

## Lab 6: Timers Application

Name: \_\_\_\_\_ ID: \_\_\_\_\_ Section: \_\_\_\_\_

### Objective

To introduce Cortex M4 timers programming for creating delays, counting events and measuring intervals

### In-Lab

**Task 1:** Setup a periodic interrupt using the SysTick timer

**Task 2:** Build a Traffic Light system using General Purpose Timers

**Task 3:** Using Pulse Width Modulation (PWM) to generate color spectrum using RGB LED

**Task 4:** HC-SR04 Ultrasonic Sensor Interfacing with TM4C123

## 1 Introduction to SysTick Timers and Interrupts

Timers are usually used for three kinds of tasks; creating delays, counting events and measuring the time between events. TM4C123GH6PM ARM Cortex M4 microcontroller provides a 24-bit system timer that supports down decrements feature. That means it counts downwards starting from a preloaded or set value. The rate of value decrements depends on the system clock frequency and we can set the value of clock frequency using a control register which is associated with the SysTick timer. Following are the some of the many applications of SysTick timer:

- Counting the number of people passing through airport security
- Measuring the time a sports car crosses the finish line
- Keeping the office lights on exactly 8 hours each day

### 1.1 SysTick Timer

Systick timer is a dedicated hardware-based timer which is built inside the ARM Cortex M4 CPU and can be used to generate an interrupt at a fixed interval of time. The systick timer will generate interrupts after a specified time and time settings can be done using the SysTick control register. There are two way the SysTick Timer is used as follow:

- **Polling Method:** Inside the loop, we will keep polling the status of the count bit of SysTick timer control register. The count bit becomes one whenever the system timer countdown to zero.
- **Interrupt Method:** Inside the loop, we will be doing nothing, instead we will write a logic (To perform our task) inside the SysTick handler function SysTick\_Handler(), which will execute after specific time interval as shown in Fig. 1.

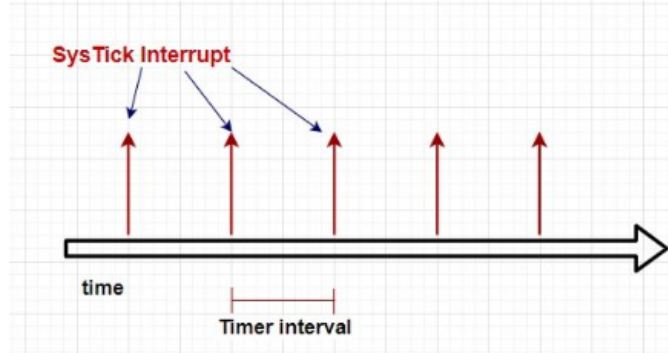


Figure 1: SysTick Interrupts

## 1.2 Functionality of the SysTick Timer

TM4C123GH6PM provides a 24 bit timer SysTick. Therefore, the maximum value that can be loaded to the load register of system timer is  $2^{24}-1$ . Hence, it starts decrement value by one from its initial set value. We reload the initial value to reload register and counter decrements from reload value to zero. The value of the counter decrements on every positive edge of the clock cycle. When counter values reach zero, the system timer generates an interrupt. Also, the counter is re-initialized with reload value again. Hence, the process keeps repeating as summarized in Fig. 2.

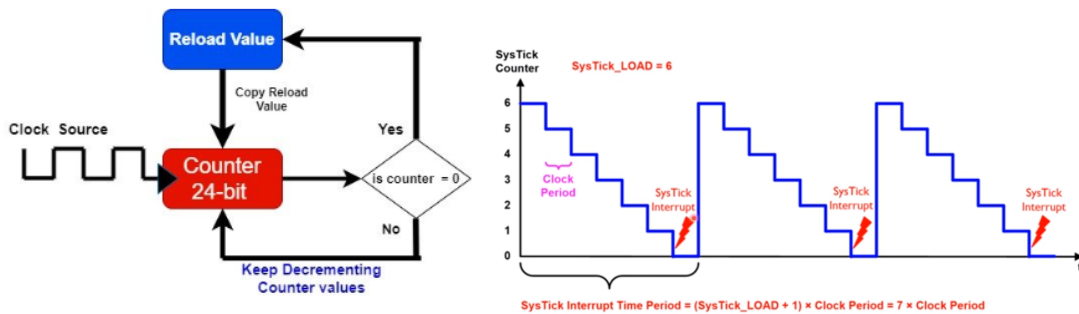


Figure 2: SysTick Timer Interrupt

### 1.3 SysTick Timer Registers

#### STCTRL - SysTick Control and Status

It is a 32-bit register but only 4 bits are used and rest of the bits are reserved as shown in the figure below with explanation given next:

