Keywords: DS1267, DS1867, DS1868, DS3900, 3-wire , digital pots, 3wire, digital potentiometers

Aug 27, 2002

APPLICATION NOTE 213

Using a PC with a DS3900 to Communicate with DS1267s, DS1867s, and DS1868s

Abstract: This application note describes how to use the DS3900 PC serial port to 3-wire interface to communicate with the three digital pots, the DS1267, DS1867, and DS1868, which use this shift register interface for programming. The DS3900 is a module that has a MAX3223 RS-232 transceiver and a microprocessor which acts as an interface between the PC's serial port the 3-wire device being controlled. The transceiver allows the module's microprocessor to communicate with a PC and the microprocessor implements a command structure, via its parallel port, to allow a PC to read or write the three digital potentiometers. The application note describes the DS3900 and how to interface it to the devices under test. The source code described in the article is available on Dallas Semiconductor's FTP site.

Introduction

The DS1267, DS1867, and DS1868 are digital potentiometers that use a unique 3-wire protocol that can be difficult to interface with a PC. This application note provides a simple hardware/software solution to generate a PC interface to adjust the potentiometer settings and example C++ algorithms to read and write to these devices. The software shown in **Figure 1** and its source code are available on Maxim's FTP site.

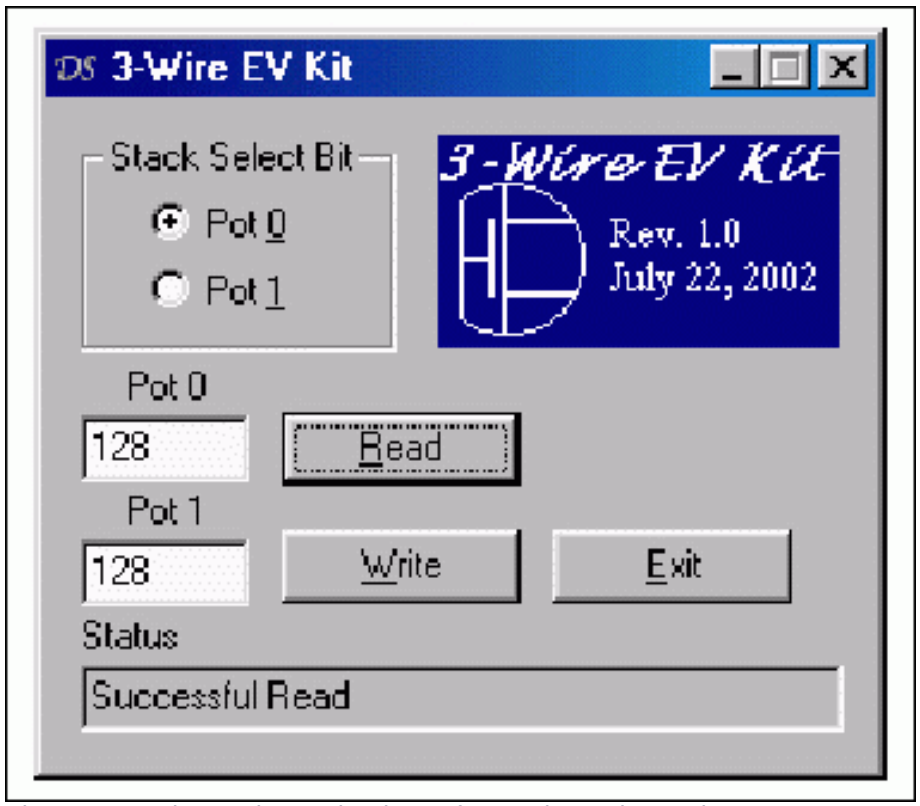

Figure 1. Maxim 3-wire evaluation software (DS3wire.exe).

Hardware

The hardware generated for this application utilizes a DS3900 to communicate with the ICs. The DS3900 is a module that has a MAX3223 RS232 transceiver and a microprocessor. The transceiver allows the module's microprocessor to communicate with a PC and the microprocessor implements a command structure to allow a PC to read or write any I/O pin on the module. In addition to the DS3900 and the 3-wire IC, decoupling capacitors should be used to reduce the noise on V_{CC} caused by the

DS3900 and the potentiometer's digital interface. **Figure 2** shows the connections required to communicate to a 3-wire device using a DS3900 and the DS3Wire application.

