Lab 3: Using the UART

Objectives

- Use the TM4C123GH6PM UART to transmit and receive data using polling and interrupt handling techniques
- Communicate with Bluetooth enabled devices utilizing the HM10 module

Introduction

Universal Asynchronous Receiver/Transmitter or **UART** for short represents the hardware circuitry (module) being used for the **serial communication**. IO pins dedicated to the UART serial communication module as shown in **Figure 1** are TX & RX.

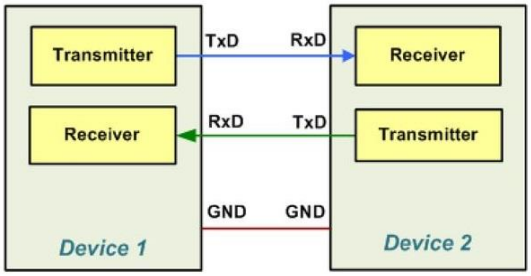

Figure 1

The TI ARM Tiva TM4C123GH6PM has **eight on-chip UART ports**. They are designated as **UART0-UART7**. In the TI LaunchPad, the UART0 port of the TM4C123GH6PM is connected to the Stellaris ICDI (In-Circuit Debug Interface), which is connected to a USB connector. The ICDI USB is located on the right of the slide switch and is labeled as Debug. It is on the short side of the board; see **Figure 2**. This **ICDI USB connection** contains **three distinct functions**:

- The programming (downloading) using LM Flash Programming software,
- The debugging using JTAG, and
- The use as a **virtual COM port** (using a UART-to-USB bridge)

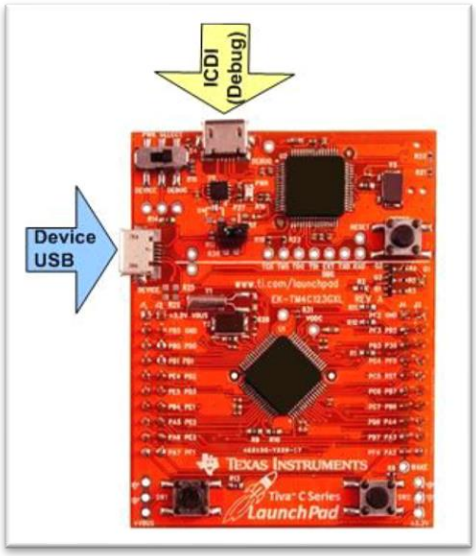

Figure 2

When the USB cable connects the Tiva LaunchPad, the device driver at the host PC establishes a **virtual connection** between the PC and the UART0 of the TM4C123GH6PM device. On the LaunchPad, the connection appears as **UART0**. On the host PC, it appears as a **COM port** and will work with communication software on the PC such as a **terminal emulator** (e.g. TeraTerm, PuTTY).

Experiment 1: Sending messages to the PC using UART0 (Polling)

Sending "Yes" to UART0 on TI ARM Launchpad (TM4C123GH6PM). UART0 is connected to the UART-to-USB bridge on the ICDI module of the launchpad board. Use TeraTerm to see the message "Yes" on the PC. Configure TeraTerm connection to use “COM X: USB Serial Device”. Change the connection settings (Setup➔ Serial Port) to be *9600,N,1* (Baud rate: 9600, No parity and 1 stop bit).

```
#include <stdint.h>
#include "tm4c123gh6pm.h"
void UART0Tx(char const c);
void delayMs(int n);

int main(void)
{
    SYSCTL->RCGCUART |= 1;  /* provide clock to UART0 */
    SYSCTL->RCGCGPIO |= 1;  /* enable clock to PORTA */

    /* UART0 initialization */
    UART0->CTL = 0;         /* disable UART0 */
    UART0->IBRD = 104;      /* 16MHz/(16*9600 baud rate) = 104.1666666666 */
    UART0->FBRD = 11;       /* fraction part = 0.1666666*64+0.5 = 11.1666666 */
    UART0->CC = 0;          /* use system clock */
    UART0->LCRH = 0x60;     /* 8-bit, no parity, 1-stop bit, no FIFO */
    UART0->CTL = 0x301;     /* enable UART0(bit0), TXE(bit8), RXE(bit9) */

    /* UART0 TX0 and RX0 use PA1 and PA0. Set them up. */
    GPIOA->DEN = 0x03;      /* Make PA0 and PA1 as digital */
    GPIOA->AFSEL = 0x03;    /* Use PA0,PA1 alternate function */
    GPIOA->PCTL = 0x11;     /* configure PA0 and PA1 for UART */

    delayMs(25);            /* wait for output line to stabilize */

    for(;;)
    {
        UART0Tx('Y');
        UART0Tx('e');
        UART0Tx('s');
        UART0Tx(' ');
    }
}

/* UART0 Transmit */
/* This function waits until the transmit buffer is available then writes */
/* the character in it. It does not wait for transmission to complete */
void UART0Tx(char const c)
{
    while((UART0->FR & 0x20)!= 0){} // Wait until Tx buffer is not full
    UART0->DR = c;        // Write byte
}

// Append the delay functions here.
```

