A Versatile LabVIEW™ Environment for Communicating with Dallas-Maxim 1-Wire™ Devices

Dale H. Litwhiler
Penn State Berks-Lehigh Valley College

Abstract

This paper presents a set of software tools using LabVIEW to communicate with the 1-Wire family of integrated circuit devices produced by Dallas-Maxim. These tools have many applications in engineering and science education where a simple computer interface is needed to make some physical measurements and/or control some physical actions. LabVIEW software is well known in the field and provides an excellent graphical programming environment that runs on both IBM-Compatible (PC) and Macintosh (Mac) computer platforms.[1] The 1-Wire family of devices includes temperature sensors, A/D converters, digital I/O, counters, and digital potentiometers. Multiple devices can be simultaneously connected to the host computer using only two-conductor wire (one wire for data/power and one wire for ground). Each device contains a unique identification number that allows it to be individually addressed and controlled.[2] The software tools presented here are completely self-contained within LabVIEW with no external code or libraries. This allows them to be used with any LabVIEW version (tested on 3.1 through 6.1 and student 6i) or platform (PC or Mac). The software is also structured such that many 1-Wire device applications can be run simultaneously without collision. An overview of the hardware is provided and examples of communicating with the devices using LabVIEW are given.

Introduction

In engineering and science education, there are many instances where a computer can be used to make physical measurements then analyze and display the results. These measurements might then be used to control certain actions and processes. In many cases, the number of parameters measured and/or controlled is very few and does not justify the use of an expensive data acquisition system. There are also applications where automated measurement and control capability can be embedded into existing equipment. In these circumstances, the 1-Wire family of devices produced by Dallas-Maxim provides a very attractive solution.

The 1-Wire family of integrated circuit devices include temperature sensors, analog-to-digital (A/D) converters, digital input/output (I/O) devices, event counters, and digital potentiometers. Communication data as well as device power is transmitted via a two conductor connection. One conductor carries the communication signal and power while the other conductor is the ground connection (thus the name 1-Wire). Many devices can be connected to the host computer using the same pair of conductors to form a bus, or MicroLAN.[3] Each device has a unique factory-programmed serial identification number. This number is used to address and communicate with individual devices. Each device also includes a Cyclic Redundancy Check (CRC) generator to help ensure data integrity.

All communication on the MicroLAN is controlled by the bus master which is typically the host processor. The master pulls up the bus to a nominal +5V through a resistance of a few thousand ohms. This pulled-up “high” state is the quiescent state of the bus. All signaling is performed by pulling the bus low for specified durations (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. A system clock is not required as each 1-Wire device is self clocked by its own

internal oscillator. The master can also perform a bus reset by holding the bus low for at least 480 microseconds.[4]

Power for each 1-Wire device is obtained by "stealing" it from the data signal. Each device contains a half-wave rectifier circuit. Whenever the data line is pulled high by the bus, the diode in the half-wave rectifier circuit turns on and charges an internal 800pF capacitor. The energy stored on this capacitor powers the device through the times when the bus is pulled low. This form of powering the 1-Wire devices is referred to as "parasite power."[4]

Figure 1 shows a typical MicroLAN layout. The bus master (computer) and slave devices can be connected anywhere along the two-conductor bus. Wiring between the master and slave devices can be done with simple telephone wire for short runs (<30m). For longer runs (up to 300m), category 5 (CAT5) data communication wire is recommended.[4]

The MicroLAN communication signaling protocol was designed to allow bus control via a single pin of a microcontroller. The required bit writing and reading time slots are not as easily handled on a personal computer (PC or Mac). To allow the MicroLAN to be controlled using a personal computer, Dallas-Maxim offers the DS9097U Universal 1-Wire COM port Adapter. Using this adapter, the host software simply writes and reads bytes to and from the computer's serial port. The DS9097U controls the MicroLAN timing, slew rates, and many other advanced functions that are beyond the scope of this paper. Figure 2 shows an electrical schematic and assembly drawing of the DS9097U adapter.

1-Wire Serial Driver

The philosophy behind the software is to create a programming environment in which commands obtained from device datasheets can be easily translated into useable applications. LabVIEW was chosen because of its ease of application development, familiarity among students, and the ability to run its student version on both PC and Mac computers. LabVIEW also contains the serial port routines necessary to communicate with the DS9097U adapter.

Because it is possible to have many types of devices simultaneously connected to the MicroLAN, the software must be capable of running several applications at once without interference. To accomplish this, the following methodology was used: One virtual instrument (VI) called "1-Wire Serial Driver" was developed through which all serial port communication must pass. A command syntax was designed such that text scripts could be written and passed to the 1-Wire Serial Driver

