EEL 4742C: Embedded Systems

Christopher Badolato
11/4/2019
EEL4742-0011
Lab 8 Universal Asynchronous Receiver and Transmitter

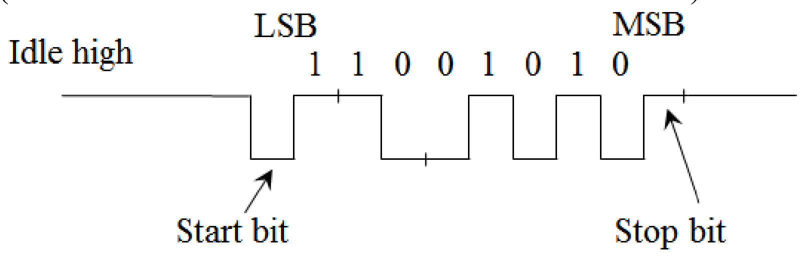
Introduction

In this lab experiment we will be using UART to transfer bytes between PC and MCU. First off, we will start by send a single character at a time to the PC from the MCU using a for loop 0 through 9 we will continuously cycle through the characters. As well as if character 1 is typed in the terminal the green LED will turn on, if character 2 is typed the green LED will shut off. Next, we configure a uart_write_uint16, which will transfer values from 0 to 65536 to the PC and display on the terminal. Then, we will send an entire string through uart to the PC. Finally, we will change the configuration and baud rate to run the uart using the 32Khz crystal as the clock source.

Part 8.1: Transmitting Bytes with UART

Uart allows us to transfer bits between two parties. UART is asynchronous meaning the transmitter and receiver have their own clock signals.
In the transmission pattern below, the line is idle at high, it drops to low for one bit to signal the start bit, after that the data is transferred bit by bit starting with the least significant bit. Then, a stop bit will transmit a high value to signal the end of the data.

(Borrowed from Abichar's LAB8 transmits 01010011)



Bit duration is defined as board rate which is the transmitter's clock rate. We will use baud rate of 9600 or 9600 Hz. We will configure the SMCLK to 1MHZ with a baud rate of 9600.

To get data to transfer we must first configure UART by, diverting the pins 3.4 and 3.5 to variable values RX (receive buffer) and TX (transmit buffer). As well, we must configure the RX and TX flags that let us know when data is ready to transmit or read. The transmit flag will be set to 1 when ready to transfer data. A byte will be copied to the transfer buffer and transmit flag will be set to 0, letting us know that our byte is ready to transmit. After transmission the Flag is reset back to 1.
The receive flag is the opposite. When receive flag is 0, there is no new data, when we receive a byte, the flag becomes 1 and our function can return the temp value storing the byte.
We write three functions: one to configure UART, one to read with UART and one to write with UART.

Finally, our main function will loop through characters 0 through 9 printing to the terminal with our UART write function each second (using ACLK in upmode with 32768 cycles). As well, if a 1 is typed into the terminal the green LED will be turned on, if a 2 is typed, the LED will shut back off. This is configured to change each second with the channel 0 clock.

```
//Christopher Badolato
//11/4/2019
//Lab 8.1
//EEL 4742 0011
//UART intro12

#include <msp430.h>
#include <stdio.h>

#define FLAGS UCA1IFG // Contains the transmit & 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 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7


// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit
// no flow control
// Initial clock: SMCLK @ 1.048 MHz with oversampling
void Initialize_UART(void){
        // Divert pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
        // Use SMCLK clock; leave other settings default
    UCA1CTLW0 |= UCSSEL_2;
        // Configure the clock dividers and modulators
        // UCBR=6, UCBRF=8, UCBRS=0x20, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;
        // Exit the reset state (so transmission/reception can 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;
}

    // The function returns the byte; if none received, returns NULL
unsigned char uart_read_char(void){
    unsigned char temp;
            // Return NULL if no byte received
    if( (FLAGS & RXFLAG) == 0)
        return NULL;
            // Otherwise, copy the received byte (clears the flag) and return it
        temp = RXBUFFER;
    return temp;
}

int main(void){

    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins
    P1DIR |= redLED; // Direct pin as output
    P9DIR |= greenLED; // Direct pin as output

    P1OUT &= ~redLED; // Turn LED Off
    P9OUT &= ~greenLED; // Turn LED Off

    Initialize_UART();

    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA0CCR0 = 32768;

    for(;;){
        unsigned char input;
        unsigned char ch;
        for(ch = '0'; ch <= '9'; ch++){
                //write the current character
            uart_write_char(ch);
                //newline
            uart_write_char('\n');
                //return
            uart_write_char('\r');
                //short 1 second delay for loop and LED toogling
            while((TA0CTL & TAIFG) == 0){}
            TA0CTL &= ~TAIFG;
                //Get next char entered into terminal
            input = uart_read_char();
            P1OUT ^= redLED;
                //if value taken from terminal is 1 turn on green LED
            if(input == '1'){
                P9OUT |= greenLED;
            }
                //if value taken from terminal is char 2 turn off green LED
            else if(input == '2'){
                P9OUT &= ~greenLED;
            }
        }
    }
}
```

