**Assignment 10:  
I2C EEPROM**

**Cal Poly CPE 329-01**

**Spring 2017**

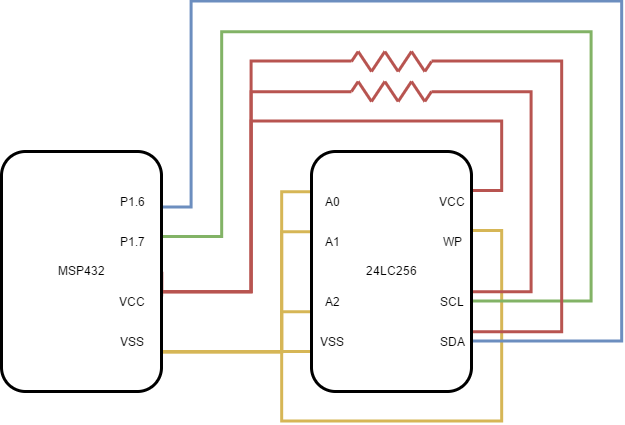
Brendan Baronia

Melinda Ong

Professor Gerfen, Jeffrey B.

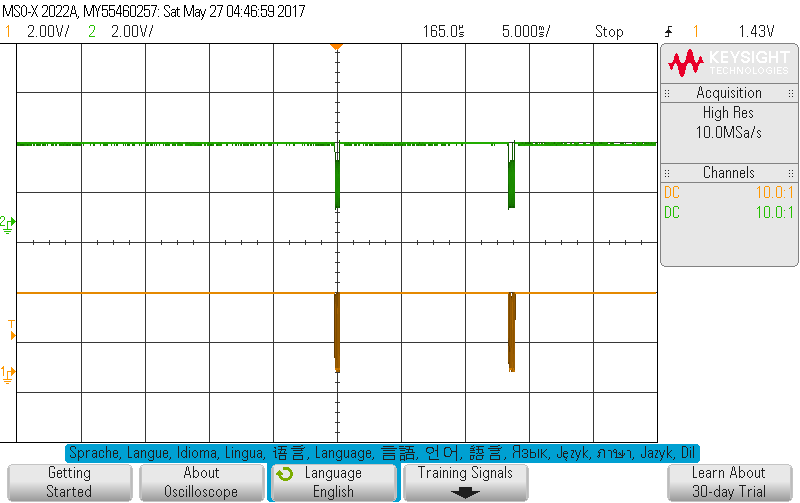
**May 26, 2017**

a.) Scan of the schematic:

