Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTULO DEL TFC: Monitorización remota de una red 1-Wire

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTORES: Narciso Cuartero Moya
Sergio Quintana Alcaraz

DIRECTOR: José Polo Cantero

FECHA: 4 de Febrero de 2008

Título: Monitorización remota de una red 1-Wire

Autores: Narciso Cuartero Moya
Sergio Quintana Alcaraz

Director: José Polo Cantero

Fecha: 4 de Febrero de 2008

Resumen

En éste trabajo, implementaremos una red completa de monitorización de temperaturas. Crearemos un bus repleto de sensores alrededor de un lago, que conectado a un ordenador nos permite monitorizar y ver la evolución a través del tiempo para cada uno de los dispositivos. Dicho bus utiliza únicamente dos hilos, por los que transmite tanto los datos como la corriente necesaria para el funcionamiento del sistema. Utiliza el protocolo 1-Wire de la empresa Dallas Maxim, con los correspondientes sensores de la misma compañía. A lo largo del trabajo se expone el funcionamiento interno de éste protocolo y se crean dos prototipos de sistemas completos de monitorización, LakeMonitor y LakeMonitorV2.

El primero, utiliza las librerías que Dallas Maxim proporciona a los programadores, para desarrollar dos ejecutables. El primero lista las direcciones físicas de los sensores conectados al bus y el segundo nos devuelve el valor de temperatura si le pasamos como parámetro la dirección de un dispositivo concreto. Los valores obtenidos se van almacenando en una base de datos MySQL. El software encargado de dibujar las gráficas será RRDtool, controlado por NetMRG, que las servirá a la red a través de un servidor web Apache. El enlace al bus se realizará a través de un adaptador USB-Serie, conectado al Master DS2480B para 1-Wire, de Dallas Maxim.

LakeMonitorV2 no usa el PDK oficial, sino la suite de programas OWFS, que como se podrá ver, ofrece muchas ventajas respecto a nuestro propio software. La conexión al bus se realiza a través de un microordenador equipado con Linux, que transforma el enlace 1-Wire en paquetes IP que pueden viajar a través de la infraestructura de red existente en la zona de la implantación del sistema. Para almacenar y visualizar las gráficas se usa el mismo software que en la primera versión.

Title: Remote monitoring of a 1-Wire network

Authors: Narciso Cuartero Moya
Sergio Quintana Alcaraz

Director: José Polo Cantero

Date: February, 4th 2008

Overview

In this document, a complete temperature monitoring network is designed. A bus full of sensors surrounding a lake is connected to a computer, allowing users of the IP network to monitor evolution through time for any single temperature device. That bus uses only a pair of wires, where data signals and power are mixed. It uses the 1-Wire protocol from Dallas Maxim combined with the DS1920 sensors, coming from the same enterprise. On that paper is exposed the working of every piece or 1-Wire hardware and two prototypes of complete monitoring systems are created, LakeMonitor and LakeMonitorV2.

The first one, uses the libraries given by Dallas Maxim on its Software Developement Kit to build two programs. The first executable lists all devices that are found connected to the bus and the second one returns the temperature value for a given phisical address. Those returning values are stored on a MySQL database. Graphics are drawn by RRDTool, controlled by NetMRG, which will serve them to the network through an Apache web server. The link to the bus is done using a USB to serial adaptor connected to a DS2480B 1-Wire Master, from Dallas Maxim.

LakeMonitorV2 does not use the official PDK but the OWFS software suite that, as the reader will see, it offers lots of advantages compared to our own programs. The conection to the bus is done using a microcomputer with a 2.6 Linux Kernel inside that transforms the 1-Wire bus data on IP traffic, which can travel along the existing network infrastructure on the system implementation zone. To store and represent the graphics, we used the same NetMRG that in the first version.

AGRADECIMIENTOS

En primer lugar queremos agradecer a nuestro tutor del TFC, Jose Polo, por sus conocimientos, dirección y paciencia que nos ha proporcionado constantemente durante la realización de este trabajo. También recalcar que siempre nos ha dejado la libertad suficiente para poder tomar nuestras propias decisiones e ir superando uno a uno todos los problemas que nos hemos ido encontrando a lo largo del trabajo. Darle las gracias por comprender nuestra situación y facilitarnos al máximo la realización y presentación de dicho trabajo.

También queremos agradecer a nuestras familias y amigos por la paciencia que han tenido estos últimos meses y, sobretodo, los últimos dias. Por último, agradecer a Mar y a sus padres, por ofrecernos un punto de vista objetivo que tanto nos ayudo en la finalización del trabajo.

Indice

INTRODUCCIÓN

La primera vez que oímos hablar sobre la tecnología 1-Wire, empezaron a llover proyectos sobre nuestras cabezas. Permitía crear una red completa de dispositivos, usando únicamente 1 hilo y masa. A través de éstos dos cables circularían, no solo los datos, sino también la alimentación necesaria para cada uno de los nodos del sistema. Por si no fuese suficiente, las redes 1-Wire resultaban extremadamente robustas, escalables y con una sensibilidad prácticamente nula contra ruido externo.

De entre el enorme catálogo de dispositivos adaptados que *Dallas-Maxim* distribuía, nos llamaron especialmente la atención sus sensores de temperatura inteligentes. Se encargaban de capturar los datos del ambiente y servirlos para nosotros en bandeja de plata, de forma digital y asegurándonos la integridad de los datos recibidos, usando para ello algoritmos de detección de errores.

Éste proyecto fue para nosotros amor a primera vista. Teníamos la posibilidad de crear de principio a fin un sistema concebido para monitorizar una red completa de sensores de temperatura 1-Wire. Un único bus repleto de pequeñas piezas maestras de tecnología que, rodeando un lago, nos permitiría conocer el estado a través del tiempo en cada uno de sus puntos. Sería nuestra labor cuidar hasta el último detalle del proceso. Queríamos un producto redondo de principio a fin, casi comercial. Nuestro primer trabajo real como ingenieros.

Desde el comienzo hemos intentado tomar nuestras decisiones a la hora de desarrollar cada uno de los puntos del TFC siguiendo una serie de criterios. Queríamos que la mezcla funcionase de forma sencilla e intuitiva sobre la mayor parte de máquinas, y que implicase el mínimo coste de producción y mantenimiento posible.

Hemos estructurado nuestro trabajo en tres capítulos:

- El primero nos da unas nociones básicas de la red 1-Wire: características, protocolo, transmisiones, comandos ROM, etc…

- En el segundo desarrollamos la primera versión de nuestro sistema de monitorización, Lake Monitor V1.

- En el tercero mejoramos la primera versión, y creamos Lake Monitor V2.

CAPÍTULO 1. RED 1-WIRE

1.1. Rasgos Generales

La red 1-Wire, también conocida como Micro Lan, es un bus de bajo coste basado en un PC o un microcontrolador que se comunica digitalmente sobre un cable de 1 par con componentes 1-Wire.

Su principal característica radica en que físicamente se compone de un único conductor, más su retorno o masa, al que se encuentran conectados todos los dispositivos 1-Wire, permitiendo distancias elevadas. El límite de dispositivos depende su tipo, longitud del cableado, tipo de master, etc..., y se suele garantizar la funcionalidad de hasta 2000 dispositivos.

Cuando el bus está en reposo, en él aparece una tensión de 5V que es aprovechada por todos los dispositivos conectados para cargar los condensadores internos que aseguran la alimentación cuando el bus se está comunicando. En esta red podemos destacar 3 elementos principales:

1) Un bus master con software de control
2) Conectores y cableado
3) Dispositivos 1-Wire

La comunicación, básicamente se realiza entre master y slave, es decir, ningún slave transmite a no ser que el master lo haya ordenado. La red 1-Wire está conformada por un master y uno o más slaves, cuya comunicación es serial asincrónica, y poseen un único pin de datos de tipo open drain (colector abierto), al que se conecta una resistencia de pullup a +5V.

En cualquier momento en que la línea de datos se coloca en 1, el diodo del rectificador de media onda, conduce y carga un condensador integrado en el chip. Cuando el voltaje en la red cae por debajo del voltaje del condensador, el diodo no conduce, lo que aísla la carga. La carga resultante alimenta al slave durante el intervalo en que la línea este baja. La carga que se disipó durante este período, es recuperada cuando la línea de datos vuelve a estar en alto. Este concepto de *robar* la energía de la red usando un rectificador de media onda se conoce como *Energía Parásita*.

Durante la comunicación (figura 2.1), el master reinicia la red manteniendo la línea baja durante 480µs, luego la libera, y espera un pulso de presencia como respuesta del slave conectado a la línea. Si el pulso de presencia es detectado, el master accede al mismo llamando a su dirección o registro, controlando la transferencia de información a través de la generación de los time slots y examinando la respuesta del slave. Una vez que esta retroalimentación es exitosa, el master emite comandos específicos necesarios del dispositivo y ejecuta cualquier transferencia de datos necesaria.

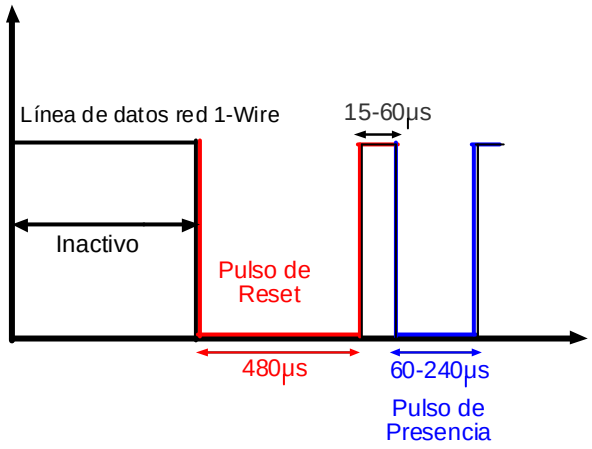


Figura 1.1 Inicio sesión Master y Slave

Una de las características de esta tecnología 1-Wire, es que cada dispositivo slave tiene una única e irrepetible identificación grabada en su memoria ROM. El master puede seleccionar un solo slave de los muchos que puede haber en la red, pues cada dispositivo posee una dirección o registro único. Éste mismo número de serie está grabado con laser a los dispositivos iButton que usaremos.

1.2. Características de la red 1-Wire

Entre las muchas características de esta red, destacamos las siguientes:

- Utiliza niveles de alimentación CMOS/TTL con un rango de operación desde 2.8V hasta 6V.

- Tanto el master como los slaves transmiten información de forma bidireccional, pero, sólo en una dirección a la vez. De esta manera la comunicación es half duplex.

- Toda la información es leída o escrita comenzando por el bit menos significativo (LSB).

- No se requiere del uso de un reloj, ya que, cada componente 1-Wire posee un oscilador interno que se sincroniza con el del master cada vez que en la línea de datos aparece un flanco de bajada.

- Todas las tensiones mayores a 2,2 Voltios son consideradas un (1) lógico mientras que un (0) lógico será cualquier voltaje menor de 0,8 V.

- La transferencia de información es a 16.3 Kbps en modo Standard y hasta 142 Kbps en modo Overdrive.

1.3. Protocolo de Comunicaciones 1-Wire

El protocolo 1-Wire consta de los siguientes pasos:

1) Inicialización
2) Comandos y funciones ROM
3) Comandos y funciones de control y memoria
4) Transferencia de datos

1.3.1. Inicialización

Todas las comunicaciones en el bus 1-Wire comienzan con una secuencia de un pulso de Reset y Presencia. El pulso de reset provee una forma limpia de iniciar las comunicaciones, ya que, con él se sincronizan todos los dispositivos slave presentes en el bus. Un reset es un pulso que genera el master al colocar la línea de datos en estado lógico bajo por unos 480 µs (figura 1.1). El Pulso de presencia lo generan los slaves para indicarle al master que están disponibles para cualquier operación.

1.3.2. Comandos y Funciones ROM

Una vez que el master recibe el pulso de presencia de los dispositivos slave, se puede enviar un comando ROM. Los comandos ROM son comunes en todos los dispositivos 1-Wire y se relacionan con la búsqueda, lectura y utilización de la dirección de 64 bits que identifica a los slaves.

Tabla 1.1. Comandos ROM en sistema 1-Wire

Comando	Valor	Acción
Read ROM	33h	Lee la identificación de 64 bits del dispositivo. Puede usarse si existe un sólo dispositivo slave, ya que si hubiesen varios se produciría una colisión cuando todos intentaran enviar información al mismo tiempo.
Match ROM	55h	Este comando, seguido de una identificación de 64 bits, permite seleccionar a un dispositivo slave en particular.
Skip ROM	CCh	Direcciona a un dispositivo sin necesidad de conocer su identificación. Puede ser utilizado solamente cuando existe un sólo slave.

Search ROM	F0h	Lee los 64 bits de identificación de los dispositivos slave conectados al bus. Se utiliza un proceso de eliminación para distinguir a cada dispositivo conectado.
Alarm Search	ECh	El bus manda una señal de alarma y el dispositivo que la tenga activada, le responde, ya sea TH o TL.

1.3.3. Comandos y Funciones de Control y Memoria

Son funciones propias de los dispositivos 1-Wire. Incluyen comandos para leer/escribir en zonas de la memoria, leer memorias de Scratchpad, controlar el inicio de la conversión, iniciar la medición de una temperatura o manipular el estado de un bit de salida, entre otros. Cada dispositivo define sus propios comandos.

1.3.4. Transferencia de Datos

La lectura y escritura de datos en el bus 1-Wire se hace por medio de Slots, la generación de éstos es responsabilidad del master.

Cuando el master lee información del bus, debe forzar la línea de datos a estado bajo durante al menos 1µs y esperar unos 15µs para entonces leer el estado de la misma. El estado lógico de la línea en ese momento, estará determinado por el dispositivo slave.

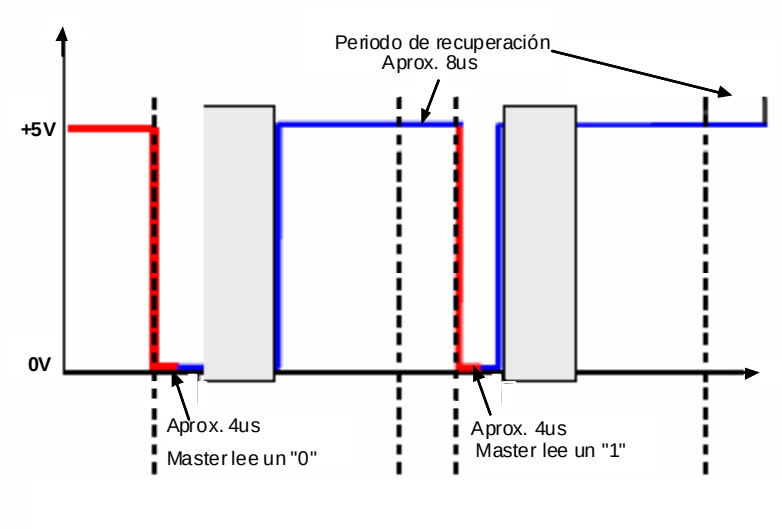


Figura 1.2 Lectura de datos en el bus 1-Wire

En el momento de efectuar la escritura de un bit en el bus ocurre algo similar, el master produce un pulso de entre 1µs y 15µs de duración, para luego colocar en el bus al bit que se desea transmitir. Este bit deberá permanecer en el bus al menos 60 µs.

1.4. Detección de errores en la red 1-Wire

En la red 1-Wire como en todas las redes, es necesario verificar la posible existencia de errores que pudieran ocurrir durante la comunicación. Existen varios métodos para efectuar este chequeo. Uno de los más simples consiste en añadir un bit extra a cada byte transmitido de modo que el número de unos (1´s) contenidos en el paquete de 9 bits siempre sea par o impar. Este procedimiento se le conoce como verificación de paridad y permite encontrar errores que ocurran en un bit, pero no es confiable cuando cambia más de un bit dentro del byte.

Uno de los mecanismos de detección de errores más eficientes es el Control de Redundancia Cíclica (CRC). El algoritmo utilizado para el cálculo de CRC de Dallas Semiconductor, cuyo resultado se incluye en la identificación de los dispositivos 1-Wire, es de 8 bits y se calcula introduciendo los primeros 56 bits, correspondientes al número de serie del dispositivo y el código de la familia a la que éste pertenece.

Los errores detectables por el CRC-8 implementado por Dallas son:

- Cualquier número impar de errores sobre los 64 bits transmitidos.
- Todos los errores de dos bits que se presenten en el serial de 64 bits.
- Cualquier grupo de errores de hasta 8 bits incorrectos.
- La gran mayoría de los errores de más de 8 bits incorrectos.

1.5. API de programación 1-Wire

Para la comunicación con dispositivos 1-Wire las diferentes interfaces de programa de aplicación (*Application Program Interface*, API) tienen características comunes que reflejan los patrones fundamentales de la comunicación que emerge del protocolo. API es la plataforma que proporciona el fabricante para acceder a los dispositivos y a la red 1-Wire desde un ordenador. Básicamente se definen como un conjunto de subprogramas o funciones de bajo nivel programadas en un entorno que depende tanto del lenguaje de programación como del sistema operativo utilizado.

Como la mayoría de los dispositivos 1-Wire poseen memoria, las funciones de memoria entrada/salida (I/O) son tratadas como un grupo de funciones API aunque las mismas no se apliquen a todos los dispositivos.

Tabla 1.1. Grupo de funciones API

SESIÓN
Incluye todas las funciones que hacen uso exclusivo del bus 1-Wire a través de cualquier interfaz de comunicación al PC. Todas las funciones incluidas en éste API, hacen uso exclusivo del puerto de comunicaciones del PC mientras dure la comunicación con el dispositivo 1-Wire.
ENLACE
Incluye todas las funciones básicas de comunicación de la red 1-Wire. Funciones básicas o primitivas son aquellas que realizan un “reset” en la red 1-Wire antes de establecer alguna comunicación con un dispositivo slave, para luego leer o escribir bits de información. Agrupa también ciertas funciones especiales que determinan las características eléctricas de funcionamiento de la red, como es el caso, de generación de pulsos de programación para escribir información en las memorias EPROM.
RED
Agrupa las funciones que permiten identificar y seleccionar dispositivos en la red 1-Wire. Utilizan como dirección de red el número de serie de cada dispositivo 1-Wire. Incluyen la totalidad de las funciones que trabajan con la memoria ROM de los dispositivos.
TRANSPORTE
Bloque de comunicación y funciones para la lectura y escritura en cualquier memoria diferente a la ROM. Estas funciones están construidas en base a las funciones encontradas en las API de Red y de Enlace.
ARCHIVO
Funciones relacionadas a las estructuras de los archivos y organización de la memoria dentro de los dispositivos. Son construidas en base a las funciones encontradas en las API de Red y de Transporte y son útiles en aquellos dispositivos con más de una página de memoria.
DISPOSITIVO
Funciones específicas para cada dispositivo, son denominadas de alto nivel, ya que dependen en todo momento de un dispositivo en específico. Estas funciones son construidas en base a las encontradas en las API de Red, Transporte, Enlace y realizan operaciones tales como la lectura de la temperatura de un dispositivo en particular o establecer el estado de un switch.

La secuencia típica para usar estas funciones se muestra en la figura 1.3.

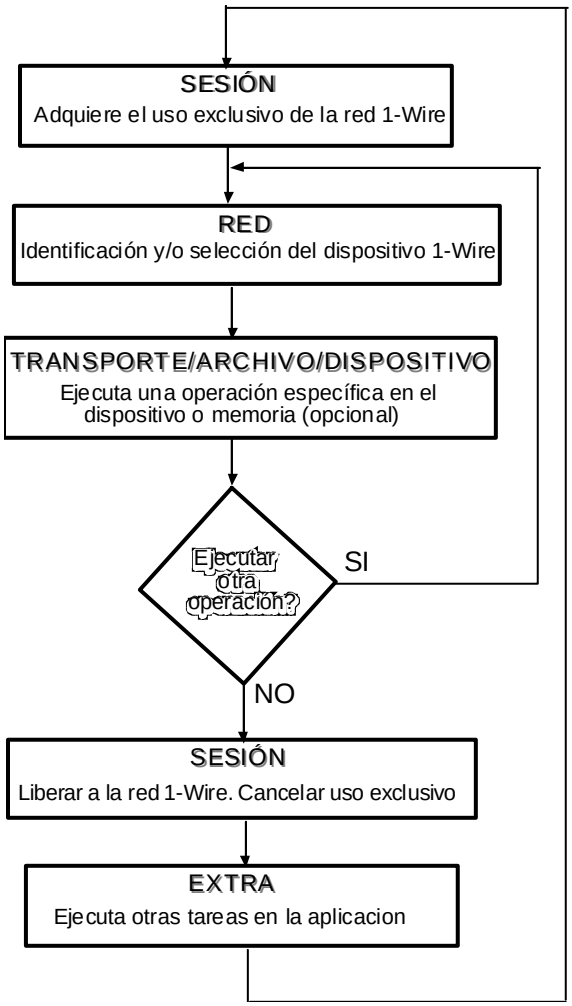


Figura 1.3 Secuencia para uso de funciones API

Existen 5 tipos de API que se presentan en la tabla 1.2 y la tabla 1.3 muestra las API's disponibles para los sistemas operativos (SO) y lenguajes que soportan.

Tabla 1.2. Tipos de API de programación 1-Wire

API	ABREVIATURA	DESCRIPCIÓN
1-Wire Public Domain	**PD**	Conjunto completo de códigos y funciones abiertas en lenguaje "C", los cuales, soportan la conexión con un PC a través de un adaptador tipo serial denominado *DS9097U* .
1-Wire API for Java	OWAPI	Conjunto completo de códigos y funciones abiertas en lenguaje Java, los cuales, soportan casi todos los dispositivos 1-Wire y la conexión con un PC a través de un adaptador tipo serial denominado DS9097U ó equivalente y DS9490 USB.
1-Wire API for .NET	OW.NET	Es el OWAPI compilado con J# para el entorno de Microsoft .NET. La capa de enlace de 1-Wire de bajo nivel ha sido llevada a C# y está disponible para descargas.

1-Wire COM	OWCOM	Modelo de componentes objetos (COM) que implementa un interfaz en base a códigos y funciones escritas y utilizadas por el API de JAVA (OWAPI), el cual, es accesible a través de lenguajes de programación como el Java y el Visual Basic Script.
TMEX API	TMEX	Conjunto completo de funciones independientes del lenguaje. Proveen soporte a todos los dispositivos y adaptadores 1-Wire que trabajen bajo la plataforma Windows de 32 Bits. El API TMEX, está diseñado para trabajar en aplicaciones multiprocesos y multitareas.

Tabla 1.3. Lenguajes y SO que admiten funciones API

SISTEMA OPERATIVO	LENGUAJE		
	TMEX/OWCOM/ OW.NET	**C**	**JAVA**
Windows Vista	TMEX/OWCOM/OW.NET	PD	OWAPI
Windows XP	TMEX/OWCOM/OW.NET	PD	OWAPI
Windows 2000	TMEX/OWCOM/OW.NET	PD	OWAPI
Windows ME	TMEX/OWCOM/OW.NET	PD	OWAPI
Windows 98	TMEX/OWCOM/OW.NET	PD	OWAPI
Windows 95	TMEX	PD	OWAPI
Win 3.1		PD	
DOS		PD	
Palm		PD	
VISOR		PD	
Pocket PC/CE	OW.NET	PD	
Linux y otros SO's basados en UNIX		PD	OWAPI
TINI (plataforma embebida con un SO basado en Java hecho por Dallas)		PD – sin SO TINI	OWAPI

1.6 Posibles dispositivos 1-Wire para nuestra aplicación

En la actualidad existe una gran diversidad de dispositivos 1-Wire. Si nos fijamos en la finalidad de nuestro trabajo, monitorización de la temperatura de un lago, podemos encontrar desde sensores de temperatura (DS1920), de humedad (TAI8540D), de temperatura y humedad (DS1923), de presión atmosférica (TAI8570), de radiación solar (S3-R1), pluviómetros (TAI8575B) …

En nuestro trabajo nos centraremos únicamente en los sensores de temperatura (DS1920), pero con el software desarrollado podremos ampliar la gama de dispositivos de una manera rápida y sencilla.

1.7. Sensor de Temperatura DS1920

Como acabamos de comentar, para monitorizar la temperatura hemos elegido el iButton DS1920.

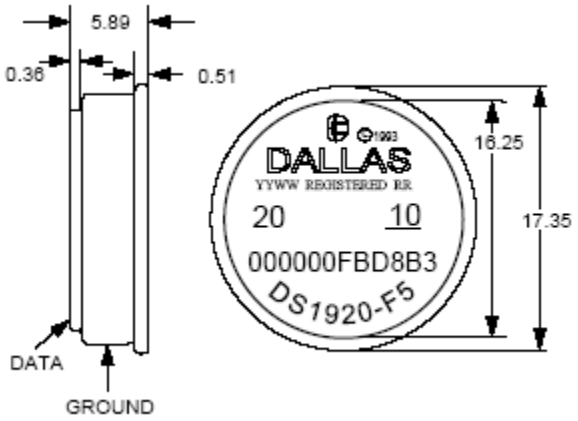


Figura 1.4 iButton DS1920

Un iButton es un chip encapsulado en acero inoxidable. Para mantener el costo bajo, la interconexión eléctrica se ha reducido al mínimo absoluto, una línea de datos y masa. Los iButtons permiten a los usuarios tener información en el transporte y la identificación de datos en un sistema completamente electrónico. Hay modelos con memoria, que actúa como almacenamiento intermedio, recopilando la información aisladamente de la red. La información entonces, se deposita en la red con un simple contacto. En contraste con las etiquetas de papel, las Memorias iButtons se pueden leer y escribir, haciéndolas reutilizables para un número virtualmente ilimitado de ciclos. Los iButton tienen alta inmunidad a la tensión mecánica, a los campos electromagnéticos y a la suciedad. Se pueden reprogramar con la misma sonda que los leen. Con los iButtons se consigue una gran flexibilidad y una excelente relación precio/prestaciones, basándose en la producción en masa.

1.7.1 Componentes principales

En el siguiente diagrama (Figura 1.6) podemos observar los componentes principales del DS1920:

- Memoria ROM de 64 bits
- Sensor de Temperatura
- Alarma de temperatura, TH (Hihg Temperature) y TL (Low Temperature)

Todos los dispositivos 1-Wire contienen una dirección única de 64-bits. Esta dirección consiste en tres porciones distintas, como muestra la figura:

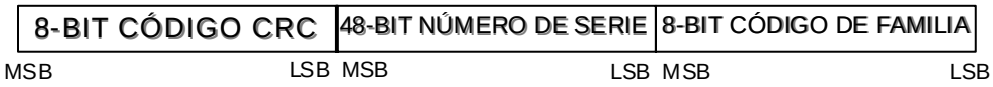


Figura 1.5 Direccion 1-Wire

El código de la familia se utiliza para determinar el tipo (o la familia) del dispositivo 1-Wire y por lo tanto saber los servicios que proporciona. Los 8 bits de CRC se utiliza para asegurar la integridad de la familia y de la identificación del dispositivo.

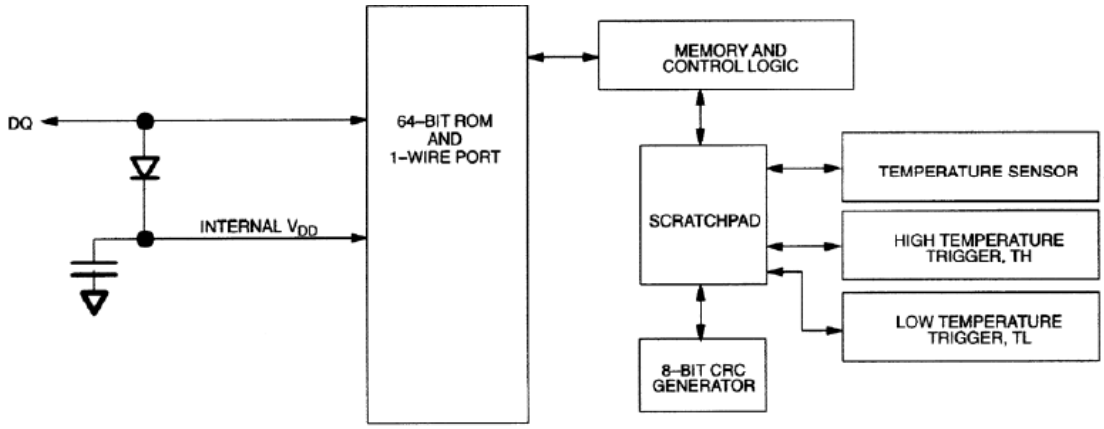


Figura 1.6 Diagrama de Bloques DS1920

El diagrama de la (Figura 1.6) nos muestra el circuito en modo *Energía Parasita*. El circuito roba energía cuando el bus emite con señal alta y lo almacena en su condensador interno para poder seguir trabajando hasta que vuelva haber otra señal alta y asi poder recargarse.

Para poder convertir las medidas se necesita tener una intensidad de 1mA durante la conversión y esto seria demasiado si en un mismo bus existieran muchos dispositivos que estuviesen convirtiendo datos al mismo tiempo.

Para solventar este problema se puede colocar un transistor MOS que conecte la línea de datos con la fuente de alimentación (Strong pullup).

1.7.2 Medición de Temperatura

La temperatura se obtiene en un formato de módulo y signo de 9 bits.

Se observa que el bit más significativo (MSB) corresponde al signo y que el bit menos significativo tiene un valor de 0.5ºC. En la Figura 1.7 podemos observar la representación de -25ºC.

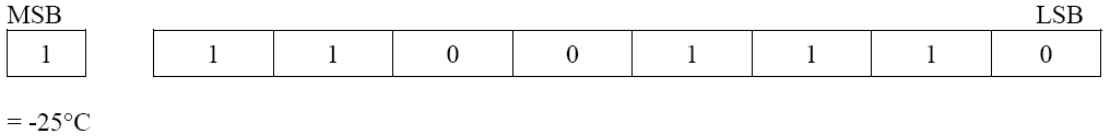


Figura 1.7 Representación de una medida

Tabla 1.4. Relaciones entre datos y temperaturas

TEMPERATURA	BINARIO	HEXADECIMAL
+100ºC	00000000 11001000	00C8H
+25 ºC	00000000 00110010	0032H
+0.5 ºC	00000000 00000001	0001H
+0 ºC	00000000 00000000	0000H
-0.5 ºC	11111111 11111111	FFFFH
-25 ºC	11111111 11001110	FFCEH
-55 ºC	11111111 10010010	FF92H

1.7.3. Memoria del DS1920

La memoria del DS1820 está organizada como se muestra en la Figura 1.8. Consiste en una memoria de *notas* (Scratchpad) SRAM con almacenamiento no volátil en una memoria EEPROM para los registros de Alarma Alta (TH), Alarma Baja (TL). Si los registros de alarma alta y baja no se usan, éstos pueden servir como memoria de propósito general.

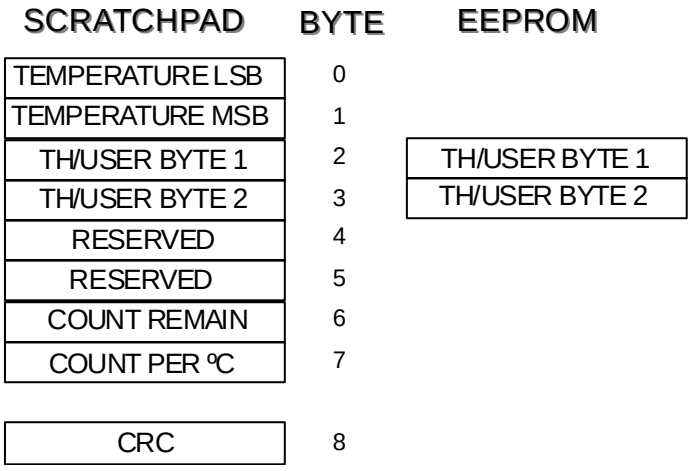


Figura 1.8 Organización Memoria DS1920

Los bytes 0 y 1 de la scratchpad contienen la parte baja y alta respectivamente del registro de temperatura. Estos bytes son sólo de lectura. Los bytes 2 y 3 corresponden a los registros de alarmas alta y baja. Los bytes 4 y 5 están reservados para uso interno por el dispositivo y no puede ser sobre escritos; estos bytes retornarán 1's si son leídos. Los bytes 6 y 7 son contadores que utiliza el sistema para obtener la temperatura más precisa. Por ultimo, el byte 8 de la scratchpad es sólo de lectura y contiene el control de redundancia cíclica (CRC) para los bytes 0 hasta el 7. Los datos son escritos en los bytes 2 y 3 usando el comando de memoria Escribir Scratchpad [4Eh].

Como se explicó a principios del presente capítulo, para iniciar la comunicación con un dispositivo 1-Wire, el master envía un pulso de reset, recibe el pulso de presencia, y una vez transmitido el comando de ROM y haber hecho conexión con el dispositivo, luego se debe enviar un comando de memoria.

Los comandos con los que cuenta el DS1920 son los siguientes: Escribir Memoria Scratchpad [4Eh], Leer Memoria Scratchpad [BEh], Copiar Scratchpad [48h], Convertir Temperatura [44h] y Recordar EEPROM [B8h].

