Application Note 206
Using a PC's RS232 Serial Port To Communicate with 2-Wire Devices

www.maxim-ic.com

Introduction

This application note discusses how to build an inexpensive microprocessor circuit to allow a PC to communicate with a 2-wire device using its serial port. In addition to providing general insight on designing hardware, firmware and software to enable PCs to communicate with ICs, a complete PIC microprocessor reference design is presented to show how PC applications can be built around 2-wire devices. The reference design includes a complete schematic, firmware, and a low level C++ serial port code to jump-start any application that would like to incorporate 2-wire devices. The schematic, firmware, and software can be downloaded from Dallas Semiconductor's FTP site.

Overview

Generally, when customers provide feedback related to Dallas Semiconductor's evaluation kits, they indicate they are using them as a way to evaluate Dallas Semiconductor IC's without having to exert their efforts writing software to exercise the parts during the process. However, one trend that seems to be becoming more prevalent in the last couple of years is customers are using the evaluation kits to communicate with the parts during prototyping stages of the design. In the past, this has meant living with both the graphical user interface (GUI) Dallas Semiconductor has written for the product they are using, and the DS9123 serial port adapter, which is a slow method of communicating with 2-wire devices.

This application note has been written to aid customers who would like to build their own circuit to facilitate communications between a PC and 2-wire devices, and to show them how to generate their own custom software specific to their application. It is broken up into four sections that concentrate on hardware design, firmware, software, and a final section that provides a step-by-step walk-through showing how to build the reference design, program the PIC, and begin writing custom software for a 2-wire application.

Designing Hardware to Interface a PC to a 2-Wire Device

First, it is good to clearly define what is being designed before work is commenced, which is hardware-capable of receiving data from a PC, and then relaying it on to a 2-wire device.

There are a lot of ways to transmit data from a PC to external hardware, so why is the serial port preferred for many applications? First, serial ports are available on every PC, and accessing the port has been clearly defined by the RS232 standard. Additionally, both PCs and microprocessors contain serial ports, so communication with a microprocessor is very simple to establish. Three other notable advantages of this standardization are the PC will send the data out at the same baud rate regardless of its processing power, the serial port is generally free on most PCs, and software written for serial ports tends to work across all Windows™ operating system platforms. It is also possible to parasitically power some adapters by stealing power from the I/O signals of the serial port; however, this is not possible for the reference design because it draws too much power. This is primarily a result of the fact the design operates at a relatively high frequency (3.6864MHz), which increases the power requirements beyond what can be parasitically powered.

The major disadvantage of choosing the serial port is it will definitely require a microprocessor to translate the RS232 data format to the 2-wire protocol, where it is possible with some other I/O devices to perform the task without the addition of a microprocessor. Additionally, the serial port uses ±12V signals for

communications. This will require an IC to translate the signal levels to levels a microprocessor can handle. Although it is possible to operate at data rates >115.2kbps, both the PC and the microprocessor need to have the ability to operate at the chosen speed. Generally PCs are able to support all standard baud rates, but the microprocessor may present some limitations with respect to the speed it can send and receive data.

There are other options for building PC hardware. The two most common other than the serial port are using the parallel port or a general-purpose input/output card (GPIO). Both of these options have problems that must be handled to successfully use them. The parallel port is not as standard as the serial port. There have been four standards for the parallel port over the course of time, and there are a variety of chipsets that operate the port in different modes. The original standard parallel port (SPP) was the first standard and is supported by most all PCs. The problem is the SPP mode of the port may have to be enabled in the BIOS of the computer running the application. Additionally, timing is difficult to handle on the parallel port since it depends heavily on the speed of the computer being used.

GPIO cards are not standard equipment on PCs, so they must be purchased separately and installed after the PC is purchased. Additionally, there is not a GPIO card standard, so it is impossible to ensure the software written for one card will work with multiple systems.

Another option that has become popular is USB. The primary advantages of USB are the ability to parasitically power larger circuits, and the bandwidth of the connection. The disadvantages are the circuit would have to operate from the system voltage of 5V, and the firmware and software become much more complex since they now must be able to communicate using device drivers.

Since the serial port is being used for communication to 2-wire devices in the reference design, three main things need to be addressed to allow communication between the PC and the microprocessor.