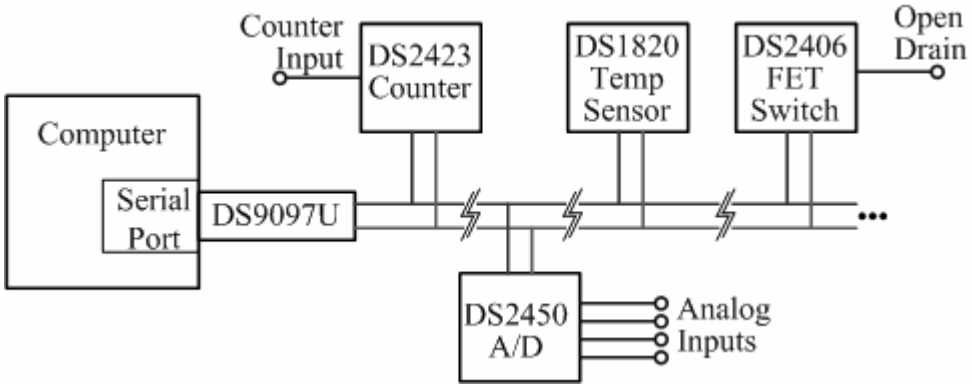


Figure 1. Typical MicroLAN Layout

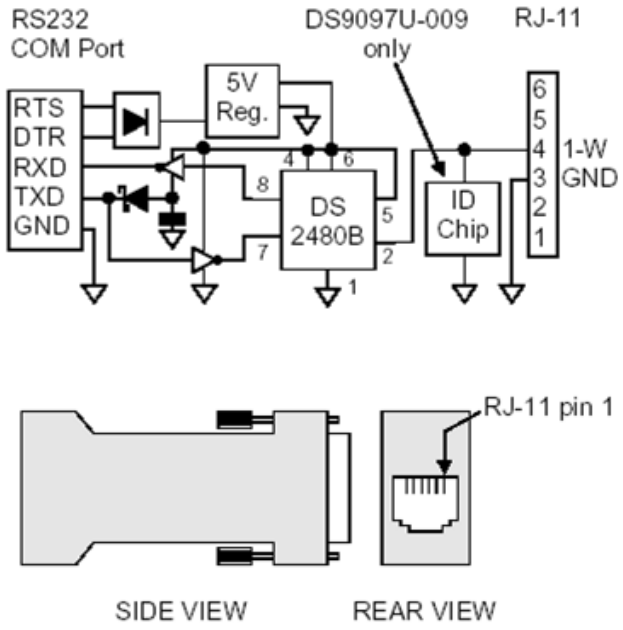


Figure 2. DS9097U Schematic and Assembly Drawing

VI for processing. The command syntax allows packets of concatenated command strings to be formed and sent to the serial port to help reduce latency in the communication process. The scripts contain the device commands as well as details about waiting for bus responses and inserting delays to wait for device operations to complete. The entire script for a particular device is executed without interruption. In this way, many devices can be "Served" without danger of fragmented command/response sets.

The DS9097U adapter contains the DS2480B serial to 1-Wire bridge device. The DS2480B converts serial port bytes to 1-Wire bus data and vice versa. The DS2480B contains a one-byte buffer to store incoming bytes from the serial port. Care must be taken not to send bytes to the DS2480B too quickly or data will be lost in buffer overflow. At 9600 baud, buffer overflow is not a concern unless a bus reset command is issued. When a bus reset command is issued, the software must wait for a response from the DS2480B before more commands can be sent. A reset command is therefore a final byte in many of the command string packets. While most command bytes have corresponding response bytes, some commands do not. These conditions must also be included in the syntax. Proper handling of the response bytes is critical because the CRC error checking algorithm depends upon the sequence of responses. The syntax for a single script command line is as follows:

<command byte>,<response expected?>,<wait for response?>,<flush?>,<delay after command>

The command byte is a hexadecimal command from the device data sheets. If a response is expected for that particular command, a 1 is placed after the comma else a 0 is used. If the software should wait for the response before continuing, a 1 is placed after the next comma. If the software should read and discard (flush) the contents of the response array, a 1 is placed after the next comma. Finally, if the software should wait to allow a device operation to complete after sending the command, the delay duration in milliseconds (in hexadecimal) is placed after the last comma. When a packet of strings is written to the serial port, the computer tick counter value is read and added to an array. These tick counter values may then be used by the calling VI for timing calculations. Figure 3 shows the front panel of the 1-Wire Serial Driver VI. The 1-Wire Serial Driver VI diagram is shown for reference in Appendix A.

The DS2480B has two primary modes of operation; command mode, and data mode. In command mode, the DS2480B interprets all incoming bytes from the serial port as commands for itself. In data mode, incoming bytes are passed on to the 1-Wire bus as commands for other devices. Switching between modes is handled by reserved hex codes E1 and E3.

Table 1 shows a script of command strings for obtaining a temperature reading from a DS1820 temperature sensor. A detailed explanation of each line is also given. This script is passed as input to the 1-Wire Serial Driver VI. Each line of the script is parsed by the VI to determine

