Micromint, Inc	**AN819**
	MicroBolt
The MicroBolt controlling an I²C EEPROM	12/30/2005

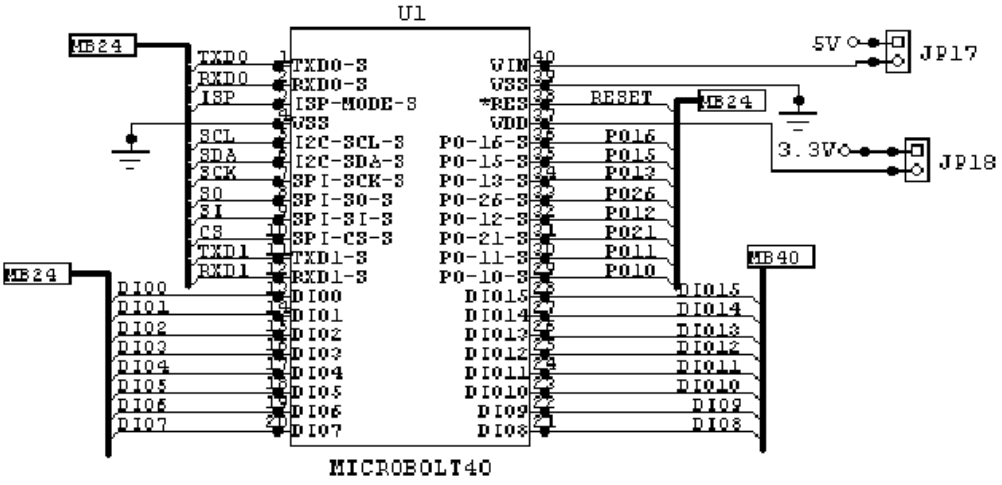
Introduction:

This application notes demonstrates how to use the MicroBolt as an I²C Master to control a slave EEPROM on the I²C serial bus.

Background:

The Philips 2 wire I²C Serial bus is a very popular 2-wire network. Many devices are available that support connection to this serial bus. Since the MicroBolt is based off from a Philips LPC2106 controller, its I²C implementation is highly integrated and provides ease of control over these devices. One such popular I²C device product line is the Microchip 24xxxx EEPROM family. The MicroBolt development board supports an EEPROM, 8 pin DIP package size, via the U7 socket. For this example, an 8K-Byte Microchip 24LC64 device was used.

Schematic:



How it works:

This ImageCraft ICCARM demo project is based upon the MicroBolt I²C Master demo project and utilizes the MicroBolt's I²C serial channel. The MicroBolt is setup as a master on the I²C bus and a Microchip 24LC64 is setup as a slave. The 24LC64 slave address is set to 0x50. The 24LC64 is located on the MicroBolt development board as U7. The example code simply writes 4 bytes to the EEPROM, reads them back continuously, and then outputs the data to the MicroBolt Serial Debugger for inspection.

By including MicroBolt_I2C_Functions.c and MicroBolt_I2C_Functions.h in your own project, the EEPROM functions found in MicroBolt_I2C_EEPROM.c can be copied and used in your own program. The EEPROM functions are very easy to use and isolate the user from any I²C coding. The EEPROM functions also implement ACK polling which polls the EEPROM after a write to verify it has finished writing to the internal EEPROM memory. This maximizes I²C bus throughput and guarantees the data has been successfully written to the EEPROM.

The following EEPROM functions, and their example use, are shown below:

```
EepromWriteByte(0x0001, 0x03);                  // Write the EEPROM at address 0x0001 with data 0x03
EepromDataValue = EepromReadByte(0x0001);       // Read the EEPROM at address 0x0001
```

As shown, for writing to the EEPROM, all that's required is the address and data. For reading, all that is required is the address and then the data is returned.

This provides read and write examples for the MicroBolt when interfacing to an EEPROM over the I²C bus.

For demo purposes, the MicroBolt's UART0 and the MicroBolt Serial Debugger was utilized. Adding this capability to any program is simply done by including the SerialDebugger.c and SerialDebugger.h files and UART0 setup into your project.

