ex.c.c

```

int main(void)
{
  WDTCTL = WDTPW+WDTHOLD;                   // Stop watchdog timer
  P7SEL |= 0x03;                            // Port select XT1

  do
  {
    UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG);
                                            // Clear XT2,XT1,DCO fault flags
    SFRIFG1 &= ~OFIFG;                      // Clear fault flags
    __delay_cycles(100000);                 // Delay for Osc to stabilize
  }while (SFRIFG1&OFIFG);                   // Test oscillator fault flag

  P1OUT = 0x000;                            // P1.0/1 setup for LED output
  P1DIR |= BIT0+BIT1;                       //
  P3SEL |= BIT4+BIT5;                       // P3.4,5 UART option select

  UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
  UCA0CTL1 |= UCSSEL_1;                     // CLK = ACLK
  UCA0BR0 = 0x03;                           // 32k/9600 - 3.41
  UCA0BR1 = 0x00;                           //
  UCA0MCTL = 0x06;                          // Modulation
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  UCA0IE |= UCTXIE + UCRXIE;                // Enable USCI_A0 TX/RX interrupt

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupts enabled
  __no_operation();                         // For debugger
}


#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  unsigned char tx_char;

    switch(__even_in_range(UCA0IV,4))
  {
    case 0: break;                          // Vector 0 - no interrupt
    case 2:                                 // Vector 2 - RXIFG
      P1OUT = UCA0RXBUF;                    // RXBUF1 to TXBUF1
      break;
    case 4:                                 // Vector 4 - TXIFG
      __delay_cycles(5000);                 // Add small gap between TX'ed bytes
      tx_char = P1IN;
      tx_char = tx_char >> 4;
      UCA0TXBUF = tx_char;                  // Transmit character
      break;
    default: break;
  }
}



int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P3SEL = 0x30;                             // P3.4,5 = USCI_A0 TXD/RXD
  UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 6;                              // 1MHz 9600 (see User's Guide)
  UCA0BR1 = 0;                              // 1MHz 9600
  UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;   // Modln UCBRSx=0, UCBRFx=0,
                                            // over sampling
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, interrupts enabled
  __no_operation();                         // For debugger
}

// Echo back RXed character, confirm TX buffer is ready first
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    while (!(UCA0IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;                  // TX -> RXed character
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
```

```
  }
}



#include <msp430.h>

unsigned char MST_Data,SLV_Data;

int main(void)
{
  WDTCTL = WDTPW+WDTHOLD;                   // Stop watchdog timer

  P1OUT |= 0x02;                            // Set P1.0 for LED
                                            // Set P1.1 for slave reset
  P1DIR |= 0x03;                            // Set P1.0-2 to output direction
  P3SEL |= 0x31;                            // P3.5,4,0 option select

  UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
  UCA0CTL0 |= UCMST+UCSYNC+UCCKPL+UCMSB;    // 3-pin, 8-bit SPI master
                                            // Clock polarity high, MSB
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 0x02;                           // /2
  UCA0BR1 = 0;                              //
  UCA0MCTL = 0;                             // No modulation
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

  P1OUT &= ~0x02;                           // Now with SPI signals initialized,
  P1OUT |= 0x02;                            // reset slave

  __delay_cycles(100);                      // Wait for slave to initialize

  MST_Data = 0x01;                          // Initialize data values
  SLV_Data = 0x00;                          //

  while (!(UCA0IFG&UCTXIFG));               // USCI_A0 TX buffer ready?
  UCA0TXBUF = MST_Data;                     // Transmit first character

  __bis_SR_register(LPM0_bits + GIE);       // CPU off, enable interrupts
}

#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
```

```
{
  switch(__even_in_range(UCA0IV,4))
  {
    case 0: break;                          // Vector 0 - no interrupt
    case 2:                                 // Vector 2 - RXIFG
      while (!(UCA0IFG&UCTXIFG));           // USCI_A0 TX buffer ready?

      if (UCA0RXBUF==SLV_Data)              // Test for correct character RX'd
        P1OUT |= 0x01;                      // If correct, light LED
      else
        P1OUT &= ~0x01;                     // If incorrect, clear LED

      MST_Data++;                           // Increment data
      SLV_Data++;
      UCA0TXBUF = MST_Data;                 // Send next value

      __delay_cycles(40);                   // Add time between transmissions to
                                            // make sure slave can process
information
      break;
    case 4: break;                          // Vector 4 - TXIFG
    default: break;
  }
}


#include <msp430.h>

unsigned char RXData;
unsigned char RXCompare;

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1OUT &= ~0x01;                           // P1.0 = 0
  P1DIR |= 0x01;                            // P1.0 output
  P3SEL |= 0x06;                            // Assign I2C pins to USCI_B0
  UCB0CTL1 |= UCSWRST;                      // Enable SW reset
  UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;     // I2C Master, synchronous mode
  UCB0CTL1 = UCSSEL_2 + UCSWRST;            // Use SMCLK
  UCB0BR0 = 12;                             // fSCL = SMCLK/12 = ~100kHz
  UCB0BR1 = 0;
  UCB0I2CSA = 0x48;                        // Slave Address is 048h
  UCB0CTL1 &= ~UCSWRST;                     // Clear SW reset, resume operation
  UCB0IE |= UCRXIE;                         // Enable RX interrupt
  RXCompare = 0x0;                          // Used to check incoming data

  while (1)
  {
    while (UCB0CTL1 & UCTXSTP);             // Ensure stop condition got sent
```

```
    UCB0CTL1 |= UCTXSTT;                    // I2C start condition
    while(UCB0CTL1 & UCTXSTT);              // Start condition sent?