Escribir Memoria Scratchpad [4Eh]: Este comando le permite al master escribir 2 bytes de datos en la memoria scratchpad. El primer byte de datos es escrito dentro del registro TH (byte 2 de la scratchpad) mientras que el segundo byte es escrito dentro del registro TL (byte 3 de la scratchpad). Los datos deben ser transmitidos empezando por el bit menos significativo. Los 2 bytes deben ser escritos antes que el master realice un Reset o los datos pueden ser corrompidos.

Leer Memoria Scratchpad [BEh]: Este comando le permite al master leer el contenido de la memoria scratchpad. La transferencia de datos se inicia con el bit menos significativo del byte 0 y continúa hasta que el byte 8 de la memoria scratchpad (byte de CRC) es leído. El master puede realizar un reset para finalizar la lectura en cualquier momento.

Copiar Scratchpad [48h]: Este comando copia el contenido de los registros TH y TL de la memoria scratchpad (bytes 2 y 3) en la memoria EEPROM. Si el dispositivo se usa en modo parásito dentro de los próximos 10 µs (máx.) después de que este comando se ejecute el master debe habilitar el Strong Pull-up en la línea de datos durante 10 ms para permitir que el dispositivo tenga la energía necesaria para realizar la grabación de los datos en la memoria EEPROM. Al igual que el comando Convertir Temperatura el master recibirá 0 si la copia está en progreso o 1 cuando ya haya sido realizada.

Convertir Temperatura [44h]: Este comando inicia una conversión de temperatura simple. Luego de la conversión, el resultado digital es almacenado en el registro de temperatura de 2 bytes en la memoria scratchpad y posteriormente el dispositivo regresa a su estado de baja potencia. Si el dispositivo está siendo usado con alimentación parásita dentro de los próximos 10 µs (máx.) después de que este comando se ejecute el master debe habilitar el Strong Pull-up en la línea de datos para la duración de la conversión.

Recordar EEPROM [B8h]: Este comando recuerda los valores de los registros de alarma (TH y TL) y los ubica en los bytes 2 y 3 respectivamente de la memoria scratchpad. El master ejecuta este comando y luego genera una ventana de lectura para que el DS1920 le indique el estado del recordatorio transmitiendo 0s mientras la operación esté en progreso y 1s cuando se haya realizado. Una vez que el comando se haya ejecutado satisfactoriamente, los datos estarán disponibles en la memoria scratchpad.

Después de haber realizado cualquiera de estas operaciones, se debe regresar a la secuencia de inicialización para esperar otro comando de memoria.

1.8. Controlador DS2480B

Como master hemos decidido utilizar el DS2480B, dado que sus características se ajustan a nuestras demandas. El DS2480B es un chip que permite trabajar con todos los dispositivos 1-Wire, y permite trabajar utilizando el puerto serie. Puede operar a velocidad regular, flexible y Overdrive, además de ser capaz de traducir cualquier trama recibida en RS-232 a su respectivo equivalente en el protocolo 1-Wire. Por otra parte puede trabajar a tasas de transmisión de 9600, 19200,57600 y 115200 bits por segundo para comunicarse al puerto serie.

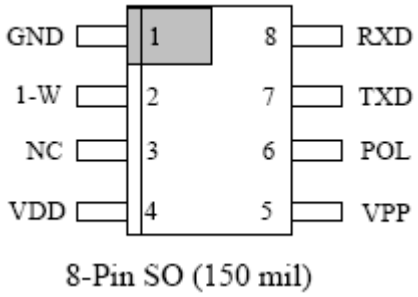


Figura 1.9 Pin's DS2480B

Tabla 1.5. Descripción pin's DS2480B

DESCRIPCION PIN'S	
GND	Masa
1-W	1-Wire Input/Output
NC	No Conexion
V_{DD}	4.5V a 5.5V
V_{PP}	Opcional EPROM Voltage programable
POL	RXD/TXD Seleccionar Polaridad
TXD	Transmitir
RXD	Recibir

El DS2480B se encarga de controlar el correcto funcionamiento del bus 1-Wire. Puede estar en distintos modos; dos estáticos y varias condiciones dinámicas, como se muestra en el diagrama de transición de estados (Figura 1.10).

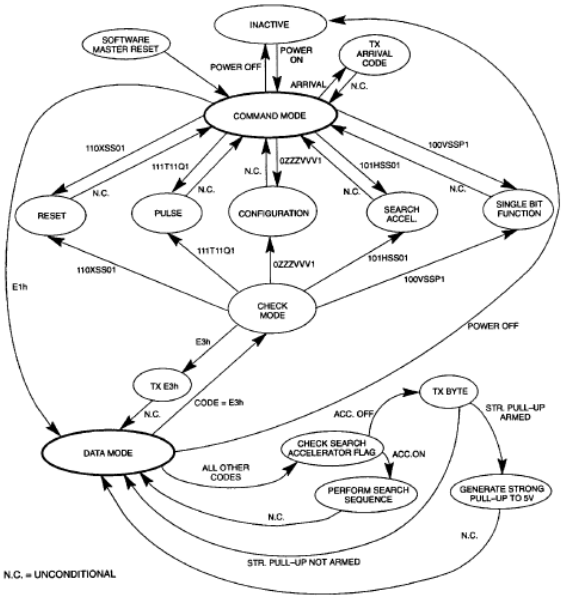


Figura 1.10 Diagrama de estados DS2480B

En el modo Command Mode el chip se dispone hacer alguna operación dentro del bus, ya sea mandar un reset, un pulso, funciones de configuración, activar/desactivar search accelerator o funciones de bit. Hay que diferenciar entre los comandos de configuración y los de comunicación. Todos estos comandos son de 8 bits. El master es el que decide si se produce cambio de modo estatico y pasar al segundo modo, el Data Mode. En este modo el chip quiere enviar comandos directamente a algun dispositivo del bus. Para hacer estas conmutaciones se envian unos comandos reservados.

Como se puede observar en la figura 1.10, existe un modo temporal, Check Mode, en el que el dispositivo espera a recibir el siguiente byte para saber si debe conmutar de estado o no.

1.9 Adaptador DS9097U

Para poder llevar a cabo nuestra comunicación con el bus 1-Wire, decidimos utilizar el adaptador DS9097U que nos permite pasar de la red 1-Wire a puerto serie.

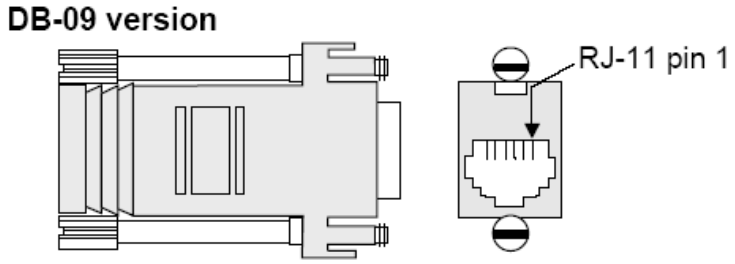


Figura 1.11 Adaptador DS9097U

Esquemáticamente este adaptador se compone como muestra la Figura 1.12:

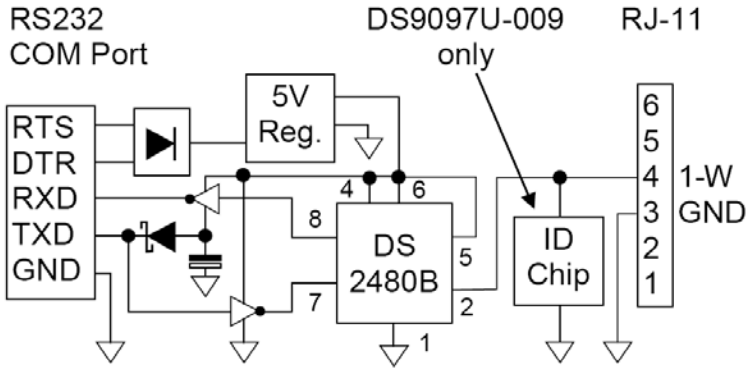


Figura 1.12 Esquema DS9097U Version DB-9

CAPITULO 2. LakeMonitorV1

2.1. Planteamiento

Como hemos comentado anteriormente, nuestra intención es crear una red completa de dispositivos 1-Wire que nos permita monitorizar remotamente la temperatura de un lago. Para ello, crearemos un bus repleto de sensores inteligentes (DS1920) rodeando dicho lago, que conectados al adaptador DS9097U, y éste posteriormente a un adaptador serie/usb nos permitirían establecer conexión con un PC (ver figura 2.1).

Figura 2.1 LakeMonitorV1

2.2. Software desarrollado

Nuestra intención era crear un sistema de monitorización de temperatura remota sin tener que pagar por ningún tipo de software. Necesitábamos software libre y de código abierto, cuyas fuentes estuviesen disponibles públicamente.

Por tanto el camino estaba claro, había que trabajar con Linux. El primer paso sería escoger de entre todas las distribuciones gratuitas, la que más se adaptara a nuestras necesidades.

2.2.1. Instalación del SO

Después de estudiar las principales distribuciones gratuitas de Linux, reducimos los candidatos a tres:

- Ubuntu
- Debian
- Fedora

Pensamos en utilizar Ubuntu, debido a que es una de las distribuciones más extendidas y con más soporte del momento, además de ser de fácil uso en comparación con otros. Una de las cosas que más nos interesaba de Ubuntu era la herramienta de instalación automática de paquetes *apt,* junto a su versión grafica *synaptic*. Finalmente lo descartamos debido a que no era tan compatible con NetMRG como nuestra opción final. De todas formas lo hemos usado para varias tareas como podréis observar en algunas capturas.

Debian fue otra de nuestras posibilidades ya que buscando información encontramos una pequeña adaptación para NetMRG, pero ésta no nos convencía en exceso. Por este motivo nos fuimos directos a la distribución que seguro sería compatible, Fedora.

Decidimos empezar con versiones antiguas ya que la ultima que NetMRG soporta sin compilar es Fedora 3, lo que nos hizo pensar que sería más compatible con versiones antiguas de este SO. Comenzamos con ésta versión y fuimos subiendo, pasando por la 4, 5, 6 y 7 hasta quedarnos con la última, Fedora 8. Las primeras versiones nos daban problemas de compatibilidad con la máquina virtual y con el hardware más reciente, y si probábamos las más actuales nos daba un error causado por un bug que más tarde pudimos solucionar (véase instalación NetMRG).

La obtenemos del siguiente link:

Una vez descargada, comenzamos la instalación. En ella configuramos, entre otras opciones, la contraseña para root o el cortafuegos. Habilitamos en él los puertos asociados al servidor http. Por último creamos una cuenta de usuario.

Una vez instalado correctamente, dejamos el sistema completamente actualizado, como puede verse en la figura 2.2.

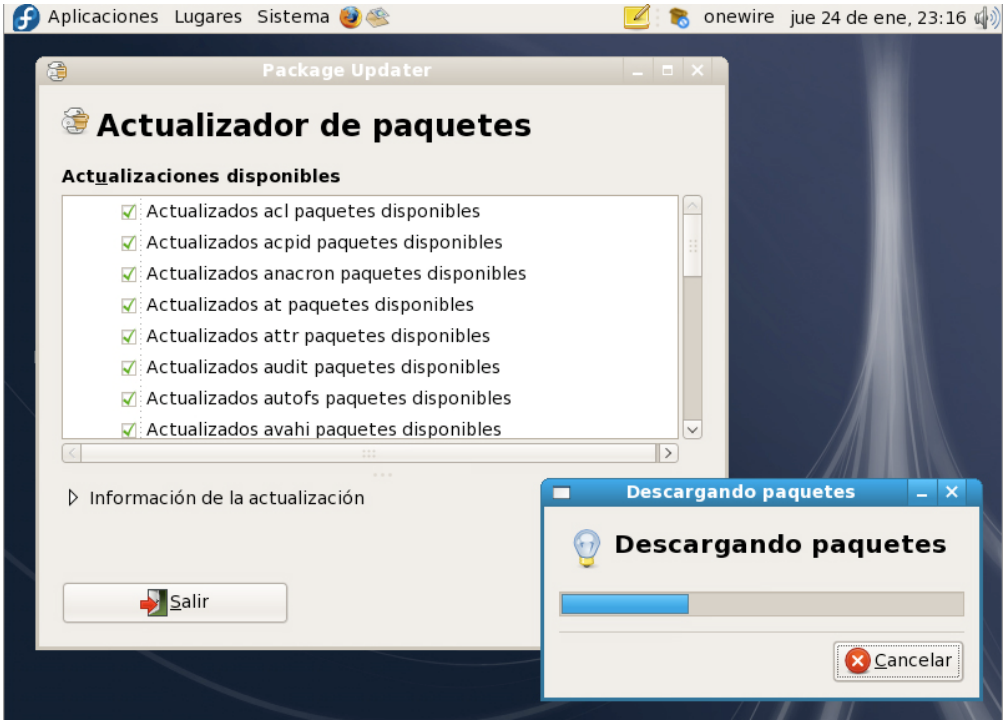


Figura 2.2 Actualización de paquetes.

2.2.2. SDK 1-Wire

Para poder desarrollar nuestros programas y poder establecer la conexión con nuestros dispositivos, utilizamos la API de *Public Domain*. Como podemos observar en la tabla 1.2, nos permite programar en c sobre Linux.

Dallas ha desarrollado un kit de librerías que nos ayudan comunicarnos con el bus. Es lo que se conoce como *1-Wire Public Domain Kit*. Podemos descargarlo desde la siguiente URL:

```
http://www.maxim-ic.com/products/ibutton/software/1wire/wirekit.cfm
```

2.2.3 Funciones utilizadas

A la hora de programar, hemos usado cinco funciones pertenecientes a las librerias ownet.h, findtype.h y temp10.h.

```
int owAcquireEx(char *puerto_conexion)
```

Se conecta al puerto que le indicamos y nos devuelve el número del enlace. Si nos devuelve un valor negativo, nos indica que no ha podido establecer la conexión en el puerto solicitado.

```
SMALLINT FindDevices(int numero, uchar FamilySN[][8],
SMALLINT codigo_Familia, int Numero_maximo_dispositivos)
```

Le pasamos el número de puerto del enlace, el vector vacío de números de serie, el código de familia (sensores de temperatura, por ejemplo) y el número máximo de dispositivos a buscar. La función nos rellena el vector con los números de serie de los dispositivos conectados al bus y devuelve el número total de dispositivos que ha encontrado.

```
void PrintSerialNum(uchar* buffer)
```

Función que nos imprime por pantalla el número de serie que le pasemos.

```
int ReadTemperature(int portnum, uchar *SerialNum, float *Temp)
```

Función que lee la temperatura del dispositivo solicitado. Para realizar la operación le proporcionamos el número de puerto, el número de serie y la dirección dónde debe guardar el valor leído. Nos devuelve un entero positivo en caso de que la lectura haya tenido éxito.

```
void owRelease(int numero_puerto)
```

Función que nos permite liberar la red 1-Wire que hayamos abierto previamente.

2.3. Nuestro software

Gracias al SDK, el funcionamiento interno de nuestros programas resulta bastante sencillo.

Para empezar, necesitamos saber qué sensores hay conectados al bus. Podemos mirar las direcciones físicas directamente de los dispositivos (están grabadas con laser sobre su superfície) pero puede resultar incómodo. Por este motivo realizaremos un programa que nos devuelva una lista con la dirección de cada uno de ellos.

2.3.1. listar

Desde el momento en que sea llamado por el sistema operativo hasta el final de su ejecución, el programa **listar** pasará por los siguientes estados:

1) Intento de conexión al bus. Si lo consigue, se lo indica al usuario y pasa al siguiente bloque. De lo contrario, muestra un mensaje de error y finaliza la ejecución.

2) Realiza una búsqueda de los sensores conectados al bus y guarda la lista en un vector de direcciones físicas que, posteriormente, muestra por pantalla. Si no hay ningún sensor conectado, lo indica.

3) Libera el puerto y avisa al usuario de que se ha desconectado de la red. Finaliza la ejecución.

En este punto, ya podemos identificar cada uno de los sensores conectados.

2.3.1.1. Código listar.c

```
#include <stdlib.h>
#include <stdio.h>
#include "ownet.h"
#include "temp10.h"
#include "findtype.h"

 // Definimos el número máximo de dispositivos y creamos un vector dónde
 // guardaremos los números de série de los sensores del bus.

#define MAXSENSORES  50
uchar VectorSN[MAXSENSORES][8];

 // Función principal

int main(int argc, char **argv)

{
   int i = 0;
   int NumSensores=0;
   int numpuerto = 0;

 // Intento de conexión a la red 1-Wire a través de /dev/ttyUSB0

   if((numpuerto = owAcquireEx("/dev/ttyUSB0")) < 0)
   {
      printf("\nError, no se puede establecer la conexión.\n");
      exit(1);
   }

 // Éxito en el enlace

   printf("\nPuerto abierto.\n");

 // Búsqueda de los sensores de temperatura conectados y impresión por  stdout

   NumSensores = FindDevices(numpuerto, &VectorSN[0], 0x10, MAXSENSORES);
   if (NumSensores>0)
   {
      printf("\n");
      printf("    Sensores encontrados:\n");
      printf("    ¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬¬\n");

      for (i = 0; i < NumSensores; i++)
      {
        printf("      ");
        PrintSerialNum(VectorSN[i]);
        printf("\n");
      }
      printf("\n");
   }
   else
      printf("\n\n\nERROR! No hay dispositivos en el bus!\n\n");

 // Liberar la red 1-Wire

   owRelease(numpuerto);
   printf("Cerrando puerto...\n\n");
   exit(0);

   return 0;
}
```

2.3.2. leertemp

El programa **leertemp** debe devolver el valor de la temperatura de un sensor concreto en el momento en que realicemos la petición. Así, si ejecutamos la orden:

```
./leertemp direcciónsensor
```

El programa mostrará por pantalla su temperatura. Desde el momento en que sea llamado por el sistema operativo hasta el final de su ejecución, **leertemp** se ejecutará de la siguiente forma:

1) Adaptación de la dirección pasada por el usuario a un formato inteligible por las funciones de la librería de Dallas Maxim. Las funciones de la librería necesitan que la dirección este contenida en los 64 bits de un vector de 8 unsignedChar. Es el cometido del primer bloque, ayudándose de nuestra función hexdec.

2) El programa comprueba que le hemos pasado un parámetro, (la dirección del dispositivo) si detecta un número distinto de parámetros, avisa al usuario del error y finaliza la ejecución.

3) Intento de conexión al bus. Si lo consigue, se lo indica al usuario y pasa al siguiente bloque. Si no, muestra un mensaje de error y finaliza la ejecución.

4) Intenta leer el valor de la temperatura del dispositivo especificado. Si lo consigue, la muestra por pantalla. Si no, avisa al usuario con un mensaje de error.

5) Libera el puerto y finaliza la ejecución.

En caso de que no suceda ningún error en la lectura, el programa leertemp únicamente mostrará el valor de la temperatura, sin ningún mensaje adicional (de éxito o de aviso de apertura y cierre de la conexión). La razón es que ésta es la única forma en que el programa puede integrarse con el módulo de representación gráfica que veremos más adelante.

Habiendo entendido la anterior sección y con intención que los programas puedan examinarse de forma más detallada, a continuación veremos sus códigos fuente.

2.3.2.1. Código leertemp.c

```
#include <stdlib.h>
#include <stdio.h>
#include "ownet.h"
#include "temp10.h"
#include "findtype.h"

 // Definimos la variable dónde guardaremos la dirección de nuestro sensor

uchar dispositivo[8];

 // La función hexdec transforma el carácter dado a su valor hexadecimal

int hexdec(char a)
{
        if(a=='0')return 0;
         else if(a=='1')return 1;
         else if(a=='2')return 2;
         else if(a=='3')return 3;
        else if(a=='4')return 4;
        else if(a=='5')return 5;
        else if(a=='6')return 6;
        else if(a=='7')return 7;
        else if(a=='8')return 8;
        else if(a=='9')return 9;
        else if((a=='A')||(a=='a'))return 10;
        else if((a=='B')||(a=='b'))return 11;
        else if((a=='C')||(a=='c'))return 12;
        else if((a=='D')||(a=='d'))return 13;
        else if((a=='E')||(a=='e'))return 14;
        else if((a=='F')||(a=='f'))return 15;
      else return -1;
}

 // Función principal

int main(int argc, char **argv)
{
   float temperatura;
   int i = 0;
   int numpuerto = 0;

 // Convierte la dirección introducida al formato usado por las librerías de
Dallas Maxim

   for (i=0;i<8;i++) { dispositivo[7-i] =
((hexdec(argv[1][2*i])<<4)+(hexdec(argv[1][2*i+1]))); }

 // Comprueba que le hayamos pasado un valor (dirección del sensor)

   if (argc != 2)
   {
      printf("Debes especificar la direccion del sensor\n");
      exit(1);
   }
```

```
 // Intenta conectarse a la red 1-Wire

   if((numpuerto = owAcquireEx("/dev/ttyUSB0")) < 0)
   {
       printf("\nError, no se puede establecer la conexión.\n");
       exit(1);
   }

 // Lee el valor de temperatura del sensor especificado

   if (ReadTemperature(numpuerto, dispositivo,&temperatura))

             {
                printf("%5.1f",temperatura);
             }
   else
                printf("Error de lectura - verifica que el sensor especificado
esta conectado al bus.\n");

 // Libera la red 1-Wire

   owRelease(numpuerto);

   exit(0);
   return 0;
```

2.3.3. Ejecución del software

Una vez escrito el código, lo compilaremos para que pueda ser ejecutado. Para que esta operación se realice de forma correcta, deberemos satisfacer las dependencias de los ficheros de las librerías de Dallas Maxim, desde el Makefile correspondiente. Basándonos en ejemplos encontrados en internet, y adaptándolos para nuestro software, la compilación se realizará de forma correcta usando el fichero Makefile incluído en el anexo 1.

En la siguiente captura podremos ver de forma gráfica cómo se compilan y ejecutan nuestros programas.

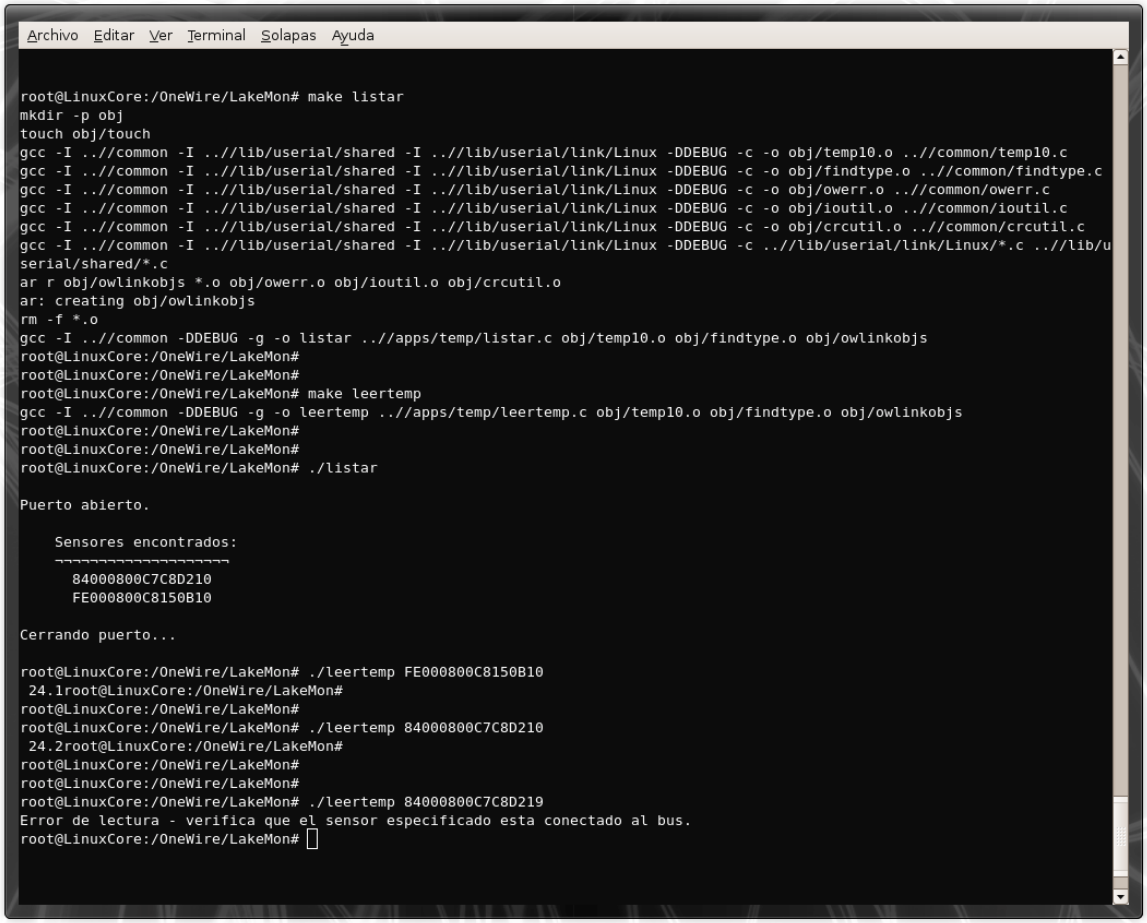


Figura 2.3 Compilación y ejecución

2.4. NETMRG

Llegados a este punto, tenemos acceso fácil a la lista de direcciones físicas de los sensores y al valor de temperatura de cada uno de ellos en el momento en que realicemos la petición. Aunque estos datos podrían ser de utilidad por sí mismos, nosotros buscamos algo más: queremos almacenarlos a lo largo del tiempo y hacer que se muestren por pantalla de forma que puedan interpretarse fácilmente. Desde nuestro punto de vista, la forma más intuitiva de entender valores a lo largo del tiempo es a través de un gráfico.

Necesitamos claramente dos cosas:

1) Una herramienta que nos permita almacenar un listado de valores.

2) Una herramienta que nos permita mostrar estos valores en forma de gráfica.

1) No queremos estar limitados. Si, como ejemplo, colocamos 2000 sensores a lo largo del perímetro del lago, y tomamos una muestra cada 5 minutos, a los 2 años tendremos un total de:

   2 x 365 x 24 x 12 x 2000 = **420.5 millones** de muestras

   Esta cifra no es ninguna tontería. Aquí no valen los ficheros de texto. Creemos que a las muestras les gustaría vivir en un sitio adecuado para sus dimensiones. En una base de datos. Queremos una gratis, de código libre y muy potente. Pensamos que MySQL estará a la altura de las circunstancias.

2) Queremos que las gráficas se puedan crear, en cada momento, a nuestro gusto. Queremos, también, que la herramienta que las cree sea gratis y de código libre. Por pedir, nos gustaría que la herramienta estuviese integrada con alguna base de datos, a poder ser con MySQL. La herramienta existe, y se llama RRDTool. Es el programa de estas características más potente, flexible y el más usado hoy en día.

Ya tenemos casi todo lo que pedíamos: una forma de almacenar los datos y un programa que nos entrega las imágenes en formato PNG en el momento que se las pedimos. *¿Qué más nos falta?*

En ningún momento hemos estado interesados en mostrar los datos directamente desde la máquina virtual. Esto nos limitaría a visualizar todo el trabajo sólo desde un ordenador y a tener el modo gráfico de Linux activado permanentemente. Queremos que todo nuestro trabajo pueda verse desde la mayor parte posible de dispositivos y sin necesidad de instalar nada. No queremos plug-ins ni programas externos a la MV. Queremos HTML puro y duro que nos permita mostrar las imágenes que nos da RRDTool. Como nos gustaría optimizar al máximo el uso de

recursos del ordenador, la mezcla de un servidor web apache con todo lo anterior nos permitiría mantener Linux en RunLevel3, cosa que haría posible dedicar la mayor parte de la potencia del servidor a nuestro LakeMonitor.

Al principio nuestra idea era empezar a crear desde cero esta herramienta, pero nos dimos cuenta de que esperábamos demasiado de ella. Era el interfaz de verdad de nuestro proyecto, la cara con la que el usuario debería interactuar todo el tiempo. Sólo hacíamos que pedir y pedir más funcionalidades, la lista era interminable:

- **Flexibilidad Absoluta.** La herramienta debería poder crear desde un gráfico de la mañana del martes de hace un mes, hasta un gráfico de dos o tres años completos, para evaluar el impacto de las estaciones. O quizá todo a la vez, en una lista, para comparar mejor.

- **Sencillez.** Insertar o eliminar dispositivos y empezar a capturar datos debería ser cosa de pocos segundos, y por supuesto, todo desde la interfaz HTML.

- **Velocidad.** Nada de pasar por todos los valores para dibujar el gráfico de un año, todo debería fluir a la velocidad del rayo.

- **Usuarios independientes,** algunos con derechos de administración y otros únicamente de visualización, por ejemplo, para separar el tráfico procedente del administrador del lago, de la red local y el de internet, o para controlar algunas ramas de sensores específicas. Cualquier número de usuarios simultáneamente. Como en un futuro la herramienta debería permitir el uso de cualquier sensor o actuador 1-Wire, no solo de temperatura, la diferenciación de perfiles con diferentes derechos de acceso era obligada.

Habíamos apuntado demasiado alto. Se escapaba de nuestras manos fabricar algo así en el tiempo del que disponíamos y por supuesto, no había nada existente que cumpliese nuestros requisitos. *¿O sí?*

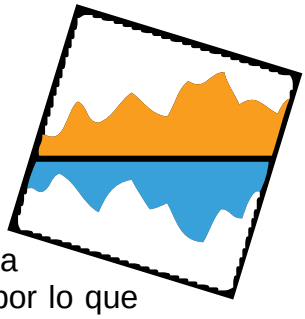
¿Cuál es una de las aplicaciones más extendidas de monitorización remota? La de PC's y servidores, usando el protocolo SNMP. Cuando nos quisimos dar cuenta habíamos llegado a la meta:

NetMRG: La joya de la corona.

El sistema gratuito más extendido de monitorización remota es sin duda Nagios. Además de tener un interfaz, para nuestro gusto, muy feo y anticuado, no realiza gráficas, por lo que no nos sirve para nada. El siguiente en la lista es NetMRG. Hace TODO lo que podíamos soñar de la lista de arriba y mucho, mucho más. A modo de regalo, desde el mismo interfaz podremos monitorizar el uso de la CPU, memoria, bits enviados y recibidos de los interfaces de red, usuarios conectados y cualquier parámetro adicional que imaginemos para nuestro servidor LakeMonitor,

así como cualquier máquina adicional accesible por la red, involucrada en el proceso o no. Podemos monitorizar cualquier cosa de cualquier manera.

No nos lo podíamos creer, NetMRG usa MySQL, RRDtool y Apache. Gratis y de Código libre. Todos nuestros esfuerzos deberían ir ahora orientados hacia la adaptación de este software a nuestra aplicación de temperatura.



2.4.1. Instalación y configuración de NetMRG

Hay dos formas de instalar NetMRG. La más común consiste en realizar la instalación sobre una plataforma soportada directamente por el programa. Este software funciona nativamente sobre RedHat Enterprise Linux 4 y versiones antiguas de Fedora (la más reciente Fedora3). Nosotros no queremos software de pago, por lo que RedHat se descarta directamente. Tampoco queremos sistemas operativos obsoletos, por lo que Fedora3 no resulta una opción a tener en cuenta. Tendremos entonces que empezar de cero y compilar el programa específicamente para nuestro sistema operativo, Fedora 8.

Empezaremos por descargar las fuentes de la última versión del programa, desde la página de NetMRG.

```
http://www.netmrg.net/download.php
```

La última versión en estos momentos es la 0.18.2 y el archivo con las fuentes se llama *netmrg-0.18.2.tar.gz.*

Una vez en nuestro disco, iremos hasta el lugar donde se encuentre a través de la consola de terminal, y procederemos a descomprimirlo.

```
# gunzip netmrg-0.18.2.tar.gz
# tar -xvf netmrg-0.18.2.tar
```

NetMRG ni mucho menos viene solo. Necesitaremos resolver una larga lista de dependencias para que el sistema nos lo deje compilar.

Los paquetes incluyen los ya previstos GCC (que ya tenemos instalado), MySQL y RRDTool, entre muchos otros:

- GCC y GCC Para C++
- MySQL, MySQL Server y MySQL Developer
- RRDTool y RRDTool Developer
- PHP
- Apache 2
- LibXML 2 y LibXML 2 Developer
- Network SNMP, Network SNMP Developer y Network SNMP Tools

GCC y GCC para C++ ya los habíamos instalado anteriormente y el Apache 2 se instaló con el sistema operativo.

Instalamos MySQL, MySQL Server, MySQL Developer, PHP, LibXML2 y LibXML2 Developer directamente a través de la herramienta yum.

```
# yum install mysql mysql-server mysql-devel php
              php-mysql libxml2 libxml2-devel
```

Yum resolverá las dependencias adicionales y nos preguntará si queremos que prepare nuestro sistema. Le responderemos que sí y dejaremos que el resto del proceso transcurra de forma automática.

