Cover Page

Lab 2 Report

Implementation of two-way communication between **Bluetooth** and **TivaC**

ECE3436, DMU

Spring 2024

Wankang Zhai

2232923

contents

1. Overview

In this experiment, we implemented two-way communication between Bluetooth and the TivaC development version. Implement input in Bittly, accept in Putty, and accept in Bittly at the same time. Vice versa is the same.

Our achievements include:

1) Transmit characters to the remote computer through UART0 and UART1 protocols

2) Set up two interrupts to implement two-way communication for the UART0 protocol and UART1 protocol respectively.

3) Echo back the same characters to the remote computer each UART protocol

This experiment deepened our understanding of interrupts and made us realize the reusability of UART protocols and ports.

2. Description of Tasks and Results

Task1：Send the characters through Bluetooth to a remote computer

2.1.1 Enable the UART and GPIO port

To enable Bluetooth functionality, we must first activate the Bluetooth port. Bluetooth comprises four ports: VCC for accessing 3.3V, the GROUND port for grounding, the PORTB port, and the PB port for receiving and transmitting data. To activate these ports, we utilize the System Control Peripheral Enable command to activate both UART and GPIO ports. Figures 1 and 2 illustrate the activation process.

```
,,
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);

ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
```

Figure 1

```
//
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
GPIOPinConfigure(GPIO_PB0_U1RX);
GPIOPinConfigure(GPIO_PB1_U1TX);
ROM_GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
//
// Configure the UART for 115,200, 8-N-1 operation.
//
ROM_UARTConfigSetExpClk(UART0_BASE, ROM_SysCtlClockGet(), 115200,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));
ROM_UARTConfigSetExpClk(UART1_BASE, ROM_SysCtlClockGet(), 9600,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                         UART_CONFIG_PAR_NONE));
//
// Enable the UART interrupt.
//
ROM_IntEnable(INT_UART0);
ROM_IntEnable(INT_UART1);

ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
ROM_UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);

//
```

Figure 2

We set the Baud Rate of the Bluetooth UART protocol to 9600 to ensure that information can be transmitted at the same frequency.

2.1.2 Set UARTSEND For UART1

We need to set up a function for the uart1 interface to ensure the transmission of characters through the UART1 protocol. As shown in Figure 3, we put the address of UART1_BASE into the function so that it can transmit characters to the computer through the UART1 protocol.

```
47 UARTSend1(const uint8_t *pui8Buffer, uint32_t ui32Count)
48 {
49     //
50     // Loop while there are more characters to send.
51     //
52     while(ui32Count--)
53     {
54         //
55         // Write the next character to the UART.
56         //
57         ROM_UARTCharPutNonBlocking(UART1_BASE, *pui8Buffer++);
58
59     }
60 }
```

Figure 3

To transmit characters to both ends, the code logic for UART1 and UART0 is identical. Each character is sent using the UARTCharPutNonBlocking function. We implement character transmission to Bluetooth in Lab2 using a string sending mechanism similar to that in Lab1, as illustrated in Figure 4.

```
UARTSend1((uint8_t *)" Characters ", strlen("Characters "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"in the ", strlen("in the "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"remote bluetooth ", strlen("remote bluetooth "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"serial ", strlen("serial "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"terminal ", strlen("terminal "));
SysCtlDelay(SysCtlClockGet()/(400));
UARTSend1((uint8_t *)"and see ", strlen("and see "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"the ", strlen("the "));
SysCtlDelay(SysCtlClockGet()/(400));

UARTSend1((uint8_t *)"color change", strlen("color change"));
SysCtlDelay(SysCtlClockGet()/(400));

//
// Loop forever echoing data through the UART.
```

Figure 4

By initializing the UART window and improving the UARTSend function, we have realized sending two strings of characters to different terminals through two different protocols and displaying them on the computer terminal.

Task 2: Receive characters entered and displayed, Realizing The Echo Back

2.2.1 Creating two interrupts

In order to display on the local serial terminal, we create two interrupts in startup_ccs.c program. Firstly we change the UART1 RX and TX name in the predefinition part and then set the external declaration for the interrupt handler used by the application. we named it as UART1Handler. See how I was doing in Figure 5.

```
//
// External declaration for the interrupt handler used by the application.
//
//*****************************************************************************
extern void UART0Handler(void);
extern void UART1Handler(void);

//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
```

Figure 5

Through this operation, we can generate two interrupt functions for the uart0 port and uart1 port. We further configure two UARTHandler functions to pass the bluetooth port data through the TX port of the UART1 port, and the RX port of the UART0 port accepts it. The data of the TivaC port is transmitted through the TX port of the UART0 port, and the RX port of the UART0 port receives it.