1) What is going to be used to shift the RS232 signal levels to levels the microprocessor can handle?
2) Does the microprocessor have a Universal Asynchronous Receiver Transmitter (UART), or will a software UART be written?
3) If a hardware UART will be used, what crystal frequencies work well with the microprocessor's baud rate generator?

Because serial ports use ±12V signals for communication, they cannot be interfaced directly to a microprocessor. Fortunately, both Dallas Semiconductor and Maxim make several RS232 level converter chips to translate the ±12V signals to 0 to 5V or 0 to 3V signals. The easiest to use that are 100% RS232 specification compliant are the DS232A, the MAX3221, and the MAX3223. The DS232A is a 5V part, and converts RS232 signal levels to 0 to 5V signals, the two Maxim chips work from 3V to 5.5V and have either one (MAX3221) or two (MAX3223) serial channels. All support baud rates up to 120kbps.

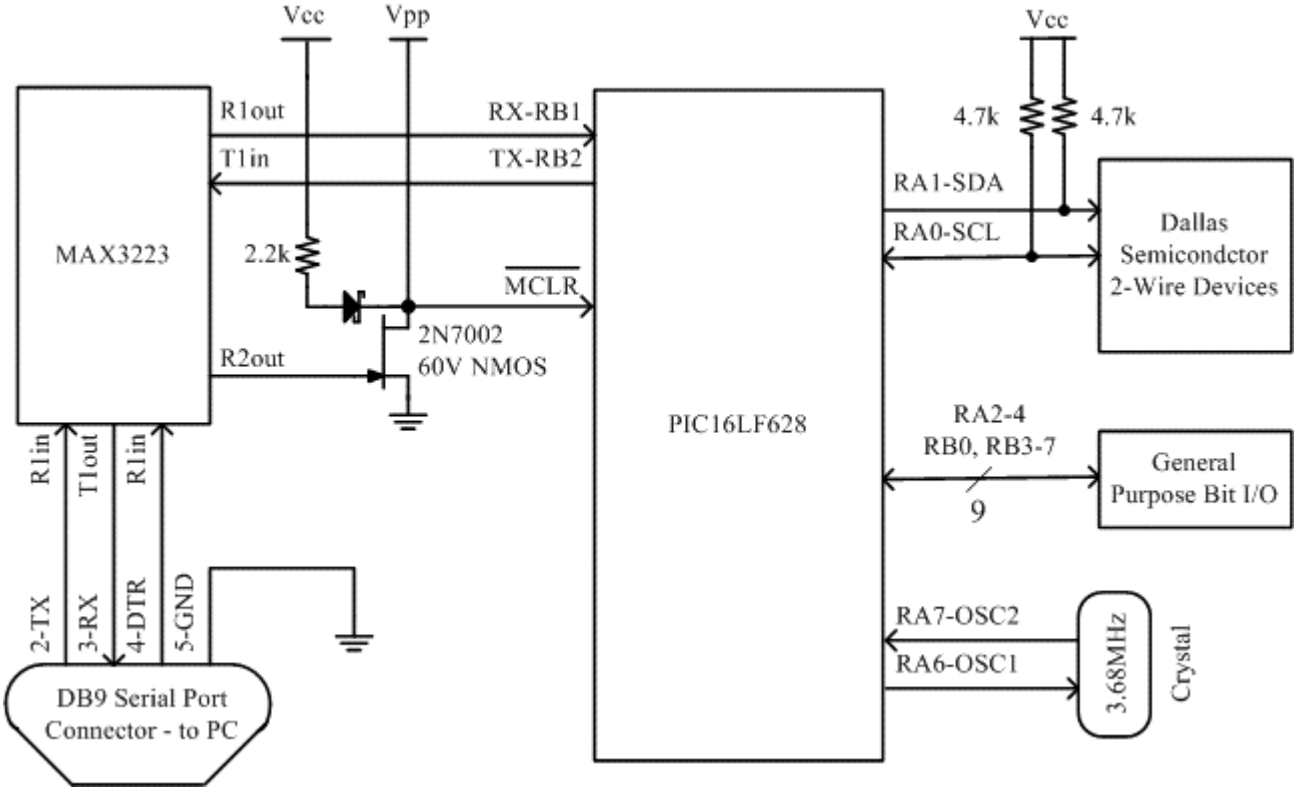
UARTs are used to maintain the timing of the communication while sending and receiving data. Many, but not all, microprocessors contain an RS232-compliant hardware UART. In addition to handling the timing, a hardware UART handles the serialization of the data from bytes to a bit stream, and it sends/receives the start and stop control bits without software intervention. All of the above mentioned can be handled by a software UART, but it generally must be at slower baud rates, and it can take a significant portion of the microprocessor's time to handle just the serial port communication. Conventional wisdom states it is generally better to buy the microprocessor with hardware UART. It allows efficient interrupt driven code to control the serial port peripheral, it tends to be more reliable, and the cost increase is generally minor due to the fact RS232 ports are very commonplace.