Required tools and/or parts

-Microchip 24LC64, 8 pin, 8K Byte EEPROM (can be purchased from Digikey)
-MicroBolt development kit (Board, cables, power supply, schematics, etc.)
-MicroBolt Serial Debugger (PC Windows program on MicroBolt development kit CD or Micromint website)
-Philips Flash Utility (PC Windows program on MicroBolt development kit CD or Philips website)

Instructions
---
1. Plug 24LC64 EEPROM into the U7 socket on MicroBolt development board.

2. Configure the following jumpers on the MicroBolt development board: JP32, JP33, and JP34 need a jumper on pins 2 & 3. Install JP37 and JP38 jumpers. Remove jumper from JP35.

3. Connect serial port cable from PC to J1 of MicroBolt development board (verify COM port jumpers via the datasheet).

4. Power up the MicroBolt development board and download the demo project to the board via the Philips serial flash utility.

5. Power off the MicroBolt development board.

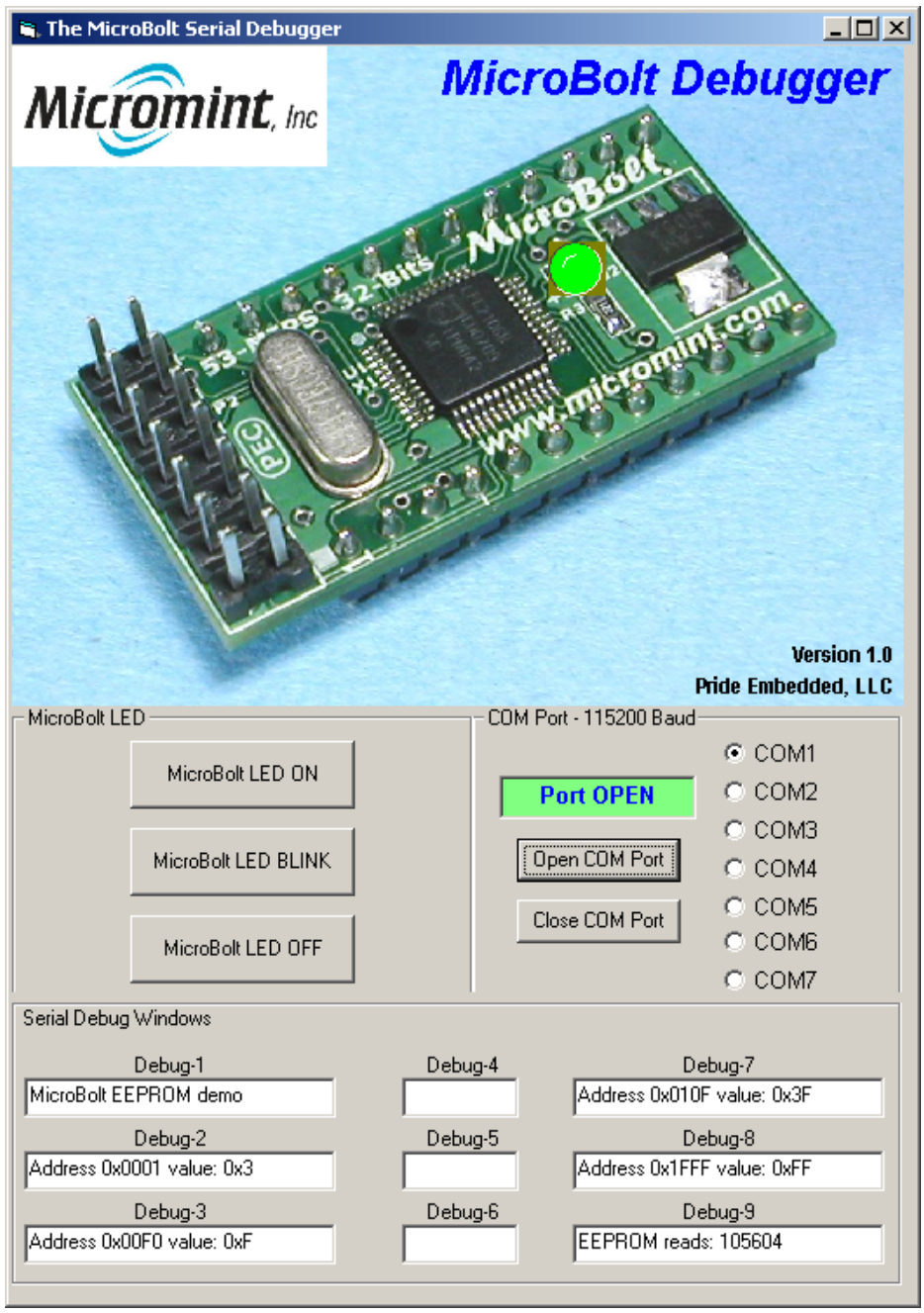
6. Open up the MicroBolt Serial Debugger, select the desired COM port, and open it.

