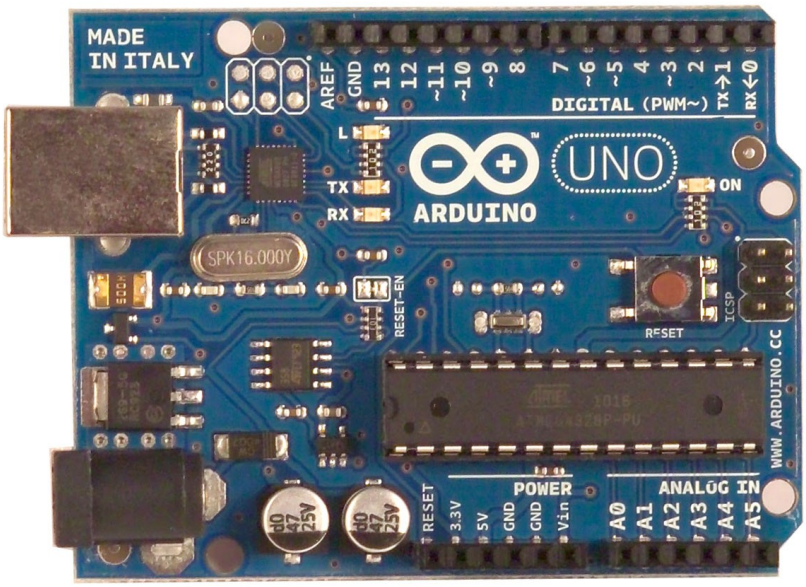
Para los Roles de Operario e Invitado se modifica el menú en el cual se pueden realizar diferentes acciones de acuerdo al Rol del Trabajo, para el Operario a diferencia del Administrador no puede eliminar ni editar los usuarios, nodos, válvulas y sensores. El Invitado a diferencia de los dos Roles antes mencionados el solo puede listar usuarios, nodos, sensores y válvulas sin poder generar reportes además este usuario solo puede consultar estado de los actuadores y valor de lo sensores sin poder cambiar el Modo del Trabajo.

ANEXO C

HOJAS DE DATOS DE LOS SENSORES, ARDUINO UNO

Y ARDUINO ETHERNET SHIELD.

Arduino UNO



Product Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduno, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Index

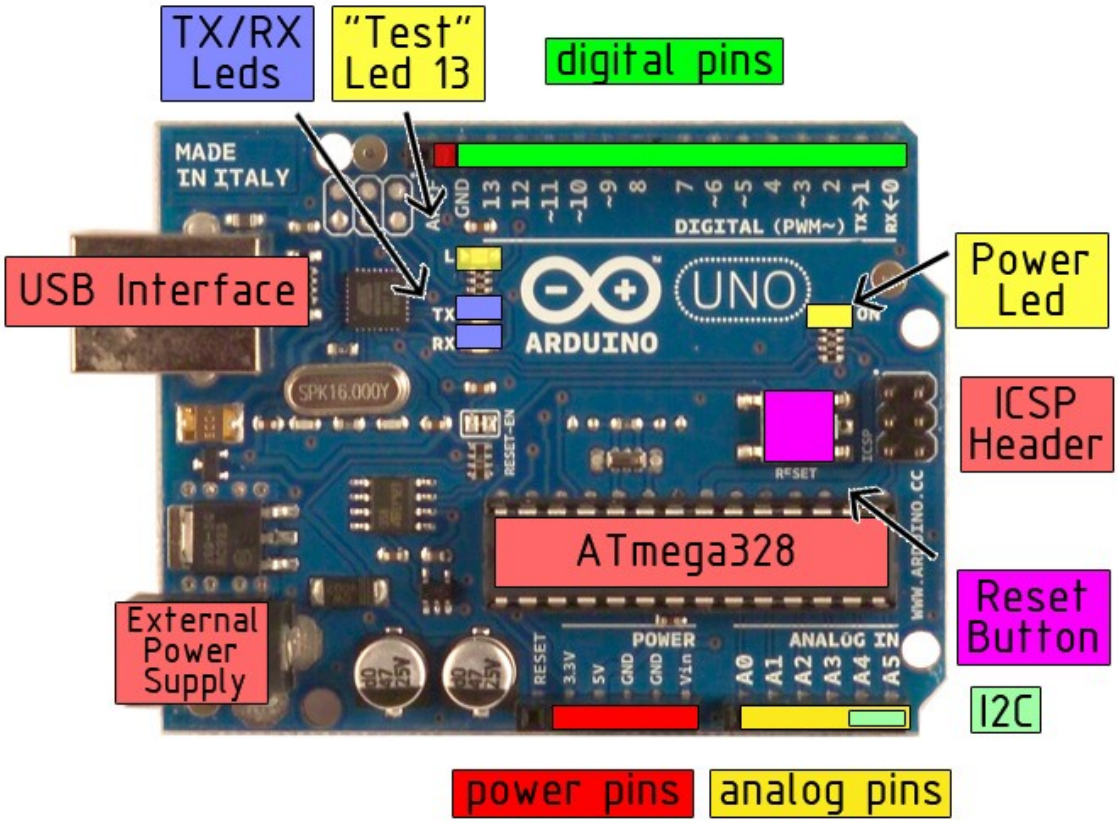
Technical Specification

EAGLE files: arduino-duemilanove-uno-design.zip Schematic: arduino-uno-schematic.pdf

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

the board



Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. TThese pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I²C: 4 (SDA) and 5 (SCL).** Support I²C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and Atmega328 ports.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, an *.inf file is required..

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. To use the SPI communication, please see the ATmega328 datasheet.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno w/ ATmega328" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega8U2 firmware source code is available . The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader).

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

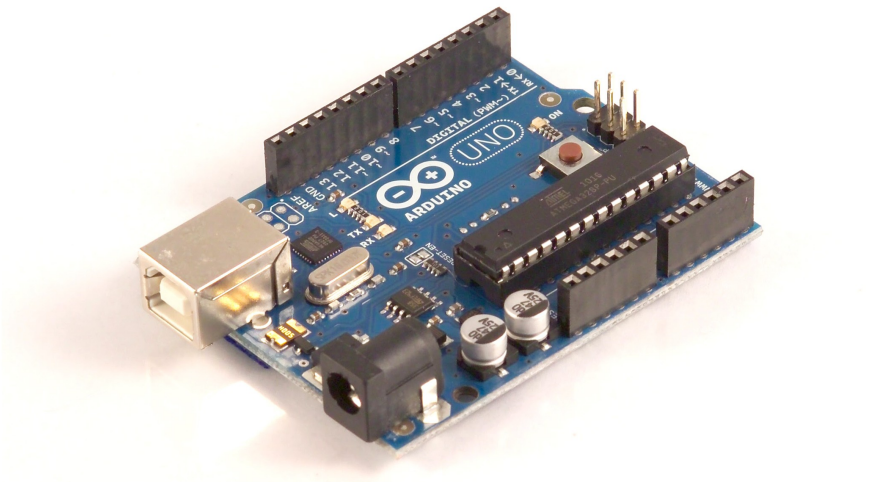
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.



How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install Windows Install Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

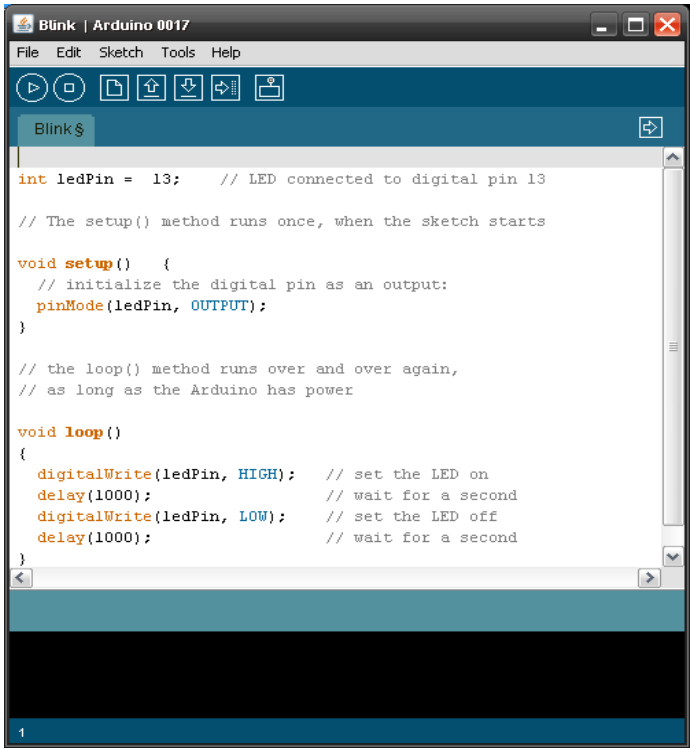
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

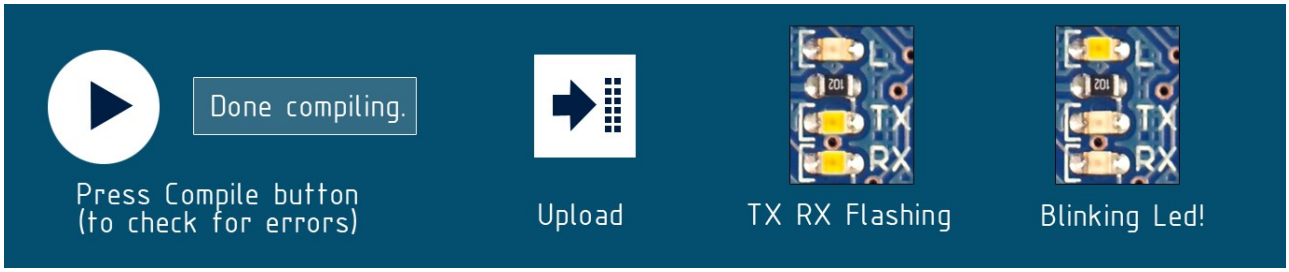
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

Once you have your skecth you'll see something very close to the screenshot on the right.

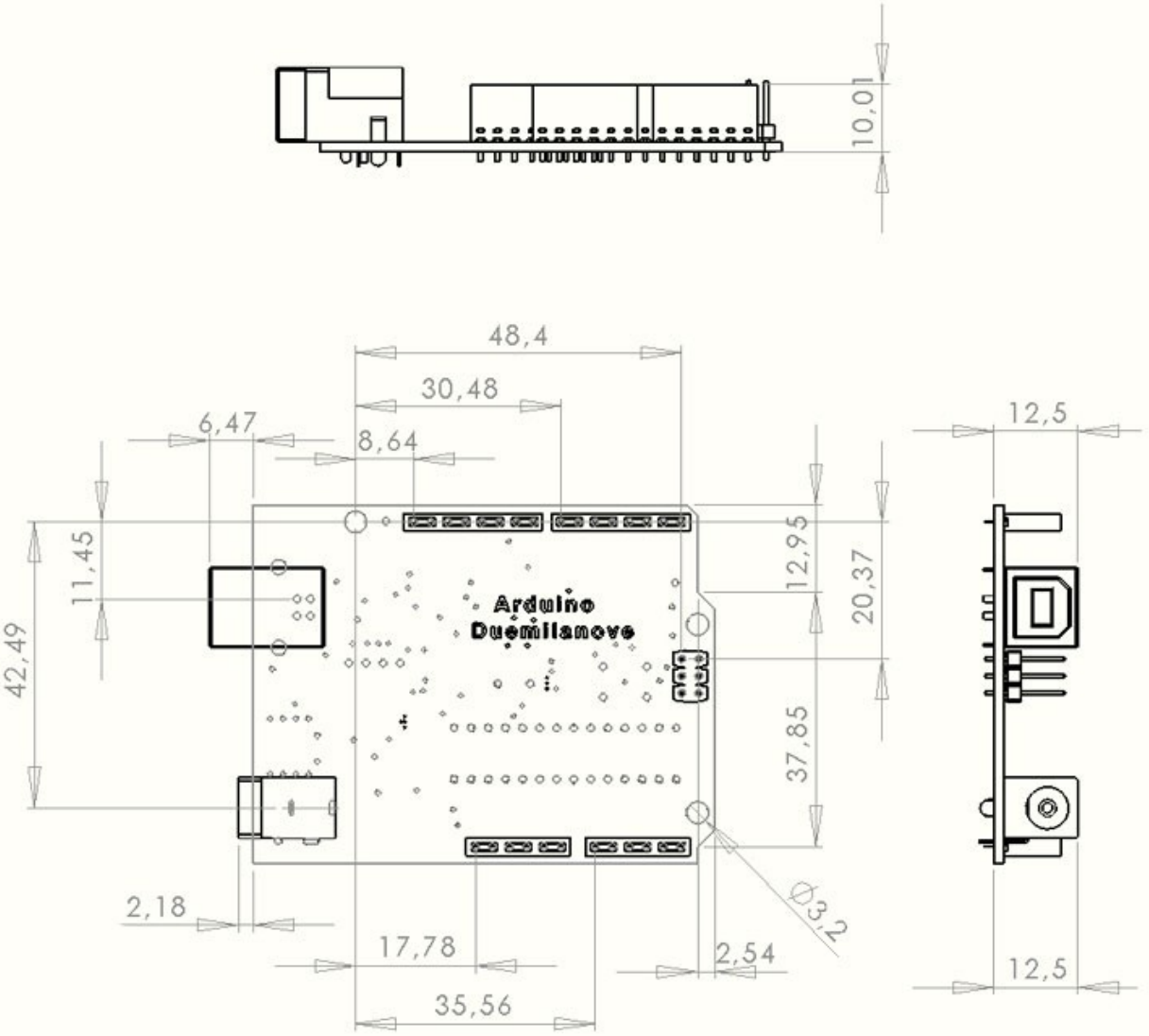
In **Tools>Board** select

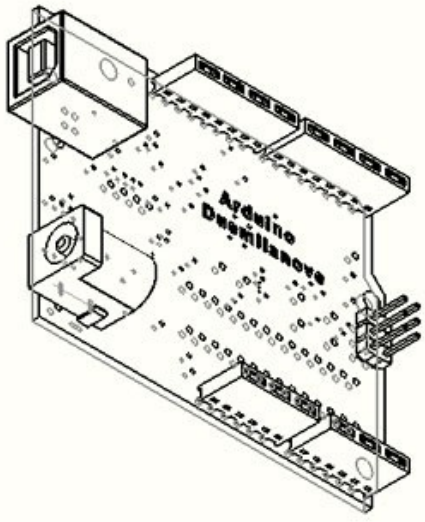
Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.





Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

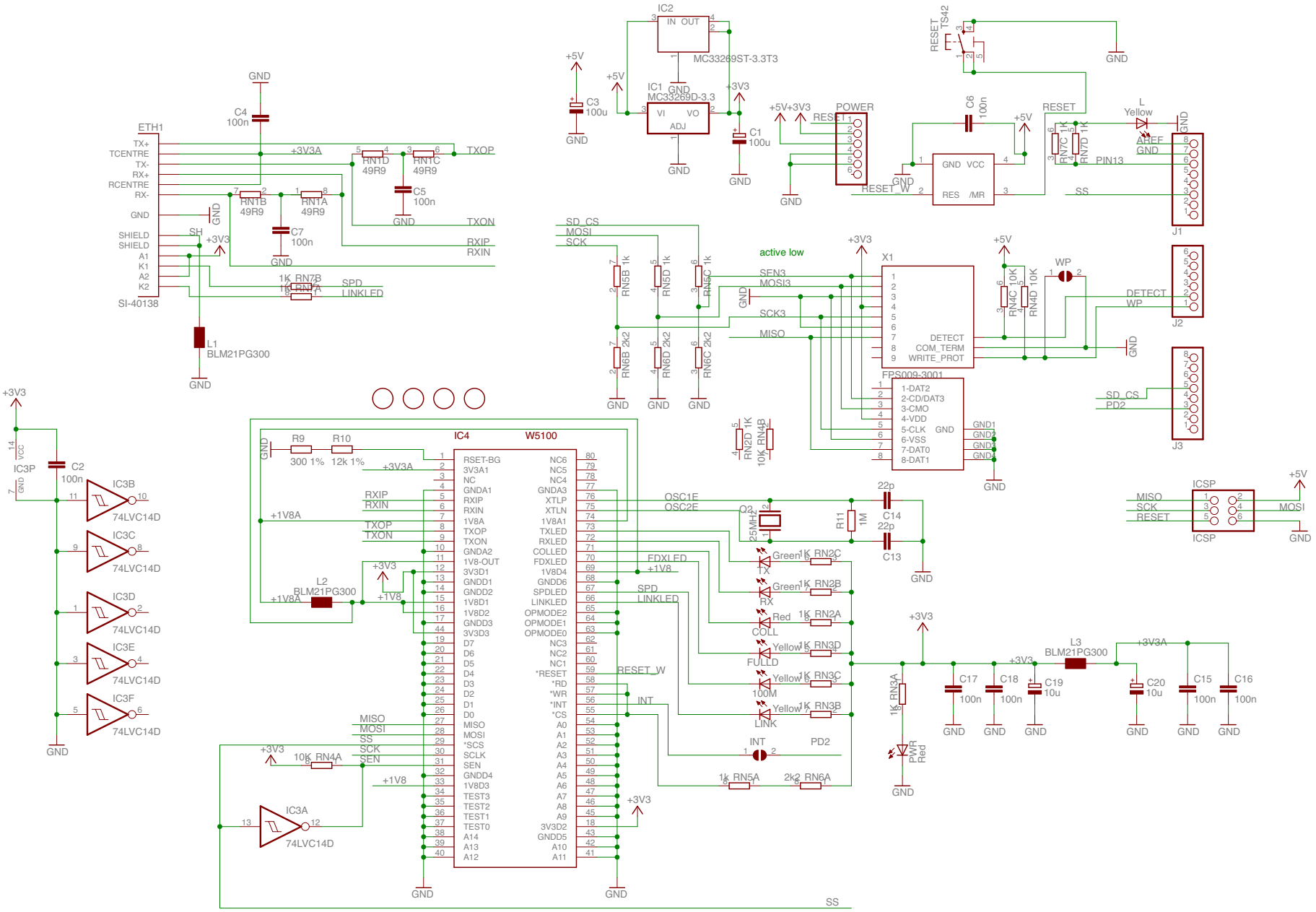
In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.



