CECS 347 Spring 2025

An Ultrasonic Ranger Finder

By

Eric Santana, Mason Doan, Hector Polanco, Kero Samaan

September 23, 2025

Introduction

This project measures distance using an HC-SR04 ultrasonic sensor and a TM4C123 LAUNCHPAD. The code sends a 10 us trigger pulse and it times the echo with Timer1, converts time to centimeters and then prints the result over UART0 and it also shows the range with the on board LEDs (red/green/blue). The design uses a 16Mhz system clock (PLL), GPIO interrupts on PB4 for echo, SYSTICK for pacing, and modules for GPIO, UART, timers and clock.
Every cycle, the board sends a short pulse to the HC-SR04, times the returning echo, and then it computes distance in centimeters and then prints in the serial terminal and then it lights the LEDs by range: <30 cm red blink, 30-69 cm green, 70-99 cm blue, >= 100cm off.

Operation

The first step in using the TM4C123–HC-SR04 Ultrasonic Range Finder program is to connect the TM4C123 to your computer and use a terminal application to read data over UART. The board transmits data at a baud rate of 57,600. Once that is done, you may place an object in front of the sensor and press Switch 1 on the board. The board uses the following parameters to decide which LED to turn on:

- Blink RED LED: distance < 10 cm
- GREEN LED: 10 cm < distance < 70 cm
- BLUE LED: 70 cm < distance < 100 cm
- LED OFF: distance >100 cm

After this, the distance data is transmitted over UART and displayed on the terminal. This process repeats every time Switch 1 is pressed.

In order to accomplish this, the main program first initializes all the required drivers. The PLL_Init function sets up the clock speed of the board to 16 MHz. The PortF_Init function configures Switch 1 and its corresponding interrupt behavior. The Timer1_Init function configures General Purpose Timer1A to generate a 10 µs pulse for the trigger pin and a 38 ms timing window to calculate

the echo width for the final distance calculation. The PortB_Init function configures the trigger pin and also configures the echo pin with its corresponding interrupt functionality. The Uart_Init function sets up UART0 with the correct baud rate and data width, and provides read and transmit functions to send data over UART0. This is what allows the data to be displayed on your PC's terminal. Finally, the SysTick_Init function is configured to use an interrupt every 0.5 s in order to blink the RED LED.

In the main program, the global variables used are distance, done, and blinkFlag. The distance variable stores the final result of the distance calculation in cm. The done variable stores either 1 or 0 depending on whether the distance calculation is complete. The blinkFlag is set to 1 or 0 depending on the value of distance.

In the main loop, the done variable is first initialized to 0, then the Trigger function is called to generate a 10 μs pulse to the HC-SR04 ultrasonic sensor. The program then waits until the distance calculation is complete and done is set to 1. At that point, the LEDs are updated, the distance is sent over UART, and the process is ready to repeat.

This program uses a total of three interrupts. The PortB_Handler takes care of the echo signal and starts and stops Timer1A; once the timer is turned off, it computes the distance value in cm and updates the global variable distance. The PortF_Handler is responsible for detecting Switch 1 button presses, sending the distance data over UART, and updating which LED should be on and whether the RED LED should blink by updating the blinkFlag based on the distance value. Lastly, the SysTick_Handler checks if the blinkFlag is set to true and blinks the RED LED.

With all of these functions working together, the TM4C123 and HC-SR04 sensor provide accurate distance data to the user's PC terminal via UART communication.

