**10/27/2020**

Joshua Frazer

Lab 8

Universal Asynchronous Receiver and Transmitter (UART)

**Introduction:**

With this lab we were given various assignments to incorporate the UART interface. For part 1, we start off basic and make a single digit counter (0-9) that gets transmitted with UART while reading input from the keyboard. For part 2, we create a counter that can take on multiple digits with a max count of 65535. In part 3, you get familiarized with parsing through characters within a string to print out within the terminal. As for part 4, we follow what we did within part 1 but change the baud rate to 4800.

**Part 1:**

After copying the functions need for this part, I approached this part as I read it. Sadly, I misread the fact that we had to configure the clock dividers and modulators myself, so I lost so much time and confusion doing this part. Other than that, on my first debug, everything worked expectedly.

**Part within video:** 0:11

// Single digit counter and key input through UART

**#include** <msp430fr6989.h>

**#include** <stdlib.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**#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

**void** **Initialize\_UART**(**void**);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **main**(**void**) {

**volatile** **unsigned** **int** i, key;

**unsigned** **char** ch;

WDTCTL = WDTPW | WDTHOLD; // Stop the 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**(;;)

{

key = uart\_read\_char();

**for**(ch='0';ch<='9';ch++)

{

**if** (key == '1')

P9OUT |= greenLED;

**else** **if** (key == '2')

P9OUT &= ~greenLED;

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

uart\_write\_char(ch);

uart\_write\_char('\n');

uart\_write\_char('\r');

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

}

P1OUT ^= redLED;

}

}

// 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=13, UCBRS=0x22, 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;

}

**Questions:**

No questions were given.

**Part 2:**

The part was a little more complicated than the previous part. Instead of one digit, I had to worry about multiple digits. Thankfully, this was very familiar to the stopwatch, much of the code that is seen within my function is close to the first few lines of the Lab 5’s incrementing stopwatch. Instead of sending the digits through the LCD display, I modified my code to transmit the digit through UART.

**Part within video:** 0:47

// Unsigned 16-bit integer counter through UART

**#include** <msp430fr6989.h>

**#include** <stdlib.h>

**#include** <string.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**#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

**void** **Initialize\_UART**(**void**);

**void** **uart\_write\_uint16**(**unsigned** **int** n);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **main**(**void**) {

**volatile** **unsigned** **int** i = 0, counter = 65530;

WDTCTL = WDTPW | WDTHOLD; // Stop the 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**(;;)

{

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

uart\_write\_uint16(counter++);

**if** (counter == 65536)

counter = 0;

uart\_write\_char('\n');

uart\_write\_char('\r');

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

}

}

// 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=13, UCBRS=0x22, UCOS16=1 (oversampling)

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;

// Exit the reset state (so transmission/reception can begin)

UCA1CTLW0 &= ~UCSWRST;

}

**void** **uart\_write\_uint16**(**unsigned** **int** n){

**unsigned** **int** r[6] = {0};

**int** i = 0, j;

**if** (n == 0)

uart\_write\_char('0');

**while** (n != 0)

{

r[i++] = n % 10;

n /= 10;

}

**for** (j = i-1; j >= 0; --j)

uart\_write\_char(r[j] + '0');

}

**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;

}

**Questions:**

No questions were given.

**Part 3:**

Part 3 was probably the most simplistic part of this lab. After doing part 2, you realize that parsing the data is required. Keeping that in mind, instead of parsing numbers, I just had to parse a string which was effortless since its an array of characters.

**Part within video:** 1:06

// Printing a custom string through UART

**#include** <msp430fr6989.h>

**#include** <string.h>

**#include** <stdlib.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**#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

**void** **Initialize\_UART**(**void**);

**void** **uart\_write\_string**(**char** \*str);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **main**(**void**) {

**volatile** **unsigned** **int** i, key;

**char** myString[] = "UART Transmission Begins...... And I don't care. KAWHAT!!!!!! Lol.";

WDTCTL = WDTPW | WDTHOLD; // Stop the 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**(;;)

{

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

uart\_write\_string(myString);

uart\_write\_char('\n');

uart\_write\_char('\r');

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

}

}

// 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=13, UCBRS=0x22, UCOS16=1 (oversampling)

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRF3|UCOS16;

// Exit the reset state (so transmission/reception can begin)

UCA1CTLW0 &= ~UCSWRST;

}

**void** **uart\_write\_string**(**char** \*str)

{

**volatile** **unsigned** **int** i;

**for** (i = 0; i<**strlen**(str); i++)

{

uart\_write\_char(str[i]);

}

}

**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;

}

**Questions:**

No questions were given.