Arduino ETHERNET - shield V5

DS18B20
Programmable Resolution 1-Wire Digital Thermometer

DESCRIPTION

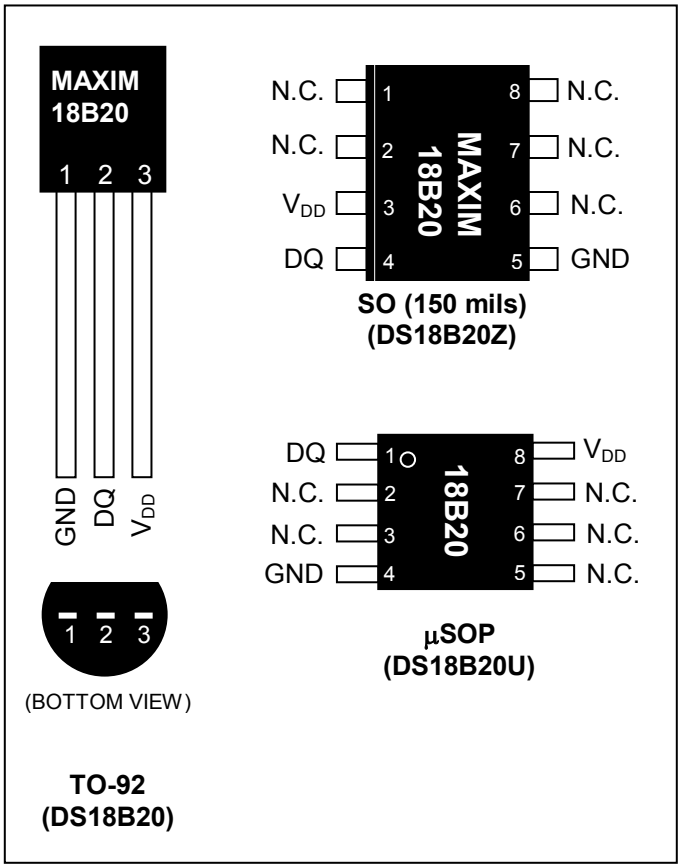
The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to +125°C and is accurate to ±0.5°C over the range of -10°C to +85°C. In addition, the DS18B20 can derive power directly from the data line ("parasite power"), eliminating the need for an external power supply.

Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

FEATURES

- Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Requires No External Components
- Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- Measures Temperatures from -55°C to +125°C (-67°F to +257°F)
- ±0.5°C Accuracy from -10°C to +85°C
- Thermometer Resolution is User Selectable from 9 to 12 Bits
- Converts Temperature to 12-Bit Digital Word in 750ms (Max)
- User-Definable Nonvolatile (NV) Alarm Settings
- Alarm Search Command Identifies and Addresses Devices Whose Temperature is Outside Programmed Limits (Temperature Alarm Condition)
- Available in 8-Pin SO (150 mils), 8-Pin µSOP, and 3-Pin TO-92 Packages
- Software Compatible with the DS1822
- Applications Include Thermostatic Controls, Industrial Systems, Consumer Products, Thermometers, or Any Thermally Sensitive System

PIN CONFIGURATIONS



1-Wire is a registered trademark of Maxim Integrated Products, Inc.

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

REV: 042208

ORDERING INFORMATION

PART	TEMP RANGE	PIN-PACKAGE	TOP MARK
DS18B20	-55°C to +125°C	3 TO-92	18B20
DS18B20+	-55°C to +125°C	3 TO-92	18B20
DS18B20/T&R	-55°C to +125°C	3 TO-92 (2000 Piece)	18B20
DS18B20+T&R	-55°C to +125°C	3 TO-92 (2000 Piece)	18B20
DS18B20-SL/T&R	-55°C to +125°C	3 TO-92 (2000 Piece)*	18B20
DS18B20-SL+T&R	-55°C to +125°C	3 TO-92 (2000 Piece)*	18B20
DS18B20U	-55°C to +125°C	8 µSOP	18B20
DS18B20U+	-55°C to +125°C	8 µSOP	18B20
DS18B20U/T&R	-55°C to +125°C	8 µSOP (3000 Piece)	18B20
DS18B20U+T&R	-55°C to +125°C	8 µSOP (3000 Piece)	18B20
DS18B20Z	-55°C to +125°C	8 SO	DS18B20
DS18B20Z+	-55°C to +125°C	8 SO	DS18B20
DS18B20Z/T&R	-55°C to +125°C	8 SO (2500 Piece)	DS18B20
DS18B20Z+T&R	-55°C to +125°C	8 SO (2500 Piece)	DS18B20

+Denotes a lead-free package. A "+" will appear on the top mark of lead-free packages.
T&R = Tape and reel.
**TO-92 packages in tape and reel can be ordered with straight or formed leads. Choose "SL" for straight leads. Bulk TO-92 orders are straight leads only.*

PIN DESCRIPTION

PIN			NAME	FUNCTION
SO	µSOP	TO-92		
1, 2, 6, 7, 8	2, 3, 5, 6, 7	—	N.C.	No Connection
3	8	3	V_{DD}	Optional V_{DD}. V_{DD} must be grounded for operation in parasite power mode.
4	1	2	DQ	Data Input/Output. Open-drain 1-Wire interface pin. Also provides power to the device when used in parasite power mode (see the *Powering the DS18B20* section.)
5	4	1	GND	Ground

OVERVIEW

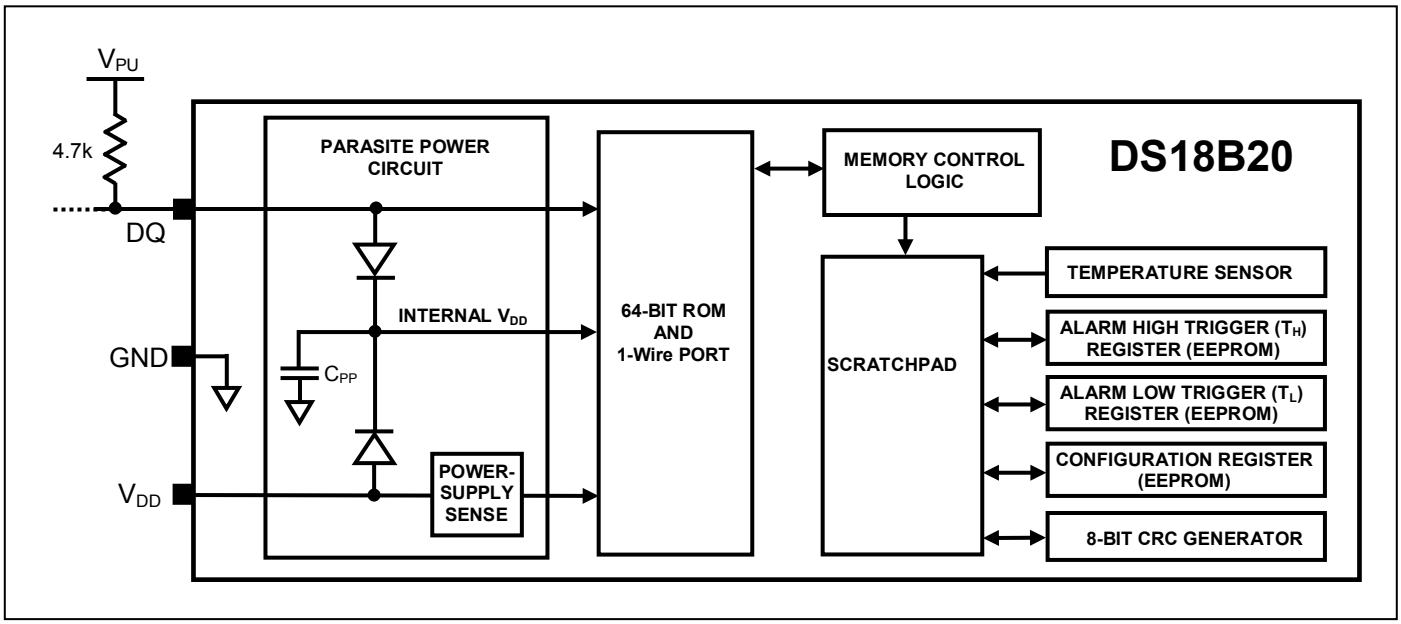
Figure 1 shows a block diagram of the DS18B20, and pin descriptions are given in the *Pin Description* table. The 64-bit ROM stores the device's unique serial code. The scratchpad memory contains the 2-byte temperature register that stores the digital output from the temperature sensor. In addition, the scratchpad provides access to the 1-byte upper and lower alarm trigger registers (T_H and T_L) and the 1-byte configuration register. The configuration register allows the user to set the resolution of the temperature-to-digital conversion to 9, 10, 11, or 12 bits. The T_H, T_L, and configuration registers are nonvolatile (EEPROM), so they will retain data when the device is powered down.

The DS18B20 uses Maxim's exclusive 1-Wire bus protocol that implements bus communication using one control signal. The control line requires a weak pullup resistor since all devices are linked to the bus via a 3-state or open-drain port (the DQ pin in the case of the DS18B20). In this bus system, the microprocessor (the master device) identifies and addresses devices on the bus using each device's unique 64-bit code. Because each device has a unique code, the number of devices that can be addressed on one

bus is virtually unlimited. The 1-Wire bus protocol, including detailed explanations of the commands and "time slots," is covered in the *1-Wire Bus System* section.

Another feature of the DS18B20 is the ability to operate without an external power supply. Power is instead supplied through the 1-Wire pullup resistor via the DQ pin when the bus is high. The high bus signal also charges an internal capacitor (C_{PP}), which then supplies power to the device when the bus is low. This method of deriving power from the 1-Wire bus is referred to as "parasite power." As an alternative, the DS18B20 may also be powered by an external supply on V_{DD}.

Figure 1. DS18B20 Block Diagram



OPERATION—MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit. The DS18B20 powers up in a low-power idle state. To initiate a temperature measurement and A-to-D conversion, the master must issue a Convert T [44h] command. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its idle state. If the DS18B20 is powered by an external supply, the master can issue "read time slots" (see the *1-Wire Bus System* section) after the Convert T command and the DS18B20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. If the DS18B20 is powered with parasite power, this notification technique cannot be used since the bus must be pulled high by a strong pullup during the entire temperature conversion. The bus requirements for parasite power are explained in detail in the *Powering the DS18B20* section.

The DS18B20 output temperature data is calibrated in degrees Celsius; for Fahrenheit applications, a lookup table or conversion routine must be used. The temperature data is stored as a 16-bit sign-extended two's complement number in the temperature register (see Figure 2). The sign bits (S) indicate if the temperature is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. If the DS18B20 is configured for 12-bit resolution, all bits in the temperature register will contain valid data. For 11-bit resolution, bit 0 is undefined. For 10-bit resolution, bits 1 and 0 are undefined, and for 9-bit resolution bits 2, 1, and 0 are undefined. Table 1 gives examples of digital output data and the corresponding temperature reading for 12-bit resolution conversions.

Figure 2. Temperature Register Format

	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

Table 1. Temperature/Data Relationship

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (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

**The power-on reset value of the temperature register is +85°C.*

OPERATION—ALARM SIGNALING

After the DS18B20 performs a temperature conversion, the temperature value is compared to the user-defined two's complement alarm trigger values stored in the 1-byte T_H and T_L registers (see Figure 3). The sign bit (S) indicates if the value is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. The T_H and T_L registers are nonvolatile (EEPROM) so they will retain data when the device is powered down. T_H and T_L can be accessed through bytes 2 and 3 of the scratchpad as explained in the *Memory* section.

Figure 3. T_H and T_L Register Format

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

Only bits 11 through 4 of the temperature register are used in the T_H and T_L comparison since T_H and T_L are 8-bit registers. If the measured temperature is lower than or equal to T_L or higher than or equal to T_H, an alarm condition exists and an alarm flag is set inside the DS18B20. This flag is updated after every temperature measurement; therefore, if the alarm condition goes away, the flag will be turned off after the next temperature conversion.

The master device can check the alarm flag status of all DS18B20s on the bus by issuing an Alarm Search [ECh] command. Any DS18B20s with a set alarm flag will respond to the command, so the master can determine exactly which DS18B20s have experienced an alarm condition. If an alarm condition exists and the T_H or T_L settings have changed, another temperature conversion should be done to validate the alarm condition.

POWERING THE DS18B20

The DS18B20 can be powered by an external supply on the V_{DD} pin, or it can operate in "parasite power" mode, which allows the DS18B20 to function without a local external supply. Parasite power is very useful for applications that require remote temperature sensing or that are very space constrained. Figure 1 shows the DS18B20's parasite-power control circuitry, which "steals" power from the 1-Wire bus via the DQ pin when the bus is high. The stolen charge powers the DS18B20 while the bus is high, and some of the charge is stored on the parasite power capacitor (C_{PP}) to provide power when the bus is low. When the DS18B20 is used in parasite power mode, the V_{DD} pin must be connected to ground.

In parasite power mode, the 1-Wire bus and C_{PP} can provide sufficient current to the DS18B20 for most operations as long as the specified timing and voltage requirements are met (see the *DC Electrical Characteristics* and *AC Electrical Characteristics*). However, when the DS18B20 is performing temperature conversions or copying data from the scratchpad memory to EEPROM, the operating current can be as high as 1.5mA. This current can cause an unacceptable voltage drop across the weak 1-Wire pullup resistor and is more current than can be supplied by C_{PP}. To assure that the DS18B20 has sufficient supply current, it is necessary to provide a strong pullup on the 1-Wire bus whenever temperature conversions are taking place or data is being copied from the scratchpad to EEPROM. This can be accomplished by using a MOSFET to pull the bus directly to the rail as shown in Figure 4. The 1-Wire bus must be switched to the strong pullup within 10µs (max) after a Convert T [44h] or Copy Scratchpad [48h] command is issued, and the bus must be held high by the pullup for the duration of the conversion (t_{CONV}) or data transfer (t_{WR} = 10ms). No other activity can take place on the 1-Wire bus while the pullup is enabled.

The DS18B20 can also be powered by the conventional method of connecting an external power supply to the V_{DD} pin, as shown in Figure 5. The advantage of this method is that the MOSFET pullup is not required, and the 1-Wire bus is free to carry other traffic during the temperature conversion time.

The use of parasite power is not recommended for temperatures above +100°C since the DS18B20 may not be able to sustain communications due to the higher leakage currents that can exist at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that the DS18B20 be powered by an external power supply.

In some situations the bus master may not know whether the DS18B20s on the bus are parasite powered or powered by external supplies. The master needs this information to determine if the strong bus pullup should be used during temperature conversions. To get this information, the master can issue a Skip ROM [CCh] command followed by a Read Power Supply [B4h] command followed by a "read time slot". During the read time slot, parasite powered DS18B20s will pull the bus low, and externally powered DS18B20s will let the bus remain high. If the bus is pulled low, the master knows that it must supply the strong pullup on the 1-Wire bus during temperature conversions.