Theory

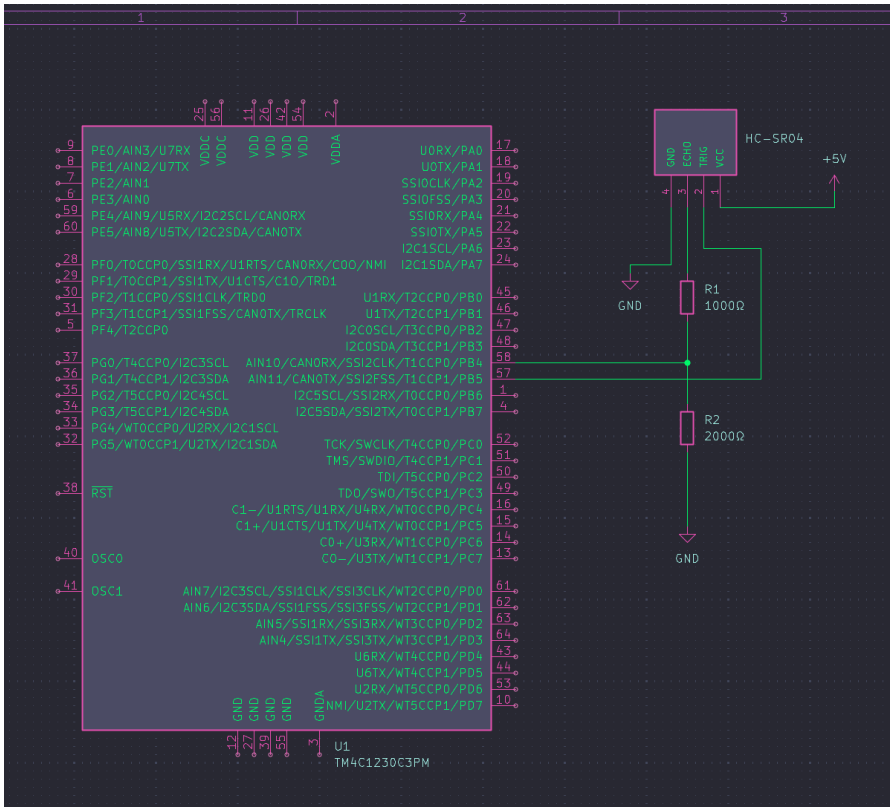
The HC-SR04 ultrasonic sensor measures distance by using sound wave propagation and reflection. When the trigger pin is driven high for 10 µs, the sensor emits an ultrasonic burst at 40 kHz. These sound waves travel through the air, bounce off the nearest object, and return to the sensor.

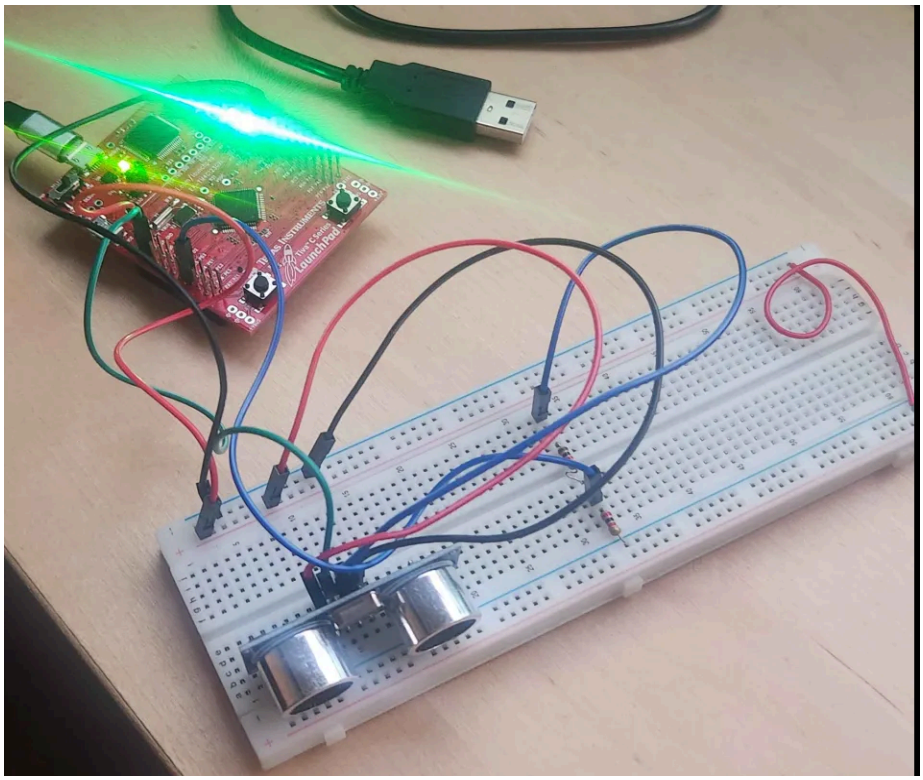
The sensor outputs a high signal on the echo pin for the duration of the round-trip travel time. Since the speed of sound in air is about 0.0343 cm/µs, the distance can be calculated with the formula:

$$distance\ (cm)\ = \frac{Echo\ Pulse\ Width\ (\mu s)\ x\ 0.0343}{2}$$

The division by two accounts for the fact that the sound must travel to the object and then back. This allows the HC-SR04 to determine distance by converting sound travel time into a measurable value.

Hardware


HC-SR04
GND
ECHO
TRIG
VCC
+5V
R1
1000Ω
R2
2000Ω
GND
GND
U1
TM4C1230C3PM



Software

The major software modules in our program are the Trigger module, the GPIO interrupt handlers for port F and port B, and the UART modules.