Experiment 2: Receive messages from PC using UART0 (Polling)

Read data from PC terminal emulator software to UART0 of the MCU and display it on the tri-color LEDs. The LEDs are connected to Port F3-1. Press any A-z, a-z, 0-9 key at the terminal emulator and see the least significant 3 bits of the ASCII value in binary displayed on LEDs of PORTF.

```
#include <stdint.h>
#include "tm4c123gh6pm.h"

char UART0Rx(void);
void delayMs(int n);

int main(void)
{
    char c;

    SYSCTL->RCGCUART |= 1;     /* provide clock to UART0 */
    SYSCTL->RCGCGPIO |= 1;     /* enable clock to PORTA */
    SYSCTL->RCGCGPIO |= 0x20; /* enable clock to PORTF */

    /* UART0 initialization */
    UART0->CTL = 0;            /* disable UART0 */
    UART0->IBRD = 104;         /* 16MHz/(16*9600 baud rate) = 104.1666666666 */
    UART0->FBRD = 11;          /* fraction part= 0.1666666*64+0.5 = 11.1666666 */
    UART0->CC = 0;             /* use system clock */
    UART0->LCRH = 0x60;        /* 8-bit, no parity, 1-stop bit, no FIFO */
    UART0->CTL = 0x301;        /* enable UART0, TXE, RXE */

    /* UART0 TX0 and RX0 use PA1 and PA0. Set them up. */
    GPIOA->DEN = 0x03;         /* Make PA0 and PA1 as digital */
    GPIOA->AFSEL = 0x03;       /* Use PA0,PA1 alternate function */
    GPIOA->PCTL = 0x11;        /* configure PA0 and PA1 for UART */

    GPIOF->DIR = 0x0E;         /* configure PortF pins 3,2,1 to control LEDs */
    GPIOF->DEN = 0x0E;
    GPIOF->DATA = 0;           /* turn LEDs off */


    for(;;){
        c = UART0Rx();         /* get a character from UART */
        GPIOF->DATA = c << 1; /* shift left & write least sig. 3 bits to LEDs */
    }
}

/* UART0 Receive */
/* This function waits until a character is received then returns it. */
char UART0Rx(void)
{
    char c;
    while((UART0->FR & 0x10) != 0){} /* wait until Rx buffer is not empty */
    c = UART0->DR;                   /* read the received data */
    return c;                        /* and return it */
}

// Append the delay functions here.
```

Experiment 3: Receive messages from PC using UART0 (Interrupts)

