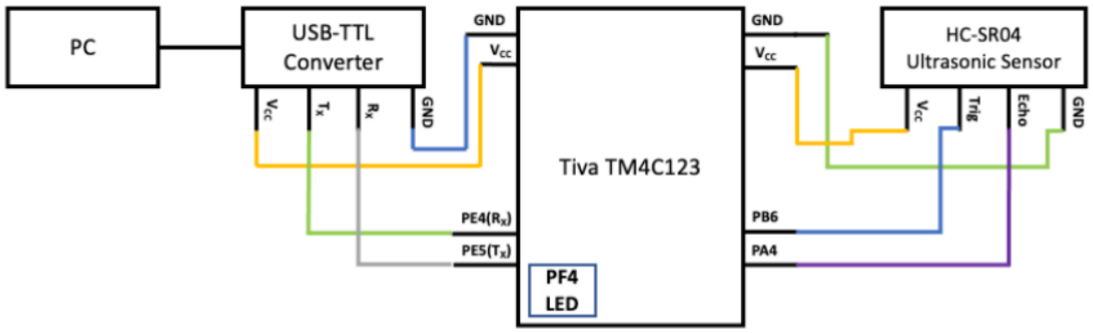
Introduction

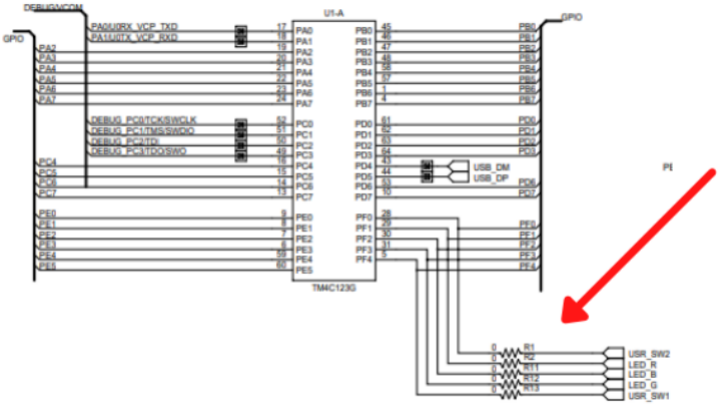
General Purpose of This Laboratory :

In this laboratory application, I sent the data I received from the HC-SR04 ultrasonic sensor to the computer using UART serial communication and observed these values on the computer screen using the Putty application.

Pin/Port Selection :

I have made my connections as in the image below with little differences.
Since I observed that the HC SR04 sensor does not work properly with +3.3 V, I fed this sensor with +5 V using an external source.
I did not connect the Vcc part because the USB TTL converter is fed from the computer. I just connected the GND line.
I didn't use PF4 as the leds are connected to PF3, PF2 and PF1 in Tiva tm4c123 launch pad.
I connected the TX-RX of the USB-TTL converter to PE4(TX)-PE5(RX).
Since there is no timer peripheral to PA4, I connected the sensor's ECHO pin to PB6 (see Tiva™ C Series TM4C123GH6PM Datasheet on page 650 - Table 10-2. GPIO Pins and Alternate Functions).





The image from Tiva c4 series user's manual from page 20 (Schematic Image)

Table 10-2. GPIO Pins and Alternate Functions (64LQFP)

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[3]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-
PB0	45	USB0ID	U1Rx	-	-	-	-	-	T2CCP0	-	-	-	-

The image from Tiva™ C Series TM4C123GH6PM Datasheet on page 650 - Table 10-2. GPIO Pins and Alternate Functions

Result And Discussion

The main purpose of the code I have written and this application I have prepared is to send the value received from the ultrasonic sensor to the computer via UART serial communication.It would be good to start my story about this laboratory application by explaining how I use the HC-SR04 ultrasonic sensor.

The signal applied from the Trig pin of the sensor provides an ultrasonic sound emission at a frequency of 40 kHz.
When this sound wave hits any object and returns to the sensor, the Echo pin becomes active.For time sensitive operations like in this application, we need a code that uses TM4C123's Timer interrupts. If I try to measure the time without using a method such as timer interrupt, the result I will get will not be very efficient.I have configured the TIMER0 peripheral to create a sensitive delay function.

