EEL 4742C: Embedded Systems

Name: Hamzah Ullah

Lab 8: Universal Asynchronous Receiver and Transmitter (UART)

<u>Introduction:</u>
This lab introduces us to a form of communication between the Microcontroller and the computer through a terminal. There are several ways to communicate data between the machines, but this lab teaches us UART. UART means Universal Asynchronous Receiver and Transmitter and is a half duplex communicator which means that the data goes only 1 direction at a time. We learn how to read via the UART through the IDE and write to the terminal from the IDE. We use methods to read and write different data types. We than learn how to change the default settings through the family users guide.
<u>Part 1: Transmitting Bytes with UART</u>
This part of the lab introduces us to what UART is and how it transmits data. It starts by explaining what the transmission pattern of UART is. It looks for the start bit and depending on the settings will read the incoming transmission bit. The rate at which the bit is shown is called the baud rate. The default baud rate is 9600hz or 1/9600 of a second duration of a bit. This means that each bit will be transferred at that rate, and the terminal would need to match or be in excess to read that transmission and vice versa.
We use UART through something called the enhanced Universal Serial Communications interface or eUSCI for short. It has all the required flags and registers we need to communicate with the UART system to allow for more user friendly programming.
The rest of the lab shows us how to implement UART by redirecting the pins, generating frequencies and initializing UART.
Finally we open the terminal and display our required program which is to increment from 0-9 and turn on and off the green led with 1 and 2 respectively. Because UART reads decimal values as ASCII, we must tailor the programming to meet this requirement. Below is the required code.
Transmit 0-9 via ASCII values using UART, and turns on and off greenLED through terminal values ‘1’, ‘2’ respectively.

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
#define redLED BIT0
#define greenLED BIT7
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

    P1DIR |= redLED;    //pins as output
    P1OUT &= ~redLED;    //red off
    P9DIR |= greenLED;    //pins as output
    P9OUT &= ~greenLED;    //red off
    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
    Initialize_UART();
    int i=65;
    int j=0;
    TXBUFFER &= ~TXFLAG;
//numbers
    while(i<58){P1OUT |= redLED;
    for(j=0; j<30000;j++){}
    uart_write_char(i++);

    uart_write_char('\n');
    uart_write_char('\r');
    //uart_read_char(i++);
    P1OUT &= ~redLED;
      if(i == 58)
        i=48;
    }
    TXBUFFER |= TXFLAG;
//TURNS ON AND OFF GREEN LED WITH INPUT 1 2
while(RXFLAG != 0)
{
 uart_read_char();
 if(RXBUFFER == '1')
 P9OUT |= greenLED;
 if(RXBUFFER == '2')
     P9OUT &= ~greenLED;
}
//    uart_write_uint16(953);
TXBUFFER |= TXFLAG;
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch){
    //wait for any ongoing transmission to complete
    while((FLAGS & TXFLAG)==0){}
    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
unsigned char uart_read_char(void){
    unsigned char temp;
    //return null if no byte received
    if((FLAGS & RXFLAG)== 0)
    return;
    //otherwise, copy the received byte (clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}
```

Part 2: Sending unsigned 16-bit Integers over UART

This part of the lab is similar to a previous lab, where we must send a 16 bit number through the UART. To do this we must take in a decimal number and output the equivalent value through UART. To do this, we must reduce the numbers to the base 0-9 values which is represented as 48-57 and transmit it through the UART to appear on the terminal, than increment all the way up to 65536. Below is the following code with the method for 16bit output below.

Increments up while counting 0-65536-1;

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
#define redLED BIT0
#define greenLED BIT7
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

    P1DIR |= redLED;    //pins as output
    P1OUT &= ~redLED;    //red off
    P9OUT &= ~greenLED;
    P9DIR |= greenLED;

    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
    Initialize_UART();
    int i=0;
    int j=0;
    TXBUFFER &= ~TXFLAG;
    while(i<=65536-1){
    uart_write_uint16(i++);
    uart_write_char('\n');
    uart_write_char('\r');
    }
```

```
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch){
    //wait for any ongoing transmission to complete
    while((FLAGS & TXFLAG)==0){}

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
unsigned char uart_read_char(void){
    unsigned char temp;
    //return null if no byte received
    if((FLAGS & RXFLAG) == 0)
    return 0;
    //otherwise, copy the received byte (clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}
void uart_write_uint16(unsigned int n)
{
    int temp;
    if(n>=10000)
    {
        temp = n/10000;
        uart_write_char(temp+48);
    }
    if(n>=1000)
    {
        temp = n/1000 % 10;
        uart_write_char(temp+48);
    }
    if(n>=100)
    {
        temp = n/100 % 10;
        uart_write_char(temp+48);
    }
    if(n>=10)
    {
        temp = (n/10) % 10;
        uart_write_char(temp+48);
    }
     n = n % 10;
     uart_write_char(n+48);

    //Write the byte to the transmit buffer
    //TXBUFFER = n;
}
```