Nos faltan por instalar los paquetes y las herramientas de RRDTool y NetworkSNMP. Deberemos ahora conseguir las fuentes desde sus respectivas páginas Web.

```
http://oss.oetiker.ch/rrdtool/download.en.html
http://net-snmp.sourceforge.net/download.html
```

Una vez hemos obtenido los paquetes *net-snmp-5.4.1.tar.gz* y *rrdtool-1.2.26.tar.gz* los descomprimiremos y expandiremos desde el terminal de consola. Usaremos los comandos:

```
# gunzip net-snmp-5.4.1.tar.gz
# gunzip rrdtool-1.2.26.tar.gz
# tar -xvf net-snmp-5.4.1.tar.gz
# tar -xvf rrdtool-1.2.26.tar.gz
```

Configuraremos y compilaremos RRDTool y NetworkSNMP, ejecutando las siguientes instrucciones dentro de cada directorio.

```
# ./configure
# make
```

Una vez haya finalizado la compilación sin errores para cada uno de los dos, introduciremos el siguiente comando:

```
# su root
```

Seguido de la contraseña de administrador. De esta forma tendremos permisos suficientes para instalar software en la máquina.

Una vez concedido el acceso, instalaremos los programas ejecutando

```
# make install
```

En cada uno de los directorios. En este punto nuestro sistema se encuentra preparado para configurar la instalación de NetMRG.

Accederemos a la carpeta de NetMRG que hemos descomprimido antes y procederemos a preparar, compilar las fuentes e instalar los archivos en sus correspondientes directorios.

```
# ./configure
# make
# make install
```

Nuestro Fedora está ahora listo para empezar a configurar NetMRG. Empezaremos por configurar la base de datos MySQL. Añadimos a la base de datos el usuario netmrg.

```
# mysqladmin create netmrg
```

Trasladamos la base de datos de NetMRG al sistema con:

```
# mysql -u root -p netmrg < share/netmrg.mysql
```

Algo ha salido mal puesto que obtenemos el siguiente mensaje:

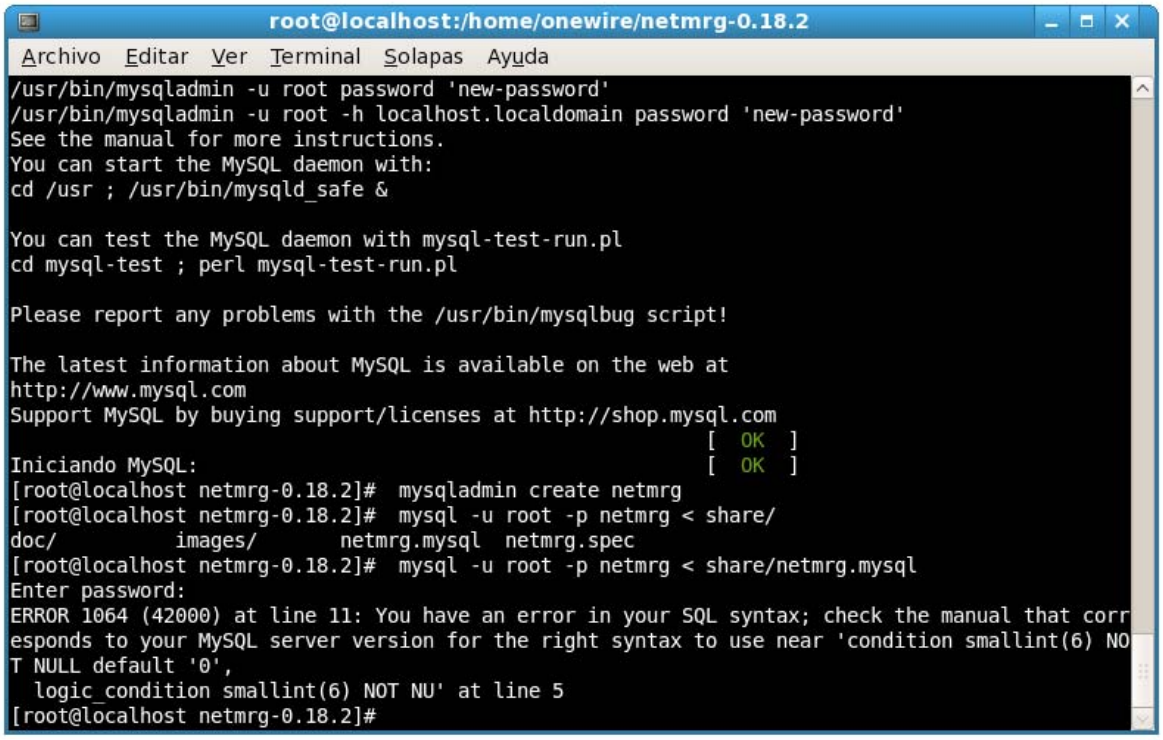
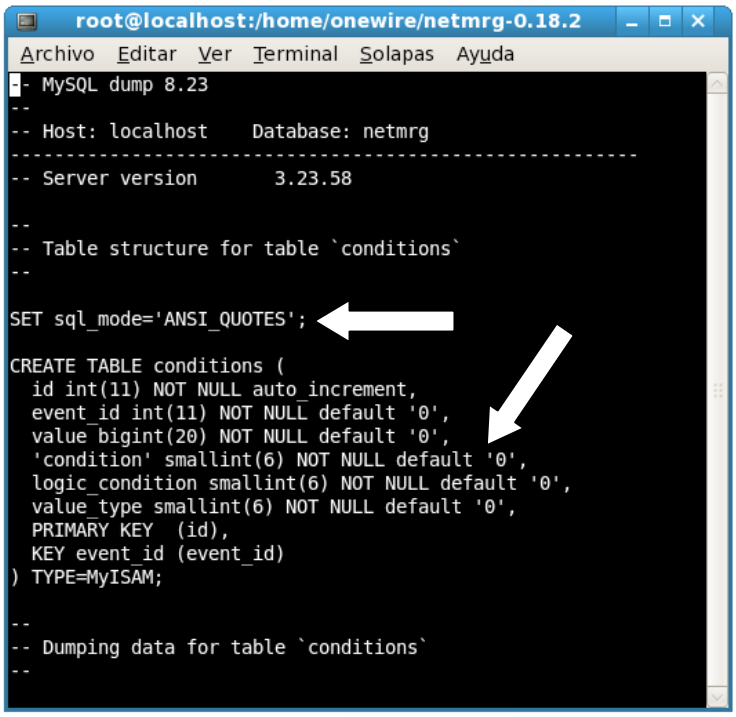


Figura 2.4 Bug NetMRG

Este problema nos tuvo atrapados durante días. Finalmente descubrimos que en algunos sistemas existe una incompatibilidad entre NetMRG y la versión 5 de MySQL. Es un Bug reconocido oficialmente por los creadores de NetMRG.



Como no queríamos realizar un downgrade hacia MySQL4, seguimos investigando por si existía alguna solución alternativa. Si la hay. La cadena de caracteres “condition” pasa a ser una palabra reservada en MySQL 5, si en el lugar del fichero dónde NetMRG la especifica, realizamos un pequeño cambio (hacemos que NetMRG la use con comillas), el problema queda resuelto. Editaremos entonces el fichero *netmrg.mysql* con *vi* y haremos dos cambios.

Antes de la primera tabla introduciremos la línea indicada por la flecha horizontal y le pondremos comillas simples a la palabra *condition* de la línea señalada por la otra flecha, como puede verse en la captura. Ya podemos guardar y salir. Cuando volvamos a introducir el comando que no funcionaba, el sistema lo aceptará sin problemas.

Ahora entraremos como root en la consola de administración de MySQL Server. Para ello introducimos el comando

```
# mysql -u root -p
```

Seguido de la contraseña de administrador de MySQL Server. Por defecto está vacía, por lo que nos limitaremos a pulsar *enter* cuando nos la pida. Una vez dentro, configuraremos el usuario y contraseña de NetMRG en MySQL server introduciendo:

```
> grant all on netmrg.* to netmrguser@localhost
  identified by 'netmrgpass';
```

Salimos de la consola de administración con:

```
> exit;
```

Con esto queda lista la base de datos. Ya podemos pasar a configurar el servidor web Apache.

Editamos la configuración del apache contenida en *httpd.conf*:

```
# vi /etc/httpd/conf/httpd.conf
```

Y debajo de la línea *Include conf.d/*.conf*, indicamos que use también el fichero de configuración de NetMRG. Así, esa parte del archivo debería quedar de la siguiente manera:

```
Include conf.d/*.conf
Include /etc/netmrg.conf
```

Con esto el apache queda listo. Añadiremos ahora al sistema el usuario netmrg y le asignaremos la propiedad de sus carpetas, para que pueda realizar de forma correcta las funciones de monitorización.

```
# useradd netmrg
# chown netmrg:netmrg /var/log/netmrg
# chown netmrg:netmrg /var/lib/netmrg/rrd
```

El ejecutable *netmrg-gatherer* que NetMRG ha instalado en nuestro sistema se encarga de “preguntar” los valores y guardarlos en la base de datos. NetMRG está diseñado para tomar un valor exactamente cada 5 minutos. Para que esto suceda deberemos daemonizar (dejar un programa en ejecución) n*etmrg-gatherer* o introducirlo en el cron (subsistema encargado de ejecutar tareas periódicamente)

con un tiempo entre ejecuciones de 5 minutos. Hemos optado por la primera opción, ya que según se indica en la misma página de los autores, es más recomendable para S.O modernos como Fedora 8.

Ahora ejecutamos el comando:

```
# su netmrg -c '/usr/local/bin/netmrg-gatherer
  -X -S -M wait'
```

Su función es daemonizar netmrg dejando por fin NetMRG totalmente funcional en nuestro sistema. Lo que nosotros queremos es que este comando se ejecute cada vez que conectamos el ordenador. Para conseguirlo, crearemos un script que contenga esta orden y lo introduciremos en el arranque.

Creamos un fichero vacío que contenga la orden anterior, y lo guardamos como *scriptnm.sh* en algún lugar del disco.

Creamos enlaces simbólicos en la carpeta de arranque de los niveles 3 y 5 de Linux, en los que queremos que NetMRG se cargue de forma automática.

```
# ln -s /scriptnm.sh /etc/rc5.d/S99nmrg
# ln -s /scriptnm.sh /etc/rc3.d/S99nmrg
```

Esto hará que al encender la máquina, nuestro script se ejecute al final, cuando el resto de módulos ya se hayan cargado. Esto es importante porque los servicios de MySQL y Apache siempre deberán ir antes.

Pondremos la consola en modo Superusuario para que nos permita arrancar, parar y reiniciar servicios. En Fedora, el comando su root no es suficiente.

Deberemos teclear:

```
# su -
```

seguido de nuestra contraseña de root. Ahora no tendremos problemas para reiniciar el apache y MySQL con:

```
# service mysqld restart
# service httpd restart
```

Justo aquí acaba la instalación de NetMRG. ¡Ya podemos entrar! Abrimos un navegador y vamos (desde la propia máquina) a *http://localhost/netmrg*.

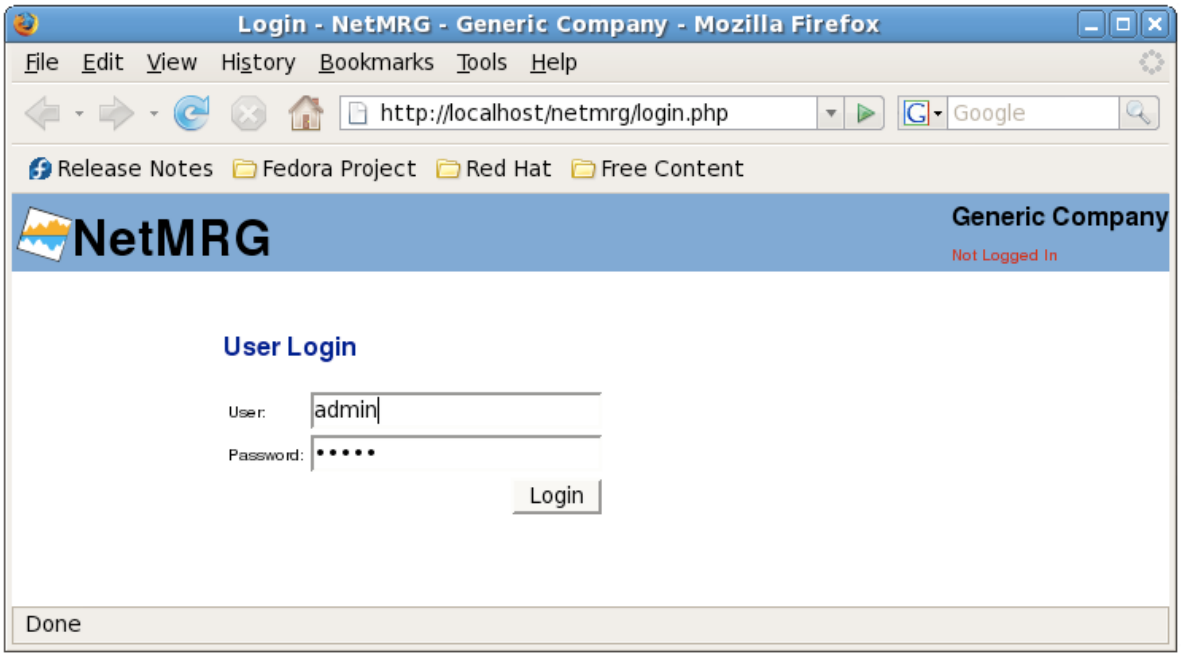


Figura 2.5. Login NetMRG

El nombre de usuario y contraseña por defecto son *admin* y *nimda*.

NetMRG está pensado y diseñado para monitorizar ordenadores a través de SNMP. Es él mismo quien se encarga de gestionar las peticiones a través de este protocolo. Tendremos que configurarlo para hacer algo totalmente distinto. Por suerte, el programa permite monitorizar parámetros a través de scripts externos.

De esta forma podremos capturar datos desde nuestro *leertemp*. Empecemos:

Nos dirigimos a la pestaña *Groups*, y hacemos clic en *add*. Introducimos el nombre del grupo de dispositivos (Lake Monitor) y en el apartado *parent* hacemos clic en *root*, para indicarle que debe colgar de la rama principal. Hacemos clic en *Save Changes*.

Ahora crearemos el tipo de dispositivo *sensor*. Hacemos clic en la pestaña *device types*, y acto seguido en el botón *add*. Le damos el nombre de *Sensor* y hacemos clic en *Save Changes*.

Una vez creado el tipo sensor intentaremos añadir nuestro software *leertemp* al apartado scripts. Para hacerlo pinchamos en la opción scripts en el menú de la izquierda. Rellenamos los campos tal y como se indica en la captura y clicamos en Save Changes. Previamente deberemos haber copiado el ejecutable *leertemp* a la carpeta de scripts de NetMRG, localizada en */usr/local/bin/libexec/netmrg*.

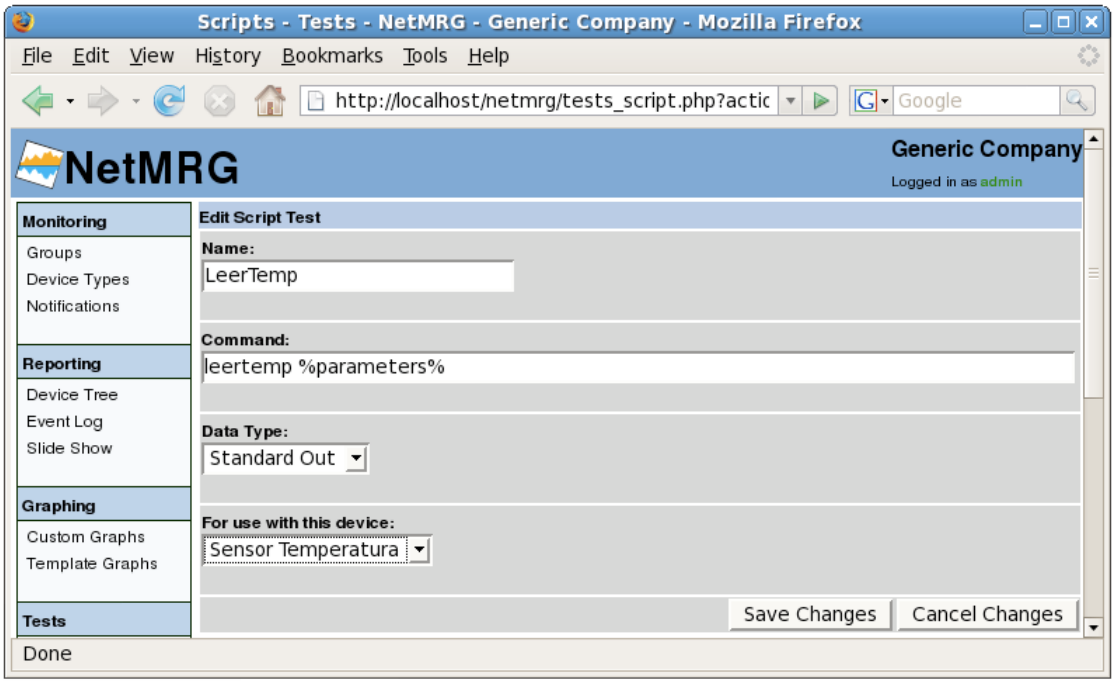


Figura 2.6. Configuración del Script

Llegados a este punto, ya podemos empezar a crear nuestro árbol y añadir el listado de sensores conectados al bus.

Pinchamos en el botón *Groups* y en el grupo que hemos creado antes. Una vez en su interior, en la parte de abajo (Monitored devices) le damos a *add*. En la ventana que aparece hacemos clic en el botón *Add a new device*. En la siguiente ventana de configuración le ponemos como nombre Sector1 y seleccionamos como tipo de diapositivo, *Sensor*. Hacemos clic en *save changes*.
Dentro del subgrupo Sector1 vamos a añadir uno de nuestros sensores 1-Wire. En la ventana en la que nos encontramos ahora hacemos clic en *add*. Configuramos los valores según se indica en la siguiente captura. En el campo donde hay 16 caracteres hexadecimales introduciremos la dirección física del sensor a agregar.

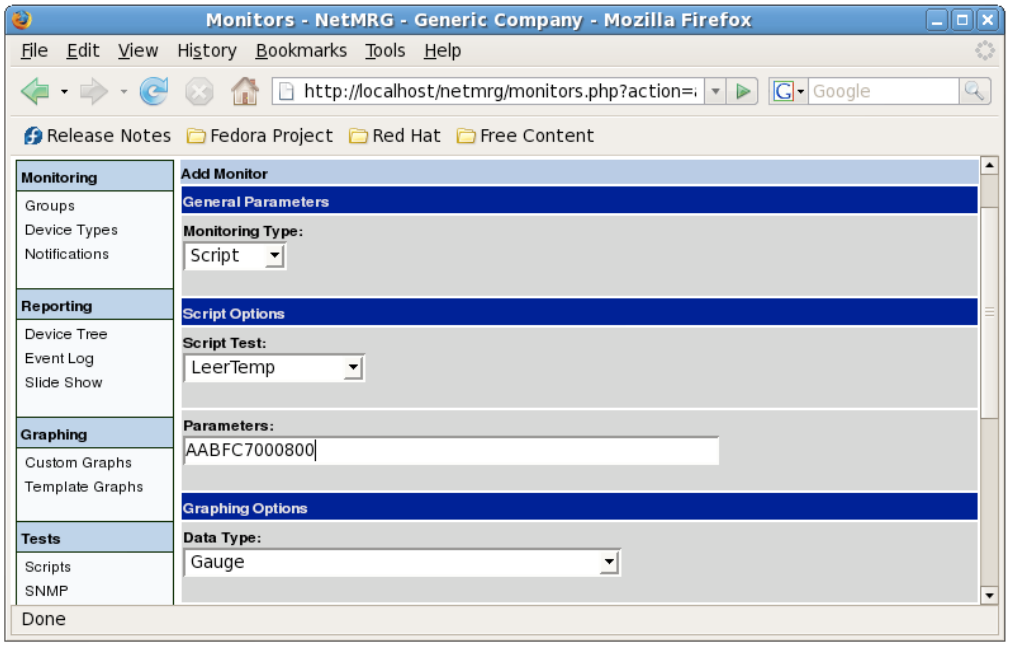


Figura 2.7. Nuevo dispositivo

Guardamos los cambios y vemos como nos aparece una pantalla como la de la siguiente captura. A los 5 minutos, en cuanto *netmrg-gatherer* se ejecute, en el campo *value* aparecerá la temperatura actual y veremos la primera línea de la gráfica en el apartado graph.

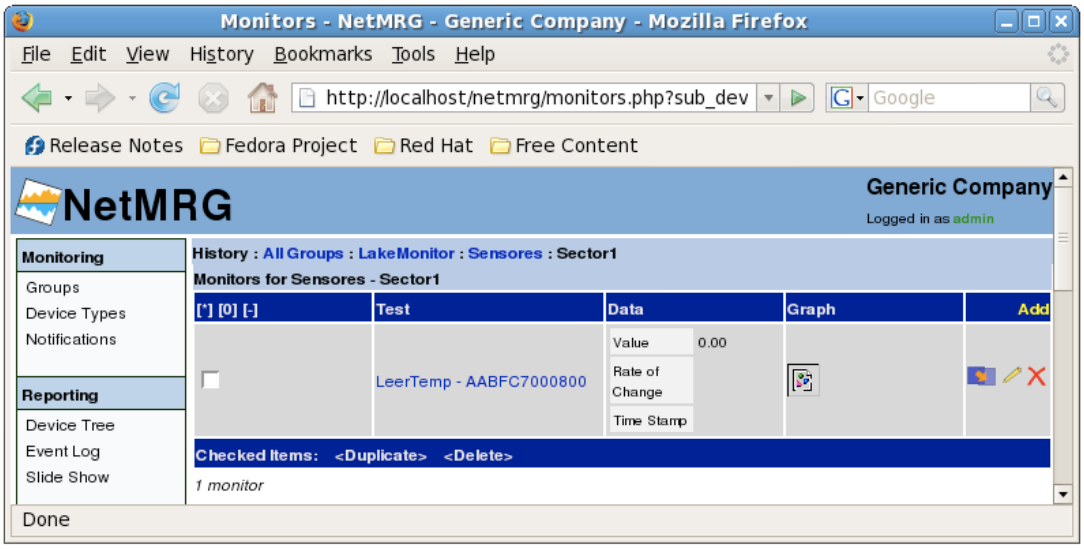


Figura 2.8. LeerTemp&NetMRG

CAPITULO 3. LakeMonitorV2

3.1. Planteamiento

Una vez completado el proyecto LakeMonitorV1, nos marcamos metas superiores. Queremos implementar una aplicación que nos permita incorporar nuevos tipos de sensores (*véase 1.6*) a nuestro bus de una manera rápida y sencilla, además queremos tener la posibilidad de separar físicamente el bus 1-Wire del ordenador que contiene el servidor con las posibilidades que nos ofrece el tráfico a través de redes ip: ethernet, wireless, internet ….

También queremos que nuestro software esté en constante desarrollo, es decir, que se pueda ir actualizando y mejorando al mismo tiempo que la lista de dispositivos 1-Wire disponibles vaya aumentando.

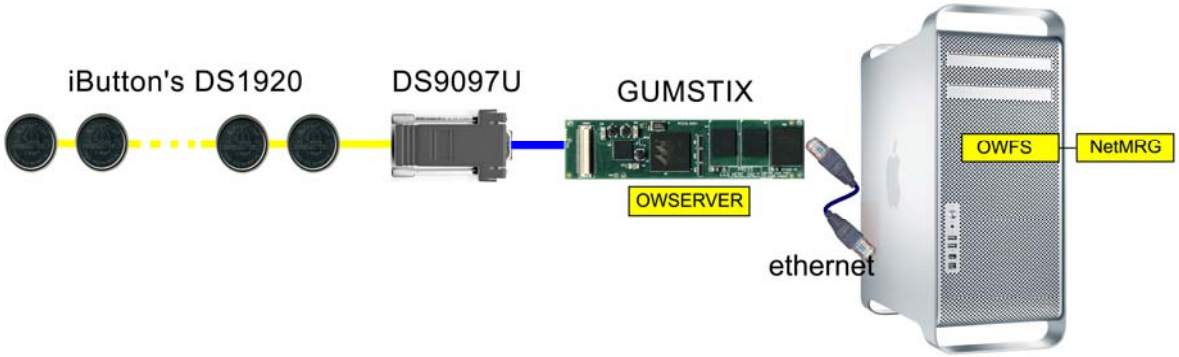


Figura 3.1. LakeMonitorV2

3.2. Virtualización

La virtualización consiste en crear un entorno no real, que simule de la forma más fiel posible las características de lo que estamos intentando clonar. En el caso de la virtualización de sistemas operativos, estamos intentando crear nada más y nada menos que un ordenador no físico, cuyo SO no debe notar ninguna diferencia respecto una máquina de verdad. Éste debe creer que está rodeado de hardware.

Porqué puede interesarnos en nuestro caso montar todo el sistema sobre una máquina virtual? Principalmente por dos razones:

- **Sencillez y comodidad:** Sólo debe ser creada una vez. Cuándo la máquina virtual esté lista, podrá ser trasladada y duplicada infinidad de veces, en infinidad de ordenadores con distinto hardware, sin tener que reconfigurar ni un sólo bit de su interior. No hará falta formatear ningún disco ni particionar. Así los datos del ordenador que la contenga no correrán ningún peligro.

- **Seguridad:** La máquina virtual corre de forma paralela a los distintos programas que se estén ejecutando sobre el SO real del PC, por lo que en caso de cuelgue del sistema, virus, contaminación exterior o agujero de seguridad en la MV, el resto de programas nunca quedarán expuestos a peligro alguno.

Es frecuente pensar que la potencia de la máquina en cuestión quedará altamente mermada. Si bien es cierto que la potencia de cálculo de la máquina virtual no es la misma, usando un software de virtualización maduro como es nuestro caso, ésta será aproximadamente un 5%-10% inferior que en la máquina real. La memoria RAM disponible será asignada manualmente por el administrador en cualquier momento y el espacio ocupado en disco irá creciendo a medida que lo haga la MV. Puede incluso asignarse un disco propio para la máquina, incrementando así la velocidad de lectura.

3.2.1. VMWare

A la hora de decantarnos por un software de virtualización concreto, la única opción que permite su uso en los 3 principales sistemas operativos con total compatibilidad (Windows, Linux, MacOS) es VMWare. Si además le sumamos que las versiones básicas de su software son gratuitas (VMWare player y VMWare server) y que es con diferencia el software de virtualización más robusto y extendido, tanto en casas como en pequeñas y grandes empresas, no cabe duda de que ésta será nuestra elección.

En el caso de windows, su instalación es extremadamente sencilla. Únicamente es necesario descargar el programa, abrir el ejecutable y seguir las instrucciones que nos proporciona el propio software. Para MacOS X, la instalación es parecida y de

igual o mayor sencillez, aunque no existen los productos anteriormente mencionados. El programa para Mac se llama VMWare Fusion. Las máquinas virtuales creadas por éste son, por supuesto, intercambiables. En el caso de Linux, para instalar cualquier producto VMWare, hay que hacerlo desde consola, además de que nos pide como requisitos indispensables disponer localmente del código fuente del kernel que estemos usando, tener instaladas las herramientas para compilación GCC y recompilar programa para nuestro núcleo. Dado que la instalación es prácticamente idéntica a la de VMWare tools, si se entienden los pasos a seguir con los drivers, no debería haber ningún problema. Es por ello que aquí no la documentaremos explícitamente.

3.2.1.1 Creación de la máquina virtual

Para crear la MV escogemos el sistema operativo que deseemos usar, en nuestro caso seleccionamos *Otro Linux 2.6.x Kernel*. Le damos un nombre y la localización que queramos:

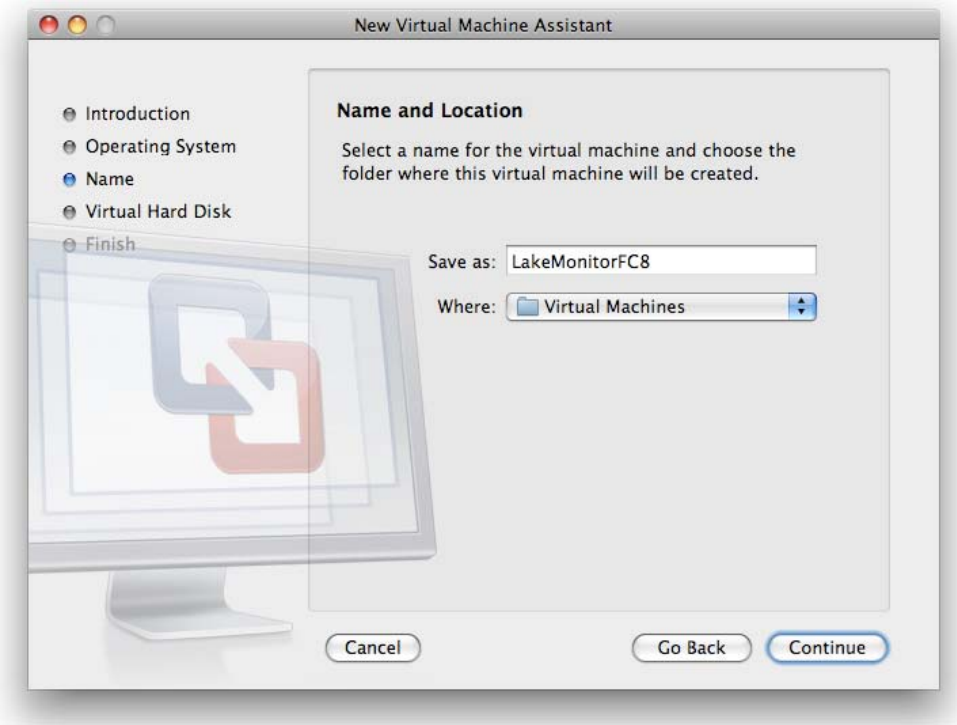


Figura 3.1. Creación de la máquina virtual

Más tarde seleccionamos el tamaño de disco que deseemos, aunque como ya hemos comentado el valor de éste ira aumentando a medida que la máquina lo vaya necesitando. En el último paso, seleccionamos la imagen del SO a instalar y reiniciamos. Una vez haya hecho esto, se instalara el SO y podremos disfrutar de la MV, aunque para poder usar de forma optima esta máquina, es indispensable instalar, una vez cargado correctamente el sistema operativo, los drivers de los dispositivos virtuales. Éstos drivers reciben el nombre de VMWare tools.

3.2.1.2. Instalación VMWare tools

En primer lugar instalamos el gcc, gcc para c++ y las fuentes del kernel.

```
# yum install gcc kernel-devel gcc-c++
```

Luego comprobamos que nuestra versión corresponda con las funciones del kernel recién instaladas, es decir, que éstas funcionen correctamente con nuestro kernel.

```
# uname -r              # running kernel
# rpm -q kernel-devel   # installed kernel headers
```

Si no coincide, tendremos que actualizar nuestro kernel a la versión correspondiente.

```
# yum -y upgrade kernel kernel-devel
# reboot
```

Una vez tenemos el kernel correcto, podemos comenzar la instalación de VMWare tools.

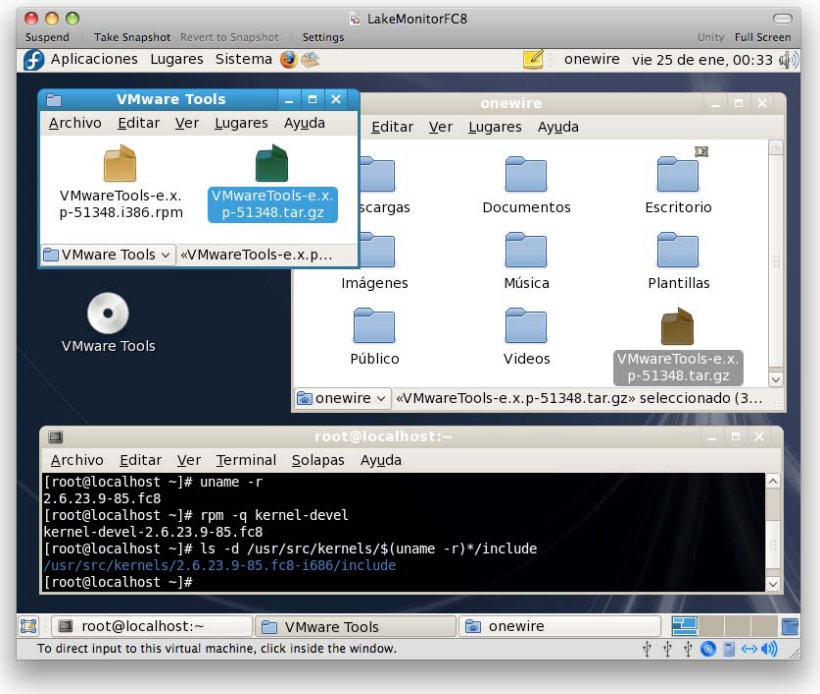


Figura 3.2 Instalación VMWare tools

Descomprimimos el archivo *VMWareTools-e.x.p-51348.tar.gz*

```
# gunzip VMWareTools-e.x.p-51348.tar.gz
# tar -xvf VMWareTools-e.x.p-51348.tar
```

Y comenzamos la instalación con

```
# ./vmware-intall.pl
```

Posteriormente indicamos los directorios donde queremos que se instalen las funciones, para más tarde compilarlas utilizando el código fuente de nuestro kernel. Una vez finalizado el proceso, ya tenemos los drivers necesarios para utilizar VMWare tools sin ningún tipo de problema.