Figure 4. Supplying the Parasite-Powered DS18B20 During Temperature Conversions

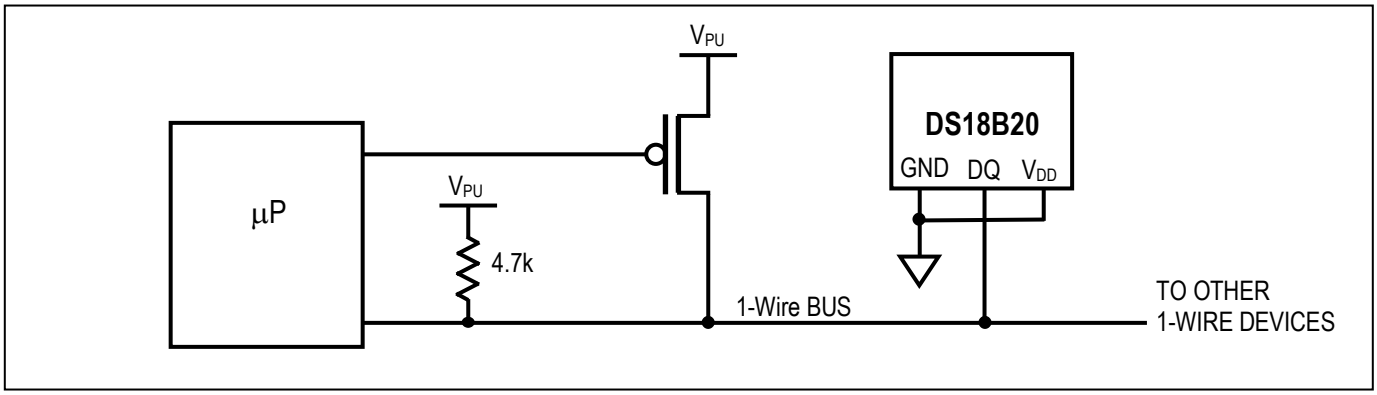
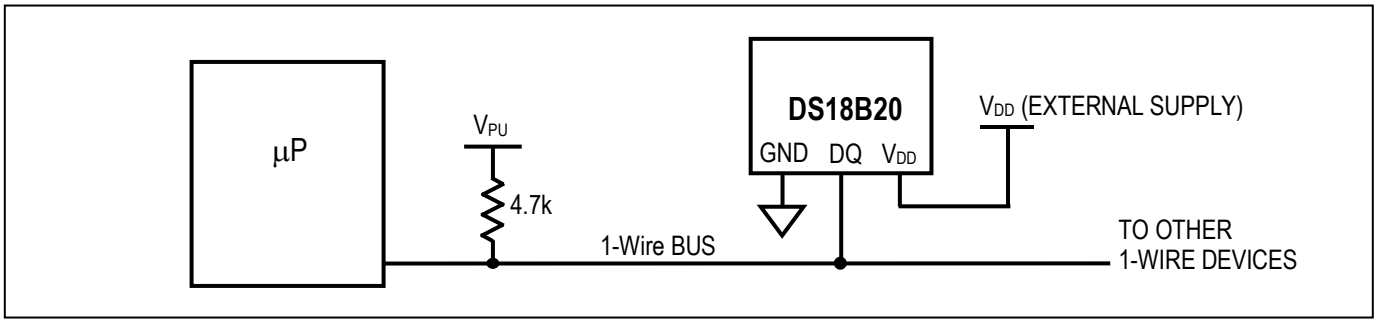


Figure 5. Powering the DS18B20 with an External Supply



64-BIT LASERED ROM CODE

Each DS18B20 contains a unique 64–bit code (see Figure 6) stored in ROM. The least significant 8 bits of the ROM code contain the DS18B20's 1-Wire family code: 28h. The next 48 bits contain a unique serial number. The most significant 8 bits contain a cyclic redundancy check (CRC) byte that is calculated from the first 56 bits of the ROM code. A detailed explanation of the CRC bits is provided in the *CRC Generation* section. The 64-bit ROM code and associated ROM function control logic allow the DS18B20 to operate as a 1-Wire device using the protocol detailed in the *1-Wire Bus System* section.

Figure 6. 64-Bit Lasered ROM Code

8-BIT CRC		48-BIT SERIAL NUMBER		8-BIT FAMILY CODE (28h)	
MSB	LSB	MSB	LSB	MSB	LSB

MEMORY

The DS18B20's memory is organized as shown in Figure 7. The memory consists of an SRAM scratchpad with nonvolatile EEPROM storage for the high and low alarm trigger registers (T_H and T_L) and configuration register. Note that if the DS18B20 alarm function is not used, the T_H and T_L registers can serve as general-purpose memory. All memory commands are described in detail in the *DS18B20 Function Commands* section.

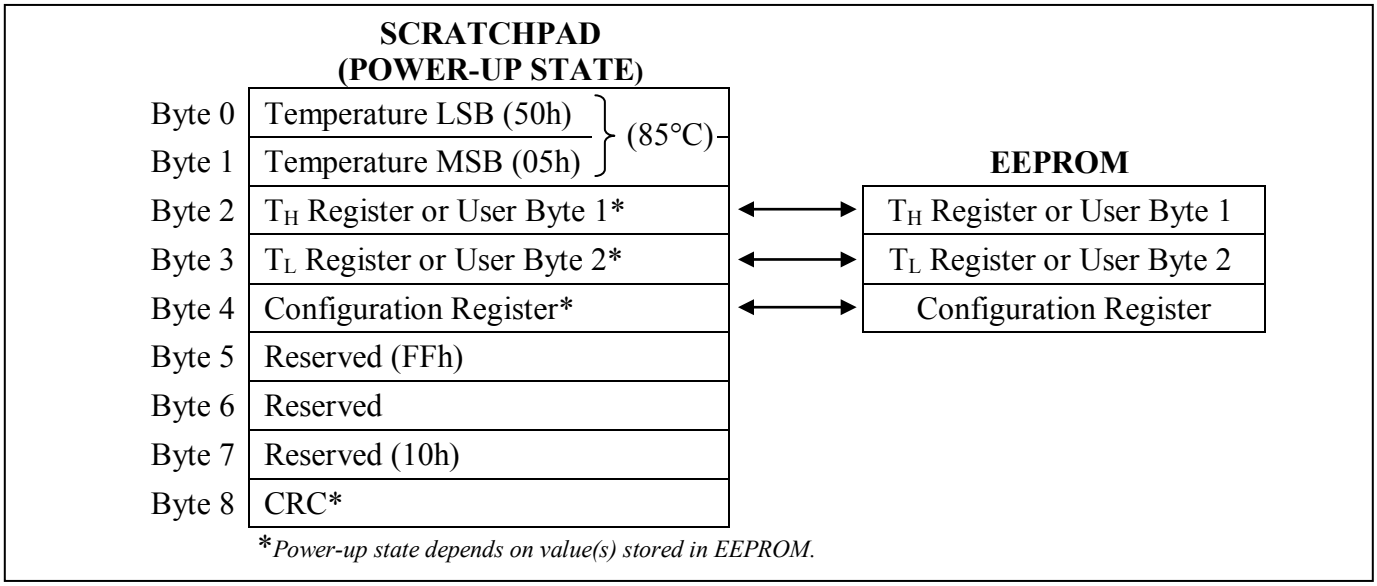
Byte 0 and byte 1 of the scratchpad contain the LSB and the MSB of the temperature register, respectively. These bytes are read-only. Bytes 2 and 3 provide access to T_H and T_L registers. Byte 4 contains the configuration register data, which is explained in detail in the *Configuration Register* section. Bytes 5, 6, and 7 are reserved for internal use by the device and cannot be overwritten.

Byte 8 of the scratchpad is read-only and contains the CRC code for bytes 0 through 7 of the scratchpad. The DS18B20 generates this CRC using the method described in the *CRC Generation* section.

Data is written to bytes 2, 3, and 4 of the scratchpad using the Write Scratchpad [4Eh] command; the data must be transmitted to the DS18B20 starting with the least significant bit of byte 2. To verify data integrity, the scratchpad can be read (using the Read Scratchpad [BEh] command) after the data is written. When reading the scratchpad, data is transferred over the 1-Wire bus starting with the least significant bit of byte 0. To transfer the T_H, T_L and configuration data from the scratchpad to EEPROM, the master must issue the Copy Scratchpad [48h] command.

Data in the EEPROM registers is retained when the device is powered down; at power-up the EEPROM data is reloaded into the corresponding scratchpad locations. Data can also be reloaded from EEPROM to the scratchpad at any time using the Recall E^2 [B8h] command. The master can issue read time slots following the Recall E^2 command and the DS18B20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done.

Figure 7. DS18B20 Memory Map



CONFIGURATION REGISTER

Byte 4 of the scratchpad memory contains the configuration register, which is organized as illustrated in Figure 8**.** The user can set the conversion resolution of the DS18B20 using the R0 and R1 bits in this register as shown in Table 2. The power-up default of these bits is R0 = 1 and R1 = 1 (12-bit resolution). Note that there is a direct tradeoff between resolution and conversion time. Bit 7 and bits 0 to 4 in the configuration register are reserved for internal use by the device and cannot be overwritten.

Figure 8. Configuration Register

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

Table 2. Thermometer Resolution Configuration

R1	R0	RESOLUTION (BITS)	MAX CONVERSION TIME	
0	0	9	93.75ms	($t_{CONV}/8$)
0	1	10	187.5ms	($t_{CONV}/4$)
1	0	11	375ms	($t_{CONV}/2$)
1	1	12	750ms	(t_{CONV})

CRC GENERATION

CRC bytes are provided as part of the DS18B20's 64-bit ROM code and in the 9th byte of the scratchpad memory. The ROM code CRC is calculated from the first 56 bits of the ROM code and is contained in the most significant byte of the ROM. The scratchpad CRC is calculated from the data stored in the scratchpad, and therefore it changes when the data in the scratchpad changes. The CRCs provide the bus master with a method of data validation when data is read from the DS18B20. To verify that data has been read correctly, the bus master must re-calculate the CRC from the received data and then compare this value to either the ROM code CRC (for ROM reads) or to the scratchpad CRC (for scratchpad reads). If the calculated CRC matches the read CRC, the data has been received error free. The comparison of CRC values and the decision to continue with an operation are determined entirely by the bus master. There is no circuitry inside the DS18B20 that prevents a command sequence from proceeding if the DS18B20 CRC (ROM or scratchpad) does not match the value generated by the bus master.

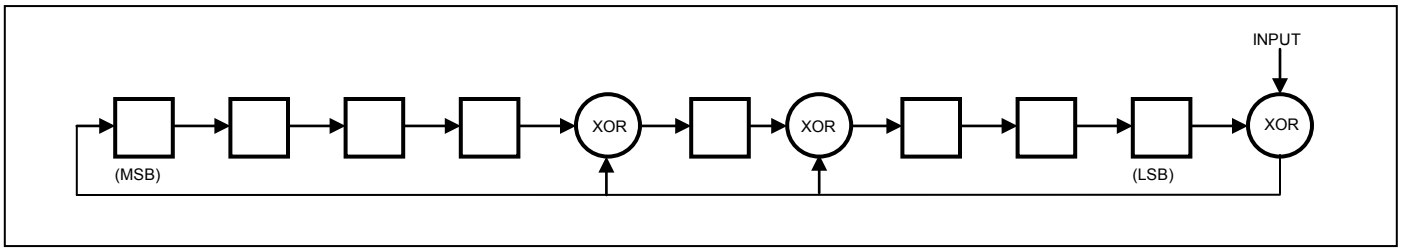
The equivalent polynomial function of the CRC (ROM or scratchpad) is:

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

The bus master can re-calculate the CRC and compare it to the CRC values from the DS18B20 using the polynomial generator shown in Figure 9. This circuit consists of a shift register and XOR gates, and the shift register bits are initialized to 0. Starting with the least significant bit of the ROM code or the least significant bit of byte 0 in the scratchpad, one bit at a time should shifted into the shift register. After shifting in the 56th bit from the ROM or the most significant bit of byte 7 from the scratchpad, the polynomial generator will contain the re-calculated CRC. Next, the 8-bit ROM code or scratchpad CRC from the DS18B20 must be shifted into the circuit. At this point, if the re-calculated CRC was correct, the shift register will contain all 0s. Additional information about the Maxim 1-Wire cyclic redundancy check

is available in *Application Note 27: Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products*.

Figure 9. CRC Generator



1-WIRE BUS SYSTEM

The 1-Wire bus system uses a single bus master to control one or more slave devices. The DS18B20 is always a slave. When there is only one slave on the bus, the system is referred to as a "single-drop" system; the system is "multidrop" if there are multiple slaves on the bus.

All data and commands are transmitted least significant bit first over the 1-Wire bus.

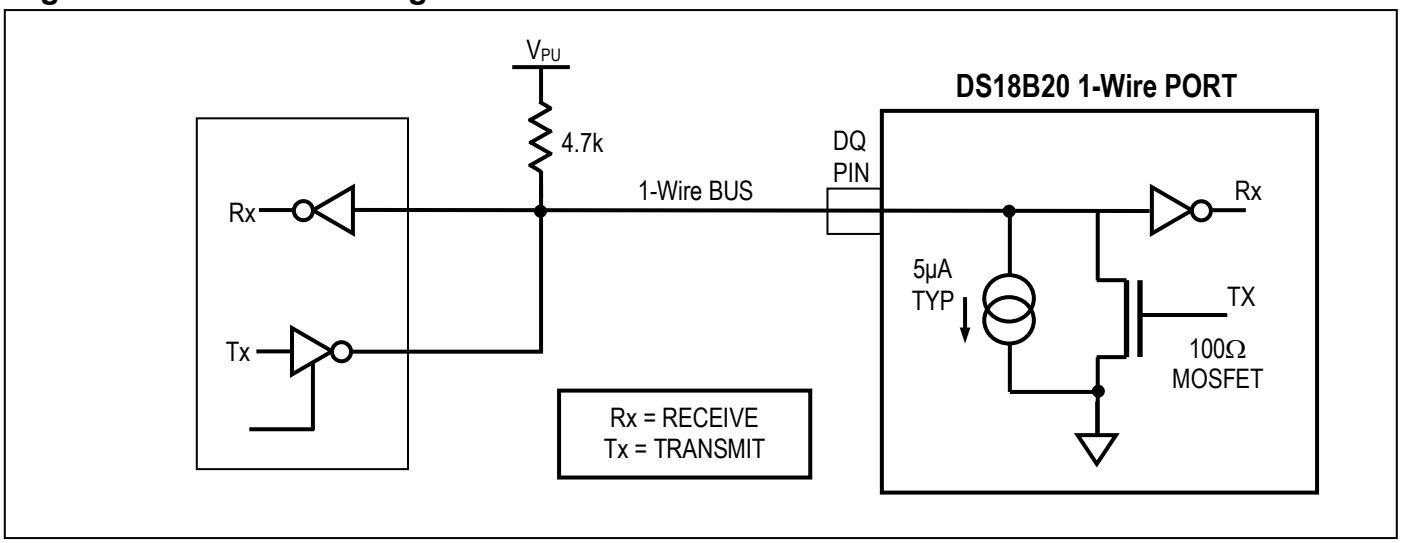
The following discussion of the 1-Wire bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-Wire bus has by definition only a single data line. Each device (master or slave) interfaces to the data line via an open-drain or 3-state port. This allows each device to "release" the data line when the device is not transmitting data so the bus is available for use by another device. The 1-Wire port of the DS18B20 (the DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 10.

The 1-Wire bus requires an external pullup resistor of approximately 5kΩ; thus, the idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480µs, all components on the bus will be reset.

Figure 10. Hardware Configuration



TRANSACTION SEQUENCE

The transaction sequence for accessing the DS18B20 is as follows:

Step 1. Initialization

Step 2. ROM Command (followed by any required data exchange)

Step 3. DS18B20 Function Command (followed by any required data exchange)

It is very important to follow this sequence every time the DS18B20 is accessed, as the DS18B20 will not respond if any steps in the sequence are missing or out of order. Exceptions to this rule are the Search ROM [F0h] and Alarm Search [ECh] commands. After issuing either of these ROM commands, the master must return to Step 1 in the sequence.

INITIALIZATION

All transactions on the 1-Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s). The presence pulse lets the bus master know that slave devices (such as the DS18B20) are on the bus and are ready to operate. Timing for the reset and presence pulses is detailed in the *1-Wire Signaling* section.

ROM COMMANDS

After the bus master has detected a presence pulse, it can issue a ROM command. These commands operate on the unique 64-bit ROM codes of each slave device and allow the master to single out a specific device if many are present on the 1-Wire bus. These commands also allow the master to determine how many and what types of devices are present on the bus or if any device has experienced an alarm condition. There are five ROM commands, and each command is 8 bits long. The master device must issue an appropriate ROM command before issuing a DS18B20 function command. A flowchart for operation of the ROM commands is shown in Figure 11.

SEARCH ROM [F0h]

When a system is initially powered up, the master must identify the ROM codes of all slave devices on the bus, which allows the master to determine the number of slaves and their device types. The master learns the ROM codes through a process of elimination that requires the master to perform a Search ROM cycle (i.e., Search ROM command followed by data exchange) as many times as necessary to identify all of the slave devices. If there is only one slave on the bus, the simpler Read ROM command (see below) can be used in place of the Search ROM process. For a detailed explanation of the Search ROM procedure, refer to the *iButton® Book of Standards* at www.maxim-ic.com/ibuttonbook. After every Search ROM cycle, the bus master must return to Step 1 (Initialization) in the transaction sequence.

READ ROM [33h]

This command can only be used when there is one slave on the bus. It allows the bus master to read the slave's 64-bit ROM code without using the Search ROM procedure. If this command is used when there is more than one slave present on the bus, a data collision will occur when all the slaves attempt to respond at the same time.

MATCH ROM [55h]

The match ROM command followed by a 64-bit ROM code sequence allows the bus master to address a specific slave device on a multidrop or single-drop bus. Only the slave that exactly matches the 64-bit ROM code sequence will respond to the function command issued by the master; all other slaves on the bus will wait for a reset pulse.