Part 8.2: Sending Unsigned 16-bit Integers over UART

In this part of the lab we will continue UART, but we will loop through every value we can represent with 16-bits and display them in the terminal. I used code very similar to part 8.1 but we will implement a function to extract each decimal bit using division and modulus.
Below is an example of how the function works

n = 65536
MSB to LSB
6 = n/10000
5 = (n/1000) % 10;
5 = (n/100) % 10;
3 = (n/10) % 10;
6 = (n) % 10;

I implemented this function and use it to iterate from 0 to 65536 using UART transmission. We will loop to 65536 displaying each value as we go. Each iteration has a little delay loop so we can see the progress (otherwise it will count too fast).

```
//Christopher Badolato
//11/4/2019
//Lab 8.2
//EEL 4742 0011
//UART intro

#include <msp430.h>
#include <stdio.h>

#define FLAGS UCA1IFG // Contains the transmit & 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 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7


// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit
// no flow control
// Initial clock: SMCLK @ 1.048 MHz with oversampling
void Initialize_UART(void){
        // Divert pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
        // Use SMCLK clock; leave other settings default
    UCA1CTLW0 |= UCSSEL_2;
        // Configure the clock dividers and modulators
        // UCBR=6, UCBRF=8, UCBRS=0x20, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;
        // Exit the reset state (so transmission/reception can 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;
}
```

```
    // The function returns the byte; if none received, returns NULL
unsigned char uart_read_char(void){
    unsigned char temp;
            // Return NULL if no byte received
    if( (FLAGS & RXFLAG) == 0)
        return NULL;
            // Otherwise, copy the received byte (clears the flag) and return it
        temp = RXBUFFER;
    return temp;
}

uart_write_uint16(unsigned int n){
    int num;

    if(n >= 10000){
        num = n/10000;
        uart_write_char('0'+ num);
    }
    if(n >= 1000){
        num = (n/1000) % 10;
        uart_write_char('0'+ num);
    }
    if(n >= 100){
        num = (n/100) % 10;
        uart_write_char('0'+ num);
    }
    if(n >= 10){
        num = (n/10) % 10;
        uart_write_char('0'+ num);
    }
    num = n % 10;
    uart_write_char('0' + num);
}

int main(void){
    volatile unsigned int i;
    unsigned int value;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins
    P1DIR |= redLED; // Direct pin as output
    P9DIR |= greenLED; // Direct pin as output

    P1OUT &= ~redLED; // Turn LED Off
    P9OUT &= ~greenLED; // Turn LED Off

    Initialize_UART();

    for(;;){
        P1OUT ^= redLED;
        for(value = 0; value < 65536; value++){
                //write the current integer
            uart_write_uint16(value);
                //newline
            uart_write_char('\n');
```

```
                 //return
             uart_write_char('\r');

             for(i = 0; i <2000; i++){}
         }
     }
}
}
```

Part 8.3 Sending an ASCII String over UART

In this part, we will use the uart_write_char(); function (part 8.1) to write an entire string to the terminal
We can simply loop through the string and send each character to the UART write function.
Using the NULL signal, we can determine when the string has finished being written.

```
//Christopher Badolato
//11/4/2019
//Lab 8.3
//EEL 4742 0011
//UART intro

#include <msp430.h>
#include <stdio.h>

#define FLAGS UCA1IFG // Contains the transmit & 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 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7


// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit
// no flow control
// Initial clock: SMCLK @ 1.048 MHz with oversampling
void Initialize_UART(void){
        // Divert pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
        // Use SMCLK clock; leave other settings default
    UCA1CTLW0 |= UCSSEL_2;
        // Configure the clock dividers and modulators
        // UCBR=6, UCBRF=8, UCBRS=0x20, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;
        // Exit the reset state (so transmission/reception can 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;
}

    // The function returns the byte; if none received, returns NULL
unsigned char uart_read_char(void){
    unsigned char temp;
            // Return NULL if no byte received
    if( (FLAGS & RXFLAG) == 0)
        return NULL;
            // Otherwise, copy the received byte (clears the flag) and return it
        temp = RXBUFFER;
    return temp;
}

uart_write_string(char * str){
    int i = 0;
        while(str[i] !=  NULL){
            uart_write_char(str[i]);
            i++;
        }
        return;
}

int main(void){

    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

    Initialize_UART();
    char mystring[] = "UART Transmission Begins...";
    uart_write_string(mystring);
    return 0;

}
```