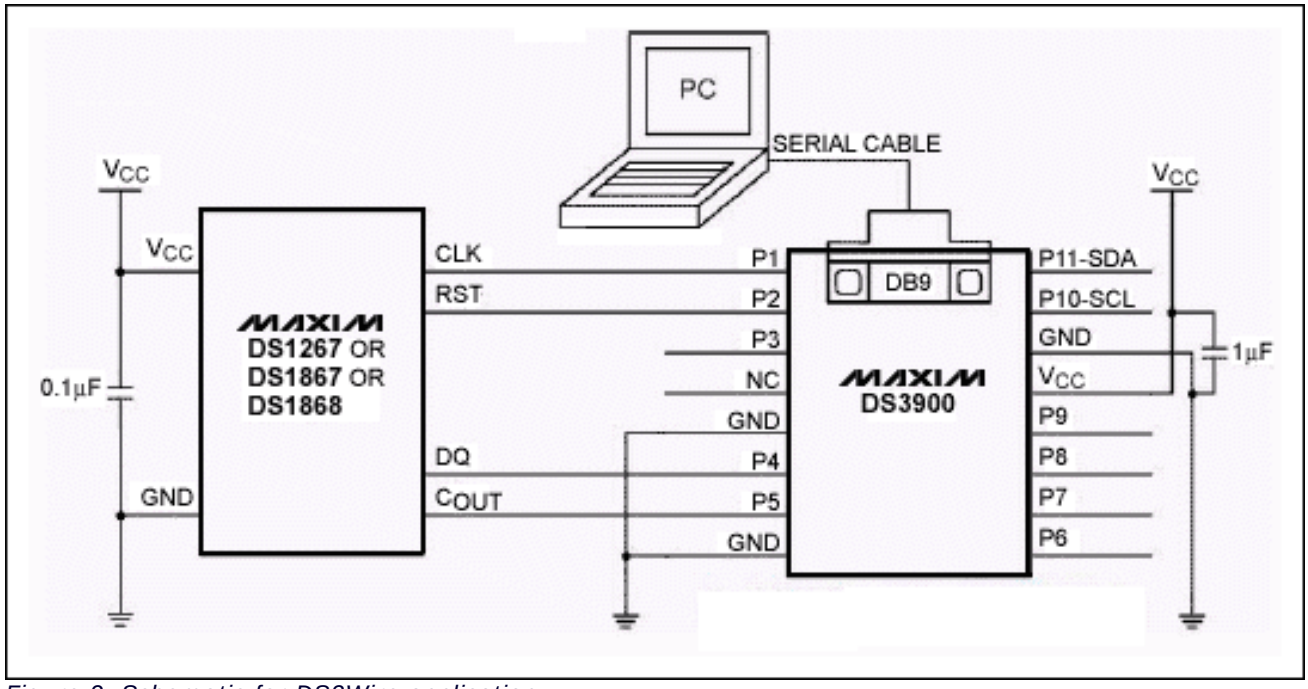

Figure 2. Schematic for DS3Wire application.

Software

The software shown in Figure 1 has three primary routines; initialization of the DS3900 and the dialog box (OnInitDialog), reading the 17-bit register (OnRead), and writing the 17-bit register (OnWrite). These routines are implemented using a C++ class called "CdsPic." CdsPic contains subroutines that allow prewritten and tested RS232/DS3900 code to be used to create the 3-wire algorithm. The instance of the CdsPic class in the DS3Wire application is called "DS3900." The CdsPic class and the RS232 functions are available on Maxim's FTP site for anyone who may be interested. For those not interested in the specifics of the DS3900 implementation, the function names are generic enough to be viewed as pseudo-code. A few example function names and their descriptions are listed in Table 1 for reference.