SKIP ROM [CCh]

The master can use this command to address all devices on the bus simultaneously without sending out any ROM code information. For example, the master can make all DS18B20s on the bus perform simultaneous temperature conversions by issuing a Skip ROM command followed by a Convert T [44h] command.

Note that the Read Scratchpad [BEh] command can follow the Skip ROM command only if there is a single slave device on the bus. In this case, time is saved by allowing the master to read from the slave without sending the device's 64-bit ROM code. A Skip ROM command followed by a Read Scratchpad command will cause a data collision on the bus if there is more than one slave since multiple devices will attempt to transmit data simultaneously.

ALARM SEARCH [ECh]

The operation of this command is identical to the operation of the Search ROM command except that only slaves with a set alarm flag will respond. This command allows the master device to determine if any DS18B20s experienced an alarm condition during the most recent temperature conversion. After every Alarm Search cycle (i.e., Alarm Search command followed by data exchange), the bus master must return to Step 1 (Initialization) in the transaction sequence. See the *Operation—Alarm Signaling* section for an explanation of alarm flag operation.

DS18B20 FUNCTION COMMANDS

After the bus master has used a ROM command to address the DS18B20 with which it wishes to communicate, the master can issue one of the DS18B20 function commands. These commands allow the master to write to and read from the DS18B20's scratchpad memory, initiate temperature conversions and determine the power supply mode. The DS18B20 function commands, which are described below, are summarized in Table 3 and illustrated by the flowchart in Figure 12.

CONVERT T [44h]

This command initiates a single temperature conversion. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its low-power idle state. If the device is being used in parasite power mode, within 10µs (max) after this command is issued the master must enable a strong pullup on the 1-Wire bus for the duration of the conversion (t_{CONV}) as described in the *Powering the DS18B20* section. If the DS18B20 is powered by an external supply, the master can issue read time slots after the Convert T command and the DS18B20 will respond by transmitting a 0 while the temperature conversion is in progress and a 1 when the conversion is done. In parasite power mode this notification technique cannot be used since the bus is pulled high by the strong pullup during the conversion.

WRITE SCRATCHPAD [4Eh]

This command allows the master to write 3 bytes of data to the DS18B20's scratchpad. The first data byte is written into the T_H register (byte 2 of the scratchpad), the second byte is written into the T_L register (byte 3), and the third byte is written into the configuration register (byte 4). Data must be transmitted least significant bit first. All three bytes MUST be written before the master issues a reset, or the data may be corrupted.

READ SCRATCHPAD [BEh]

This command allows the master to read the contents of the scratchpad. The data transfer starts with the least significant bit of byte 0 and continues through the scratchpad until the 9th byte (byte 8 – CRC) is read. The master may issue a reset to terminate reading at any time if only part of the scratchpad data is needed.

COPY SCRATCHPAD [48h]

This command copies the contents of the scratchpad T_H, T_L and configuration registers (bytes 2, 3 and 4) to EEPROM. If the device is being used in parasite power mode, within 10µs (max) after this command is issued the master must enable a strong pullup on the 1-Wire bus for at least 10ms as described in the *Powering the DS18B20* section.

RECALL E^2 [B8h]

This command recalls the alarm trigger values (T_H and T_L) and configuration data from EEPROM and places the data in bytes 2, 3, and 4, respectively, in the scratchpad memory. The master device can issue read time slots following the Recall E^2 command and the DS18B20 will indicate the status of the recall by transmitting 0 while the recall is in progress and 1 when the recall is done. The recall operation happens automatically at power-up, so valid data is available in the scratchpad as soon as power is applied to the device.

READ POWER SUPPLY [B4h]

The master device issues this command followed by a read time slot to determine if any DS18B20s on the bus are using parasite power. During the read time slot, parasite powered DS18B20s will pull the bus low, and externally powered DS18B20s will let the bus remain high. See the *Powering the DS18B20* section for usage information for this command.

Table 3. DS18B20 Function Command Set

COMMAND	DESCRIPTION	PROTOCOL	1-Wire BUS ACTIVITYAFTER COMMAND IS ISSUED	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	DS18B20 transmits conversion status to master (not applicable for parasite-powered DS18B20s).	1
MEMORY COMMANDS				
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18B20 transmits up to 9 data bytes to master.	2
Write Scratchpad	Writes data into scratchpad bytes 2, 3, and 4 (T_H, T_L, and configuration registers).	4Eh	Master transmits 3 data bytes to DS18B20.	3
Copy Scratchpad	Copies T_H, T_L, and configuration register data from the scratchpad to EEPROM.	48h	None	1
Recall E^2	Recalls T_H, T_L, and configuration register data from EEPROM to the scratchpad.	B8h	DS18B20 transmits recall status to master.	
Read Power Supply	Signals DS18B20 power supply mode to the master.	B4h	DS18B20 transmits supply status to master.	

Note 1: For parasite-powered DS18B20s, the master must enable a strong pullup on the 1-Wire bus during temperature conversions and copies from the scratchpad to EEPROM. No other bus activity may take place during this time.
Note 2: The master can interrupt the transmission of data at any time by issuing a reset.
Note 3: All three bytes must be written before a reset is issued.

Figure 11. ROM Commands Flowchart

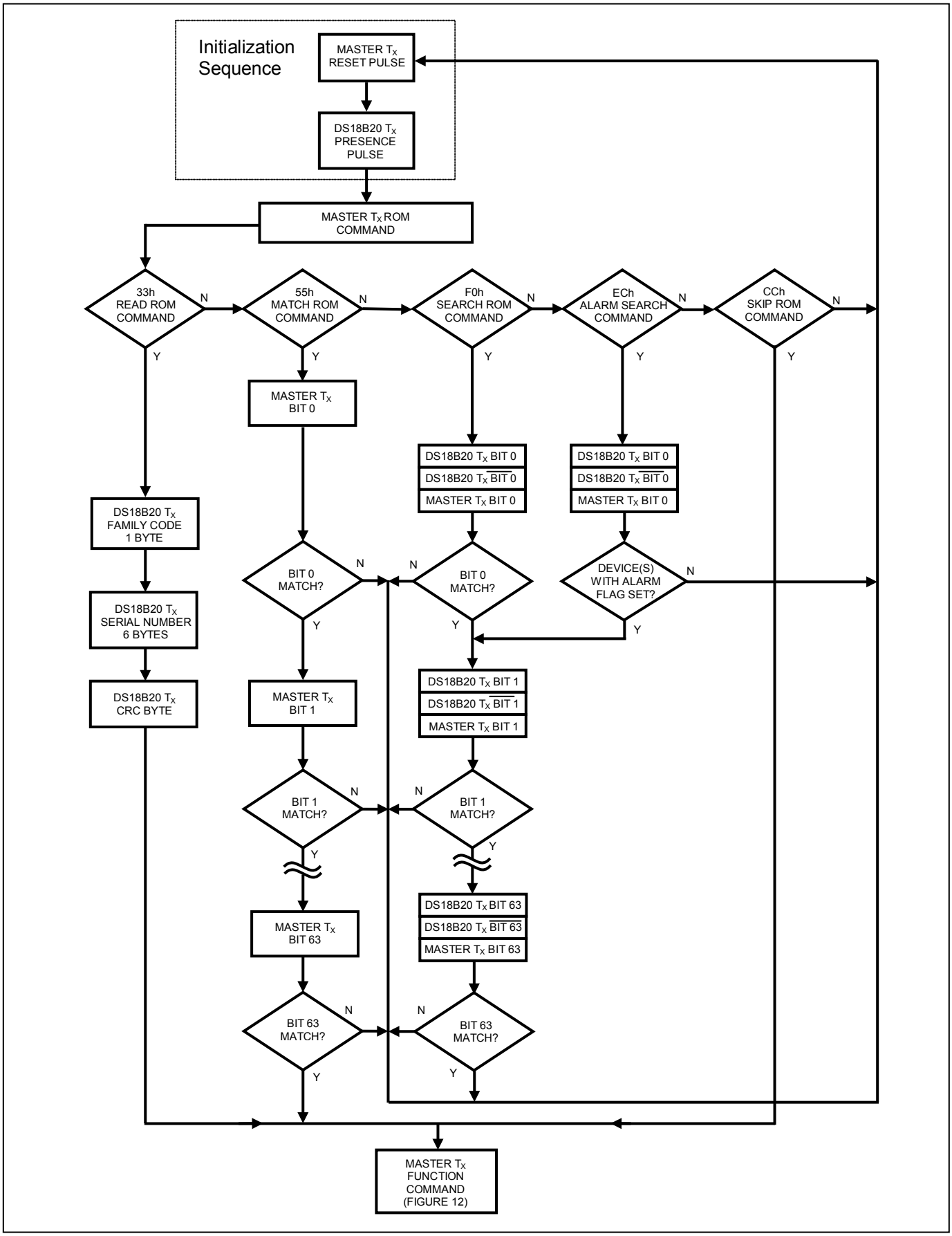
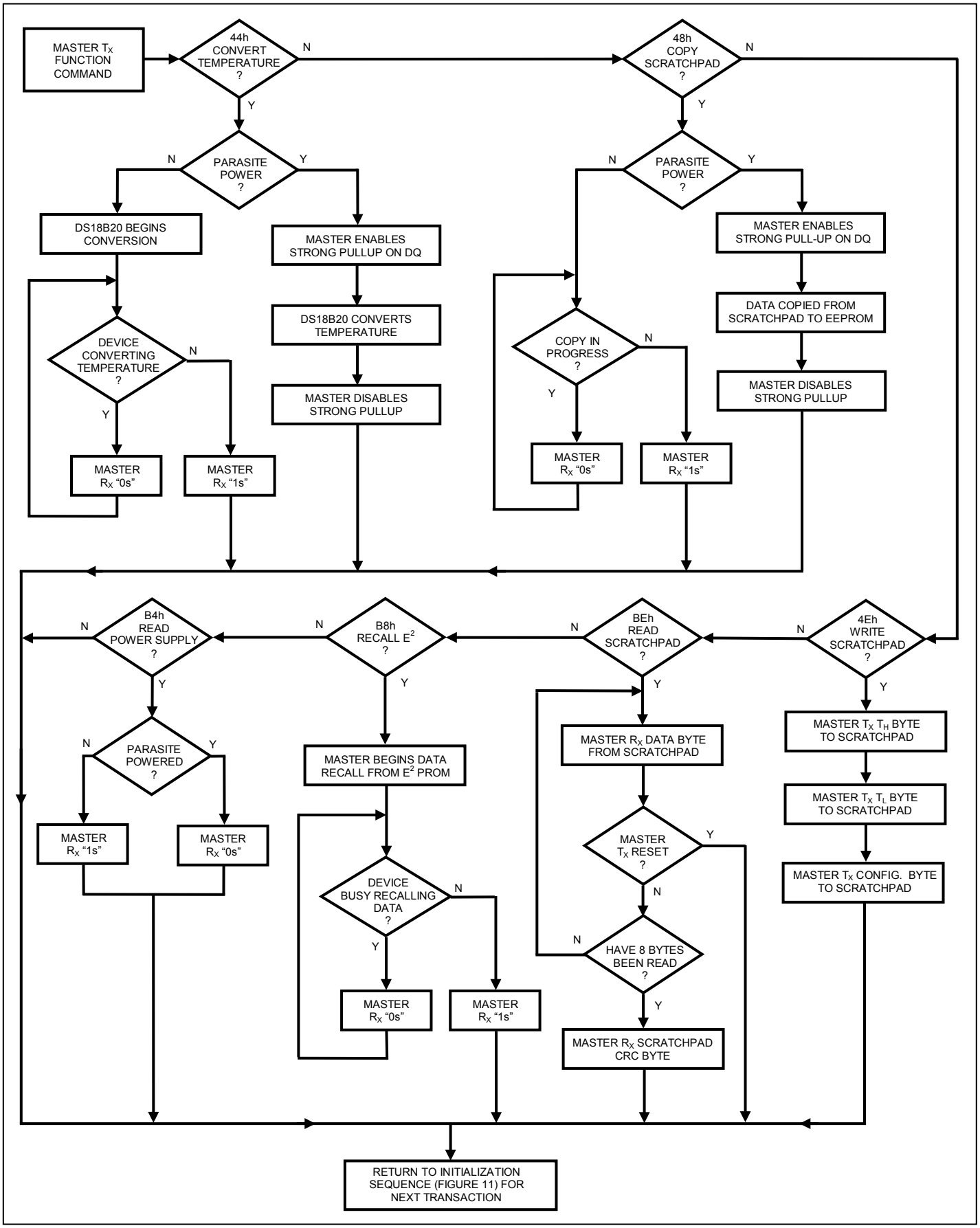


Figure 12. DS18B20 Function Commands Flowchart



1-WIRE SIGNALING

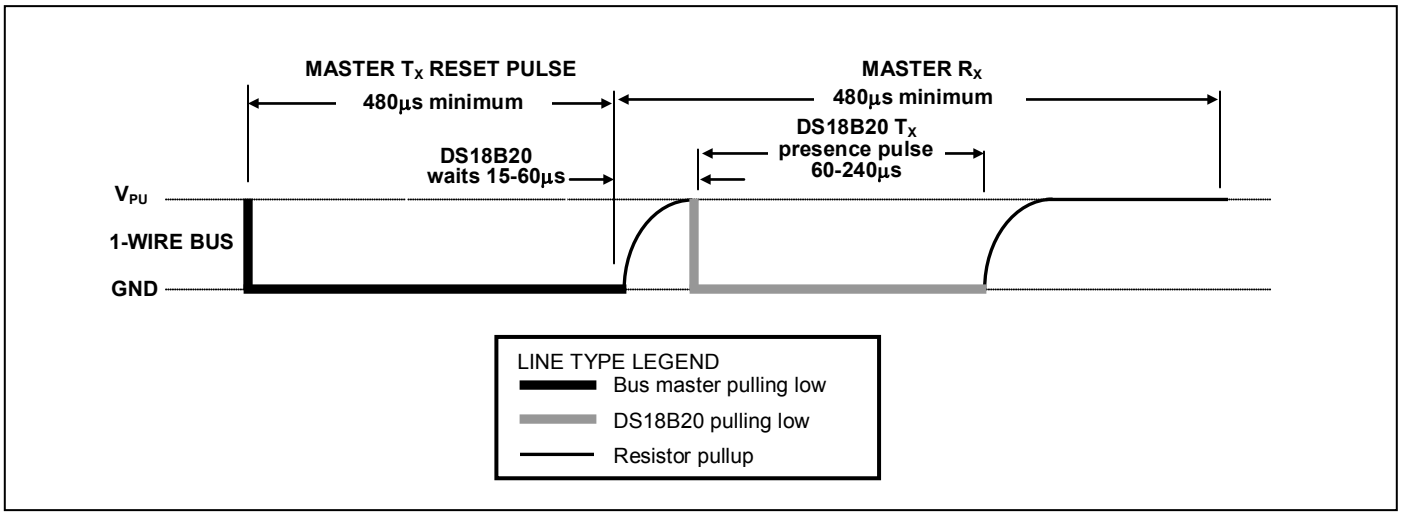
The DS18B20 uses a strict 1-Wire communication protocol to ensure data integrity. Several signal types are defined by this protocol: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. The bus master initiates all these signals, with the exception of the presence pulse.

INITIALIZATION PROCEDURE—RESET AND PRESENCE PULSES

All communication with the DS18B20 begins with an initialization sequence that consists of a reset pulse from the master followed by a presence pulse from the DS18B20. This is illustrated in Figure 13. When the DS18B20 sends the presence pulse in response to the reset, it is indicating to the master that it is on the bus and ready to operate.

During the initialization sequence the bus master transmits (T_X) the reset pulse by pulling the 1-Wire bus low for a minimum of 480µs. The bus master then releases the bus and goes into receive mode (R_X). When the bus is released, the 5kΩ pullup resistor pulls the 1-Wire bus high. When the DS18B20 detects this rising edge, it waits 15µs to 60µs and then transmits a presence pulse by pulling the 1-Wire bus low for 60µs to 240µs.

Figure 13. Initialization Timing



READ/WRITE TIME SLOTS

The bus master writes data to the DS18B20 during write time slots and reads data from the DS18B20 during read time slots. One bit of data is transmitted over the 1-Wire bus per time slot.

