Lab 6 – UART

# Youssef Samwel

# [yo800238@ucf.edu](mailto:yo800238@ucf.edu)

# EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

# 2/29/2024



# **Project Description**

In this lab, we will learn using the UART interface and program the backchannel UART link that connects our board to the PC.

# **2.0 Experiment Code**

#include <msp430fr6989.h>

#include <stdint.h>

#include <stdbool.h>

#include <math.h>

#include <string.h>

#include "images.h"

#define BUTTON1 BIT1

#define BUTTON2 BIT2

#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\_UART\_2(void);

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

unsigned char uart\_read\_char(void);

void LightRedLED(bool state);

void LightGreenLED(bool state);

void ToggleRedLED();

void ToggleGreenLED();

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

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

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

/\*\*

 \* main.c

 \*/

void Part1Demo()

{

    // char data[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

    int i = 0;

    for (;;)

    {

        for (i = 0; i < 10; i++)

        {

            uart\_write\_string("Hello from MSP430, UART Function\r\n");

            uart\_write\_uint16(1 << i);

            uart\_write\_string("\r\n");

        }

    }

}

void terminal\_clear()

{

    uart\_write\_string("\033[2J");

}

void terminal\_reset\_cursor()

{

    uart\_write\_string("\033[1;1H");

}

void terminal\_move\_up(uint8\_t count)

{

    while (count > 0)

    {

        uart\_write\_string("\033[1A");

        count--;

    }

}

void terminal\_show\_cursor(uint8\_t show)

{

#define CSI "\e["

    if (show)

    {

        uart\_write\_string(CSI "?25h");

    }

    else

    {

        uart\_write\_string(CSI "?25l");

    }

#undef CSI

}

void terminal\_display\_takeoff()

{

    char hexBytes[] = {0xf0, 0x9f, 0x9b, 0xab};

    uart\_write\_char(hexBytes[0]);

    uart\_write\_char(hexBytes[1]);

    uart\_write\_char(hexBytes[2]);

    uart\_write\_char(hexBytes[3]);

}

void terminal\_display\_logo()

{

    unsigned char hexBytes[] = {0xe2, 0x9c, 0x88, 0xef, 0xb8, 0x8f};

    uart\_write\_char(hexBytes[0]);

    uart\_write\_char(hexBytes[1]);

    uart\_write\_char(hexBytes[2]);

    uart\_write\_char(hexBytes[3]);

    uart\_write\_char(hexBytes[4]);

    uart\_write\_char(hexBytes[5]);

}

void application\_display()

{

    terminal\_clear();

    terminal\_reset\_cursor();

    terminal\_show\_cursor(false);

    uart\_write\_string("ORLANDO EXECUTIVE AIRPORT RUNWAY CONTROL ");

    terminal\_display\_logo();

    uart\_write\_string("\r\n\r\n");

    uart\_write\_string("             \tRunway 1\tRunway 2\r\n");

    uart\_write\_string("Request (RQ):\t  1\t\t  3\r\n");

    uart\_write\_string("Forfeit (FF):\t  7\t\t  9\r\n");

    uart\_write\_string("\n\n\n\n");

    uart\_write\_string("------------\t\t\t------------\r\n");

    uart\_write\_string("  RUNWAY 1\t\t\t  RUNWAY 2\r\n");

    uart\_write\_string("------------\t\t\t------------\r\n");

    uart\_write\_string("\n\n\n\r\r\n\r\n\r\n");

    uart\_write\_string(IMG\_PLANE);

    terminal\_move\_up(16);

}

static bool runway1\_requested = false;

static bool runway2\_requested = false;

static bool runway1\_granted = false;

static bool runway2\_granted = false;

static bool runway1\_inquiry = false;

static bool runway2\_inquiry = false;

void application\_display\_requested\_status()

{

    terminal\_move\_up(3);

    uart\_write\_char('\r');

    uart\_write\_string(runway1\_requested ? "REQUESTED\t\t\t" : "         \t\t\t");

    uart\_write\_string(runway2\_requested ? "REQUESTED\t\t\t" : "         \t\t\t");

    uart\_write\_string("\r\n\n\n");

}

void application\_bootup\_sequence()

{

    terminal\_clear();

    terminal\_reset\_cursor();

    terminal\_show\_cursor(false);

    uart\_write\_string(IMG\_ORLANDO);

    \_\_delay\_cycles(6e5);

    uart\_write\_string(IMG\_UCF\_KNIGHTS);

    uart\_write\_string("\r\n\r\n");

    uart\_write\_string(IMG\_F22);

    uart\_write\_string("\r\n\r\nPress any key to continue...\r\n");

    uint8\_t input = NULL;

    do

    {

        input = uart\_read\_char();

    } while (!input);

}

void application\_display\_grant\_status()

{

    terminal\_move\_up(2);

    uart\_write\_char('\r');

    if (runway1\_granted)

