Dallas 1-Wire / MicroLan in a nutshell.

Table of Contents

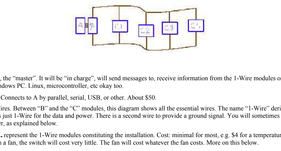
1. Introduction

You should derive from this document an overview of the usefulness and functioning of the MicroLan products from Dallas Semiconductor. It tries to explain what they are, what they can do, how to make them do their tricks. The trademark "1-Wire" also used for the product line.

The document is written by an electronic hobbyist, not employed by Dallas. It has not been endorsed by Dallas. For more information, go to http://sheepdogsoftware.co.uk

2. Overview

A useful MicroLan installation will look something like the following...



A: A computer, the "master". It will be "in charge", will send messages to, receive information from the 1-Wire modules on the system. Typically a Windows PC. Linux, microcontroller, etc okay too.

B: An adapter. Connects to A by parallel, serial, USB, or other. About $50.

Brown line: Wires. Between "B" and the "C" modules, this diagram shows all the essential wires. The name "1-Wire" derives from the fact that there is just 1-Wire for the data and power. There is a second wire to provide a ground signal. You will sometimes supply additional power, as explained below.

C1, C2, C3.... represent the 1-Wire modules constituting the installation. Cost: minimal for most, e.g. $4 for a temperature sensor. If, say, C2 turns on a fan, the switch will cost very little. The fan will cost whatever the fan costs. More on this below.

What's it Good For? Don't let the following very simple illustration fool you. MicroLans are capable of much more sophisticated systems, larger systems! But, as an example:

A security system could me made with nothing more than an old PC, a suitable program, and the following MicroLan modules: a light sensor, a sensor wired to a simple toggle switch, a sensor wired to a motion detector, an output wired to a bell, and an output wired to a floodlight. The installation could include the following features:

When the switch was 'off', if it was dark and the motion sensor was triggered, the floodlight would come on for 3 minutes.

When the switch was 'on', the floodlight would work as before. In addition, if the motion sensor was triggered three times, each at least a minute after the previous one which counted, and all within five minutes, then the bell would ring for ten minutes.

Forgive me, but I can't resist a small digression to explain a simple- to- achieve, remarkable for the increase in security resulting embellishment that is possible within the 1-Wire product line. The toggle switch could be replace with a small clip (just a mechanical part). If you wanted to turn the alarm system off, you would insert into that clip an iButton... they look like the flat disc batteries used in watches. Each has a unique serial number, and you could set your alarm system to recognize only certain serial numbers. You could even set the system up to allow someone with the boss's iButton to access the site at any time, but someone with the gardener's iButton to access the site only between 8am and 6pm. Even if you knew the serial number of the boss's iButton, it would be very, very difficult to "forge" a copy. You can't just reprogram some other iButton to have the same serial number. Forgive the digression?

Sensors (Inputs). There are MicroLan chips which can sense digital inputs. There are analog to digital converters (ADCs). Because there are MicroLan ADCs, you can interface sensors for almost any physical parameter to a MicroLan installation.

Actuators (Outputs): There are MicroLan chips which can be used to turn things on and off.

"Memory":Storage: There are MicroLan chips with EEPROM which can be written to or read from. You will often see them described as "memory", but it might be argued that "storage" would be term for their role in any MicroLan.

See http://www.arunet.co.uk/tkboyd/e1dbw.htm for links to various suppliers.

3. Hardware detail

We'll look at the 1-Wire modules first, then consider the master, then the adapter.

3a: Modules

MicroLans may have one or more temperature sensors. In a simple configuration, a temperature sensing "module" would consist merely of a small blob of plastic with three legs. One would join to the data wire of the MicroLan and the other two would connect to the other wire. (For the nervous among you, let me add that the "not simple" configuration merely adds a voltage source connected to one of the wires otherwise connected to the 1-Wire ground.)

3b: Master

Typically the master is an ordinary PC. It doesn't have to be a $1800 Windows XP Monster Machine. I have an old laptop running Windows 95 reliably logging weather data with a MicroLan. I paid $40 for it. You can also use a Linux PC, or a microcontroller. The TINI might be of interest to some readers... http://www.maxim-ic.com/TINIplatform.cfm.

A MicroLan without a master is a chicken without a head.

The modules attached to the MicroLan can do the things described in the previous section, but you need the master to "conduct the orchestra", to read sensors (and maybe storage devices) at the appropriate time, record that data (if need be- either to the master's storage, or to storage in modules on the MicroLan), send appropriate messages to the actuators.

