// Lab 8: Universal Asynchronous Receiver and Transmitter (UART)

// Learn how to use UART interface for transmitting data between the microcontroller and the PC.

// **8.1: Transmitting Bytes with UART**

// UART 🡪 A simple interface that allows transmitting bytes between two parties!

// UART 🡪 An asynchronous in the sense that the transmitter and the receiver each has its own clock cycle signal.

// The rising and the falling edges are not guaranteed to coincide!

// UART 🡪 Using one wire and transmits data in the same direction over the wire.

// Implementation of it with one wire to transmit in one direction (known as half-duplex) or with two wires to allow bidirectional simultaneous transmission (known as full-duplex)!

// The line drops to low for one bit duration to signal the Start Bit!

// Transmission of data bit by bit 🡪 Starting with the least significant bit (LSB)!

// Stop Bit 🡪 Having a high value for transmission to signal at the end of the transmission!

// Bit duration 🡪 Defined by the baud rate (which is simply the transmitter’s clock rate)!

// A popular rate is 9600 baud (corresponding to a clock frequency of 9600 Hz).

// A bit takes 1/9600 seconds to complete!

// eUSCI Module 🡪 Performing UART communication with eUSCI (enhanced universal serial communication interface) module of the MSP430!

// The hardware module implements all the details of UART transmission and reception and our code interfaces with it using a few registers and flags!

// Organizing the eUSCI module into two channels! Supporting Channel A from UART and SPI!

// LaunchPad Board Setup 🡪 Setting up them with a back-channel UART over USB!

// Causes enabling the transmission of data between the LaunchPad and the PC!

// A virtual COM port is generated on the PC! 🡪 Any application that uses COM ports can communicate with the board!

// Using a terminal application (e.g. yperTerminal or TeraTerm)!

// Option 1: P3.4/UCA1TXD | P3.5/UCA1RXD

// Option 2: P5.4/UCA1TXD | P5.5/UCA1RXD

// Term UCA1 🡪 Referring to the eUCSI module #1 Channel A

// P3DIR 🡪 Having X (don’t care) while P3SEL1 have 0 for both bits and P3SEL0 have 1 for both bits.

// LCDS bits 🡪 They should be 0 (remaining in their default condition).

// Divert pins to backchannel UART functionality

// (UCA1TXD same as P3.4) (UCA1RXD same as P3.5)

// (P3SEL1=00, P3SEL0=11) (P2DIR=xx)

P3SEL1 &= ˜(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Jampers on the board 🡪 It can close the connections between the MSP430 chip and the emulation chip!

// **Generating the UART Clock Frequencies**

// A set of well-known baud rates 🡪 Usage for UART! 🡪 1200, 2400, 4800, 9600, 38400, …

// Clock Signal Need for UART Transmission 🡪 9600 baud, the transmitter uses a 9600 Hz clock signal!

// **eUSCI Module Configuration**

// Configuring the cUSCI module!

// The default values are corresponding to the most popular configuration!

// 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|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

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

UCA1CTLW0 &= ˜UCSWRST;

}

// **Programming Model**

// eUSCI Hardware 🡪 the UART transmission and reception of data and interfaces with the code using a few registers and flags!

// Renaming the flags and registers with user-friendly names by defining these symbolic constants at the beginning of the code!

#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

// The transmit flag is one when the module is ready to transmit!

// Transmitting a byte 🡪 It happens through copying the transmit buffer!

// Finishing the transmission 🡪 the transmit flag goes back to one!

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 receive flag is zero when there is no new data!

// Receiving a byte then the receive flag becomes one!

// 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;

}

// Transmitting bytes over UART to the terminal application on the PC!

// **Testing the UART Transmission**

// Combining the three functions along with the mask definitions.

// Writing an infinite loop for transmission of the characters from 0 to 9 over the UART transmission.

// Writing a delay loop that introduces a small delay between the characters!

// Toggling the red LED to indicate the ongoing activity.

// Writing (char ch=’A’;) makes the variable equal to the ASCII of ‘A’.

// Writing this loop for(ch=’0’; ch<=’9’; ch++) for generation of the ASCII number of the digits from 0 to 9.

// After every character 🡪 Transmitting a new line character ‘\n’ followed by the carriage return character ‘\r’.

// New line character ‘\n’ 🡪 Followed by the carriage return character ‘\r’!

// The new line character 🡪 causing the cursor to go down one line and the carriage return character causes the cursor to go to the leftmost column of the line!

// Reading the characters transmitted from the terminal application on the PC.

// User types 1 on the keyboard + turning ON the green LED!

// Using types 2 on the keyboard and turning OFF the green LED!

// Finding out which COM port the MSP430 UART maps to on the PC!

// TeraTerm 🡪 Showing the active COM ports within the application!