Table 1. CdsPic member function examples

Example	Description
DS3900.BoardPresent()	Returns TRUE if the DS3900 is detected during the CdsPic class initialization code, returns FALSE if the DS3900 is not detected.
DS3900.Write1(true)	Pin P1 is set to an output and forces its high level. Returns TRUE if no errors are detected during communications.
DS3900.Write1(false)	Pin P1 is set to an output and forces its low level. Returns TRUE if no errors are detected during communications.
DS3900.Read1(state)	Pin P1 is set to an input, read, and P1's input level is returned to a Boolean variable "state." Returns TRUE if no errors are detected during communications.
DS3900Read4(state)	Pin P4 is set to an input, read, and P4's input level is returned to a Boolean variable "state." Returns TRUE if no errors are detected during communications.

3-Wire Basics

This particular version of the 3-wire interface is easiest to understand if it is viewed as a 17-bit shift register with two control signals, clock (CLK) and shift enable ("active-low RST"), and two data signals, data in (DQ) and data out (C_{OUT}). When "active-low RST" is high, the interface is not in reset; therefore, the shift register is enabled. Any positive edge on CLK received while "active-low RST" is high will cause the shift register to shift all of the data one position, moving DQ's present value into the 1st position. This shift will also change the value of C_{OUT}, which always displays the current value of the 17th bit in the register.

Once all 17-bits of data are shifted into the device, the reset signal is brought low, which will transfer the new settings to the registers that control the potentiometer's position and disable the 3-wire interface.

The C_{OUT} pin was designed to provide the ability to cascade multiple 3-wire devices together on the same 3-wire bus, but it does provide read access to the shift register. A read is performed by enabling the interface (setting "active-low RST" high) and clocking the 17-bits in the shift register to C_{OUT} where they can be read one bit at a time. When "active-low RST" is deactivated, the values in the shift register will be the values written to the device by DQ during the read cycle because data is being shifted into the device as it is being shifted out. This requires DQ to be rewritten to the current value of C_{OUT} before each clock pulse is initiated or the read operation will be destructive. A destructive read operation will cause the potentiometer to change position when "active-low RST" is deactivated.

This interface has two primary problems to avoid:

1. Partial writes (shifting less than 17 bits) will produce shift register garbage that is a product of the previous data and the new data. Thus, it is not possible to change one resistor's value without writing all 17-bits.
2. A read function is destructive unless the data shifted out of the shift register is rotated back into the shift register.

The first issue is easily addressed, do not perform partial writes. The second issue can be addressed two different ways. The data sheet shows using a feedback resistor (10kΩ) that will automatically write the value of C_{OUT} to DQ unless DQ is being driven by an output during a write cycle. Although this can be implemented with a DS3900, the design was implemented with C_{OUT} driving a separate input pin. This demonstrates how to implement the interface when a feedback resistor cannot be used. The microprocessor, or DS3900 in this instance, will have to perform the feedback resistor's function in firmware/software. When a value is read on C_{OUT}, it will be written to DQ before the 3-wire device is clocked.

The most common case when the feedback resistor cannot be used is when a 3-wire device is interfaced to an open-collector I/O port. The open-collector I/O pins will have a pull-up resistor to output a high voltage level. This resistor will be in contention with the feedback resistor. If the feedback resistor is smaller than the pull-up, C_{OUT} will always be written into DQ, including during write cycles when the intent is to write new value to the device. If the feedback resistor is larger than the pull-up, DQ will always be high during reads.

OnInitDialog

OnInitDialog is the function called by Windows to initialize the dialog box. If Microsoft Developer Studio is used to generate the dialog box, one of the dialog box construction options leaves comments for the programmer in the code. This will leave a "*TODO*" comment at the end of this function stating to place extra initialization code here.

The following code was added in this instance to ensure the DS3900 powered up correctly, set both CLK and "active-low RST" low, and to initialize the edit boxes with the potentiometer's current position. If the DS3900 is not detected, an error message will inform the user.

