

## APPLICATION NOTE 3230

## How to Use a PC's Parallel Port to Communicate with 2-Wire Devices

*Abstract: The purpose of this application note is to show how to build a quick, simple, and cheap 2-wire (I<sup>2</sup>C-compatible) interface using a PC's parallel port. This application note was specifically written for those who just received or just ordered Dallas Semiconductor device samples and realized that they now need a way to communicate with the 2-Wire devices. Although many solutions exist, the one presented here does not require a microcontroller, firmware, nor any difficult to procure ICs. To further aid in the task at hand, Windows® 95/98 software is provided to eliminate the headaches associated with the daunting task of trying to debug both hardware and software simultaneously. Being given the software, along with a couple debugging tools, all efforts can be applied solely to getting the hardware up and running as quick as possible.*

### Introduction

The purpose of this application note is to show how to build a quick, simple, and cheap 2-wire (I<sup>2</sup>C-compatible) interface using a PC's parallel port. This application note was specifically written for those who just received or just ordered Dallas Semiconductor device samples and realized that they now need a way to communicate with the 2-Wire devices. Although many solutions exist, the one presented here does not require a microcontroller, firmware, nor any difficult to procure ICs. To further aid in the task at hand, Windows 95/98 software is provided to eliminate the headaches associated with the daunting task of trying to debug both hardware and software simultaneously. Being given the software, along with a couple debugging tools, all efforts can be applied solely to getting the hardware up and running as quick as possible.

Since this hardware does interface to a PC, the standard disclaimers apply. Although the presented hardware and software was built and tested together, use at your own risk. Be sure to double and triple check all connections and power supply voltage since it will directly connect to the PC's parallel port circuitry. Dallas Semiconductor/Maxim cannot be held responsible for any damages that can result. This solution is intended for evaluation purposes only and not intended to be used in production or distribution. Although the current software is Win95/98 only, a future version will be available which will support Win NT.

### Why Use the Parallel Port Instead of the Serial Port or USB?

The parallel port was chosen in this application note because the goal was to build an interface quick, simple, and cheap. The parallel port interface is based around a 74HC05 logic chip along with some passive components. A serial port solution on the other hand would require a microcontroller, firmware, programmer or in-system-programmability, level translators, crystal, as well as many more passive components. Likewise, USB also requires a considerable amount of hardware, in addition to a much greater software and firmware investment. Furthermore, debugging USB hardware often requires very expensive protocol analyzers. So in terms of our goal, the selection is a obvious. But before the final decision is made, the drawbacks must also be analyzed to ensure that they are tolerable.

Probably the biggest drawback of the parallel port is its future. More and more PC manufactures are dropping parallel ports as well as serial ports in favor of USB. Another drawback of the parallel port is that most of its configuration is set in the PC's BIOS. This means that if communication cannot be established between the PC and the hardware, then the PC must be restarted so that you can enter the BIOS utility (usually by pressing F1 or DEL during startup) and tweak the parallel port settings. This is not a show-stopper in an engineering/evaluation environment, but is a huge support headache when in the hands of the general public. Next, although more of an inconvenience than a drawback is PC-to- PC variation. Since the parallel port is legacy hardware, it has seen changes, variations, and improvements over the years. However when designing hardware, you would like for it to work with old parallel ports as well as new and future. This results in assumptions (timing and electrical) that must be made when designing the interface hardware. And finally, another drawback, which is a result of the operating system, is the need of a device driver on Windows NT systems. Windows NT does not allow user applications to directly read/write hardware registers. This does complicate matters on the software side and is expensive if a driver is purchased.

Fortunately, Windows 95 and Windows 98 are still common enough not to be too much of an inconvenience in an evaluation environment, not to mention this topic will be moot once the software is modified to support Windows NT.

### Hardware

The schematic for the parallel port 2-wire interface is shown in **Figure 1**. All of the parts are easily obtainable through Radio Shack and Digi-Key. The interface may be built using either surface-mount or through-hole components.



drain and requires R2 to realize high logic levels.

Communication on SDA from the 2-wire device to the PC is achieved using one of the parallel ports input pins (pin 12) in addition to another one of the 74HC05's inverters (U1B). It is very important to point out that in order for the device to be able to communicate back to the PC, the application software must force D1 low so the output of the inverter releases SDA, allowing the 2-wire device to toggle SDA. The input of the inverter U1B monitors SDA. The output of the inverter is pulled-up to  $V_{CC}$  through R6 which is responsible for determining the high logic level.

