Cover Page

Lab 1 Report
A preliminary understanding of UART and INTERRUPT

ECE3436, DMU

Spring 2024

Wankang Zhai

2232923

contents

1. Overview

In this experiment, we made significant progress. We managed to install TI's CCS, configure Putty effectively, and utilize the computer as a terminal. Moreover, we successfully sent messages and ensured that both Putty and the TivaC development version were configured with the same port and receiving rate.

Our achievements include:

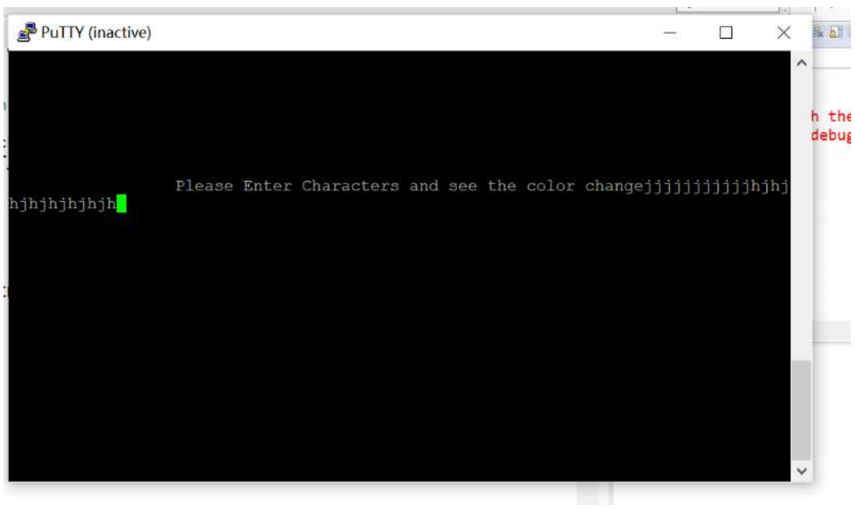
1) Configuring the PortF interface and **clock peripheral**, and successfully lighting the LED using **GPIOPinWrite**.
2) Mastering the technique of transmitting characters to **UART TX through Putty and setting the baud rate.**
3) Learning how to utilize **UART TX** output via Interrupt, TivaC UART RX receive, and TivaC UART TX send back to the terminal, thereby achieving output return in Putty's display interface.

These accomplishments mark significant whole milestones.

2. Description of Tasks and Results

Task1：Configure UART0 and send a message to a putty

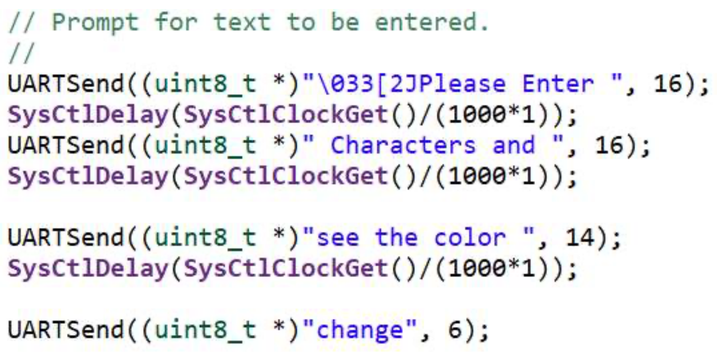
In this task, we effectively displayed the message "Please enter characters and observe the color change:" in the serial terminal window, as depicted in Figure 1.



```
                    Please Enter Characters and see the color changejjjjjjjjjjjhjhj
hjhjhjhjhjh
```

Figure 1

My c code was written as Figure 2



```
// Prompt for text to be entered.
//
UARTSend((uint8_t *)"\033[2JPlease Enter ", 16);
SysCtlDelay(SysCtlClockGet()/(1000*1));
UARTSend((uint8_t *)" Characters and ", 16);
SysCtlDelay(SysCtlClockGet()/(1000*1));

UARTSend((uint8_t *)"see the color ", 14);
SysCtlDelay(SysCtlClockGet()/(1000*1));

UARTSend((uint8_t *)"change", 6);
```

Figure 2

In completing this task, I introduced an innovative approach by incorporating delays into each sentence of UART communication. I encountered a limitation wherein it was not feasible to output more than 16 characters through UARTSend. As a workaround, I divided the message into multiple inputs, each restricted to 16 characters. However, I observed that without introducing delays, the output would often become disordered or even garbled initially. This inconsistency arose due to the overlap of the two transmissions caused by the rapid rate of information being transmitted each time.