Figure 3. Extra initialization code added to OnInitDialog function

```
      // TODO: Add extra initialization here <-Developer Studio Comments
      if(DS3900.BoardPresent()) // <- BoardPresent() checks for DS3900
      {                                                          // <-If found path
            m_sEDIT_Status= "DS3900 Found!";                     // new status message, all systems go
            DS3900.Write1(false);                                // initialize clock
            DS3900.Write2(false);                                // initialize reset
            OnRead();                                            // read pots and update edit boxes
      }
      else
      {                                                          // <-If not found path
            m_sEDIT_Status = "DS3900 not found!@#$";             // new status message, error detect
            UpdateData(FALSE);                                   // Update Dialog Values
            MessageBox("DS3900 Not Found
Check Power and Serial Cable
Restart
                        Applicaion","DS3900 Error");
      }
```

OnRead

The OnRead function reads the 17-bit shift register. It is executed when the read button is pressed and during OnInitDialog. The algorithm assumes the C_{OUT} pin is connected to a separate DS3900 input as shown in Figure 2. This requires the software

to write the value read on C_{OUT} to DQ before the 3-wire device is clocked or the potentiometers will be adjusted unintentionally during read operations. **Figure 4** shows the algorithm used by the application. In addition to reading the 17-bits, it reconstructs the data into variables representing Pot0, Pot1, and the stack select bit, and it updates the dialog box with the UpdateData(FALSE) function. Each transmission to the DS3900 from the PC is monitored for errors, and any disturbance to the communication will result in the termination of the transaction and an error message. If no errors occur, a "Read Successful" message is written to the status box. This function takes approximately 95ms to execute, although results may vary depending on the speed of the PC used.

Figure 4. OnRead function

```
void OnRead()
{
      //Variables used by subroutine
      int success;
      bool bit;
      unsigned char mask=0x80;
      unsigned char pot0=0;
      unsigned char pot1=0;

      if(DS3900.BoardPresent())                                 // Only Read if DS3900 found
      {
            success = DS3900.Write2(true);                      // Pull reset high
            if(success)                                         // Abort Read if comm fail.
            {
                  success += DS3900.Read5(bit);                 // Read Cout (stack bit first)
                  success += DS3900.Write4(bit);                // Copy Read Contents to DQ
                  success += DS3900.Write1(true);               // Clock bit
                  success += DS3900.Write1(false);              // Clock bit
                  m_RADIO_Stack = bit;                          // <-Update Dialog Box Variable
                  if(success == 5)                              // Abort Read if comm fail.
                  {
                        for(int x = 0; x <8 ; x++)              // Pot 1 Read Loop
                        {
                              success += DS3900.Read5(bit);     // Read Cout (stack bit first)
                              success += DS3900.Write4(bit);    // Copy Read Contents to DQ
                              success += DS3900.Write1(true);   // Clock bit
                              success += DS3900.Write1(false);  // Clock bit
                              if(bit)                           // If bit set, set bit in Pot variable
                                    pot1 |= mask;
                              mask = mask >> 1;                 // Adjust Mask for next pass
                        }
                        m_ucEDIT_Pot1 = pot1;                   // <-Update Dialog Box Variable
                        mask=0x80;                              // Reset Mask
                        for(int y = 0; y <8 ; y++)              // Pot 0 Read Loop
                        {
                              success += DS3900.Read5(bit);     // Read Cout (stack bit first)
                              success += DS3900.Write4(bit);    // Copy Read Contents to DQ
                              success += DS3900.Write1(true);   // Clock bit
                              success += DS3900.Write1(false);  // Clock bit
                              if(bit)                           // If bit set, set bit in Pot variable
                                    pot0 |= mask;
                              mask = mask >> 1;                 // Adjust Mask for next pass
                        }
                        m_ucEDIT_Pot0 = pot0;                   // <-Update Dialog Box Variable
                  }
            }
            success += DS3900.Write2(false);                    // Pull reset low
            if(success == 70)                                   // Determine if comm has failed

                  m_sEDIT_Status = "Successful Read";           // Success Message
            else
                  m_sEDIT_Status = "Read Failed";               // Fail Message
      }
      UpdateData(FALSE);                                        // <-Triggers Dialog Box Update
}
```

Notice the last if statement of the read algorithm updates the status message of the dialog box. This will overwrite the

"DS3900 Found!" message of the initialization with "Successful Read" before the dialog box is displayed if the DS3900 is initialized successfully.