3.3 OWFS

Aunque el software que hemos programado es perfecto para nuestra aplicación, en la segunda versión de LakeMonitor hemos decidido utilizar una plataforma completamente distinta para interactuar con el bus 1-Wire. Un sistema integral de comunicación con los dispositivos que en ningún momento hace uso del PublicDomainKit proporcionado por Dallas Maxim. Ahora ya no disponemos de una librería de funciones C con las que escribir nuestro programa, sino que accederemos a cualquier dispositivo del bus de la misma forma en que interactuamos con el sistema de ficheros en linux; usando las funciones de entrar, salir de directorios y listar información que nos proporciona el sistema operativo.

OWFS es su nombre, y significa OneWireFilesystem (Sistema de ficheros OneWire). Existen varias razones por las que hemos cambiado nuestros propios programas “listar” y “leertemp” por esta suite de software:

- Permite el uso sin modificación de cualquier dispositivo 1-Wire distribuido por Dallas Maxim y por el 95% de los producidos por empresas ajenas que usen éste protocolo. Las posibilidades son infinitas. Sin una sola línea de código, podemos usar directamente cualquier sensor, actuador, switch, maestro o dispositivo existente a día de hoy en el mercado sin adaptaciones. Enchufar y funcionar. Auténtico plug&play para 1-Wire.

- Evoluciona por sí mismo. Hoy por hoy, cientos, quizá miles de personas contribuyen desinteresadamente en el proyecto OWFS, cosa que nos asegura estar siempre al día sin ningún esfuerzo.

- Suite con Infinidad de módulos que encajan perfectamente entre ellos. OWFS es sólo una pieza del puzzle. Para nuestro LakeMonitorV2, el hecho de separar el enlace al bus 1-Wire con el interfaz de los datos es exactamente lo que veníamos buscando; será el módulo OWServer, que introduciremos dentro del Gumstix, el que nos permita realizar múltiples sesiones desde varios clientes con OWFS, que a su vez serán servidores de gráficas. Con OWhttpd podremos monitorizar desde cualquier lugar y a través de una interfaz HTML, el estado exacto del bus, haciendo posible, por ejemplo, diagnosticar errores en tiempo real. También permite al administrador establecer cualquier parámetro de configuración interno de forma independiente para cada dispositivo conectado.

3.3.1. Instalación OWFS

En LakeMonitor v2, usaremos:

OWFS: Como ya hemos comentado, se encarga de montar en la máquina un sistema de ficheros virtual, con el contenido completo del bus 1-Wire. Con él podremos visualizar y configurar todos y cada uno de los parámetros de cada dispositivo conectado, de manera sencilla e intuitiva. Para entrar o salir de una carpeta, utilizaremos el típico *cd*, y para listar alguna información, por ejemplo, el comando *cat*.

OWFS utiliza el software FUSE (Filesystem in Userpace). Éste es el encargado de montar un sistema de ficheros inexistente y vivo, con el contenido que OWFS le indique en cada momento. Así, al conectar o desconectar dispositivos, los cambios se reflejarán en el sistema de forma automática.

OWHttpd: Funciona de forma muy similar a OWFS. La principal diferencia radica en que no usa FUSE para montar el Filesystem, sino que vuelca todo el contenido sobre Apache, para colocar cada dispositivo y sus respectivos parámetros de configuración o diagnóstico al alcance de cualquiera conectado a la red. En nuestra aplicación conectaremos OWFS con el subsistema de representación de gráficas y desde OWHttpd realizaremos cualquier operación adicional. De esta forma no dependeremos de conectarnos por terminal remoto ni será necesario hacer uso del modo gráfico en la máquina virtual.

OWServer: Es un pequeño módulo al que se conectan OWFS y OWHttpd. Como ya se ha comentado, su única función es la de permitir separar físicamente el bus 1-Wire de la máquina virtual, a través de un enlace IP.

Pasaremos ahora a realizar la instalación en el sistema.

Empezaremos por descargar, de sus respectivas páginas, las versiones más recientes de OWFS y FUSE.

```
http://owfs.org/index.php?page=download
http://fuse.sourceforge.net/
```

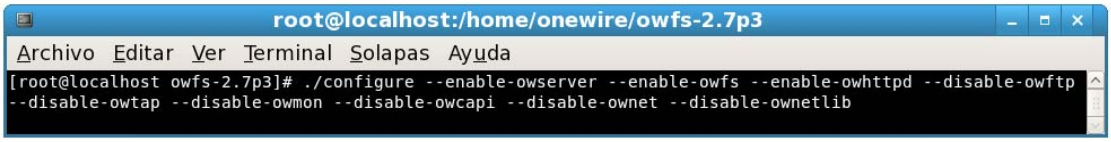
Ya tenemos los ficheros *owfs-2.7p3.tar.gz* y *fuse-2.7.2.tar.gz* (en nuestro caso).Procederemos a descomprimirlos desde la carpeta que los contiene:

```
# gunzip owfs-2.7p3.tar.gz
# tar -xvf owfs-2.7p3.tar.gz
# gunzip fuse-2.7.2.tar.gz
# tar -xvf fuse-2.7.2.tar.gz
```

Ahora instalaremos FUSE. Entramos en la carpeta y realizamos la configuración, compilación e instalación. El último paso nos pedirá la contraseña de root.

```
# ./configure
# make
# sudo make install
```

Es ahora cuando podemos empezar a configurar la instalación de OWFS. Nos dirigimos al directorio correspondiente y tecleamos:



Con esto, le indicamos al configurador de OWFS que solo estamos interesados en usar OWFS, OWServer y OWHttpd, forzándole a no incluir programas que no usaremos para nada.

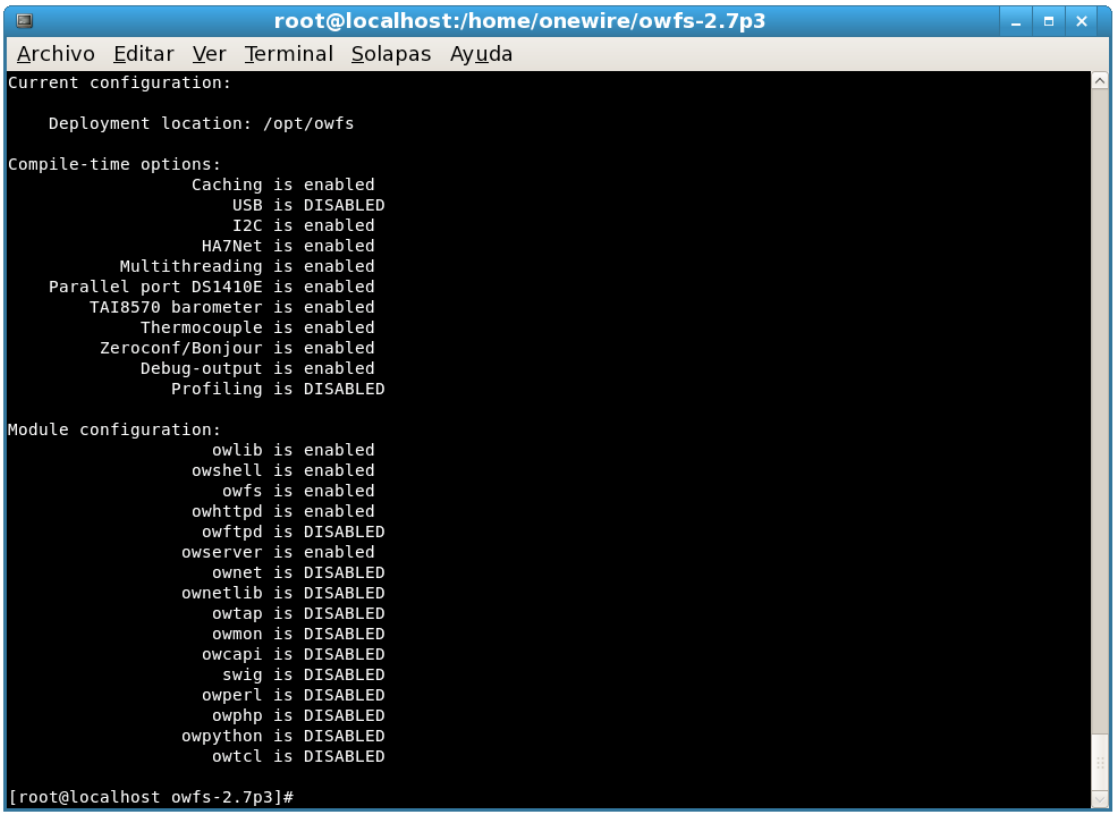


Figura 3.3. OWFS Configurado

Cuando el configurador acabe de preparar los Makefiles correspondientes, nos mostrará la configuración. Es, en este punto, donde comprobamos que FUSE se instaló correctamente. De no ser así, no se hubiese incluido OWFS aún habiéndolo forzado. Compilamos e instalamos la suite con:

```
# make
# sudo make install
```

Ahora intentaremos arrancar OWFS en el sistema. Para probar si funciona correctamente, creamos una carpeta vacía donde montar el Filesystem. En este caso, está en */tmp/* y se llama *1wire*. Después introduciremos el comando:

```
# /opt/owfs/bin/owfs -d /dev/ttyUSB0 -m /tmp/1wire
```

Con la opción *–d*, le indicamos al programa que usaremos el DS2480B conectado al puerto série */dev/ttyUSB0*. Con –m, le forzamos a que monte el sistema de ficheros en la carpeta. Si entramos en */tmp/1wire/* y hacemos un *ls,* veremos:

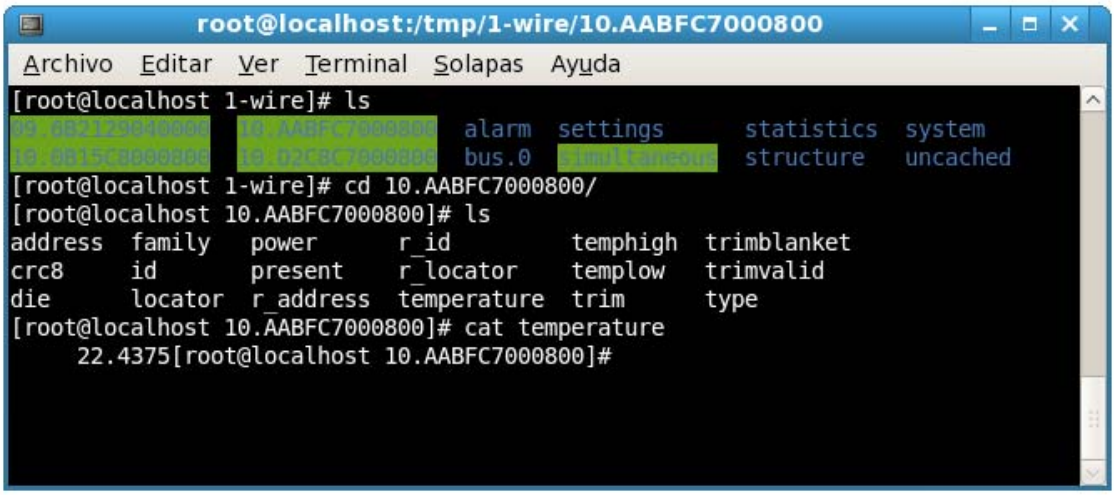


Figura 3.4 OWFS en funcionamiento

Donde, las carpetas que empiezan por 10.X, son sensores de temperatura. La carpeta 09.X Corresponde al ID 1-Wire device (dispositivo 1-Wire que únicamente contiene un número de série) interno que contiene el adaptador DS9097U. El resto de carpetas corresponden a parámetros de configuración y diagnóstico del bus. Si entramos en el sensor de temperatura 10.AABFC70008000, listamos el contenido, podremos ver varios parámetros, entre los que destacan la propia temperatura (temperature), y las alarmas de temperatura excesivamente alta o baja (*temphight*

y *templow*). Para ver el valor de la temperatura podemos usar el comando *cat,* como puede observarse en la captura.

OWFS evita leer continuamente del bus si no es necesario. Dispone de una caché de la que saca los datos si le pedimos lecturas con una frecuencia alta. Si queremos evitar su uso, deberemos realizar las mismas operaciones, pero desde el directorio *uncached/*. Éste contiene una réplica del Filesystem, pero realiza las lecturas siempre directamente del bus.

OWFS, por defecto, monta el sistema de ficheros con permisos de root. Si queremos que cualquiera pueda usarlo (el usuario netmrg por ejemplo) necesitaremos especificarlo al realizar el montaje.

Como queremos que OWFS se monte de forma automática cada vez que iniciemos el sistema, crearemos un script que ejecute la orden, y haremos que se cargue en el arranque, en los niveles 3 y 5 de Linux. En él incluiremos también el arranque OWHttpd.

Para preparar el script, creamos un fichero de texto con el siguiente contenido:

```
/opt/owfs/bin/owserver -d /dev/ttyUSB0 -p 12345
/opt/owfs/bin/owfs -s 127.0.0.1:12345 -m /tmp/1wire --allow-other
/opt/owfs/bin/owhttpd -s 127.0.0.1:12345 -p 3003
```

Lo guardamos como *owfs.sh*. Lo primero que se cargará será OWServer, que se quedará esperando peticiones en el puerto 12345. Usamos OWServer para poder leer del bus de forma simultánea desde OWFS y OWHttpd. Ambos se conectarán al bus desde *localhost:12345.* La opción *–allow-other* dará permisos de lectura a todos los usuarios para la carpeta montada. OWHttpd se quedará esperando peticiones http desde el puerto 3003.

Creamos un link simbólico en las carpetas de arranque de nivel 3 y 5 con:

```
# ln -s ./owfs.sh /etc/rc5.d/S98owfs
# ln -s ./owfs.sh /etc/rc3.d/S98owfs
```

Esto hará que nuestro script se cargue hacia el final del arranque, pero no al final del todo. Estamos reservando el último puesto (S99) para NetMRG, que debe ir después de OWFS. Es importante que tampoco arranque demasiado pronto puesto que, en el periodo de pruebas, tuvimos problemas con los drivers del adaptador usb-serie que arrancan hasta casi al final.

Como tenemos OWFS montado y escuchando por */dev/ttyUSB0* directamente, reiniciamos la máquina virtual para probar la nueva configuración. Si ahora nos conectamos por http usando el navegador al puerto 3003 de la máquina virtual, podremos ver el interfaz de OWHttpd.

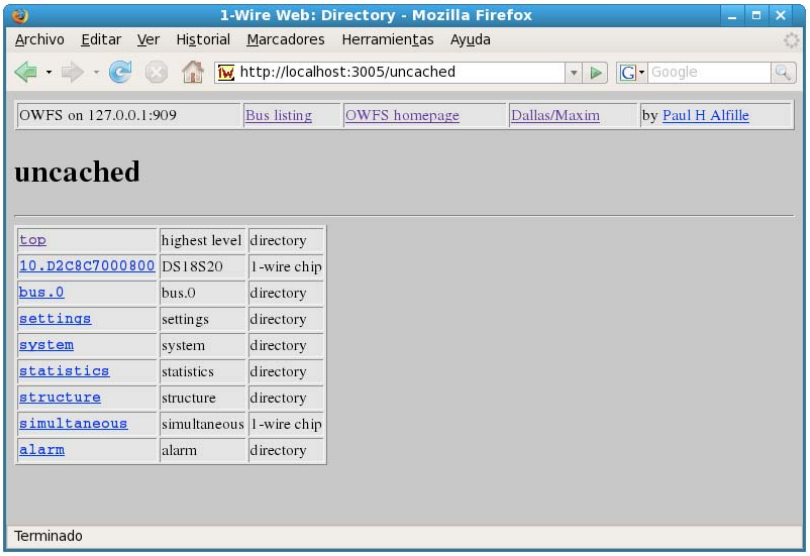


Figura 3.5. OWHttpd funcionando

Si entramos dentro del sensor de temperatura D2C8C7000800, podemos comprobar la potencia de OWHttpd.

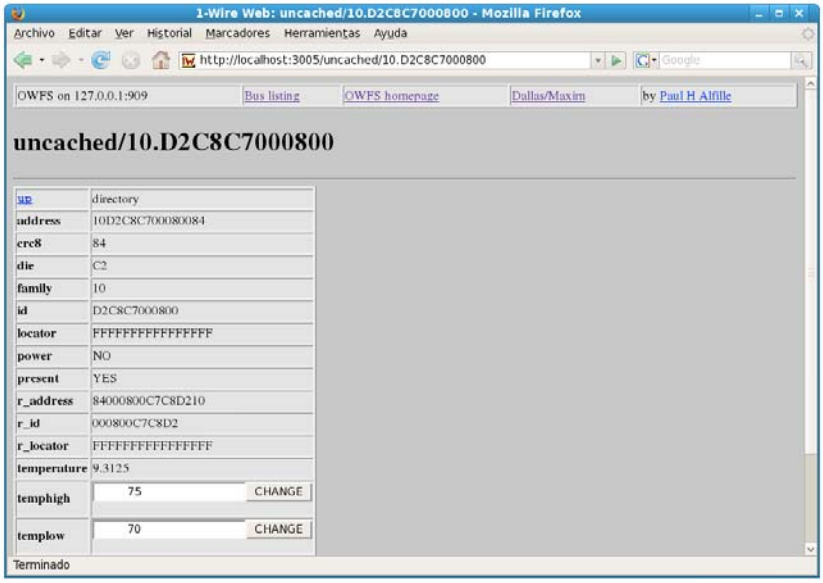


Figura 3.6. Parámetros del sensor de temperatura

3.4. GUMSTIX

Queremos hacer que OWServer corra en un ordenador aparte del resto de la suite OWFS. Si fuese posible hacer que OWHttpd también estuviese fuera de éste servidor podríamos, en caso que NetMRG funcionase incorrectamente, determinar el estado del bus sin depender de la máquina virtual. Si el ordenador dónde montamos OWServer es de tamaño normal, la idea pierde bastante la gracia. Si en cambio el ordenador puede ser fácilmente camuflado dentro de una caja de conexiones, enterrado en algún lugar del exterior o protegido de las condiciones externas, estableciendo la conexión a través de una red Wifi o Internet a través de GPRS, la separación gana mucho sentido.

Sabemos que el ordenador tiene que ser pequeño. Según sus creadores, Gumstix es el ordenador más pequeño del mundo. Ocupa el mismo volumen que un chicle de tira, de ahí su nombre. Por si fuera poco, dispone de infinitas opciones de ampliación. Conexiones Ethernet, Bluetooth, Wireless, GPRS, múltiples puertos USB y série, lectores de tarjetas MicroSD y Compact Flash… Y lo más importante… todo éste hardware es gobernado por un Linux de kernel 2.6. Por si fuese poco, en las especificaciones se indica que consume menos de 100mA por lo que podría incluso ser alimentarlo con una pequeña placa solar y una batería.
Parecía perfecto para nuestra aplicación. Nos convenció. Compramos los módulos que creímos más oportunos: Un Gumstix-verdex con un procesador ARM a 400Mhz con Bluetooth integrado, una placa de ampliación con 3 puertos série y USB y otra con Ethernet 10/100, 802.11G y lector de tarjetas Micro SD, así como 2 antenas externas para las conexiones inalámbricas. En ésta foto podremos apreciar el tamaño de los componentes.

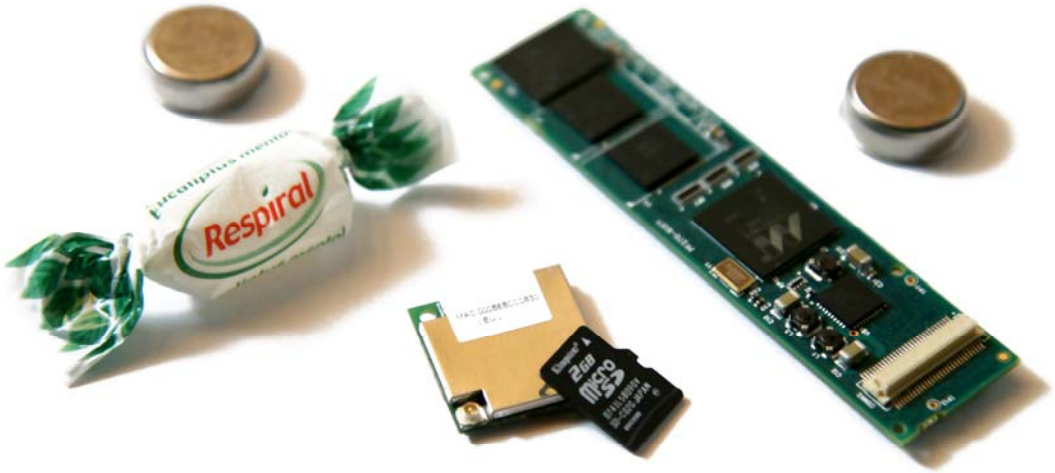


Figura 3.7. Gumstix

Parece un sueño, pero la realidad es bien distinta. La información referente al dispositivo, es casi nula: una mailing-list penosa en tamaño y fatal estructurada. Por si no fuese suficiente, es el peor hardware que hemos comprado nunca. Más adelante veremos el porqué. Podemos anticipar que tuvimos que comprar 2, ya que el primero vino completamente roto de la fábrica. Respecto al segundo… Aquí empieza su historia.

3.4.1 Instalación GUMSTIX

Para poder configurar el gumstix necesitaremos poder acceder a él a través de un terminal serie. Nosotros lo hicimos usando HyperTerminal en un Windows XP, a través de un cable especial que transforma la entrada en un minipuerto circular con los mismos pines.

Curiosamente, el dispositivo se suministra flasheado de fábrica con una versión de Linux que no tiene por que funcionar bien, de hecho en nuestro caso, la máquina ni siquiera arrancaba. Necesitaremos entonces sustituirla por una a medida, compilada para nuestro dispositivo y especificando los accesorios de los que disponemos. Ésta operación únicamente se puede realizar desde Línux. Nosotros lo hicimos desde un ordenador de sobremesa potente con Ubuntu instalado de forma nativa. La razón de no usar la máquina virtual sobre el portátil, no es otra que ganar tiempo teniendo en cuenta que necesitamos compilar un kernel.

Lo primero que deberemos hacer será resolver algunas dependencias, de manera que Ubuntu nos deje proseguir. Actualizaremos a través de *apt-get*, introduciendo los comandos:

```
# sudo apt-get update
# sudo apt-get install build-essential
  libncurses5-dev bison flex texinfo zlib1g-dev
  gettext libssl-dev libc6-dev make g++ gcc
  automake1.9 subversion
```

Nos preguntará si queremos realizar la descarga e instalación, contestamos que sí, y esperamos a que termine de forma correcta.

Ahora descargaremos la última versión del gumstix-buildroot (bootloader + kernel + software) en nuestra máquina. Para ello, abrimos un nuevo terminal, creamos una carpeta vacía y descargamos dentro la versión más reciente a través de la herramienta subversión, de la siguiente manera:

```
# svn co http://svn.gumstix.com/gumstix-buildroot/
  trunk gumstix-buildroot
```

Una vez descargadas las fuentes, entramos en el directorio que nos ha creado subversión, borramos la configuración que trae inicialmente el paquete y lo configuramos a medida para nuestro dispositivo.

```
# scd gumstix-buildroot
# rm .config
# make defconfig
```

Nos desplazamos por los menús de configuración uno por uno, adaptando ésta a nuestro gumstix-verdex. De entre los valores que configuramos cabe destacar la velocidad de la CPU (400MHZ), el tipo de arquitectura de nuestra máquina (imwxt) y el soporte para red cableada y Wireless.

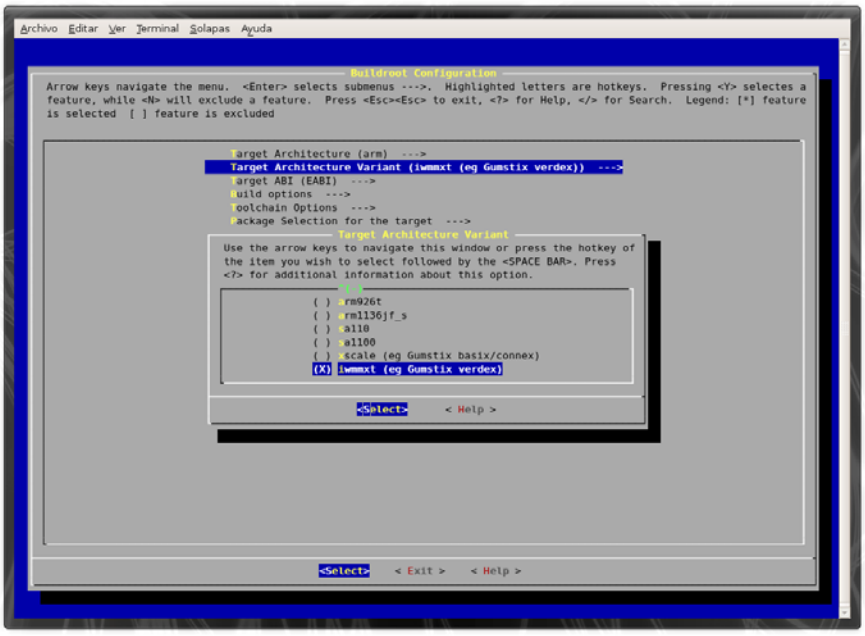


Figura 3.8. Configuración Gumstix

Guardamos la configuración, esperamos a que termine el proceso y empezamos la compilación del kernel.

```
# make
```

Nuestra máquina de sobremesa Core2duo a 2,6GHZ tardó unos 45 minutos en acabar. Una vez finalizada la operación, deberemos coger del path donde hemos compilado los ficheros:

- rootfs.arm_nofpu.jffs2
- u-boot.bin
- uImage

Y trasladarlos hasta la máquina donde haremos la conexión por consola. Como ya hemos dicho, nosotros lo haremos desde un portátil con HyperTerminal, aunque también puede hacerse desde Línux usando un emulador de terminal série.

Vamos a flashear todo el gumstix, Bootloader y kernel. En este punto debemos extremar todas las precauciones. Si se va la alimentación mientras estamos flasheando el sector de arranque, tendremos un pisapapeles nuevo.

En un principio barajamos la opción de no grabar la parte peligrosa, sólo el kernel, el problema, según leímos en la página de gumstix, es que no todas las versiones del bootloader funcionan bien con todos los kernel, por lo que preferimos empezar de cero y curarnos en salud.

Abrimos HyperTerminal. Lo configuramos según los parámetros siguientes:

Velocidad: 115200 bps
Bits de datos: 8
Paridad: Ninguna
Bits de parada: 1

Montamos la tarjeta de de consola. Conectamos la alimentación y el puerto serie central, que es el que usa por defecto como salida estándar. Empiezan a salir caracteres por pantalla y muy rápidamente pulsamos espacio, de manera que entramos en el menú del cargador de arranque.

Podemos ver el mensaje

```
GUM>
```

Empezaremos por sustituir el sector de arranque. Para enviar un fichero hacia el Gumstix escribimos:

```
GUM> loady
```

Y pulsamos enter. El microordenador se queda esperando a que le enviemos un fichero. Nos vamos al menú *Transferir* y pulsamos sobre *Enviar archivo*. Seleccionamos el archivo u-boot.bin y seleccionamos el protocolo Y-modem. Pulsamos sobre el botón *Enviar*.

La transferencia se completa en menos de un minuto. Pasamos ahora a desproteger el sector de arranque y eliminar la información que contiene. Ésta es la parte más peligrosa de todo el proceso. Introducimos:

```
GUM> protect off 1:0-1
GUM> era 1:0-1
```

Y grabamos encima del sector borrado el archivo que hemos enviado antes.

```
GUM> cp.b a2000000 0 ${filesize}
```

Tarda también menos de un minuto. Ya podemos volver a dejar el sector de arranque protegido.

```
GUM> pro on 1:0-1
```

¡Salvados! En la captura puede verse como transcurre el tenso proceso.

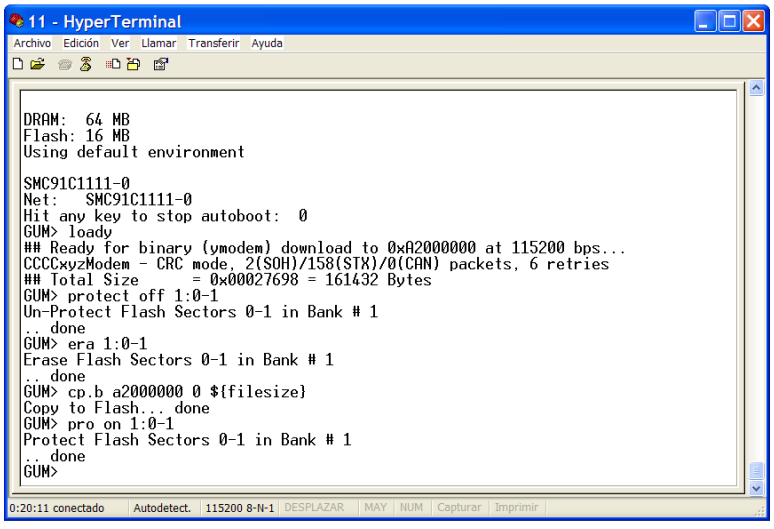


Figura 3.9. Flasheo sector de arranque

Reiniciamos la placa para flashear el resto de ficheros desde el nuevo bootloader.

```
GUM> res
```

Pulsamos de nuevo la tecla espacio al arrancar. Para enviar el FileSystem de nuevo tecleamos

```
GUM> loady
```

Y desde el menú enviamos el archivo *rootfs.arm_nofpu.jffs2* usando el protocolo Ymodem. El proceso parece interminable y el gumstix parece una cafetera, está tan caliente que podría encender una cerilla con solo acercarla. Decidimos montarle un disipador improvisado con un trozo de aluminio de una ventana.

Figura 3.10. Disipador improvisado

Tras más de 20 minutos, por fin acaba. Borramos todo excepto los sectores protegidos y grabamos el fichero enviado al espacio recién liberado.

```
GUM> jera all && cp.b a2000000 40000 ${filesize}
```

Tarda unos 10 minutos. Por lo menos mientras va grabando, van saliendo unos puntitos por consola, para que el usuario vea que la operación no se ha detenido.

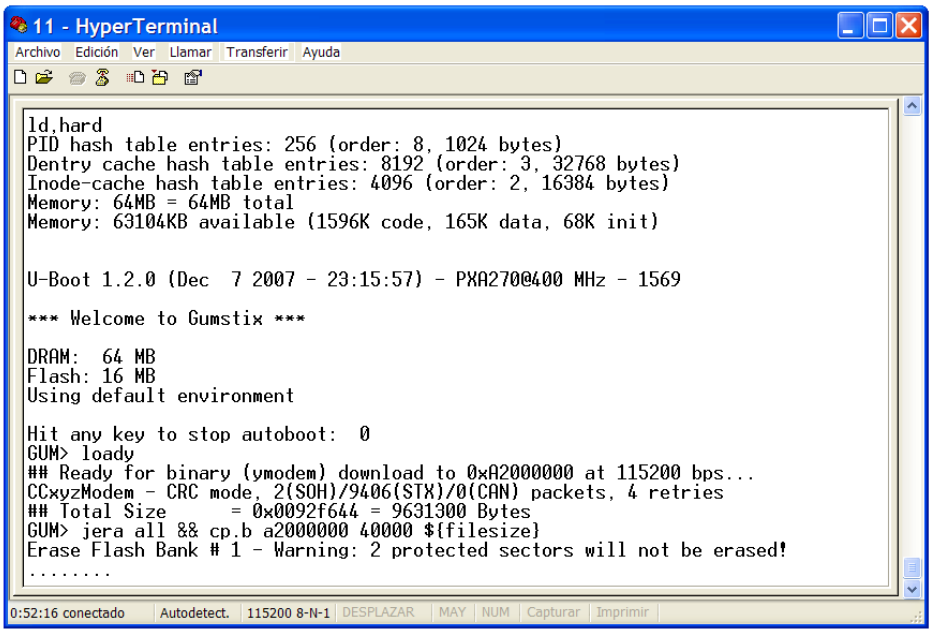


Figura 3.11. Copia del Filesystem

Cuando termina, pasamos a subir el tercer fichero, el kernel. Para ello, usamos el mismo comando y protocolo que las 2 veces anteriores, pero enviando el archivo *uImage*. En menos de un minuto el archivo está en el gumstix. Lo instalamos y cargamos en el gumstix con:

```
GUM> katinstall 100000
GUM> katload 100000
```

Y arrancamos desde la nueva imagen introduciendo:

```
GUM> bootm
```

Desconectamos la alimentación e introducimos la tarjeta de red. A los 5 segundos de arrancar se nos ocurre tocar el modulo Wifi y casi nos sale humo del dedo. Algo raro pasa con ésta placa. Ya no podemos refrigerar el procesador ni el módulo

Bluetooth, que se calientan muchísimo porque tienen la placa con la tarjeta de red encima. Arrancamos sin la Wireless. Por fin vemos el menú de inicio.