Series resistor, R5, is used as protection in the event of any contention. This series resistance is not a necessity in this example because pin 12 of the parallel port is input only. If we were using one of the parallel port input/output pins on the other hand, then the resistor is important in case a voltage was externally being applied to the parallel port while the parallel port was trying to output a high. Even if your application software was careful not to cause contention, it is possible that another application may access the port even for a short period of time (for example, the OS periodically scanning for a printer).

Since only three of the six 74HC05's inverters are required to implement 2-wire communication, three inverters are unused. To prevent them from going to waste, the schematic includes an optional circuit which uses two of the unused gates to drive an LED which can be turned on and off under software control. Parallel port pin 17 is an output pin used to control inverters U1C and U1D, which in parallel provide ample sink current to drive a wide variety of LEDs. Resistor R7, like R3 and R4 ensure that the inverter inputs see voltages high enough to interpret them as high. Resistor R8 is the LED current limiting resistor used to control the brightness of the LED. An interesting point to note regarding parallel port pin 17, is that it has an inverter internal to the PC, so writing a '0' to the pin 17 bit gets inverted to a high before reaching the DB25. The high entering U1C and U1D result in a low at the output, turning on the LED.

The inputs of the remaining unused inverters should be tied to GND so the output will be hi-z to prevent any contention problems.

Capacitors C1 and C2 are included as a good practice to bypass the power supply voltage close to where the voltage is applied to the PCB. Likewise, decoupling capacitor C3 should be placed as close as possible to  $V_{CC}$  and GND of the 74HC05.

When connecting SCL and SDA from the interface to the 2-wire device, it is important to make sure that the interface ground is common to the 2-wire device's ground. Also, if SCL and SDA of the interface are being connected to a circuit that contains the 2-wire device, be aware that you may end up with R1 and R2 in parallel with the 2-wire pullup resistors of the application circuit. If you do plan on interfacing to the 2-wire device "in circuit", you can adjust resistors R1 and R2 accordingly or eliminate them if the application circuit already has the SDA and SCL pullups.

Finally, as stated in the schematic, a female DB25 connector was used in the interface prototype. The vendor and part number are shown in the schematic along with the vendor and part number of the male-male DB25 cable. Depending on the supplies you have on hand, it is just as easy to use a male DB25 connector along with a DB25 male-female cable.

## Parallel Port Registers

Communication between the PC and the interface is achieved using only four bits residing in three registers (see Table 1). The default base address for the parallel port (LPT1) is typically 378h. Two other possible locations are 3BCh and 278h. The base address is the address of the 8-bit Data Byte. This byte contains the bits that control pins D7-D0, where the LSB is D0. The byte following the Data Byte, at location base address + 1, is the Status Byte. For the given schematic, the bit of interest in the Status Byte is bit 5 (where bit 0 is the LSB) which is the input of SDA. Reading bit 5 of the Status Byte indicates the state of SDA (through the inversion of U1B). Before performing a read, it is important to first make sure D1 in the Data Byte is written low so inverter U1A is hi-z, allowing inverter U1B to sense the state of SDA. The byte following the Status Byte is the Control Byte located at base address + 2. The bit of interest in the Control Byte is bit 3 (where bit 0 is the LSB). Bit 3 is fed through an inverter internal to the PC and then output at pin 17 of the PC's DB25. This bit enables and disables the LED. Examples of reading and writing the described bits are provided later.

**Table 1. Parallel Port Registers**

REGISTER	LOCATION	LPT1 (DEFAULT)	BINARY**				
			MSB Bit 7				LSB Bit 0
DATA BYTE	base address	378h*					D1   D0
STATUS BYTE	base address + 1	379h		P12 IN			
CONTROL BYTE	base address + 2	37Ah			P17 OUT		

\* Other valid base addresses are 3BCh and 278h.

\*\* Only the bits used in this application are shown.

## Verifying and Debugging the Hardware

Once the hardware is built, be sure to give it a quick visual inspection. Before connecting the interface to the PC or power supply, make sure all the connections are secure, correct, and make sure no undesired short circuits exist. Make sure that  $V_{CC}$  is not accidentally shorted to GND.

Once these precautionary measures are performed, connect a 5V power supply to the interface (do not connect the parallel port cable yet). If the power supply has a current readout, make sure no more than 10mA to 15mA is being pulled. If a larger current is seen, then triple-check all connections.

Make sure the IC is oriented correctly and double check the polarity of any polarized capacitors that may have been used (such as C1). Another possible cause of larger than expected supply current is incorrect pullup resistors values. Accidentally using a 470  $\Omega$  resistor instead of 4.7k  $\Omega$  will cause a noticeable increase.