3c: Adapter

It isn't a complicated device, as far any user need be concerned... but you do need one. There are different models for different needs. I'd be inclined to buy the one that plugs into a serial port, the DS9097U-009, or a similar product, not from Dallas, but from ww.ibuttonlink.com. At 2/05, their price for one without an address chip (unnecessary, in my view... but something you might as well have, if "free"., as it used to seem to be in the DS9097 line) was $39 (plus P&P, taxes, I suspect.)

There are also adapters for the parallel port, and for USB. If your master is a TINI, it has an adapter built into it, so you have nothing extra to buy.

Once the adapter is plugged into the master, you plug the rest of the MicroLan into the only remaining socket on the adapter. The middle two contacts of the RJ-11 or RJ-45 will be for

i) 1-Wire data+power, and
ii) the 1-Wire ground signal.

Beware: Some adapters and some units for plugging into MicroLans from third party sources use other contacts in the RJ-45 or RJ-11 for other purposes... some of which are *incompatible*. You can get around these issues quite easily, but you do **need** to be **careful**. Plugging things together "to see if it works" will sometimes result in **permanent damage**.

In some of my research for this application note, I found mention that only a DS9097U-S25 was capable of programming MicroLan eprom devices. I **think** this refers to "old fashioned" eproms, and that any of the adapters you would buy today are capable of programming the modern eeprom (Z's) devices, e.g. a DS2433, or the iButton version, DS1973. If anyone at Dallas could confirm this, I'd be delighted to revise this note. I started wondering while reading www.maxim-ic.com/quick_view2.cfm/qv_pk/2983.

4. Software detail

Any computer which is going to be the master for a MicroLan needs to have some drivers installed. The process is not complicated. The drivers are free. You get them, via the internet, from Dallas.

If you are going to write programs to manage MicroLans, you need to install certain libraries on your development PC. You would also be advised to install the documentation provided as part of the SDK. Again, it is all available from Dallas, via the internet, and free.

I've never traced a problem with any system to the presence of the drivers or the development tools, nor have I detected any unwelcome, unnecessary things like hooks forcing me to check for updates, etc.

4a: User's view of what is happening

If, for instance, you purchased the security system described in the "What's it Good for" paragraph of the Overview, above, then you wouldn't need to know anything about what the MicroLan was up to, any more than when you add a printer to your computer you need to know what the printer driver is doing. It Just Works. (Until Windows crashes, though no fault of the MicroLan.)

The security system could have a user interface showing the state of the various sensors and actuators, showing a log of recent events (e.g. "motion detected"). There could be a "virtual switch" to turn the light on or off, bell on or off, etc. Still... user would have no need of understanding the details of how the information flowed across the MicroLan.

If you want to go one step beyond a "plug and play" package, you can set up MicroLan systems, say, to monitor humidity and temperature in various parts of a warehouse. Someone *could* write a super duper program which would allow you to simply plug in (or remove) sensors, and the software would automatically take note of the sensors in the network... but if you have that type of application in mind, you'll have a wider choice of software to monitor your hardware if you can accept the need to tinker a little with an ini file... no big deal. You can use a wordprocessor and tie your shoes (not at the same time), and if the software manual is decent, then you'll be able to manage the ini file.

But the real fun comes when you learn to program. Assuming you have the time. More on this in the next section.

4c: How to write programs for MicroLans

Dallas provides extensive documentation on how the 1-Wire chips work, and on using the software development kit (SDK). Dallas give plenty of examples, in a variety of languages, including the one I like to work in, Delphi.

In theory, you can drive a MicroLan from a mere DOS PC, but there's one little gap in my understanding of what is involved which I've never pursued in a clean, and there doesn't seem to be anyone out there who remembers the DOS days to help me out. Pity... I think a simple DOS based machine would be cheap and reliable. You can, however, also run MicroLans from a Linux box, so perhaps that's the way to go if you want cheap and reliable.

For an extensive collection of Delphi tutorials, visit http://sheepdogsudes.com/tut.htm It includes "getting started with Delphi" tutorials, and, at http://sheepdogsudes.com/dtl1main.htm, tutorials on interacting with specific 1-Wire devices.

4c i: Overview of what happens when a 1-Wire program is running

Before we go any further, you need to know that each 1-Wire chip has a unique ID number built into it. Thus, a MicroLan with 5 temperature sensing chips can work, because the master has a way of knowing which temperature sensor, among the five on the MicroLan, is the one saying "The temperature here is..."