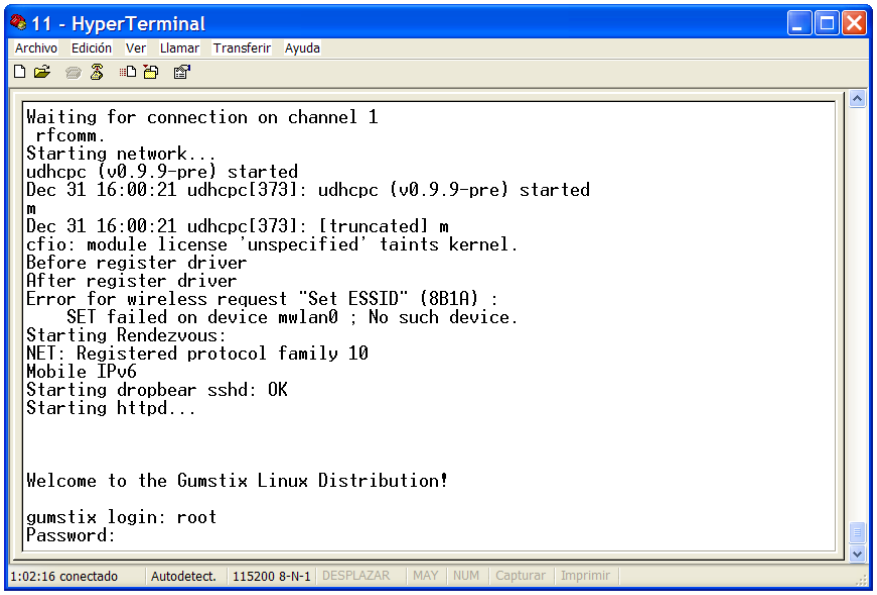


Figura 3.12. Primer arranque

El password de *root* por defecto es *gumstix*. Entramos correctamente, navegamos por algunas carpetas y vemos que aparentemente el gumstix funciona bien. Eso si, sigue muy caliente.

Si hacemos ifconfig vemos que dhcp nos ha dado una dirección. ¡Ya estamos en la red!

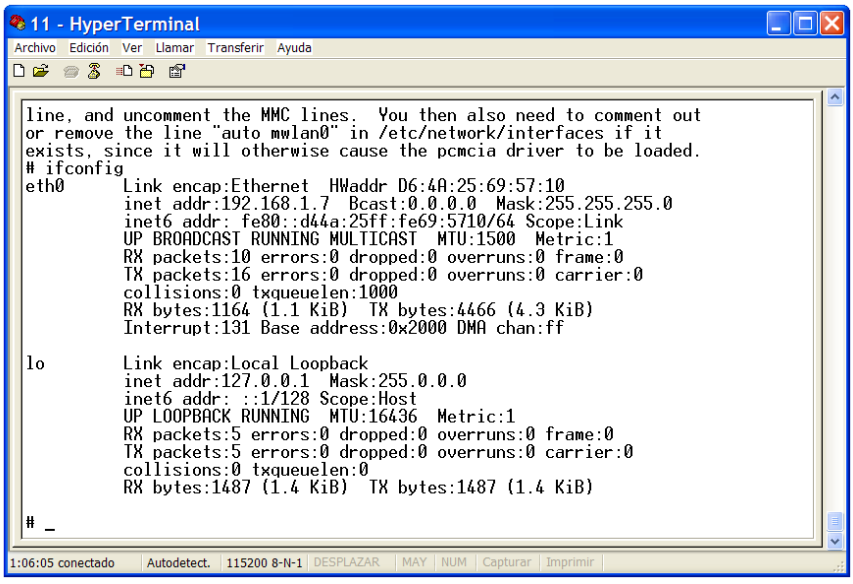


Figura 3.13. Ifconfig Gumstix

Nos conectamos por http desde otro ordenador y…

Figura 3.14. Servidor Web funcionando

¡Perfecto! ¡Servidor web funcionando! El siguiente paso es compilar OWServer y OWHttpd para que puedan correr sobre el pequeño procesador Arm del gumstix. Para hacerlo, deberemos descargar OWFS en una máquina donde hayamos compilado el gumstix-buildroot anteriormente y configurar la instalación de la siguiente manera:

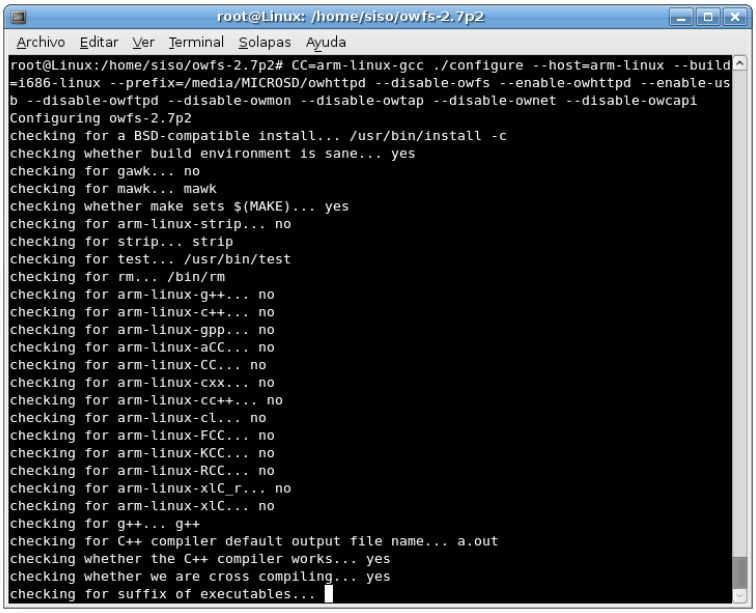


Figura 3.15. Configuracion OWFS Gumstix

De esta forma, le indicamos varias cosas al compilador:

- Que estamos haciendo una compilación cruzada para otra plataforma distinta a la del propio PC (ARM)
- Que queremos que, al realizar la instalación, los ficheros no se instalen en nuestra máquina sino que vayan a parar directamente a la tarjeta microSD que hay insertada en el lector y montada.
- Que queremos desactivar todas las opciones excepto OWServer y OWHttpd.

Al acabar la configuración, deberíamos ver algo parecido a la captura.

Figura 3.16. Resumen configuración OWFS

Ahora tenemos que compilar e instalar el software en la tarjeta.

```
# make
# sudo make install
```

Introducimos la contraseña de root. Cuando la instalación termine, cogemos la tarjetita y la insertamos en el lector del módulo de Red del Gumstix. Conectamos la alimentación y esperamos a que cargue el SO.

Hacemos el LogIn y activamos el puerto serie de la izquierda, redireccionándolo hacia */dev/ttyS2*.

```
# modprobe proc_gpio
# echo “AF2 in” > /proc/gpio/GPIO46
# echo “AF1 out” > /proc/gpio/GPIO47
```

Copiamos el contenido recién instalado en la tarjeta en los directorios */bin* y */lib* del gumstix. Probamos que el ejecutable arranque.

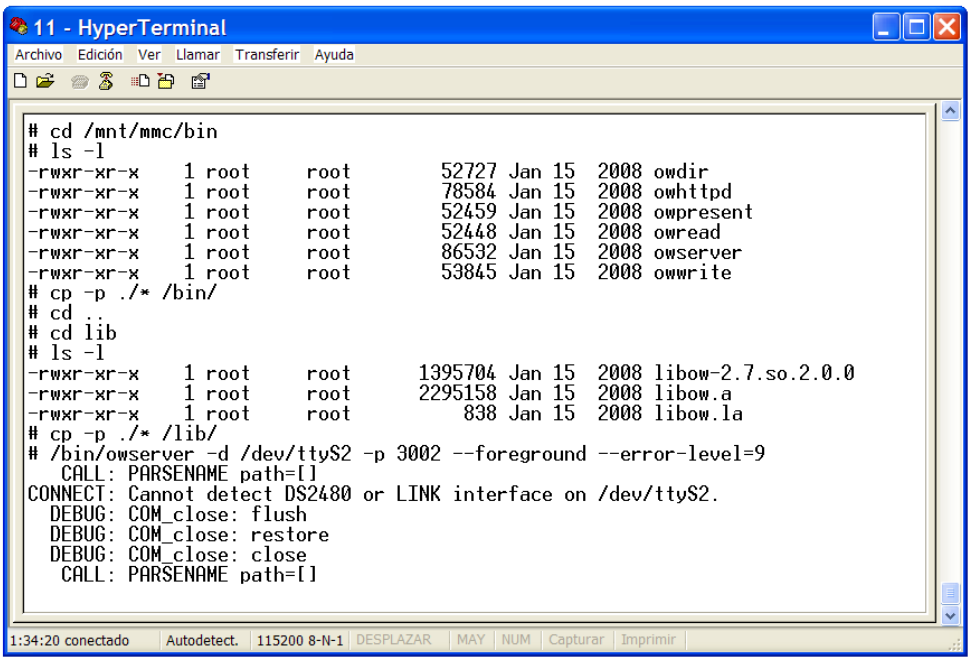


Figura 3.17. Prueba arranque OWFS

Vamos a la tienda a comprar un adaptador hembra-hembra para puerto serie. Hasta ese momento habíamos intentado enchufarlo con 9 cables pelados por los dos extremos, pero no eran lo suficientemente gordos y bailaban. La verdad es que no parecía demasiado fiable.

A la vuelta teníamos una sorpresa. Creemos que su propio calor no le sentó muy bien.

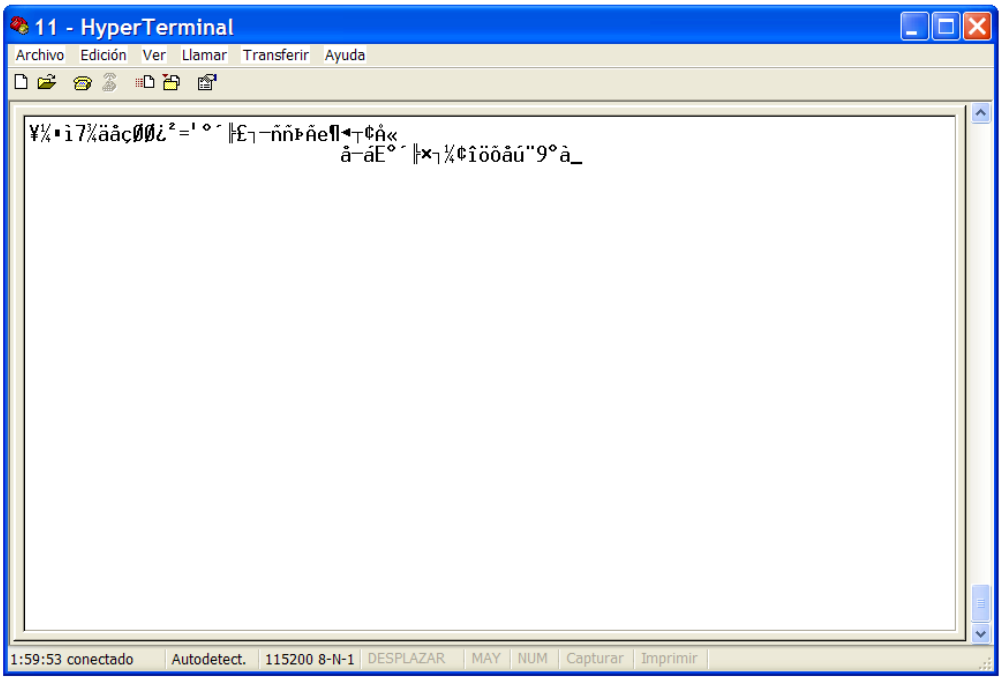


Figura 3.18. Error Gumstix

Conectamos y desconectamos varias veces la alimentación, reiniciamos el ordenador y lo probamos en otro PC, pero nunca volvió a funcionar. El primero lo enviaron roto y el segundo duró una hora y media. Hacía mucho tiempo que no comprábamos algo tan malo.

Después de tal decepción nos hemos decidido saltar este paso. La máquina virtual se conectará directamente al adaptador USB-serie.

Tenemos OWFS y OWHttp corriendo sobre nuestro servidor. Sólo falta que empiece a dibujar gráficas. NetMRG nos parece un software excelente, por lo que lo volveremos a usar. Es lo único que conservamos de la primera versión de LakeMonitor.

Antes de nada vamos a transformar OWFS en un formato que pueda entender NetMRG. Queremos un programa que se comporte exactamente igual que *leertemp*, pero que obtenga los valores a través del sistema de ficheros.

Para reproducir este comportamiento crearemos un script, que contenga el siguiente texto:

```
cat /tmp/1wire/10.$1/temperature
```

De esta forma, al ejecutar el programa, $1 se cambiará por el parámetro que especifiquemos, es decir, la dirección física del sensor.

Si queremos una lista con los sensores conectados, sólo tenemos que entrar en el menú principal de OWHttpd, o listar los archivos de la carpeta */tmp/1wire/.*

Llegados a este punto, realizaremos la instalación exactamente igual que en la primera parte, cambiando el programa *leertemp* por nuestro nuevo script.

De todas formas, esta vez queremos llegar aún un poco más lejos. Queremos que la cara de nuestro LakeMonitorV2, es decir, NetMRG se muestre más a la medida de nuestra aplicación.

Para cambiar el título principal del programa (las letras grandes que aparecen a la izquierda) editaremos el fichero */usr/local/var/www/netmrg/include/config.php* y cambiaremos el texto 'NetMRG' por 'LakeMonitor'. Modificaremos también */usr/local/etc/netmrg.xml* para que el texto 'Generic Company' de la izquierda pase a ser 'TFC EPSC Telecos'.

Hemos cambiado también varios ficheros contenidos en la carpeta */usr/local/var/www/netmrg/webfiles/* para traducir al castellano algunas palabras del menú principal.

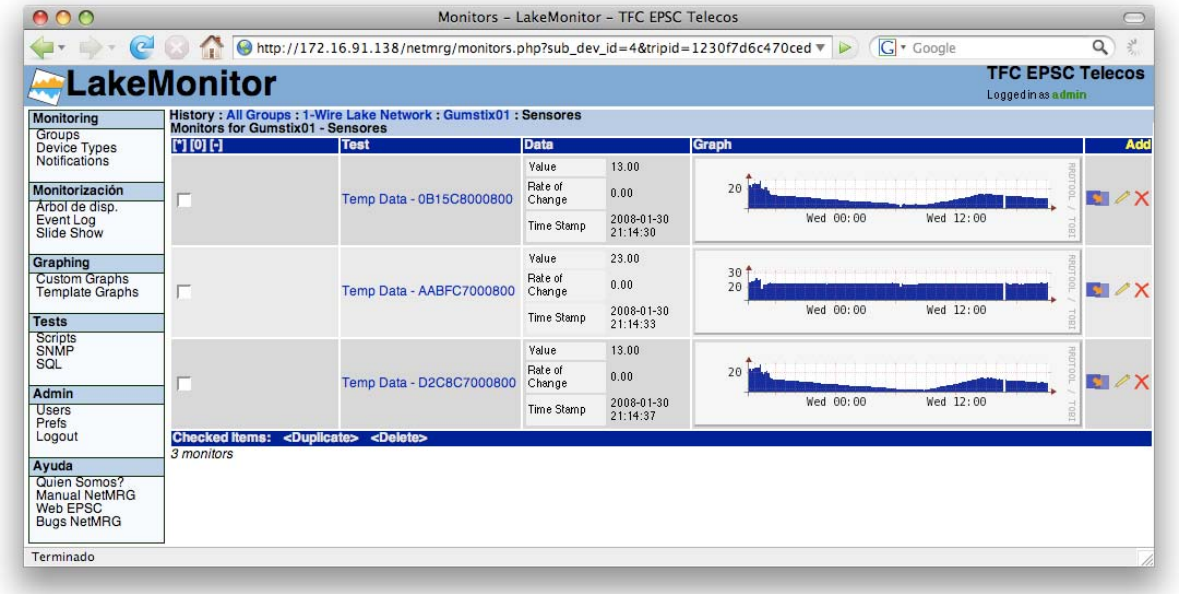
3.5 Manual LakeMonitorV2

Ya tenemos la maquina virtual completamente preparada. El modo gráfico de Linux ya no es necesario. Para pasar el sistema a nivel de ejecución 3, abrimos un terminal de consola nuevo. Nos loggeamos como administrador e introducimos la secuencia

```
# init 3
```

Linux apagará los servicios innecesarios y optimizará el uso de los recursos en la máquina. Si necesitamos de nuevo el modo gráfico, ejecutaremos la misma órden cambiando el 3 por un 5.

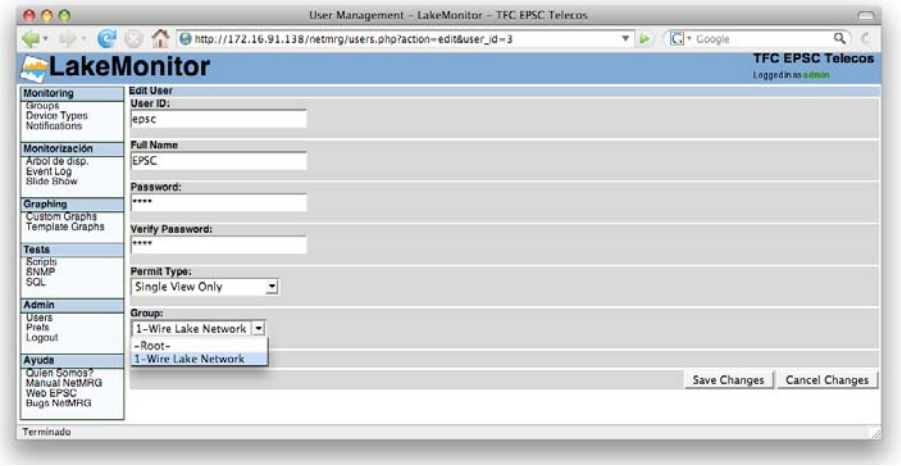
Para entrar en LakeMonitor desde el exterior, deberemos conocer la IP de la máquna virtual. Desde cualquier navegador HTML, nos dirigiremos a http://<@IP>/netmrg. Introduciremos el nombre de usuario y password por defecto (admin, nimda).



Una vez dentro, lo primero que haremos será cambiar éstos parámetros por defecto. Haremos clic en la pestaña users, en la sección Admin del menú de la izquierda y pulsaremos sobre el botón con el gráfico de un lápiz. En su interior, en la sección password, introduciremos la nueva clave y pincharemos en Save Changes.

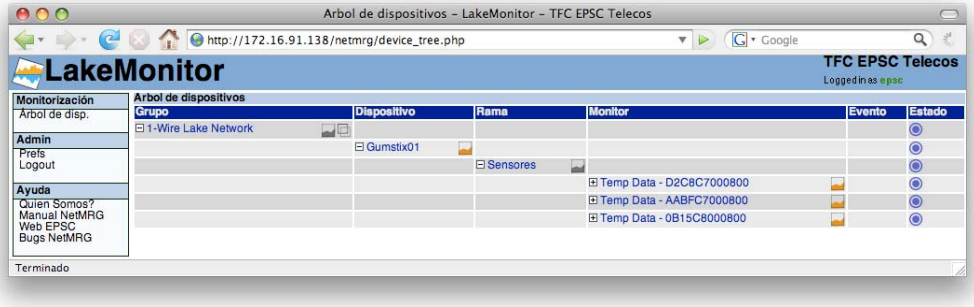
Una vez establecida la contraseña de root, pasaremos a crear un nuevo usuario, sin permisos de administración. Hacemos clic en *Add* y rellenamos los campos de una forma similar a la de la siguiente captura.

Si tenemos varias zonas diferenciadas, podemos asignar permisos al usuario para que solo pueda visualizar los sensores de ese sector. Lo haremos desde la opción Group.

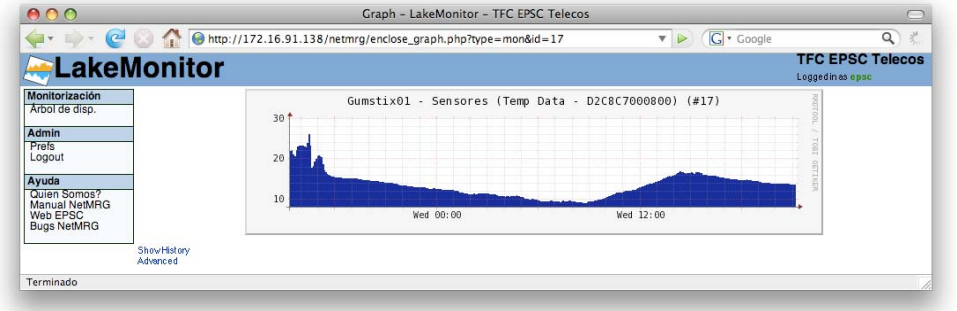


Si pinchamos en *Logout* y nos conectamos con el nuevo usuario, podremos comprobar cómo las opciones disponibles en el menú se han reducido considerablemente.

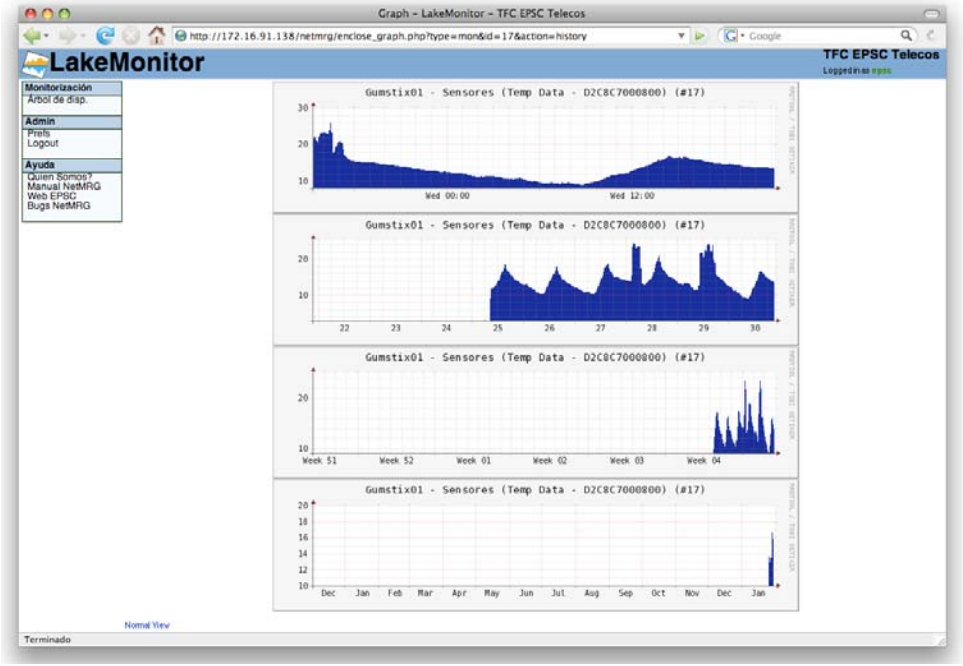
Únicamente podremos realizar tareas de monitorización. Para ver el estado de un sensor, sólo hay que pinchar sobre el icono naranja, al lado del nombre del dispositivo.



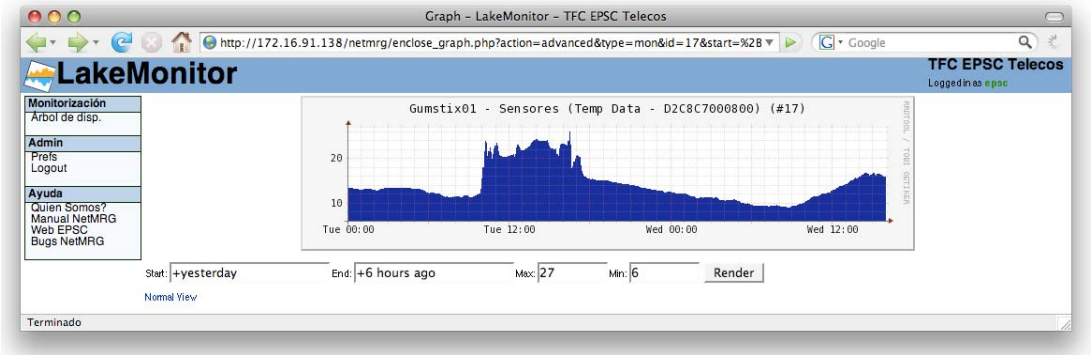
Una vez dentro, veremos la gráfica de las últimas 24 horas.



Si pinchamos sobre el botón *Show history,* el software nos mostrará un resumen completo de la actividad del sensor. Desde un día hasta un año.



Si queremos un gráfico a medida, haremos clic sobre el botón *Advanced*.



Desde el interfaz que nos proporciona NetMRG en éste punto, podremos controlar cualquier parámetro de la representación. Si las temperaturas máxima y mínima que utiliza por defecto no nos convence, sólo tenemos que adaptarlas a nuestro gusto. Para elegir el punto de inicio y fin de la grafica, utilizaremos la siguiente nomenclatura.

+ X minutes ago
+ X hours ago
+ yesterday
+ X days ago
+ X weeks ago
+ X years ago

4. Problemas

A continuación enumeramos un resumen de los principales problemas que hemos sufrido a lo largo de este trabajo:

- A la hora de obtener los materiales tuvimos problemas con la aduana y los cobros de aranceles en el gumstix; y destacar que a día de hoy todavía tenemos material retenido en DHL, concretamente la TINI (Tiny InterNet Interface), un dispositivo embebido con capacidad de comunicación Ethernet creado por Dallas.

- Después de pagar una cantidad excesiva en la aduana, conseguimos tener el gumstix en nuestro poder, pero obtuvimos una gran sorpresa al observar que venía defectuoso. Nada mas conectarlo, se empezaba a calentar y no llegaba a encender.

- Volvimos a comprar otro, éste por lo menos se encendió, pero como habéis podido leer, no tardo mucho en calentarse y dejar de funcionar. Además de todo eso, de los 3 puertos serie que disponía, solo funcionaban 2 y de aquella manera.

- Dificultad para encontrar información para compilar programas en el gumstix.

- Dificultad para desarrollar el 1-Wire PDK para Linux, casi toda la información era para Windows

- El problema del bug del NetMRG nos tuvo mucho tiempo bloqueados.

- El adaptador usb de 1-Wire era totalmente incompatible para Linux

- Dificultad para encontrar un adaptador de serie a usb que funcionase correctamente con Linux (a la cuarta fue la vencida).

- Grandes incompatibilidades de software

- ….

5. Conclusiones

Con largos días y sudorosas noches de trabajo, con la ayuda que nos han proporcionado Google y nuestro tutor, nos llenamos de orgullo al afirmar que finalmente, lo hemos conseguido.

Hemos creado un sistema de monitorización remota de temperatura, fácilmente adaptable a infinidad de parámetros adicionales (humedad, ph, velocidad del viento, cantidad de lluvia por metro cuadrado...) cuyo servidor puede correr sobre cualquier ordenador equipado con usb, independientemente del sistema operativo que use (Linux, Windows, MacOS), que nos permite estudiar la evolución de los valores sobre cualquier dispositivo equipado con un navegador html (ordenadores, pda's, móviles...) con un número ilimitado de usuarios simultáneos y sin gastar ni un solo euro en software. Por supuesto, con todo el código disponible para que, llegado el momento podamos adaptar el sistema a cualquier necesidad futura.

6. Bibliografía

Webs

1. http://www.maxim-ic.com/products/1-wire/
2. http://www.1wire.org/
3. http://www.roso-control.com/
4. http://www.aagelectronica.com/aag/index.html
5. http://www.owfs.org/
6. http://owfs.sourceforge.net/WRT54G.html
7. http://gumstix.com/
8. http://www.gumstix.org/
9. http://docwiki.gumstix.org/Main_Page
10. http://www.vmware.com/
11. http://www.thoughtpolice.co.uk/

Datasheets

1. http://datasheets.maxim-ic.com/en/ds/DS1920.pdf
2. http://datasheets.maxim-ic.com/en/ds/DS2480B.pdf
3. http://www.maxim-ic.com.cn/pdfserv/en/ds/DS9097U-009-DS9097U-S09.pdf

Guías

1. http://www.datsi.fi.upm.es/docencia/Micro_C/dallas/tb1.pdf
2. http://pdfserv.maxim-ic.com/en/an/AN192.pdf
3. http://pdfserv.maxim-ic.com/en/an/AN192.pdf
4. http://pdfserv.maxim-ic.com/en/an/AN1100.pdf
5. http://pdfserv.maxim-ic.com/en/an/AN148.pdf
6. http://pdfserv.maxim-ic.com/en/an/AN155.pdf

ANEXOS

TÍTULO DEL TFC: Monitorización remota de una red 1-Wire

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

AUTORES: Narciso Cuartero Moya
Sergio Quintana Alcaraz

DIRECTOR: José Polo Cantero

FECHA: 4 de Febrero de 2008

ANEXO 1

```
Makefile
CC = gcc
AR = ar

# directories
PRE    = ../
APPS   = $(PRE)/apps
COMMON = $(PRE)/common
LINK   = $(PRE)/lib/userial/link/Linux
SHARED = $(PRE)/lib/userial/shared

CFLAGS = -I $(COMMON) -I $(SHARED) -I $(LINK) -DDEBUG -c
LFLAGS = -I $(COMMON) -DDEBUG -g -o $@

ONEWIREOBJS = obj/owlinkobjs
WEATHEROBJS = obj/temp10.o $(ONEWIREOBJS)

PROGS = listar leertemp

all: $(PROGS)

listar: $(APPS)/temp/listar.c obj/temp10.o obj/findtype.o $(ONEWIREOBJS)
          $(CC) $(LFLAGS) $< obj/temp10.o obj/findtype.o $(ONEWIREOBJS)

leertemp: $(APPS)/temp/leertemp.c obj/temp10.o obj/findtype.o $(ONEWIREOBJS)
          $(CC) $(LFLAGS) $< obj/temp10.o obj/findtype.o $(ONEWIREOBJS)

obj/touch:
          mkdir -p obj
          touch obj/touch

obj/%.o: $(COMMON)/%.c obj/touch
          $(CC) $(CFLAGS) -o obj/$*.o $<

obj/owlinkobjs: $(LINK)/*.c $(SHARED)/*.c obj/owerr.o obj/ioutil.o obj/crcutil.o obj/touch
          $(CC) $(CFLAGS) $(LINK)/*.c $(SHARED)/*.c
          $(AR) r obj/owlinkobjs *.o obj/owerr.o obj/ioutil.o obj/crcutil.o
          rm -f *.o

clean:
          rm -rf obj
          rm -f *.o
          rm -f ${PROGS}
```

ANEXO 2

DALLAS SEMICONDUCTOR

Tech Brief 1
MicroLAN Design Guide

What is the 1-Wire net?

The 1-Wire® net, sometimes known as a MicroLAN, is a low cost bus based on a PC or microcontroller communicating digitally over twisted pair cable with 1-Wire® components. The network is defined with an open drain (wired-AND) master/slave multidrop architecture that uses a resistor pull-up to a nominal 5V supply at the master. A 1-Wire net based system consists of three main elements: a bus master with controlling software, the wiring and associated connectors and 1-Wire devices. The 1-Wire net allows tight control because no node is allowed to speak unless requested by the master, and no communication is allowed between slaves, except through the master.

Any standard microcontroller such as an 8051 with a 1.8MHz or greater clock, as well as a PC using a 115.2 kbps capable UART can serve as master for the net. The UART supplies 1-Wire timing by sending a byte for each 1-Wire bit, creating short and long time slots to encode the binary 1's and 0's. At this 14.4 kbps data rate (115.2 divided by 8 =14.4 kbps) the PC can address a node on the bus and start receiving data in less than 7 milliseconds. Since timing is controlled by the UART, microprocessor clock speed does not affect the time required to find and read a slave ID on the net. **Diagram 1** illustrates a portion of a typical communication sequence for 1-Wire protocol. Software such as TMEX™ to control and monitor bus activity is available for downloading at http://www.ibutton.com/software/tmex/.

1-Wire net protocol uses conventional CMOS/TTL logic levels, where 0.8V or less indicates a logic zero and 2.2V or greater represents a logic one. Operation is specified over a supply voltage range of 2.8 to 6 volts. Both the master and slaves are configured as transceivers allowing data to flow in either direction, but only one direction at a time. Technically speaking, data transfers are half-duplex and bit sequential over a single pair of wires, data and return, from which the slaves "steal" power by use of an internal diode and capacitor. Data is read and written least significant bit first. An economical, DS9097 COM Port Adapter is available to interface RS232 to the net. Newer more versatile adapters based on the DS2480 Serial 1-Wire Line Driver chip provide more capability such as active pull-up and slew-rate control. The DS2480 is designed to interface between RS232 and the 1-Wire bus generating the proper signals and programmable waveforms that provide maximum performance. Regardless of whether a DS9097 or a DS2480 based adapter is used, readily available, low capacitance, unshielded, Category 5 twisted pair phone wire is recommended for the bus.