The trigger module starts by setting PB5, the trigger pin low for 2us. It then sets the trigger pin high for 10us then sets it low again. In order to trigger the Ultra Sonic Sensor to send 8 40khz it requires the trigger pin to be high for ~10us. We set the pin high and use the delay function for 10us before setting it low to create our trigger signal. The delay function used GPIO Timer1A in periodic countdown mode with the function parameter ticks to generate delays. The ticks parameter is assigned to the interval load GPIO register which will count down from that tick value and then exit. We calculated our ticks values by multiplying our desired delay time in seconds by 16 million to represent our 16Mhz clock since our clock has 16 million ticks in one second, .000010 seconds takes 160 ticks but we load 159 into the register after subtracting 1 tick. Once a trigger is sent to the sensor the next step is to capture the distance between the sensor and whatever the sound waves from the sensor bounces off of. The echo pin receives the sound waves and creates a high signal on PB4. The longer the signal is high the further away an object is so we can use this signal to calculate the distance but we first have to measure how long that signal is high before we can do any calculations.

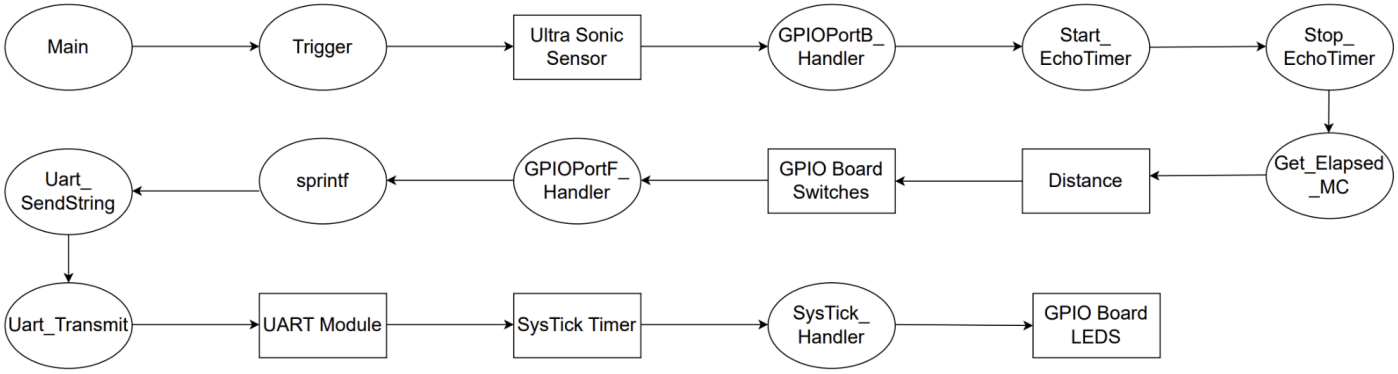
The GPIOPortB_handler function does this as when PB4 goes high an edge triggered interrupt occurs entering our handler function. From here we start the timer1 module with new values in certain registers to make capturing our signal length possible. We change the timer to count down 38ms since that is when the sensor will go low if there is no object detected by changing the prescale to 12 and the interval load register. For 38ms you take the amount of ticks for a 38ms delay which is 16million * .038, and divide by the prescale to get the load value 50,666. With this we can set the timer to count down and stop counting with the Stop_EchoTimer function when the echo signal goes low. Once the signal goes low we use the Get_Elapsed_MC function to measure how many ticks were between when the echo signal went high and when it went low by subtracting the value in

the timer count register from our reload value. Distance is calculated by first converting the measured ticks into measurable time then multiplying that by 0.0343 which represents the speed of sound and then dividing that by 2. After that we set the done global variable to 1 and clear the RIS register for PB4 so that we can wait for another edge triggered interrupt to measure distance.

