Name: Junzhe Wu NetID: junzhew3

2. Answer:

(1) 0 and 1023.

(2) ADC10BUSY

Code:

```
#include "msp430g2553.h"
#include "UART.h"

void print_every(int rate);

char newprint = 0;
long NumOn = 0;
long NumOff = 0;
int statevar = 1;
int timecheck = 0;

int ADC_value;

void main(void) {

    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

    // Initializing ADC10
    ADC10CTL0 = SREF_0 + ADC10SHT_1 + ADC10ON + ADC10IE; // Vr+ = Vcc, Vr-
= Vss, 8 x ADC10CLKS, ADC10 on, ADC Interrupt enable
    ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_0 + ADC10SSEL_0 + CONSEQ_0; //
P2: Input channel A0, ADC10SC, Clock divide /1, Clock source = ADC10OSC,
Single-channel-single-conversion
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_3 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /4
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_7 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /8

    ADC10AE0 = 0x01;  // P2: Enable A0 ADC channel

    ADC10CTL0 |= ENC; // ADC10 enabled

    if (CALBC1_16MHZ ==0xFF || CALDCO_16MHZ == 0xFF) while(1);

    DCOCTL = CALDCO_16MHZ;    // Set uC to run at approximately 16 Mhz
    BCSCTL1 = CALBC1_16MHZ;

    // Initialize Port 1
    P1SEL &= ~0x01;  // See page 42 and 43 of the G2553's datasheet, It
```

```
shows that when both P1SEL and P1SEL2 bits are zero
    P1SEL2 &= ~0x01; // the corresponding pin is set as a I/O pin.
Datasheet:
http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/msp430g2553datasheet.pdf
    P1REN = 0x0;  // No resistors enabled for Port 1
    P1DIR |= 0xf0; // Set P1,4,P1.5, P1.6, and P1.7 to output to drive LED on LaunchPad board.
    P1OUT &= ~0x01;  // Initially set P1.0 to 0



    // Timer A Config
    TACCTL0 = CCIE;            // Enable Periodic interrupt
    TACCR0 = 16000;               // period = 1ms
    TACTL = TASSEL_2 + MC_1; // source SMCLK, up mode


    Init_UART(115200,1);   // Initialize UART for 115200 baud serial communication

    _BIS_SR(GIE);      // Enable global interrupt


    while(1) {  // Low priority Slow computation items go inside this while loop.  Very few (if anyt) items in the HWs will go inside this while loop

// for use if you want to use a method of receiving a string of chars over the UART see USCI0RX_ISR below
//      if(newmsg) {
//          newmsg = 0;
//      }

       // The newprint variable is set to 1 inside the function "print_every(rate)" at the given rate
       if ( (newprint == 1) && (senddone == 1) )  { // senddone is set to 1 after UART transmission is complete

           // UART_printf is called every 0.25s
           UART_printf("A0: %d \n\r", ADC_value);
           newprint = 0;
       }

    }
```

```
}


// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    timecheck++; // Keep track of time for main while loop.
    ADC10CTL0 |= ENC + ADC10SC; // Trigger ADC10 every 1  millisecond
    if (timecheck == 250) {
        timecheck = 0;
        newprint = 1; // do the print job
    }

}

// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have
completed
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {

    ADC_value = (int)ADC10MEM; // save conversion value to adc_value

    switch (statevar) {
        case 1:  //start state, all four LEDs off

            P1OUT &= ~0xf0;          // Clear P1.4 to P1.7 LED off
            if (ADC10MEM < 256) {
                statevar = 1;  // stays the same.
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 2:  // two LEDs are on
            P1OUT &= ~0xf0; // Clear P1.4 to P1.7 LED off
            P1OUT |= 0x30;  // Set P1.4 and P1.5 LED on
            if (ADC10MEM < 256) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
```

```
                statevar = 2;  // stays the same.
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 3:  // all four LEDs on

            P1OUT |= 0xf0;  // Set all four LEDs on
            if (ADC10MEM < 256) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                statevar = 3;  // stays the same.