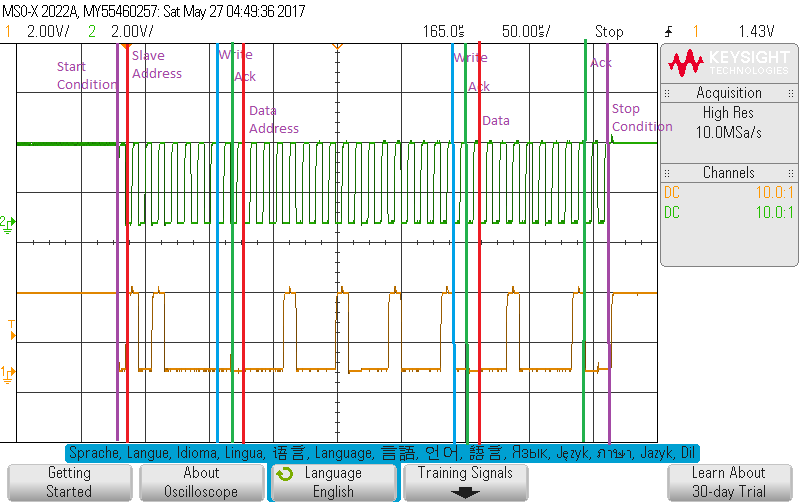


**Schematic 1:** MSP432 interfaced to the Microchip 24LC256 I2C EEPROM

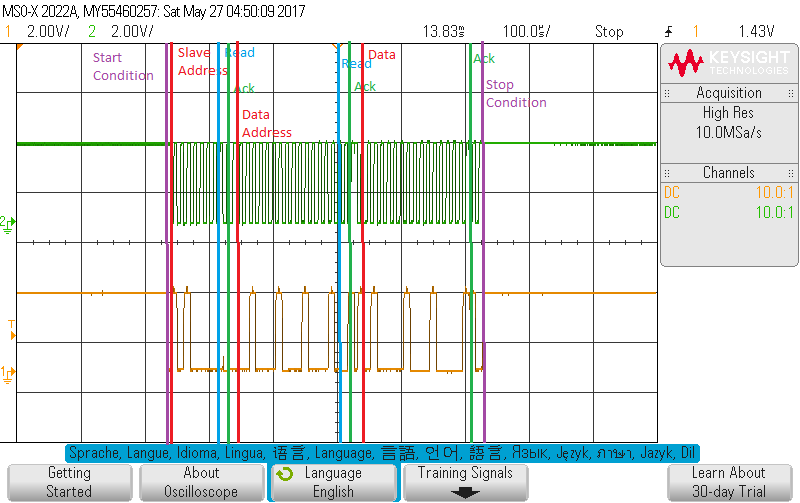
b.) Screenshots of SCL and SDA



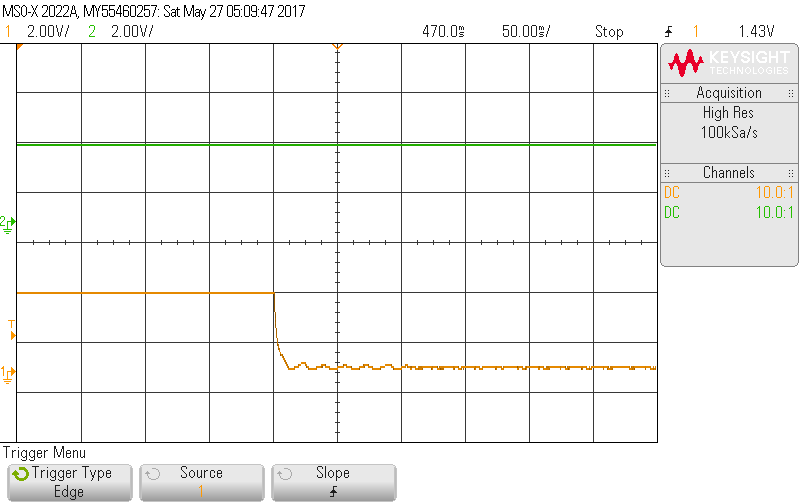
**Figure 1:** SCL Write and Read Together



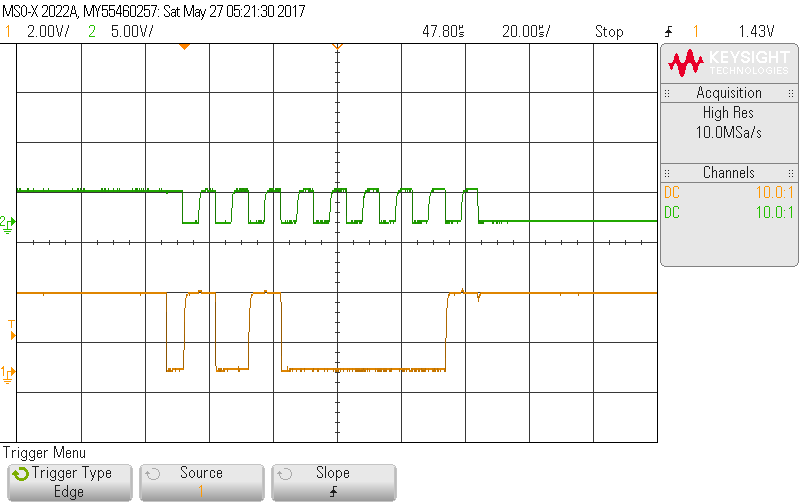
**Figure 2:** SCL Write



**Figure 3:** SCL Read



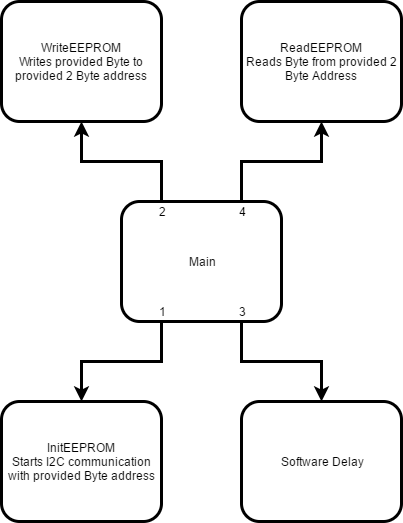
**Figure 4:** Removing SDA pull-up resistor



**Figure 4:** Removing SDA wire to input

c.) Answers to the questions:

***Review the C code provided and draw a detailed flowchart for describing the data flow and sequence of transmissions on the I2C bus for a write and read transmission.***



***Observe what happens when you remove the pull-up resistor.***

Removing the pull-up resistor of the SDA kills the command.