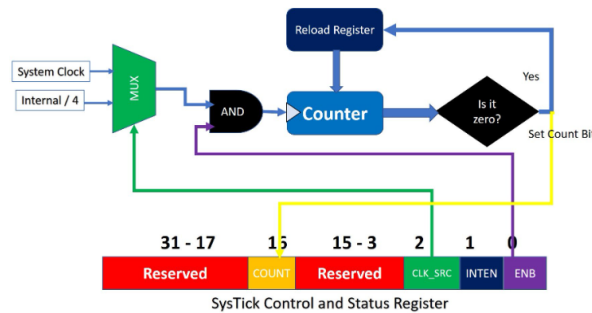


Figure 3: SysTick Control and Status Register

1. ENABLE Bit: This bit enables and disables the counter.  
**1: Enables the counter and 0 : Disables the counter**
2. INTEN Bit: In this mode, CPU reads the respective interrupt signal and executes its respective handler routine SysTick\_Handler(). Whenever the counter reaches zero from a reload value, the SysTick timer requests the interrupt signal of TM4C123 microcontroller.  
**1: Enables the interrupt and 0: Disables the interrupt**
3. CLK SRC Bit: It is a clock source selection bit.  
**1: Selects the system clock and 0 : Selects the external clock**
4. COUNT Bit: This is a counter flag or status bit for the counter. This bit becomes 1 when counter decrements from 0 to one.

#### STRELOAD - SysTick Reload Value Register

This register holds the reload value that will be copied to counter on every systick interrupt. In other words, the counter is re-initialized with a reload register value on every count down to zero. STRELOAD register is a 32-bit register but only the first 24-bits are used and the rest of the bits are reserved As shown in the figure below. As systick timer has a 24-bit counter.

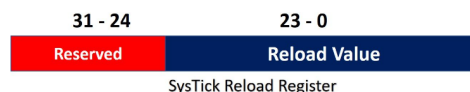


Figure 4: SysTick Reload Value Register

## STCURRENT - SysTick Current Value Register

As its name suggests, this register holds the current value of the counter. For instance, you want to perform some action after clock pulses, you can read the value of this register and perform an action. If you this register value, it will return the current value of the counter.



Figure 5: SysTick Current Value Register

## Calculating Delay Using Timers

We can use the formula to calculate any required delay:

$$\text{Reload} = ( \text{Interrupt Delay} / \text{Clock Period} ) - 1$$

Note that TM4C123G has a 16MHz of clock bus. it should also be noted that the value of reload register should not be greater than the maximum value a 24-bit register can hold which is  $2^{24}-1$ .

## 2 General-Purpose Timer Module (GPTM) - TM4C123G

In TI TivaC micro controllers, the timers are called General-Purpose Timer Module (GPTM). The General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. In other words, there are 12 Timer Blocks in the TI Tiva TM4C123G, 6 of them are 16/32-bit timers and the other 6 are 32/64-bit. Each 16/32-bit GPTM block can provide two 16-bit (half-width) timers/counters that are referred to as Timer A and Timer B. These timers/counters can be further configured to operate independently as timers or event counters, or concatenated together to operate as one 32-bit (full-width) timer or one 32-bit Real-Time Clock (RTC).

Similarly, each 32/64-bit Wide GPTM block provides two 32-bit timers, also called Timer A and Timer B, and they can be concatenated together to form a 64-bit timer. All timers have interrupt controls and separate interrupt vectors as well as separate interrupt handlers. GPTM can be utilized to for External Even Measurement, PWM Generation, Frequency Measurement, Motor Control etc. In this lab, we would be using these timers to generate PWM to control RGB LED. Fig. 6 summarizes the available capacities for which GPTM can be employed.

Refer to Page 704 of data sheet to read more about GPTM registers and usage. Refer to Page 716 of data sheet to study how GPTM can be utilized as PWM Generators.