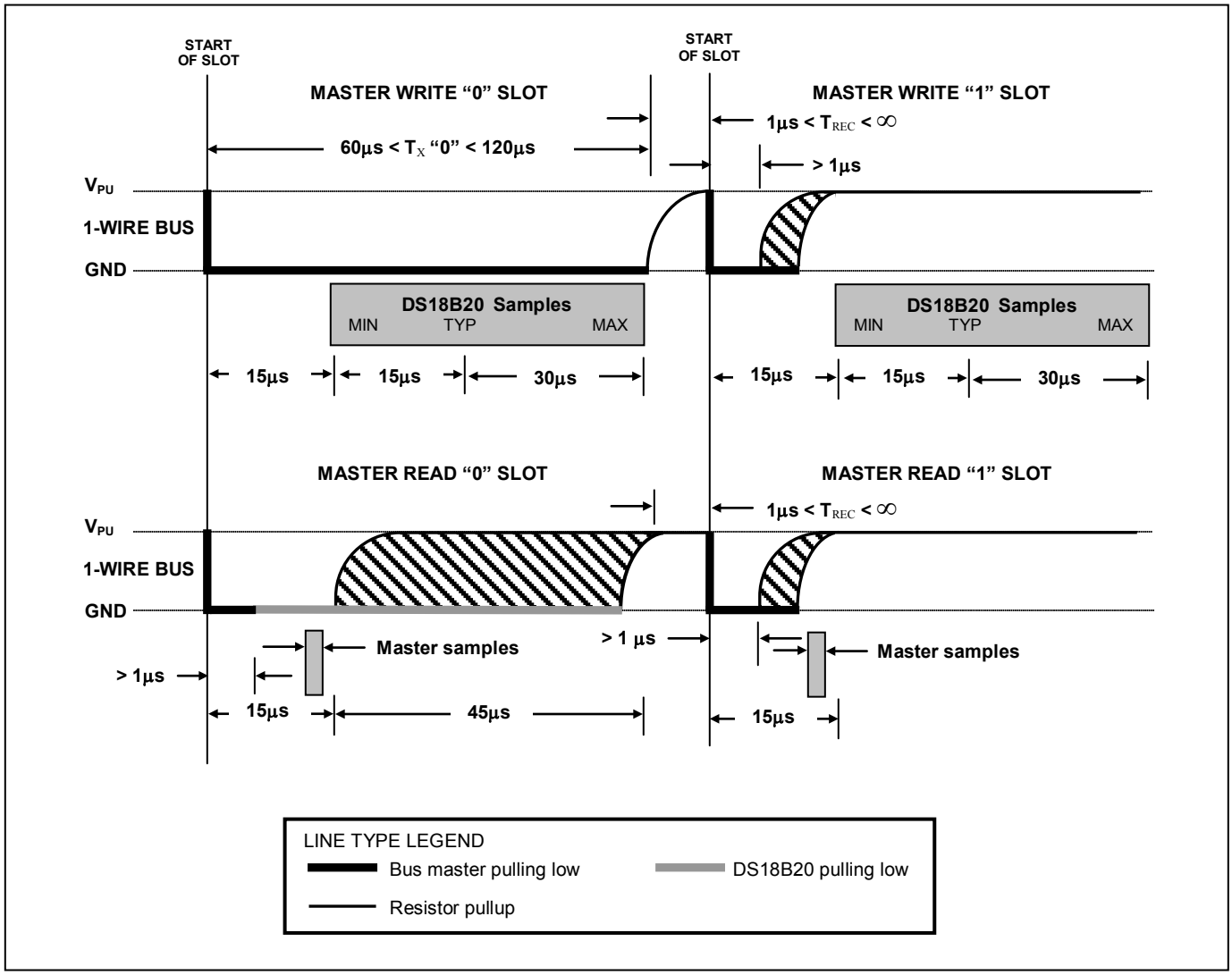
WRITE TIME SLOTS

There are two types of write time slots: “Write 1” time slots and “Write 0” time slots. The bus master uses a Write 1 time slot to write a logic 1 to the DS18B20 and a Write 0 time slot to write a logic 0 to the DS18B20. All write time slots must be a minimum of 60µs in duration with a minimum of a 1µs recovery time between individual write slots. Both types of write time slots are initiated by the master pulling the 1-Wire bus low (see Figure 14).

To generate a Write 1 time slot, after pulling the 1-Wire bus low, the bus master must release the 1-Wire bus within 15µs. When the bus is released, the 5kΩ pullup resistor will pull the bus high. To generate a Write 0 time slot, after pulling the 1-Wire bus low, the bus master must continue to hold the bus low for the duration of the time slot (at least 60µs).

The DS18B20 samples the 1-Wire bus during a window that lasts from 15µs to 60µs after the master initiates the write time slot. If the bus is high during the sampling window, a 1 is written to the DS18B20. If the line is low, a 0 is written to the DS18B20.

Figure 14. Read/Write Time Slot Timing Diagram



READ TIME SLOTS

The DS18B20 can only transmit data to the master when the master issues read time slots. Therefore, the master must generate read time slots immediately after issuing a Read Scratchpad [BEh] or Read Power Supply [B4h] command, so that the DS18B20 can provide the requested data. In addition, the master can generate read time slots after issuing Convert T [44h] or Recall E^2 [B8h] commands to find out the status of the operation as explained in the *DS18B20 Function Commands* section.

All read time slots must be a minimum of 60µs in duration with a minimum of a 1µs recovery time between slots. A read time slot is initiated by the master device pulling the 1-Wire bus low for a minimum of 1µs and then releasing the bus (see Figure 14). After the master initiates the read time slot, the DS18B20 will begin transmitting a 1 or 0 on bus. The DS18B20 transmits a 1 by leaving the bus high and transmits a 0 by pulling the bus low. When transmitting a 0, the DS18B20 will release the bus by the end of the time slot, and the bus will be pulled back to its high idle state by the pullup resister. Output

data from the DS18B20 is valid for 15µs after the falling edge that initiated the read time slot. Therefore, the master must release the bus and then sample the bus state within 15µs from the start of the slot.

Figure 15 illustrates that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15µs for a read time slot. Figure 16 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as short as possible and by locating the master sample time during read time slots towards the end of the 15µs period.

Figure 15. Detailed Master Read 1 Timing

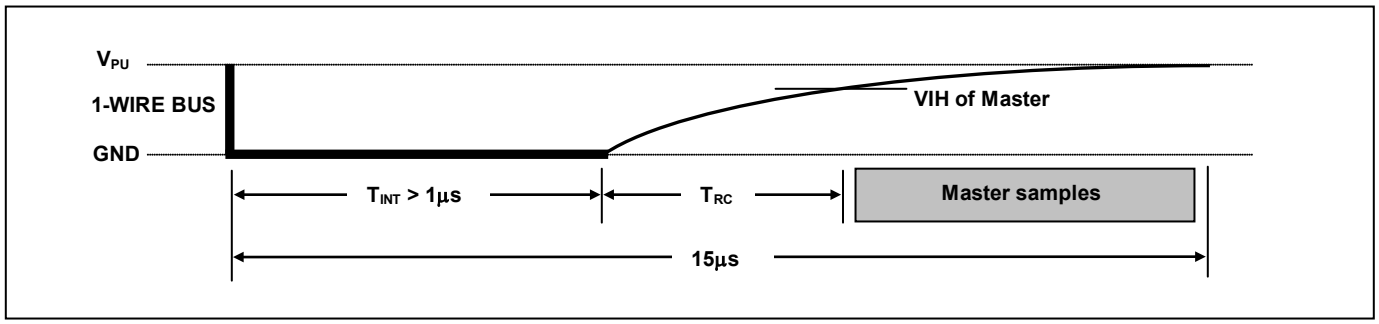
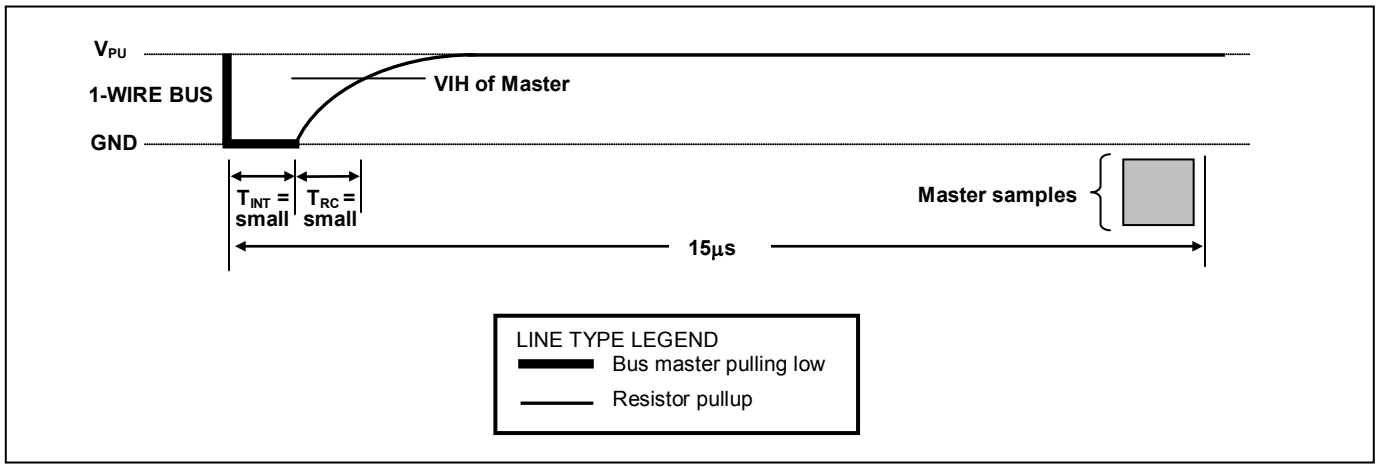


Figure 16. Recommended Master Read 1 Timing



RELATED APPLICATION NOTES

The following application notes can be applied to the DS18B20 and are available on our website at www.maxim-ic.com.

Application Note 27: Understanding and Using Cyclic Redundancy Checks with Maxim iButton Products
Application Note 122: Using Dallas' 1-Wire ICs in 1-Cell Li-Ion Battery Packs with Low-Side N-Channel Safety FETs Master
Application Note 126: 1-Wire Communication Through Software
Application Note 162: Interfacing the DS18x20/DS1822 1-Wire Temperature Sensor in a Microcontroller Environment
Application Note 208: Curve Fitting the Error of a Bandgap-Based Digital Temperature Sensor
Application Note 2420: 1-Wire Communication with a Microchip PICmicro Microcontroller
Application Note 3754: Single-Wire Serial Bus Carries Isolated Power and Data

Sample 1-Wire subroutines that can be used in conjunction with *Application Note 74: Reading and Writing iButtons via Serial Interfaces* can be downloaded from the Maxim website.

DS18B20 OPERATION EXAMPLE 1

In this example there are multiple DS18B20s on the bus and they are using parasite power. The bus master initiates a temperature conversion in a specific DS18B20 and then reads its scratchpad and recalculates the CRC to verify the data.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20s respond with presence pulse.
Tx	55h	Master issues Match ROM command.
Tx	64-bit ROM code	Master sends DS18B20 ROM code.
Tx	44h	Master issues Convert T command.
Tx	DQ line held high by strong pullup	Master applies strong pullup to DQ for the duration of the conversion (t_{CONV}).
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20s respond with presence pulse.
Tx	55h	Master issues Match ROM command.
Tx	64-bit ROM code	Master sends DS18B20 ROM code.
Tx	BEh	Master issues Read Scratchpad command.
Rx	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.

DS18B20 OPERATION EXAMPLE 2

In this example there is only one DS18B20 on the bus and it is using parasite power. The master writes to the T_H, T_L, and configuration registers in the DS18B20 scratchpad and then reads the scratchpad and recalculates the CRC to verify the data. The master then copies the scratchpad contents to EEPROM.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	4Eh	Master issues Write Scratchpad command.
Tx	3 data bytes	Master sends three data bytes to scratchpad (T_H, T_L, and config).
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	BEh	Master issues Read Scratchpad command.
Rx	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.
Tx	Reset	Master issues reset pulse.
Rx	Presence	DS18B20 responds with presence pulse.
Tx	CCh	Master issues Skip ROM command.
Tx	48h	Master issues Copy Scratchpad command.
Tx	DQ line held high by strong pullup	Master applies strong pullup to DQ for at least 10ms while copy operation is in progress.

ABSOLUTE MAXIMUM RATINGS

Voltage Range on Any Pin Relative to Ground .....-0.5V to +6.0V
Operating Temperature Range .....-55°C to +125°C
Storage Temperature Range.....-55°C to +125°C
Solder Temperature .....Refer to the IPC/JEDEC J-STD-020 Specification.

These are stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	+3.0		+5.5	V	1
Pullup Supply Voltage	V_{PU}	Parasite Power	+3.0		+5.5	V	1,2
		Local Power	+3.0		V_{DD}		
Thermometer Error	t_{ERR}	-10°C to +85°C			±0.5	°C	3
		-55°C to +125°C			±2		
Input Logic-Low	V_{IL}		-0.3		+0.8	V	1,4,5
Input Logic-High	V_{IH}	Local Power	+2.2		The lower of 5.5 or V_{DD} + 0.3	V	1, 6
		Parasite Power	+3.0				
Sink Current	I_L	$V_{I/O}$ = 0.4V	4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	7,8
Active Current	I_{DD}	V_{DD} = 5V		1	1.5	mA	9
DQ Input Current	I_{DQ}			5		µA	10
Drift				±0.2		°C	11

NOTES:

1) All voltages are referenced to ground.
2) The Pullup Supply Voltage specification assumes that the pullup device is ideal, and therefore the high level of the pullup is equal to V_{PU}. In order to meet the V_{IH} spec of the DS18B20, the actual supply rail for the strong pullup transistor must include margin for the voltage drop across the transistor when it is turned on; thus: $V_{PU\_ACTUAL} = V_{PU\_IDEAL} + V_{TRANSISTOR}$.
3) See typical performance curve in Figure 17.
4) Logic-low voltages are specified at a sink current of 4mA.
5) To guarantee a presence pulse under low voltage parasite power conditions, V_{ILMAX} may have to be reduced to as low as 0.5V.
6) Logic-high voltages are specified at a source current of 1mA.
7) Standby current specified up to +70°C. Standby current typically is 3µA at +125°C.
8) To minimize I_{DDS}, DQ should be within the following ranges: GND ≤ DQ ≤ GND + 0.3V or V_{DD} – 0.3V ≤ DQ ≤ V_{DD}.
9) Active current refers to supply current during active temperature conversions or EEPROM writes.
10) DQ line is high (“high-Z” state).
11) Drift data is based on a 1000-hour stress test at +125°C with V_{DD} = 5.5V.

AC ELECTRICAL CHARACTERISTICS—NV MEMORY

(-55°C to +100°C; V_{DD} = 3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
NV Write Cycle Time	t_{WR}			2	10	ms
EEPROM Writes	N_{EEWR}	-55°C to +55°C	50k			writes
EEPROM Data Retention	t_{EEDR}	-55°C to +55°C	10			years

AC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD} = 3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}	9-bit resolution			93.75	ms	1
		10-bit resolution			187.5		
		11-bit resolution			375		
		12-bit resolution			750		
Time to Strong Pullup On	t_{SPON}	Start Convert T Command Issued			10	µs	
Time Slot	t_{SLOT}		60		120	µs	1
Recovery Time	t_{REC}		1			µs	1
Write 0 Low Time	t_{LOW0}		60		120	µs	1
Write 1 Low Time	t_{LOW1}		1		15	µs	1
Read Data Valid	t_{RDV}				15	µs	1
Reset Time High	t_{RSTH}		480			µs	1
Reset Time Low	t_{RSTL}		480			µs	1,2
Presence-Detect High	t_{PDHIGH}		15		60	µs	1
Presence-Detect Low	t_{PDLOW}		60		240	µs	1
Capacitance	$C_{IN/OUT}$				25	pF	

NOTES:

1) See the timing diagrams in Figure 18.
2) Under parasite power, if t_{RSTL} > 960µs, a power-on reset may occur.