    UCB0CTL1 |= UCTXSTP;                    // I2C stop condition

    __bis_SR_register(LPM0_bits + GIE);     // Enter LPM0, enable interrupts
    __no_operation();                       // For debugger

    if (RXData != RXCompare)                // Trap CPU if wrong
    {
      P1OUT |= 0x01;                        // P1.0 = 1
    }

    RXCompare++;                            // Increment correct RX value
  }
}

// USCI_B0 Data ISR
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_B0_VECTOR))) USCI_B0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCB0IV,12))
  {
  case  0: break;                           // Vector  0: No interrupts
  case  2: break;                           // Vector  2: ALIFG
  case  4: break;                           // Vector  4: NACKIFG
  case  6: break;                           // Vector  6: STTIFG
  case  8: break;                           // Vector  8: STPIFG
  case 10:                                  // Vector 10: RXIFG
    RXData = UCB0RXBUF;                     // Get RX data
    __bic_SR_register_on_exit(LPM0_bits);   // Exit active CPU
    break;
  case 12: break;                           // Vector 12: TXIFG
  default: break;
  }
}



#include "msp430.h"
#define SMCLK_115200     0
#define SMCLK_9600       1
#define ACLK_9600        2

#define UART_MODE        SMCLK_115200//ACLK_9600//


void initUART()
{
    // Configure USCI_A0 for UART mode
    UCA0CTLW0 = UCSWRST;                        // Put eUSCI in reset
#if UART_MODE == SMCLK_115200

    UCA0CTLW0 |= UCSSEL__SMCLK;                 // CLK = SMCLK
    // Baud Rate calculation
    // 16000000/(16*115200) = 8.6805
    // Fractional portion = 0.6805
    // Use Table 24-5 in Family User Guide
    UCA0BR0 = 8;                                 // 16000000/16/9600
    UCA0BR1 = 0x00;
    UCA0MCTL |= UCOS16 | UCBRF_11 | UCBRS_0;

#elif UART_MODE == SMCLK_9600

    UCA0CTLW0 |= UCSSEL__SMCLK;                 // CLK = SMCLK
    // Baud Rate calculation
    // 16000000/(16*9600) = 104.1667
    // Fractional portion = 0.1667
    // Use Table 24-5 in Family User Guide
    UCA0BR0 = 104;                              // 16000000/16/9600
    UCA0BR1 = 0x00;
    UCA0MCTL |= UCOS16 | UCBRF_3 | UCBRS_0;

#elif UART_MODE == ACLK_9600

    UCA0CTLW0 |= UCSSEL__ACLK;                 // CLK = ACLK
    // Baud Rate calculation
    // 32768/(9600) = 3.4133
    // Fractional portion = 0.4133
    // Use Table 24-5 in Family User Guide
    UCA0BR0 = 3;                               // 32768/9600
    UCA0BR1 = 0x00;
    UCA0MCTL |= UCBRS_3;    //0x0300 is UCBRSx = 0x03

#else
    # error "Please specify baud rate to 115200 or 9600"
#endif

    UCA0CTLW0 &= ~UCSWRST;                      // Initialize eUSCI
    UCA0IE |= UCRXIE;                           // Enable USCI_A0 RX interrupt
}

//******************************************************************************
// Device Initialization ******************************************************
//******************************************************************************

```

```
void initGPIO()
{
    P3SEL = BIT4 | BIT5;                          // P3.4,5 = USCI_A0 TXD/RXD
    P7SEL |= BIT0 | BIT1;                           // Select XT1
}

void initClockTo16MHz()
{
    UCSCTL3 |= SELREF_2;                        // Set DCO FLL reference = REFO
    UCSCTL4 |= SELA_0;                          // Set ACLK = XT1CLK
    __bis_SR_register(SCG0);                    // Disable the FLL control loop
    UCSCTL0 = 0x0000;                           // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_5;                        // Select DCO range 16MHz operation
    UCSCTL2 = FLLD_0 + 487;                     // Set DCO Multiplier for 16MHz
                                                // (N + 1) * FLLRef = Fdco
                                                // (487 + 1) * 32768 = 16MHz
                                                // Set FLL Div = fDCOCLK
    __bic_SR_register(SCG0);                    // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // UG for optimization.
    // 32 x 32 x 16 MHz / 32,768 Hz = 500000 = MCLK cycles for DCO to settle
    __delay_cycles(500000);//
    // Loop until XT1,XT2 & DCO fault flag is cleared
    do
    {
        UCSCTL7 &= ~(XT2OFFG + XT1LFOFFG + DCOFFG); // Clear XT2,XT1,DCO fault
flags
        SFRIFG1 &= ~OFIFG;                          // Clear fault flags
    }while (SFRIFG1&OFIFG);                         // Test oscillator fault flag
}

bool increaseVCoreToLevel2()
{
    uint8_t level = 2;
    uint8_t actlevel;
    bool status = true;

    //Set Mask for Max. level
    level &= PMMCOREV_3;

    //Get actual VCore
    actlevel = PMMCTL0 & PMMCOREV_3;

    //step by step increase or decrease
    while((level != actlevel) && (status == true))
    {
        if(level > actlevel)
        {
            status = setVCoreUp(++actlevel);
        }
    }

    return (status);
}



int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;                 // Stop Watchdog

  initGPIO();
  increaseVCoreToLevel2();
  initClockTo16MHz();
  initUART();

#if UART_MODE == ACLK_9600
    __bis_SR_register(LPM3_bits + GIE);       // Since ACLK is source, enter LPM3,
interrupts enabled
#else
    __bis_SR_register(LPM0_bits + GIE);       // Since SMCLK is source, enter LPM0,
interrupts enabled
#endif
  __no_operation();                         // For debugger
}

// UART rx int
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    while (!(UCA0IFG & UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;                  // TX -> RXed character
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}

```