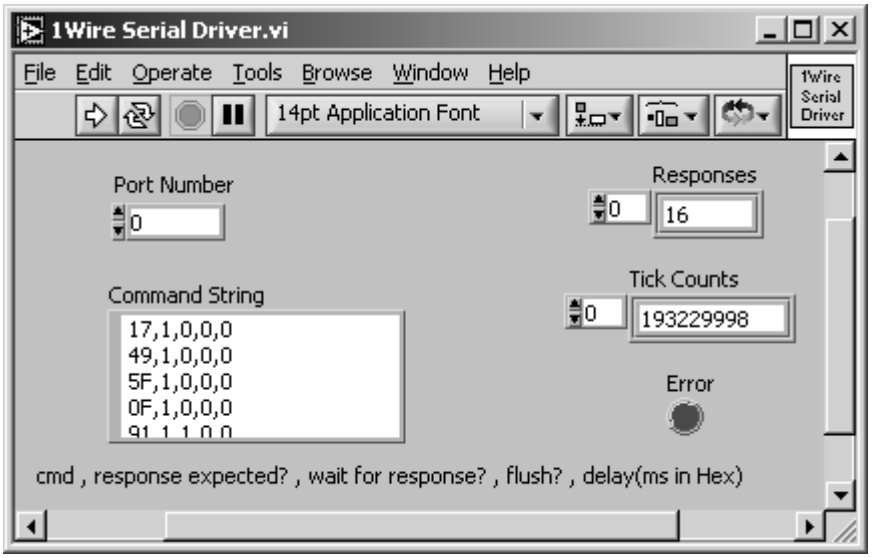


Figure 3. 1-Wire Serial Driver VI Front Panel

what byte to send to theserial port and what to do after the byte is sent. By using this script language, 1-Wire device applications can more easily be developed in LabVIEW.

As shown in Table 1, the script begins with a 1-Wire bus reset. This tells all of the devices on the bus to get ready for a new round of commands. Next, the DS2480B is switched to data mode to allow commands to be sent out directly to the 1-Wire bus. The 55h "Match ROM" command tells the devices on the bus to check the bytes that follow for a match of their own unique ROM ID number. Next the eight bytes of the ROM ID are sent. After these eight bytes are sent, there will be only one device still listening on the bus. The 44h "Convert Temperature" command tells the remaining device (DS1820 temperature sensor) to start a temperature conversion procedure. This process takes about 750ms to complete during which no other 1-Wire bus activity is allowed as the DS1820 needs to be continuously powered from the bus high state. A delay of 1000ms is used to ensure that enough time is allowed for the temperature conversion to complete. The DS1820 places the result of the measurement into its scratchpad memory area. To retrieve the measurement, the bus must be reset and the device match ROM ID procedure repeated. The BEh "Read Scratchpad" command then prepares the device to transmit the contents of its scratch pad memory over the bus. This is done by sending one FFh read byte for each desired scratchpad byte starting with byte zero. The response bytes received for each FFh byte sent correspond to the data for that scratchpad location. All kept response bytes are passed as output from the 1-Wire Serial Driver VI as an array. This array contains one byte for each command that has a response that was issued after the last flush. In this example, the response byte array contains 20 elements. The appropriate elements must then be selected and processed to determine the temperature. (Element selection and processing can be seen in the LabVIEW diagram shown later in Figure 4.)

1-Wire Device Virtual Instruments

Virtual instruments have been developed for many of the devices in the 1-Wire family. Each of these VIs contain scripts written with device-specific algorithms to perform particular functions for that device as shown in Table 1 for the DS1820 temperature sensor. Various front panel control inputs are used and properly formatted into the script syntax to be sent to the 1-Wire Serial Driver VI. The response bytes from the 1-Wire Serial Driver VI are then decoded to extract the desired information.

Command Script String	Command Line Function	Response Expected?	Wait for Response?	Flush?	Delay
C5,1,1,1,0	Reset 1-Wire Bus	Yes	Yes	Yes	0
E1,0,0,0,0	Switch to Data Mode	No	No	No	0
55,1,0,0,0	Match ROM ID	Yes	No	No	0
10,1,0,0,0	ROM ID byte 0	Yes	No	No	0
A8,1,0,0,0	ROM ID byte 1	Yes	No	No	0
24,1,0,0,0	ROM ID byte 2	Yes	No	No	0
05,1,0,0,0	ROM ID byte 3	Yes	No	No	0
00,1,0,0,0	ROM ID byte 4	Yes	No	No	0
08,1,0,0,0	ROM ID byte 5	Yes	No	No	0
00,1,0,0,0	ROM ID byte 6	Yes	No	No	0
16,1,0,0,0	ROM ID byte 7	Yes	No	No	0
44,1,0,0,3E8	Convert Temperature	Yes	No	No	1000ms
E3,0,0,0,0	Switch to Command Mode	No	No	No	0
C5,1,1,1,0	Reset 1-Wire Bus	Yes	Yes	Yes	0
E1,0,0,0,0	Switch to Data Mode	No	No	No	0
55,1,0,0,0	Match ROM ID	Yes	No	No	0
10,1,0,0,0	ROM ID byte 0	Yes	No	No	0
A8,1,0,0,0	ROM ID byte 1	Yes	No	No	0
24,1,0,0,0	ROM ID byte 2	Yes	No	No	0
05,1,0,0,0	ROM ID byte 3	Yes	No	No	0
00,1,0,0,0	ROM ID byte 4	Yes	No	No	0
08,1,0,0,0	ROM ID byte 5	Yes	No	No	0
00,1,0,0,0	ROM ID byte 6	Yes	No	No	0
16,1,0,0,0	ROM ID byte 7	Yes	No	No	0
BE,1,0,0,0	Read Scratchpad	Yes	No	No	0
FF,1,0,0,0	Read Byte 0	Yes	No	No	0
FF,1,0,0,0	Read Byte 1	Yes	No	No	0
FF,1,0,0,0	Read Byte 2	Yes	No	No	0
FF,1,0,0,0	Read Byte 3	Yes	No	No	0
FF,1,0,0,0	Read Byte 4	Yes	No	No	0
FF,1,0,0,0	Read Byte 5	Yes	No	No	0
FF,1,0,0,0	Read Byte 6	Yes	No	No	0
FF,1,0,0,0	Read Byte 7	Yes	No	No	0
FF,1,0,0,0	Read Byte 8	Yes	No	No	0
E3,0,0,0,0	Switch to Command Mode	No	No	No	0
C5,1,1,0,0	Reset 1-Wire Bus	Yes	Yes	No	0