If everything appears to be operational, connect the parallel port cable. The current draw should only change slightly, again due to the pullup resistors. At this point, you can either give the accompanying software a try, or you can work through the next section and learn how to use the

Microsoft Debug utility to directly read and write the parallel port register which can be used to verify and/or debug the hardware. Reading and Writing the Parallel Port Registers Using Debug Microsoft Windows 95 and 98 come standard with a DOS utility named Debug which can be used to read and write hardware registers. Although Windows NT also includes Debug, the operating system does not allow direct hardware access so using Debug on NT has no effect on the hardware. To run Debug, simply go to Start and select Run... In the edit box, type Debug. A DOS window should appear with a "-" prompt. **Figure 2** shows a screenshot of Debug. Using Debug is easy. For example, to read the byte at address 378h, at the "-" prompt, type the following followed by entering:

```
i 378
```

The result will be the two digit (one byte) hex value read from the specified address. Writing memory can also be performed using Debug. For example, to write 00h to location 378h, type the following followed by entering:

```
o 378 00
```

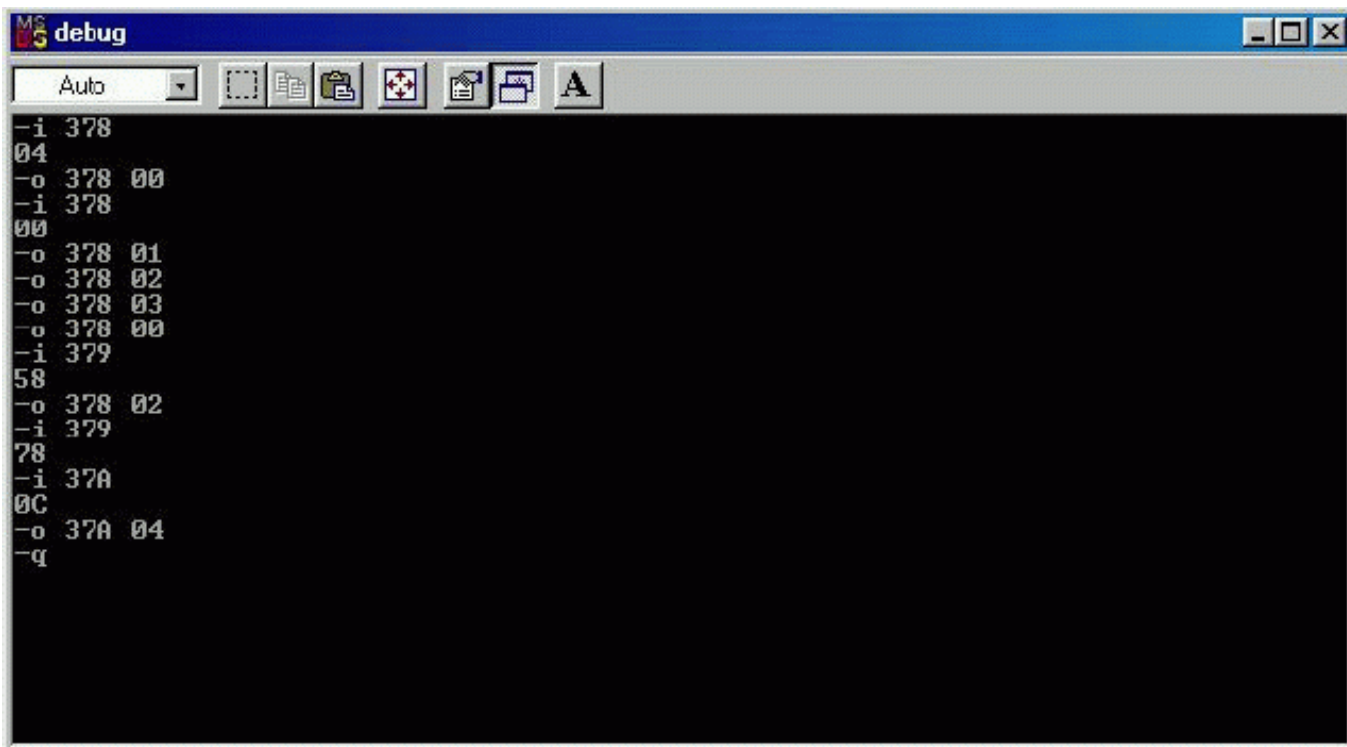
It is highly recommended that caution be used with Debug and that only known registers are accessed. Accessing other registers will likely cause unknown/undesired side effects. To quit Debug, type q, and for help, type ?.