To address this issue, I strategically inserted a Delay function between each UARTSend operation. Although the interval was brief, it provided the microprocessor with sufficient time to complete each transmission, thereby mitigating the risk of overlap and ensuring orderly output. Through this experience, I concluded that incorporating delays during serial communication offers several advantages:

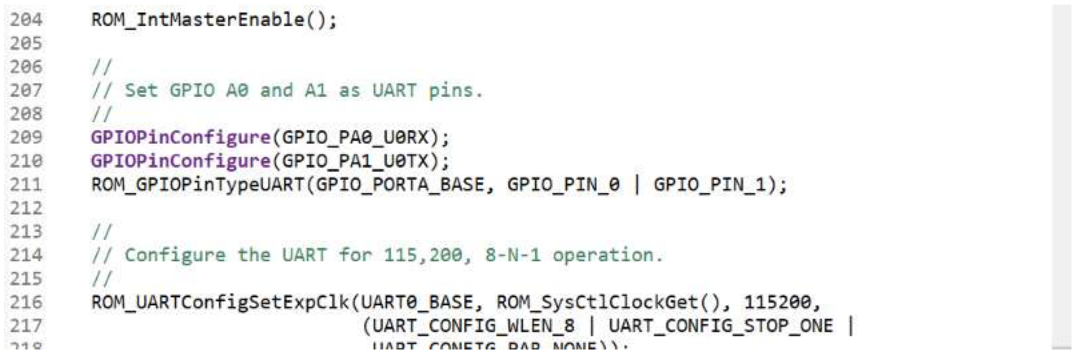
Waiting for data readiness: By waiting for a brief period before sending or receiving data, I ensured that all data was fully prepared for transmission, minimizing the likelihood of missing or incomplete data.

Avoiding data conflicts: The introduction of delays helped mitigate data conflicts, which could otherwise lead to garbled code or transmission errors. By spacing out transmissions, I minimized the risk of overlap and maintained data integrity throughout the communication process.

Task 2: Implement sequential lighting changes based on the UART interrupt

2.1 Setting UART and Interrupt initial parameters

First, we set the initial value of uart. Here we set the value of Baud rate to be the same as putty. One so that the two devices can communicate. Figure 3 shows how we set it up.

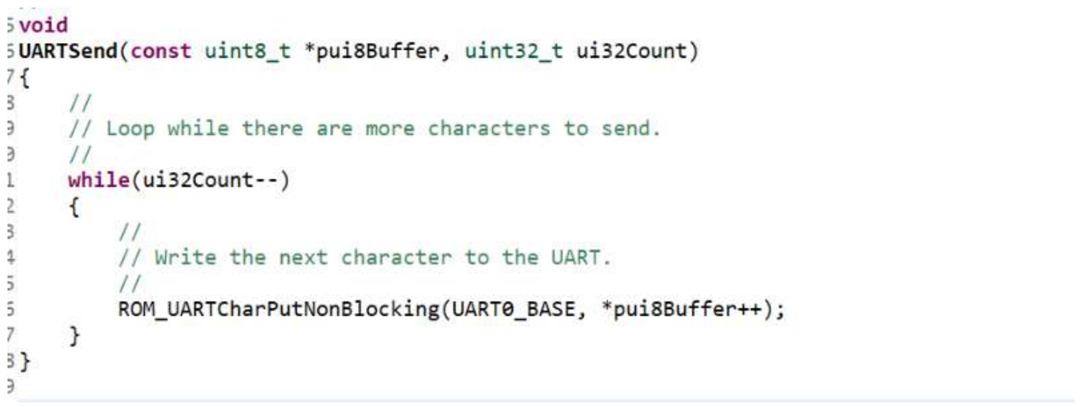


```
ROM_IntMasterEnable();

//
// Set GPIO A0 and A1 as UART pins.
//
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
ROM_GPIOPinTypeUART(GPIO_PORTA_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));
```

Figure 3

We set the baud rate of UART to 115200, which is the same as putty, to ensure that we can transmit synchronously.
GPIOPinConfigure ensures uart activation.