As previously mentioned, data on the 1-Wire net is transferred with respect to time slots. For example, to write a logic one to a 1-Wire device, the master pulls the bus low and holds it for 15 microseconds or less. To write a logic zero, the master pulls the bus low and holds it for at least 60 microseconds to pro-

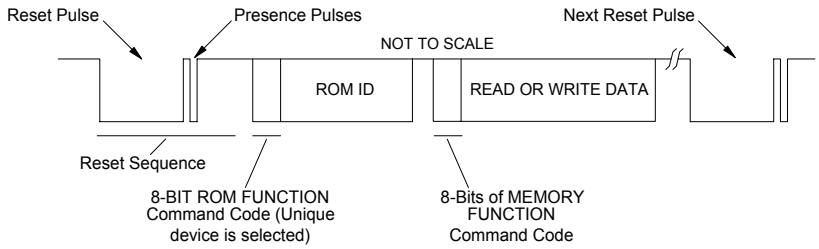


Diagram 1 A typical 1-Wire communication sequence.

vide timing margin for worse case conditions. A system clock is not required, as each 1-Wire part is self clocked by its own internal oscillator that is synchronized to the falling edge of the master. Power for chip operation is derived from the bus during idle communication periods when the DATA line is at 5V by including a half wave rectifier onboard each slave.

In **Figure 1**, whenever the data line is pulled high by the bus pull-up resistor the diode in the half wave rectifier turns on and charges the internal 800pF capacitor. When it drops below the voltage on the capacitor, the diode is reverse biased, isolating the charge. The isolated charge stored on the capacitor provides the energy source to power the slave during the intervals the bus is pulled low. The amount of charge lost during these periods is proportional to the time the bus is low. It is replenished when the data line again turns on the half wave rectifier diode. This concept of "stealing" power from the data line by a half wave rectifier is referred to as "parasite power."

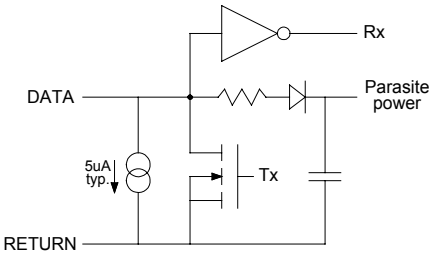


Figure 1 1-Wire parasite power circuit.

In operation, the master resets the network by holding the bus low for at least 480 microseconds, (**Diagram 1** again) releasing it, and then looking for a responding Presence pulse from a slave connected to the line. If a Presence pulse is detected, it then accesses the slave by calling its address. The master issues any device specific commands required, and performs any needed data transfers between it and the slave. It controls the information transfer by generating time slots and examining the response from the slave. For a more detailed description of the communication sequence, refer to "So that's how it works!" in Chapter 2. Complete technical and timing details are available in the Book of DS19xx iButton® Standards.

Within each 1-Wire slave created is stored a lasered ROM section with its own guaranteed unique, 64-bit serial number that acts as its node address. This unique address is composed of eight bytes divided into three main sections. Starting with the LSB, the first byte stores the 8-bit family code that identifies the device type. The next 6 bytes store a customizable 48-bit individual address while the last byte (MSB) contains a cyclic redundancy checksum (CRC) with a value based on the data contained in the first seven bytes. This allows the master to determine if an address was read without error. With 2^{48} serial numbers available, conflicting or duplicate node addresses on the LAN will never be a problem.

Floppy in a Button

1-Wire devices can be formatted with a file directory just like a floppy disk. This allows files to be randomly accessed and changed without disturbing other records. Information is read or written when a device connected to the bus is addressed by the master, or an identification badge or Decoder Ring is touched to a port somewhere along the 1-Wire net. A typical access port consists of outer ring and insulated spring loaded center conductors mounted in an appropriate housing. The ring touches the case of the iButton or Decoder Ring and connects it to the return line of the bus while the spring loaded center contact connects the lid to the bus data line.

The inclusion of up to 64K of memory in 1-Wire chips allows standard information such as employee name, ID number etc., to be stored within the device. For example, it would only take about one fourth of the available memory to store the equivalent of a business card and digitized black and white photo ID. This still leaves space for additional important data such as medical records, credit information or security level to be included. With such information literally "at hand" in the case of the Decoder Ring, reliable identification and access is readily available and machine readable. Memory devices are also useful for storing "tag" information for nodes on the bus and for sensor calibration or function information.

Packaged and ready to go

Because the 1-Wire net only requires a single wire plus return, iButtons can be packaged in a coin style battery case 16mm in diameter, and 5.8mm thick. This is about the size of a stack of five dimes. The two piece stainless steel package acts as both protective housing and electrical connection point. The case serves as return contact (ground), and the lid as data contact. The package size allows inclusion of a Lithium cell to provide 10 years of standby power to maintain data in volatile RAM memory when not connected to the net. A variety of memory configurations are available, including iButtons containing up to 64K of memory. 1-Wire devices that do not require a memory backup battery are also available in more traditional solder-mount packages.

SECTION 2
Working the LAN

Control the slew rate of the driver

In a typical system using a PC running Dallas' 1-Wire Operating System Software (TMEX) and using a COM port adapter, communication occurs in time slots of 8.68 microseconds under control of the UART. A communication cycle begins when the transistor in the master actively pulls the line to a logic zero. This one to zero transition is the synchronizing edge for all 1-Wire communications. A 1-wire slave holds the zero if appropriate, and the resistor returns the line to the supply voltage after both the master and slave release the line. In ROM search, which is required to identify the devices on the bus, the most critical part of 1-Wire communication is the read data time slot, especially if a one is being transmitted. In the general case, there may be many arbitrarily placed devices on the bus, each seeing the falling edge of the time slot issued by the master at a slightly different time. Because communication requires that a signal travel the length of the cable and return, the electrical length of the bus must be less than one half the time interval allowed for a single data bit slot. That is, round trip propagation time for a signal must be less than 4.34 microseconds (8.68 microseconds divided by 2). Devices beyond this range will not be seen by the master.

In the COM port adapter, there is an active pull down transistor that is either turned fully on or completely off under control of the bus master. The falling edge generated by turning it on signals the start of a time slot on the net. When the switch is turned off, the line is pulled toward the supply voltage by the bus pull-up resistor. Due to the rapid response and low impedance of the active pull down transistor used to generate a logic zero, the signal fall time will be in the sub-microsecond range. If switching occurs in less time than the transition takes to traverse the electrical length of the cable and return, the 1-Wire net is operating in a transmission line environment and reflections from the line end can disrupt communications.

Normally, the solution would be to terminate the ends of the cable in its characteristic impedance with fixed resistors. These resistors would then absorb the energy that otherwise would be reflected by the impedance mismatch and cause communications problems. Unfortunately, the recommended cabling has a 100 Ohm typical impedance which would result in the inability to generate a logic one with an acceptable pull-up resistor value. It is interesting to note that since the port transistor inside 1-Wire devices has a 100 Ohm on-resistance, the bus is properly terminated anytime one at the cable end is turned on. It may be possible on some 1-Wire Nets to ac terminate the bus using a series resistor and capacitor connected to ground. After the capacitor charges, it blocks DC current so the series resistor presents no load to the bus. During transitions however, the capacitor appears as a short circuit and the resistor terminates the line. A general rule of thumb for selecting the capacitor, is 3 times the rise time divided by cable impedance. For a 4 microsecond rise time on 100 Ohm cable, this works out to be 0.1µF. The disadvantage of ac termination is the bit dependent timing jitter it generates due to the charging and discharging of the terminating capacitor shifting the waveforms.

Since it is not possible to terminate the ends of the cable in its characteristic impedance, the alternative is to control the slew rate of the bus master pull down transistor. For bus lengths of 100 meters or more a 1.1 volt per microsecond slew rate is recommended. This provides a one to zero transition

that takes about 4 microseconds to ramp to the 0.8V logic low threshold. Because port transistors in 1-Wire devices only hold the line down after the bus master pulls it low, they do not normally exhibit slew rate problems. The exception is when a Presence pulse is generated in response to a Reset command from the bus master since the slew rate of the Presence pulse of a 1-Wire slave is essentially uncontrolled. Disruptive edge rates also occur when a new 1-Wire device is connected to the net.

Phantom Presence pulse

If the net consists of a fixed collection of 1-Wire slaves, the slew rate problem of their Presence pulse can be overcome by generating a "Phantom Presence Pulse" with the bus master. This is simply an artificial slew rate controlled Presence pulse created by the bus master that starts at 10μS and terminates at 60μS. A sample pulse can then be generated at 70μS to see if a slave is on the bus or node if desired. This technique masks the high slew rate of the slaves high-to-low Presence pulse transition because the bus is already at logic zero when they occur. However, the Phantom Presence pulse technique is obviously ineffective with new slaves which arrive at unpredictable times on the bus.

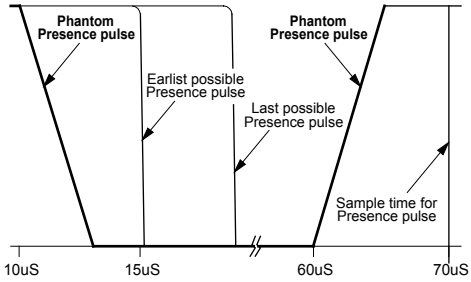


Figure 2 A Phantom Presence pulse can be generated by the bus master prior to any possible Presence pulse from a slave. It must last until the slowest possible Presence pulse has started.

Controlling the edge

Figure 3 shows a suggested slew rate control circuit. The 2N7000 shown, is a commonly available general purpose n-channel FET, but transistor characteristics are not critical and almost any general purpose n-type transistor may be substituted. A bipolar type such as the 2N2222 may also be used with minor component value changes to provide the recommended slew rate. Refer to the Appendix for additional information and a 1-Wire waveform template.

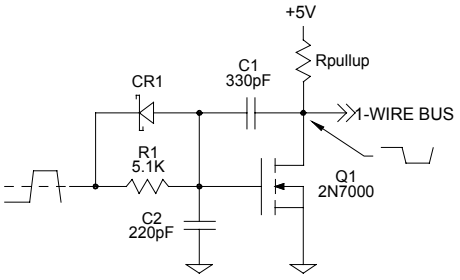


Figure 3 A controlled slew rate pull down for the 1-Wire net bus master. Transistor type is not critical.

Pulling up the line

Once both master and slave turn off, the bus pull-up resistor pulls the data line high. As the capacitive load on the net increases by adding 1-Wire devices, the time to raise the data line to the supply voltage also increases. This also occurs when the network is lengthened due to the 50pF of capacitance added per meter of twisted pair cable. This can be seen in **Figure 4** as the number of slaves is increased from 1 to 300. If the product of the total capacitive load (including cable, device, stray capacitance, etc.), and the pull-up resistor value results in a time constant (RC) that exceeds the bit time slot defined by 1-Wire

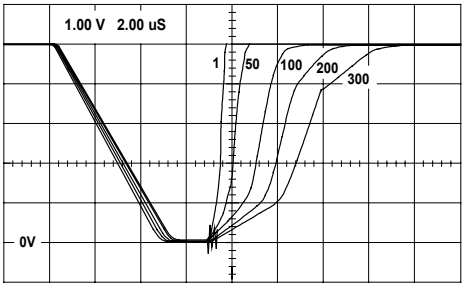


Figure 4 Loading effect of increasing the number of 1-Wire devices using active pull-up, and 2m cable.

protocol, communication stops. Because grounding unused wires or shields in a cable adds capacitance

which can significantly increase the RC time constant, they should be left disconnected.

As can be seen in **Figure 6**, the input capacitance of 1-Wire devices contribute to the capacitive load on the network. However, the 800 pF parasitic power supply capacitance only exists at voltage levels above 2.8V minimum. Ignoring the capacitance of the parasitic power capacitor, the bus pull-up resistor value, together with the cable capacitance and 1-Wire device input capacitance represent the network time constant τ. This is a reasonable omission since parasite capacitance does not become a factor until the bus has already passed the 2.2V logic one threshold. The network time constant determines the rate at which the data line returns to a logic one voltage. With the requirement that at t equals 13.02 microseconds (the original data sample time) the 1-Wire voltage needs to have reached the 2.2V threshold of a logic one, the value of τ can be calculated as follows.

$$\tau = 13.02\ \mu s / \ln (Vs/(Vs-2.2V)) = 22.4\mu s$$

Where Vs is the pull-up supply voltage. Using the recommended 1.5K minimum pull-up resistor value and 5V supply voltage, τ is calculated as 22.4 microseconds. Assuming the net is loaded with the maximum fanout as calculated in a later section, the cable capacitance alone must not exceed 12nF to yield a network time constant no more than the value just calculated. Using 50 pF/m for the typical cable capacitance implies that the theoretical maximum cable length is 240 meters. If the data sample time is recalculated with the new value of 21.7 microseconds, τ becomes 37.4µs. This permits 22nF of cable capacitance, which represents a cable length that exceeds the maximum allowed round trip propagation time. The effect of cable capacitance on the signal can be seen in **Figure 5**, where 100 1-Wire slaves were addressed at the end of a 2 and 100 meter cable. The 100 meter cable added 5000pF of capacitance.

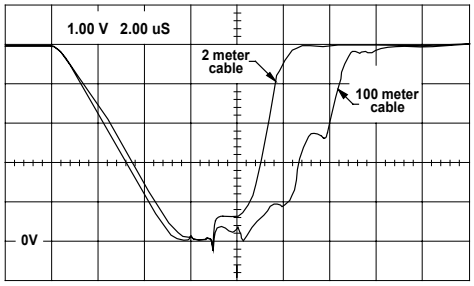


Figure 5 Effect of cable capacitance on signal driving 100 1-Wire devices. The 100m cable adds 5nF.

The risetime can be improved by reducing the value of the pull-up resistor, using lower capacitance cable, shortening the cable, or reducing the number of devices on the bus. The pull-up resistor, however, should not be reduced below 1.5 kΩ. Reducing the value of the pull up resistor increases the logic zero voltage on the network, reducing system noise immunity. If the value of the pull-up resistor is already minimum, an active pull-up may be substituted. This also allows use of longer cables by decreasing the network time constant. Of course, the same rules apply to an active pull-up as to a pull down, and slew rate must be controlled to avoid operating in a transmission line environment. The effect of residual current flowing in the bus when the active pull-up turns off can be seen in **Figure 8**. This was of little concern with a passive pull-up as an RC time constant has an inherently slow slew rate.

An active pull-up

One convenient source of an active pull-up is the MAX6314. This part was designed by MAXIM as a bi-directional open-drain µP reset for the 68HC11. However, it contains circuitry to automatically enable a 20mA p-FET pull-up for 2 microseconds when the data line exceeds a trip voltage of about 0.6V on the rising edge. Unfortunately, it does generate a logic zero output whenever the supply voltage drops below its reset threshold (it's intended use). For use on the 1-Wire net this can be detrimental as heavily loaded data lines can cause a reset, disabling communication, so selection of a part with a low reset threshold is suggested. One possibility is the MAX6314US31D3-T. This part has a 3 volt trip level that allows the supply to drop 2 volts before it

generates a reset. The MAX6314 comes in a four pin SOT143 package that requires little PCB area.

Although the MAX6314 contains an internal 4.7K pull-up resistor, it is recommended that an external 2.2K resistor be added in parallel. This provides the equivalent of the recommended minimum 1.5K pull-up resistor, which results in the bus crossing the trip voltage in minimum time. The bus waveform then exhibits three distinct segments. When both the master and the 1-Wire devices release the data line, it starts rising at a rate determined by the value of the pull-up and the total capacitive load (RC time constant). When it passes its trip threshold of approximately 0.6V, the 20mA p-FET is turned on and accelerates the bus toward the supply voltage. If the bus is heavily loaded, the 2 microsecond one-shot may time out with the bus well below the supply voltage. If this occurs, the 1.5K equivalent pull-up resistor continues to raise the bus voltage at the rate seen when the bus was first released. All three of these segments can be seen in **Figure 4**. This data however, was taken with a discrete proprietary design and not with the MAX6314. Refer to **Chapter 2** for additional information on active pull-ups.

The maximum voltage to which the bus pull-up resistor can raise the data line is determined by the product of the pull-up resistor value and the idle current of all devices on the line. The more devices, the greater the voltage drop across the pull-up resistor. The fanout limit of a particular 1-Wire net is reached as the voltage drop across the pull-up resistor reduces the net supply voltage to 2.8V. This is the minimum voltage that will recharge the parasitic power supply of the 1-Wire devices. From this, the maximum theoretical fanout may be calculated. It is equal to the supply voltage (Vs) minus 2.8V (the minimum operating voltage) divided by the pull-up resistor value. The resultant is divided by 15µA, the worse case device supply current. For a 5V supply and 1.5K minimum pull-up resistor value we have the following.

Fanout $_{MAX}$ = (5-2.8)/1.5K=1.47mA/15µA=98 devices

This represents the theoretical maximum number of 1-Wire devices that can successfully communicate with the master using a 1.5K pull-up resistor and 5V supply over worse case conditions of current and temperature. The assumptions being that all devices are drawing the maximum supply current and operating in a -40 to 85°C environment. In the real world, all devices will only be drawing the 15µA maximum supply current during System Reset and Presence Detect. At that time all device oscillators turn on for 5T times. Since 1T time typically lasts 30 microseconds, 5T times represents 150 microseconds, with a worse case of 255 microseconds. Circuit design ensures that all 1-Wire devices will be able to operate from their internal parasite power source for the duration of this interval once fully charged. Thereafter, they will be drawing 5µA maximum which permits tripling the previously calculated fanout of 98. In addition, most systems will be operating over a much narrower temperature range which allows still larger fanouts. For example, in a typical lab environment, over 500 1-Wire devices in continuous communication had only a 1.2V drop across the pull-up resistor. This implies that typical idle current per device is less than 2µA when environmental and supply ranges are limited.

SECTION 3
Its all in the cable

Take care in selecting a cable

As previously mentioned, the 1-Wire net consists of three segments, a bus master, the wiring and connectors and the 1-Wire slaves, as shown in the electrical equivalent circuit of **Figure 6**. The wiring between the master and the slaves is modeled by the inductance and resistance of the data and return lines and the lumped capacitance of the cable. Cable capacitance is simply the product of cable length times unit capacitance. This is typically 50pF/m for the recommended Category 5 twisted pair cable. Similarly, line resistance represents cable length multiplied by the specified resistance per meter of a single wire. A 1-Wire slave is modeled by its input capacitance (Cin), a constant discharge current (Idisc), the parasitic power supply circuitry (Di, Ri, Cload) and its operating current (Iop) of 10 µA during communication. The 5µA idle current per slave is required to keep its interface synchronized with the communication protocol. When the 1-Wire port transistor is on, its impedance is nominally less than 100 Ω, which provides a 0.4V logic zero with a 4mA current sink. If multiple 1-Wire devices are residing

on the bus, Cin, Idisc, Iop and Cload should be multiplied by the number of devices to determine the total. Ri needs to be divided by the number of devices. The port transistor inside the slave allows it

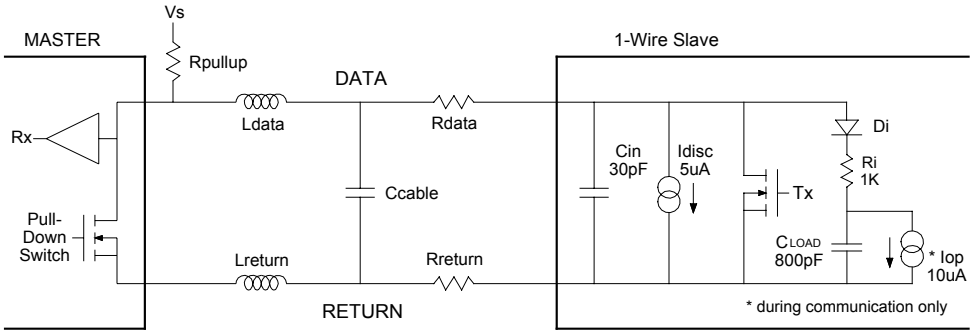


Figure 6 Electrical Equivalent Circuit of the 1-Wire net

to place a logic zero on the network. Except for the presence detect cycle, Search, Skip and Read ROM commands, only one of them will be conducting when addressed by the bus master.

While parasitic resistance reduces the zero logic level noise margin of the digital signals, cable capacitance together with the parasitic power supply of 1-Wire devices adversely affect the size of the network. At power up, this capacitive loading can require several milliseconds to charge before communication can start on the net, especially if a passive (resistive) pull-up is used. Also on long lines with many slaves grouped at the end, the parasitic power requirements create a dip or slope change in the rising edge of the waveform at about 2.8V as the energy reservoirs of the devices are filled. Once full, the recovery time after each time slot will be sufficient to maintain the charge. **Figure 7** shows the effect charging the parasite power capacitance has on different numbers of 1-Wire devices at the end of 100 meters of Category 5 cable. Notice that the slope of the rising edge decreases as it crosses the 2.8V threshold finally reversing direction to form a dip when loaded with 100 or more devices. These “dips” become more pronounced the longer the bus remains low.

Clearly the physical properties of the cable connecting the master and 1-Wire slaves strongly dominates the network. Comparison of the waveforms shown in **Figure 4** taken with two meters of Category 5 cable to those of **Figure 7** taken with 100 meters makes this plain. For short runs up to 30 meters, cable selection for use on the 1-Wire net is less critical as the impedance characteristics are generally insufficient to have a marked adverse effect on the bus. Even flat modular phone cable works with limited numbers of 1-Wire devices. However, the longer the net, the more pronounced cable effects become, and consequently the greater the importance placed on cable selection.

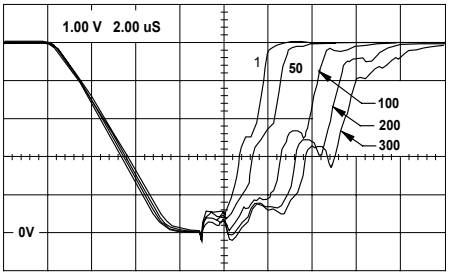


Figure 7 Parasite power loading of 0 to 1 transition. Notice that the dip becomes more pronounced with increasing number of devices at end of 100m cable.

Given sufficient length, any cable exhibits transmission line effects. Real cables display distinct properties of resistance, capacitance and inductance, which in turn are determined by cable geometry, the size and spacing of the conductors and their surrounding dielectric. These physical properties define the characteristic impedance, the signal bandwidth supported and the propagation

velocity of the cable. Specifically, cable resistance reduces the zero logic level noise margin, although values to 100 Ohms for the total cable length are acceptable. Cable capacitance however, which can range from 30pF/m to 100pF/m, loads the 1-Wire net driver, increasing not only the network time constant (τ) computed earlier, but also the peak current flowing in the cable as the bus pull down transistor turns on and discharges the line.

If this transistor turns off before the charge stored in the line capacitance is completely discharged, the residual current left flowing in the line determines the amplitude of a transient voltage spike generated as a product of this current and the cable inductance. The resulting voltage spike seen at the driver can become large enough to interfere with communication. The effect of residual current flowing in the cable when the bus pull down transistor and the active pull-up turns off can be seen in **Figure 8**. Notice that in each case, the spike generated is in the direction of the opposite rail. At the far end of the cable, when the pull down transistor turns off it's inductively

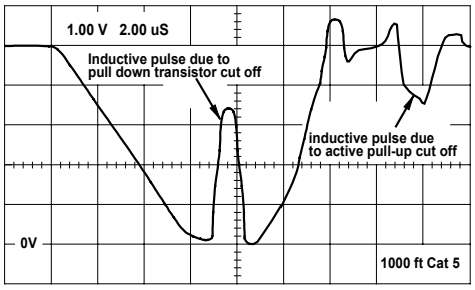


Figure 8 Inductively generated voltage spikes such as seen here, can occur on long lines due to residual current flow when pull down or pull-up circuits turn off.

generated voltage spike swings negative, reverse biasing the substrate of the 1-Wire device closest to the cable end which clamps the voltage excursion at a diode drop. This device will then not respond to the bus master.

The problem is differential inductance, which is measured across the cable input with the line shorted together at the far end. Differential inductance is substantially lower than the inductance of a single wire because the current flows in opposite directions in the pair and in the ideal case would cancel completely. Because differential inductance decreases as the distance between conductors is reduced, use of adjacent and preferably, twisted pair is recommended. Twisted pairs help reduce unwanted coupling from nearby interference sources because the currents induced in the two wires flows in opposite directions and tends to cancel.

Another concern is that the recommended category 5 unshielded twisted pair cable is commonly available with multiple pairs. While the capacitance between wires in a single pair is approximately 50 pF/m; that between wires of different pairs is closer to 30 pF/m. Because grounding the unused wires will add this 30 pF/m to the 1-Wire capacitive load, unused wires and shields need to be left unconnected at both ends of the cable. Grounding them can increase the capacitive load to the point that the bus pull-up cannot raise the line above the logic switching threshold within the bit time slot and communication stops. Refer back to Section 2 and the discussion of network time constant in "Pulling up the line" for more information. It is also not recommended to simultaneously run two 1-Wire nets in the same cable bundle, because the capacitive load varies dynamically dependent upon data pattern, which can lead to erratic operation.

Keep the bus running

As line inductance increases, the product of L di/dt can generate voltage excursions that cause bit errors and reverse bias the substrate of at least the first 1-Wire device at the far end of the cable. These voltages spikes are generated by the current still flowing in the data and return lines of the cable when the transistor in the master is turned off before the charge stored in the line capacitance is fully discharged. The obvious and recommended solution is to maintain the pull down transistor in the on state until the current in the line discharges. If it is not possible to stretch the timing, a Schottky diode placed across the bus at the far end is suggested to clamp the inductive generated voltage overshoot. Connect the diode across the cable with the cathode on the data line and the anode on the return. Only one diode is required for each branch.

SECTION 4
Rewiring the LAN

The DS2409 MicroLAN Coupler

Historically, the 1-Wire net was envisioned as a single twisted pair routed throughout the area of interest with 1-Wire slaves being attached in a daisy chain fashion where needed. However, if the network is heavily loaded, and/or very long, it may be preferable or even necessary, to separate the bus into sections. This has the added benefit of providing information about the physical position of a 1-Wire device on the bus which facilitates trouble-shooting. By using one section as the main "trunk", and adding or removing segment "branches" to it with a DS2409 as needed, a true 1-Wire net is created. It is also possible of course to add or remove additional segments to the branches using a MicroLAN Coupler as the node control. This approach reduces the capacitive and idle current loads the bus master sees to that of the trunk and those segments connected to it by activated DS2409's. However, the limitations to the total capacitive load, idle current and line length covered in earlier portions of this document still apply. That is, for the trunk and activated sections of the LAN, the capacitive load must be low enough to allow the bus pull-up to raise the line above the logic one threshold within the time slot. The combined idle current of the 1-wire devices must not reduce the supply voltage below 2.8V. And the electrical length of activated cable must allow a transition from the master to reach the cable end and return within the time slot.

The DS2409 MicroLAN Coupler is a key component for creating complex 1-Wire nets. It contains the MAIN and AUX transmission gate outputs, plus an open drain CONT output transistor, each of which can be remotely controlled by the bus master over the 1-Wire network. Both the MAIN and AUX outputs support a "smart-on" command which generates a reset/presence sequence on the selected output before connecting to the 1-Wire bus. This allows a subsequent ROM function command to apply only to the devices on the just activated segment. The main caution is that the DS2409 requires a 5V supply without which it shorts the 1-Wire bus and no communication is possible. Since the DS2409 contains no user available memory, the AUX output can be used to label the node by connecting a 1-Wire memory chip to store the required data. A simple 1-Wire net branch with EEPROM labels connected as described is shown in **Figure 9**. Tagging protocol information is available on the Dallas Semiconductor ftp site at ftp://ftp.dalsemi.com/pub/auto_id/softdev/softdev.html.

In the example, a DS2430 EEPROM is connected to the DS2409 AUX output. This provides tagging information specific to that particular node such as location, function, etc. The LED attached to the CONT output provides visual indication of the specific branch being addressed and can be caused to flash via software for extra visual impact. A single DS1820 Digital Thermometer is shown on the branch output but multiple 1-Wire devices can be placed as required.

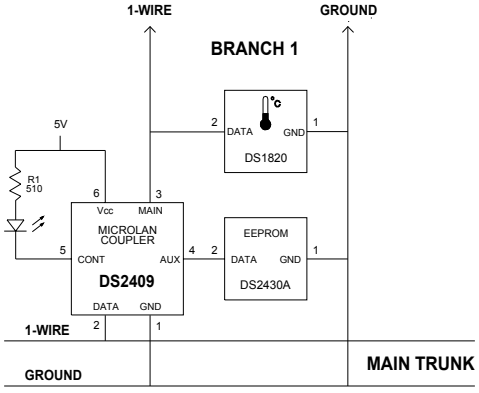


Figure 9 Separating the 1-Wire bus into branches using the DS2409 MicroLAN Coupler.

To avoid loading the entire 1-Wire net with the capacitance associated with routing power along with the 1-Wire DATA and GND, the CONT output of the DS2409 can be used to operate a pass gate transistor to switch the 5V supply along with the MAIN output switching the branch. The MICREL MIC94031 transistor shown in **Figure 10** contains the gate pull-up resistor shown, so implementing the circuit requires only the addition of a single SOT-23 package. By default, whenever the MAIN output of the DS2409 is turned on, the CONT output is pulled low, turning on the pass gate and routing power to

the selected branch. Turning MAIN off automatically turns CONT and therefore the pass gate off, so no software revisions are required for the circuit. The DS2409 is available in a 6 pin TSOC surface-mount package. For further information about the DS2409, refer to the datasheet available on the Dallas Semiconductor website at www.dalsemi.com.

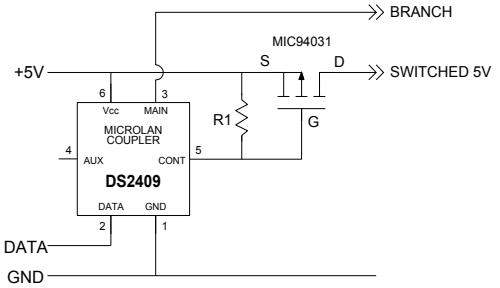


Figure 10 Method of switching local power to a branch using a DS2409 and a pass transistor.

The DS2406 dual addressable switch

The DS2406 is a low side addressable switch intended for remote control perhaps in combination with an opto-coupler or relay; or to provide visual indication of an operation or end-of-limb by means of an LED (Light-Emitting-Diode). The DS2406 is **not** recommended as a means for providing branches on a 1-Wire net, because branching requires use of a high side switch. This is because current flowing through the on-resistance of the port transistor develops a voltage that raises the ground pin of all devices connected to it above the ground reference. Since the 1-Wire slaves will be hard wired to the data the potential on the data pin becomes negative with respect to the voltage on its ground pin. This polarity acts to forward bias various p-n junctions in the slaves resulting in device dependent disruptions. For example, when this voltage reaches approximately negative 0.3V, it can continuously activate the POR (power-on-reset) of any attached DS2406, effectively removing it from the bus.

The DS2406 addressable switch was designed to perform closed-loop control of its two open drain output transistors, PIO-A and PIO-B, over twisted pair cable up to 300 meters from a PC. Each output can also sense and report to the PC the logic level at its port. When turned on by the bus master, the PIO-A port is connected to the IC ground pin and can sink up to 50 mA, and hold off a maximum of 13V, while the PIO-B sinks 8 mA and holds off 6.5V. The on-resistance of the DS2406 PIO-A output transistor is about 10 Ohms, whereas the PIO-B output is about 50 Ohms. The off impedance of both addressable switches exceeds 10 Meg Ohms. The DS2406 is available in a 6 pin TSOC surface-mount package, or as a single channel (PIO-A) version in a TO-92 package. For more information on the DS2406 dual addressable switch which contains 1K of one time user programmable EPROM, refer to the datasheet available on the Dallas Semiconductor website at www.dalsemi.com.

Using both the high and the low switch

While the DS2409 and DS2406 are very useful when used alone, together they can form the basis of a general purpose Pick system. **Figure 11** shows two DS2409s being used to select an arbitrary row, while a dual DS2406 low-side switch is used to select an arbitrary column. As shown, they form a simple 2X2 array with LEDs to visually indicate the specific intersection being addressed by the bus master. However, the array can be easily expanded in either the X or Y direction by the addition of more DS2409s and/or DS2406s. In this manner an M by N array of any required size may be implemented limited only by net loading.

In operation, the master selects the Aux output of the DS2409 that controls the row of interest, and the column output of the corresponding DS2406 that intersects that row at the required position. For example, if the AUX output of the top DS2409 and the B output of the DS2406 are both turned on, the position in the upper right hand corner is selected. This connects the iButton port at the intersection of the selected row and column to the master so the serial number of the 1-Wire device (if any) at that point can be read. To indicate which intersection is being addressed, the master switches the selected DS2409 from its Aux output to its Main output. By default this causes the CONT pin to turn on, grounding the gate of the associated pMOS transistor and turning it on. With the pass transistor on, power is supplied to the LED at the selected intersection and turns it on. If desired, the DS2409 can be switched repeatedly between Main and Aux causing the LED to blink for greater visual effect. If

the Main outputs of all DS2409s are turned on, the LEDs in the entire column of the selected DS2406 turn on. Alternatively, if the outputs of all DS2406s are turned on, the LEDs in the entire row of the selected DS2409 turn on. Consequently, it follows that turning on all column and row switches illuminates the entire array which serves as a convenient test to verify that the system is fully functional. While an iButton port was shown in the example, a blue-dot connector or even solder mount devices could be substituted.