***Observe what happens when you remove the wire connecting the SDA input on the slave. Is the first ACK bit affected? What about the rest of the bits?***

The first ACK bit is not affected, but the last bit is left high because there’s the EEPROM does not acknowledge the rest of the bits,

d.) Reference Code from Professor Hummel:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// CPE 329 - Assignment 9  
//  
// Description: This demo connects an MSP432 to a Microchip 24LC256 EEPROM via  
// the I2C bus. The MSP432 acts as the master and the EEPROM is a slave.  
// The EEPROM uses 3 external connections A2 A1 A0 to set the lower 3 bits of  
// its bus address. This creates a bus address of "1 0 1 0 A2 A1 A0". The code  
// below assumes those three connections are all connected to VSS (Ground) and  
// are logic 0. This gives the EEPROM a bus address of 0x50.  
//  
//  
// /|\ /|\  
// MSP432P401 10k 10k 24LC256 EEPROM  
// master | | Slave  
// ----------------- | | -----------------  
// | P1.6/UCB0SDA|<-|----|->|SDA (5) |  
// | | | | |  
// | | | | |  
// | P1.7/UCB0SCL|<-|------>|SCL (6) |  
// | | | |  
//  
// Paul Hummel  
// Cal Poly  
// May 2017 (created)  
// Built with CCSv7.1  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
#include "msp.h"  
#include <stdint.h>  
  
#define EEPROM\_ADDRESS 0x50  
  
void InitEEPROM(uint8\_t DeviceAddress);  
void WriteEEPROM(uint16\_t MemAddress, uint8\_t MemByte);  
uint8\_t ReadEEPROM(uint16\_t MemAddress);  
  
uint16\_t TransmitFlag = 0;  
  
int main(void)  
{  
 uint32\_t i;  
 uint8\_t value;  
  
 WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Stop watchdog timer  
  
 P2->DIR |= BIT2 | BIT1 | BIT0; // Configure LED2  
 P2->OUT &= ~(BIT2 | BIT1 | BIT0);  
  
 \_\_enable\_irq(); // Enable global interrupt  
  
 InitEEPROM(EEPROM\_ADDRESS);  
  
 WriteEEPROM(0x1122, 0x21);  
  
 for (i = 4000; i > 0; i--); // Delay for EEPROM write cycle (5 ms)  
  
 value = ReadEEPROM(0x1122); // Read value from EEPROM  
  
 P2->OUT |= (value & (BIT2 | BIT1 | BIT0)); // Set LED2 with 3 LSB of value  
  
 \_\_sleep(); // go to lower power mode  
}  
  
////////////////////////////////////////////////////////////////////////////////  
//  
// Initialize I2C bus for communicating with EEPROM.  
//  
////////////////////////////////////////////////////////////////////////////////  
void InitEEPROM(uint8\_t DeviceAddress)  
{  
  
 P1->SEL0 |= BIT6 | BIT7; // Set I2C pins of eUSCI\_B0  
  
 // Enable eUSCIB0 interrupt in NVIC module  
 NVIC->ISER[0] = 1 << ((EUSCIB0\_IRQn) & 31);  
  
 // Configure USCI\_B0 for I2C mode  
 EUSCI\_B0->CTLW0 |= EUSCI\_A\_CTLW0\_SWRST; // Software reset enabled  
 EUSCI\_B0->CTLW0 = EUSCI\_A\_CTLW0\_SWRST | // Remain eUSCI in reset mode  
 EUSCI\_B\_CTLW0\_MODE\_3 | // I2C mode  
 EUSCI\_B\_CTLW0\_MST | // Master mode  
 EUSCI\_B\_CTLW0\_SYNC | // Sync mode  
 EUSCI\_B\_CTLW0\_SSEL\_\_SMCLK; // SMCLK  
  
 EUSCI\_B0->BRW = 30; // baudrate = SMCLK / 30 = 100kHz  
 EUSCI\_B0->I2CSA = DeviceAddress; // Slave address  
 EUSCI\_B0->CTLW0 &= ~EUSCI\_A\_CTLW0\_SWRST; // Release eUSCI from reset  
  
 EUSCI\_B0->IE |= EUSCI\_A\_IE\_RXIE | // Enable receive interrupt  
 EUSCI\_A\_IE\_TXIE;  
}  
  
////////////////////////////////////////////////////////////////////////////////  
//  
// Function that writes a single byte to the EEPROM.  
//  
// MemAddress - 2 byte address specifies the address in the EEPROM memory  
// MemByte - 1 byte value that is stored in the EEPROM  
//  
// Procedure :  
// start  
// transmit address+W (control+0) -> ACK (from EEPROM)  
// transmit data (high address) -> ACK (from EEPROM)  
// transmit data (low address) -> ACK (from EEPROM)  
// transmit data (data) -> ACK (from EEPROM)  
// stop  
//  
////////////////////////////////////////////////////////////////////////////////  
void WriteEEPROM(uint16\_t MemAddress, uint8\_t MemByte)  
{  
 uint8\_t HiAddress;  
 uint8\_t LoAddress;  
  