One important thing to consider when using a hardware UART is to use either use an oscillator or crystal with a frequency that will work with the microprocessor's baud rate generator. Baud rate generators generally use the clock frequency divided by a power of 2 to set the baud rate. Clock frequencies that are fractions or multiples of 11.0592MHz are generally suitable for this task. Once a crystal frequency has been chosen, the baud error should be calculated using the equation in the microprocessor's datasheet. If it is greater than 3%, it is likely communications will not be able to be established between the microprocessor and the PC. Also, a microprocessor using a resistor/capacitor (RC) clock source will most likely not be able to maintain serial communications due to fact the frequency of operation is likely to drift more than 3%. Since the baud rate error will follow the percentage change in the microprocessors clock frequency, this poses the same problem as having a 3% static baud rate error.

Once the PC and the microprocessor are communicating, two open collector I/O lines, with pull-ups to the 2-wire device's V_{CC} level are required for communication to the 2-wire device. The PIC reference design uses two tri-stateable totem pole outputs, but it emulates an open-collector device by either driving the signal low or tri-stating the output. The only difference between this and a true open-collector output is the V_{CC} level of the 2-wire device must be at or below the V_{CC} level of the PIC circuit. If that relationship is not maintained the voltage level of the input will violate the PIC's specification for input levels.

The remaining part of this section concentrates on specifics of the reference design's hardware. Figure 1 shows a block diagram of the reference design circuit.

Figure 1. Reference Design's Hardware Block Diagram



In addition to the criteria at the beginning of this section, the following items were design goals for this circuit:

1) V_{CC} range of 3.0-5.5V
2) 57600 baud serial port communication
3) In-circuit programmable microprocessor
4) Circuit must be able to be reset from software
5) Inexpensive components
6) The board should be small as possible

To accomplish these goals, a PIC16LF628 processor was chosen primarily because of the low-voltage operation and low cost. Other desirable features include a hardware UART for fast interrupt based communication, crystal inputs that allow accurate bard rates, and it is available in a small 20-pin 173mil TSSOP package.

To translate the signal levels for the RS232 port, a MAX3223 was chosen because it provides true RS232 signal levels when used with a single power supply (3.0V to 5.5V). Additionally it has two channels, which will allow the use of the serial port's DTR (data terminal ready) signal to reset the board. This part is also available in a tiny 20-pin TSSOP package. The MAX3223 and the PIC can operate off of the same V_{CC} supply, and both chips will work over a 3V to 5.5V range. This allows the board to work with both 3V and 5V 2-wire parts.

Signals RA0 and RA1 are used to communicate with the 2-wire devices and pull-ups are connected to them for the open collector 2-wire bus. RB1 and RB2 are interfaced to the serial port via the MAX3223, which uses external capacitors (not shown) to generate true RS232 levels. The remaining I/O pins are being used for bit I/O. They can be used for other serial protocols, or to control the other digital inputs on the 2-wire parts if desired. This will be described more in depth in the firmware section. The most complicated section of the circuit shown on the block diagram is the reset circuitry. On a PIC microprocessor, MCLR is an active-low reset signal. The gate of the NMOS is connected to the DTR signal, which is being level shifted by the MAX3223. If the DTR signal at the gate of the NMOS is high, the NMOS will be on, which will hold the PIC in reset. If the DTR signal is low at the gate the NMOS, it will release the MCLR signal, which will allow the MCLR signal to adjust itself to be either V_{CC} in normal operation or V_{PP} if the PIC is being in-circuit programmed. The Schottky diode is present to isolate the V_{PP} supply from the V_{CC} supply during programming, and the resistor is limiting the diode's through current when the NMOS is forcing the processor into reset. Although MCLR is brought out to a connector, this pin should be left disconnected during normal operation. It is used only for in-circuit programming the PIC.

A complete schematic and bill of materials (BOM) for the entire reference design are available on Dallas Semiconductor's FTP site. A link to the location on the FTP site has been provided in the walk-though section of this document.