Table 1. DS1820 Example Command Script

Figure 4 shows the diagram of the VI developed for the DS1820 temperature sensor. The command script is that shown in Table 1 but here the device ROM ID is a string input that is then properly formatted and placed into the script at the appropriate locations. The entire script string is then passed to the 1-Wire Serial Driver VI. The output of the 1-Wire Serial Driver VI is the response byte array. The appropriate elements of this array are then selected to determine the temperature measurement and to perform a CRC data check.[5] The process is repeated if an error is detected. CRC8 and CRC16 Vis have been developed for use where needed in the various device VIs. Figure 5 shows the LabVIEW front panel for the DS1820 temperature sensor VI.

As another example, Figure 6 shows the LabVIEW diagram of the DS2406/7 addressable FET switch VI. This VI can be used with either the newer DS2406 or the obsolete DS2407

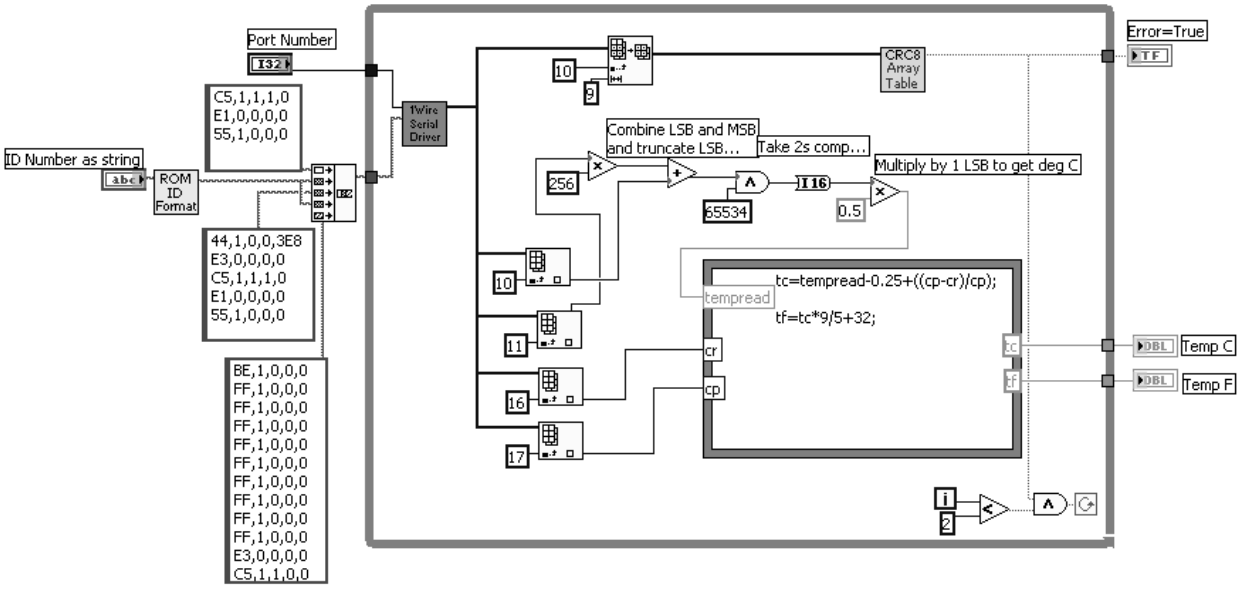


Figure 4. DS1820 Temperature Sensor VI Diagram

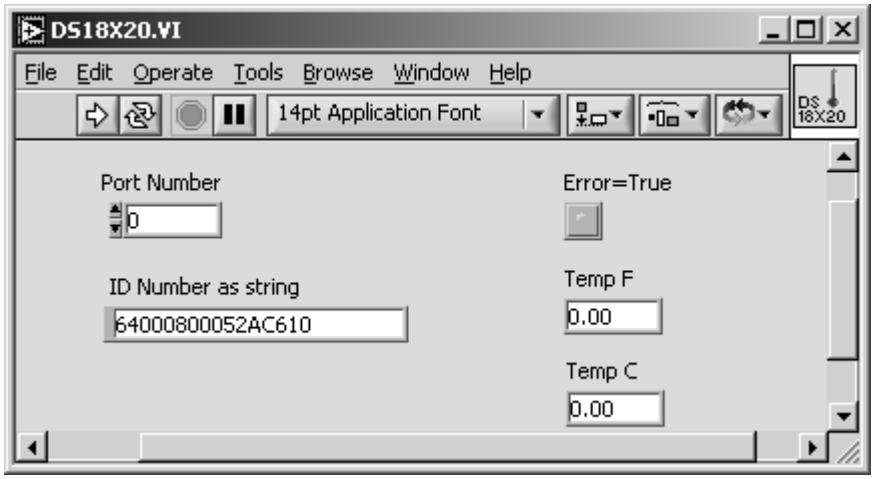


Figure 5. DS1820 Temperature Sensor LabVIEW VI Front Panel

devices. Here the state of the front panel Boolean switches is used to format a command byte in the script. With this device, the response byte data is verified by performing a CRC16 error detection algorithm. Table 2 details line-for-line the commands used to communicate with the device. Figure 6 shows the LabVIEW front panel for the DS2406/7 VI.