    {

        uart\_write\_string("In Use ");

        terminal\_display\_takeoff();

        uart\_write\_string("\t\t\t");

    }

    else

    {

        uart\_write\_string("        \t\t\t");

    }

    if (runway2\_granted)

    {

        uart\_write\_string("In Use ");

        terminal\_display\_takeoff();

        uart\_write\_string("\t\t\t\t");

    }

    else

    {

        uart\_write\_string("        \t\t\t\t");

    }

    uart\_write\_string("\r\n\n");

}

void application\_display\_inquiry\_status()

{

    uart\_write\_char('\r');

    uart\_write\_string(runway1\_inquiry ? "\*\*\* Inquiry \*\*\*\t\t\t" : "               \t\t\t");

    uart\_write\_string(runway2\_inquiry ? "\*\*\* Inquiry \*\*\*\t\t\t" : "               \t\t\t");

}

void application\_display\_reset\_status()

{

    application\_display\_inquiry\_status();

    application\_display\_grant\_status();

    application\_display\_requested\_status();

}

void application\_demo()

{

    TA0CCR0 = 10000;

    TA0CCTL0 |= CCIE;

    TA0CCTL0 &= ~CCIFG;

    TA0CTL = TASSEL\_\_ACLK | ID\_\_1 | MC\_\_UP | TACLR;

    \_enable\_interrupts();

    application\_bootup\_sequence();

    application\_display();

    while (true)

    {

        uint8\_t input = NULL;

        do

        {

            input = uart\_read\_char();

        } while (!input);

        uart\_write\_char('\r');

        switch (input)

        {

        // request runway

        case '1':

        case '3':

            // if runway is already requested do nothing

            // otherwise:

            // set request flag for runway 1

            // print on terminal "Requested" under Runway 1 label

            runway1\_requested |= input == '1';

            runway2\_requested |= input == '3';

            LightRedLED(runway1\_requested);

            LightGreenLED(runway2\_requested);

            // display requested line

            application\_display\_requested\_status();

            break;

        // release runway

        case '7':

        case '9':

            runway1\_requested &= (input == '7') ? false : true;

            runway2\_requested &= (input == '9') ? false : true;

            // clear granted access if the pilot forfieted access

            runway1\_granted &= runway1\_requested;

            runway2\_granted &= runway2\_requested;

            // clear inquiry

            runway1\_inquiry &= runway1\_requested;

            runway2\_inquiry &= runway2\_requested;

            // update display

            application\_display\_reset\_status();

            break;

        }

    }

}

#pragma vector = PORT1\_VECTOR

\_\_interrupt void ISR\_PORT1()

{

    if (P1IFG & BUTTON1)

    {

        // button 1 has been pressed

        // this signals that the controller has decided to release runway 1

        // if the runway was requested we then can release the runway

        // otherwise do nothing

        if (runway1\_requested && !runway1\_granted)

        {

            // runway was previously requeseted; grant access

            runway1\_granted = true;

            application\_display\_grant\_status();

        }

        // if runway1 is already granted, then the operator is asking operator

        // to see if runway is still in use

        else if (runway1\_granted)

        {

            runway1\_inquiry = true;

            application\_display\_inquiry\_status();

        }

    }

    if (P1IFG & BUTTON2)

    {

        // button 2 has been pressed

        // this signals that the controller has decided to release runway 2

        // if the runway was requested we then can release the runway

        // otherwise do nothing

        if (runway2\_requested && !runway2\_granted)

        {

            // runway was previously requeseted; grant access

            runway2\_granted = true;

            application\_display\_grant\_status();

        }

        // if runway2 is already granted, then the operator is asking operator

        // to see if runway is still in use

        else if (runway2\_granted)

        {

            runway2\_inquiry = true;

            application\_display\_inquiry\_status();

        }

    }

    // attempted debouncing

    \_\_delay\_cycles(2e4);

    P1IFG &= ~(BUTTON1 | BUTTON2);

}

#pragma vector = TIMER0\_A0\_VECTOR

\_\_interrupt void ISR\_T0A0()

{

    if (runway1\_granted)

    {

        // runway 1 was granted, flash the red led

        ToggleRedLED();

    }

    else if (!runway1\_requested)

    {

        // runway 1 is no longer granted or requested therefore turn off led

        LightRedLED(false);

    }

    if (runway2\_granted)

    {

        // runway 2 was granted, flash the green led

        ToggleGreenLED();

    }

    else if (!runway2\_requested)

    {

        // runway 2 is no longer granted or requested therefore turn off led

        LightGreenLED(false);

    }

    TA0CCTL0 &= ~CCIFG;

}

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

    // Set input direction for buttons

    P1DIR &= ~(BUTTON1 | BUTTON2);

    // enable pull-up resistors to avoid false triggers caused by static fields

    P1REN |= BUTTON1 | BUTTON2;