Mode	Timer Use	Count Direction	Counter Size		Prescaler Size <sup>a</sup>		Prescaler Behavior (Count Direction)
			16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
One-shot	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
Periodic	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
RTC	Concatenated	Up	32-bit	64-bit	-	-	N/A
Edge Count	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
Edge Time	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
PWM	Individual	Down	16-bit	32-bit	8-bit	16-bit	Timer Extension

a. The prescaler is only available when the timers are used individually

Figure 6: GPTM Module Capabilities

## Setting up Values for Registers

In order to specify a particular value for register, we use the following bit-wise operations primarily to set, toggle reset the bits of particular register. You will be using these in your code to modify registers value.

Set nth bit of a register:  $|=(1<<n)$

Reset nth bit of a register:  $\&= \sim(1<<n)$

Toggle nth bit of a register:  $\wedge=(1<<n)$

## In-Lab Tasks

### Task 1: Setup a periodic interrupt using the SysTick timer

Create a new Keil Project and set it up for the Tiva C board as performed in previous labs, do not forget to check CORE and Startup in "Manage Run-Time Environment". Also select "Stellaris ICDI" in debug from "Options from Target".

Follow the steps next to setup/initialize SysTick timer to create interrupt to blink tivaC on-board LED on Port F with delay of 1 second at each interrupt call.

Given that the clock frequency of the Tiva C board is 16MHz, determine the clock cycles value needed to generate delay for one second:

--

1. Turn on the bus clock for GPIOF by setting the 5th bit of the RCGCGPIO register (Refer to page 340 of the datasheet).
2. Enable the GPIO pins PF1, PF2, and PF3 as digital output pins by setting the 1st, 2nd, and 3rd bit of the GPIO DEN and GPIO DIR registers (refer to page 682 and page 663 of the datasheet for this).
3. Load the value of clock cycles calculated above in the SysTick Reload register (refer to page 140 of the datasheet for this).
4. Enable the SysTick timer by setting the 0th bit of the SysTick Control and Status Register. Set the 1st bit of the same register to enable the SysTick interrupt. And finally, set the 2nd bit of the register to select system clock as the source of the systick timer (refer to page 138 of the datasheet for this).
5. Clear the Current Value register of the SysTick timer to initialize the timer with 0 value (refer to page 141 of the datasheet).
6. Populate the "void SysTick\_Handler(void)" function with the logic to toggle the three LEDs on interrupt call. Use the appropriate command from "Setting up Values for Registers" above.

## Lab 6: Timers Application

---

*Provide code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. **Make sure the code is readable.***

## Task2: Build a Traffic Light system using General Purpose Timers

In this task we will use General Purpose Timer 0 to write the delay function for a specified time in milliseconds. This delay function will then be used to generate delay between RED, GREEN and BLUE color of tivaC on-board LED. This time, instead of using interrupt of 1 second, we are using timer to perform pretty much the same tasks and make a Traffic Light System.

- a Given that the clock frequency of the Tiva C board is 16MHz, determine the clock cycles needed to generate delay for one millisecond:  
(i)\_\_\_\_\_
- b Calculate the clock cycles required to generate a delay of 5 seconds:  
(ii)\_\_\_\_\_
- c How many bits are required to hold the value calculated in (b)? Should we use the 16-bit mode or the 32-bit mode of the clock?  
(iii)\_\_\_\_\_

Follow the steps below to configure the Timer 0 and create desired delay for Traffic Signal:

1. Define a function timer0\_delay(int ms) for timer 0 delay which accepts a parameter "int ms" in milliseconds.
2. Turn on the bus clock for Timer 0 by setting the 0th bit of the RCGCTIMER register (Refer to page 338 of the datasheet for this).
3. Disable the Timer before initializing it by resetting the 0th bit of the GPTM Control register (CTL) (refer to page 737 of the datasheet for this).
4. Set the GPTM Configuration register (CFG) according to the mode determined in (c) above as per Fig. 7 (refer to page 727 of the datasheet for this).
5. Configure Timer A in one-shot mode by setting the last 2 bits of the GPTM A mode (TAMR) register to 0x1 (refer to page 729 of the datasheet for this).
6. Load the counter bound in the GPTM Interval Load register (TAILR) for timer A using the value calculated in (a), (refer to page 756 of the datasheet for this). The value for this register can be calculated using formula below:  
**(ClockCycles(Part (a))/1000) \* ms(The input argument) - 1**  
**E.g. TIMER0-> TAILR = (16000000/1000)\*ms - 1;**
7. Clear the status flag of the timer by setting 0th bit of GPTM ICR (refer to page 754 of the datasheet).
8. Enable the Timer after initializing it by setting the 0th bit of the GPTM Control register (CTL) (refer to page 737 of the datasheet).