The following tries to convey an idea of typical operation using only simple functions. There are "clever" things you can do too, but for now....

After the PC and MicroLan have settled down after booting, everyone is standing by, waiting for something to happen.

At some point determined by the software, the master will begin a communication with the MicroLan. The first thing the master will do will be to send out messages which "say" to all of the chips on the MicroLan, *except for one*, "Disregard-*almost* everything until further notice." The one thing that all the chips will continue to "watch" for is a message saying "Okay- now wake up and take notice again."

The one chip which has *not* been "put to sleep" continues to pay attention to things "said" by the master. Because the master knows which chip it is speaking to, it knows what things "mean" something to that chip. The further "conversation" might consist of instructions to change the state of an output, or to read and report the chip's temperature. (The chips in the MicroLan can "answer" "questions" from the master... but they are subservient. They (generally, in simple scenarios anyway) "speak only when spoken to."

When the master is finished with the chip it was "speaking to", and has had any "answers" requested, it sends the message that "wakes up" all of the other chips. At this point, everything can go back to standing by until the program has further need of "talking" to the chips in the MicroLan.

4c ii: Details of how to write code to make MicroLans work

Think for a moment about how your internet browser works. It is a program, running on your PC. "Underneath the surface", as it were, if you want to look at some page from the internet, you use at least *two* programs: Your browser, and your connection to the internet, e.g. Winsock. Many of us have our browser set up to automatically start and stop Winsock as needed... but the two are only "joined" for convenience. You can run either without the other. (Of course the browser will only access local pages while Winsock is off.)

Using a MicroLan is similar in some ways, although the "Winsock" element is always started and stopped by your program.

The software development toolkit (SDK) tells you about the calls you can use within your program. Those calls "talk" to the program that Dallas (Winsock parallel). That in turn sends and receives messages to/from the adapter, which is the master's interface to the MicroLan.

Whew! Hard to describe... but it really does all work quite transparently in practice.

In Delphi (and it won't be much different in other languages), you add to your **uses** clause a reference to a unit Dallas provides. (IBTMEXPW.PAS). You do this to "add words" to your programming language. The unit you add will depend on what language you are using.

Note: The IBTMEXPW.PAS (or equivalent) file does not have to be present of machines which will run your program. It is only needed on the machine *compiling* your program. You do need to have the "TMEX.run time package" installed on any computer which will run your MicroLan program. The run time package contains the functions and procedures that IBTMEXPW.PASS makes reference to. "TMEX" is the name for this collection of subroutines.

You will need to set up some variables to enable you to interact with the functions and procedeures of TMEX... but that's not hard.

Not only does TMEX give you the things you need to talk to the MicroLan, it also gives you what you need to talk to TMEX, and the "Winsock-like" "thing" that operates alongside your program.!

Onward! Now we turn to what code to put into your program.

Somewhere, somehow within your program, you need to learn the serial numbers, a.k.a. IDs, of the chips in your MicroLan. The simplest programs merely read this information from a human created ini file. Likewise, the program needs to know what sort of adapter you are using, and where it is connected. Still no big deal.

Now then... Continuing to use my browser/Winsock metaphor, before you can use the heart of TMEX, you must establish communication with the equivalent of Winsock. Your program talks to TMEX, TMEX talks to the MicroLan. Answers back from the MicroLan also pass through TMEX. This is good. As new machines come along, all Dallas have to do is create new TMEX-machine code. All of the developer created program-TMEX stuff stays the same, and something that worked on a Windows machine will work on a WhoKnowsWhatsNext machine, just as soon as Dallas provide the new run time package.

Don't be confused by the fact that in some Dallas datasheets, the Memory Function Commands are presented before the ROM Function Commands, even though you use at least one of the latter before using any of the former!

Finally in a single iteration of "do something with something on the MicroLan" you should send the "everyone wake up again" signal so that the system is ready for your next attempt to do something.

Conclusion:

I hope that has answered questions you may have about "Is MicroLan for me?"; "Could I make a MicroLan do things?". As a hobbyist, I've had a lot of fun with my MicroLan projects. The product does allow you to accomplish many things which you might not suspect could be done without investing far more time and or money.

For further information, visit **http://www.arunet.co.uk/tkboyd/eldidx.htm**
If that's down, try **http://sheepdogsoftware.co.uk/**
You can learn how to email me from either page.