    // pull high buttons

    P1OUT |= BUTTON1 | BUTTON2;

    // interrupt on falling edge

    P1IES = BUTTON1 | BUTTON2;

    // reset interrupt flags

    P1IFG &= (BUTTON1 | BUTTON2);

    // enable PORT1 interrupts

    P1IE |= BUTTON1 | BUTTON2;

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

    Initialize\_UART();

    LightRedLED(false);

    LightGreenLED(false);

    application\_demo();

    \_low\_power\_mode\_3();

    return 0;

}

void uart\_write\_uint16(unsigned int number)

{

    static const char ascii\_digits[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

    while (number)

    {

        uint8\_t digit = number % 10;

        number /= 10;

        uart\_write\_char(ascii\_digits[digit]);

    }

}

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

{

    size\_t length = strlen(str);

    size\_t i;

    for (i = 0; i < length; i++)

    {

        uart\_write\_char(str[i]);

    }

}

void uart\_write\_char(unsigned char ch)

{

    // Wait for any ongoing transmission to complete

    while ((FLAGS & TXFLAG) == 0)

    {

    }

    // Copy the byte to the transmit buffer

    TXBUFFER = ch; // Tx flag goes to 0 and Tx begins!

    return;

}

// The function returns the byte; if none received, returns null character

uint8\_t uart\_read\_char(void)

{

    uint8\_t temp;

    // Return null character (ASCII=0) if no byte was received

    if ((FLAGS & RXFLAG) == 0)

        return 0;

    // Otherwise, copy the received byte (this clears the flag) and return it

    temp = RXBUFFER;

    return temp;

}

// Configure UART to the popular configuration

// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit

// no flow control, oversampling reception

// Clock: SMCLK @ 1 MHz (1,000,000 Hz)

void Initialize\_UART(void)

{

    // Configure pins to UART functionality

    P3SEL1 &= ~(BIT4 | BIT5);

    P3SEL0 |= (BIT4 | BIT5);

    // Main configuration register

    UCA1CTLW0 = UCSWRST; // Engage reset; change all the fields to zero

    // Most fields in this register, when set to zero, correspond to the

    // popular configuration

    UCA1CTLW0 |= UCSSEL\_\_SMCLK; // Set clock to SMCLK

    // Configure the clock dividers and modulators (and enable oversampling)

    UCA1BRW = 6; // divider

    // Modulators: UCBRF = 8 = 1000 --> UCBRF3 (bit #3)

    // UCBRS = 0x20 = 0010 0000 = UCBRS5 (bit #5)

    UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;

    // Exit the reset state

    UCA1CTLW0 &= ~UCSWRST;

}

void Initialize\_UART\_2(void)

{

    // Configure pins to UART functionality

    P3SEL1 &= ~(BIT4 | BIT5);

    P3SEL0 |= (BIT4 | BIT5);

    // Main configuration register

    UCA1CTLW0 = UCSWRST; // Engage reset; change all the fields to zero

    // Most fields in this register, when set to zero, correspond to the

    // popular configuration

    UCA1CTLW0 |= UCSSEL\_\_ACLK; // Set clock to ACLK

    // Configure the clock dividers and modulators (and enable oversampling)

    UCA1BRW = 6; // divider

    // Modulators: UCBRF = 8 = 1000 --> UCBRF3 (bit #3)

    // UCBRS = 0x20 = 0010 0000 = UCBRS5 (bit #5)

    UCA1MCTLW = UCBRF6 | UCBRS5;

    // Exit the reset state

    UCA1CTLW0 &= ~UCSWRST;

}

void LightRedLED(bool state)

{

    if (state)

        P1OUT |= redLED;

    else

        P1OUT &= ~redLED;

}

void LightGreenLED(bool state)

{

    if (state)

        P9OUT |= greenLED;

    else

        P9OUT &= ~greenLED;

}

void ToggleRedLED() { P1OUT ^= redLED; }

void ToggleGreenLED() { P9OUT ^= greenLED; }

// Configures ACLK to 32 KHz crystal

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;

}

# **3.0 Student Q&A**

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

UART is the protocol for serial communications between devices, the eUSCI is a hardware-based implementation of UART for the MSP430 microcontroller.

2. What is the backchannel UART?

A secondary UART channel which is used to avoid interfering with the main channel communication. This allows multiple independent UART communication to run in parallel.

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

These lines configure the PORT3 pins to be used for UART mode (the default is general IO, they also could be used by the LCD display).

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.

Since we cannot directly divide the clock to 9600 Hz, we can divide the clock twice at rate slightly higher than 9600 and slight lower than 9600. For example, with 104 division we get 9615 Hz and with 105 we get 9523 Hz. The eUSCI modulator combines these two frequencies to average 9600 Hz.

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

1200 Hz

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?

1200 \* 16 = 19200 Hz