Bit/Field	Name	Type	Reset	Description												
2:0	GPTMCFG	RW	0x0	<p>GPTM Configuration</p> <p>The GPTMCFG values are defined as follows:</p> <table><tr><th>Value</th><th>Description</th></tr><tr><td>0x0</td><td><p>For a 16/32-bit timer, this value selects the 32-bit timer configuration.</p><p>For a 32/64-bit wide timer, this value selects the 64-bit timer configuration.</p></td></tr><tr><td>0x1</td><td><p>For a 16/32-bit timer, this value selects the 32-bit real-time clock (RTC) counter configuration.</p><p>For a 32/64-bit wide timer, this value selects the 64-bit real-time clock (RTC) counter configuration.</p></td></tr><tr><td>0x2-0x3</td><td>Reserved</td></tr><tr><td>0x4</td><td><p>For a 16/32-bit timer, this value selects the 16-bit timer configuration.</p><p>For a 32/64-bit wide timer, this value selects the 32-bit timer configuration.</p><p>The function is controlled by bits 1:0 of GPTMTAMR and GPTMTBMR.</p></td></tr><tr><td>0x5-0x7</td><td>Reserved</td></tr></table>	Value	Description	0x0	<p>For a 16/32-bit timer, this value selects the 32-bit timer configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 64-bit timer configuration.</p>	0x1	<p>For a 16/32-bit timer, this value selects the 32-bit real-time clock (RTC) counter configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 64-bit real-time clock (RTC) counter configuration.</p>	0x2-0x3	Reserved	0x4	<p>For a 16/32-bit timer, this value selects the 16-bit timer configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 32-bit timer configuration.</p> <p>The function is controlled by bits 1:0 of GPTMTAMR and GPTMTBMR.</p>	0x5-0x7	Reserved
Value	Description															
0x0	<p>For a 16/32-bit timer, this value selects the 32-bit timer configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 64-bit timer configuration.</p>															
0x1	<p>For a 16/32-bit timer, this value selects the 32-bit real-time clock (RTC) counter configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 64-bit real-time clock (RTC) counter configuration.</p>															
0x2-0x3	Reserved															
0x4	<p>For a 16/32-bit timer, this value selects the 16-bit timer configuration.</p> <p>For a 32/64-bit wide timer, this value selects the 32-bit timer configuration.</p> <p>The function is controlled by bits 1:0 of GPTMTAMR and GPTMTBMR.</p>															
0x5-0x7	Reserved															

Figure 7: Bit Operations for GPTM

- Wait for the GPTM Raw Interrupt Status flag to set (refer to page 748 of the datasheet).

Once we have the "timer0\_delay(int ms)" function ready, we can use that delay function to toggle between RED, GREEN and YELLOW LEDs for Traffic Control System. Write a Logic to provide delay of 5 seconds between red and blue light, and then delay of 2 seconds between blue and green light with help of delay function written above. Use the on-board TivaC LED on PORTF to make this system. You have to set data on Pin 1,2 and 3 of PORTF for red, blue and green color using following command: e.g. for RED color LED, *GPIOF* → *DATA* = 0x2

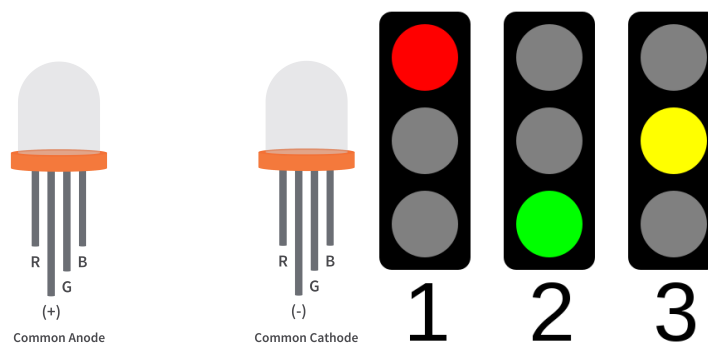


Figure 8: Sequence for RGB LED

## Lab 6: Timers Application

---

*Provide code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. **Make sure the code is readable.***