The registers I want to mention in this configuration I set the CFG register to 0x4 for 16-bit conversion. I set the TAMR register to 0x17 to make it work in capture mode. Finally, I enabled the TIMER0 peripheral and finished my configuration.It would be nice if I mention the delayMicroSecond() function after configuration. The important point here is that the counter starts counting with a loop and the increase of each counter depends on the timeout flag being removed. For this, I had the RIS register of the TIMER constantly checked. After the timeout flag was removed, I cleared the timeout flag value thanks to the ICR register.

The distance measurement is performed with the measureDistance() function that I have prepared.The working logic of the measureDistance() function is as follows:
The first trigger pin becomes high, then thanks to the TIMER peripheral in the while loop, the echo pin performs the distance measurement by counting the rising edges during this time.While doing this, the timer capture flag is cleared with the ICR register, when the flag is removed (when the RIS register is equal to 4), the TAR register is kept in a variable. Since the capture flag changes with every logic change, I read the value from the TAR register twice in the same loop (while loop). I returned the value found by subtracting the falling edges from the total changed logic edge state.I processed the measureDistance( ) function in the main function by assigning it to the time variable and calculated the distance. Later, I sent this data to the PC via UART via COM connection.

Now, let me briefly explain how I use the UART serial communication protocol and peripheral on the launch pad.
By short definition, UART is a hardware communication protocol that uses asynchronous serial communication with configurable speed. Asynchronous means there is no clock signal to synchronize the output bits from the transmitting device going to the receiving end.It uses two buses, TX transmitting data and RX receiving data.

Let me now explain how I initialize the UART peripheral.While performing the UART communication, the baudrate (data transport rate) must be set first after the clock configurations connected to the UART peripharel are made.I assigned 104 to the IBRD register and 11 to the FBRD register to set the baudrate to 9600.
Since the data bit length is usually 8 and the frame is set as one stop bit, I have configured it this way.That's why I assigned 0x60 to the LCRH register. Thus, my data length became 8 bits.I declared that with CC register it will work with system clock of UART.Then I activated the UART with the CTL register. Finally ,I determined my tx & rx pins by configuring GPIO AFSEL and GPIO PCTL registers.

UART Functions :
printChar(unsigned char data) : If transmitter bus is available , it sends the data to UART Data register .
printingString(char *str) : If transmitter bus is available , it sends the data to UART Data register as byte byte .

Conclusion

While doing this lab experiment, I learned the UART serial communication down to the last detail. I learned how to configure the UART peripheral on the TIVA TM4C123 launch pad and how to transfer data with the UART. Initially, I read and configured the registers of the UART peripheral from the datasheet. But I failed. Later, I saw and corrected my mistakes by looking at the information on the internet.
Again, I learned the very important working principle of the TIMER peripheral and how to configure and use it on the TIVA TM4C123 launch pad. After learning these, it was much easier for me to learn the working principle of the HC-SR04 sensor. In this way, I learned how to write a sensitive delay function, which I think will be very useful for me. I think I understand all the issues I mentioned above very well. Thank you to all my teachers who contributed.

References

- https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#:~:text=By%20definition%2C%20UART%20is%20a,going%20to%20the%20receiving%20end.
- http://armofthings.com/2015/12/25/uart-in-tm4c123gh6pm-launchpad/
- https://microcontrollerslab.com/uart-communication-tm4c123-tiva-c-launchpad/
- https://www.ti.com/lit/ds/spms376e/spms376e.pdf?ts=1622656731588&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FEK-TM4C123GXL
- https://diyot.net/hc-sr04-ultrasonik-mesafe-sensoru/