Pressing a key at the terminal emulator causes the PC to send the ASCII code of the key to the Tiva C TM4C123 LaunchPad. When the character is received by UART0, the interrupt handler reads the character and write its least significant 3 bits on LEDs of PORTF.

```
/* Read data from UART0 and display it at the tri-color LEDs. The LEDs are
connected to Port F 3-1. Press any A-z, a-z, 0-9 key at the terminal emulator
and see ASCII value in binary is displayed on LEDs of PORTF. */

#include "tm4c123gh6pm.h"
int main(void)
{
    SYSCTL->RCGCUART |= 1;     /* provide clock to UART0 */
    SYSCTL->RCGCGPIO |= 1;     /* enable clock to PORTA */
    SYSCTL->RCGCGPIO |= 0x20; /* enable clock to PORTF */

    /* UART0 initialization */
    UART0->CTL = 0;            /* disable UART0 */
    UART0->IBRD = 104;         /* 16MHz/(16*9600 baud rate) = 104.1666666666 */
    UART0->FBRD = 11;          /* fraction part= 0.1666666*64+0.5 = 11.1666666 */
    UART0->CC = 0;             /* use system clock */
    UART0->LCRH = 0x60;        /* 8-bit, no parity, 1-stop bit, no FIFO */
    UART0->IM |= 0x0010;       /* enable RX interrupt */
    UART0->CTL = 0x301;        /* enable UART0, TXE, RXE */

    /* UART0 TX0 and RX0 use PA1 and PA0. Set them up. */
    GPIOA->DEN = 0x03;         /* Make PA0 and PA1 as digital */
    GPIOA->AFSEL = 0x03;       /* Use PA0,PA1 alternate function */
    GPIOA->PCTL = 0x11;        /* configure PA0 and PA1 for UART */

    GPIOF->DIR = 0x0E;         /* configure Port F to control the LEDs */
    GPIOF->DEN = 0x0E;
    GPIOF->DATA = 0;           /* turn LEDs off */

    /* enable interrupt in NVIC and set priority to 3 */
    NVIC->IP[5] = 3 << 5;        /* set interrupt no 5 priority to 3 */
    NVIC->ISER[0] |= 0x00000020; /* enable IRQ5 for UART0 */

    __enable_irq(); /* global enable IRQs */

    for(;;){}
}

void UART0_Handler(void)
{
    volatile int readback;

    char c;
    if (UART0->MIS & 0x0010)    /* if a receive interrupt has occurred */
    {
        c = UART0->DR;          /* read the received data */
        GPIOF->DATA = c << 1;   /* shift left and write it to LEDs */
        UART0->ICR = 0x0010;    /* clear Rx interrupt flag */
        readback = UART0->ICR;  /* a read to force clearing of interrupt flag */
    }
```

```
    else
    {
        /* should not get here. But if it does, */
        UART0->ICR = UART0->MIS;  /* clear all interrupt flags */
        readback = UART0->ICR;  /* a read to force clearing of interrupt flag */
    }
}

// Remember to disable IRQs in SystemInit function in generated startup C src.
```

Experiment 4: HM-10 UART to Bluetooth Bridge test

We need to develop an embedded application that gets a command from a smart phone over Bluetooth. For that, the UART to Bluetooth Bridge (HC-05 or HC-06 or HM-10 or JDY-08) is used.

We'll start by testing and debugging our Bluetooth module by connecting it directly to the PC. Use a smart Phone to send arbitrary text to the BT module using a Bluetooth Terminal App, and receive this text on Teraterm on PC. See **Figure 3**.

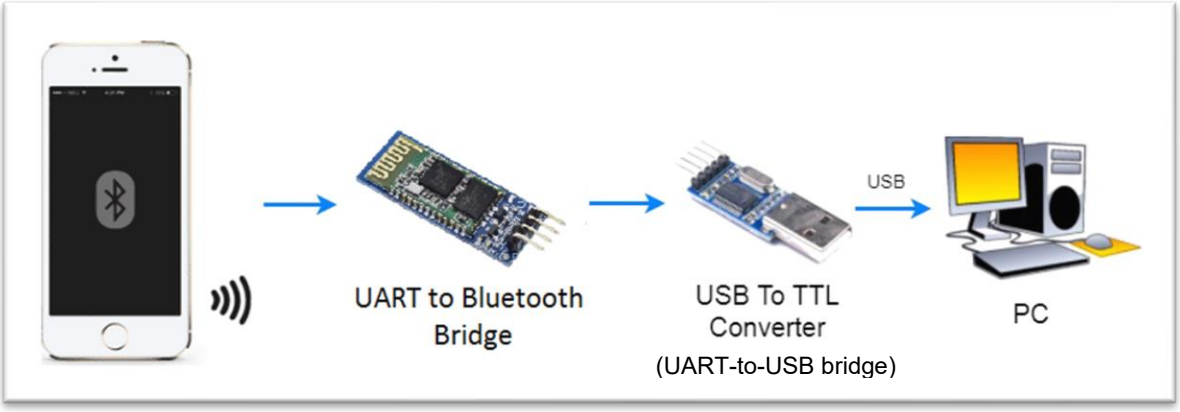

Figure 3

Notes:

i. For debugging purposes, you are given a USB-UART cable or a USB-to-TTL module to connect the BT module to a PC. See **Figure 4**.