### Task 3: Using Pulse Width Modulation (PWM) to generate color spectrum using RGB LED

In this task, we will control the brightness of each channel in the external RGB LED to output differing color spectrum. Connect your Pin 1,2 and 4 of RGB LED with PB0, PB2 and PB4, connect Pin 3 (+ve) to 3.3V with resistance of 470 ohms as shown in Figure Below.

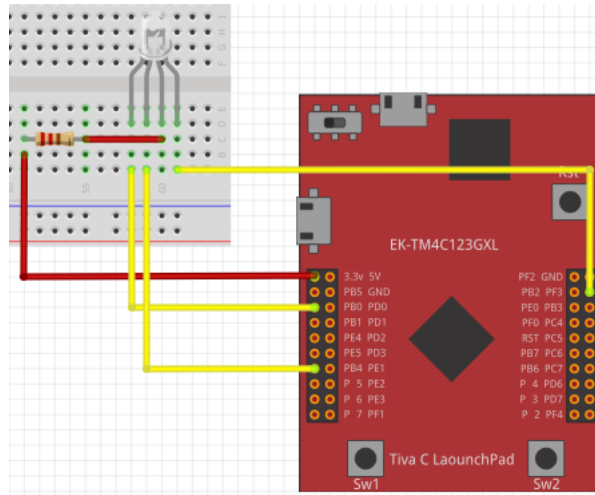


Figure 9: Pin Connections between RGB LED and TivaC

- a Write the last 3 digits of your HU ID (0xABC): (i)\_\_\_\_\_
- b You will use the following duty cycles for the RGB channel of LED from Fig. 8:  
 RED : A0% (i)\_\_\_\_\_  
 GREEN : B0% (ii)\_\_\_\_\_  
 BLUE : C0% (iii)\_\_\_\_\_

**NOTE :** For ID 05491, duty cycles for RGB should be 40 %, 90%, and 10% respectively.

We will control the brightness of each of the RGB led connected at Pin 1,2 and 4 using PWM, which we will generate using timers. Follow the steps below to configure the timer 1A in PWM mode and configure the associated GPIO pin to output the PWM signal from Pin1.

1. Turn on the bus clock for Timer 0-3 by setting the relevant bits of the RCGCTIMER register. (Refer to page 338 of the datasheet)
2. Enable clock for Port B
3. Enable the GPIO pin assigned to the CCP pin timer by setting the relevant bit of the GPIO DEN and GPIO DIR registers (refer to page 682 and page 663 of the datasheet for this).
4. Enable the alternate function for the same pin (refer to page 671 of the datasheet for this).

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type <sup>3</sup>	Description
T1CCP0	30 58	PF2 (7) PB4 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 0.
T1CCP1	31 57	PF3 (7) PB5 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 1.
T2CCP0	5 45	PF4 (7) PB0 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 0.
T2CCP1	46	PB1 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 1.
T3CCP0	47	PB2 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 0.
T3CCP1	48	PB3 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 1.

Timer	Up/Down Counter	Even CCP Pin	Odd CCP Pin
16/32-Bit Timer 0	Timer A	T0CCP0	-
	Timer B	-	T0CCP1
16/32-Bit Timer 1	Timer A	T1CCP0	-
	Timer B	-	T1CCP1
16/32-Bit Timer 2	Timer A	T2CCP0	-
	Timer B	-	T2CCP1
16/32-Bit Timer 3	Timer A	T3CCP0	-
	Timer B	-	T3CCP1