2.2.2 Realize Echo Back and set the details in the UARTHandler Function.

First, we set UART1Handler to follow the order and content design of UARTHander. But there is one difference, we change all UART0_BASE to UART1_BASE. The purpose of this is because we want to execute the entire program by setting two interrupts, and the UART0 and UART1 protocols are performed in parallel. As shown in Figure 6.

```
void
UART1Handler(void)
{
    uint32_t ui32Status;

    //
    // Get the interrrupt status.
    //

    ui32Status = ROM_UARTIntStatus(UART1_BASE, true);


    //
    // Clear the asserted interrupts.
    //
    ROM_UARTIntClear(UART1_BASE, ui32Status);



    //
    // Loop while there are characters in the receive FIFO.
    //
    while(ROM_UARTCharsAvail(UART1_BASE))
    {
        //
        // Read the next character from the UART and write it back to the UART.
        //
```

Figure 6

One thing worth noting is that we added a string to this while loop to ensure that there will be no overlap in each transmission. When two characters overlap, encoding errors will occur, leading to garbled characters. Moreover, if the transmission rate is too fast, it will also cause the protocol to be transmitted out of order, and problems will occur in the order. Therefore, in Figure 7, we first define a char variable, and then input the char variable to TX, passing each char as a unit. When we pass UART1_BASE to ourselves, we also pass it to

UART0_BASE, successfully achieving two-way transmission.

```
58    // Loop while there are characters in the receive FIFO.
59    //
70    while(ROM_UARTCharsAvail(UART1_BASE))
71    {
72        //
73        // Read the next character from the UART and write it back to the UART.
74        //
75        i++;
76        char happy = ROM_UARTCharGetNonBlocking(UART1_BASE);
77        ROM_UARTCharPutNonBlocking(UART0_BASE,
78                happy );
79        ROM_UARTCharPutNonBlocking(UART1_BASE,
80                happy );
81
82
83
84        //
85        // Blink the LED to show a character transfer is occuring.
86        //
```

Figure 7

Task3：Realize green -> blue -> red -> green…

We combined the Lab1 code to realize this Lab. When we input into Bluetooth terminal there will be a light change, when we input into TivaC terminal, there is also a light change. So we add the counters in two interrupt functions. Figure 8 shows how the counter was done in the function.

```
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2);

    //
    // Delay for 1 millisecond.  Each SysCtlDelay is about 3 clocks.
    //

    }
    if(i%3==2){

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0);

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, GPIO_PIN_1);


    //
    // Turn off the LED
    //
    }
    if(i%3==0){

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);

    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, GPIO_PIN_3);
```

Figure 8

C code in Figure 8 tells how the light changed when we sent information into UART.

3. Description of the whole Lab

When we use the bluetooth module, we need to initialize it first, because we want bluetooth to implement the UART1 protocol, and at the same time use the GPIO B0 and B1

ports when connecting, these ports need to be initialized. After initializing the port we activated the Interrupt program of UART1. And two serial port communications are implemented in UARTCharsAvail. The specific logic is detailed in Figure 9.

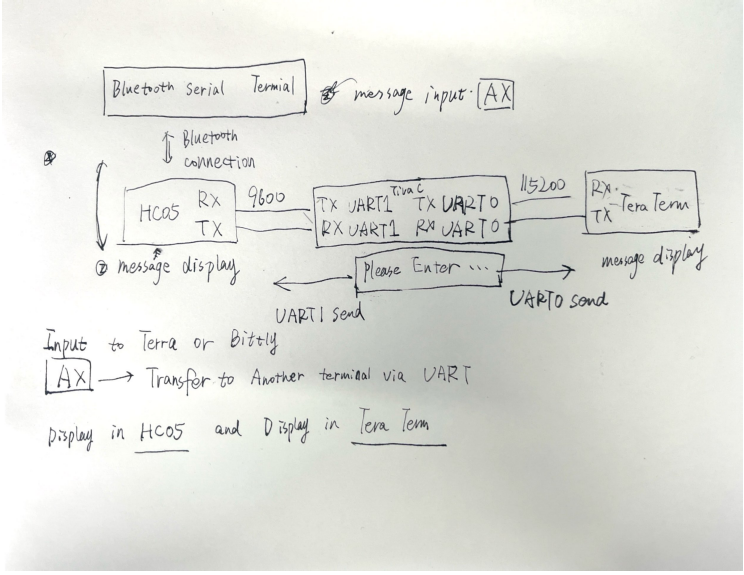

Figure 9

4. Overview

In this experiment, we successfully configured the Bluetooth module and applied it to serial communication. We have a deeper understanding of the interrupt mechanism, and realized the link between the Bluetooth serial port and the TivaC serial port through interrupt. Laying the foundation for our future robot projects