 HiAddress = MemAddress >> 8;  
 LoAddress = MemAddress & 0xFF;  
  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TR; // Set transmit mode (write)  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TXSTT; // I2C start condition  
  
 while (!TransmitFlag); // Wait for EEPROM address to transmit  
 TransmitFlag = 0;  
   
 EUSCI\_B0 -> TXBUF = HiAddress; // Send the high byte of the memory address  
  
 while (!TransmitFlag); // Wait for the transmit to complete  
 TransmitFlag = 0;  
   
 EUSCI\_B0 -> TXBUF = LoAddress; // Send the high byte of the memory address  
  
 while (!TransmitFlag); // Wait for the transmit to complete  
 TransmitFlag = 0;  
  
 EUSCI\_B0 -> TXBUF = MemByte; // Send the byte to store in EEPROM  
  
 while (!TransmitFlag); // Wait for the transmit to complete  
 TransmitFlag = 0;  
  
 EUSCI\_B0 -> CTLW0 |= EUSCI\_B\_CTLW0\_TXSTP; // I2C stop condition  
}  
  
////////////////////////////////////////////////////////////////////////////////  
//  
// Function that reads a single byte from the EEPROM.  
//  
// MemAddress - 2 byte address specifies the address in the EEPROM memory  
// ReceiveByte - 1 byte value that is received from the EEPROM  
//  
// Procedure :  
// start  
// transmit address+W (control+0) -> ACK (from EEPROM)  
// transmit data (high address) -> ACK (from EEPROM)  
// transmit data (low address) -> ACK (from EEPROM)  
// start  
// transmit address+R (control+1) -> ACK (from EEPROM)  
// transmit data (data) -> NACK (from MSP432)  
// stop  
//  
////////////////////////////////////////////////////////////////////////////////  
uint8\_t ReadEEPROM(uint16\_t MemAddress)  
{  
 uint8\_t ReceiveByte;  
 uint8\_t HiAddress;  
 uint8\_t LoAddress;  
  
 HiAddress = MemAddress >> 8;  
 LoAddress = MemAddress & 0xFF;  
  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TR; // Set transmit mode (write)  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TXSTT; // I2C start condition  
  
 while (!TransmitFlag); // Wait for EEPROM address to transmit  
 TransmitFlag = 0;  
  
 EUSCI\_B0 -> TXBUF = HiAddress; // Send the high byte of the memory address  
  
 while (!TransmitFlag); // Wait for the transmit to complete  
 TransmitFlag = 0;  
  
 EUSCI\_B0 -> TXBUF = LoAddress; // Send the low byte of the memory address  
  
 while (!TransmitFlag); // Wait for the transmit to complete  
 TransmitFlag = 0;  
  
 EUSCI\_B0->CTLW0 &= ~EUSCI\_B\_CTLW0\_TR; // Set receive mode (read)  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TXSTT; // I2C start condition (restart)  
  
 // Wait for start to be transmitted  
 while ((EUSCI\_B0->CTLW0 & EUSCI\_B\_CTLW0\_TXSTT));  
  
 // set stop bit to trigger after first byte  
 EUSCI\_B0->CTLW0 |= EUSCI\_B\_CTLW0\_TXSTP;  
  
 while (!TransmitFlag); // Wait to receive a byte  
 TransmitFlag = 0;  
  
 ReceiveByte = EUSCI\_B0->RXBUF; // Read byte from the buffer  
  
 return ReceiveByte;  
}  
  
////////////////////////////////////////////////////////////////////////////////  
//  
// I2C Interrupt Service Routine  
//  
////////////////////////////////////////////////////////////////////////////////  
void EUSCIB0\_IRQHandler(void)  
{  
 if (EUSCI\_B0->IFG & EUSCI\_B\_IFG\_TXIFG0) // Check if transmit complete  
 {  
 EUSCI\_B0->IFG &= ~ EUSCI\_B\_IFG\_TXIFG0; // Clear interrupt flag  
 TransmitFlag = 1; // Set global flag  
 }  
  
 if (EUSCI\_B0->IFG & EUSCI\_B\_IFG\_RXIFG0) // Check if receive complete  
 {  
 EUSCI\_B0->IFG &= ~ EUSCI\_B\_IFG\_RXIFG0; // Clear interrupt flag  
 TransmitFlag = 1; // Set global flag  
 }  
}