Figure 10: Pin Configuration for Timers

- Configure the pin as T1CCP0 pin by clearing the associated block of the GPIO Port Control register and then setting the same block 0x7 to enable mux of the CCP pin (refer to page 688 of the datasheet for this).
- Define a function of "timer1a\_pwm(int duty\_cycle)" which accepts duty cycle as percentage between 0 and 100.
- Inside Function, disable the Timer before initializing it by resetting the 0th bit of the GPTM Control register (CTL) (refer to page 737 of the datasheet for this).
- Inside Function, set the GPTM Configuration register (CFG) according to the 16-Bit configuration (refer to page 727 of the datasheet for this).
- Inside Function, configure the timer in PWM mode by setting the 3rd bit of the Timer A mode register (refer to page 729 of the datasheet for this).
- Inside Function, configure the timer in Edge Count Mode by resetting the 2nd bit of the Timer A mode register (refer to page 729 of the datasheet for this).
- Inside Function, configure the timer in Periodic mode by setting the last 2 bits of the Timer A mode register as 0x0 (refer to page 729 of the datasheet for this).
- Inside Function, load the counter bound in the GPTM Interval Load register (TAILR) for timer A such that the period of the timer is 1ms (refer to page 756 of the datasheet for this).

13. Inside Function, load the value of the PWM cut-off in the GPTM Match register for timer A using the duty cycle passed in the PWM timer function (refer to page 758 of the datasheet).
14. Inside Function, enable the Timer after initializing it by setting the 0th bit of the GPTM Control register (CTL) (refer to page 737 of the datasheet for this).

Once the timer 1A is initialized to generate PWM, perform the following tasks:

- Repeat all of the above steps to configure timer 2A and timer 3A for PWM on pin 2 and pin 4 accordingly.
- Input the three duty cycles calculated above in (b) to input your "timerXa\_ pwm(int duty\_ cycle)" function to create color spectrum.

## Lab 6: Timers Application

---

*Provide code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. **Make sure the code is readable.***

## Task4: HC-SR04 Ultrasonic Sensor Interfacing with TM4C123

An important application of Timer Interrupts is sampling data from sensors. Timer interrupts can be used to sample data from sensors at a regular, periodic rate. This ensures that the data is sampled evenly and consistently. It also allows the program to perform other tasks while the sensor data is being sampled. In this task, we will be measuring the distance using HC-SR04 Ultrasonic range sensor. This task will be similar to what we have already done in Lab-04, i.e. the logic used to measure distance will be the same. However, this time we will be using timers to measure the pulse duration of the Echo Output.

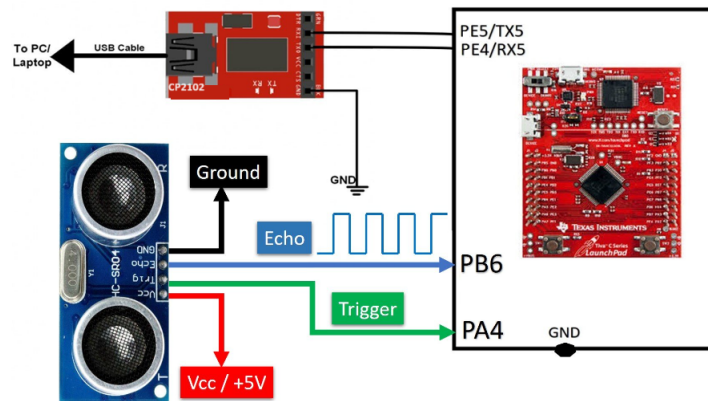


Figure 11: Schematic for Task 4

### 2.1 HC-SR04 Ultrasonic Sensor

First, let's begin with a fundamental overview of the HC-SR04 ultrasonic sensor. This sensor is designed for non-contact distance measurement and can accurately measure distances within a range of 2cm to 400cm. It comprises three primary components: an ultrasonic transmitter, an ultrasonic receiver, and a control circuit. The figure given below shows the pinout diagram of the HC-SR04 ultrasonic sensor. It consists of 4 pins.

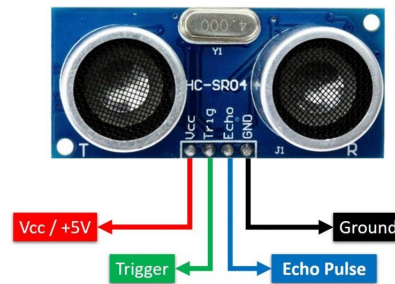


Figure 12: HC-SR04 Pinout Diagram

Two of them are power supply pins such as +5V and ground. The other two pins are Trigger

and echo pin. The echo output pulse pin is used to get output from the sensor. Trigger pin is used to initiate the sensor to start ranging. For more information on how this sensor works you can view your *Lab04*

## 2.2 Instruction

We use Timer block 0 and sub-timer A in this tutorial. This void Timer0ACapture\_init(void) function configures the Timer0A in input-edge capture mode by selecting input-edge time mode. PB6 pin of PORTB is used as an external event capture pin. For a more detailed understanding of input\_output capture mode please refer to page 724 of the datasheet.