Designing Firmware to Communicate with PCs and 2-Wire Devices

The task at hand is a data translation from the RS232 serial protocol to the 2-wire protocol, but there are other things to consider. Primarily, start and stop bus commands must be issued for a 2-wire master to establish communication with a slave device. Additionally, the amount of data to be written to or read from a specific device in a specific application will vary greatly, as will the device address and the functions performed by reading and writing at specific addresses. Thus, the firmware written contains no device-specific commands, and it was designed to be a low-overhead protocol executing any given command as fast as possible. This allows the software to make the decisions and control the application program flow, while

the firmware just receives basic commands and executes them. Figure 2 shows the command protocol used by the reference design.

The items considered essential and basic for 2-wire communication were the start (re-start) bus command, stop bus command, write a data byte, read a data byte with acknowledge, and read a data byte without acknowledge. Three other items are supported by the firmware. The first is a command to toggle SCL nine times, which is useful to reset the 2-wire bus if there is an error detected during communication. The last two are bit I/O reads and writes, which either reads the state of an I/O pin or sets the state of the I/O pin depending on which command is issued. These commands are present to allow using the remaining I/O signals on ports A and B of the PIC for whatever may be desired by the designer.

Figure 2. Serial Port Commands to Communicate with the PIC Circuit

Command	First Byte Sent	Second Byte Sent	Returned Byte(s)
Start (Re-Start)	A0h	0x00, Ignored	0xB0 command ack 0xFA command failed
Write Byte	A1h	Data	0xB1 command ack 0xFA command failed
Read Byte	A2h	Acknowledge 0x01 = Ack, 0x00 = Nack	First Byte Returned = Data Second Byte Returned 0xB2 = command Ack 0xFA= Failed
Stop	A3h	0x00, Ignored	0xA3 command ack 0xFA command failed
Toggle SCL 9 Times	A4h	0x00, Ignored	0xA4 command ack 0xFA command failed
Bit I/0 Read	E?h	0x00 Ignored	0x00 read bit clear 0x01 read bit set 0xFA command failure
Bit I/0 Write	F?h	0x00 clear bit 0x01 set bit	0xF0 command ack 0xFA command failure
? Values	? Designates one of the unused pins microprocessor pins. This portion of the command processor's code was added to facilitate using this adapter for other non-standard protocols such as the 17-bit 3-wire shift register in the DS1867.		

Figure 3 demonstrates the sequence required to communicate with a 2-wire device responding to address 0x40. Each operation requires the PC to send two bytes to the PIC. Once the second byte is received by the PIC, it will begin to process the data it has received. The first byte received determines the type of operation (start, send data, etc.) to be executed. If the command requires a data operand, it will look at the second byte sent, else the second command byte is ignored. Since the processor expects two bytes per instruction a dummy byte must be sent if the second byte is not required for the command. If the PIC receives an invalid command, it will return 0xFA, which indicates a failure.

At least one value is always returned to acknowledge the command has completed successfully or failed, and in the case of a 2-wire read byte operation, both the data byte and the command's acknowledgement is returned. The acknowledge byte the PIC is returning for each instruction is really informing the software that two items are both happening successfully. First, it confirms the PIC is communicating with the PC. This may seem simple and reliable most of the time, but it does provide feedback to inform the user of a disconnected serial cable, or power has not been applied to the application board. Second, it verifies the PIC, which is continually monitoring the 2-wire communication, is seeing the acknowledgement it expects. This

means there is no breakdown in communication from either the PC to the PIC, or from the PIC to the 2-wire device.

Figure 3. Example 2-Wire Write and Read Communication Sequences

Send Start Command, Write 0x23 to 2-Wire Device at Address 0x40, Send Stop Command					
Seq #	***1st Byte Sent***	***2nd Byte Sent***	***1st Byte Returned***	***2nd Byte Returned***	***Description***
1	0xA0	0x00 place holder	0xB0 command ack	None	2-Wire Start
2	0xA1	0x40 data	0xB1 command ack	None	Send 0x40 over 2-Wire Bus (Write Addr. Byte)
3	0xA1	0x23 data	0xB1 command ack	None	Send 0x23 over 2-Wire Bus (Data to 2-Wire part)
4	0xA3	0x00 place holder	0xB3 command ack	None	2-Wire Stop
Send Start Command, Read 0x23 from 2-Wire Device at Address 0x40, Send Stop Command					
Seq #	***1st Byte Sent***	***2nd Byte Sent***	***1st Byte Returned***	***2nd Byte Returned***	***Description***
1	0xA0	0x00 place holder	0xB0 command ack	None	2-Wire Start
2	0xA1	0x41 data	0xB1 command ack	None	Send 0x41 over 2-Wire Bus (Read Addr. Byte)
3	0xA2	0x00 read w/ nack	0x23 data	0xB2 command ack	Read w/Nack 1st Byte 0x23 (Data from 2-Wire part) 2nd Byte 0xB2 (command ack)
4	0xA3	0x00 place holder	0xB3 command ack	None	2-Wire Stop