DS9097U Initialization

Prior to sending the first commands on the 1-Wire bus, the DS9097U must be properly initialized. Power for the DS9097U electronics is derived from the RTS and DTR serial port signals being set high. Various other parameters of the DS2480B must also be configured at this time (1-Wire bus slew rates, timing, etc.). This initialization is performed by a separate VI writing command bytes directly to the serial port as indicated in the DS2480B application note.[6]

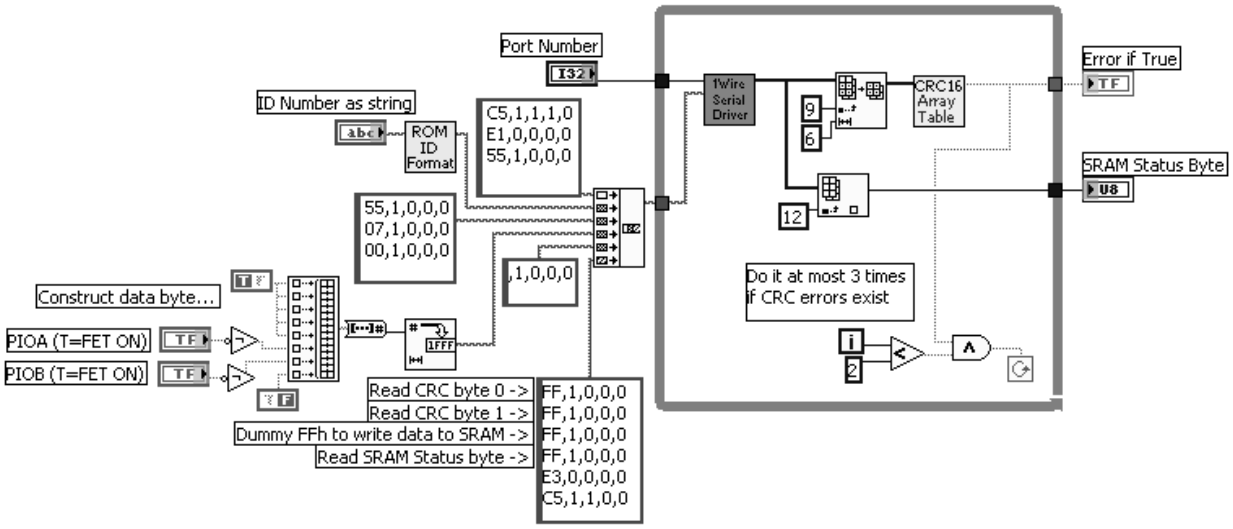


Figure 6. DS2406/7 LabVIEW Diagram

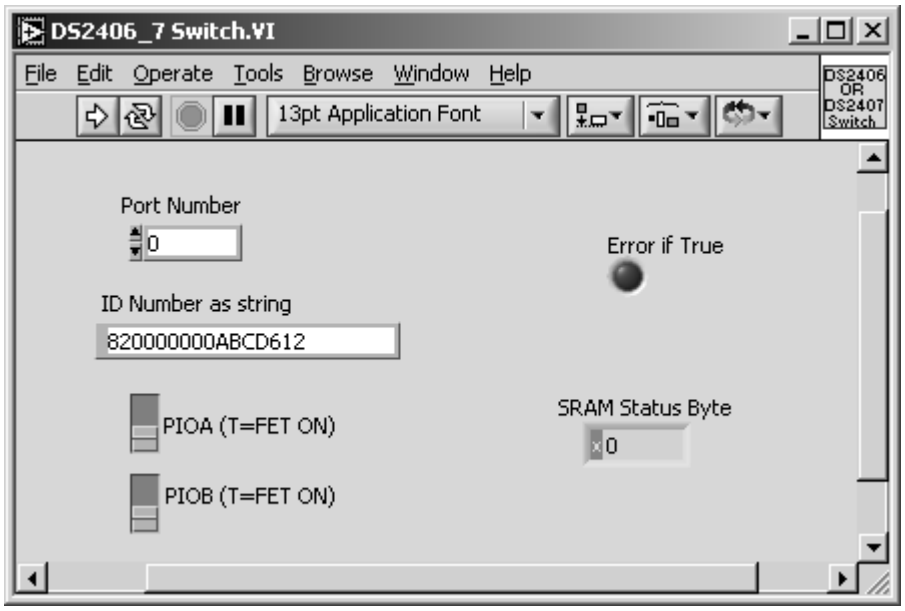


Figure 7. DS2406/7 LabVIEW Front Panel

Searching for Devices

Before a device can be used, its ROM ID must be determined. This is done by connecting the device to the bus and performing a ROM search algorithm.[7] The search algorithm was implemented in LabVIEW to be compatible with the philosophy of this programming environment. Running the 1-Wire Search VI algorithm.[7] The search algorithm was implemented in LabVIEW to be compatible with the philosophy of this programming environment. Running the 1-Wire Search VI.