MY YOUTUBE VIDEO FOR THE LAB 6

https://www.youtube.com/watch?v=pbC9lK_-fNs

Appendix

```
// Omer Karslioglu - 17050211061

#include "TM4C123GH6PM.h"
#include <stdio.h>

//---SYSTEM CONTROL REGISTERS---//
#define SYS_CTRL_RCGC2  (*((volatile unsigned long *)0x400FE108))   //offset of RCGC2 register is 0x108
#define CLK_GPIOF   0x20
//---GPIO-F REGISTER---//
#define PORTF_DATA  (*((volatile unsigned long *)0x40025038))   //offset of DATA register for red, blue, green is 0x38 [PF7:PF0::9:2]
#define PORTF_DIR   (*((volatile unsigned long *)0x40025400))   //offset of DIR register is 0x400
#define PORTF_DEN   (*((volatile unsigned long *)0x4002551C))   //offset of DEN register is 0x51C
//---PORT-F I/O---//
#define red 0x02
#define blue 0x04
#define green 0x08

uint32_t measureDistance(void);
void Timer0ACapture_init(void);
void delayMicroSecond(int time);
void init_UART(void);
void printChar(unsigned char data);
void printingString(char *str);
void Delay(unsigned long counter);

// global variables to store and display distance in cm
uint32_t time; //stores pulse on time */
uint32_t distance; // stores measured distance value
char mesg[20];  // string format of distance value

// main code to take distance measurement and send data to UART terminal
int main(void)
{
Timer0ACapture_init();  //initialize Timer0A in edge edge time
init_UART(); // initialize UART5 module to transmit data to computer

SYSCTL->RCGCGPIO |= 0x20;                   // initialize clock of Port F
GPIOF->DIR |=0x0E;         // set PF4 as a digial output pin
GPIOF->DEN |=0x1F;
PORTF_DATA = 0;
            while(1)
            {

            time = measureDistance(); // take pulse duration measurement
            distance = (time * 10625)/10000000; // convert pulse duration into distance
            sprintf(mesg, "Distance: %d cm \r\n",distance); //convert float type distance data into string
            printingString(mesg); //transmit data to computer */
            Delay(100);
            if(distance<=10){
                        PORTF_DATA = red;                                   // if distance smaller than 10 , red led high.*/
            }
            else if (distance<=20){
                        PORTF_DATA = blue;                                  // if distance is between 10-20 cm , blue led high*/
            }
            else if (distance<=30){
                        PORTF_DATA = green;                                 // if distance is between 20-30 cm , green led high*/
            }
            else{
                        PORTF_DATA=0x00;                                                        // if distance is more than 30, led off             */
            }

            }

}


// This function captures consecutive rising and falling edges of a periodic signal */
// from Timer Block 0 Timer A and returns the time difference (the period of the signal). */
uint32_t measureDistance(void)
{
   int lastEdge, thisEdge;

              // Given 10us trigger pulse */
              GPIOA->DATA &= ~(1<<4); // make trigger  pin high */
              delayMicroSecond(10); //10 seconds delay */
              GPIOA->DATA |= (1<<4); // make trigger  pin high */
              delayMicroSecond(10); //10 seconds delay */
              GPIOA->DATA &= ~(1<<4); // make trigger  pin low */

            while(1)
            {
   TIMER0->ICR = 4;           // clear timer0A capture flag */
   while((TIMER0->RIS & 4) == 0) ;    // wait till captured */
               if(GPIOB->DATA & (1<<6)) //check if rising edge occurs */
                           {
   lastEdge = TIMER0->TAR;     // save the timestamp */
                           // detect falling edge */
```

```
    TIMER0->ICR = 4;            // clear timer0A capture flag */
    while((TIMER0->RIS & 4) == 0) ;    // wait till captured */
    thisEdge = TIMER0->TAR;     // save the timestamp */