Part 3: Sending an ASCII String over UART

Next we have to send string to the UART. To do so we simply have to find when the character string(char myString) ends or reaches NULL. When it does we end the transmission and leave the method. This was a relatively easy part of the lab (3-5 lines of code).

Passes string, 'myString' to uart_write_string() method, outputs value

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
#define redLED BIT0
#define greenLED BIT7
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

    P1DIR |= redLED;    //pins as output
    P1OUT &= ~redLED;    //red off
    P9OUT &= ~greenLED;
    P9DIR |= greenLED;

    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
    Initialize_UART();
    int i=0;
    int j=0;
    TXBUFFER &= ~TXFLAG;
    //uart string method
    char myString[] = {"UART TRANSMISSION BEGINS..."};
    uart_write_string(myString);
    uart_write_char('\n');
    uart_write_char('\r');
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch){
    //wait for any ongoing transmission to complete
    while((FLAGS & TXFLAG)==0){}

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
unsigned char uart_read_char(void){
    unsigned char temp;
    //return null if no byte received
    if((FLAGS & RXFLAG) == 0)
    return 0;
    //otherwise, copy the received byte (clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}

void uart_write_string(char * str)
{
    int i;
    for(i=0; i < strlen(str); i++)
    uart_write_char(str[i]);
}
```

Part 4: Changing the Configuration

In this part of the lab, we are to reconfigure the default settings of UART to a new baud rate and clock signal. The new clock signal is a 32khz ACLK with a 4800 baud rate. Copying over the old 32khz ACLK function, we must now create a new Initialize function for UART. In the family users guide, it informs us that for a 4800 baud rate in ACLK, oversampling is not supported. With that, we know that COS16 which deals with oversampling is set to 0. In turn, the BRF is a don't care because we are not oversampling. To change the clock type, we modify UCA1CTLW0 to ACLK which is UCSSEL_1;. To determine the correct values for BR and BRS value we take the frequency and divide it by the baud rate we want. (32768/4800 = 6.822). This tells us the BR is 6 and the BRS is determined by a chart in the family users guide. This gives us a hexadecimal value of 0xEE. To get this value, we OR BRS values 8+4+2 to get E for both bit 0 and 1. Finally we set the baud rate to 4800 to match the incoming otherwise the message will not display properly.

Changing to 32khz clock with 4800 baud rate where BRS = oxEE, BR = 6, BRF=X, COS16 = 0

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
#define redLED BIT0
#define greenLED BIT7
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins

    P1DIR |= redLED;    //pins as output
    P1OUT &= ~redLED;    //red off
    P9OUT &= ~greenLED;
    P9DIR |= greenLED;

    config_ACLK_to_32KHz_crystal();

    //Divert Pins to backchannel UART functionality
    //(UCA1TXD same as 3.4)(UCA1RXD same as 3.5)
    //P3SEL=00, P3SEL0 = 11 P2DIR=XX
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
    Initialize_UART2();
    int i=48;
    int j=0;
    while(i<58)
     {
        P1OUT |= redLED;
     for(j=0; j<10000;j++){}
     uart_write_char(i++);

     uart_write_char('\n');
     uart_write_char('\r');

     P1OUT &= ~redLED;
     if(i == 58)
         i=48;
     }
    TXBUFFER &= ~TXFLAG;

}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void Initialize_UART2(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use aclk clock;
    UCA1CTLW0 |= UCSSEL_1;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = x, UCBRS = 0xEE, UCOS16=0 (no oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS7 | UCBRS6 | UCBRS5 | UCBRS3 | UCBRS2 | UCBRS1;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch){
    //wait for any ongoing transmission to complete
    while((FLAGS & TXFLAG)==0){}

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
unsigned char uart_read_char(void){
    unsigned char temp;
    //return null if no byte received
    if((FLAGS & RXFLAG) == 0)
    return 0;
    //otherwise, copy the received byte (clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}
void uart_write_string(char * str)
{
    int i;
    for(i=0; i < strlen(str); i++)
    uart_write_char(str[i]);
}
void config_ACLK_to_32KHz_crystal()
    {
        //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

        //Reroute pins to LFXIN/LFXOUT functionality
        PJSEL1 &= ~BIT4;
        PJSEL0 |= BIT4;

        //Wait until the oscillator fault flags remain cleared
        CSCTL0 = CSKEY;
        do
        {
            CSCTL5 &= ~LFXTOFFG;    //local fault flag
            SFRIFG1 &= ~OFIFG;      //Global fault flag
        }
        while((CSCTL5 & LFXTOFFG) != 0);

        CSCTL0_H = 0; //lock CS registers
        return;
    }
```