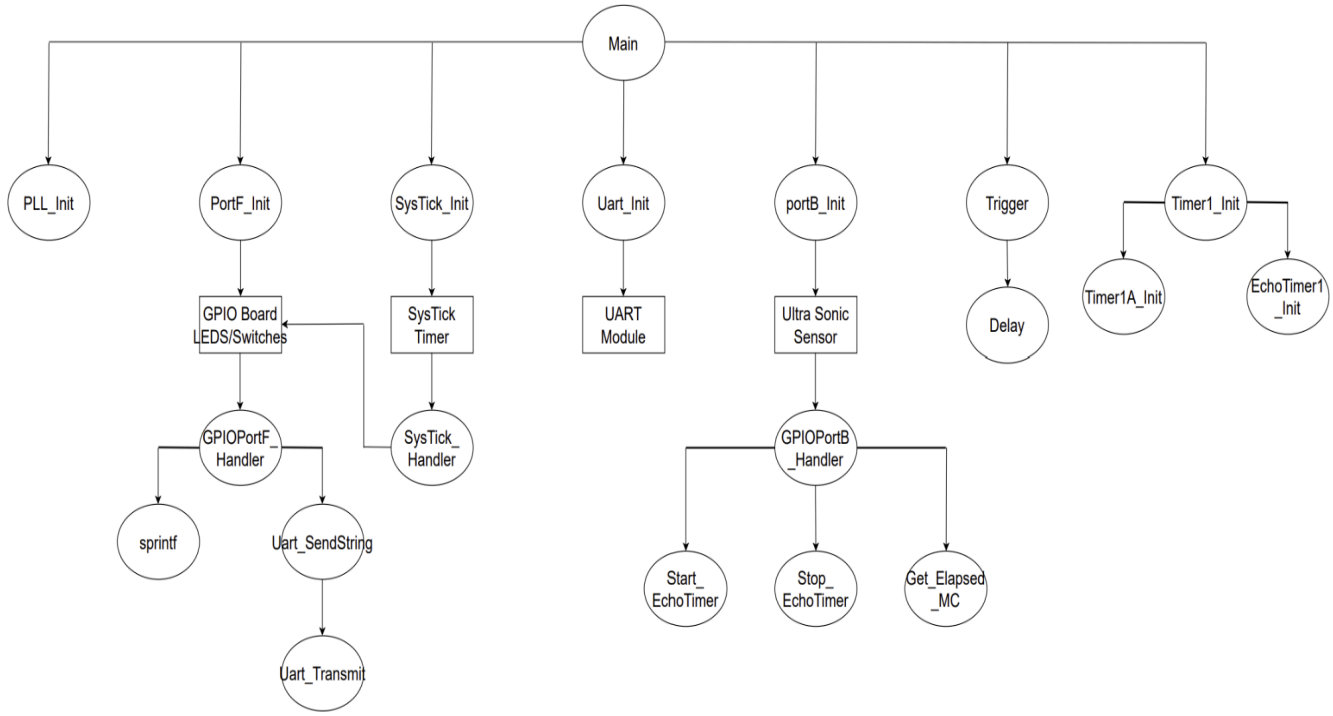
When switch 1 is pressed we enter the GPIOPortF handler which is a rising edge triggered interrupt handler routine. When the ISR is entered it will start by checking if the done variable is high so that it has a distance to work with. If the distance is less than 100 it will print a message with the distance and call Uart_SendString which takes a parameter for the MSG string. The UART_SendString function will then transmit that string while there are still characters left using the Uart_Transmit function. This function passes characters to the data register so that it could transmit the message. New bits of data are added to the transmit FIFO while it isn't full and while it is full the function waits until there is room again to add more data. When the distance is greater than 100 a message is sent using the aforementioned UART module stating that that object is out of range. In both cases once the messages are sent the RIS register for PF4 is cleared so that when the button is pressed again we can enter the ISR.

The Systick_handler is the last major module and it is in charge of illuminating the onboard LEDS in accordance with a distance range. When the distance is less than 10cm then the Red LED will blink at 2hz. We configure this by first finding how long our signal will be high which is found by dividing 1 by our 2hz to get 0.5s. From there we multiply .5 by 16 million and subtract 1 to get 3999999. We use else if statements to get a green led for less than 70cm and a blue led for greater than 100cm and no leds for when there is no distance to be found.