            }
            break;
    }

}



// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another
character)
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void) {

    if(IFG2&UCA0TXIFG) {       // USCI_A0 requested TX interrupt
        if(printf_flag) {
            if (currentindex == txcount) {
                senddone = 1;
                printf_flag = 0;
                IFG2 &= ~UCA0TXIFG;
            } else {
                UCA0TXBUF = printbuff[currentindex];
                currentindex++;
            }
        } else if(UART_flag) {
            if(!donesending) {
                UCA0TXBUF = txbuff[txindex];
                if(txbuff[txindex] == 255) {
```

```
                donesending = 1;
                txindex = 0;
            }
            else txindex++;
        }
    } else {  // interrupt after sendchar call so just set senddone flag since only one char is sent
        senddone = 1;
    }

    IFG2 &= ~UCA0TXIFG;
}

if(IFG2&UCB0TXIFG) {   // USCI_B0 requested TX interrupt (UCB0TXBUF is empty)

    IFG2 &= ~UCB0TXIFG;   // clear IFG
}
}


// USCI Receive ISR - Called when shift register has been transferred to RXBUF
// Indicates completion of TX/RX operation
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {

if(IFG2&UCB0RXIFG) {  // USCI_B0 requested RX interrupt (UCB0RXBUF is full)

    IFG2 &= ~UCB0RXIFG;   // clear IFG
}

if(IFG2&UCA0RXIFG) {  // USCI_A0 requested RX interrupt (UCA0RXBUF is full)

//   Uncomment this block of code if you would like to use this COM protocol that uses 253 as STARTCHAR and 255 as STOPCHAR
/*      if(!started) { // Haven't started a message yet
            if(UCA0RXBUF == 253) {
                started = 1;
                newmsg = 0;
            }
        }
```

```
        else {  // In process of receiving a message
            if((UCA0RXBUF != 255) && (msgindex < (MAX_NUM_FLOATS*5))) {
                rxbuff[msgindex] = UCA0RXBUF;

                msgindex++;
            } else {    // Stop char received or too much data received
                if(UCA0RXBUF == 255) { // Message completed
                    newmsg = 1;
                    rxbuff[msgindex] = 255;    // "Null"-terminate the array
                }
                started = 0;
                msgindex = 0;
            }
        }
*/

        IFG2 &= ~UCA0RXIFG;
    }

}

// This function takes care of all the timing for printing to UART
// Rate determined by how often the function is called in Timer ISR
int print_timecheck = 0;
void print_every(int rate) {
    if (rate < 15) {
        rate = 15;
    }
    if (rate > 10000) {
        rate = 10000;
    }
    print_timecheck++;
    if (print_timecheck == rate) {
        print_timecheck = 0;
        newprint = 1;
    }

}
```

3. Code:

```
#include "msp430g2553.h"
#include "UART.h"

void print_every(int rate);
```

```
char newprint = 0;
long NumOn = 0;
long NumOff = 0;
int statevar = 1;
int timecheck = 0;

int ADC_value;

void main(void) {

    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

    // Initializing ADC10
    ADC10CTL0 = SREF_0 + ADC10SHT_1 + ADC10ON + ADC10IE; // Vr+ = Vcc, Vr-
= Vss, 8 x ADC10CLKS, ADC10 on, ADC Interrupt enable
    ADC10CTL1 = INCH_3 + SHS_0 + ADC10DIV_0 + ADC10SSEL_0 + CONSEQ_0; //
P3: Input channel A3, ADC10SC, Clock divide /1, Clock source = ADC10OSC,
Single-channel-single-conversion
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_3 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /4
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_7 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /8

    ADC10AE0 = 0x08;  // P3: Enable A3 ADC channel

    ADC10CTL0 |= ENC; // ADC10 enabled

    if (CALBC1_16MHZ ==0xFF || CALDCO_16MHZ == 0xFF) while(1);

    DCOCTL = CALDCO_16MHZ;    // Set uC to run at approximately 16 Mhz
    BCSCTL1 = CALBC1_16MHZ;

    // Initialize Port 1
    P1SEL &= ~0x01;  // See page 42 and 43 of the G2553's datasheet, It
shows that when both P1SEL and P1SEL2 bits are zero
    P1SEL2 &= ~0x01; // the corresponding pin is set as a I/O pin.
Datasheet:
http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/msp430g255
3datasheet.pdf
    P1REN = 0x0;  // No resistors enabled for Port 1
    P1DIR |= 0xf0; // Set P1,4,P1.5, P1.6, and P1.7 to output to drive LED
on LaunchPad board.