Part 8.4 Changing the Configuration

In this part of the lab we are repeating code from 8.2, but we will change the signal to 32KHz ACLK with a buad rate of 4800 as opposed to the previous versions that were using 1MHZ SMCLK with 9600 baud rate.

```
//Christopher Badolato
//11/4/2019
//Lab 8.4
//EEL 4742 0011
//UART intro baud rate must be set to 4800 in terminal.

#include <msp430.h>
#include <stdio.h>

#define FLAGS UCA1IFG // Contains the transmit & 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 // Red LED at P1.0
#define greenLED BIT7 // Green LED at P9.7

void config_ACLK_to_32KHz_crystal() {
        // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
        // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

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

    CSCTL0_H = 0; // Lock CS registers
    return;
}

// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit
// no flow control
// Initial clock: SMCLK @ 1.048 MHz with oversampling
void Initialize_UART2(void){
        // Divert pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);
        // Use ACLK clock; leave other settings default
    UCA1CTLW0 |= UCSSEL_1;
        // Configure the clock dividers and modulators
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS1 | UCBRS2 | UCBRS3 | UCBRS5 | UCBRS6 | UCBRS7 | UCBRF3|
UCBRF2 | UCBRF0;
        // Exit the reset state (so transmission/reception can 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;
}

    // The function returns the byte; if none received, returns NULL
unsigned char uart_read_char(void){
    unsigned char temp;
            // Return NULL if no byte received
    if( (FLAGS & RXFLAG) == 0)
        return NULL;
            // Otherwise, copy the received byte (clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}

uart_write_uint16(unsigned int currentValue){
    int number;

    if(currentValue >= 10000){
        number = currentValue/10000;
        uart_write_char('0'+ number);
    }
    if(currentValue >= 1000){
        number = (currentValue/1000) % 10;
        uart_write_char('0'+ number);
    }
    if(currentValue >= 100){
        number = (currentValue/100) % 10;
        uart_write_char('0'+ number);
    }
    if(currentValue >= 10){
        number = (currentValue/10) % 10;
        uart_write_char('0'+ number);
    }
    number = currentValue % 10;
    uart_write_char('0' + number);
}

int main(void){

    unsigned int value;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins


        //32khz clock
    config_ACLK_to_32KHz_crystal();
        //configure uart
    Initialize_UART2();
        //ACLK, div 1, upmode
    TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
    TA0CCR0 = (65536 - 1);

    for(;;){
        for(value = 0; value <= 65535; value++){
                //write the current integer
            uart_write_uint16(value);
                //newline
            uart_write_char('\n');
                //return
            uart_write_char('\r');
            while((TA0CTL & TAIFG) == 0){}
            TA0CTL &= ~TAIFG;
        }
    }
}
```

Student Q&A

1. What's the difference between UART and eUSCI?

Uart allows transmitting bytes between two parties. eUSCI is the hardware module of the MSP430 that implements all the details of UART transmission and reception that allows our code to interface with the registers and flags.

2. What is the backchannel UART?

The usb is the backchannel UART, using a COM port to generate the bytes on a PC.

3. What's the function of the two lines of code that have P3SEL1 and P3SEL0?

These two lines are used to divert the pins to the backchannel UART functionality.

4. The microcontroller has a clock of 1,000,000 Hz and we want to setup a UART connection at 9600 baud. How do we obtain a clock rate of 9600 Hz? Explain the approach at a high level.

We modulate and divide our clock signal to receive the buad rate we intend. The terminal must also be set to the same baud rate.

5. A UART transmitter is transmitting data at a 1200 baud. What is receiver's clock frequency if oversampling is not used?

If oversampling is not used we will not take samples at 16x the rate, but at the same rate, so the baud rate is also 1200.

6. A UART transmitter is transmitting data at at 1200 baud. What is receiver's clock frequency if oversampling is used? What's the benefit of oversampling?

16 x 1200 buad = 19,200 baud or 19,200 Hz. This way we can sample the data multiple times and do a majority vote on those samples.

Conclusion

In conclusion, in this lab we learned how to configure and use UART in a few different ways. First, we implemented UART and displayed a counter 0 to 9 on the terminal. We also implemented push button LED toggling. If 1 is entered in the terminal, then the green Led will turn on, if a 2 is entered the green LED shuts off. Next, we print a continuous counter representing each value up to 16-bits. Then, we use UART to print an entire string to the terminal. Finally, we changed the buad rate to 4800 and configure the clock to 32Khz to learn to change the UART configuration settings.