**#include** <msp430fr6989.h>

**#include** <stdio.h>

**#define** FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

**#define** RXFLAG UCRXIFG // Receive flag of UART

**#define** TXFLAG UCTXIFG // Transmit flag of UART

**#define** TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

**#define** RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

**#define** redLED BIT0 // Red LED is located at P1.0 on the MSP430

**#define** greenLED BIT7 // Green LED is located at P9.7 on the MSP 430

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that returns a byte; if nothing is returned then it returns NULL

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

**unsigned** **char** temp;

// Returns NULL if no byte was ever received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

// If something was received then it returns it

temp = RXBUFFER;

**return** temp;

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

**volatile** **unsigned** **int** i,j;

**unsigned** **char** ch, input;

// LEDs Configuration

P1DIR |= redLED;

P9DIR |= greenLED;

// LEDs OFF

P1OUT &= ~redLED;

P9OUT &= ~greenLED;

**for**(;;){

// Transmit characters from 0 to 9 over UART Communication while also toggling the RED LED

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

uart\_write\_char(ch);

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

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

P1OUT ^= redLED;

**for**(j = 0; j < 20000; j++);

}

// Receives data from keyboard and stores it

input = uart\_read\_char();

// if received data is 1 then turn on green LED, otherwise if data received is 2 then turn off green LED

**if**(input == '1'){

P9OUT |= greenLED;

}**else** **if**(input == '2')

P9OUT &= ~ greenLED;

}

}

// **8.2: Sending Unsigned 16-Bit Integers over UART**

// Using UART Connection 🡪 Interested in sending 16-Bit unsigned numbers to the PC!

// Sending measurements and inspect them or log them on the PC!

// void uart write uint16(unsigned int n);

// 16-Bit Unsigned Integer 🡪 Range of [0-65,535]

// Function Disassemble 🡪 the integer into digits and sending the corresponding ASCII values! One by One!

// Modifying the code with the goal of sending incremental numbers to the terminal:

**#include** <msp430fr6989.h>

**#include** <stdio.h>

**#define** FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

**#define** RXFLAG UCRXIFG // Receive flag of UART

**#define** TXFLAG UCTXIFG // Transmit flag of UART

**#define** TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

**#define** RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

**#define** redLED BIT0 // Red LED is located at P1.0 on the MSP430

**#define** greenLED BIT7 // Green LED is located at P9.7 on the MSP 430

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that returns a byte; if nothing is returned then it returns NULL

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

**unsigned** **char** temp;

// Returns NULL if no byte was ever received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

// If something was received then it returns it

temp = RXBUFFER;

**return** temp;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

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

**volatile** **unsigned** **int** digit = 0;

**int** k = 0,l = 0;

**int** array[5];

// Extracts digit by digit of the input number and stores it in an array

**do**{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}**while**(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

**for**(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

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

// Carry return

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

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Initializes the UART communication using ACLK @ 32KHz with a baud rate 4800

Initialize\_UART();

**volatile** **unsigned** **int** i,j;

**for**(;;){

// Transmits integers from 0 to 65535 over UART Communication

**for**(i = 0; i <= 65535; i++){

uart\_write\_uint16(i);

**for**(j = 0; j <= 40000;j++ );

}

}

}

// **8.3: Sending an ASCII String over UART**

// Transmitting a string over the UART connection.

// void uart write string(char \* str);

// Using a string of any size and calling the function (uart write char()) for sending ASCII characters!

// char mystring[] = "UART Transmission Begins...";

**#include** <msp430fr6989.h>

**#include** <stdio.h>

**#define** FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

**#define** RXFLAG UCRXIFG // Receive flag of UART

**#define** TXFLAG UCTXIFG // Transmit flag of UART

**#define** TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

**#define** RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

**#define** redLED BIT0 // Red LED is located at P1.0 on the MSP430

**#define** greenLED BIT7 // Green LED is located at P9.7 on the MSP 430

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

UCA1CTLW0 &= ~UCSWRST;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that returns a byte; if nothing is returned then it returns NULL

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

**unsigned** **char** temp;

// Returns NULL if no byte was ever received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

// If something was received then it returns it

temp = RXBUFFER;

**return** temp;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

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

**volatile** **unsigned** **int** digit = 0;

**int** k = 0,l = 0;

**int** array[5];

// Extracts digit by digit of the input number and stores it in an array

**do**{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}**while**(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

**for**(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

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

// Carry return

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

}

// Function that transmit a string (character array) through UART communication

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

**unsigned** **int** i = 0;

// Transmits character by character in the UART communication until it reaches the NULL terminated

**while**( str[i] != '\0'){

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

i++;

}

// New Line

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

// Carry Return

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

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Initializes the UART communication using SMCLK @ 1.048 MHz with oversampling at a baud rate of 9600

Initialize\_UART();

**volatile** **unsigned** **int** i,j;

// String to be transmitted to the PC

**char** mystring[] = "UART Transmission Begins...";

**for**(;;){

// Transmit the string continuously with a delay loop

uart\_write\_string(mystring);

**for**(j = 0; j < 40000; j++);

}

}

// **8.4: Changing the Configuration**

// Modification of the UART configuration by making two changes!