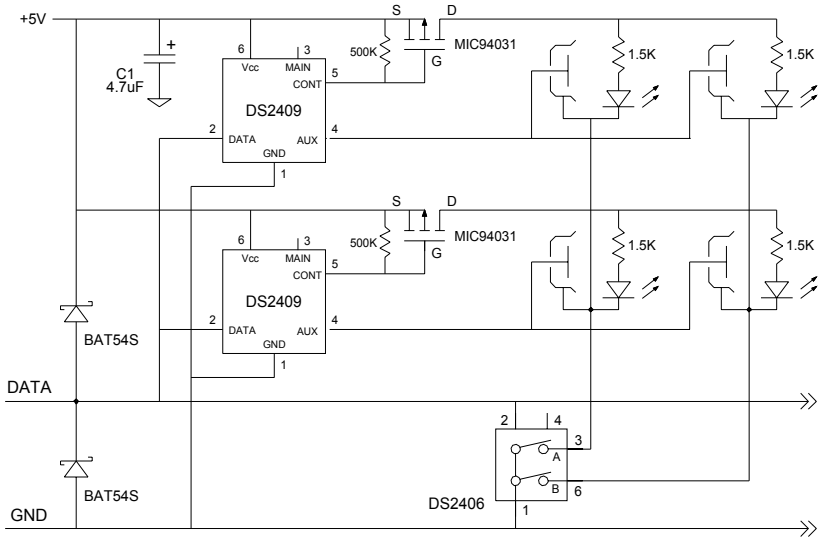


Figure 11 Using DS2409s and DS2406s to form an X Y matrix Pick system with visual indicators.

SECTION 5
Policing the LAN

Protection and Noise

All 1-Wire net compatible devices contain built-in ESD (electrostatic discharge) protection circuitry able to withstand ± 10kV minimum Human Body Model ESD events. However, when interfacing a computer to a 1-Wire net it is good engineering practice to provide additional means to protect the system from ESD and EMI (electromagnetic interference) damage. Economical protection can be provided at the bus master by means of diodes that short negative spikes or shunt positive ones higher than 5V to the power supply bus where they are absorbed by capacitors and other protection devices. When selecting protection diodes or surge suppressers, choose devices with minimum junction capacitance and fast switching times. Two good examples are the ERA82 Schottky diode from FUJI or the multi-source 1N5817. The BAT54x series of Schottky diodes available in the SOT-23 surface mount package is also acceptable. Refer to **Figure 11** for an example using the BAT54S dual diode.

If 1-Wire EPROM parts will not be programmed using the RS232 to 1-Wire interface adapter, excellent protection can be provided with a single DS9502 ESD protection diode. These behave like a 7.5V zener diode during normal operation, however, if the voltage across them exceeds their 9V trigger voltage they fold back to 5.5V. If exposed to an ESD event beyond their ratings, the DS9502 will eventually fail shorted, preventing damage to the protected circuitry. The DS9502 is available in a 6 lead TSOC package, or as a CSP solder mount “bumped” die.

Introducing the DS2480

In order to reduce the engineering load of setting up a 1-Wire net, Dallas Semiconductor has developed the DS2480 Serial 1-Wire Driver. This IC connects

directly to UARTs and 5V RS232 systems. Adapters are available using the DS2480 that connect directly to a standard COM port and provide 1-Wire outputs. The IC contains programmable pull-down slew-rate control and an active pull-up. Other features include support for data rates of 9.6 (default), 19.2, 57.6 and 115.2 Kbps and programmable 1-Wire timing. The DS2480 is available as the DS9097U-x09, which is a DB9 to RJ11 COM port adapter, and the DS1411 with a DB9 to DS9098 iButton socket. These are true ground Crypto-capable adapters with FCC class B approval. A schematic of the DS9097U is provided in Chapter 2.

Its the results that count

By applying information presented in this application note such as using Category 5 twisted pair, controlling slew rates and substituting an active pull-up, reliable communication over 300m of cable with more than 500 assorted 1-Wire devices was demonstrated. Without slew rate control and active pull-up, the limit is about 100 meters with 150 1-Wire devices.

Recommendation summary for operating long or heavily loaded MicroLANs:

Select cable

- Good -twisted pair phone
- Better -Category 5 twisted pair

Use a diode clamp

- Use a Schottky diode across the cable's end. Connect the diode reverse biased with the cathode on the data line and anode on the return.

Use multilevel branching

- Use DS2409 MicroLAN Couplers to separate the net into branches.

Use a DS2480 based COM port adapter

- a DS9097U-009, a DB9 to RJ11 adapter
- **or a**
- DS1411, a DB9 to DS9098 socket adapter

If a DS2480 type adapter is <u>not</u> used,

Control slew rate

- Limit the slew rate of the pull down to about 1.1 volts per microsecond.

Use an active pull-up

- Replace the pull-up resistor with an active pull-up, limiting slew rate to about 1 V/μS.

Keep the bus running

- Keep the bus master transistor on until residual line current dissipates.

Use later sampling

- Use 21.7 microsecond timing (2.5T), instead of the original 13.02 microsecond (1.5T) timing.

CHAPTER 2
Hardware for the 1-Wire net

PC to 1-Wire net adapters

Any PC with a UART capable of 115.2 kbps can serve as bus master for reading and writing 1-Wire devices on the net. However, a serial COM port to 1-Wire adapter is needed to interface the computer RS232 levels to the 5V 1-Wire bus. A schematic for the original DS9097 COM port adapter developed by Dallas Semiconductor as an economic and easy to use interface, is shown in **Figure 12**. The adapter is powered entirely from the computer COM port. Note that the circuitry makes maximum use of the UART controlled output using only clamping and level shifting diodes. Although this provides a simple and reliable circuit that directly couples to the serial port, it lacks a ground reference. While not of concern in operating the net, it can present complications during trouble shooting, as the oscilloscope ground cannot be connected to the 1-Wire return. Connecting the DS9097 1-Wire return to ground can result in permanent damage to the adapter.

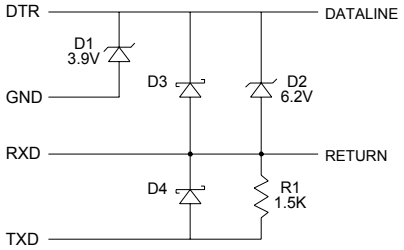


Figure 12 The DS9097 COM port adapter. Note that "return" is not ground. R1 is the bus pull-up.

In **Figure 12**, current limiting and slew rate control are provided by the UART as RS232 requirements, while zener D1 clamps the data line at 3.9V. Zener diode D2 limits the maximum voltage range on the 1-Wire bus to 6.2V. It also restricts the most negative voltage swing on RXD to minus 2.3V. When TXD is positive, Schottky diode D3 limits the voltage difference between the 1-Wire data and return lines to 0.2V, and D4 connects TXD to RXD. This bypasses R1 the passive bus pull-up, and provides a low impedance path to initiate a time slot. The 1.5K minimum value pull-up resistor generates a 0.3V logic zero across the 100 Ohm on-resistance of a 1-Wire device. Since the maximum voltage still recognized as a logic zero is 0.8V, this leaves 0.5V of noise margin.

The DS9097 comes with a DB9 for attachment to a PC serial port and an RJ11 for connecting to the 1-Wire net. The same circuitry is available in a DB9 to DS9098 socket as the DS1413. A DS9097E version in a DB25 case is also available for programming EPROM based 1-Wire products such as the DS198x series. It provides a power jack to accept an external 12V auxiliary supply as required.

A true ground COM port adapter

If the return line of the net must be grounded at some point, a true ground COM port adapter interface is required. The schematic for a true ground adapter is given in **Figure 13**. Its positive supply is derived from DTR and RTS by Schottky diodes CR1 and CR2, and filter capacitor C1. A well-regulated 5 volts is provided by the low dropout LP2980 voltage regulator as long as the COM port provides at least 5.1V. If the positive level of both DTR and RTS drops below this the regulator drops out of regulation. CR3 and C3 provide the negative supply required by the DS275 RS232 Transceiver chip from TXD during read data and idle time slots. The DS275 connects directly to the 1-Wire net converting it's CMOS/TTL levels to the RS232 levels required by the UART. If surface mount components are used, the circuitry will fit comfortably on a 0.6" by 0.9" printed circuit board. This can be mounted inside a DB9 female to RJ11 PC adapter.

A slew rate controlled pull down

Because it is not possible to terminate the end of the cable in its characteristic impedance, the alternative is to control the slew rate of the bus master pull down transistor to prevent adverse transmission line effects. For more information on this subject, refer back to Chapter 1 and "Control the slew rate of the driver." In **Figure 13**, transistor Q1 and the components associated with its gate, limit the slew rate to about 1.1 volts per microsecond. This provides a one to zero transition that takes about 4 microseconds to ramp from the 10 to 90% points. This has proven to be adequate for cable lengths exceeding 300 meters. The 2N7000 shown is a commonly available general

purpose n-channel FET, but transistor characteristics are not critical and almost any general purpose n-type transistor may be substituted. A bipolar type such as the 2N2222 may also be used with a base resistor and minor component value changes to provide the recommended slew rate.

In the pull down circuit of **Figure 13**, Miller integrator capacitor C5 across Q1 acts as the primary slew rate control element. On lightly loaded LANs using about 50 devices on 50 meters or less of cable, it may be possible to eliminate C5. Sufficient slew rate control is then provided by R1 and C4. In operation, when TXD switches from its minus 12V rail to the plus 12V rail, the signal is passed to the gate of Q1 by input resistor R1. This transition is slowed and delayed slightly by the RC time constant of R1 and C4. When the voltage level at the gate reaches the threshold voltage of Q1, the transistor begins to turn on. The effect of the transistor turning on with C5 across its terminals results in a nearly linear voltage ramp from the supply voltage to ground. This one to zero transition on the bus is the synchronizing edge for 1-Wire communications. Q1 remains in the on state for the duration of time that TXD is held at plus 12V by the UART. Afterward, the bus is returned to the supply voltage by the action of pull-up resistor R2. Zener diode D1 serves several functions in the circuit. It protects the gate of Q1 by limiting positive voltage excursions to 12V, and negative ones to minus 0.6V. Gate input resistor R1 limits the current through the zener. The capacitance of D1 also adds to that of C4, helping control the transistor transition.

While they are not shown for clarity reasons, good engineering practice would place reverse biased Schottky diodes across the output of the circuit. One diode should be from the supply voltage to the data line, the other from the data line to ground. Use of ultra fast low capacitance Schottky diodes such as the 1N5817 or the ERA82 from FUJI is suggested. The BAT54x series available in SOT-23 packages is also acceptable.

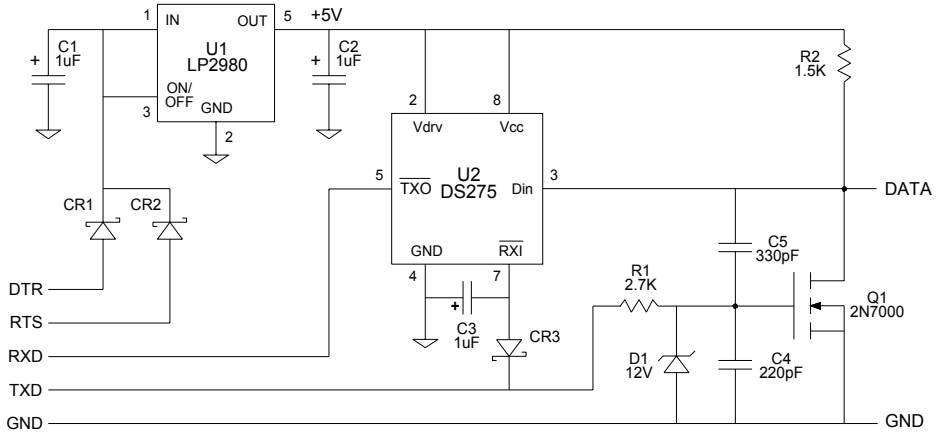


Figure 13 A PC serial COM port to 1-Wire net true ground adapter. Q1 is the slew rate controlled pull down and R2 is the passive bus pull-up resistor.

So that's how it works!

A cycle in the sequence of 1-Wire net communication over the true ground COM port adapter would proceed as follows. Initially, TXD would be at minus 12V (an RS232 "mark" or logic one), so the gate of Q1 would be clamped at minus 0.6V by the reverse bias on zener diode D1. Consequently Q1 is off, and the bus will have been raised to the 5V supply by pull-up resistor R2. When TXD switches between the minus 12V and plus 12V rails, Q1 will be turned on and ramp the bus voltage down at the rate set by the Miller integrator capacitor C5. This one to zero transition has two paths, one continues down the bus, the other returns to the UART through U2. The

signal returned to the UART, will be used to determine when to reexamine its registers for a response from a 1-Wire device on the bus.

Meanwhile, the transition traveling down the bus is received sequentially by any attached 1-Wire slaves as the edge propagates past them. If this signal from the UART is a Reset pulse, it lasts for at least 480 microseconds and then releases the bus. The bus pull-up raises the line to the supply, and approximately 30 microseconds later the UART reexamines its registers to see if a Presence pulse is being sent by a 1-Wire device. In the interim, the internal oscillators and controllers within each 1-Wire device have determined that a Reset has been sent and at the appropriate time will pull the bus low. The exact time the bus is pulled low and its duration, is a function of individual device variation. The fastest will pull the bus low first, and the slowest release it last. When the final 1-Wire device releases, the pull-up resistor raises the bus toward the supply voltage at a rate determined by its value times the capacitive load it sees on the line (RC time constant). If the pull-up cannot raise the bus above the logic one threshold within the time slot, the master will always see a logic zero and conclude the line is shorted. Consequently, communication cannot occur. Assuming however that the timing is acceptable, the UART sees a proper Presence pulse indicating there are 1-Wire devices on the bus. It then proceeds to call serial numbers to identify them.

In order to write logic one or zero values onto the bus, the UART turns on Q1 for short (less than 15 microseconds), or long (greater than 60 microseconds) time slots respectively. To read, the UART begins by turning on Q1 for a short time slot exactly as if a logic one was being sent. If this time slot remains unchanged, the UART defines this as reading a logic one. If a 1-Wire slave extends the time slot initiated by the master by continuing to hold the line low (even though the UART has released Q1 and R2 attempts to pull the data line high) the UART defines this as reading a logic zero.

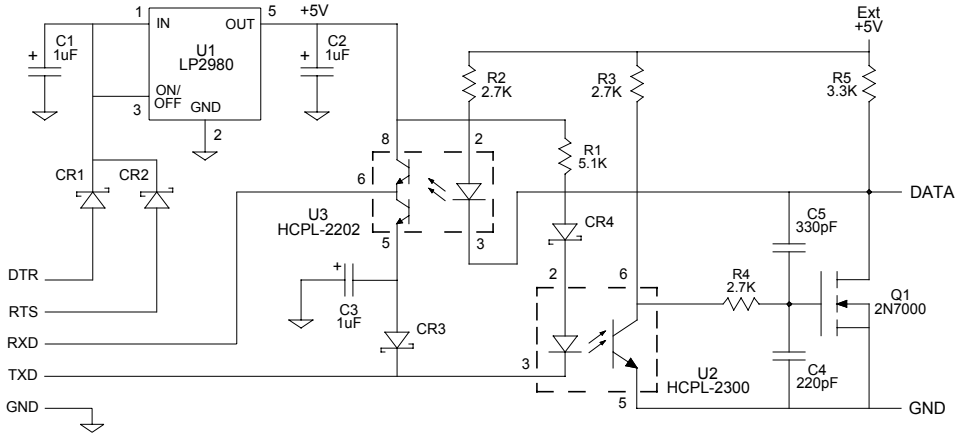


Figure 14 An optically isolated COM port to 1-Wire net adapter.

An optocoupler isolated adapter

In some applications, for safety reasons or because of ground loops, it is necessary to provide galvanic isolation between the computer master and the net. For such cases, the optically coupled adapter circuit in **Figure 14** is suggested. Optical isolation of the COM port requires special purpose optocouplers, such as the high speed, low input current HCPL-2300 used in the transmit section. An HPCL-2202 with 20V totem pole output is used in the 1-Wire to RS232 channel.

The HCPL-2300 has an 0.5mA input LED, and an output transistor with 200 nanosecond maximum

propagation time. In the circuit, the LED current is set with resistor R1, but a current source such as the J503 or CR056 from Siliconix will provide superior performance over the diversity of logic levels available with RS232. If a constant current source is not used it may be necessary to recalculate the value of R1 to maintain 0.5mA with the voltages in use on a particular RS232 COM port. CR4 is required to protect the LED, which only has a 5V breakdown in the reverse direction.

In operation, the HCPL-2300 coupler U2, provides the necessary isolation and level shifting from the double rail plus and minus 12V RS232 to the single rail 1-Wire net. The coupler is connected as non-inverting, so when TXD is at minus 12V, the LED input is on. Consequently, the coupler output transistor is also on and the bus pull down transistor Q1 is off. With Q1 off, bus pull-up resistor R5 holds the LAN at the supply voltage. U3, the receive optocoupler is also off, so its output is at minus 12V.

When TXD changes state, the HCPL-2300 input LED is reversed biased, turning the coupler output transistor off. Its collector is then pulled high by resistor R3. This zero to one signal is slowed and delayed slightly by the RC time constant of R4 and C4. When the voltage level at the gate reaches the threshold voltage of Q1, the transistor begins to turn on. The effect of the transistor turning on with Miller capacitor C5 across its terminals results in a nearly linear voltage ramp from the supply voltage to ground. When Q1 pulls the data line to ground, it also turns on the LED in U3 the HCPL-2202 coupler, causing its totem-pole output stage to switch to plus 5V. This transition will be received by the UART and used to determine when to reexamine its registers for a response from any 1-Wire devices on the bus.

The data line remains at ground until TXD returns to its original state, and Q1 along with any 1-Wire slaves turn off. At that time, the bus pull-up resistors (R5 in parallel with R2) raise the data line toward the 5V supply at a rate determined by their equivalent value times the capacitive load seen on the line (RC time constant). This turns off the input LED of U3, and causes its output stage to switch to the minus RS232 voltage level. Assuming the pull-up can raise the bus above 2.2V within the time slot, the UART will see a logic one, and communication proceeds. If the pull-up cannot raise the bus above the logic threshold within the time slot, the master will always see a logic zero and conclude the line is shorted. Consequently, communication stops.

An active pull-up

On a 1-Wire net where the minimum acceptable pull-up resistor value cannot raise the data line above the logic one threshold within the time slot, an active pull-up must be used. Obviously, an active pull-up circuit should be on only during a defined range of the rising edge (zero to one transition). Conversely, it should not respond on the falling edge, nor be active during logic zero time intervals. It must trigger on the rising edge at about 0.9V plus or minus 0.1V to provide acceptable noise margin. Preferably, once triggered it will remain on until the line is raised above a specified threshold ($\geq$3V) rather than for a set time interval (one shot). This insures that the data line will be raised above the 2.8V level required to recharge the parasite power capacitors regardless of load. The maximum current supplied should be limited to about 15mA. Larger currents when flowing in cable inductance can cause problems. Refer to Chapter 1 Section 3 “Its all in the cable” and see **Figure 8** for more information on the effects of cable inductance.

In Chapter 1 Section 2 under “An active pull-up," use of the MAX6314 is suggested as an available circuit. This part was designed by MAXIM as a bi-directional reset intended for use with the 68HC11 microprocessor. As such, an important function of the chip is to monitor the supply voltage and assert a reset (logic zero) during power-up, power-down, or during supply droop. This obviously is an undesirable function on the net. However, the part includes an active pull-up to solve the RC time constant problem faced on a high capacitance bus. Coincidentally, this consists of a p-FET in parallel with a 4.7K pull-up resistor (the maximum 1-Wire net value). The FET is turned on for 2 microseconds when the waveform on the reset pin exceeds 0.9V maximum by a comparator triggered one-shot. Additional control circuitry ensures that the active pull-up is disabled at all other times.

Operation of the MAX6314 when connected to the 1-Wire net proceeds as follows. When the bus (connected to the MAX6314 reset pin) is pulled below 0.5V by the master, a comparator sets an internal flip-flop enabling the active pull-up control circuitry. When the bus is released, the internal 4.7K pull-up resistor starts raising the bus toward Vs (the bus supply voltage) at a rate determined by it's RC time constant. This generates a ramp that starts at the logic zero level and ends at the trip voltage of a

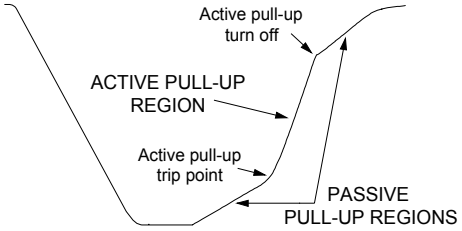


Figure 15 Characteristics of an active pull-up.

comparator. When the trip voltage of the active pull-up enable comparator is exceeded, it triggers a 2 microsecond one-shot that turns the 20mA p-FET on. This starts a second much steeper ramp that begins at the comparator trip voltage and ends at Vs. However, if the bus is heavily loaded, a third ramp may be created if the 2 microsecond one-shot times out, and the bus voltage has not reached Vs. This ramp starts where the one-shot times out, and ends at the maximum voltage to which the passive pull-up can raise the line with the idle current load of 1-Wire devices on the bus. This third ramp will have a slope similar to the first. The characteristic waveform produced by an active pull-up is illustrated in **Figure 15**.

Although the MAX6314 contains an internal 4.7K pull-up resistor, it is suggested that an external 2.2K resistor be added in parallel. This yields the equivalent of the recommended 1.5K minimum pull-up resistor, and allows the bus to cross the enable trip voltage of the active pull-up in minimum time. The combination of the MAX6314 and 2.2K resistor can be substituted for R2 in **Figure 13** to provide a true ground COM port adapter with active pull-up.

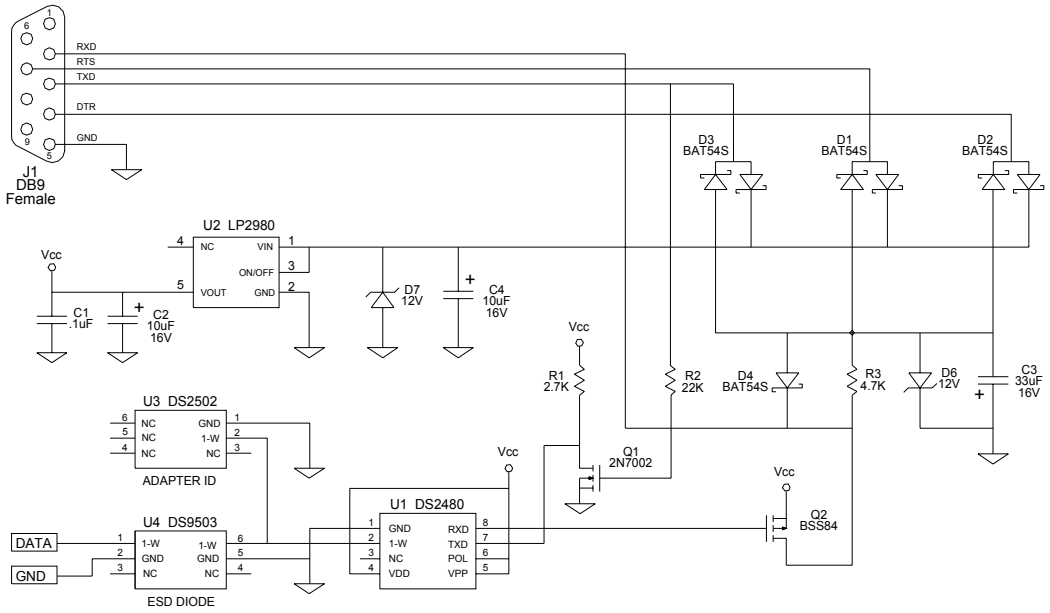


Figure 16 The schematic for the DS2480 based DS9097U COM port Adapter.

In order to reduce the engineering load of setting up a 1-Wire net, Dallas Semiconductor developed the DS2480 based series of COM port adapters. The basic schematic shown in **Figure 16** is used through-

out the series, the major difference being the type and number of 1-Wire connections on the output. The DS2480 provides programmable slew-rate control on the pull-down and an active pull-up. Other features include support for data rates of 9.6 (default), 19.2, 57.6 and 115.2 Kbps and programmable 1-Wire timing. The DS2480 also provides a strong pull-up mode to facilitate the higher current requirements of devices such as the DS1820 temperature sensor during conversion. The DS2502 shown is used to provide the adapter ID.

The DS2480 is available with and without an ID in a DB9 to RJ11 COM port adapter as the DS9097U-009 and DS9097U-S09 respectively. The circuitry is also available as the DS1411 with a DB9 to DS9098 iButton socket. All adapters are true ground Crypto-capable adapters with FCC and CE approval.

CHAPTER 3
Supplying power on the 1-wire net

1-Wire chips get both their power and communications over the same line making installation very economical. When adding power-consuming devices to the 1-Wire node, the same line can also supply power to them using one or more of the techniques described here. Typical examples could be as simple as driving an LED or operating a solenoid, or as complex as powering a pressure sensor. In such cases, the most convenient method would be to transfer the energy they require over the same communication line. This chapter will review some methods for providing power on the net including a technique for accumulating low level energy and releasing it on demand as a high energy burst.

In general, solutions to the problem of supplying power on the 1-Wire line fall into one of the following four methods. Keep in mind that regardless of which method is chosen, consideration must also be given to the energy required, its duration and its distance from the bus master.

- Sourcing power whenever the line is above 3.5V.
- Sourcing power by transferring charge to a capacitor through a blocking diode.
- Sourcing power with a strong pull-up during idle communication time.
- Alternative power source using additional wires and connections.

Tap the power between 3.5 and 5V

Because 1-Wire devices can operate with as little as a 3V supply, the energy available between the bus supply levels of 3.5 and 5 volts can be tapped. This is equivalent to operating the load in shunt mode and can be used to operate clamp type loads such as LEDs. This requires that the total voltage drop across the LED(s) be at least 3.5V. While it is possible to connect the shunt load permanently across the bus, preferably, the load would be operated under bus master control by connecting it between the 1-Wire DATA lead and the output of an addressable switch as shown in **Figure 17**. In this mode, 1-Wire communication takes place below 3.5V and power delivery occurs above that value. Whenever the output of the DS2406 is pulled low the LED is on and the voltage on the bus is approximately equal to 3.5V, the forward voltage of the LED. When the output is turned off, the LED is off and the bus voltage is at its nominal 5V value. Operational current is supplied by the bus master which for the DS2480 based DS9097U COM port adapter is normally limited to about 5 milliamps, but increases to about 15 milliamps when the active pull-up turns on. As shown in **Figure 17**, the bus voltage will be clamped to a level which keeps the active pull-up on and supplying 15 milliamps. A current limiting resistor connected between the DATA line and the LEDs will allow the active pull-up to turn off.

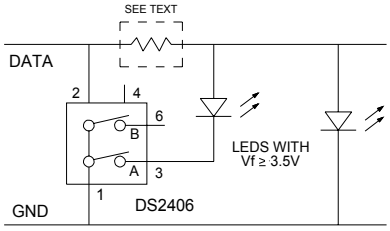


Figure 17 Using the power available between 3.5 and 5V directly and under bus master control.

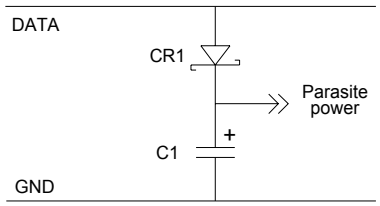


Figure 18 Using a Schottky diode and capacitor to supply local power on the 1-Wire net.

Transfer charge to a capacitor through a blocking diode

For some applications it may be acceptable to use a series Schottky diode and capacitor across the 1-Wire bus to generate a local supply on the net at the point of interest. Refer to **Figure 18**. The wind speed sensor in the 1-Wire weather station which uses a BAT54S for the Schottky diode and a .01μF ceramic capacitor to power the DS2423 counter uses this technique. See **Figure 23**. During idle communication periods when the bus is at 5V, the circuit 'steals' power from the line to charge the capacitor and power the load. This is a discrete implementation of the parasite power technique used internally by 1-Wire devices to provide their own operating power. The value used for C1 depends on the current consumption of its load and how long the voltage must be held above a design value. While simple and economical the circuit adds both leakage and capacitive loads that reduce the range and capability of the 1-Wire net. This loading places an upper limit on the capacitance value used and the number that can be placed on the net. Consideration must also be given to the fact that in the event the capacitor is shorted or held in a discharged state by its load, the net will also be shorted and inoperable. No further communication can take place until the capacitor is charged above 3.5 volts.

Deliver energy under bus master control

As shown in **Figure 19**, the half wave rectifier of **Figure 18** can be isolated between two addressable switches controlled by the master. When the input switch is closed, the capacitor receives charge over the DATA line of the 1-Wire net in the same manner as the circuit of **Figure 18**. A significant advantage of the arrangement is that when the switch is opened the capacitor and its charge is isolated from the net and normal communication resumes without the burden of the capacitive and leakage loads of C1. When the stored energy is needed, the output (power dump) switch is closed and the capacitor is discharged through the load. Note that while reference was made to a capacitor, a rechargeable battery could be used equally as well. Important elements of the concept and architecture are the low-level transfer of energy from the bus master to a storage element, and subsequent use in a high energy burst. Conceptionally, this is somewhat similar to the way circuitry in a flash camera develops the energy needed to fire a flashbulb. Equally important is the isolation of the storage element from the 1-Wire net so a failure doesn't bring down the net, and the complete control of energy source delivery by the bus master.

A practical example of the concept using a DS2406 as the control element and pFETs for the switches is shown in **Figure 20**. Notice that the MICREL MIC94031 FET isolation switches specified are four terminal devices with the substrate terminal brought out. This provides for correct biasing of the terminal under all operating conditions. The gate pull-up resistor shown unlabeled is internal to the chip but shown for clarity. In order to insure that both switches can not be turned on at the same time and

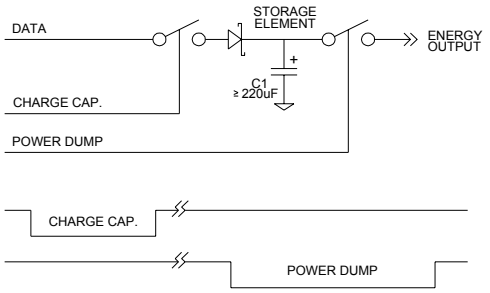


Figure 19 A parasite powered 1-Wire remote high-energy source concept.

possibly bring down the net, a lockout circuit is constructed using U2, a 74HC126 tri-state gate. By design U2 only allows alternate enabling of the pass gates to charge and discharge the energy storage element C1. As shown in the truth table of **Figure 20**, if both outputs of the DS2406 are simultaneously placed in the same logic state, either intentionally or by accident, U2 insures that neither pass gate is turned on.

In operation, C1 is charged by commanding output B of U1 (pin 6) the DS2406 to a logic zero. This turns on Q1, connecting the 1-Wire DATA line to C1 through diode CR2 which prevents C1 from discharging back through the 1-Wire net. If the diode were not present and a 1-Wire device were to be placed on the bus when the pass gate was turned on, its Presence Pulse would short and discharge the capacitor possibly damaging the chip. In the initial state with no charge on C1, the gate of Q2 the discharge pass gate, is held at a higher potential then the source terminal by pull-up resistor R4, so Q2 is off. When the bus master turns output B of U1 off, the charge stored on C1 is isolated from both the 1-Wire net and the load and only leakage paths exist to discharge it. When the bus master commands output A of U1 (pin 3) to a logic zero, pass gate Q2 turns on and C1 discharges through the load.