Here are some examples of using Debug to verify/debug the hardware (located at the default LPT address). These examples are also shown in the Debug screenshot (Figure 2).

```
-i 378 Read the Data Byte.
-o 378 00 Write the Data Byte to 00h. This will make SDA and SCL output a logic high.
-i 378 Read the Data Byte. This should return 00 since we just wrote it to 00.
-o 378 01 This will make SCL low and SDA high.
-o 378 02 This will make SDA low and SCL high.
-o 378 03 This will make both SDA and SCL low
-o 378 00 Make sure SDA high before executing the following command.
-i 379 This returns the state of SDA. If bit 5 is 0, then SDA is high. If 1, then SDA is low. Keep
in mind that to read SDA, Data Byte bit 1 should first be written to a 0 so SDA will be
released by inverter U1A. If nothing external is pulling SDA low, then bit 5 should read as a 0.
-o 378 02 Now force SDA low.
-i 379 Read the state of SDA again. This time bit 5 should read as a 1.
```

To turn the LED on and off it is recommended that the unused bits in the Control Byte are left unchanged. This is done by first reading the register and then AND'ing or OR'ing it to set or clear bit 3. Keep in mind that there is an additional inverter internal to the PC.

```
-i 37A Read the Control Byte. As you can see in the screenshot, my PC returned a value of 0C (and
the LED is currently off). To turn the LED on, bit 3 must be cleared. This can be achieved by
ANDing the reading of the Control Byte with F7h. So for my reading of 0C, clearing bit 3
results in 04. This will be written back to the Control Byte to turn on the LED.
-o 37A 04 Turn on LED
```



```
MS-DOS debug
Auto
-i 378
00
-o 378 01
-o 378 02
-o 378 03
-o 378 00
-i 379
58
-o 378 02
-i 379
78
-i 37A
0C
-o 37A 04
-q
```

Figure 2. Debug screenshot.

## Parallel Port Software Utility

Figure 3 shows a screenshot of the parallel port software that was written specifically for the hardware presented in this application note. It can be downloaded from: [http://files.dalsemi.com/system\\_extension/AppNotes/AN3230/ParallelPort.exe](http://files.dalsemi.com/system_extension/AppNotes/AN3230/ParallelPort.exe)

The screenshot may vary slightly from what is downloaded since improvements/additions are likely.