    P1OUT &= ~0x01;  // Initially set P1.0 to 0
```

```
    // Timer A Config
    TACCTL0 = CCIE;             // Enable Periodic interrupt
    TACCR0 = 16000;                // period = 1ms
    TACTL = TASSEL_2 + MC_1; // source SMCLK, up mode


    Init_UART(115200,1);    // Initialize UART for 115200 baud serial
communication

    _BIS_SR(GIE);       // Enable global interrupt


    while(1) {  // Low priority Slow computation items go inside this while
loop.  Very few (if anyt) items in the HWs will go inside this while loop

// for use if you want to use a method of receiving a string of chars over
the UART see USCI0RX_ISR below
//      if(newmsg) {
//          newmsg = 0;
//      }

        // The newprint variable is set to 1 inside the function
"print_every(rate)" at the given rate
        if ( (newprint == 1) && (senddone == 1) )  { // senddone is set to 1
after UART transmission is complete

            // UART_printf is called every 0.25s
            UART_printf("A3: %d \n\r", ADC_value);
            newprint = 0;
        }

    }
}


// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    timecheck++; // Keep track of time for main while loop.
    ADC10CTL0 |= ENC + ADC10SC; // Trigger ADC10 every 1  millisecond
    if (timecheck == 250) {
```

```
        timecheck = 0;
        newprint = 1; // do the print job
    }

}

// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have
completed
int timecheck1 = 0;
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {

    ADC_value = (int)ADC10MEM; // save conversion value to adc_value

    switch (statevar) {
        case 1:  //start state, all four LEDs off, the dark state
            P1OUT &= ~0xf0;          // Clear P1.4 to P1.7 LED off

            if (ADC10MEM < 100) {
                timecheck1++; // if the next state is still 1, count
timechek1
            }
            else{
                timecheck1 = 0; // otherwise set timechek1 to 0 since the
state 1 is not continued
            }

            if (timecheck1 == 2000) {
                statevar = 3; // When the lights have been turned off for 2
seconds, turn on all the LEDs.
                break;
            }

            if (ADC10MEM < 100) {
                statevar = 1;  // stays the same.
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 2:  // all four LEDs off, the normal state
```

```
            P1OUT &= ~0xf0;          // Clear P1.4 to P1.7 LED off
            if (ADC10MEM < 100) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // stays the same.
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 3:  // all four LEDs on
            P1OUT |= 0xf0;  // Set all four LEDs on
            if ((timecheck1 == 2000)&& (ADC10MEM < 100) ) {
                statevar = 3; // When the lights have been turned off for 2
seconds and it is still dark, turn on all the LEDs.
                break;
            }
            else if (ADC10MEM < 100) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
                timecheck1 = 0;
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                timecheck1 = 0;
                statevar = 3;  // stays the same.
            }
            break;
    }

}

// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another
character)
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void) {

    if(IFG2&UCA0TXIFG) {        // USCI_A0 requested TX interrupt
        if(printf_flag) {
            if (currentindex == txcount) {
                senddone = 1;
                printf_flag = 0;
```

```
            IFG2 &= ~UCA0TXIFG;
        } else {
            UCA0TXBUF = printbuff[currentindex];
            currentindex++;
        }
    } else if(UART_flag) {
        if(!donesending) {
            UCA0TXBUF = txbuff[txindex];
            if(txbuff[txindex] == 255) {
                donesending = 1;
                txindex = 0;
            }
            else txindex++;
        }
    } else {  // interrupt after sendchar call so just set senddone flag since only one char is sent
        senddone = 1;
    }

    IFG2 &= ~UCA0TXIFG;
}

if(IFG2&UCB0TXIFG) {    // USCI_B0 requested TX interrupt (UCB0TXBUF is empty)

    IFG2 &= ~UCB0TXIFG;   // clear IFG
}
}


// USCI Receive ISR - Called when shift register has been transferred to RXBUF
// Indicates completion of TX/RX operation
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {

    if(IFG2&UCB0RXIFG) {  // USCI_B0 requested RX interrupt (UCB0RXBUF is full)

        IFG2 &= ~UCB0RXIFG;   // clear IFG
    }