2.2 UartSend ：Send characters through UART



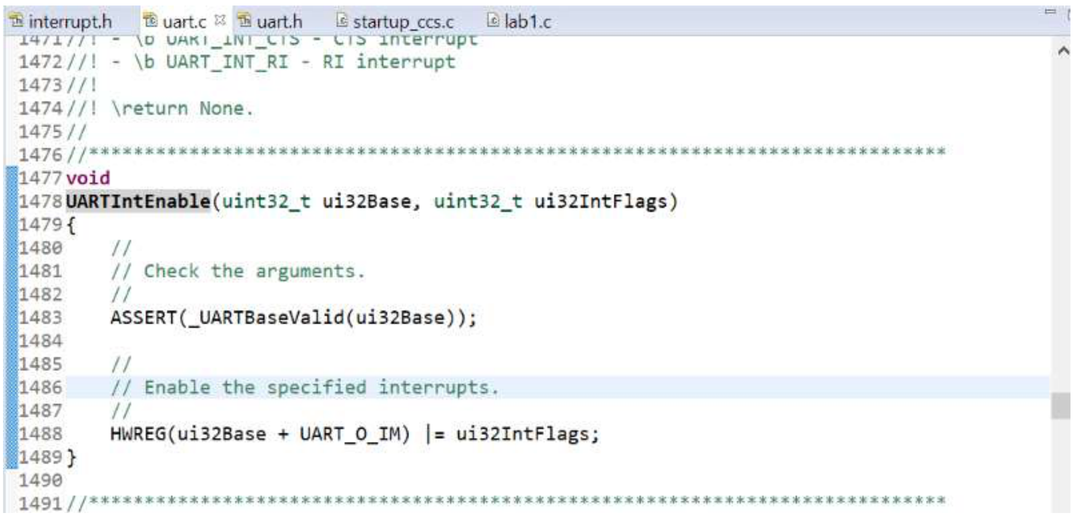
```
void
UARTSend(const uint8_t *pui8Buffer, uint32_t ui32Count)
{
    //
    // Loop while there are more characters to send.
    //
    while(ui32Count--)
    {
        //
        // Write the next character to the UART.
        //
        ROM_UARTCharPutNonBlocking(UART0_BASE, *pui8Buffer++);
    }
}
```

Figure 4

Each time the characters ui32Count are transmitted one by one through UART TX, note that the maximum value here is 32 bits. So in Task1, we pass the character multiple times.

2.3 UARTIntHandler： Interrupt Handler, do the processing when interrupt.

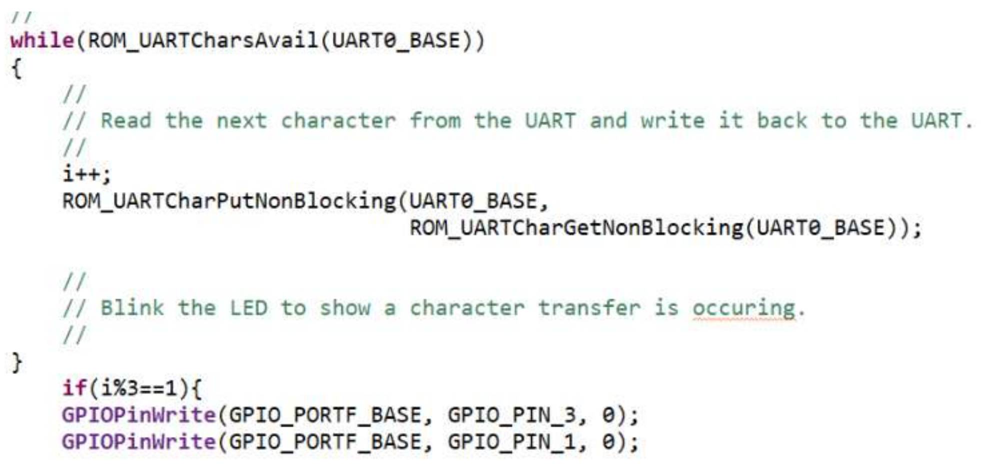
This function will be called when an Interrupt occurs. We find the definition of this function in the uart.c file. Coming to Figure 5, enable the interrupt specified in the UART module so that the interrupt handler is triggered when the corresponding conditions are met.



```
//! - \b UART_INT_CTS - CTS interrupt
//! - \b UART_INT_RI - RI interrupt
//!
//! \return None.
//
//*****************************************************************************
void
UARTIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags)
{
    //
    // Check the arguments.
    //
    ASSERT(_UARTBaseValid(ui32Base));

    //
    // Enable the specified interrupts.
    //
    HWREG(ui32Base + UART_O_IM) |= ui32IntFlags;
}

//*****************************************************************************
```