// Using ACLK with the crystal frequency of 32 KHz as the clock source (rather than SMCLK)

// Setting up a baud rate of 4800 instead of 9600.

// Generation of the receiver clock signal @ 16 x 4800 Hz starting with a clock signal @ 32,768 Hz.

// Two Changes in Setup Configuration: (A) Using a suitable value of UCSSEL for selecting ACLK as the clock source; (B) Finding the new values of the dividers and modulators!

// Doing the changes in a new UART initialization function {Initialize\_UART\_2()}

+ testing it by transmitting incrementing numbers (0,1,2,…) in an infinite loop!

// Configuring ACLK based on 32 KHz crystal!

// Changing the settings of the terminal application to a baud rate of 4800!

**#include** <msp430fr6989.h>

**#include** <stdio.h>

**#define** FLAGS UCA1IFG // Variable that possesses Transmit & Receive Flags of UART

**#define** RXFLAG UCRXIFG // Receive flag of UART

**#define** TXFLAG UCTXIFG // Transmit flag of UART

**#define** TXBUFFER UCA1TXBUF // Transmit buffer contains the transmit byte of UART

**#define** RXBUFFER UCA1RXBUF // Receive buffer contains the received byte of UART

**#define** redLED BIT0 // Red LED is located at P1.0 on the MSP430

**#define** greenLED BIT7 // Green LED is located at P9.7 on the MSP 430

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

// Popular Configuration of UART

// 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 to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Sets the clock source to SMCLK

UCA1CTLW0 |= UCSSEL\_2;

// Configures the dividers and modulators of the clock with the popular configuration of UART

// Configurations can be found in the Family User's Guide Page 783

UCA1BRW = 6;

UCA1MCTLW = UCBRS5|UCBRS1|UCBRF3|UCBRF2|UCBRF0|UCOS16;

// Enables transmission/reception to start

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: ACLK @ 32KHz with no oversampling

**void** **Initialize\_UART\_2**(**void**){

// Divert pins to UART functionality to enable transmission/reception of data between PC and MCU

P3SEL1 &= ~(BIT4|BIT5);

P3SEL0 |= (BIT4|BIT5);

// Configures ACLK to 32K

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

// Sets the clock source to ACLK

UCA1CTLW0 |= UCSSEL\_1;

// Configure the clock dividers and modulators

// UCBR=6, UCBRF=13, UCBRS=0x22, UCOS16=1 (oversampling)

UCA1BRW = 6;

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

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

UCA1CTLW0 &= ~UCSWRST;

}

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

// Ongoing transmission to be over

**while** ( (FLAGS & TXFLAG)==0 ) {}

// Write the input byte into the buffer to display in the terminal

TXBUFFER = ch;

}

// Function that returns a byte; if nothing is returned then it returns NULL

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

**unsigned** **char** temp;

// Returns NULL if no byte was ever received

**if**( (FLAGS & RXFLAG) == 0)

**return** NULL;

// If something was received then it returns it

temp = RXBUFFER;

**return** temp;

}

// Function that takes a 16-bit unsigned integer and transmit it through UART communication

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

**volatile** **unsigned** **int** digit = 0;

**int** k = 0,l = 0;

**int** array[5];

// Extracts digit by digit of the input number and stores it in an array

**do**{

digit = n % 10; // Extracts the digit

array[k] = digit; // Stores it

n = n/10; // Truncates the input number

k++;

l++;

}**while**(n != 0);

// Extracts from the array backwardly to transmit the each digit through UART

**for**(k = l - 1 ; k >= 0;k--)

uart\_write\_char(array[k] + '0');

// New line

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

// Carry return

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

}

// Function that transmit a string (character array) through UART communication

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

**unsigned** **int** i = 0;

// Transmits character by character in the UART communication until it reaches the NULL terminated

**while**( str[i] != '\0'){

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

i++;

}

// New Line

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

// Carry Return

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

}

/\*\*

\* main.c

\*/

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

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

**volatile** **unsigned** **int** i,j;

// Initializes the UART communication using ACLK @ 32KHz with a baud rate 4800

Initialize\_UART\_2();

**for**(;;){

// Prints numbers from 0 to 65535 through UART Communication

**for**(i = 0; i <= 65535; i++){

uart\_write\_uint16(i);

**for**(j = 0; j < 40000; j++);

}

}

}