In a more sophisticated implementation of the concept, a barometric sensor was constructed using U1, a DS2450 Quad ADC as the 1-Wire addressable switch control element. The DS2450 also reads the charge level of the energy storage capacitor C1 and controls a sample-and-hold on the output of the

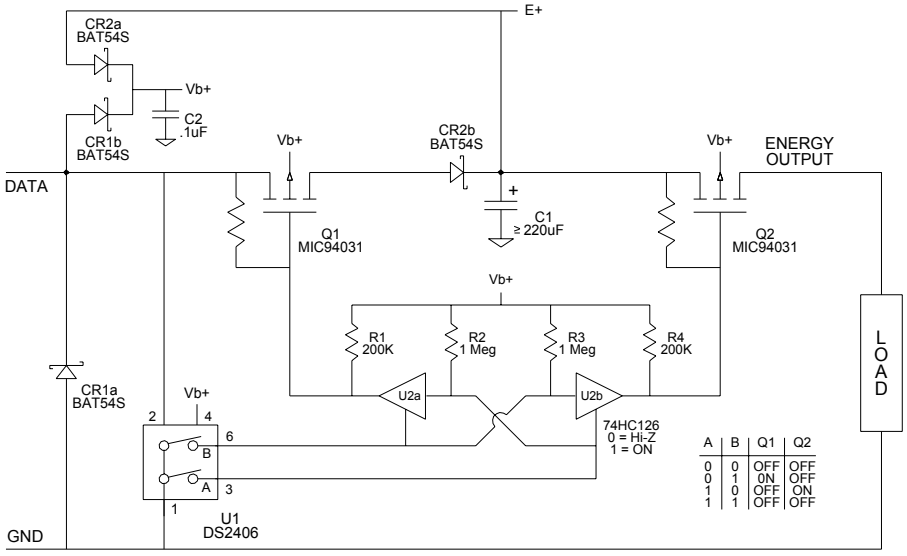


Figure 20 An isolated remote energy source based on the DS2406 addressable switch.

sensor. A major design consideration of the circuit was that the barometer required the energy source C1 to provide up to 10mA for 22mS. A schematic of the prototype circuit is shown in **Figure 21**. In the circuit, two DS2450 I/O pins are used as digital outputs that control the capacitor charge and discharge via analog switch U3 and a specialized switched-capacitor voltage regulator U4 used in place of the output (discharge) analog switch. The charge pump, a MAX684 from MAXIM, provides a regulated 5V ± 4% output as the energy capacitor discharges down to 2.7V. Surprisingly, the efficiency increases as the input voltage drops, a very useful feature when using a discharging capacitor as the energy source. The two remaining I/O pins of U1 are used as analog inputs which read the voltage on storage capacitor C1, and the voltage from the sample-and-hold (U7) that stores the output from the barometer representing the current barometric pressure. The circuit performed as expected with values up to 0.22 Farad for C1 the energy storage capacitor. Obviously, the higher the capacitance, the longer it takes to charge and the longer the voltage level is maintained relatively constant.

In operation, pulling U1.7 low closes analog switch U3 allowing C1 to charge through CR1. CR1 prevents C1 from discharging back through the 1-Wire net. The voltage generated by charging C1 can be read as needed by U1.8 to insure that sufficient energy exists to operate the load. When U1.7 is turned off, analog switch U3 also turns off and the charge stored on C1 is completely isolated from the net. At the appropriate time, U1.6 is pulled low which enables voltage regulator U4 providing a path for C1 to discharge through barometer U5. The MPXA4115 Motorola part requires 22mS maximum to turn on and stabilize, at the end of which time the output voltage representing current atmospheric pressure is stored on C3 the sampling capacitor. After the sample, U4 turns off to minimize energy loss from the storage capacitor C1. Instead of the wide interval used in the prototype circuit to sample the barometer it would be preferable to use a narrow pulse immediately after the output has settled.

Use additional available wires

In order to obtain maximum performance, Dallas Semiconductor recommends CAT 5 UTP (unshielded twisted pair) for use in routing the 1-Wire net. However, since CAT 5 typically comes with multiple pairs, there is a natural inclination to use an extra pair to route power. A look at some cable properties will help in understanding how such an arrangement affects net performance. In a CAT 5 cable with multiple twisted pairs, on average any given conductor in

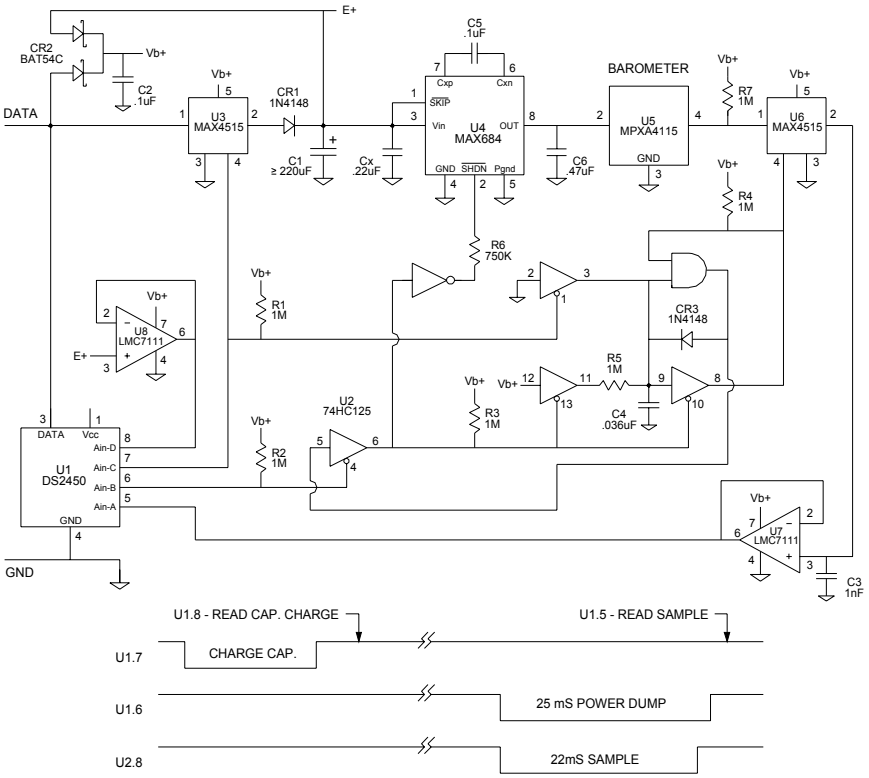


Figure 21 A 1-Wire barometer powered off the bus by an isolated and regulated energy source.

a pair is adjacent to a second conductor in another pair for half its length. Typically, the capacitance between the two conductors of a twisted pair is approximately 50 pF/m while that between the conductors of two different pair runs about 30 pF/m. Because grounding unused conductors will increase the capacitive load seen by the master, Dallas Semiconductor recommends that unused wires and shields be left unconnected at both ends of the cable. Still, within the limits set by increased capacitive loading and potential cross talk caused by using the extra pairs within the cable, external power can be routed along with the 1-Wire DATA and GND pair. However, the loading can reduce how long the bus can be and/or the number of 1-Wire devices the net can support. In addition, current and voltage variations on the pair carrying power can induce cross talk on the 1-Wire DATA and GND that disrupts communication.

Since the bus master sees less capacitive loading when routing power over flat cable where the conductors maintain a fixed and distant separation, flat 6-conductor phone (silver satin) may be used up to about 200 feet. Insure that DATA and GND have maximum separation from the power conductors by using the outer two conductors next to the 1-Wire GND to carry power. As shown in **Figure 22**, the wiring sequence should proceed in this order; NC (no connection); NC; 1-Wire DATA; 1-Wire GND, then external power and ground on the two outermost conductors. This arrangement helps shield the critical DATA lead from the additional capacitive load and cross talk of the external power leads. Notice again that the two conductors prior to the 1-Wire DATA line shown in dashed lines are to be left uncommitted. As repeatedly emphasized, if connected they will substantially increase the capacitive load seen by the DATA line. One possibility is to use 4-conductor silver satin and assemble the cable with

these two slots in the RJ11 connector empty. Unfortunately, a significant disadvantage of flat cable is that it lacks the noise rejection properties of twisted pair cable, so EMI may present a significant performance problem if the net is routed near sources of electrical noise.

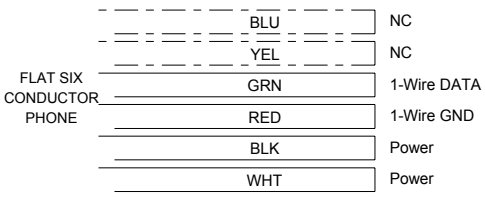


Figure 22 If power is to be routed over flat phone wire along with 1-Wire DATA and GND, use the outer two conductors by 1-Wire GND to minimize cross-talk and capacitive loading.

CHAPTER 4
1-Wire Instrumentation

In addition to 1-Wire control chips such as the DS2406 and DS2409, several digital functions such as temperature sensors and analog-to-digital (ADCs) converters are available. These make it possible to measure a wide variety of physical properties over the 1-Wire net. A distinct advantage of 1-Wire instruments is that all interface to the master in the same manner regardless of the particular property being measuring. Whether the basic sensing element is voltage, current, resistive or capacitive based, they all communicate over the net using 1-Wire protocol. Other methods employ a variety of signal conditioning circuitry such as instrumentation amplifiers and voltage-to-frequency converters which of necessity makes their outputs different and may require separate cables for each sensor. The unique ID address or serial number of each device is the key for the bus master to interpret what parameter a particular 1-Wire instrument is measuring. Since the science of meteorology requires a variety of diverse sensors, several examples of 1-Wire instrumentation for use in a weather station will be given here. For example, the DS2423 Counter has inputs which respond to logic level changes or switch closures making them suitable to implement a variety of rate sensors. An example using magnetically actuated reed switches suitable for a rain gauge or wind speed sensor is shown in **Figure 23**.

Counting with the DS2423 1-Wire Counter

In the figure, the dual diode BAT54S serves both to protect the circuit from signals that go below ground, and with C1, to provide a local source of power. While the DS2423 has an internal pull-up resistor to keep the input from floating, its high value (≈ 22 Meg Ohms) can make it susceptible to noise. To prevent generating spurious counts during turn-on, and minimize noise pick-up, an external 1 Meg Ohm pull down resistor is used instead. Except for Lithium back-up (not shown), this is the counter circuit used in the 1-Wire rain gauge. In that application, a small permanent magnet moves past the reed switch each time a tipping bucket fills and empties. This momentarily closes the reed switch which increments the counter indicating .01 inch of rain has fallen.[1] A similar circuit is used in the 1-Wire weather station to measure wind speed.[2] The same circuit with Lithium backup has also been used as a hub mounted wheel odometer. Conveniently, the DS2423 also contains 4096 bits of user accessible SRAM, which is useful for temporary storage, or with Lithium back-up, for calibration, location and function information.

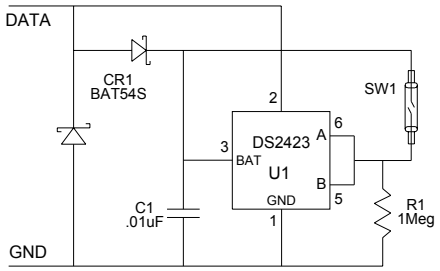


Figure 23 The basic DS2423 Counter circuit.

The DS2438, a versatile performer

Originally designed to perform multiple functions in a battery pack, the DS2438 contains two ADCs and a temperature sensor. The main ADC performs 10-bit conversion on a 0-10V input, or 9-bit conversion on a 0-5V signal, with an internal multiplexer that allows it to read the voltage on its power supply pin. The other converter was intended to measure the voltage developed by large currents flowing across an external .05 Ohm resistor with 10-bit accuracy at a full-scale reading of ± 250mV. The 13-bit internal temperature sensor is similar to the DS18B20. Additional features include a Real Time Clock and 40-bytes of nonvolatile memory useful for storing calibration, location and function information.

Measuring Humidity over the 1-Wire

Humidity is an important factor in many manufacturing operations as well as affecting personal comfort. With the proper sensing element, it can be measured over the 1-Wire net. The Honeywell sensing element specified here develops a linear voltage versus relative humidity (RH) output that is ratiometric to supply

voltage. That is, when the supply voltage varies, the sensor output voltage follows in direct proportion. This requires that the voltage across the sensor element as well as its output voltage be measured. In addition, calculation of True RH requires knowledge of the temperature at the sensing element. Because it contains all the necessary measurement functions to do the calculations, the DS2438 makes an ideal choice to construct a humidity sensor.

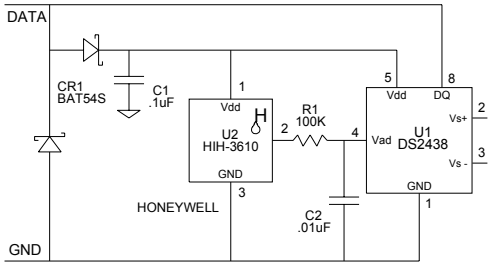


Figure 24 A humidity sensor using the DS2438.

In **Figure 24**, the analog output of the HIH-3610 humidity sensing element is converted to digital by the main ADC input of a DS2438. As in the DS2423 Counter circuit, a dual-diode BAT54S serves both to protect the circuit from signals that go below ground, and with C1, to provide a local source of power. Notice that C1 has been increased from .01uF to .1uF to handle the operating current required by U2, the HIH-3610. The RC network on the output of U2 is a low pass filter used to remove the low level clock feed-through from the sensing element's signal conditioning circuitry. However, if averaging is done in software, R1 and C2 are not necessay and may be omitted.

In operation, the bus master first has U1, the DS2438, report the supply voltage level on its Vdd pin, which is also the supply voltage for U2, the sensing element. Next, the master has U1 read the output voltage of U2 and report local temperature from its on-chip sensor. Finally, the master calculates true relative humidity from the three parameters supplied by U1.[3]

A 1-Wire Barometer Using the DS2438

Barometric pressure is another important meteorological parameter that can be measured over a 1-Wire net using the DS2438. By selecting a pressure sensor that contains comprehensive on-chip signal conditioning circuitry the circuit in **Figure 25** is very straightforward. As was the case with the humidity sensing element, the specified pressure sensing element is ratiometric. This requires that both the output voltage representing atmospheric pressure and the supply voltage across the element be known in order to accurately calculate barometric pressure.

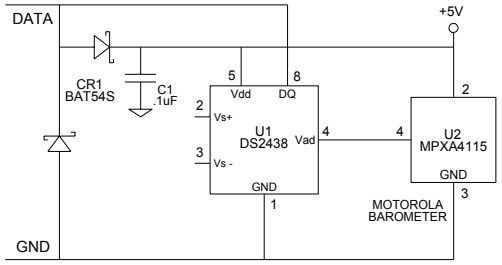


Figure 25 The 1-Wire barometric pressure sensor.

As is typical for 1-Wire instrumentation, a dual-diode BAT54S serves to protect the circuitry from signals that go below ground, and with C1, to provide a local source of power. In this case, because U2, the MPXA4115 pressure sensor can require as much as 10 mA at 5V, an external source of power is needed. Notice that the external power is also connected to the power pin of the DS2438. This allows the DS2438 to measure the supply voltage applied to the pressure sensing element. Alternatively, the barometer can be powered directly off the 1-Wire net by using concepts described in **Chapter 3**. However, in many installations supplying external power is not a problem as the barometer will be mounted inside near the bus master and a source of power. Flexible tubing can be routed to sample the outside air pressure and avoid unwanted pressure changes (noise) caused by the opening and closing of doors and windows or elevators moving inside the building.

A Wind Direction Sensor Using the DS2450

While the original 1-Wire weather station used a DS2401 Silicon Serial Number to label each of the eight magnetic reed switches in its wind direction sensor, a single D2450 Quad ADC can perform the same functions.[2] In keeping with recommendations for an ADI, a dual-diode BAT54S serves to protect the circuitry from signals that go below ground, and with C1, to provide a local source of power. Note that C1 has been increased from .01uF to 10uF to insure that the voltage across the resistor network remains relatively constant. As shown in **Figure 26**, a single DS2450 replaces the eight DS2401s originally used with five resistors. As the wind rotates the wind vane, a magnet mounted on a rotor that tracks it opens and closes one (or two) of the reed switches. When a reed switch closes, it changes the voltages seen on

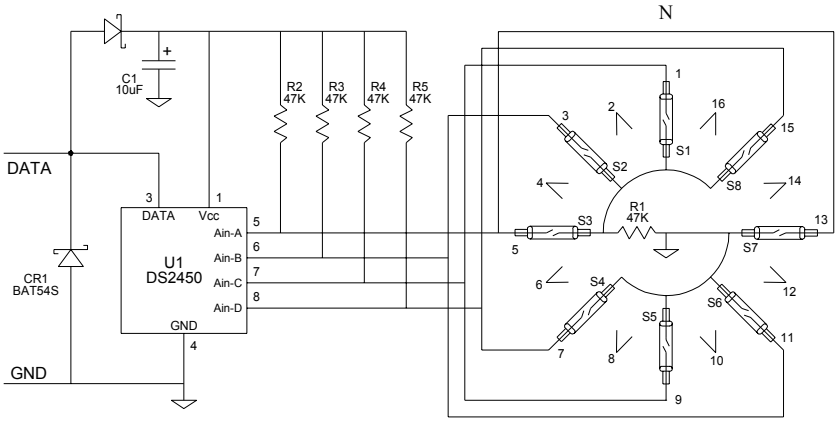


Figure 26 A wind direction sensor using the DS2450 Quad ADC.

the input pins of U1, the DS2450. For example, if the magnet is in a position to close S1 (North), the voltage seen on pin 7 changes from Vcc to 1/2Vcc, or approximately, from 5V to 2.5V. Since all sixteen positions of the wind vane produce unique four-bit signals from the ADC, there is no need to initialize the sensor, or store a tagging code on the board as was required by the original 1-Wire weather station. It is only necessary to indicate North, or equivalently, which direction the wind vane is pointing. **Table 1** lists the voltages seem at the ADC inputs for all sixteen cardinal points.

POS.	D	C	B	A
1	5	2.5	5	5
2	5	3.3	3.3	5
3	5	5	2.5	5
4	5	5	3.3	3.3
5	5	5	5	2.5
6	0	5	5	2.5
7	0	5	5	5
8	0	0	5	5
9	5	0	5	5
10	5	0	0	5
11	5	5	0	5
12	5	5	0	0
13	5	5	5	0
14	2.5	5	5	0
15	2.5	5	5	5
16	3.3	3.3	5	5

Table 1 Wind vane position versus voltage seen at the four DS2450 ADC inputs.

Because two reed switches are closed when the magnet is midway between them sixteen compass points are indicated with just eight reed switches. Referring to the schematic and position 2 in **Table 1**, observe that when S1 and S2 are closed 3.3 Volts is

applied to ADC inputs B and C. This occurs because pull-up resistors R2 and R3 are placed in parallel and the pair connected in series with R1 to form a voltage divider with .66Vcc across R1. Notice that this also occurs twice more at switch positions 4 and 16.

Measuring Solar Radiance on the 1-Wire

The amount of sunlight and its duration are additional parameters that meteorologists are interested in measuring. The ***amount*** is a measure of air and sky conditions, while ***duration*** is related to the equinoxes and the length of the day. While mounting and filtering tend to be complex, as shown in the following two figures, the electronics can be easily implemented using the DS2438. **Figure 27** illustrates a Solar radiance sensor using a photodiode, while **Figure 28** uses a photovoltaic cell. In each case a dual-diode BAT54S serves to protect the circuitry from signals that go below ground, and with C1, to provide a local source of power.

In **Figure 27** a sense resistor is connected in series with a photodiode and between the two 'current' ADC pins. Light striking the photodiode generates photo

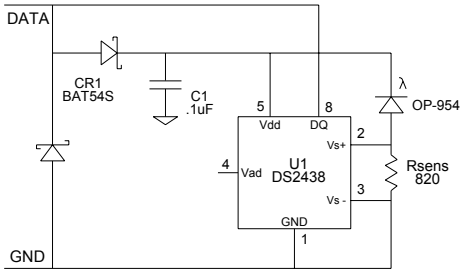


Figure 27 A Photodiode Solar radiance sensor.

currents that in turn develop a voltage drop across the sense resistor that is read by the ADC. In commercial units, optical filters are added to control both the wavelength and bandpass to which the sensor responds. More sophisticated units add such desirable features as a translucent hemisphere that collects light to enable the sensor to view the sky from horizon to horizon. In this case, the sensor actually focuses on the inside of the hemisphere to obtain its reading.

An interesting variation of a solar radiance sensor can be constructed using an LED in reverse bias mode. Select an LED that has acceptable current levels when exposed to the sun at high noon on a clear day. Size the resistor to develop 250 millivolts maximum using the formula R = E divided by I where E is 0.25 Volts and I is the maximum current generated. One example is the EFA5364X from Stanley. This is a super-bright orange ALGaInP LED with a peak response at 609nm and a narrow 15° spectral field of view. A 4.7K sense resistor provides acceptable outdoor performance, which may be increased to 100K if the circuit is only to be used with indoor lighting. LEDs made from other compounds will have their peak response in a different portion of the spectrum making them useful in specific applications.

Another approach to a solar radiance sensor is shown in **Figure 28** where a suitable solar cell is connected to the 'current' ADC inputs of the DS2438 through voltage divider R1 and R2. The divider is necessary to ensure that the voltage between pins 2 and 3 of the DS2438 does not exceed its maximum limit of 300 millivolts. Chose resistor values for the divider so as to not unduly load the power capacity of the cell. C2 and R3 form a low-pass filter to reduce noise sensitivity. One advantage of this technique is the ability to use several cells facing toward different

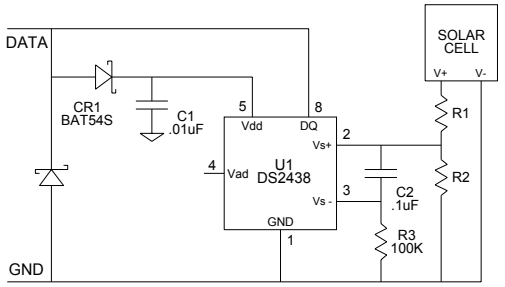


Figure 28 A Solar radiance sensor using a photovoltaic cell.

sectors of the sky to obtain horizon to horizon coverage. Connect the cells in parallel and size R2 to develop 0.25 Volts with maximum sun light across the sensor as described in the preceding paragraph.

MEASURING A THERMOCOUPLE WITH 1-WIRE

It is also posssible to measure extreme temperatures using conventional thermocouples that are directly digitized at the cold junction using a DS2760 multi-

function 1-Wire chip. The twisted pair cable of the 1-Wire net serves to cover the distance between the (TC) and bus master effectively replacing the expensive thermocouple (TC) extension cable normally used. Because of its unique ID address multiple smart thermocouples may be placed where needed anywhere along the net greatly minimizing the positioning and cost of an installation. Although information associated with the TC may be stored within the chip itself ("tagging"), this same ID allows reference data to be stored at the bus master. [4]

REVIEWING THE THERMOCOUPLE

The fundamental operating principle of the thermocouple was discovered in 1821, when Thomas Seebeck discovered that if two dissimilar metals were joined at one end a voltage (the Seebeck Voltage) proportional to the temperature difference between the joined and open ends was generated. Two of the more popular industry standards are the type K and type E. By convention capital letters are used to indicate composition according to American National Standards Institute (ANSI) conventions. For example, type E thermocouples use nickel-chromium as one conductor and constantan (a copper-nickel alloy) as the other. While the full-scale output voltage of all TCs falls in the low millivolt range, the type E generates the highest Seebeck Voltage/°C (62μV/°C @20°C) resulting in an output of almost 80 millivolts at 900°C; more than any other standard. Obviously, in order to measure this output voltage it is necessary to make connections to the open ends of the wires forming the thermocouple. These connections in turn form a second thermocouple, for example nickel-chromium/copper in series with the original or 'hot' junction when copper conductors are used. Historically to correct for these 'cold' junctions (one for each TC wire) they were placed in an ice bath at the triple point, whereas most modern instruments electronically correct the reading to zero degrees.

When electronic correction is used, the temperature at the cold junction is measured and the voltage that would be generated by the thermocouple at that temperature is subtracted from the actual reading. If the voltage versus temperature transfer function of the thermocouple were highly linear this would be all that was necessary to correct the reading. Unfortunately, since the Seebeck Voltage /°C varies with temperature, the full-scale transfer function is usually fairly complex which can require several piece-wise approximations to maintain a specified accuracy depending upon the temperature range of interest. In this respect, the type K TC with its lower Seebeck Voltage (51μV/°C @20°C) has an advantage over the type E as it is significantly more linear over the 0°C to 1000°C range. For in-depth information on thermocouples, check the reference material available on the web by manufacturers such as Omega Engineering Inc.[5]

While there are obvious variations, a typical modern electronic thermocouple consists of several basic building blocks. As illustrated in **Figure 29**, these blocks consist of a TC with secondary temperature sensor to measure the junction where thermocouple and connecting wires join; a signal conditioning block and an analog-to-digital converter (ADC). Usually, the thermocouple is connected to a precision low noise or instrumentation amplifier, which provides the gain, offset and impedance adjustments necessary to match the low level signal generated by the TC to the input of a multi-bit ADC. The ADC in turn converts the conditioned signal from the amplifier into a digital format that is sent to a microprocessor or PC. From the ADC and cold junction sensor inputs, the μP (or PC) computes the actual temperature seen at the hot junction of the thermocouple. Some custom conditioning chips such as the MAX6675 from Maxim or the AD594 from Analog Devices are available that contain both the instrumentation amplifier and the cold junction compensation circuitry for a particular TC type such as the J or K in one IC. These chips replace the first two blocks and plug directly into an ADC input.

THE DS2760

Originally designed to monitor a Lithium-Ion battery pack, the DS2760 provides several new capabilities to transform a simple thermocouple into a smart sensor.[6] The chip can directly digitize the millivolt level output produced between the hot and cold junctions of the thermocouple, while it's on-chip temperature sensor continuously monitors the temperature at the cold junction of the TC. With its unique ID address it provides a label that permits

multiple units to operate on the same twisted pair cable. And it contains user accessible memory for

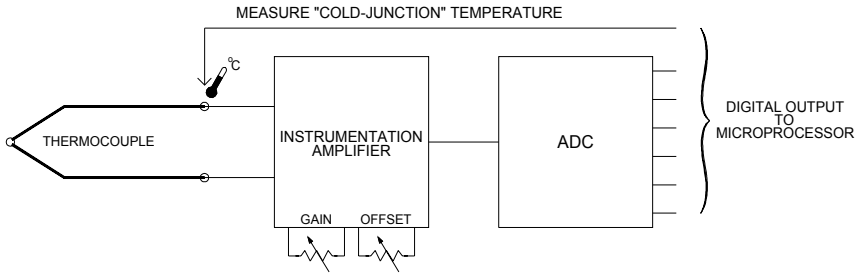


Figure 29 A typical electronically compensated thermocouple consists of these three building blocks.

storage of sensor specific data such as TC type, location and the date it was placed into service. This allows a DS2760 to be used with any TC type as the bus master uses the stored data to determine the correct calculations to make based on the type TC in use and the temperature of the cold junction as reported by the on-chip temperature sensor.

As a complete signal conditioning and digitizing solution for use with a thermocouple, the DS2760 contains a 10-bit voltage ADC input, a 13-bit temperature ADC and a 12-bit plus sign current ADC. It also provides 32-byte of lockable EEPROM memory where pertinent user or sensor documentation may be stored which can minimize the probability of error due to the mislabeling of sensors. In the present application, the thermocouple is directly connected to the current ADC inputs that were originally designed to read the voltage drop developed across a 25 milli-Ohm resistor as a Lithium-Ion battery pack is charged and discharged. With a full scale range of ± 64 millivolts (LSB of 15.625μV) the converter provides resolution exceeding one degree C even with the lower output of a Type K thermocouple.

THE 1-WIRE THERMOCOUPLE

The schematic in **Figure 30** illustrates both the simplicity and ease with which a DS2760 can be used to convert a standard thermocouple into a smart sensor with multi-drop capability. In the circuit, C1 and one of the Schottky diodes in CR1 form a half wave rectifier that provides power for the DS2760 by 'stealing' it from the bus during idle communication periods when the bus is at 5V. This is a discrete implementation of the parasite power technique used internally by 1-Wire devices to provide their own operating power. The remaining Schottky diode in the package is connected across DATA and GND and provides circuit protection by restricting signal excursions that go below ground to about minus four tenths of a volt. Without this diode, negative signal excursions on the bus in excess of six tenths of a volt forward bias the parasitic substrate diode of the DS2760 chip and interfere with the proper functioning of the chip. Under bus master control U1 the DS2760, monitors the voltage developed between the hot and cold junctions of the thermocouple as well as measuring the temperature of the cold junction with its internal temperature sensor. The master uses this information to calculate the actual temperature at the hot junction of the TC. By adding the optional resistor (R1), Vdd may also be measured. This can be useful in trouble-shooting to verify that the voltage available on the 1-Wire net is within acceptable limits.

When mounting the thermocouple to the board, it should be connected as close to the DS2760 as practical so minimal temperature difference exists between these connections and the chip inside the DS2760 package. To maintain the junctions at the same temperature use copper pour and lead placement to create an isothermal area in and around the point where the thermocouple leads attach to the copper traces of the PCB. Keeping in mind that temperature differentials generate voltage differentials over their entire length, route the PCB

traces together and maintain equal numbers of junctions on each conductor.

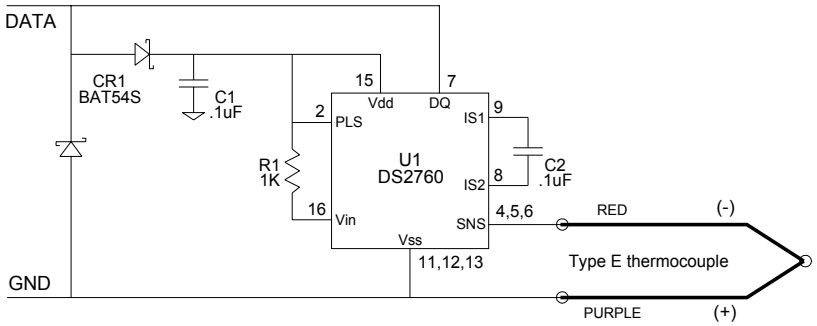


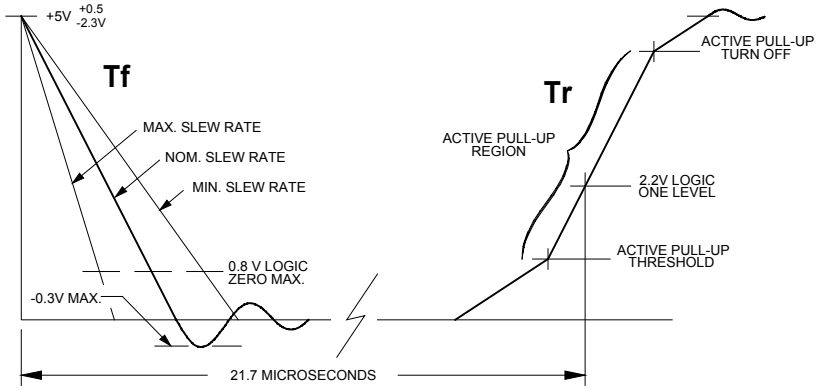
Figure 30 Using a DS2760 to read a thermocouple on the 1-Wire net.

1-Wire®, TMEX AND 1-Wire net are trademarks of Dallas Semiconductor.

REFERENCE

1. Dan Awtrey. Dec 1999. “A 1-Wire Rain Gauge,” ***Sensors***:56-59.
2. Dan Awtrey. Jun 1998. “The 1-Wire Weather Station,” ***Sensors***:34-40.
3. Dan Awtrey. Aug 2000. “A 1-Wire Humidity Sensor,” ***Sensors***:62-63.
4. Tagging protocol may be downloaded at: ftp://ftp.dalsemi.com/pub/auto_id/public/xmltag.pdf
5. “Using Thermocouples” www.omega.com/temperature/Z/pdf/z021-032.pdf
6. DS2760 data sheet. http://pdfserv.maxim-ic.com/arpdf/DS2760.pdf.

APPENDIX A
Waveform template for a 1-Wire net exceeding 100 meters



This section defines the 1-Wire template for use with bus lengths exceeding 100 meters. Notice that the zero to one transition is shown with the characteristic waveform for an active pull-up as the use of a passive pull-up resistor can be problematic with cables of this length. A 5V pull-up supply is recommended, but can range from 3 to 6 volts.

The falling edge (Tf)

Because it controls all system timing, the falling edge of the waveform from the bus master should be monotonic. To prevent signal disruptions caused by unterminated transmission line effects, the slew rate must be controlled to be much longer than twice the electrical length of the cable. This works out to be about 1.1volts per microsecond (V/µS) for the given conditions. As shown in the template, this requires 4 microseconds to cross the 0.8V logic zero threshold level. This slew rate provides acceptable performance with bus lengths up to 300 meters. It performs well over that range with 1-Wire loading varying from one to 500 devices.

For net communication to be successful, the bus must be raised to a voltage greater than the 2.2V logic one level before the master samples. Beginning with TMEX v3.0, sampling occurs 21.7µS after the master pulls the bus low. If the bus pull-up fails to raise the voltage above the logic one threshold before this sampling occurs, the master will always see a logic zero and conclude the bus is shorted, and communication can not occur.

Negative Undershoot

Signal excursions below ground must be clamped to less than 0.6V to prevent turning on parasitic diodes in the substrate.

The rising edge (Tr)

As the number of 1-Wire devices on the bus grows, the time required to raise the line above the logic one threshold increases. This also occurs as the network is lengthened due to the 50pF of capacitance added per meter of twisted pair cable. Because of these effects, for a 1-Wire net of 100 meters or more, an active pull-up must be used. Obviously, an active pull-up circuit should be on only during a defined range of the rising edge (zero to one transition). Conversely, it should not respond on the falling edge, nor be active during logic zero time intervals. It must trigger on the rising edge at about 0.9V plus or minus 0.1V to provide acceptable noise margin. Preferably, once triggered it will remain on until the line is raised above a specified threshold (≥3V) rather than for a set time interval (one-shot). This insures that the data line will be raised above the 2.8V level required to recharge the parasite power capacitors regardless of load. The maximum current the circuit can supply should be limited to about

15mA. Larger currents flowing in cable inductance can cause problems.

Consult the Dallas Semiconductor iButton website at **www.ibutton.com** for current product information, application notes and data sheets.