How to Develop a Range Project with the HC-SR04 Ultrasonic Distance Sensor.

ref1: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf - Data sheet and operations summary.
ref2: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/ - Arduino example with explanations.
ref3: https://www.davidpilling.com/wiki/index.php/HCSR04 - Analysis and testing examples. Information links.
ref4: https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/DM320115 - ATmega4809 Curiosity Nano Evaluation Kit.

This tutorial will suggest steps to take if you want to use the HC-SR04 sensor module in a project. The methods used could also be applied to most projects that use a time based sensor.
The target for this tutorial will be the ATmega4808/4809 Curiosity development board. Atmel Studio 7 will be the development environment. The code is written in 'C' and should easily port to other processors.
The tutorial assumes that the reader can create a project in Atmel Studio 7 and configure it to interface to the Curiosity board[4] or whatever target hardware they are using.

To start, we need the relevant technical information about the HC-SR04.[1]

Power:	Voltage: +5v DC.	Current: 15ma	
Timing:	Trig pulse: 10us	Max Echo time: 38 ms	Timing sequence: see [1]

From [1], the timing sequence shows that the Echo pulse does not go HIGH until after the transmission burst. Distance is determined by the width of the Echo pulse.
NOTE: The ATmega4809 is a 3.3v device. A resistor divider will be needed to reduce the 5v output of the sensor to be compatible to the 3.3v input.

For a readout, the on-board LED will be used. The LED is connected, through a current limiting resistor, to PF5 and is active LOW. (i.e. a 0 written to I/O pin PF5 will turn the LED ON.)

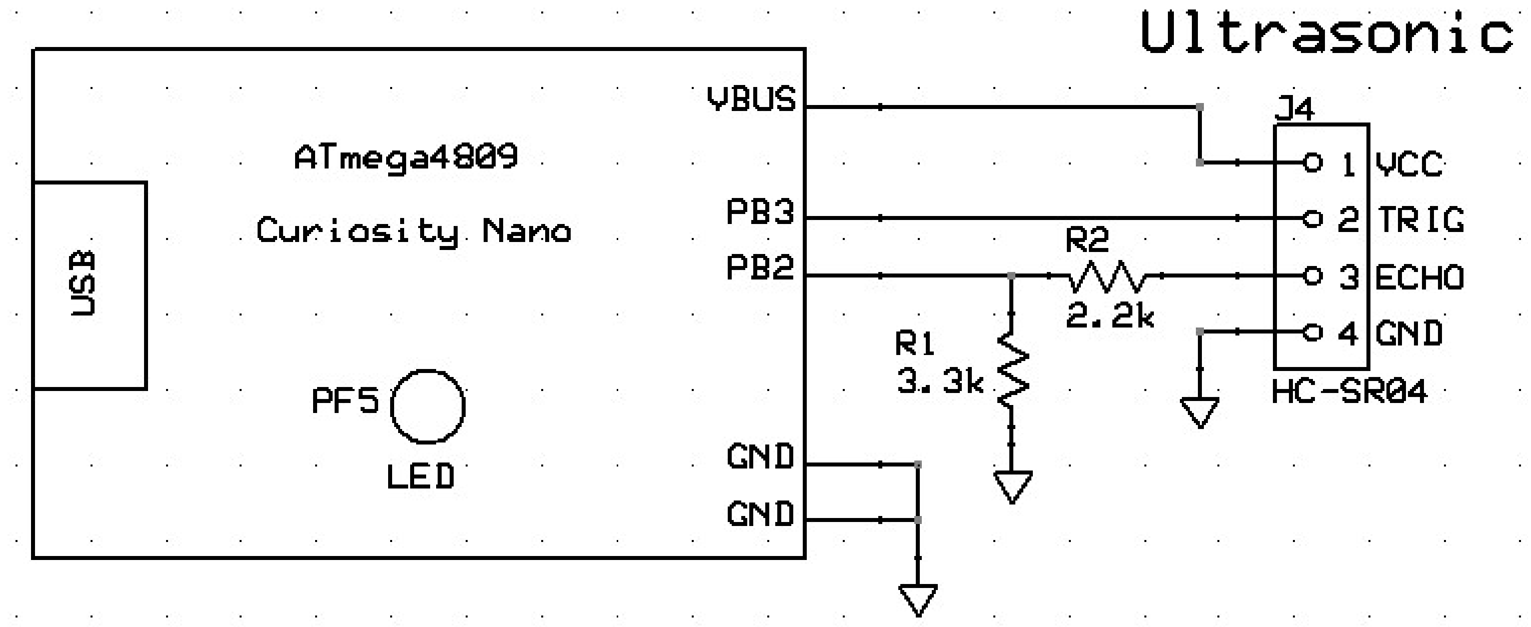

Figure 1 - Hardware schematic

The first effort is to use minimal code to verify that the sensor and readout can be controlled by the processor.
Project: HC-SR04_1
Resources needed:
- I/O output pin to trigger the sensor.
- I/O input pin to detect the Echo pulse.
- I/O output pin to control the LED.
- A timer to measure the width of the Echo pulse.
  - The timer needs to be able to count continuously for 38ms. For best resolution, should be 16bits.