    if(IFG2&UCA0RXIFG) {  // USCI_A0 requested RX interrupt (UCA0RXBUF is full)
```

```
//    Uncomment this block of code if you would like to use this COM
protocol that uses 253 as STARTCHAR and 255 as STOPCHAR
/*      if(!started) {  // Haven't started a message yet
            if(UCA0RXBUF == 253) {
                started = 1;
                newmsg = 0;
            }
        }
        else {  // In process of receiving a message
            if((UCA0RXBUF != 255) && (msgindex < (MAX_NUM_FLOATS*5))) {
                rxbuff[msgindex] = UCA0RXBUF;

                msgindex++;
            } else {    // Stop char received or too much data received
                if(UCA0RXBUF == 255) {  // Message completed
                    newmsg = 1;
                    rxbuff[msgindex] = 255; // "Null"-terminate the array
                }
                started = 0;
                msgindex = 0;
            }
        }
*/

        IFG2 &= ~UCA0RXIFG;
    }

}

// This function takes care of all the timing for printing to UART
// Rate determined by how often the function is called in Timer ISR
int print_timecheck = 0;
void print_every(int rate) {
    if (rate < 15) {
        rate = 15;
    }
    if (rate > 10000) {
        rate = 10000;
    }
    print_timecheck++;
    if (print_timecheck == rate) {
        print_timecheck = 0;
        newprint = 1;
```

```
    }

}
```

4. Code:

```
#include "msp430g2553.h"
#include "UART.h"

void print_every(int rate);

char newprint = 0;
long NumOn = 0;
long NumOff = 0;
int statevar = 1;
int timecheck = 0;

int ADC_value;

void main(void) {

    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

    // Initializing ADC10
    ADC10CTL0 = SREF_0 + ADC10SHT_1 + ADC10ON + ADC10IE; // Vr+ = Vcc, Vr-
= Vss, 8 x ADC10CLKS, ADC10 on, ADC Interrupt enable
    ADC10CTL1 = INCH_3 + SHS_0 + ADC10DIV_0 + ADC10SSEL_0 + CONSEQ_0; //
P3: Input channel A3, ADC10SC, Clock divide /1, Clock source = ADC10OSC,
Single-channel-single-conversion
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_3 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /4
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_7 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /8

    ADC10AE0 = 0x08;  // P3: Enable A3 ADC channel

    ADC10CTL0 |= ENC; // ADC10 enabled

    if (CALBC1_16MHZ ==0xFF || CALDCO_16MHZ == 0xFF) while(1);

    DCOCTL = CALDCO_16MHZ;    // Set uC to run at approximately 16 Mhz
    BCSCTL1 = CALBC1_16MHZ;

    // Initialize Port 1
    P1SEL &= ~0x01;  // See page 42 and 43 of the G2553's datasheet, It
shows that when both P1SEL and P1SEL2 bits are zero
```

```
    P1SEL2 &= ~0x01; // the corresponding pin is set as a I/O pin.
Datasheet:
http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/msp430g255
3datasheet.pdf
    P1REN = 0x0;  // No resistors enabled for Port 1
    P1DIR |= 0xf0; // Set P1,4,P1.5, P1.6, and P1.7 to output to drive LED
on LaunchPad board.
    P1OUT &= ~0x01;  // Initially set P1.0 to 0

    // Initialize Port 2 - TA1.1 PWM Output
    P2DIR |= 0x04;   // P2.2 output
    P2SEL |= 0x04;   // P2.2 TA1.1 option
    P2SEL2 &= ~0x04; // P2.2 TA1.1 option

    // PWM Config
    TA1CCR0 = 1600;             // PWM Period
    TA1CCTL1 = OUTMOD_7;         // TA1CCR1 reset/set
    TA1CCR1 = 0;                // TA1CCR1 PWM duty cycle
    TA1CTL = TASSEL_2 + MC_1;    // SMCLK, up mode

    // Timer A Config
    TACCTL0 = CCIE;            // Enable Periodic interrupt
    TACCR0 = 16000;               // period = 1ms
    TACTL = TASSEL_2 + MC_1; // source SMCLK, up mode



    Init_UART(115200,1);    // Initialize UART for 115200 baud serial
communication

    _BIS_SR(GIE);       // Enable global interrupt


    while(1) {  // Low priority Slow computation items go inside this while
loop.  Very few (if anyt) items in the HWs will go inside this while loop

// for use if you want to use a method of receiving a string of chars over
the UART see USCI0RX_ISR below
//      if(newmsg) {
//          newmsg = 0;
//      }

       // The newprint variable is set to 1 inside the function
"print_every(rate)" at the given rate
```

```
        if ( (newprint == 1) && (senddone == 1) )  { // senddone is set to 1
after UART transmission is complete

            // UART_printf is called every 0.25s
            UART_printf("A0: %d \n\r", ADC_value);
            newprint = 0;
        }

    }
}


// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    timecheck++; // Keep track of time for main while loop.