The skeleton code for task 4 titled *Task4.c* is given. In that code, you have to implement Timer0ACapture\_init(void). Carefully read the commented instructions given in the code.

## 2.3 Keil Uvision V5 Instructions

The skeleton code provided results in a different system clock rate than the default 16 MHz frequency. But the code works on the default 16 MHz system clock. Therefore, it leads to timing issues with Keil v5. You may retain the default system clock rate in Keil v5, by the following steps:

1. Expand the Project→Device to show system\_TM4C123.c (startup)
2. Double click to open the file in the editor window
3. Find the line `#define CLOCK_SETUP 1` as the figure below
4. Comment out the line
5. Rebuild the project

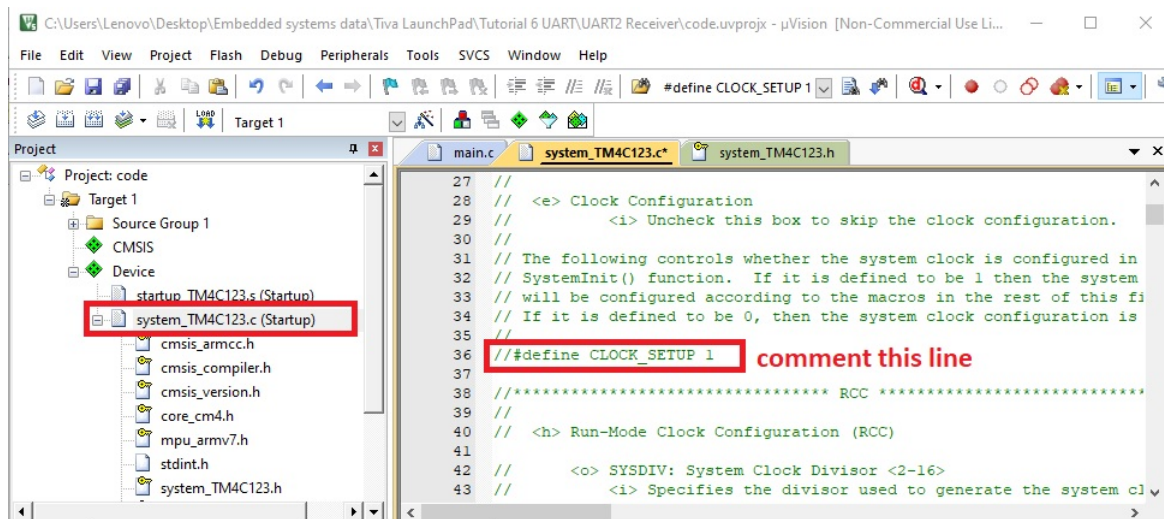
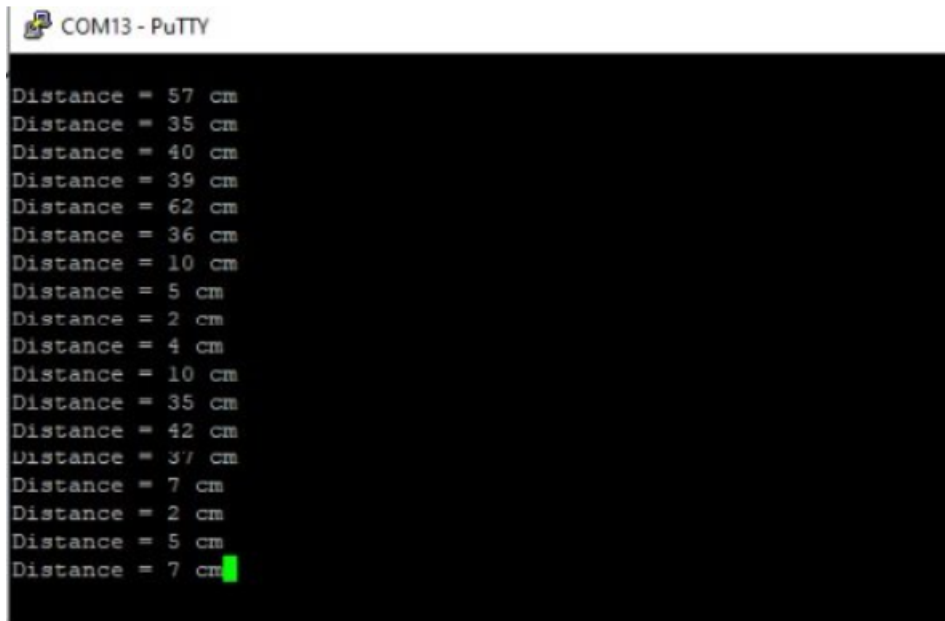