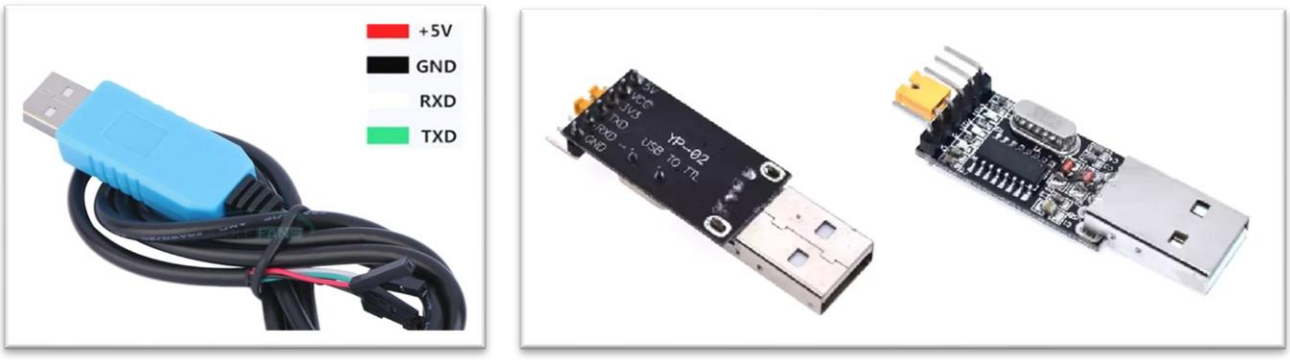

Figure 4

ii. If the USB-UART cable or a USB-to-TTL module is not identified on your PC please try one of the drivers below:

https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers
https://sparks.gogo.co.nz/ch340.html
https://www.connectix.nl/pl2303hxa-phased-out-since-2012-please-contact-your-supplier-solved/

iii. The BT module default passcode for pairing is "1234" for HC-05 and HC-06 and HM-10, and "123456" for JDY-08.

iv. The BT module is configured to operate @ 9600,N,1 for HC-05 and HC-06 and HM-10, and 115200,N,1 for JDY-08.

v. To send a command from an **Android Phone** you need to download and install a Bluetooth Terminal App from Google Play store. Any Bluetooth Terminal App will serve well, except for HM-10 module which needs *Arduino Bluetooth Controller (HM-10 Module)* App, and JDY-08 module which needs *Serial Bluetooth Terminal* App.

vi. To send a command from an **iPhone** you need to download and install *ArduinoBlue* App from Apple app store and use the HM-10 Bluetooth module.

vii. BT module pinout:

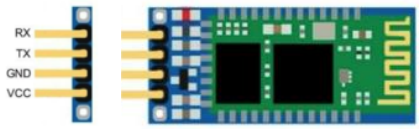

Figure 5

viii. Watch for the I/O pins voltage levels and supply voltage for BT module. The BT module RXD and TXD pins are 3.3V.

ix. The BT module has a default name of HC06 or HC05 or HMSoft or JDY-08, but it can have another name after some of last semester students altered its default name. So just search for any near Bluetooth device with any name.

x. You can use any smart phone you have with you, or you can borrow one from the workshop.

Experiment 5: Receive data from phone using UART1 and send it to PC using UART0

Develop an embedded application that receives characters from a smart phone using the BT module, passes these characters to the MCU through any UART other than UART0 (e.g., UART1), and finally sends these characters to PC Teraterm software using UART0, as displayed in **Figure 6**. If the character received is R or G or B, the red/green/blue LED should light up.