QUESTIONS:

1. **What's the difference between UART and eUSCI?**
   UART is a method of communicating between the MCU and CPU through the terminal, and the eUSCI interfaces with UART to hold all the registers and flags for us to program. eUSCI is the one actually pushing those values through UART to send to the CPU and vice versa. eUSCI is like the middle man between the MCU/CPU and the user programing them. eUSCI also implements the protocols for all the communication devices, while UART is simply a communication method. There are multiple ways to communicate more than UART which eUSCI can implement.
2. **What is backchannel UART**
   The backchannel UART is the method in which UART connects to the PC. It allows us to transmit data between the MCU and the CPU. We are using a USB to transfer data over the backchannel which can be read through a COM port(communication port) that the PC can access through a terminal program.
3. **Whats the function of the two lines of code that have P3SEL1 and P3SEL0?**
   To divert the backchannel UART to the eUSCI TX/RX pins. According to the users guide, there are specific values to set when deciding which functions to use in the port for P3SEL1/P3SEL0 where SEL1 = 00, and SEL0 = 11. So we ~AND 3.x bits 4 and 5 (p3.4/p3.5) and turn on P3SEL0 by ORing the bits in the right port (p3.4/p3.5).
4. **The microcontroller has a clock of 1,000,000 hz and we want to setup UART connection at 9600 baud. How do we obtain a clock rate of 9600 hz? Explain the approach at a high level.**
   To get a 9600hz clock we have to understand what 9600 baud means. It means that the duration at which remains visible for reading which is 1/9600 seconds. This represents essentially 1 period at which the bit remains before moving on to the next bit. A period(T) is defined as 1 over the frequency. Frequency is defined as the rate at which something occurs. Hertz is defined as the a unit of frequency which equals one cycle per second(Google Dictionary). Therefore, the rate at which a bit is shown is 1/9600s or 9600 cycles per second which translates to 9600Hz.
5. **A UART transmitter is transmitting data at 1200 baud. What is the receivers clock frequency if oversampling is not used?**
   The maximum frequency is 19114hz. To determine we do F/1200 baud = N, where N <16. Using the family users guide, the highest fractional portion is .9288. so F/1200= 15.928 = ~19114hz max or F/1200 = 1 = 1200hz min.

6. **A UART transmitter is transmitting data at 1200 Baud. What is receivers clock frequency if oversampling is used? Whats the benefit of oversampling?**
   The typical rate as defined by the user's guide is 32768hz at 1200 baud, but the minimum value can be as low as 19200hz with zero fractional potion. To calculate we use the formula F/1200 = N where N >16. We can also use the typical ACLK of 1200khz. The benefits of oversampling is that the receiver checks the signal at 16x the rate of the frequency which means a more accurate transmission without the receiver having a chance of missing or misreading a bit in order.