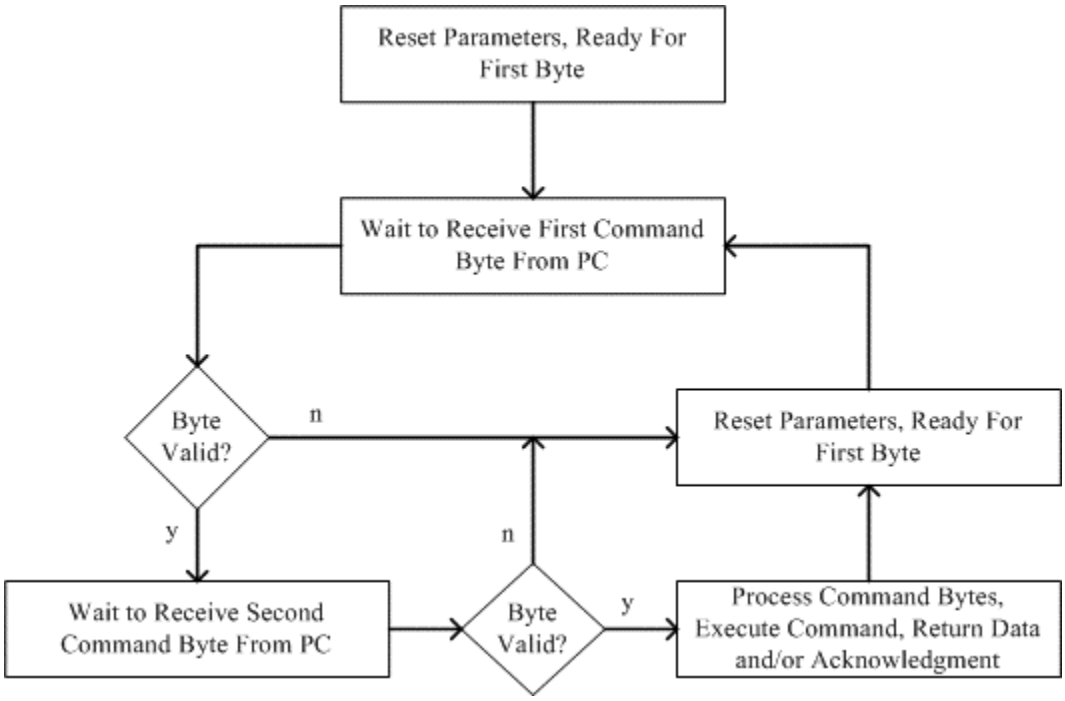
The bit I/O commands (0xE? and 0xF?) can used to set the state of an I/O pin as an output, or it can place the pin in a high-impedance state and read it as an input. Since this application note is not aimed at bit I/O operations, it will not be discussed in depth at this junction. However, the command protocol is included in Figure 2, and the ? values that identify a specific I/O pin can be seen in Figure 4.

Figure 4. Bit I/O Read and Write Addresses

Description	? Values	Port Pin
Bit I/O Read (0xE?) Bit I/O Write (0xF?)	1	RA2
	2	RA3
	3	RA4
	4	RB0
	5	RB3
	6	RB4
	7	RB5
	8	RB6
	9	RB7
	A	RA0
	B	RA1

To implement the firmware discussed above, the program flow shown in Figure 5 was constructed. The program waits to receive two command bytes, and validates each byte as it is received. Once two valid bytes are received, the program executes the command. If two valid bytes are not received, the firmware rejects the command, and returns an error code instead of the command acknowledgment. The firmware (*dsio.hex*) is available on the FTP site.