OnWrite

The OnWrite function reads the values typed into the dialog box by the user and writes them to the 3-wire device. To read the dialog box values, the UpdateData(TRUE) function is called. In addition to reading the data, UpdateData(TRUE) converts the ASCII strings to unsigned characters for the potentiometer registers, and to an integer value for the stack select bit's radio button. These values are automatically stored in the m_ucEDIT_Pot0, m_ucEDIT_Pot1 and m_RADIO_Stack variables. After the new desired settings are read, the write algorithm instructs the DS3900 to send out the 17-bits of data one bit at a time.

During the write process, communications to the DS3900 are monitored for errors, and the "success" variable keeps a running total of the number of successful data transmissions. If every read/write operation to the DS3900 is successful, the program will update the status box so it reads "Successful Write"; else it aborts the command and returns an error message. The function, which takes about 70ms to execute, is shown in **Figure 5**.

Figure 5. OnWrite function

```
void OnWrite()
{
      UpdateData(TRUE);                                    //Read values of Dialog Box
      //variables used by subroutine
      int success;
      unsigned char mask = 0x80;
      unsigned char pot0 = m_ucEDIT_Pot0;
      unsigned char pot1 = m_ucEDIT_Pot1;
      bool bit = false;

      if(m_RADIO_Stack)                                    // place stack select bit into "bit" variable.
      bit = true;

      if(DS3900.BoardPresent())                            // Only Write if DS3900 Found
      {
            success = DS3900.Write2(true);                 // Pull reset high

            if(success)                                    // Write abortion if comm. fail
            {
                  success += DS3900.Write4(bit);           // write stack select bit
                  success += DS3900.Write1(true);          // Clock bit
                  success += DS3900.Write1(false);         // Clock bit

                  if(success == 4)                         // Write abortion if comm. fail
                  {
                        for(int x = 0; x <8 ; x++)         // Loop for 8 bits of pot 1
                        {
                              if(pot1 & mask)              // Read next DQ value with mask
                                    success += DS3900.Write4(true);
                              else
                                    success += DS3900.Write4(false);
                              success += DS3900.Write1(true);    // Clock bit
                              success += DS3900.Write1(false);   // Clock bit
                              mask = mask >> 1;                  // Adjust mask to next position
                        }

                        mask = 0x80;                       // Reset mask
                        for(int y = 0; y <8 ; y++)         // Loop for 8 bits of pot 0
                        {
                              if(pot0 & mask)              // Read next DQ value with mask
                                    success += DS3900.Write4(true);
                              else
                                    success += DS3900.Write4(false);
                              success += DS3900.Write1(true);    // Clock bit
                              success += DS3900.Write1(false);   // Clock bit
                              mask = mask >> 1;                  // Adjust mask to next position.
                        }
                  }
            }
```

```
            success += DS3900.Write2(false);                // Pull reset low
      }
      if(success == 53)                                     // Comm. Pass/Fail notification.

            m_sEDIT_Status = "Successful Write";            // Pass Message
      else
            m_sEDIT_Status = "Write Failed";                // Fail Message
      UpdateData(FALSE);                                    // <-Trigger Dialog Update
}
```

Conclusion

This application note provides a simple C++ algorithm for reading and writing to the 3-wire devices containing 17-bit shift registers using a DS3900. The write operation takes approximately 70ms, and the read operation, which does not use the feedback resistor, takes about 95ms to execute. Although this is not fast with respect to the 3-wire interface's maximum data rate, it is adequate to evaluate the potentiometers. The software shown in Figure 1 can be downloaded from Maxim's FTP site.

Application Note 213: http://www.maxim-ic.com/an213

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

DS1267: QuickView -- Full (PDF) Data Sheet -- Free Samples

DS1867: QuickView -- Full (PDF) Data Sheet -- Free Samples

DS1868: QuickView -- Full (PDF) Data Sheet -- Free Samples

DS3900: QuickView -- Full (PDF) Data Sheet