    ADC10CTL0 |= ENC + ADC10SC; // Trigger ADC10 every 1  millisecond
    if (timecheck == 250) {
        timecheck = 0;
        newprint = 1; // do the print job
    }

}

// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have
completed
int timecheck1 = 0;
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {

    ADC_value = (int)ADC10MEM; // save conversion value to adc_value
    TA1CCR1 = (ADC_value*1600L)/1023;
}

// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another
character)
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void) {

    if(IFG2&UCA0TXIFG) {        // USCI_A0 requested TX interrupt
        if(printf_flag) {
            if (currentindex == txcount) {
                senddone = 1;
```

```
                printf_flag = 0;
                IFG2 &= ~UCA0TXIFG;
            } else {
                UCA0TXBUF = printbuff[currentindex];
                currentindex++;
            }
        } else if(UART_flag) {
            if(!donesending) {
                UCA0TXBUF = txbuff[txindex];
                if(txbuff[txindex] == 255) {
                    donesending = 1;
                    txindex = 0;
                }
                else txindex++;
            }
        } else {  // interrupt after sendchar call so just set senddone flag
since only one char is sent
            senddone = 1;
        }

        IFG2 &= ~UCA0TXIFG;
    }

    if(IFG2&UCB0TXIFG) {    // USCI_B0 requested TX interrupt (UCB0TXBUF is
empty)

        IFG2 &= ~UCB0TXIFG;   // clear IFG
    }
}


// USCI Receive ISR - Called when shift register has been transferred to
RXBUF
// Indicates completion of TX/RX operation
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {

    if(IFG2&UCB0RXIFG) {  // USCI_B0 requested RX interrupt (UCB0RXBUF is
full)

        IFG2 &= ~UCB0RXIFG;   // clear IFG
    }

    if(IFG2&UCA0RXIFG) {  // USCI_A0 requested RX interrupt (UCA0RXBUF is
```

```
full)

//    Uncomment this block of code if you would like to use this COM
protocol that uses 253 as STARTCHAR and 255 as STOPCHAR
/*      if(!started) {  // Haven't started a message yet
            if(UCA0RXBUF == 253) {
                started = 1;
                newmsg = 0;
            }
        }
        else {  // In process of receiving a message
            if((UCA0RXBUF != 255) && (msgindex < (MAX_NUM_FLOATS*5))) {
                rxbuff[msgindex] = UCA0RXBUF;

                msgindex++;
            } else {    // Stop char received or too much data received
                if(UCA0RXBUF == 255) {  // Message completed
                    newmsg = 1;
                    rxbuff[msgindex] = 255; // "Null"-terminate the array
                }
                started = 0;
                msgindex = 0;
            }
        }
*/

        IFG2 &= ~UCA0RXIFG;
    }

}

// This function takes care of all the timing for printing to UART
// Rate determined by how often the function is called in Timer ISR
int print_timecheck = 0;
void print_every(int rate) {
    if (rate < 15) {
        rate = 15;
    }
    if (rate > 10000) {
        rate = 10000;
    }
    print_timecheck++;
    if (print_timecheck == rate) {
        print_timecheck = 0;
```

```
        newprint = 1;
    }

}
```

5. Code:

```
#include "msp430g2553.h"
#include "UART.h"

void print_every(int rate);

char newprint = 0;
long NumOn = 0;
long NumOff = 0;
int statevar = 1;
int timecheck = 0;

int ADC_value;

void main(void) {

    WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

    // Initializing ADC10
    ADC10CTL0 = SREF_0 + ADC10SHT_1 + ADC10ON + ADC10IE; // Vr+ = Vcc, Vr-
= Vss, 8 x ADC10CLKS, ADC10 on, ADC Interrupt enable
    ADC10CTL1 = INCH_3 + SHS_0 + ADC10DIV_0 + ADC10SSEL_0 + CONSEQ_0; //
P3: Input channel A3, ADC10SC, Clock divide /1, Clock source = ADC10OSC,
Single-channel-single-conversion
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_3 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /4
    //ADC10CTL1 = INCH_0 + SHS_0 + ADC10DIV_7 + ADC10SSEL_0 + CONSEQ_0; //
Clock divide /8

    ADC10AE0 = 0x08;  // P3: Enable A3 ADC channel

    ADC10CTL0 |= ENC; // ADC10 enabled

    if (CALBC1_16MHZ ==0xFF || CALDCO_16MHZ == 0xFF) while(1);

    DCOCTL = CALDCO_16MHZ;    // Set uC to run at approximately 16 Mhz
    BCSCTL1 = CALBC1_16MHZ;

    // Initialize Port 1
    P1SEL &= ~0x01;  // See page 42 and 43 of the G2553's datasheet, It
```

```
shows that when both P1SEL and P1SEL2 bits are zero
    P1SEL2 &= ~0x01; // the corresponding pin is set as a I/O pin.
Datasheet:
http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/msp430g2553datasheet.pdf
    P1REN = 0x0;  // No resistors enabled for Port 1
    P1DIR |= 0xf0; // Set P1,4,P1.5, P1.6, and P1.7 to output to drive LED on LaunchPad board.