7. Power up the MicroBolt development board.

8. EEPROM test data should now be displayed in the Serial debugger debug windows.

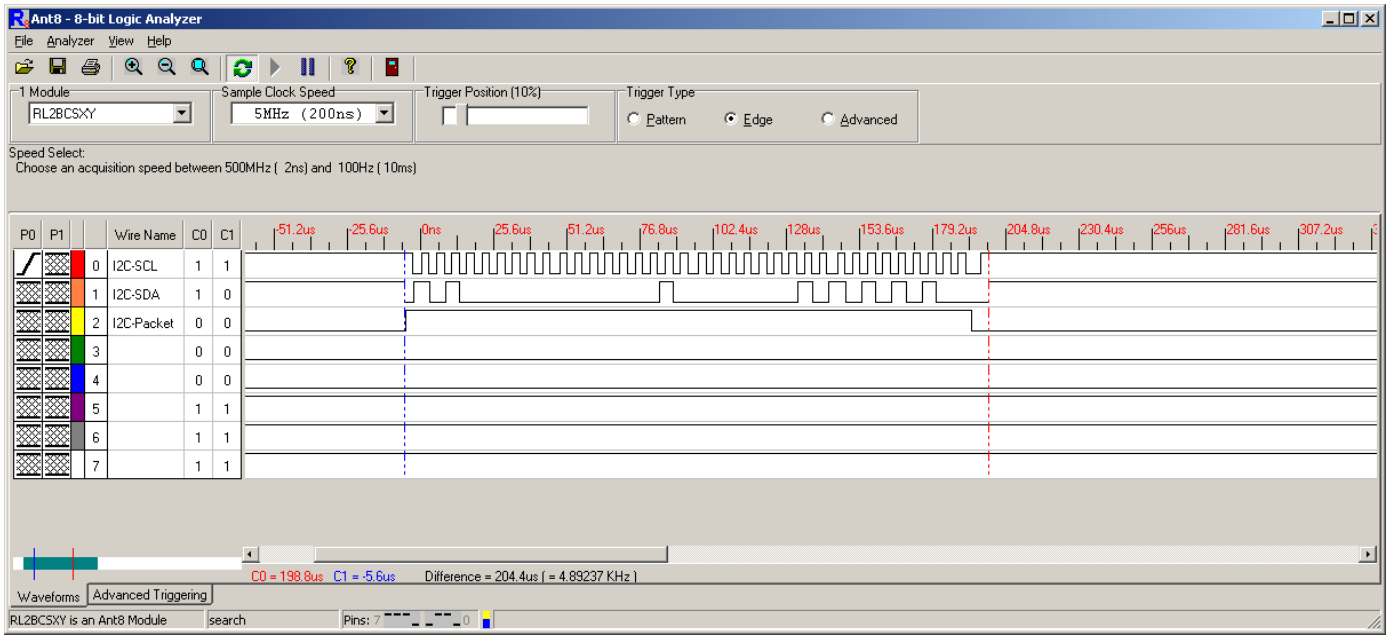
9. Change the EEPROM write data in the code, redownload, and verify the displayed EEPROM data in the debugger changes accordingly.

MicroBolt Serial Debugger screenshot:

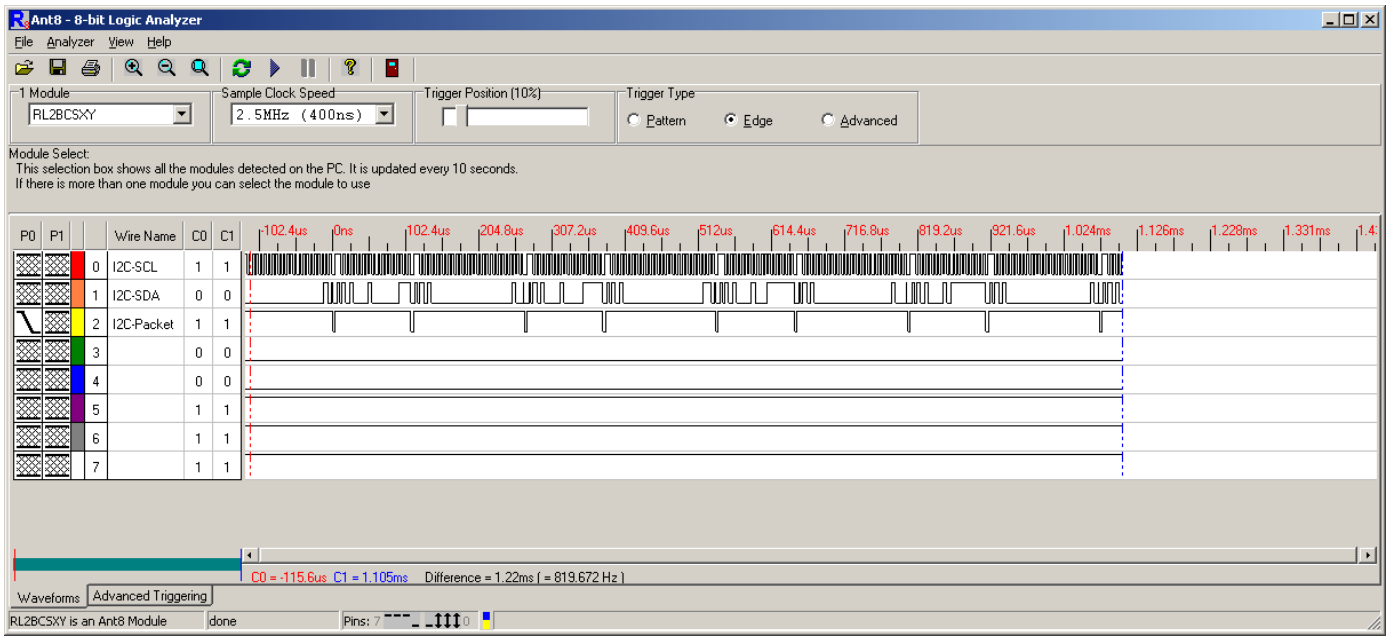


Waveforms:

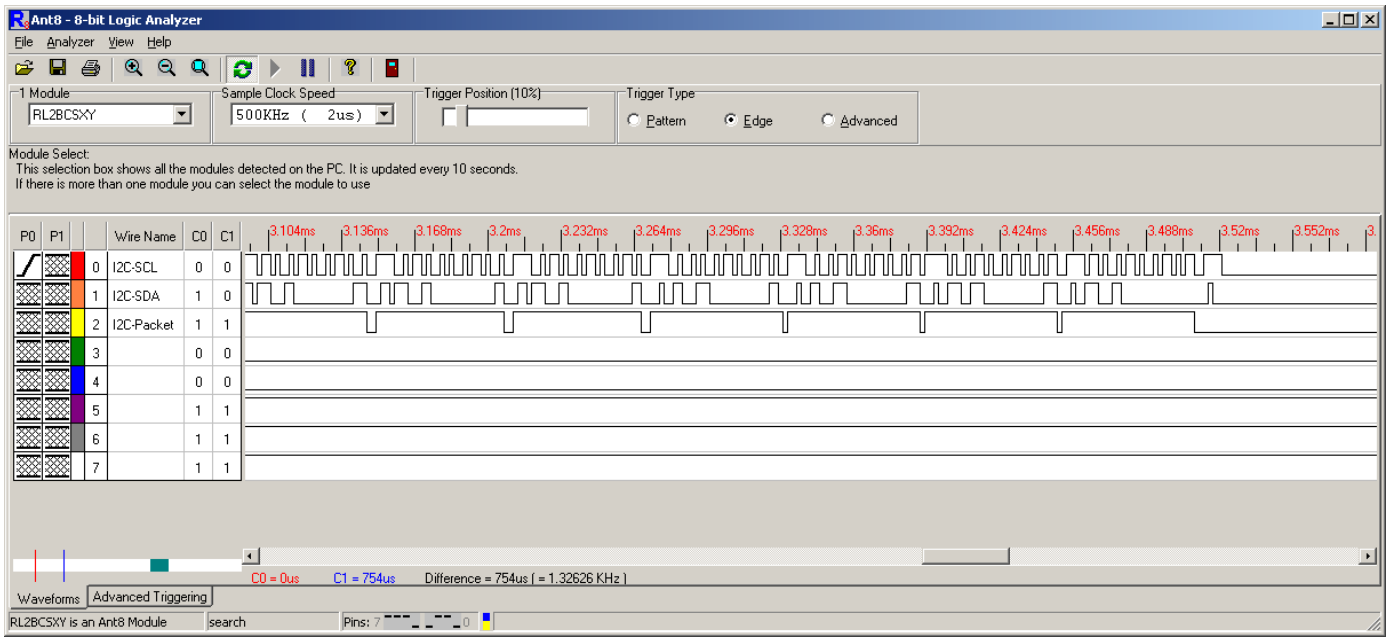
The following waveform capture shows a byte being written to the EEPROM from the MicroBolt over the I²C bus. The first byte is the EEPROM slave address of 0x50, the next 2 bytes are the EEPROM address 0x0101, and the last byte is the associated data 0xAA.