Figure 13: Instruction for Keil Timing Setup



### 2.3.1 Output

In this lab, we will be using the Universal Asynchronous Receiver-Transmitter (UART) protocol. For the purpose of this Lab, you are not required to know how UART works. However, in order to display the distance on your computer, you will need PuTTY. PuTTY is a free and open-source terminal emulator, serial console, and network file transfer application.



```
COM13 - PuTTY
Distance = 57 cm
Distance = 35 cm
Distance = 40 cm
Distance = 39 cm
Distance = 62 cm
Distance = 36 cm
Distance = 10 cm
Distance = 5 cm
Distance = 2 cm
Distance = 4 cm
Distance = 10 cm
Distance = 35 cm
Distance = 42 cm
Distance = 37 cm
Distance = 7 cm
Distance = 2 cm
Distance = 5 cm
Distance = 7 cm
```

Figure 14: Output

Moreover, for UART communication between the microcontroller and your computer, you will need the CP2102 UART to USB module. For this, you are required to install CP2102 drivers on your computer. The drivers are available on LMS.



Figure 15: cp2102 UART to USB

When selecting a serial line, be sure to select the port given in device manager for CP2102 and not for the microcontroller.

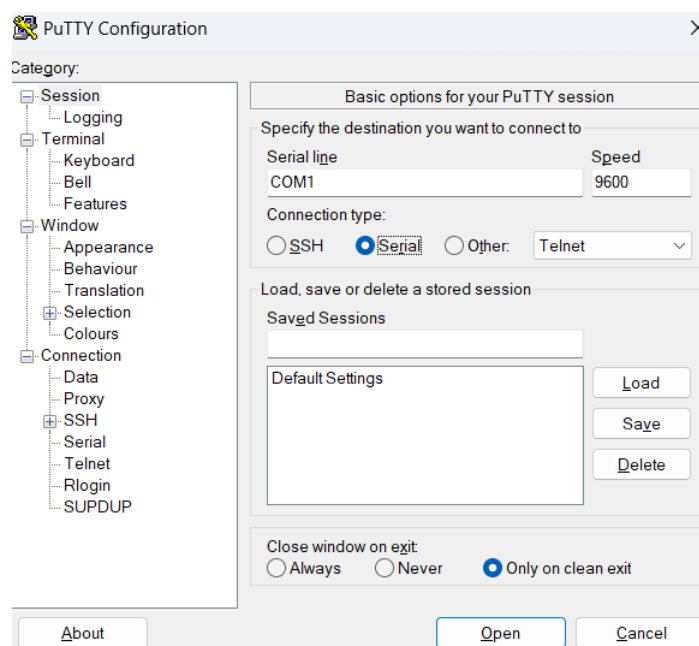


Figure 16: Putty Configuration

## Lab 6: Timers Application

---

*Provide code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. **Make sure the code is readable.***

### 3 Assessment Rubrics

#### Marks distribution

		LR2	LR4	LR5	LR9
In-lab	Task 1	10 points	-	10 points	10 points
	Task 2	10 points	10 points	10 points	
	Task 3	10 points	10 points	10 points	
	Task 4	5 points	-	5 points	
Total Marks 100					

#### Marks obtained

		LR2	LR4	LR5	LR9
In-lab	Task 1		-		
	Task 2				
	Task 3				
	Task 4		-		
Obt. Marks out of 100					