Figure 5. Firmware Program Flow



In the event custom firmware is required for a project, it is highly recommended the firmware is written and debugged separate of the PC software. This can be accomplished using a terminal program to emulate what is required of the PC during firmware development. This allows a separation of issues, and it can keep the debug time minimal.

Writing Low Level Software for the PC to Control the PIC

The main goal to strive for when writing the low-level PC code to communicate with the PIC circuit is to make the code re-useable. This makes using the PIC circuit with multiple 2-wire projects simple once the initial work to provide PC communications code is complete. This section concentrates on what will be referenced as the "communications code," which allows the PC to communicate with the PIC's firmware. This should not be confused with the final application code to be generated by the GUI software developer. The next section will concentrate on building an application (hardware and software) from the ground up. Additionally, although this topic could be addressed without getting into the specifics of a language, it will be discussed from the point of view of C++ because it is the language of the provided code.

C++ is a very powerful language, which incorporates both a large number of predefined variable types, as well as classes, which allow the definition of user-defined objects and variables. The way the code was made reusable in this instance was a C++ class was written to handle all communication to the PIC circuit. Because all of the communication requirements were contained within a single class, any instance of the class is hence able command the circuit to do any of its functions. The provided class is called CdsPic, and it is contained in two files, DSPIC.cpp, and DSPIC.h.

The C++ class initializes the serial port in the class's constructor. The initialization opens COM1, resets the PIC, and then waits for the PIC's serial port initialization banner to identify the PIC circuit to the PC. If the PIC is "not found", the constructor will close COM1, and try COM2, then COM3, and finally COM4. Once the correct port is found, it will exit the constructor, and the DetectBoard() function will return true when called. If the PIC is not found after all four ports are checked, the constructor will exit, and the DetectBoard() function will return false when called. If the function returns false, it is up to the application software to handle the issue.

Assuming the adapter is "found," all of the 2-wire functions of the class can be called as long as the class remains within scope. These functions include, Start2W(), WriteSlave2W(), ReadSlave2W(), and Stop2W(). Additionally, there is a command (ToggleSCL9x()) to clock the 2-wire bus nine times which can be used to reset the bus in the event communications are disturbed during any transmission. To communicate with the PIC's firmware, these commands call several routines to read and write data via the serial port. These routines are present in two additional files, DSIOLIB1.cpp and DSIOLIB1.h.

Once the class leaves scope, generally when the application is exited, application will deallocate all of the memory being used for its variables. This will call the CdsPic class's destructor, and the destructor will close the serial port.

Since the 2-wire routines are provided for customers, the most of the details of the implementation will not be discussed in this application note. One thing that should be mentioned is the serial port code included will only work in a Window's environment (Windows NT 3.1, Windows 95, or subsequent versions of either). If a different programming language or operating system is desired, the communications software will have to be rewritten to accommodate the OS and language requirements. The easiest way to do this would be to look at the provided C++ code to see what must be sent and what is received while communicating with the firmware. Then mimic the transactions with the new software. The serial port settings required to establish communication are 57600 baud, 1 stop bit, and no parity.

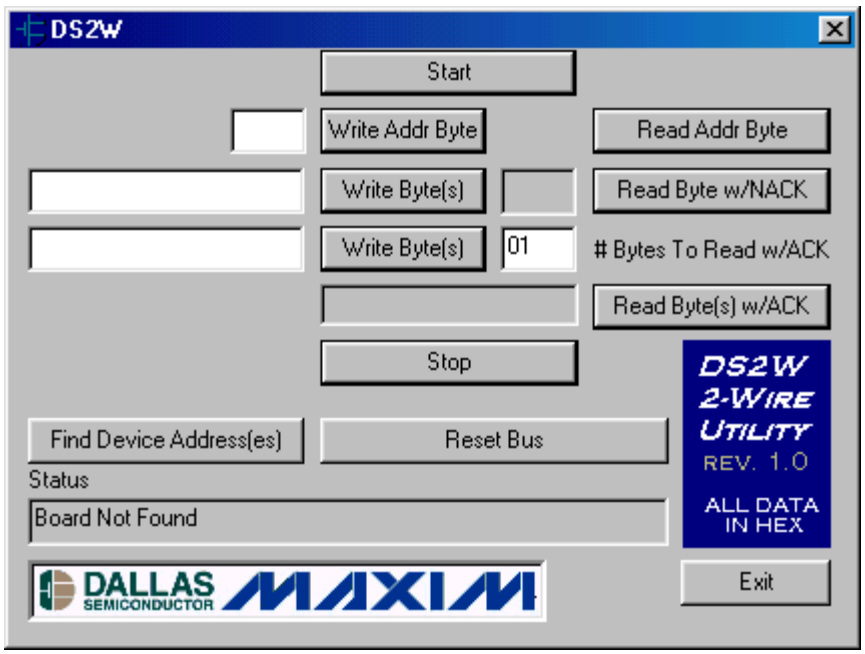
Building Application Hardware and PC Software Using the Reference Design

There are four tasks to complete to build an application based on the PIC reference design.

