Lab 7 Report

Ansh Vivek Roll No: 210020002
Suhas Reddy Roll No: 210020030

October 10, 2024

Aim

The goal of this lab is to program the microcontroller to transmit specific characters over UART based on button presses. If SW1 is pressed, transmit "F0", and if SW2 is pressed, transmit "AA". The UART should be configured with a baud rate of 9600 and odd parity.

Additionally, the program should listen for incoming data on the UART. If "AA" is received, the green LED should light up. If "F0" is received, the blue LED should light up. If any transmission or reception errors occur, the red LED should be activated. Test the program by communicating with a neighboring group's microcontroller.

Procedure

The program is divided into two main parts: transmission and reception. Transmission is handled by a dedicated transmit function, which fills the UART's data register with the data to be sent. This data is determined by which switch is pressed, SW1 or SW2, and is transmitted accordingly by calling the UART transmit function. GPIO interrupts are set up to detect the button presses.

Reception is continuously enabled in the program, where the UART receive function is called inside a while(1) loop. This function checks the receive status, and when data is received, the corresponding LED is lit based on the data in the UART data register. Additionally, for testing purposes, the received data is re-transmitted in a loopback configuration.

An interrupt handler for the UART is also implemented, which is triggered if any errors occur during transmission or reception. When an error is detected, the red LED is turned on.

Code

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

uint8_t TxWord;
uint8_t RxWord;

#define MASK_BITS 0x11 // pins for which interrupts should be enabled
#define RED_LED 1
#define BLUE_LED 2
#define GREEN_LED 3

void UART_Init()
{

    SYSCTL_RCGCUART_R |=(1<<2);//Enable UART Module 2
    UART2_ICR_R|=(1<<10)|(1<<9)|(1<<8)|(1<<7)|(1<<6)|(1<<5)|(1<<4)|(1<<1);
    UART2_IM_R|=(1<<10)|(1<<9)|(1<<8)|(1<<7)|(1<<6)|(1<<5)|(1<<4)|(1<<1);
    //Baud Calc: for 9600: Integer part: 104=0x68 and frac part=11= 0xB
    UART2_IBRD_R=0x68;//Integer Divider for baud rate
    UART2_FBRD_R=0xB;//Fraction Divider for baud rate

    UART2_LCRH_R|=(1<<6)|(1<<5);//8-bit Word length
    UART2_LCRH_R&=~(1<<2);//Odd parity
    UART2_LCRH_R|=(1<<1);//parity enable
    UART2_CTL_R|=(1<<9);//Rx Enable
    UART2_CTL_R|=(1<<0);//UART Enable
}

void UART_Transmit(uint8_t TxWordF)
```

```
{
    UART2_CTL_R&=~(1<<0);//UART Disable
    UART2_DR_R=TxWordF;//Data Register
    UART2_CTL_R|=(1<<8);//Tx Enable
    UART2_CTL_R|=(1<<0);//UART Enable
}
void GPIO_PortF_Handler()//change name in startup.ccs
{
    GPIO_PORTF_ICR_R = 0x11; // Clear the interrupt flag
    if(GPIO_PORTF_DATA_R & 0x01)//SW2 pressed
    {
        UART_Transmit(0xF0);
    }

    else if(GPIO_PORTF_DATA_R & 0x10)//SW1 pressed
    {
        UART_Transmit(0xAA);
    }

}
void UART_Receive()
{
    if((UART2_RIS_R & (1<<4))==(1<<4))//check if receive is done
    {
        RxWord=UART2_DR_R;
        UART_Transmit(RxWord);//to check
        if(RxWord==0xAA)
        {
        GPIO_PORTF_DATA_R |= (1<<BLUE_LED);//Turn ON Blue LED
        GPIO_PORTF_DATA_R &= ~(1<<GREEN_LED);//Turn OFF Green LED
        GPIO_PORTF_DATA_R &= ~(1<<RED_LED);//Turn OFF RED LED
        }
        else if(RxWord==0xF0)
        {
        GPIO_PORTF_DATA_R |= (1<<GREEN_LED);//Turn ON Green LED
        GPIO_PORTF_DATA_R &= ~(1<<BLUE_LED);//Turn OFF BLUE LED
          GPIO_PORTF_DATA_R &= ~(1<<RED_LED);//Turn OFF RED LED
```

```
        }
        UART2_ICR_R|=(1<<10)|(1<<9)|(1<<8)|(1<<7)|(1<<6)|(1<<5)|(1<<4)|(1<<1);
        UART2_CTL_R&=~(1<<0);//UART Disable
        UART2_LCRH_R&=~(1<<4);//Flush the FIFO
        UART2_CTL_R|=(1<<0);//UART Enable
    }
}

void UART_Interrupt_Handler()//change name in startup.ccs
{
    if((UART2_RIS_R & (1<<10))==(1<<10))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<9))==(1<<9))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<8))==(1<<8))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<7))==(1<<7))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<6))==(1<<6))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<5))==(1<<5))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    if((UART2_RIS_R & (1<<4))==(1<<4))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
```

```
    if((UART2_RIS_R & (1<<1))==(1<<1))
      {
          GPIO_PORTF_DATA_R |= (1<<RED_LED);//Turn ON RED LED
      }
    UART2_ICR_R|=(1<<10)|(1<<9)|(1<<8)|(1<<7)|(1<<6)|(1<<5)|(1<<4)|(1<<1);
    UART2_CTL_R&=~(1<<0);//UART Disable
}


int main()
{

    SYSCTL_RCGCGPIO_R |= (1<<5)|(1<<3);    /* Enable system clock to PORTF
                                              and PORTD */
    SYSCTL_RCGCUART_R |=(1<<2);//Enable UART Module 2
    GPIO_PORTF_LOCK_R = 0x4C4F434B;     /* unlock commit register */
    GPIO_PORTF_CR_R = 0xFF;             /* make PORTF configurable */
    GPIO_PORTF_DIR_R &= ~0x11; // Set Pin 0 and 4 as input
    GPIO_PORTF_DIR_R |= 0x0E; // Set Pin 1,2,3(LEDs) as output
    GPIO_PORTF_PUR_R |= 0x11;  // Enable the internal pull-up resistor for
                               Pin 0 and 4
    GPIO_PORTF_DEN_R |= 0x1F;  // Digital enable Pin 0,1,2,3,4