The following waveform capture shows 4 data bytes being read from the EEPROM by the MicroBolt over the I²C bus. This is done via 8 I²C packets. Each read requires an address selection, then a data read from that address. The 4 data bytes being read back are 0x03 0x0F 0x3F 0xFF.



The following waveform capture shows ACK polling from the MicroBolt to the EEPROM over the I²C bus. To determine when the EEPROM has finished writing to its internal memory, the MicroBolt continuously sends the EEPROM slave address of 0x50 and looks for an ACK. If it receives a NAK, then it continues the ACK polling as shown below. On the last packet, an ACK was received and the ACK polling stopped.



Program Listing:

```
/*
|-----------------------------------------------------------------------------------------
|-----------------------------------------------------------------------------------------
| File Name                     : MicroBolt_I2C_EEPROM.c
| Author                        : Micromint, Inc.
| Copyright                     : Copyright © 2005, Micromint, Inc.
| Creation Date                 : 12/30/05
| Version                       : 1.00
| Spaces per tab                : 2
| Description                   : Main C file
| Revision                      : Initial
|-----------------------------------------------------------------------------------------
|-----------------------------------------------------------------------------------------
*/

/*
|-------------------------------------------------
|     Includes
|-------------------------------------------------
*/

#include <ARM/philips/lpc210x.h>
#include <arm_macros.h>
#include "MicroBolt_I2C_EEPROM.h"
#include "MicroBolt_I2C_Functions.h"
#include "SerialDebugger.h"

/*
|-----------------------------------------------------------------------------------------
|     Function        :   main
|     Inputs          :   None
|     Outputs         :   None
|     Purpose         :   Main function for system
|     Author          :   Micromint, Inc.
|-----------------------------------------------------------------------------------------
*/
```

```
unsigned char EepromDataValue = 0;
unsigned int Delay, ReadCount;

void main(void)
  {

  __DISABLE_INTERRUPT();                        // Disable all interrupts

/*
|-------------------------
| Config MAM
|-------------------------
*/
  MAM_CR = 0x00;                                // Turn MAM off (default)
  MAM_TIM = 0x04;                               // Set flash timing to 4 clock cycles

  MAM_CR = 0x02;                                // Fully enable the Memory Accleration Module

/*
|-------------------------
| Config PLL and CCLK
|-------------------------
*/
  SCB_PLLCFG |= 0x23;                           // Set to 59 MHz (0x03 is multiply value of 4)
  SCB_PLLCON |= 0x01;                           // Enable the PLL
  SCB_PLLFEED = 0xAA;                           // Shadow register copy to enable changes
  SCB_PLLFEED = 0x55;                           // in PLLCON and PLLCFG

/*
|-------------------------
| Config PCLK
|-------------------------
*/
  SCB_VPBDIV = 0;                   // Peripheral clock is 1/4th Processor clock which equals 14.7456 MHz

/*
|-------------------------
| Configure VIC
|-------------------------
*/
  VICVectAddr0 = (unsigned)pll_isr;             // Assign the PLL lock ISR vector address
  VICVectCntl0 = INTERRUPT_CHANNEL_FOR_PLL;     // Assign the VIC address to the actual interrupt
  VICIntEnable = INTERRUPT_ENABLE_FOR_PLL;      // Enable the interrupt

  VICVectAddr1 = (unsigned)I2c_ISR;             // Assign the I2C ISR vector address
  VICVectCntl1 = INTERRUPT_CHANNEL_FOR_I2C;     // Assign the VIC address to the actual interrupt
  VICIntEnable = INTERRUPT_ENABLE_FOR_I2C;      // Enable the interrupt

/*
|-------------------------
| Config GPIO
|-------------------------
*/
  PCB_PINSEL0=0x00000055;         // Setup with ICCARM App builder - MicroBolt_I2C_EEPROM.bcf (I2C & UART0)