Start by defining the hardware interface to be used by the code using names to reference the I/O pins and ports based on Figure 1. (see also main.c)
A 16bit timer clocked at 20 MHz takes 3.28ms to overflow.
    (1/20^6) * 2^16 = 0.00328 = 3.28 ms
Using a clock divide of 16 will cause it to take 52.4ms. This will allow it to time the longest Echo pulse.

For the ATmega4809, configure TCA (a 16bit Timer) for Normal Operation with a clock div of 8 and a Main Clock prescaler of 2. Set the Fuses for CLK20 when programming device.
For other processors, use a combination of clock and prescaler to have the maximum count of the timer close to the 38ms requirement.

Next, configure the port pins used to drive the sensor Trig pin and read the sensor Echo pin. Clear the Trig pin to 0 (LOW) to establish a starting condition.
Configure the pin for the LED drive to be an output and set the pin so that the LED is OFF.
(see main.c)
Now, with the hardware interface set up and the internal resources configured, some simple logic is needed to verify that the controller has control of the LED and sensor. The requirements for a simple test process could be:

```
Test Process:
    Blink the LED three times before entering a while() loop. This verifies LED control and notifies you of a RESET.
    In while() loop
        Generate a 10us pulse on the TRIG line.[1]
            Set TRIG HIGH to start the sensor transmission.
            delay 10us.
            Set TRIG LOW to end the pulse.
        Wait for the ECHO line to go HIGH to indicate that echo timing has started.[1]
        Start the Timer.
        Wait for the ECHO line to go LOW. This will occur after 38ms if no echo has returned. (i.e. no object detected)
        Stop the Timer.
        Read the Timer count.
        Reset the Timer to 0 in preparation for the next measurement.
        Use some criteria to display the results of the measurement.
            If the Timer count is < 1000 (about 150 cm), set the LED ON, else set the LED OFF.
    Repeat the while() loop.
```

Once this is working, basic control and input have been established. Now is the time to organize the code so that it can be used in other projects. (see Project: HC-SR04_1 : main.c)

Project HC-SR04_2:
There ended up being several initializations and operations that were called upon during the code execution in project HC-SR04_1. They could be grouped into three modules: Timer, LED, & Sensor. This would make it easier to reuse this code in other projects.

Initializations:
1. Configuration of the processor timer peripheral.
2. Configuration of the hardware for use with the HC-SR04 sensor.
3. Configuration of the hardware for use with the LED.

Operations:
1. Trigger the sensor.
2. Measure the Echo time.
3. Calculate the distance based on the Echo time.
4. Turn the LED on/off.
5. Sanity check(blink LED on RESET)

6. Start the Timer.
7. Stop the Timer.
8. Read the Timer.

This project will have a timer.c,.h module, a hc_SR04.c,.h module, and an led.c/.h module to better organize the code.
Allocating the above code elements to their respective modules we see that:

timer.c will contain
1. Configuration of the processor timer peripheral.
6. Start the Timer.
7. Stop the Timer.
8. Read the Timer.

hc_SR04.c will contain
2. Configuration of the hardware for use with the HC-SR04 sensor.
1. Trigger the sensor.
2. Measure the Echo time.

led.c will contain
3. Configuration of the hardware for use with the LED.
4. Turn the LED on/off.

The main() function will still contain
3. Calculate the distance based on the Echo time.
5. Sanity check(blink LED on RESET)

Although this seems a bit overkill, it allows flexibility. It's generally easier to combine code than it is to separated it.
By separating the code into their own modules, it will make it easier later to use a different timer, a different display,
or even a different sensor without having to modify the other code areas or even the main loop.

The next step is to define the functions to move the existing code into.

First, the timer.c module. Looking at it's list, four functions seem to be needed.

void init_timer()	- Configuration of the processor timer peripheral.
void start_timer()	- Start the Timer.
void stop_timer()	- Stop the Timer.
uint16_t timer_read()	- Read the Timer.

(see timer.c and timer.h)

Then the HC-SR04 sensor module. It will need three functions.

void init_hc_SR04()	- Configuration of the hardware for use with the HC-SR04 sensor.
void trigger_hc_SR04()	- Trigger the sensor. Blocking: This takes a maximum of 11 us.
uint16_t service_hc_SR04()	- Measure the Echo time. Blocking: This takes a maximum of 38 ms.

(see hc_SR04.c and hc_SR04.h)

And last, the LED module will need two functions.

void init_led()	- Configuration of the hardware for use with the LED.
void set_Led(bool state)	- Turn the LED on/off.

(see led.c and led.h)

Also add a sysdefs.h to take care of the delay.h and F_CPU definition.
(see sysdefs.h)

So, all that was done here was to break up the code into modules to better see what is happening. No interrupts have
been used.

This structure may be ok for a single task project like measuring the height of water in your pool or blinking a
light when your car is to close to the back of the garage, but the 10ms to 38ms blocking call to read the sensor takes up too much time for a multi-tasking project like a mobile robot or a project with multiple sensors. The next tutorial will address those issues.