Command Script String	Command Line Function	Response Expected?	Wait for Response?	Flush?	Delay
C5,1,1,1,0	Reset 1-Wire Bus	Yes	Yes	Yes	0
E1,0,0,0,0	Switch to Data Mode	No	No	No	0
55,1,0,0,0	Match ROM ID	Yes	No	No	0
12,1,0,0,0	ROM ID byte 0	Yes	No	No	0
D6,1,0,0,0	ROM ID byte 1	Yes	No	No	0
BC,1,0,0,0	ROM ID byte 2	Yes	No	No	0
0A,1,0,0,0	ROM ID byte 3	Yes	No	No	0
00,1,0,0,0	ROM ID byte 4	Yes	No	No	0
00,1,0,0,0	ROM ID byte 5	Yes	No	No	0
00,1,0,0,0	ROM ID byte 6	Yes	No	No	0
82,1,0,0,0	ROM ID byte 7	Yes	No	No	0
55,1,0,0,0	Write Status Function	Yes	No	No	0
07,1,0,0,0	Memory Address Lower Byte	Yes	No	No	0
00,1,0,0,0	Memory Address Upper Byte	Yes	No	No	0
xF,1,0,0,0	Byte to write to memory: x=7, both switches OFF x=5, A ON, B OFF x=3, A OFF, B ON x=1, both switches ON	Yes	No	No	0
FF,1,0,0,0	Read CRC Byte 0	Yes	No	No	0
FF,1,0,0,0	Read CRC Byte 1	Yes	No	No	0
FF,1,0,0,0	"Dummy" write to SRAM	Yes	No	No	0
FF,1,0,0,0	Read SRAM Status Byte	Yes	No	No	0
E3,0,0,0,0	Switch to Command Mode	No	No	No	0
C5,1,1,0,0	Reset 1-Wire Bus	Yes	Yes	No	0

Table 2. DS2406/7 Example Command Script

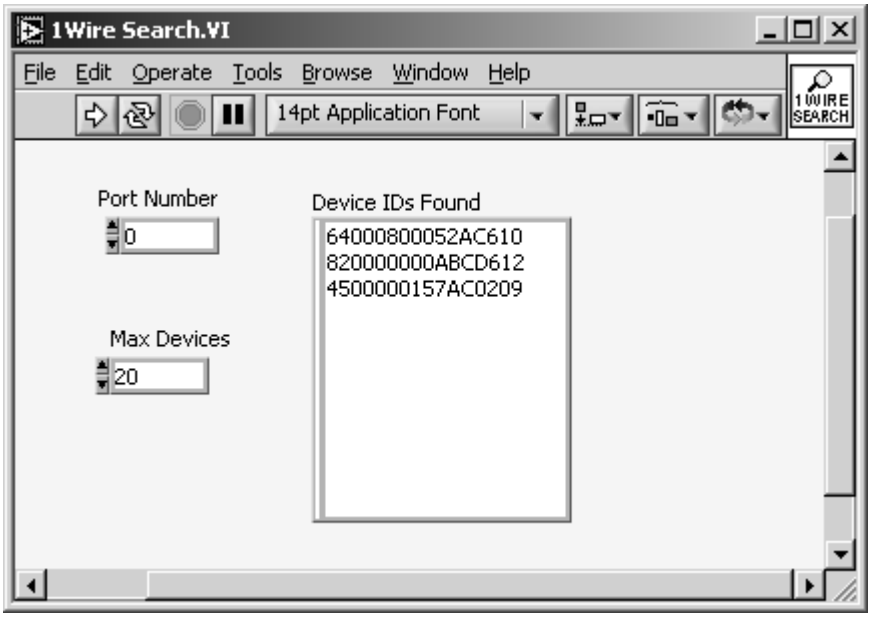


Figure 8. 1-Wire Search VI Front Panel

Running on a Macintosh Computer

As stated earlier, it is desirable to have the ability to run the 1-Wire device applications on Macintosh computers as well as IBM compatible PCs. The student version of LabVIEW can be installed on either type of computer and some students prefer to use Macs. The Macintosh serial port (GeoPort) is not an RS232 standard port. The GeoPort contains differential transmit and receive lines and does not have the RTS and DTR handshaking lines found in RS232 serial ports. Figure 9 shows a schematic of an adapter cable that can be used to emulate the required serial port for DS9097U use. In this adapter, the differential receive line is ground referenced and an external 12V power supply (a simple wall-mounted transformer power supply) is connected to provide power to the DS9097U via the usual RTS pin. The same DS9097U initialization described earlier must be performed however setting RTS and DTR high is omitted as it has no meaning and is not available on the Mac version of LabVIEW. Because all of the 1-Wire device VIs and the ROM ID search VI are written entirely within LabVIEW without external DLLs or components, they are identical for PC and Mac.

Example Application

The 1-Wire device VIs are written to allow their use as subroutines (subVIs) in higher level applications. As an example, consider the simple application of a thermostat. The temperature of a given space is to be controlled by cycling a heater on and off. The temperature of the space is measured with a DS1820 temperature sensor. Power to the heater is controlled by energizing a relay with a DS2406 FET switch. Figure 10 shows the LabVIEW VI diagram for this thermostat example. The desired temperature and allowable temperature swing are entered on the front panel user interface shown in Figure 11. The present measured temperature and heater status are also displayed.

Conclusions

The LabVIEW programming tools for 1-Wire devices presented here provides students and instructors with a familiar environment in which to quickly develop sophisticated applications. When the 1-Wire device-level VIs are written as functional blocks, their usage is more intuitive and they can easily be incorporated into higher level software designs.