  PCB_PINSEL1=0x55400000;         //                                   (Secondary JTAG pins)

  GPIO_IODIR |= MICROBOLT_LED;                  // Setup MicroBolt LED as output

  GPIO_IOCLR=0xffffffff;                        // Clear all pins to start with

/*
|-------------------------
| Config UART-O
|-------------------------
*/
  UART0_IER = 0x00000001;                        // Receive interrupts
  UART0_FCR = 0x00000001;                        // Enable the fifos
  UART0_LCR = 0x00000083;                        // Enable the divisor
  UART0_DLM = 0;                                // Divisor latch MSB (for baud rates < 4800)
  UART0_DLL = BAUD_RATE_115200;                 // Divisor latch LSB
  UART0_LCR = 0x00000003;                       // Close it, then UART works with divisor

/*
```

```
|-------------------------
| Config I2C Master
|-------------------------
*/
  I2C_I2SCLH = 37;                              // I2C clock is 200 KHz (14.7456 MHz/(SCLH + SCLL))
  I2C_I2SCLL = 37;
  I2C_I2CONSET = I2C_ENABLE_BIT;                // Enable the I2C channel

  __ENABLE_INTERRUPT();                         // Enable all interrupts

/*
|-------------------------
| Write to the EEPROM
|-------------------------
*/
  EepromWriteByte(0x0001, 0x03);                // Write the EEPROM at address 0x0001 with data 0x03
  EepromWriteByte(0x00F0, 0x0F);                // Write the EEPROM at address 0x00F0 with data 0x0F
  EepromWriteByte(0x010F, 0x3F);                // Write the EEPROM at address 0x010F with data 0x3F
  EepromWriteByte(0x1FFF, 0xFF);                // Write the EEPROM at address 0x1FFF with data 0xFF

  GPIO_IOSET = MICROBOLT_LED;                  // EEPROM written to, turn on MicroBolt LED
  for (Delay = 0; Delay < 370000; Delay++);    // Delay for 100 mS before starting serial routines
  SerialDebugLed(LED_ON);                      // Turn on MicroBolt serial debugger LED
  SerialDebug(1,"MicroBolt EEPROM demo");      // Output string to MicroBolt Serial Debugger Debug window 1

/*
|-----------------------------------------------------
|     Start of application
|-----------------------------------------------------
*/
  while(1)                                      // Do this forever
    {
/*
|-------------------------
| Read the EEPROM
|-------------------------
*/
    EepromDataValue = EepromReadByte(0x0001);  // Read the EEPROM at address 0x0001
    SerialDebug(2,"Address 0x0001 value: 0x%X", EepromDataValue); // Output string to Debug window 2

    EepromDataValue = EepromReadByte(0x00F0);                     // Read the EEPROM at address 0x00F0
    SerialDebug(3,"Address 0x00F0 value: 0x%X", EepromDataValue); // Output string to Debug window 3

    EepromDataValue = EepromReadByte(0x010F);                     // Read the EEPROM at address 0x010F
    SerialDebug(7,"Address 0x010F value: 0x%X", EepromDataValue); // Output string to Debug window 7

    EepromDataValue = EepromReadByte(0x1FFF);                     // Read the EEPROM at address 0x1FFF
    SerialDebug(8,"Address 0x1FFF value: 0x%X", EepromDataValue); // Output string to Debug window 8

    ReadCount++;                                                  // Increment read counter
    SerialDebug(9,"EEPROM reads: %d", ReadCount);                 // Output string to window 9

    for (Delay = 0; Delay < 37000; Delay++);   // Delay 10 mS between serial data so as not to bog down PC
    }
  }

/*
|-------------------------------------------------------------------------------------------
|     Function        :   EepromWriteByte
|     Inputs          :   EEPROM Address and Data
|     Outputs         :   None
|     Purpose         :   Write a byte to the EEPROM
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/

void EepromWriteByte(unsigned short EepromAddress, unsigned char EepromData)
  {
  extern unsigned char I2cSlaveAddress;         // I2C slave address