1) Program the PIC16LF628 microprocessor with the *dsio.hex* file available on the FTP site. Use either the PICSTART Plus programmer available from Microchip or an engineering programmer such as ones available from BP Microsystems.
2) Build the circuit shown in the detailed schematic, which is available on the FTP site. If it is desirable to have the ability to change the firmware at a later time, make sure there is a way to isolate MCLR, V_{CC}, RB6, and RB7 of the microprocessor. These are the pins used to in-circuit program the microprocessor. If a surface mount version of the PIC is being used, it can be more convenient to in-circuit program the PIC than to find a suitable adapter socket to work with the programmer's socket.
3) Download the C++ code (DSIOLIB1.cpp, DSIOLIB.h, DSPIC.cpp, DSPIC.h), which is available on the FTP site.
4) Write the application software. Include the four files listed above in the project, and add a #include "DSPIC.h" directive at the top of the software. Instantiate a member of the CdsPic class *in global space*. It is done in global space for two reasons. It will allow all of the program's subroutines access to the 2-wire functions, *and only a single instance of this class is allowed the application*. If a second instance of the class is opened it will not be able to open the serial port controlled by the prior instance, and therefore it will not be able to communicate with the PIC. After the class is instantiated, use the BoardPresent() member function to determine if the serial port was successfully opened. If the board is detected, continue to call the CdsPic member functions to perform 2-wire functions as needed, else inform the user that the PIC circuit was not found.

An example 2-wire application was generated that can be downloaded from Dallas Semiconductor's FTP site. It is called DS2W, and it is a generic 2-wire tool that allows the user to communicate with 2-wire devices from Window's dialog box interface. The GUI for the program is shown below.

Figure 6. GUI Interface for DS2W



The source code for this application is included on the FTP site, and can be used as an example to aid in development with the PIC circuit. The code shows how to use all of the 2-wire related functions available in CdsPic class to build a Windows-based application.

Additionally, the executable DS2Wa.exe can be downloaded and executed if the hardware is built as described above.

The FTP site information referenced above is available at the following location:

ftp://ftp.dalsemi.com/pub/system_extension/AN206/

Summary

This application note provides an example set of hardware, firmware, and software, which can be used to build a custom application to communicate with 2-wire devices. The solution is simple to implement, requiring only the PIC to be programmed with the provided firmware, the circuit to be built, and the communications software to be called. To aid with the software development, an example 2-wire interface program has been supplied to demonstrate how to include the provided communications software into the end application. If only a simple 2-wire program is required to manually communicate with a 2-wire device, the example application's executable file can be downloaded and used.

Hopefully this application note will simplify new product development using Dallas Semiconductor's 2-wire devices. Please direct any questions regarding development of 2-wire applications to the Mixed Signal Applications email address, MixedSignal.Apps@dalsemi.com.

Note: As an example, Dallas Semiconductor has provided a reference hardware design, firmware, and PC software with this application note; however, use the provided materials at your own risk. Dallas Semiconductor will not be held liable for any complications or damages associated with use of the provided materials.

Contact Information

Company Addresses:

Maxim Integrated Products, Inc
120 San Gabriel Drive
Sunnyvale, CA 94086
Tel: 408-737-7600
Fax: 408-737-7194

Dallas Semiconductor
4401 S. Beltwood Parkway
Dallas, TX 75244
Tel: 972-371-4448
Fax: 972-371-4799

Product Literature / Samples Requests:
(800) 998-8800

Sales and Customer Service:
(408) 737-7600

World Wide Website:
www.maxim-ic.com

Product Information:
http://www.maxim-ic.com/MaximProducts/products.htm

Ordering Information:
http://www.maxim-ic.com/BuyMaxim/Sales.htm

FTP Site:
ftp://ftp.dalsemi.com