**Part 4:**

With part 4, there was not much that needed to be done other than changing the configuration of your dividers, modulators, and the ACLK to the 32KHz crystal. Once you get the right values from the family user’s guide for your dividers and modulators and copy the code for the function config\_ACLK\_to\_32KHz\_crystal this part is done.

**Part within video:** 1:25

// unsigned 16-bit integer counter through UART with 4800 baud rate

**#include** <msp430fr6989.h>

**#include** <stdlib.h>

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**#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

**void** **Initialize\_UART**(**void**);

**void** **Initialize\_UART2**(**void**);

**void** **uart\_write\_uint16**(**unsigned** **int** n);

**void** **uart\_write\_char**(**unsigned** **char** ch);

**unsigned** **char** **uart\_read\_char**(**void**);

**void** **config\_ACLK\_to\_32KHz\_crystal**();

**void** **main**(**void**) {

**volatile** **unsigned** **int** i, counter = 0;

WDTCTL = WDTPW | WDTHOLD; // Stop the 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\_UART2();

config\_ACLK\_to\_32KHz\_crystal();

**for**(;;)

{

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

uart\_write\_uint16(counter++);

**if** (counter == 65536)

counter = 0;

uart\_write\_char('\n');

uart\_write\_char('\r');

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

}

}

// 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 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;

}

// Configure UART to the popular configuration

// 4800 baud, 8-bit data, LSB first, no parity bits, 1 stop bit

// no flow control

// Initial clock: SMCLK @ 32 KHz

**void** **Initialize\_UART2**(**void**){

// Divert pins to UART functionality

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Use ACLK clock; leave other settings default

UCA1CTLW0 |= UCSSEL0;

// Configure the clock dividers and modulators

//

UCA1BRW = 6;

UCA1MCTLW = UCBRS7|UCBRS6|UCBRS5|UCBRS3|UCBRS2|UCBRS1;

// Exit the reset state (so transmission/reception can begin)

UCA1CTLW0 &= ~UCSWRST;

}

**void** **uart\_write\_uint16**(**unsigned** **int** n){

**unsigned** **int** r[6] = {0};

**int** i = 0, j;

**if** (n == 0)

uart\_write\_char('0');

**while** (n != 0)

{

r[i++] = n % 10;

n /= 10;

}

**for** (j = i-1; j >= 0; --j)

uart\_write\_char(r[j] + '0');

}

**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;

}

**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**;

}

**Questions:**

No questions were given.

**Student Q&A:**

1. What’s the difference between UART and eUSCI?
   1. **UART is a transmission scheme while eUSCI is an interface that holds different transmission schemes. You can use UART through the eUSCI.**
2. What is the backchannel UART?
   1. **Backchannel UART is a transmitter that is used to communicate between the microcontroller and PC.**
3. What’s the function of the two lines of code that have P3SEL1 and P3SEL0?
   1. **The two lines of code that have P3SEL1 and P3SEL0 are used to divert the pins on the microcontroller 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.
   1. **In order to obtain a clock rate of 9600 Hz we would need to properly set up the dividers and modulators accordingly. You could simply use the family user’s guide to find out which bits need to be configured to what on the UCA1MCTLW and UCA1BRW.**
5. A UART transmitter is transmitting data at 1200 baud. What is receiver’s clock frequency if oversampling is not used?
   1. **The receiver’s clock frequency would be 1200 Hz if oversampling is not used.**
6. A UART transmitter is transmitting data at 1200 baud. What is receiver’s clock frequency if oversampling is used? What’s the benefit of oversampling?
   1. **The receiver’s clock frequency would be 16x1200 Hz if oversampling is used. The benefit of oversampling is to gather a single data bit several times then choose the accordingly out of the samples.**

**Conclusion:**

This lab was not my favorite. Although, the UART interface is brilliant. The terminal gave me difficulties within the first part of the lab. I would have to say this was the most significant part of the lab. Once you learn how to configure the UART everything else is just previous knowledge.

**Video Link:**

[**https://www.youtube.com/watch?v=PmWP8Gq763w**](https://www.youtube.com/watch?v=PmWP8Gq763w)

[](https://www.youtube.com/embed/PmWP8Gq763w?feature=oembed)