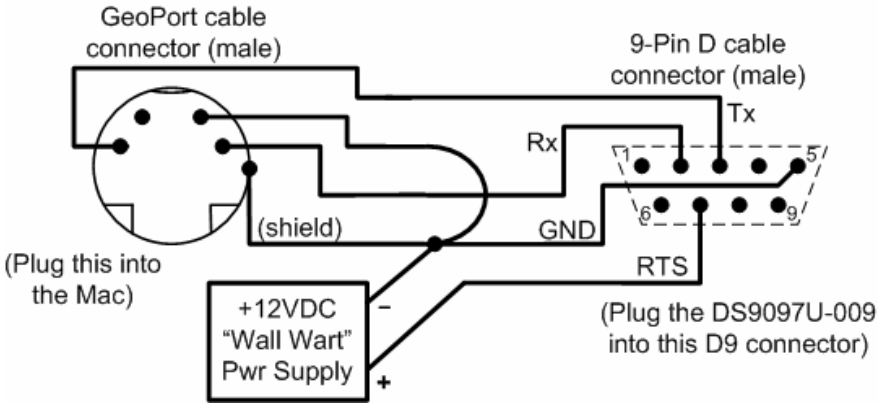


Figure 9. Mac GeoPort to Serial Port Adapter Schematic

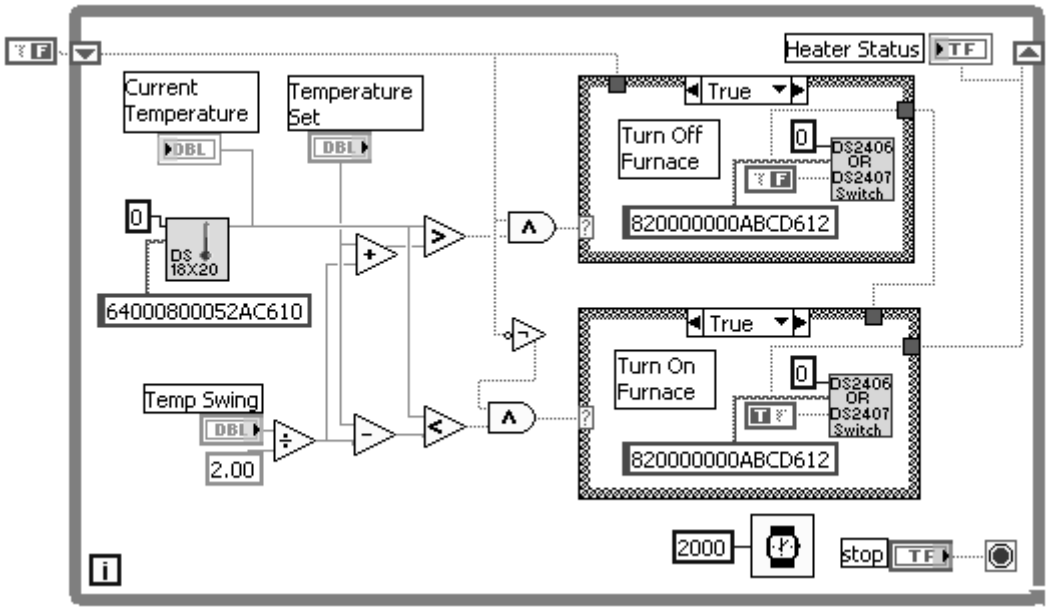


Figure 10. Thermostat Example LabVIEW VI Diagram

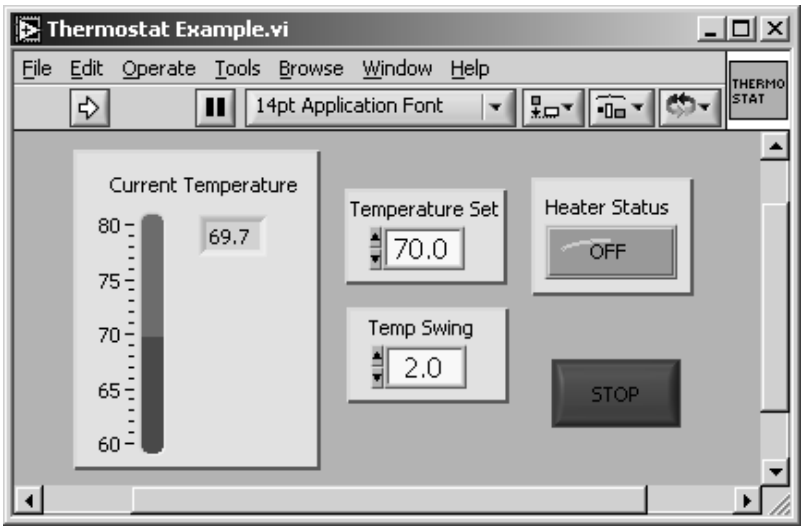
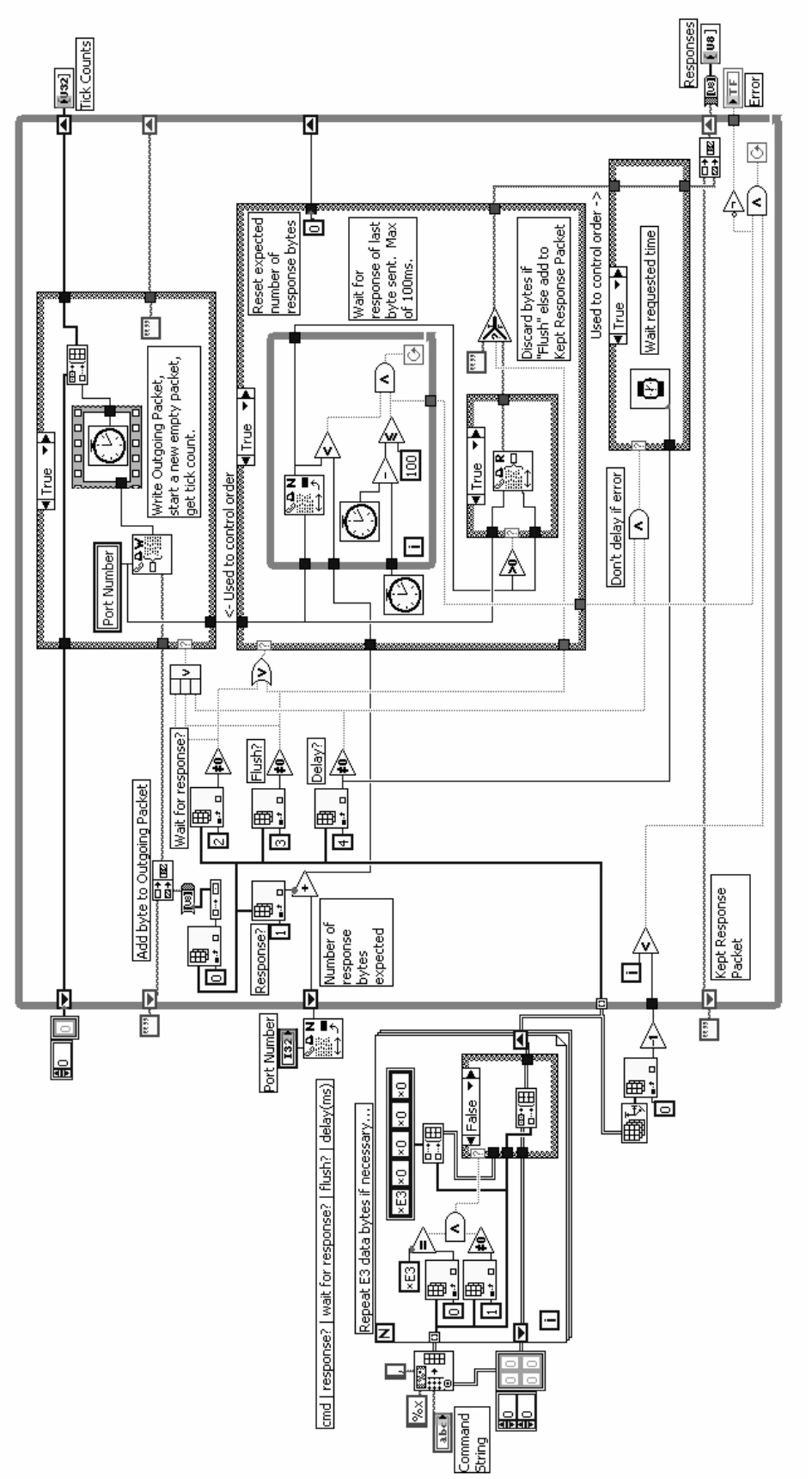


Figure 11. Thermostat Example LabVIEW VI Front Panel

Because the code necessary to communicate with the 1-Wire devices is written entirely in LabVIEW, the VIs presented here can be run on either a PC or Mac platform. Also, because only fundamental functions are used, these VIs can be implemented on very early versions of LabVIEW and relatively slow processors. These are key features necessary for deploying useful 1-Wire device applications on heritage or otherwise obsolete educational laboratory equipment.

LabVIEW virtual instruments for communicating with 1-Wire devices can be obtained from the author by contacting him at dhl10@psu.edu.



Appendix A. 1-Wire Serial Driver VI Diagram

References

1. National Instruments website. http://www.ni.com

2. Dallas Semiconductor / Maxim IC website. http://www.maxim-ic.com

3. Awtrey, D., "Transmitting Data and Power over a One-Wire Bus," February 1997, *Sensors*, Vol. 14, No. 2.

4. Awtry, D., "MicroLAN Design Guide." (Also called the "1-Wire Net Design Guide") http://www.maxim-ic.com/appnotes.cfm/appnote_number/570/ln/en

5. Application Note 27: Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor iButton™ Products. http://www.maxim-ic.com/appnotes.cfm/appnote_number/542/ln/en

6. Application Note 192: Using the DS2480B Serial 1-Wire Line Driver. http://www.maxim-ic.com/appnotes.cfm/appnote_number/990/ln/en

7. Application Note 187: 1-Wire Search Algorithm. http://www.maxim-ic.com/appnotes.cfm/appnote_number/950/ln/en

Biographical Information

Dale H. Litwhiler is an Assistant Professor at Penn State Berks-Lehigh Valley College in Reading, PA. He received his B.S. from Penn State University (1984), his M.S. from Syracuse University (1989) and his Ph.D. from Lehigh University (2000) all in electrical engineering. Prior to beginning his academic career in 2002, he worked with IBM Federal Systems and Lockheed Martin Commercial Space Systems as a hardware and software design engineer. He is a licensed professional electrical engineer in Pennsylvania, USA and his research interests include data acquisition and analysis and sensor technology.