    //gpio Pins initialisation(PORTD Pins)
        GPIO_PORTD_LOCK_R = 0x4C4F434B;     /* unlock commit register */
        GPIO_PORTD_CR_R = 0xFF;             /* make PORTD configurable */
        GPIO_PORTD_DIR_R &= ~0x40; // Set Pin 6 as input
        GPIO_PORTD_DIR_R |= 0x80; // Set Pin 7 as output
        GPIO_PORTD_PUR_R |= 0x40;  // Enable the internal pull-up resistor
                                   for Pin 6
        GPIO_PORTD_DEN_R |= (0x80)|(0x40);  // Digital enable Pin 6 and 7

//Alternate function settings:
    GPIO_PORTD_AFSEL_R |= (1<<6)|(1<<7); /* PD6 and PD7 set for alternate
                                        function */
    GPIO_PORTD_PCTL_R |= 0x11000000;    /* make PD7 and PD6 for UART2 Rx and
                                        Tx */
```

```
    //GPIO interrupts
    GPIO_PORTF_IM_R &= ~MASK_BITS; // mask interrupt by clearing bits
    GPIO_PORTF_IS_R &= ~MASK_BITS; // edge sensitive interrupts
    GPIO_PORTF_IBE_R &= ~MASK_BITS; // interrupt NOT on both edges
    GPIO_PORTF_IEV_R &= ~MASK_BITS; // falling edge trigger

    //Set protity using NVIC
    NVIC_PRI7_R = (NVIC_PRI7_R & 0xF1FFFFFF) | (3 << 21); // bits 21-23 for
                                                          interrupt 30 (port F)
    NVIC_EN0_R |= 1 << 30; // enable interrupt 30 (PORTF)

    GPIO_PORTF_ICR_R = MASK_BITS; // clear any prior interrupt
    GPIO_PORTF_IM_R |= MASK_BITS; // unmask interrupt by setting bits

    UART_Init();
    while(1)
    {
        UART_Receive();
    }

}
```

Output

Below are the images showing the transmission and reception waveforms for both the "F0" and "AA" data:

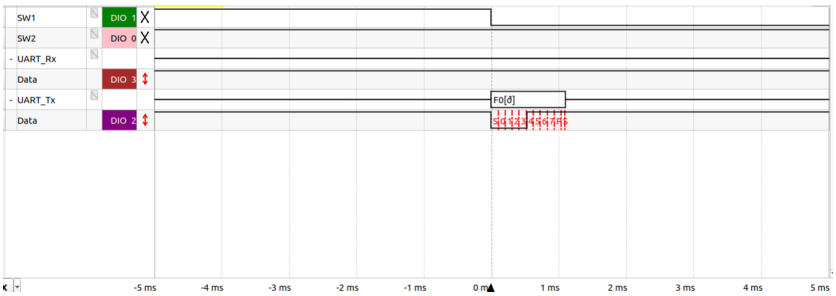

Figure 1: F0 Transmission

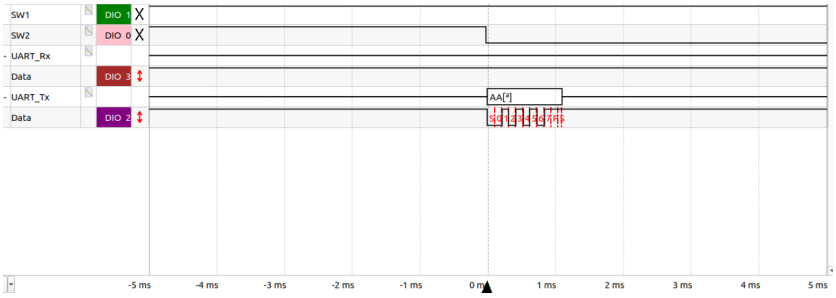

Figure 2: AA Transmission

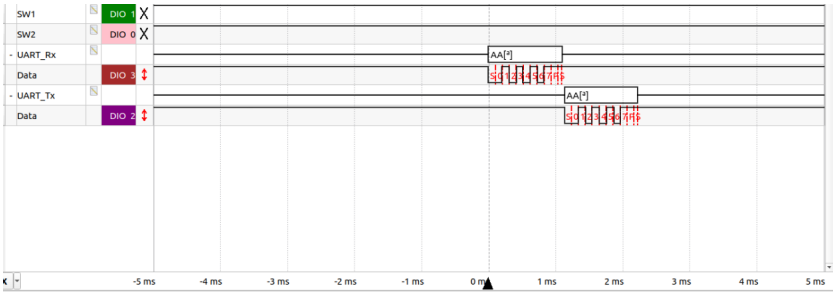

Figure 3: AA Reception and Transmission

Discussion and Conclusion

In this lab, we successfully established UART communication between two boards with appropriate transmission and reception configurations. We were able to send and receive the designated characters ("F0" and "AA") and verify the UART configuration. The LEDs responded correctly to the received data, and errors were indicated by the red LED. During testing, we also observed the correct start and stop bits in the transmitted data using waveform analysis.

FIFO was disabled in this case, as we were dealing with single-byte transmissions, so there was no need for buffering multiple bytes.