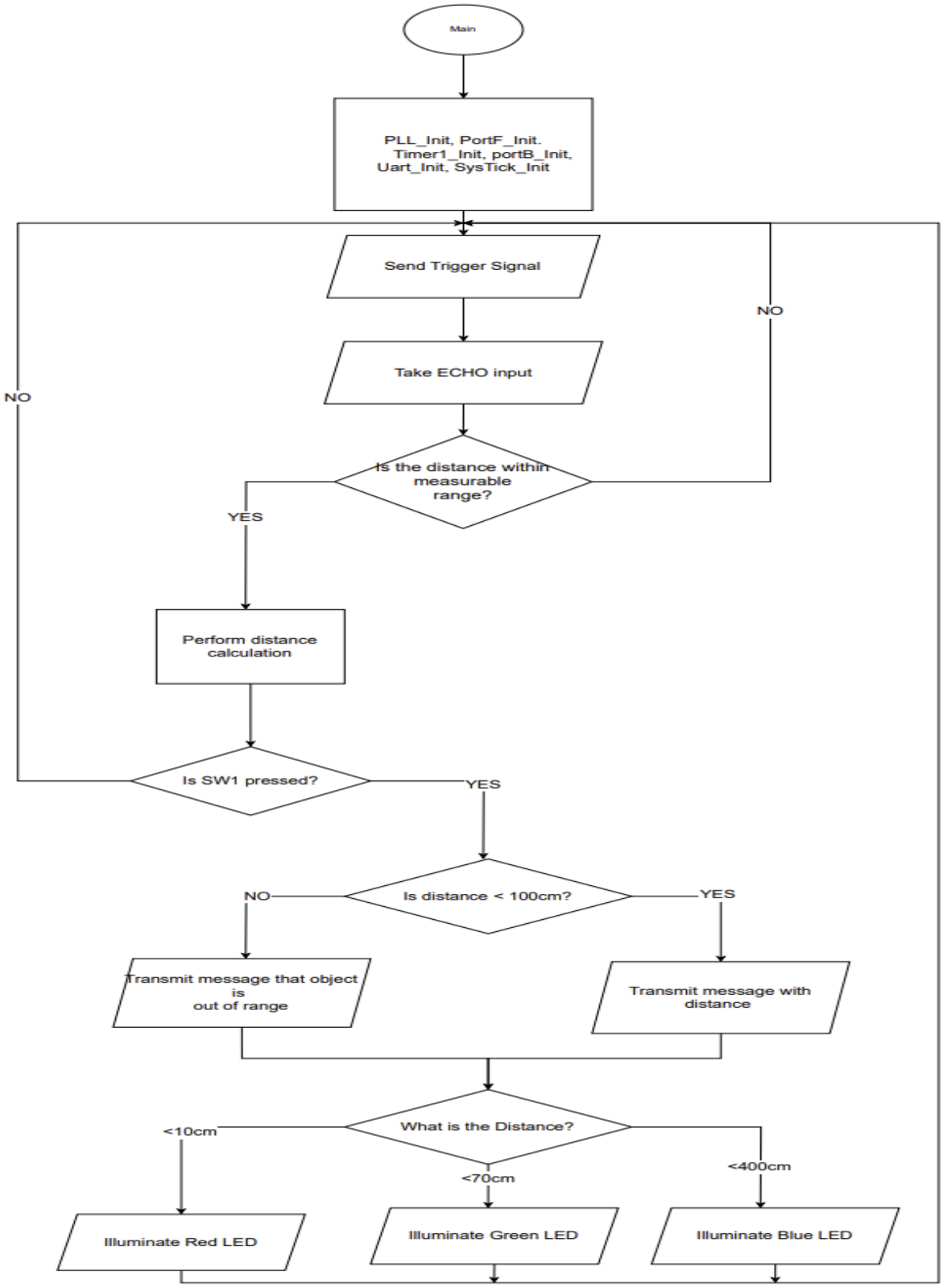
DATAFLOW CHART:



CALL GRAPH:



FLOWCHART:


Main
PLL_Init, PortF_Init.
Timer1_Init, portB_Init,
Uart_Init, SysTick_Init
Send Trigger Signal
Take ECHO input
Is the distance within measurable range?
NO
YES
NO
Perform distance calculation
Is SW1 pressed?
NO
YES
Is distance < 100cm?
NO
YES
Transmit message that object is out of range
Transmit message with distance
What is the Distance?
<10cm
<70cm
<400cm
Illuminate Red LED
Illuminate Green LED
Illuminate Blue LED

Conclusion

First, our timing was off. The trigger pulse, the echo window, and the overall pacing did not line up. After we carefully read the professor's comments on Part 3 and applied them in Part 4, we fixed up the timing. Timer1measurements became consistent, PB4 edge handling matched the echo edges, and the SysTick pacing felt correct during repeated tests. Next, UART slowed us down at the beginning since it was our first time using it in a project. We learned how to use UART and got it working. After that, we could see the readings on the computer without issues, and it was easy to follow what the program was doing. It also made testing the rest of the system simpler and faster. Finally, we adjusted the LED logic. Instead of multiple colors blinking, only the red LED blinks under 30 cm. Green for 30 to 69 cm and blue for 70 to 99 cm stay solid. Everything turns off at 100 cm or more. This made the distance ranges easy to read at a glance and matched what we were seeing in the serial output.