Figure 17. Typical Performance Curve

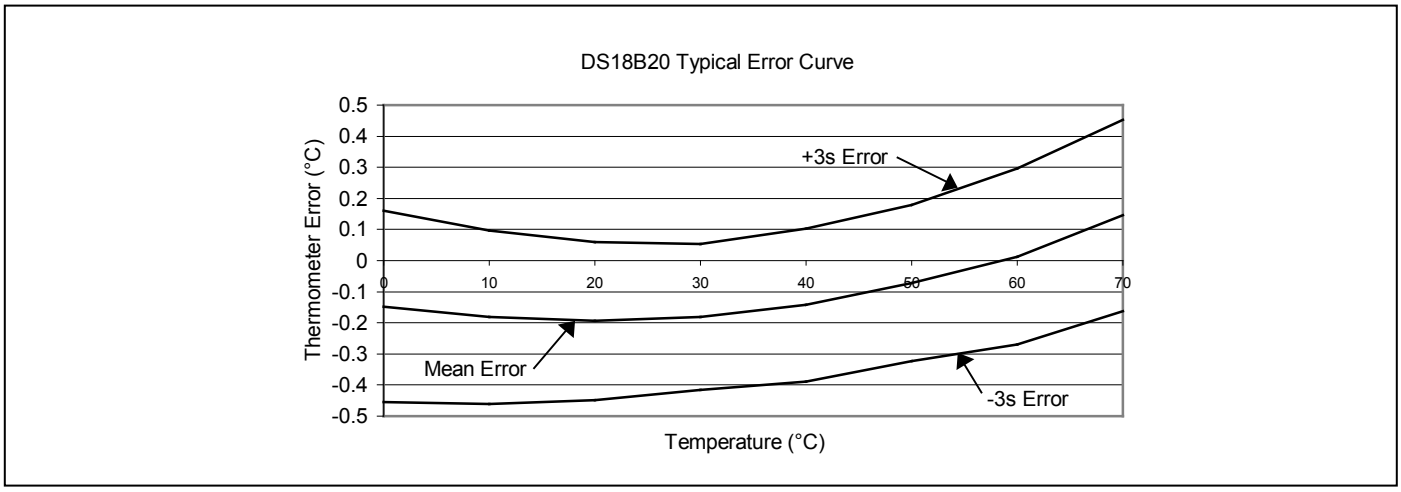
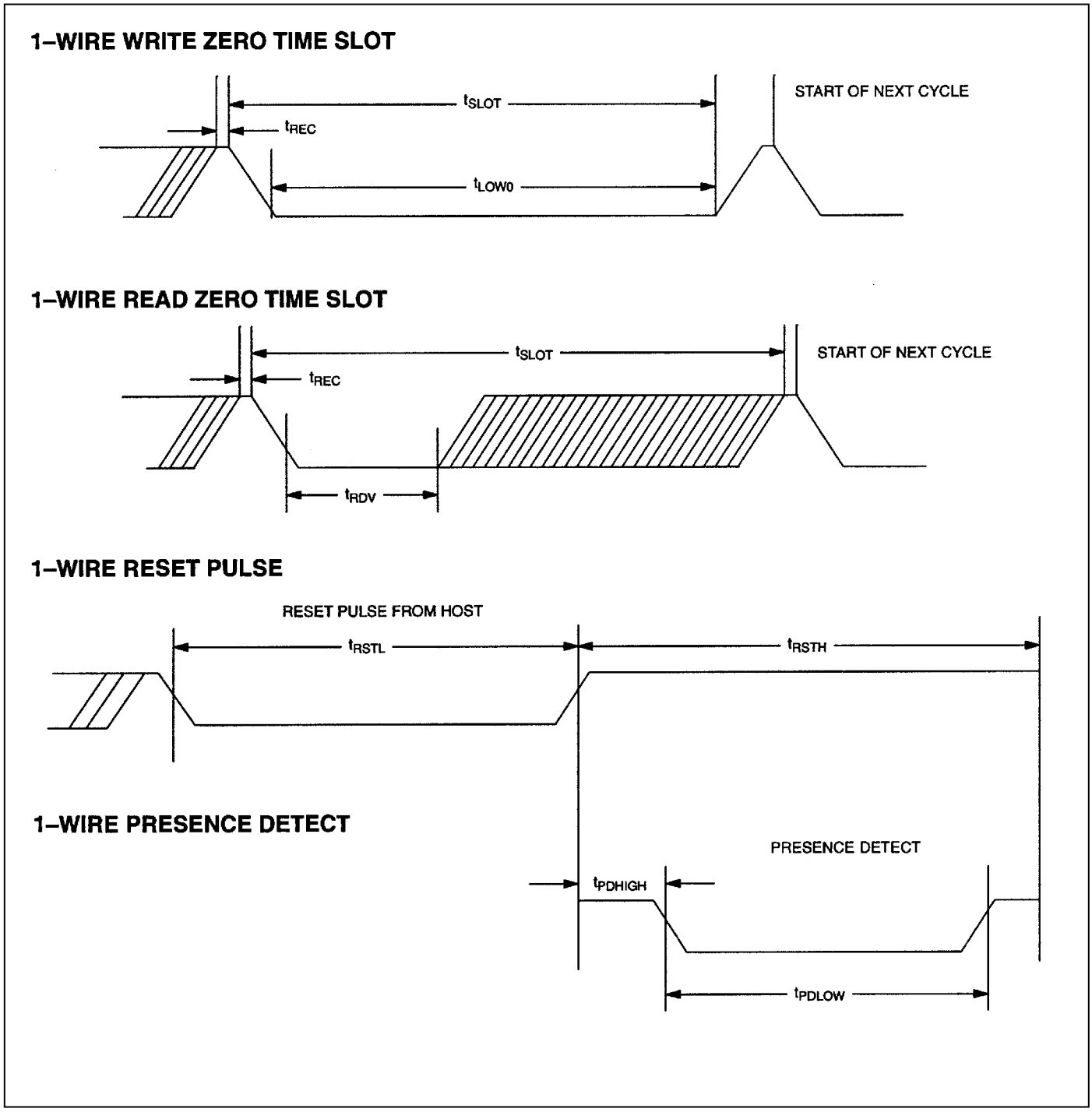


Figure 18. Timing Diagrams



REVISION HISTORY

REVISION DATE	DESCRIPTION	PAGES CHANGED
030107	In the *Absolute Maximum Ratings* section, removed the reflow oven temperature value of +220°C. Reference to JEDEC specification for reflow remains.	19
101207	In the *Operation—Alarm Signaling* section, added “or equal to” in the desciption for a TH alarm condition	5
	In the *Memory* section, removed incorrect text describing memory.	7
	In the *Configuration Register* section, removed incorrect text describing configuration register.	8
042208	In the *Ordering Information* table, added TO-92 straight-lead packages and included a note that the TO-92 package in tape and reel can be ordered with either formed or straight leads.	2

FOTORRESISTENCIA LDR 4,3mm x Ø 5,1mm

C-2795

DESCRIPCION GENERAL.

Fotorresistencia o resistencia dependiente de la luz, consistente en una célula de Sulfuro de Cadmio, altamente estable, encapsulada con una resina epoxi transparente, resitente a la humedad. La respuesta espectral es similar a la del ojo humano. Su nivel de resistencia aumenta cuando el nivel de luz disminuye.

Aplicaciones: Control de contraste en televisores y monitores, control automático de la iluminación en habitaciones, juguetes y juegos electrónicos, controles industriales, interruptores crepusculares, boyas y balizas de encendido automático, auto-flash, etc...

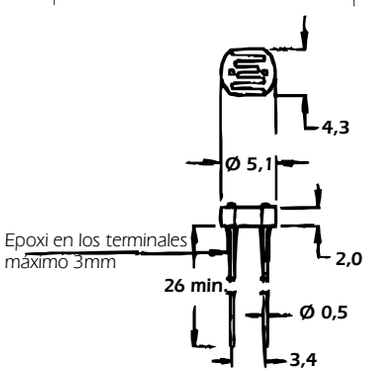
CARACTERÍSTICAS TÉCNICAS.

Modelo	Valores máximos			Características a 25ºC (nota E)						
	Tensión a 25ºC	Potencia disipable	Temperatura ambiente	Resistencia (notaA)			? (notaC)	Tiempos de respuesta a 10 lx (notaD)		Respuesta espectral
				10 lux (2856K)		0 lux (notaB)				
	(Vdc)	(mW)	(ºC)	Min.(k?)	Max.(k?)	Min.(M?)	100-10 lx (M?)	t. subida (ms)	t. bajada (ms)	(pico) (nm)
C-2795	**150**	**90**	**-25 a 75**	**50**	**140**	**20**	**0.9**	**60**	**25**	**570**

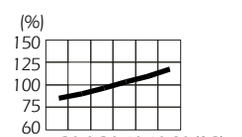
Notas: **A)** Medido con una fuente luminosa formada por una lámpara de tungsteno, trabajando a una temperatura de color de 2856K.
B) Medición efectuada 10 segundos después de retirar una iluminación incidente de 10 lux.
C) Sensibilidad entre 10 y 100 lux, dada por:

$$? = \frac{\log(R100)-\log(R10)}{\log(E100)-\log(E10)}$$

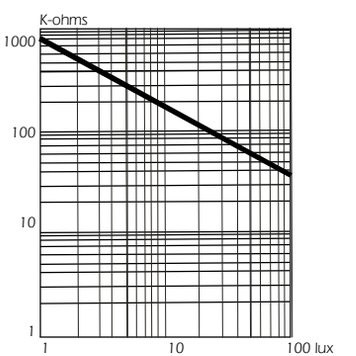
donde R100, R10 son las resistencias a 100 y 10 lux respectivamente, y E100, E10 las iluminancias de 100 y 10 lx respectivamente.
D) Tiempo de subida es el tiempo necesario para alcanzar el 63% del nivel de saturación. Tiempo de bajada es el necesario para que la célula alcance el 37% desde el nivel saturación.
E) Todas las características están medidas con la célula LDR expuesta a la luz (100-500 lux) durante 1 o 2 horas.



Cotas en milímetros



Variación de la resistencia de la célula iluminada en función de la temperatura



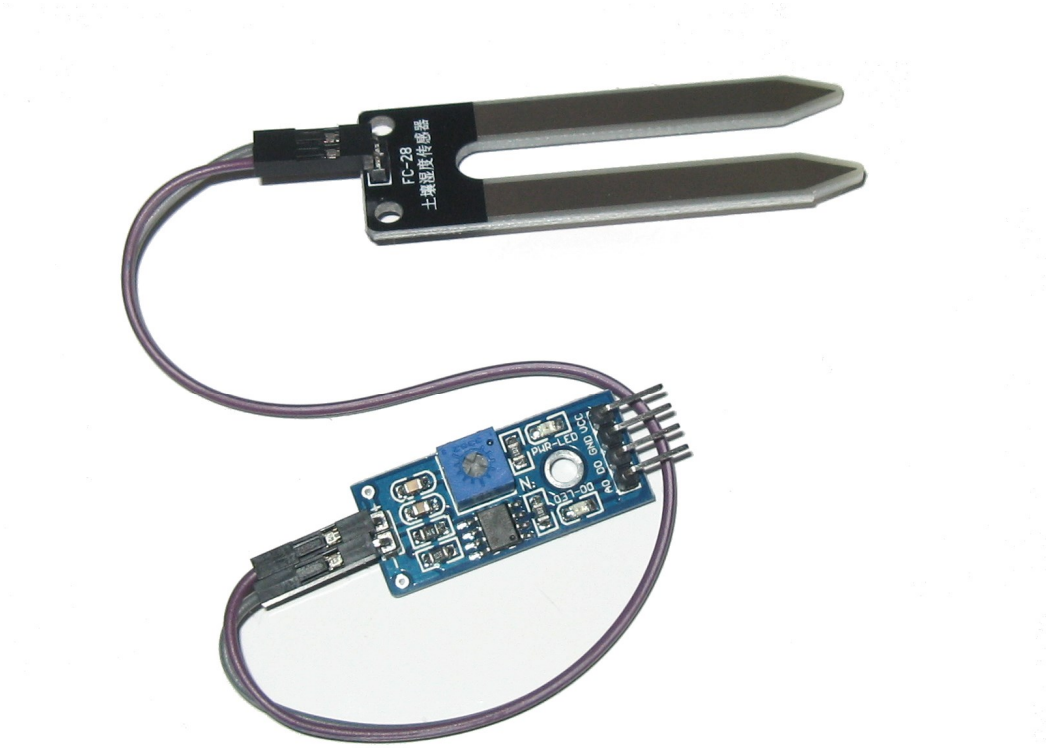
Resistencia de la célula en función de la iluminancia

CONSIDERACIONES.

Este componente está destinado para su uso por parte de profesionales, o usuarios con un nivel técnico o conocimientos suficientes, que les permita desarrollar por sí mismos los proyectos o aplicaciones deseados. Por este motivo no se facilitará asistencia técnica sobre problemas de implementación del citado componente en las aplicaciones en las que sea empleado.
Para cualquier problema relativo al funcionamiento del producto (excluidos los problemas de aplicación), póngase en contacto con nuestro **departamento técnico**. **Fax 93 432 29 95.**
Correo electrónico: sat@fadisel.com. La documentación técnica de este producto responde a una transcripción de la proporcionada por el fabricante.
Los productos de la família "Componentes" de Cebek disponen de **1 año de garantía** a partir de la fecha de compra. Quedan excluidos el trato o manipulación incorrectos.
Disponemos de más productos que pueden interesarle, visítenos en: **www.fadisel.com** ó **SOLICITE GRATUITAMENTE nuestro catálogo.**

FC-28

Sensor de humedad del suelo



Voltaje de 3,3 a 5V cc
Salidas analógica y comparadora
Ajuste de sensibilidad
Dimensiones del sensor 6x-20mm contactos 45mm
Dimensión del comparador 30x14mm

Los cuatro conectores son

VCC positivo de 3,3 a 5V CC
GND masa
OD salida de señal digital (a una determinada humedad en el sensor se dispara) se puede ajustar con el potenciómetro el nivel a controlar
OA salida de señal analógica a cambia la tensión de salida en función de la cantidad de humedad que registra el sensor