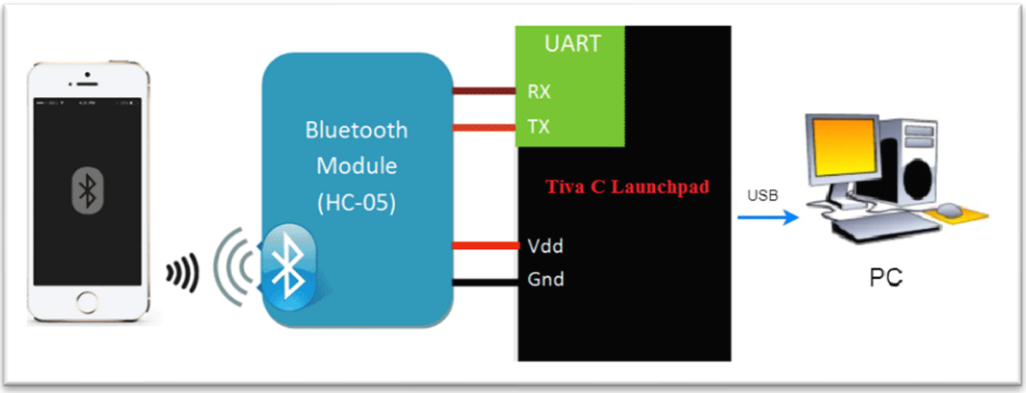

Figure 6

Complete the code provided below.

```
/* Receive characters from phone using UART1 and send it to PC using UART0 */

#include "TM4C123GH6PM.h"

void UART0Tx(char c);

int main(void)
{
        /* provide clock to UART0 */
        /* enable clock to PORTA */

        /* provide clock to UART1 */
        /* enable clock to PORTB */

        /* enable clock to PORTF */

        /* UART0 initialization */

        /* UART0 TX0 and RX0 use PA1 and PA0. Set them up. */

        /* UART1 initialization, enabling RX interrupt */

        /* UART1 TX0 and RX0 use PB1 and PB0. Set them up. */

        /* configure Port F pins 3,2,1 to control the LEDs */

        /* enable UART1 interrupt in NVIC and set priority to 3 */

        /* global enable IRQs */

        while (1) {}
}


void UART1_Handler(void)
{
        /* handle a receive interrupt and pass to UART0 and LEDs */
}


void UART0Tx(char c)
{
    /* send a character to UART0 */
}
```

Lab Report [10 pts]
(Deadline: Monday of next week 11:59 pm) (Individual submission)

1. [1.5 pts] Provide your group C code of experiment 5 conducted in the lab.

2. [3.5 pts] Repeat lab experiment 1 but this time using UART TX interrupts rather than polling. Provide your code along with a small sized video of the application running on the Launchpad.

3. [5 pts] Develop an embedded application that gets a command from a smart phone over a UART to Bluetooth Bridge (HC-05 or HC-06 or HM-10 or JDY-08).
The following commands must be implemented (sent from the smartphone to the TIVA C Launchpad board) to turn on/off each individual RGB color component on the LED:

AT+G=0	→	Turn Green color component off
AT+G=1	→	Turn Green color component on
AT+P=0	→	Turn Purple color component off
AT+P=1	→	Turn Purple color component on
AT+R=0	→	Turn Red color component off
AT+R=1	→	Turn Red color component on
AT+Y=0	→	Turn Yellow color component off
AT+Y=1	→	Turn Yellow color component on
AT+ON	→	Turn all LED color components on
AT+OFF	→	Turn all LED color components off

Every command is terminated by CR character (ASCII code: 13).
Provide your code along with a small sized video of the application running on the Launchpad.

Notes:

a. Connect HM-10 module to any UART other than UART0 and UART1 (e.g., UART5).
b. Use interrupts only (no polling).
c. Keep the interrupt handler as small as possible
d. Implement a producer-consumer pattern. In this pattern, the ISR acts as a producer and the main function acts as a consumer. They communicate through a buffer. The buffer is filled by the ISR (the producer) till CR character is received. At this event, a flag is set. The main function (the consumer) waits for the flag to be set before it reads from the buffer. Once read, the main function clears the flag.