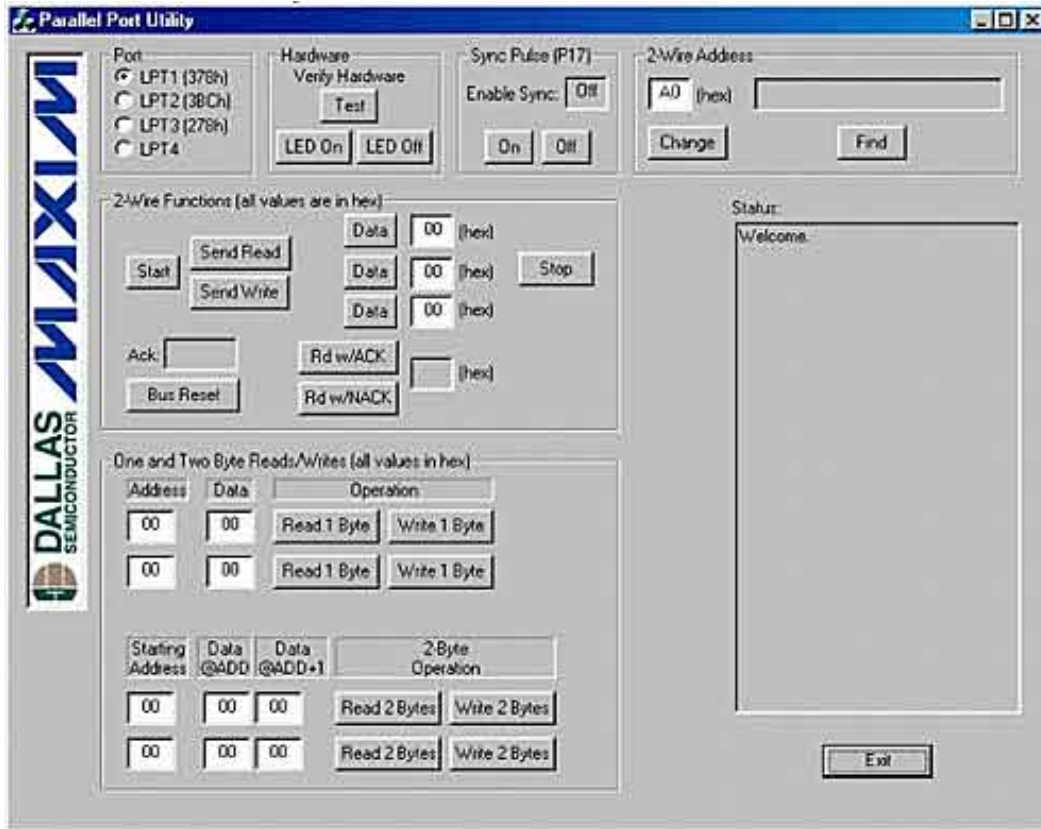


Figure 3. Parallel port utility screenshot.

The software defaults to the LPT address of 378h. If you determine that the software is unable to communicate with the hardware, then another address should be tried by selecting a different Port radio button. The easiest way to determine if the software is communicating with the hardware is by clicking the LED On and LED Off buttons. If that is successful, then try the Test button, which toggles D1 (U1A) and reads the value back through U1B. The results will be displayed in the status box.

The Sync Pulse can be ignored and left off. Later, if you wish to use an oscilloscope to capture 2-wire communication you can enable the sync pulse and place the scope probe on pin 17 of the DB25 and use it as a trigger.

At this point you should be ready to communicate with any 2-wire devices you have connected. Make sure the devices' SDA, SCL, and GND are connected to the hardware. Ideally you would also like to make sure the 74HC05's  $V_{CC}$  is close as possible, if not equal to the  $V_{CC}$  of the device.

There are a couple ways that you can communicate with the device, but first you need to make sure the software knows which 2-wire slave address to communicate with. The software default is A0 (hex), which happens to be the most common for our digital potentiometers although this often depends on the devices' address pins, if it has any. To change the 2-wire address, type it in the address edit box and click "Change". If you do not know the 2-wire address of the device, then check in the devices' datasheet.

The easiest way to read and write one and/or two byte device registers is by using the "One and Two Byte Reads/Writes" section of the software. This section is ideal to program/configure a device since all of the 2-wire details (start, stop, ACK, NACK, etc.) are automatically taken care of. Simply enter the desired memory address to read (in hex) and click on either read or write. If you wish to write, then you also need to fill in the "Data" field with the desired data to write.

Another way of communicating with the devices is by using the "2-Wire Functions" section. This section is ideal if you want to learn the details of the 2-wire protocol. Before showing some examples, a couple of the buttons deserve an explanation. The "Send Write" button uses the 2-wire address from the 2-wire address field and then masks the LSB (the R/W bit) to a 0. The 8 bits are then sent. The "Send Read" button also uses the same 2-wire address but makes the LSB a 1, indicating a read will follow. The 8 bits are then sent. The "Bus Reset" button clocks SCL 9 times, resetting the 2-wire bus.

Examples (using the default slave address):

1. Write memory address 7Fh to 01h.

Start  
Send Write  
Data [7F]  
Data [01]  
Stop

2. Read memory address 7Fh (one byte read)

Start  
Send Write  
Data [7F]  
Start - This often called a repeated start.  
Send Read  
Rd w/NACK - Rd w/NACK is used here because only 1 byte will be read.  
Stop

3. Read memory addresses F0h and F1h (multiple byte read)

Start  
Send Write  
Data [F0]  
Start  
Send Read  
Rd w/ACK - If more than 2 bytes need to be read then this command can be repeated  
Rd w/NACK - Last read must be w/NACK  
Stop

## Conclusion

This application note describes how to build a simple, cheap, and quick 2-wire interface using a PC parallel port. In addition to a schematic, software is provided so that all efforts can be directed towards building the hardware. Once the hardware is built, the software can be used to immediately communicate with Dallas Semiconductor 2-wire devices, without the need to write any software. Questions/comments/suggestions concerning this application note can be sent to: [MixedSignal.Apps@dalsemi.com](mailto:MixedSignal.Apps@dalsemi.com)

Windows is a registered trademark of Microsoft Corp.

---

Application Note 3230: <http://www.maxim-ic.com/an3230>

### More Information

For technical questions and support: <http://www.maxim-ic.com/support>

For samples: <http://www.maxim-ic.com/samples>

Other questions and comments: <http://www.maxim-ic.com/contact>

### Related Parts

DS1845: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS1847: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS1848: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS1855: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS1859: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN3230, AN 3230, APP3230, Appnote3230, Appnote 3230

Copyright © by Maxim Integrated Products

Additional legal notices: <http://www.maxim-ic.com/legal>