  extern unsigned char I2cBuffer[20];           // I2C application buffer
  extern unsigned char I2cPacketDataSize;       // Number of data bytes for an I2C packet
  extern unsigned char I2cPacketInProgress;      // I2C Packet in progress flag
  unsigned short EepromAddressTemp;             // Temp storage register for address calculation
```

```
  I2cSlaveAddress = I2C_SLAVE_ADDR_EEPROM;      // Address the EEPROM
  I2cPacketDataSize = 3;                        // How many bytes from the I2c buffer to send

  EepromAddressTemp = EepromAddress & 0xFF00;   // Mask top byte of address
  EepromAddressTemp = EepromAddressTemp >> 8;   // Shift it over by 8
  I2cBuffer[0] = (unsigned char)EepromAddressTemp;        // MSB address byte for EEPROM
  I2cBuffer[1] = (unsigned char)(EepromAddress & 0x00FF); // LSB address byte for EEPROM

  I2cBuffer[2] = EepromData;                    // Data byte for EEPROM

  I2cStart(I2C_WRITE);                          // Send out an I2C Start condition for a Write packet
  while(I2cPacketInProgress == TRUE);           // Wait here for I2C packet to complete

  I2cSlaveAckPoll();                            // Go ACK poll the EEPROM and wait for data to be written
  }

/*
|-------------------------------------------------------------------------------------------
|     Function        :   EepromReadByte
|     Inputs          :   EEPROM Address
|     Outputs         :   Byte from the EEPROM
|     Purpose         :   Read a byte to the EEPROM
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/

unsigned char EepromReadByte(unsigned short EepromAddress)
  {
  extern unsigned char I2cSlaveAddress;         // I2C slave address
  extern unsigned char I2cBuffer[20];           // I2C application buffer
  extern unsigned char I2cPacketDataSize;       // Number of data bytes for an I2C packet
  extern unsigned char I2cPacketInProgress;      // I2C Packet in progress flag
  unsigned short EepromAddressTemp;             // Temp register

  I2cSlaveAddress = I2C_SLAVE_ADDR_EEPROM;      // Address the EEPROM
  I2cPacketDataSize = 2;                        // How many bytes from the I2c buffer to send

  EepromAddressTemp = EepromAddress & 0xFF00;   // Mask top byte of address
  EepromAddressTemp = EepromAddressTemp >> 8;   // Shift it over by 8
  I2cBuffer[0] = (unsigned char)EepromAddressTemp;        // MSB address byte for EEPROM
  I2cBuffer[1] = (unsigned char)(EepromAddress & 0x00FF); // LSB address byte for EEPROM

  I2cStart(I2C_WRITE);                          // Send out an I2C Start condition for a Write packet
  while(I2cPacketInProgress == TRUE);           // Wait here for I2C packet to complete

  I2cSlaveAddress = I2C_SLAVE_ADDR_EEPROM;      // Address the EEPROM
  I2cPacketDataSize = 1;                        // How many bytes from the I2c buffer to send

  I2cStart(I2C_READ);                           // Send out an I2C Start condition for a Read packet
  while(I2cPacketInProgress == TRUE);           // Wait here for I2C packet to complete

  return(I2cBuffer[0]);                         // Return the byte read from the EEPROM
  }

/*
|-------------------------------------------------------------------------------------------
|     Function        :   pll_isr
|     Inputs          :   None
|     Outputs         :   None
|     Purpose         :   Once PLL has locked, connect it and use for system clock
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/
#pragma interrupt_handler pll_isr

void pll_isr(void)
  {
  SCB_PLLCON |= 0x02;                           // Connect the PLL
  SCB_PLLFEED = 0xAA;                           // Shadow register copy to enable changes
  SCB_PLLFEED = 0x55;                           // in PLLCON and PLLCFG
  VICIntEnClear = PLL_CLR;                      // Clear PLL interrupt flag
  VICVectAddr = VIC_ACK;                        // Acknowledge Interrupt
  }
```