    P1OUT &= ~0x01;  // Initially set P1.0 to 0

    // Initialize Port 2 - TA1.1 PWM Output
    P2DIR |= 0x04;   // P2.2 output
    P2SEL |= 0x04;   // P2.2 TA1.1 option
    P2SEL2 &= ~0x04; // P2.2 TA1.1 option

    // PWM Config
    TA1CCR0 = 1600;             // PWM Period
    TA1CCTL1 = OUTMOD_7;         // TA1CCR1 reset/set
    TA1CCR1 = 0;                // TA1CCR1 PWM duty cycle
    TA1CTL = TASSEL_2 + MC_1;    // SMCLK, up mode

    // Timer A Config
    TACCTL0 = CCIE;            // Enable Periodic interrupt
    TACCR0 = 16000;               // period = 1ms
    TACTL = TASSEL_2 + MC_1; // source SMCLK, up mode



    Init_UART(115200,1);    // Initialize UART for 115200 baud serial communication

    _BIS_SR(GIE);       // Enable global interrupt


    while(1) {  // Low priority Slow computation items go inside this while loop.  Very few (if anyt) items in the HWs will go inside this while loop

// for use if you want to use a method of receiving a string of chars over the UART see USCI0RX_ISR below
//      if(newmsg) {
//          newmsg = 0;
//      }

        // The newprint variable is set to 1 inside the function
```

```
"print_every(rate)" at the given rate
        if ( (newprint == 1) && (senddone == 1) )  { // senddone is set to 1
after UART transmission is complete

            // UART_printf is called every 0.25s
            //UART_printf("A0: %d \n\r", ADC_value);
            newprint = 0;
        }

    }
}


// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    timecheck++; // Keep track of time for main while loop.
    //ADC10CTL0 |= ENC + ADC10SC; // Trigger ADC10 every 1  millisecond
    if (timecheck == 250) {
        timecheck = 0;
        newprint = 1; // do the print job
    }

}

// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have
completed
int timecheck1 = 0;
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
    /*
    ADC_value = (int)ADC10MEM; // save conversion value to adc_value
    TA1CCR1 = (int)((unsigned long)ADC_value*1600/1023);

    switch (statevar) {
        case 1:  //start state, all four LEDs off
            P1OUT &= ~0xf0;          // Clear P1.4 to P1.7 LED off

            if (ADC10MEM < 256) {
                timecheck1++; // if the next state is still 1, count
timechek1
            }
            else{
```

```
            timecheck1 = 0; // otherwise set timechek1 to 0 since the
state 1 is not continued
            }

            if (timecheck1 == 2000) {
                timecheck1 = 0;
                statevar = 3; // When the lights have been turned off for 2
seconds, turn on all the LEDs.
                break;
            }

            if (ADC10MEM < 256) {
                statevar = 1;  // stays the same.
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 2:  // two LEDs are on
            P1OUT |= 0x30;  // Set P1.4 and P1.5 LED on
            if (ADC10MEM < 256) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // stays the same.
            }
            else {
                statevar = 3;  // Next Timer_A call go to state 3
            }
            break;
        case 3:  // all four LEDs on
            P1OUT |= 0xf0;  // Set all four LEDs on
            if (ADC10MEM < 256) {
                statevar = 1;  // Next Timer_A call go to state 1
            }
            else if (ADC10MEM < 512) {
                statevar = 2;  // Next Timer_A call go to state 2
            }
            else {
                statevar = 3;  // stays the same.