ANEXO D

CÓDIGO FUENTE DEL CONTROLADOR

```
//librerias a utilizar

#include <LiquidCrystal_I2C.h>

#include <Wire.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include <SPI.h>

#include <Ethernet.h>

#include <EthernetUdp.h>

#include <String.h>

#include <stdlib.h>


//Constantes Globales

byte mac[] = { 0x98, 0x4F, 0xEE, 0x01, 0x5B, 0x2D };  // MAC Arduino

byte ip[] = { 192,168,1,8};  // IP Arduino

byte remoteIp[] = { 192,168,1,3 };  // IP Servidor

#define localPort 5050U  // Remote UDP Test Tool

#define remotePort 5051U  // Host UDP Test Tool


//Puertos Digitales

#define Temperatura 2  //Pin para el sensor de Temperatura(pin2 Digital)

#define switchModo 3  // Modo Remoto(Automatico o Manual)(HIGH) o Modo Local(Automatico o Manual)(LOW)

#define switchOperacion 4  // Modo Operacion (Automatico(HIGH) o Manual(LOW)) de forma Local.

#define switchRefrigeracion 5  //Switch para Activar Refrigeracion(solo funciona en Modo de Operacion Manual Local).
```

```
#define switchVentilacion 6  //Switch para Activar Ventilacion(solo funciona en Modo de Operacion Manual Local).

#define switchValvula 7  //Switch para Activar Riego(solo funciona en Modo de Operacion Manual Local).

#define Refrigeracion 8  //Salida cuando se activa la Refrigeracion.

#define Ventilacion 9  //Salida cuando se activa la Ventilacion.


//Puertos Analogicos

#define Humedad A0  //Pin para el sensor de Humedad.

#define Luminosidad A1  //Pin Para el sensor de Luminosidad.

#define LuzArtificial A2  //Salida cuando se activa la Luz Artificial.

#define Riego A3  //Salida cuando se activa el Riego.


//Para Buffer de Paquetes UDP

#define PACKET_SIZE 12

#define MAX_SIZE 16


//Variables para Obtener y Setear Estado a las Salidas

int estadoRef=0;

int estadoVen=0;

int estadoVal=0;

int estadoLuz=0;


OneWire ds(Temperatura);
```

```
DallasTemperature sensors(&ds);

DeviceAddress Temperatura1;

EthernetUDP Udp;  //Variable tipo Udp

LiquidCrystal_I2C lcd(0x27,20,4);  //Variable tipo LiquidCrystal_I2C para la LCD por comunicacion I2C


//Declaramos el modo de los Puertos y la Informacion de Sensores

void setup(){

  sensors.begin();  // se inicia comunicacion con los sensores

  pinMode(switchModo,INPUT);  // switchModo es declarado puerto de Entrada

  pinMode(switchOperacion,INPUT);  // switchModo es declarado puerto de Entrada

  pinMode(switchRefrigeracion,INPUT);  // switchModo es declarado puerto de Entrada

  pinMode(switchVentilacion,INPUT);  // switchModo es declarado puerto de Entrada

  pinMode(switchValvula,INPUT);  // switchModo es declarado puerto de Entrada

  pinMode(Refrigeracion,OUTPUT);  //Refrigeracion es declarado puerto de Salida

  pinMode(Ventilacion,OUTPUT);  //Ventilacion es declarado puerto de Salida

  pinMode(Riego,OUTPUT);  //Riego es declarado puerto de Salida

  pinMode(LuzArtificial,OUTPUT);  //LuzArtificial es declarado puerto de Salida

  Ethernet.begin(mac,ip);  // Se inicia comunicacion Ethernet

  Udp.begin(localPort);  // Se inicia comunicacion UDP

  lcd.init();  //Se inicia la comunicacion con la LCD

  lcd.begin(20,4);  //Se inicia la comunicacion con la LCD de 20x4

}


//Iniciamos el Programa Principal
```

```
void loop(){

  LeerPaquetes();  //Funcion que me Envia y Recibe paquetes UDP(Remoto), ademas de trabajar de manera Local(Automatica o Manual).

}


//Funcion LeerTemperatura

float LeerTemperatura(){

  float TemperaturaA= 0;  //Variable tipo float

  sensors.requestTemperatures();  //Aqui pedimos al sensor que nos envie la temperatura

  TemperaturaA = sensors.getTempCByIndex(0);  //Aqui obtenemos la temperatura del primer sensor de temperatura que encuentre

  return TemperaturaA;  //retorna el valor sensado

}


//Funcion LeerHumedad

int LeerHumedad(){

   int humedad=0;  //Variable tipo int

   int resultado=0;  //Variable tipo int

   int result=0;  //Variable tipo int

   humedad=analogRead(Humedad);  //Leemos al sensor en el puerto Analogico 0; valores 1023=seco ,0=totalmente humedo

   result = map(humedad, 0, 1023, 0, 100);

   resultado=100-result;  //Restamos 100-result debido a que result me devuelve 100=seco, 0=totalmente humedo

   return resultado;  // retornamos el valor sensado

}
```

```
//Funcion LeerLuminosidad

int LeerLuminosidad(){

  int sensorValue=0;  //Variable tipo int

  int outputValue= 0;  //Variable tipo int

  int resultado=0;  //Variable tipo int

  sensorValue = analogRead(Luminosidad); // guardamosen sensorValue el valor del sensor conectado en el pin Luminosidad

  outputValue = map(sensorValue, 0, 1023, 0, 100); //cambiamos el rango de 0-1023 a 0-255

  resultado=100-outputValue;  //Restamos 100-result debido a que result me devuelve 100=oscruo, 0=totalmente iluminado

  return resultado;  // retornamos el valor sensado

}


//Funcion CompararTemperatura

void CompararTemperatura(float temp){

  float valorMinTemp= 18.00;  //Valor Minimo= 18 grados Centigrados

  float valorMedTemp= 22.00;  //Valor Medio= 22 grados Centigrados

  float valorMaxTemp= 25.00;  //Valor Maximo= 25 grados Centigrados

  int lumi=LeerLuminosidad();  //lumi obtiene el valor que devuelve la funcion


  if(lumi<=70)

  {

    if(lumi<50)

      estadoLuz=1;  //estado de la Luz Artificial encendida
```

```
else
  estadoLuz=0;  //estado de la Luz Artificial apagada

if(temp<=valorMinTemp || (temp>valorMinTemp && temp<valorMedTemp))
{
  digitalWrite(Refrigeracion,LOW); //Desactivamos Refrigeracion
  digitalWrite(Ventilacion,LOW); //Desactivamos Ventilacion
  estadoRef=0;  //estado de la Refrigeracion apagada
  estadoVen=0;  //estado de la Ventilacion apagada
}

if(temp>=valorMedTemp && temp<=valorMaxTemp)
{
  digitalWrite(Ventilacion,HIGH);
  digitalWrite(Refrigeracion,LOW);
  estadoRef=0;  //estado de la Refrigeracion apagada
  estadoVen=1;  //estado de la Ventilacion encendida
}

if(temp>valorMaxTemp)
{
  digitalWrite(Ventilacion,HIGH); //Activamos Ventilacion
  digitalWrite(Refrigeracion,HIGH); //Desactivamos Refrigeracion
  estadoRef=1;  //estado de la Refrigeracion encendida
```

```
      estadoVen=0;  //estado de la Ventilacion apagada
    }
  }

  if(lumi>70)
  {
    estadoLuz=0;  //estado de la Luz Artificial apagada
    if(temp<=valorMinTemp || (temp>=valorMinTemp && temp<=valorMaxTemp))
    {
      digitalWrite(Refrigeracion,LOW); //Activamos Refrigeracion
      digitalWrite(Ventilacion,LOW); //Desactivamos Ventilacion
      estadoRef=0;  //estado de la Refrigeracion apagada
      estadoVen=0;  //estado de la Ventilacion apagada
    }

    if(temp>valorMaxTemp)
    {
      digitalWrite(Ventilacion,HIGH); //Activamos Ventilacion
      digitalWrite(Refrigeracion,HIGH); //Desactivamos Refrigeracion
      estadoRef=1;  //estado de la Refrigeracion encendida
      estadoVen=0;  //estado de la Ventilacion apagada
    }
  }
}
```

```
//Funcion ComparарHumedad
void CompararHumedad(int hum){
  int valorMinHum= 60; //valorMinHumReal=60%
  int valorMaxHum= 80; //valorMaxHumReal=80%
  int lumi= LeerLuminosidad();  //lumi obtiene el valor que devuelve la funcion
  boolean estadoPasado=false;  //estado por Defecto del Riego(sin regar)
  boolean estadoActual;  //estado Actual del Riego

  if(lumi<=70)
  {
    if(lumi<50)
      estadoLuz=1;  //estado de la Luz Artificial encendida
    else
      estadoLuz=0;  //estado de la Luz Artificial apagada

    if(hum<valorMinHum)
    {
      analogWrite(Riego,255); // Activamos el Riego
      estadoActual=true;  //estado Actual de Riego en 1, que es regando
      estadoVal=1;  //estado de la Valvula Abierta
    }

    if(hum>=valorMinHum && hum<=valorMaxHum)
```

```
  {
    if(estadoActual!=estadoPasado)
    {
      analogWrite(Riego,255); // Activamos el Riego
      estadoActual=true;  //estado Actual de Riego en true, que es regando
      estadoVal=1;  //estado de la Valvula Abierta
    }
    else
    {
      analogWrite(Riego,0); //Desactivamos el Riego
      estadoActual=false;  //estado Actual de Riego en false, que es sin regar
      estadoVal=0;  //estado de la Valvula Cerrada
    }
  }

  if(hum>valorMaxHum)
  {
    analogWrite(Riego,0); //Desactivamos el Riego
    estadoActual=false;  //estado Actual de Riego en false, que es sin regar
    estadoVal=0;  //estado de la Valvula Cerrada
  }
}

if(lumi>70)
```

```
  {
    estadoLuz=0;  //estado de la Luz Artificial apagada
    if(hum<valorMinHum)
    {
      analogWrite(Riego,255); // Activamos el Riego
      estadoVal=1;  //estado de la Valvula Abierta
    }

    if(hum>=valorMinHum && hum<=valorMaxHum)
    {
      analogWrite(Riego,255); // Activamos el Riego
      estadoVal=1;  //estado de la Valvula Abierta
    }

    if(hum>valorMaxHum)
    {
      analogWrite(Riego,0); //Desactivamos el Riego
      estadoVal=0;  //estado de la Valvula Cerrada
    }
  }
}

//Funcion ComparарLuminosidad
void CompararLuminosidad(int lum){
```

```
  int valorMinLum=50; // valorMinLum=50% de luz
  int valorMaxLum=70; // valorMaxLum=70% de luz

  if(lum<valorMinLum)
  {
    analogWrite(LuzArtificial,255); // Activamos el Foco de Luz Artificial
    estadoLuz=1;  //estado de la Luz Artificial encendida
  }

  if(lum>=valorMinLum)
  {
    analogWrite(LuzArtificial,0); // Desactivamos el Foco de Luz Artificial
    estadoLuz=0;  //estado de la Luz Artificial apagada
  }
}

void LeerPaquetes(){
  float temp;
  float tempEnv;
  String CharAStr;
  String Cadena;
  String valorMinTemp="18.00";  //Valor de Umbral Minimo
  String valorMaxTemp="25.00";  //Valor de Umbral Maximo
```

```
int hum;

int humEnv;

String EntAChar;

String CharAStr2;

String Cadena2;

String valorMinHum="00060";  //Valor de Umbral Minimo

String valorMaxHum="00080";  //Valor de Umbral Maximo


int lum;

int lumEnv;

String EntAChar2;

String CharAStr3;

String Cadena3;

String valorMinLum="00050";  //Valor de Umbral Minimo

String valorMaxLum="00070";  //Valor de Umbral Maximo


char temporalBuff[PACKET_SIZE];

char Buffer[MAX_SIZE];

char packetBuffer[PACKET_SIZE];


String estadoActual;  //estado Actual Guarda los modos(Automatico o Manual), substring de estadoTemporal

String estadoPasado="AUTOON";  //estado Pasado predeterminado es Automatico y cambia dependiendo el proceso

String estadoTemp;  //Variable que guarda el modo cuando no se recibe paquetes UDP de forma Remota
```

```
String estadoTemporal;  //Variable que guardara el modo y la consulta tipo String

String consultArray;    //consultArray Guarda las consultas o activaciones, substring de estadoTemporal


int contador=0;  //Bandera que me ayudara a detectar cambio de Modos Automatico a Manual y visceversa.


if(digitalRead(switchModo)==HIGH)  //se pregunta si esta activado el Modo Remoto

{

  lcd.noDisplay();  //Desactivamos la impresion la LCD

  lcd.noBacklight();  //Desactivamos la luz trasera de la LCD


  int packSize = Udp.parsePacket();  //Pregunta si esta disponible un Paquete UDP

  delay(500);  //retardo de un segundo


  if(packSize)  //si se recibio un paquete UDP entra a leer lo que recibio

  {

    Udp.read(packetBuffer,PACKET_SIZE);  //Lectura del Paquete UDP presente

    estadoTemporal=(String)packetBuffer;      //Cast a String del paquete y guardo en estadoTemporal

    estadoActual=estadoTemporal.substring(0,6);          //Obtenemos el Modo de Operacion(Automatico o Manual)

    consultArray=estadoTemporal.substring(6,12);    //Obtenemos la Consulta o Activacion a Realizar


    if (estadoActual != estadoPasado)  //si detecta un cambio de Automatico a Manual o Visceversa
```

```
{  //Modo Manual de forma Remota
  if (estadoActual=="AUTOFF")
  {
    contador++;  //actualizamos el valor de la bandera

    if(consultArray=="I1T1FI")
    {
      tempEnv= LeerTemperatura();
      dtostrf(tempEnv,5,2,temporalBuff);     //Convertimos el valor int a Char Array en temporalBuffer
      CharAStr= (String) temporalBuff;  //Convertimos de Char Array a String
      Cadena= CharAStr + valorMinTemp + valorMaxTemp;  //Concatenamos para formar la trama a enviar
      Cadena.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer
      Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer
    }

    if(consultArray=="I1H2FI")
    {
      humEnv= LeerHumedad();
      EntAChar= (String)humEnv;  //Convertimos el valor int a String

      if(humEnv>=0 && humEnv<=9)
      {
```

```
        EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena2= "0000" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar

        Cadena2.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(humEnv>=10 && humEnv<=99)

      {

        EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena2= "000" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar

        Cadena2.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(humEnv==100)

      {

        EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
```

```
            CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

            Cadena2= "00" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar

            Cadena2.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }

        }


        if(consultArray=="I1L3FI")

        {

          lumEnv= LeerLuminosidad();

          EntAChar2= (String)lumEnv;  //Convertimos el valor int a String


          if(lumEnv>=0 && lumEnv<=9)

          {

            EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);  //Convertimos el String a Char Array en temporalBuffer

            CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

            Cadena3= "0000" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

            Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }
```

```
        if(lumEnv>=10 && lumEnv<=99)

        {

          EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);  //Convertimos el String a Char Array en temporalBuffer

          CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

          Cadena3= "000" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

          Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

          Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

        }


        if(lumEnv==100)

        {

          EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);  //Convertimos el String a Char Array en temporalBuffer

          CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

          Cadena3= "00" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

          Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

          Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

        }

      }
```

```
if (consultArray=="ICALON")
{
  digitalWrite(Refrigeracion, HIGH);  //Activamos la Refrigeracion
  estadoRef=1;  //estado Refrigeracion encendida
}

if (consultArray=="ICALOF")
{
  digitalWrite(Refrigeracion, LOW);  //Desactivamos la Refrigeracion
  estadoRef=0;  //estado Refrigeracion apagada
}

if (consultArray=="IVENON")
{
  digitalWrite(Ventilacion, HIGH);  //Activamos la Ventilacion
  estadoVen=1;  //estado Ventilacion encendida
}

if (consultArray=="IVENOF")
{
  digitalWrite(Ventilacion,LOW);  //Desactivamos la Ventilacion
  estadoVen=0;  //estado Ventilacion apagada
}
```

```
if (consultArray=="I1VONF")
{
   analogWrite(Riego,255);  //Activamos el Riego
   estadoVal=1;  //estado Valvula Abierta
}

if (consultArray=="I1VOFF")
{
   analogWrite(Riego,0);  //Desactivamos el Riego
   estadoVal=0;  //estado Valvula Cerrada
}

if(consultArray=="ILUZON")
{
  analogWrite(LuzArtificial,255);  //Activamos la Luz Artificial
  estadoLuz=1;  //estado Luz Artificial encendida
}

if(consultArray=="ILUZOF")
{
  analogWrite(LuzArtificial,0);  //Desactivamos la Luz Artificial
  estadoLuz=0;  //estado Luz Artificial apagada
}
```

```
        if(consultArray=="I1CEST")
        {
          if(estadoRef==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);     //enviamos estado Refrigeracion Apagada
          else
            Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Refrigeracion Encendida
        }

        if(consultArray=="I1VEST")
        {
          if(estadoVen==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);      //enviamos estado Ventilacion Apagada
          else
            Udp.sendPacket("Encendida",remoteIp,remotePort);     //enviamos estado Ventilacion Encendida
        }

        if(consultArray=="I1LEST")
        {
          if(estadoLuz==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);     //enviamos estado Luz Artificial Apagada
          else
```

```
            Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Luz Artificial Encendida
          }

          if(consultArray=="I1BEST")
          {
            if(estadoVal==0)
              Udp.sendPacket("Cerrada",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
            else
              Udp.sendPacket("Abierta",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
          }
        }
      }

      if(estadoActual=="AUTOON")
      {  //Modo Automatico de forma Remota
        lum=LeerLuminosidad();
        temp=LeerTemperatura();
        hum=LeerHumedad();

        CompararLuminosidad(lum);
        CompararTemperatura(temp);
        CompararHumedad(hum);
        delay(250);
```

```
 if(consultArray=="I1T1FI")

{

  tempEnv= LeerTemperatura();

  dtostrf(tempEnv,5,2,temporalBuff);  //Convertimos el valor int a Char Array en temporalBuffer

  CharAStr= (String) temporalBuff;  //Convertimos de Char Array a String

  Cadena= CharAStr + valorMinTemp + valorMaxTemp;  //Concatenamos para formar la trama a enviar

  Cadena.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

  Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

}


if(consultArray=="I1H2FI")

{

  humEnv= LeerHumedad();

  EntAChar= (String)humEnv;  //Convertimos el valor int a String


  if(humEnv>=0 && humEnv<=9)

  {

   EntAChar.toCharArray(temporalBuff,PACKET_SIZE);  //Convertimos el String a Char Array en temporalBuffer

   CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

   Cadena2= "0000" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar
```

```
        Cadena2.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(humEnv>=10 && humEnv<=99)

      {

        EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena2= "000" + CharAStr2 + valorMinHum + valorMaxHum;   //Concatenamos para formar la trama a enviar

        Cadena2.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(humEnv==100)

      {

        EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena2= "00" + CharAStr2 + valorMinHum + valorMaxHum;   //Concatenamos para formar la trama a enviar

        Cadena2.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer
```

```
          Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer
        }
      }

      if(consultArray=="I1L3FI")
      {
        lumEnv= LeerLuminosidad();
        EntAChar2= (String)lumEnv;  //Convertimos el valor int a String

        if(lumEnv>=0 && lumEnv<=9)
        {
          EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
          CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String
          Cadena3= "0000" + CharAStr3 + valorMinLum + valorMaxLum;   //Concatenamos para formar la trama a enviar
          Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer
          Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer
        }

        if(lumEnv>=10 && lumEnv<=99)
        {
          EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
```

```
            CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

            Cadena3= "000" + CharAStr3 + valorMinLum + valorMaxLum;   //Concatenamos para formar la trama a enviar

            Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }


          if(lumEnv==100)

          {

            EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

            CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

            Cadena3= "00" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

            Cadena3.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer

            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }

        }


        if(consultArray=="I1CEST")

        {

          if(estadoRef==0)

           Udp.sendPacket("Apagada",remoteIp,remotePort);      //enviamos estado Refrigeracion Apagada
```

```
        else
          Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Refrigeracion Encendida
      }

      if(consultArray=="I1VEST")
      {
        if(estadoVen==0)
          Udp.sendPacket("Apagada",remoteIp,remotePort);      //enviamos estado Ventilacion Apagada
        else
          Udp.sendPacket("Encendida",remoteIp,remotePort);     //enviamos estado Ventilacion Encendida
      }

      if(consultArray=="I1LEST")
      {
        if(estadoLuz==0)
          Udp.sendPacket("Apagada",remoteIp,remotePort);     //enviamos estado Luz Artificial Apagada
        else
          Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Luz Artificial Encendida
      }

      if(consultArray=="I1BEST")
```

```
    {
      if(estadoVal==0)
        Udp.sendPacket("Cerrada",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
      else
        Udp.sendPacket("Abierta",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
    }
  }


  if(contador==0)  //entra aqui solo si el estadoActual es igual al estadoPasado
  {  //Modo Manual de forma Remota
    if(estadoActual=="AUTOFF")
    {
      if(consultArray=="I1T1FI")
      {
        tempEnv= LeerTemperatura();
        dtostrf(tempEnv,5,2,temporalBuff);      //Convertimos el valor int a Char Array en temporalBuffer
        CharAStr= (String) temporalBuff;  //Convertimos de Char Array a String
        Cadena= CharAStr + valorMinTemp + valorMaxTemp;  //Concatenamos para formar la trama a enviar
        Cadena.toCharArray(Buffer,MAX_SIZE+1);  //Guardamos la cadena anterior en un Char Array Buffer
        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer
      }
```

```
if(consultArray=="I1H2FI")
{
  humEnv= LeerHumedad();
  EntAChar= (String)humEnv;  //Convertimos el valor int a String

  if(humEnv>=0 && humEnv<=9)
  {
    EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
    CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String
    Cadena2= "0000" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar
    Cadena2.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer
    Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer
  }

  if(humEnv>=10 && humEnv<=99)
  {
    EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
    CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String
    Cadena2= "000" + CharAStr2 + valorMinHum + valorMaxHum;  //Concatenamos para formar la trama a enviar
    Cadena2.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer
```

```
            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }


          if(humEnv==100)

          {

            EntAChar.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

            CharAStr2 = (String) temporalBuff;  //Convertimos de Char Array a String

            Cadena2= "00" + CharAStr2 + valorMinHum + valorMaxHum;   //Concatenamos para formar la trama a enviar

            Cadena2.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

            Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

          }

        }


        if(consultArray=="I1L3FI")

        {

          lumEnv= LeerLuminosidad();

          EntAChar2= (String)lumEnv;  //Convertimos el valor int a String


          if(lumEnv>=0 && lumEnv<=9)

          {

            EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer
```

```
        CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena3= "0000" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

        Cadena3.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(lumEnv>=10 && lumEnv<=99)

      {

        EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String

        Cadena3= "000" + CharAStr3 + valorMinLum + valorMaxLum;  //Concatenamos para formar la trama a enviar

        Cadena3.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }


      if(lumEnv==100)

      {

        EntAChar2.toCharArray(temporalBuff,PACKET_SIZE);   //Convertimos el String a Char Array en temporalBuffer

        CharAStr3 = (String) temporalBuff;  //Convertimos de Char Array a String
```

```
        Cadena3= "00" + CharAStr3 + valorMinLum + valorMaxLum;   //Concatenamos para formar la trama a enviar

        Cadena3.toCharArray(Buffer,MAX_SIZE+1);   //Guardamos la cadena anterior en un Char Array Buffer

        Udp.sendPacket(Buffer,remoteIp,remotePort);  //Enviamos al servidor la trama guardada en Buffer

      }

    }


    if (consultArray=="ICALON")

    {

      digitalWrite(Refrigeracion, HIGH);  //Activamos la Refrigeracion

      estadoRef=1;  //estado Refrigeracion encendida

    }


    if (consultArray=="ICALOF")

    {

     digitalWrite(Refrigeracion, LOW);  //Desactivamos la Refrigeracion

     estadoRef=0;  //estado Refrigeracion apagada

    }


    if (consultArray=="IVENON")

    {

      digitalWrite(Ventilacion, HIGH);  //Activamos la Ventilacion

      estadoVen=1;  //estado Ventilacion encendida

    }
```

```
if (consultArray=="IVENOF")
{
   digitalWrite(Ventilacion,LOW);  //Desactivamos la Ventilacion
   estadoVen=0;  //estado Ventilacion apagada
}

if (consultArray=="I1VONF")
{
   analogWrite(Riego,255);  //Activamos el Riego
   estadoVal=1;  //estado Valvula Abierta
}

if (consultArray=="I1VOFF")
{
   analogWrite(Riego,0);  //Desactivamos el Riego
   estadoVal=0;  //estado Valvula Cerrada
}

if(consultArray=="ILUZON")
{
  analogWrite(LuzArtificial,255);  //Activamos la Luz Artificial
  estadoLuz=1;  //estado Luz Artificial encendida
}
```

```
        if(consultArray=="ILUZOF")
        {
          analogWrite(LuzArtificial,0);  //Desactivamos la Luz Artificial
          estadoLuz=0;  //estado Luz Artificial apagada
        }

        if(consultArray=="I1CEST")
        {
          if(estadoRef==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);     //enviamos estado Refrigeracion Apagada
          else
            Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Refrigeracion Encendida
        }

        if(consultArray=="I1VEST")
        {
          if(estadoVen==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);      //enviamos estado Ventilacion Apagada
          else
            Udp.sendPacket("Encendida",remoteIp,remotePort);     //enviamos estado Ventilacion Encendida
        }
```

```
        if(consultArray=="I1LEST")
        {
          if(estadoLuz==0)
            Udp.sendPacket("Apagada",remoteIp,remotePort);     //enviamos estado Luz Artificial Apagada
          else
            Udp.sendPacket("Encendida",remoteIp,remotePort);    //enviamos estado Luz Artificial Encendida
        }

        if(consultArray=="I1BEST")
        {
          if(estadoVal==0)
            Udp.sendPacket("Cerrada",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
          else
            Udp.sendPacket("Abierta",remoteIp,remotePort);  //enviamos estado Valvula Cerrada
        }
       }
     }
     estadoPasado = estadoActual;  // guarda el estado actual como el ultimo estado
     contador=0;  //vuelve la Bandera al estado Inicial por si hay un cambio de Modo
     Udp.flush();  //vaciamos el contenido del packetBuffer
   }
   delay(500);
```

```
 }
 else
 {  //Modo Automatico de forma Local
   lcd.display();  //Activamos la impresion en la LCD
   lcd.backlight();  //Activamos la luz Trasera de la LCD
   lcd.clear();
   lcd.noCursor();  //Desactivamos el Cursor de la LCD

   if(digitalRead(switchOperacion)==HIGH)   //se pregunta si esta activado el Modo Automatico Local
   {

     lum=LeerLuminosidad();  //Leemos la Luminosidad
     CompararLuminosidad(lum);
     lcd.setCursor(0,0);  //Llevamos al cursor a la Posicion(0,0)
     lcd.print("Luminosidad: ");  //Presentamos en la LCD
     lcd.setCursor(14,0);  //Llevamos al cursor a la Posicion(14,0)
     lcd.print(lum);  //Presentamos en la LCD
     lcd.setCursor(18,0);  //Llevamos al cursor a la Posicion(18,0)
     lcd.print("%");  //Presentamos en la LCD
     delay(500);

     temp=LeerTemperatura();  //Leemos la Temperatura
     CompararTemperatura(temp);
     lcd.setCursor(0,1);  //Llevamos al cursor a la Posicion(0,1)
```

```
  lcd.print("Temperatura: ");  //Presentamos en la LCD
  lcd.setCursor(13,1);  //Llevamos al cursor a la Posicion(13,1)
  lcd.print(temp);  //Presentamos en la LCD
  lcd.setCursor(18,1);  //Llevamos al cursor a la Posicion(18,1)
  lcd.print("\337C");  //Presentamos en la LCD el simbolo "°"
  lcd.setCursor(19,1);  //Llevamos al cursor a la Posicion(19,1)
  lcd.print("C");  //Presentamos en la LCD
  delay(500);

  hum=LeerHumedad();  //Leemos la Humedad
  CompararHumedad(hum);
  lcd.setCursor(0,2);  //Llevamos al cursor a la Posicion(0,2)
  lcd.print("Humedad: ");  //Presentamos en la LCD
  lcd.setCursor(10,2);  //Llevamos al cursor a la Posicion(10,2)
  lcd.print(hum);  //Presentamos en la LCD
  lcd.setCursor(14,2);  //Llevamos al cursor a la Posicion(14,2)
  lcd.print("%");  //Presentamos en la LCD
  delay(500);

  lcd.setCursor(0,3);  //Llevamos al cursor a la Posicion(0,3)
  #define cadena1 "**Modo Auto. Local**"  //Definimos la Cadena a presentar en la LCD
  lcd.print(cadena1);
  delay(2000);
}
```

```
    else
    {  //entra aqui cuando esta en Modo Manual Local
      PresentarLuz();
      BorrarUltimaLinea();
      PresentarTemperatura();
      BorrarUltimaLinea();
      PresentarHumedad();
      BorrarUltimaLinea();
      Activaciones();
      delay(500);
    }
  }
}

void Activaciones(){
  if(digitalRead(switchRefrigeracion)==LOW && digitalRead(switchVentilacion)==HIGH)
  {
    digitalWrite(Refrigeracion, LOW);  //Activamos la Refrigeracion
    digitalWrite(Ventilacion, HIGH);  //Activamos la Ventilacion
  }

  if(digitalRead(switchRefrigeracion)==HIGH && digitalRead(switchVentilacion)==LOW)
  {
    digitalWrite(Refrigeracion, HIGH);  //Activamos la Refrigeracion
```

```
    digitalWrite(Ventilacion, LOW);  //Activamos la Ventilacion
  }

  if(digitalRead(switchRefrigeracion)==HIGH && digitalRead(switchVentilacion)==HIGH)
  {
    digitalWrite(Refrigeracion, HIGH);  //Activamos la Refrigeracion
    digitalWrite(Ventilacion, LOW);  //Activamos la Ventilacion
  }

  if(digitalRead(switchRefrigeracion)==LOW && digitalRead(switchVentilacion)==LOW)
  {
    digitalWrite(Refrigeracion, LOW);  //Activamos la Refrigeracion
    digitalWrite(Ventilacion, LOW);  //Activamos la Ventilacion
  }

  if(digitalRead(switchValvula)==HIGH)
    analogWrite(Riego,255);  //Activamos el Riego
  else
    analogWrite(Riego,0);  //Desactivamos el Riego
}

void PresentarLuz(){
  int lum;
  Activaciones();
```

```
lum=LeerLuminosidad();  //Leemos la Luminosidad
lcd.setCursor(0,0);  //Llevamos al cursor a la Posicion(0,0)
lcd.print("Luminosidad: ");  //Presentamos en la LCD
lcd.setCursor(14,0);  //Llevamos al cursor a la Posicion(14,0)
lcd.print(lum);  //Presentamos en la LCD
lcd.setCursor(18,0);  //Llevamos al cursor a la Posicion(18,0)
lcd.print("%");  //Presentamos en la LCD

lcd.setCursor(0,3);  //Llevamos al cursor a la Posicion(0,3)

if(lum<50)
{
  #define cadena1 "*Luz Artificial ON.*"  //Definimos la Cadena a presentar en la LCD
  Activaciones();
  lcd.print(cadena1);
  analogWrite(LuzArtificial,255);  //Activamos la Luz Artificial
}

if(lum>=50 && lum<=70)
{
  #define cadena1 "*Luz Artificial OFF."  //Definimos la Cadena a presentar en la LCD
  Activaciones();
  lcd.print(cadena1);
  analogWrite(LuzArtificial,0);  //Desactivamos la Luz Artificial
```

```
  }

  if(lum>70)
  {
    #define cadena1 "***Activar Riego.***"  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
    analogWrite(LuzArtificial,0);  //Desactivamos la Luz Artificial
  }
}

void PresentarTemperatura(){
  float temp;
  Activaciones();
  temp=LeerTemperatura();  //Leemos la Temperatura
  lcd.setCursor(0,1);  //Llevamos al cursor a la Posicion(0,1)
  lcd.print("Temperatura: ");  //Presentamos en la LCD
  lcd.setCursor(13,1);  //Llevamos al cursor a la Posicion(13,1)
  lcd.print(temp);  //Presentamos en la LCD
  lcd.setCursor(18,1);  //Llevamos al cursor a la Posicion(18,1)
  lcd.print("\337C");  //Presentamos en la LCD el simbolo "°"
  lcd.setCursor(19,1);  //Llevamos al cursor a la Posicion(19,1)
  lcd.print("C");  //Presentamos en la LCD
```

```
  lcd.setCursor(0,3);  //Llevamos al cursor a la Posicion(0,3)

  if(temp<=18 || (temp>18 && temp<22))
  {
    #define cadena1 "                    "  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
  }

  if(temp>=22 && temp<=25)
  {
    #define cadena1 "Activar Ventilacion."  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
  }

  if(temp>25)
  {
    #define cadena1 "*Activar Refriger.**"  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
  }
}
```

```
void PresentarHumedad(){
  int hum;
  Activaciones();
  hum=LeerHumedad();  //Leemos la Humedad
  lcd.setCursor(0,2);  //Llevamos al cursor a la Posicion(0,2)
  lcd.print("Humedad: ");  //Presentamos en la LCD
  lcd.setCursor(10,2);  //Llevamos al cursor a la Posicion(10,2)
  lcd.print(hum);  //Presentamos en la LCD
  lcd.setCursor(14,2);  //Llevamos al cursor a la Posicion(14,2)
  lcd.print("%");  //Presentamos en la LCD

  lcd.setCursor(0,3);  //Llevamos al cursor a la Posicion(0,3)

  if(hum<60)
  {
    #define cadena1 "***Activar Riego.***"  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
  }

  if(hum>=60 && hum<=80)
  {
    #define cadena1 "*Humedad Suficiente."  //Definimos la Cadena a presentar en la LCD
    Activaciones();
```

```
    lcd.print(cadena1);
  }

  if(hum>80)
  {
    #define cadena1 "**Desactivar Riego.*"  //Definimos la Cadena a presentar en la LCD
    Activaciones();
    lcd.print(cadena1);
  }
}

void BorrarUltimaLinea(){
  delay(500);
  Activaciones();
  delay(500);
  Activaciones();
  delay(500);
  Activaciones();
  lcd.setCursor(0,3);
  lcd.print("                    ");
```