                                return (thisEdge - lastEdge); // return the time difference */
                                }
                }
}

// Timer0A initialization function */
// Initialize Timer0A in input-edge time mode with up-count mode */
void Timer0ACapture_init(void)
{
    SYSCTL->RCGCTIMER |= 1;     // enable clock to Timer Block 0 */
    SYSCTL->RCGCGPIO |= 2;      // enable clock to PORTB */

    GPIOB->DIR &= ~0x40;        // make PB6 an input pin */
    GPIOB->DEN |= 0x40;         // make PB6 as digital pin */
    GPIOB->AFSEL |= 0x40;       // use PB6 alternate function */
    GPIOB->PCTL &= ~0x0F000000;  // configure PB6 for T0CCP0 */
    GPIOB->PCTL |= 0x07000000;

                // PB2 as a digital output signal to provide trigger signal */
                SYSCTL->RCGCGPIO |= 1;      // enable clock to PORTA */
                GPIOA->DIR |=(1<<4);        // set PB2 as a digial output pin */
                GPIOA->DEN |=(1<<4);        // make PB2 as digital pin */

    TIMER0->CTL &= ~1;          // disable timer0A during setup */
    TIMER0->CFG = 4;            // 16-bit timer mode */
    TIMER0->TAMR = 0x17;        // up-count, edge-time, capture mode */
    TIMER0->CTL |=0x0C;         // capture the rising edge */
    TIMER0->CTL |= (1<<0);          // enable timer0A */
}



// Create one microsecond second delay using Timer block 1 and sub timer A */
void delayMicroSecond(int time)
{
    int i;
    SYSCTL->RCGCTIMER |= 2;     // enable clock to Timer Block 1 */
    TIMER1->CTL = 0;            // disable Timer before initialization */
    TIMER1->CFG = 0x04;         // 16-bit option */
    TIMER1->TAMR = 0x02;        // periodic mode and down-counter */
    TIMER1->TAILR = 16 - 1;  // TimerA interval load value reg */
    TIMER1->ICR = 0x1;          // clear the TimerA timeout flag */
    TIMER1->CTL |= 0x01;        // enable Timer A after initialization */

    for(i = 0; i < time; i++)
    {
        while ((TIMER1->RIS & 0x1) == 0) ;      // wait for TimerA timeout flag */
        TIMER1->ICR = 0x1;      // clear the TimerA timeout flag */
    }
}
void init_UART(void)
{
                SYSCTL->RCGCUART |= 0x20;  // Enable the UART module using the RCGCUART reg . (from datasheet 344)
    SYSCTL->RCGCGPIO |= 0x10;  // Enabling clock of GPIO peripheral (from datasheet 1351)
    // BRD      =  16,000,000 / (16 * 9600) = 104,c-> Baudrate value for 9600 bit unit rate
                                // UARTFBRD = (0,c*64+0.5)        = 11
    UART5->CTL = 0;         // UART5 module disbable
    UART5->IBRD = 104;      // for 9600 baud rate, integer = 104
    UART5->FBRD = 11;       // for 9600 baud rate, fractional = 11
    UART5->CC = 0;          //select system clock*/
    UART5->LCRH = 0x60;     // data lenght 8-bit, not parity bit, no FIFO
    UART5->CTL = 0x301;     // Enable UART5 module, Rx and Tx */

    // UART5 TX5 and RX5 use PE4 and PE5. Configure them digital and enable alternate function
    GPIOE->DEN = 0x30;      // set PE4 and PE5 as digital
    GPIOE->AFSEL = 0x30;    // Use PE4,PE5 alternate function
    GPIOE->AMSEL = 0;    // Turn off analg function*/
    GPIOE->PCTL = 0x00110000;     // configure PE4 and PE5 for UART
}

// @Def : ( printChar ) if transmitter bus is available ,  ...
//  it sends the data to UART Data register .
void printChar(unsigned char data)
{
    while((UART5->FR & (1<<5)) != 0); // wait until Tx buffer not full */
    UART5->DR = data;                  // before giving it another byte */
}
// @Def : ( printString ) if transmitter bus is available ,  ...
// it sends the data to UART Data register byte by byte .
void printingString(char *str)
{
  while(*str)
                {
                                printChar(*(str++));
                }
```

```
}
void Delay(unsigned long counter)
{
	unsigned long i = 0;

	for(i=0; i< counter*1000; i++);
}
```