            }
```

```
            break;

    }
    */
}



// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another
character)
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void) {

    if(IFG2&UCA0TXIFG) {        // USCI_A0 requested TX interrupt
        if(printf_flag) {
            if (currentindex == txcount) {
                senddone = 1;
                printf_flag = 0;
                IFG2 &= ~UCA0TXIFG;
            } else {
                UCA0TXBUF = printbuff[currentindex];
                currentindex++;
            }
        } else if(UART_flag) {
            if(!donesending) {
                UCA0TXBUF = txbuff[txindex];
                if(txbuff[txindex] == 255) {
                    donesending = 1;
                    txindex = 0;
                }
                else txindex++;
            }
        } else {  // interrupt after sendchar call so just set senddone flag
since only one char is sent
            senddone = 1;
        }

        IFG2 &= ~UCA0TXIFG;
    }

    if(IFG2&UCB0TXIFG) {    // USCI_B0 requested TX interrupt (UCB0TXBUF is
empty)

        IFG2 &= ~UCB0TXIFG;   // clear IFG
```

```
    }
}


// USCI Receive ISR - Called when shift register has been transferred to
RXBUF
// Indicates completion of TX/RX operation
char character = 0;
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {

    if(IFG2&UCB0RXIFG) {  // USCI_B0 requested RX interrupt (UCB0RXBUF is
full)

        IFG2 &= ~UCB0RXIFG;   // clear IFG
    }

    if(IFG2&UCA0RXIFG) {  // USCI_A0 requested RX interrupt (UCA0RXBUF is
full)

        character = UCA0RXBUF;


        // ASCII character '1' = 49 : turn on LED 1
        // ASCII character '2' = 50 : turn on LED 2
        // ASCII character '3' = 51 : turn on LED 3
        // ASCII character '4' = 52 : turn on LED 4
        // ASCII character '5' = 53 : turn off LED 1
        // ASCII character '6' = 54 : turn off LED 2
        // ASCII character '7' = 55 : turn off LED 3
        // ASCII character '8' = 56 : turn off LED 4
        // Any other ASCII character : turn all LEDs off

        if (character == 49) {
            P1OUT |= 0x10; // turn on LED 1
        }
        else if (character == 50){
            P1OUT |= 0x20; // turn on LED 2
        }
        else if (character == 51){
            P1OUT |= 0x40; // turn on LED 3
        }
        else if (character == 52){
            P1OUT |= 0x80; // turn on LED 4
```

```
        }
        else if (character == 53){
            P1OUT &= ~0x10; // turn off LED 1
        }
        else if (character == 54){
            P1OUT &= ~0x20;// turn off LED 2
        }
        else if (character == 55){
            P1OUT &= ~0x40; // turn off LED 3
        }
        else if (character == 56){
            P1OUT &= ~0x80; // turn off LED 4
        }
        else
        {
            P1OUT &= ~0xF0; // turn all LEDs off
        }

        sendchar(character);


//    Uncomment this block of code if you would like to use this COM
protocol that uses 253 as STARTCHAR and 255 as STOPCHAR
/*      if(!started) {  // Haven't started a message yet
            if(UCA0RXBUF == 253) {
                started = 1;
                newmsg = 0;
            }
        }
        else {  // In process of receiving a message
            if((UCA0RXBUF != 255) && (msgindex < (MAX_NUM_FLOATS*5))) {
                rxbuff[msgindex] = UCA0RXBUF;

                msgindex++;
            } else {    // Stop char received or too much data received
                if(UCA0RXBUF == 255) {  // Message completed
                    newmsg = 1;
                    rxbuff[msgindex] = 255; // "Null"-terminate the array
                }
                started = 0;