BIBLIOGRAFÍA

[1] Telégrafo, La sequía afecta a los cultivos de maíz en 3 parroquias Lojanas, http://www.telegrafo.com.ec/economia/item/la-sequia-afecta-a-los-cultivos-de-maiz-en-3-parroquias-lojanas.html, fecha de consulta enero 2015.

[2] Gosálbez Celia, Rábano: Cultivo rapidísimo, http://www.planetahuerto.es/revista/rabano-cultivo-rapidisimo_00022, fecha de consulta febrero 2015.

[3] Wikipedia, Invernadero, http://es.wikipedia.org/wiki/Invernadero, fecha de consulta enero 2015.

[4] Maxim Integrated, DSB1820 Programmable Resolution 1-Wire Digital Thermometer, http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, fecha de consulta diciembre 2014.

[5] ylvesdxlelwgadgets, Wordpress, Sensor de temperatura digital, https://ylvesdxlelwgadgets.wordpress.com/2014/08/25/wxm08-ds18b20-digital-temperature-sensor-module-blue-200518/ fecha de consulta enero del 2015

[6] Micropik, FC-28 Sensor de Humedad del Suelo, http://www.micropik.com/PDF/FC_28.pdf, fecha de consulta diciembre 2014.

[7] Cebek componentes y accesorios, Fotorresistencia LDR, http://www.electan.com/datasheets/cebek/CE-C2795.pdf, fecha de consulta diciembre 2014.

8] WebElectro, Sensor de luz, http://www.webelectro.com.ar/articulo/otros/ldr-modulo-sensor-de-luz-arduino-ldr-light-sensor-module fecha de consulta enero del 2015

[9] Electro Components, Arduino Uno, http://docs-europe.electrocomponents.com/webdocs/0e8b/0900766b80e8ba21.pdf, fecha de consulta diciembre 2014.

[10] Arduino.CC, Arduino Ethernet Shield, http://arduino.cc/en/pmwiki.php?n=Guide/ArduinoEthernetShield, fecha de consulta diciembre 2014.

[11] Arduino e Cia, Display LCD 20x4 Arduino, http://www.arduinoecia.com.br/2014/06/arduino-display-lcd-20x4.html, fecha de consulta enero 2015.

[12] ElectroHobby, Conversor I2C para Arduino maneja tu LCD con solo 2 hilos, http://www.electrohobby.es/conversor-lcd-i2c-para-arduino-maneja-tu-lcd-con-solo-2-hilos/, fecha de consulta enero 2015.

[13] Revista Luminica, Iluminación horticultural y floral: la importancia de la luz en los invernaderos, http://www.revistaluminica.es/WP/iluminacion-horticultural-y-floral-la-importancia-de-la-luz-en-los-invernaderos/, fecha de consulta enero 2015.

[14] TP Link, Router inalámbrico N a 300Mbps, http://www.tp-link.ec/products/details/?categoryid=&model=TL-WR841ND, fecha de consulta marzo 2015.

[15] Unesco, Árbol de Problemas, http://www.unesco.org/new/es/culture/themes/cultural-diversity/diversity-of-

cultural-expressions/tools/policy-guide/planificar/diagnosticar/arbol-de-problemas/, fecha de consulta febrero 2015.

[16] Wikipedia, El niño (fenómeno), http://es.wikipedia.org/wiki/El_Ni%C3%B1o_%28patr%C3%B3n_clim%C3%A1tico%29, fecha de consulta enero 2015.

[17] Wikipedia, Modelo de Prototipos, http://es.wikipedia.org/wiki/Modelo_de_prototipos, fecha de consulta febrero 2015.

[18] Megakywi, http://www.kywi.com.ec/catalogo, fecha de compra marzo 2015.

[19] Miguel Parrales-Acrílico, fecha de compra marzo 2015.

[20] IDETEC S.A, http://ideastechnology.com/, fecha de compra enero 2015.

[21] MercadoLibre Ecuador, www.mercadolibre.com.ec/, fecha de compra noviembre 2014.

[22] INTROMEX S.A, fecha de compra marzo 2015.