Figure 5

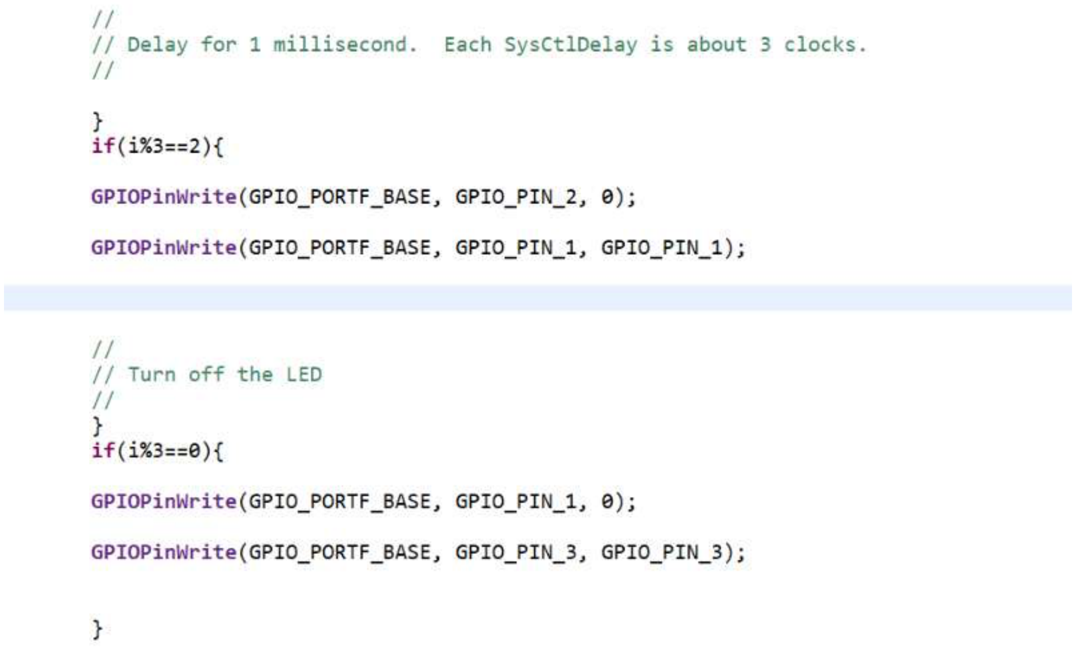
In our main file, we implement the following method to trigger an interrupt and light up LEDs of different colors every time a character is entered. In Figure 6, first, we define i=0 outside and set i++ in the while loop of the interrupt



```
//
while(ROM_UARTCharsAvail(UART0_BASE))
{
    //
    // Read the next character from the UART and write it back to the UART.
    //
    i++;
    ROM_UARTCharPutNonBlocking(UART0_BASE,
                               ROM_UARTCharGetNonBlocking(UART0_BASE));

    //
    // Blink the LED to show a character transfer is occuring.
    //
}
    if(i%3==1){
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1, 0);
```

Figure 6

Because there are three LEDs cycling, we use the modulo operator (%) to obtain the remainder when dividing by 3. Based on this remainder, we implement logic to illuminate different colored LEDs for each interrupt as follows:



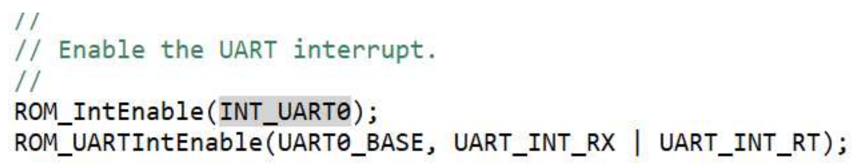
```
//
// Delay for 1 millisecond.  Each SysCtlDelay is about 3 clocks.
//

}
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 7

1. Initially, the green LED is lit.
2. Upon the first interrupt, we set bits 1 and 3 to 0, and set bit 2 to illuminate the blue LED.
3. Upon the second interrupt, we set bit 2 to 0 and set bit 1 to illuminate the red LED.
4. Upon the third interrupt, we set bit 1 to 0 and set bit 3 to illuminate the green LED again.

By following this method, we ensure that different LEDs are sequentially illuminated for each interrupt.

2.4 How do we initially set an Interrupt



```
//
// Enable the UART interrupt.
//
ROM_IntEnable(INT_UART0);
ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
```

Figure 8

We enable the UART interrupt as shown in Figure 8.

3. conclusion

In this experiment, we successfully initialized UART and Interrupt. More importantly, we understand the principle of interrupt and understand that through the UARTIntHandler function, we can perform further operations on interrupt. We used the counter method to complete this experimental task and lay the foundation for future Milestones.