                msgindex = 0;
            }
        }
*/
```

```
        IFG2 &= ~UCA0RXIFG;
    }

}

// This function takes care of all the timing for printing to UART
// Rate determined by how often the function is called in Timer ISR
int print_timecheck = 0;
void print_every(int rate) {
    if (rate < 15) {
        rate = 15;
    }
    if (rate > 10000) {
        rate = 10000;
    }
    print_timecheck++;
    if (print_timecheck == rate) {
        print_timecheck = 0;
        newprint = 1;
    }

}
```

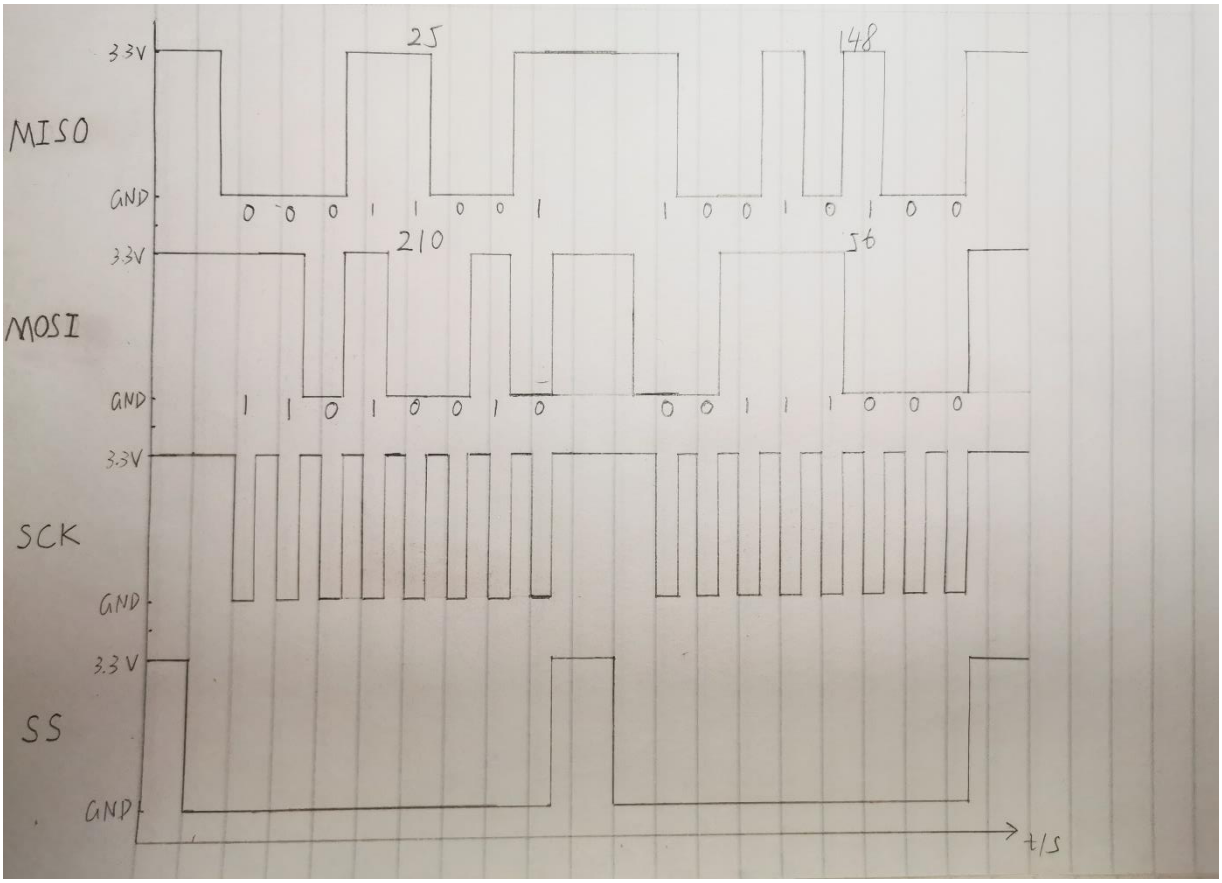
6. Answer:

(1) Address: B8h

(2) Data: B7h, EAh, 2Bh

(3) $T_{SDA} \approx \frac{0.5 \times 10^{-4}}{6} sec, \therefore baud\ rate = \frac{8}{9T_{SDA}} \approx 106667\ bits/sec$

7. Answer:



8. Answer:

```
switch(state) {
    case 1: // No money
        Display 0 cents on Soda Machine
        if (coin == 5) { // if statements to decide want to do next time into switch statements
            state = 2;
        }
        if (coin == 10) {
            state = 3;
        }
        if (coin == 25) {
            state = 6;
        }
        break;
    case 2: // 5 cents deposited
        Display 5 cents on Soda Machine
        if (coin == 5) { // if statements to decide want to do next time into switch statements
            state = 3;
```

```
        }
        if (coin == 10) {
            state = 4;
        }
        if (coin == 25) {
            state = 7;
        }
        break;
case 3: // 10 cents deposited
        Display 10 cents on Soda Machine
        if (coin == 5) { // if statements to decide want to do next time into switch statements
            state = 4;
        }
        if (coin == 10) {
            state = 5;
        }
        if (coin == 25) {
            state = 8;
        }
        break;
case 4: // 15 cents deposited
        Display 15 cents on Soda Machine
        if (coin == 5) { // if statements to decide want to do next time into switch statements
            state = 5;
        }
        if (coin == 10) {
            state = 6;
        }
        if (coin == 25) {
            state = 9;
        }
        break;
case 5: // 20 cents deposited
        Display 20 cents on Soda Machine
        if (coin == 5) { // if statements to decide want to do next time into switch statements
            state = 6;
        }
        if (coin == 10) {
            state = 7;
        }
        if (coin == 25) {
            state = 5;
        }
        break;
```

```
        case 6: // 25 cents deposited
            Display 25 cents on Soda Machine
            if (coin == 5) { // if statements to decide want to do next time into switch statements
                state = 7;
            }
            if (coin == 10) {
                state = 8;
            }
            if (coin == 25) {
                state = 6;
            }
            break;
        case 7: // 30 cents deposited
            Display 30 cents on Soda Machine
            if (coin == 5) { // if statements to decide want to do next time into switch statements
                state = 8;
            }
            if (coin == 10) {
                state = 9;
            }
            if (coin == 25) {
                state = 7;
            }
            break;
        case 8: // 35 cents deposited
            Display 35 cents on Soda Machine
            if (coin == 5) { // if statements to decide want to do next time into switch statements
                state = 9;
            }
            if (coin == 10) {
                state = 8;
            }
            if (coin == 25) {
                state = 8;
            }
            break;
        case 9: // 40 cents deposited
            Dispense Soda
            state = 0;
            break;
}
```