PING))) DISTANCE MEASUREMENT

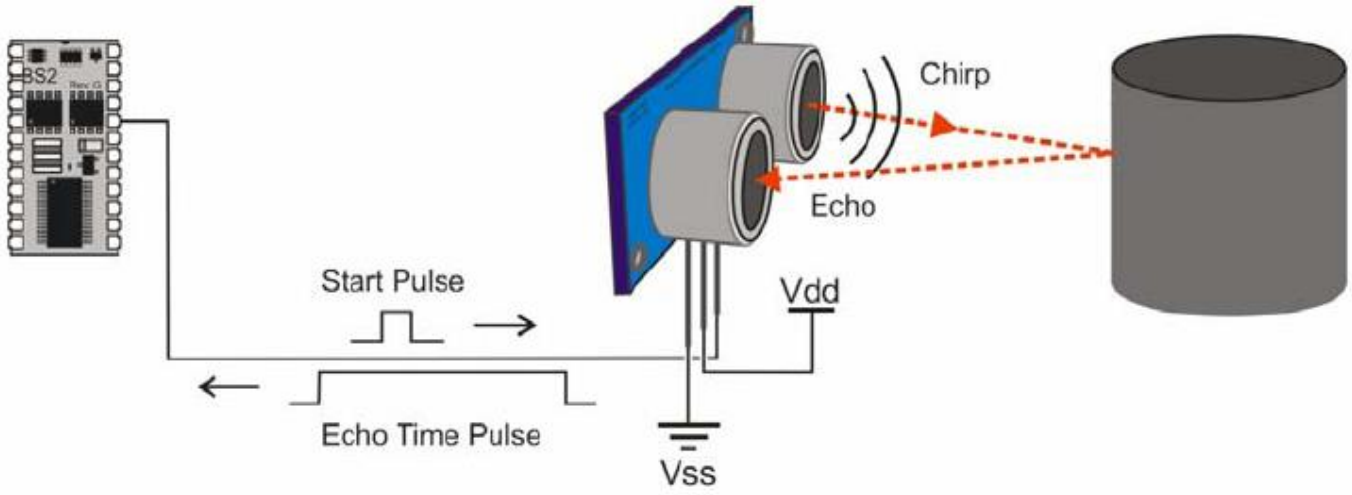
Introduction

Last week you learned how to use the ADC module and take distance measurements using the IR sensor. This week in lab you will be working with the timer module in order to pulse the PING))) sensor and take distance measurements. Your success in this lab is heavily based off of your understanding of the timer module and your ability to extract information from your resources, such as the Tiva Datasheet. As you work through this lab, think about the lab you completed for data analysis. When is taking measurements with the PING))) sensor preferred to taking measurements with the IR sensor?

PING))) Sensor

As previously mentioned, the range of the PING))) SONAR sensor is 2 cm to 3 m and will only report one echo for any given pulse. This one echo is the first received to meet the minimum threshold used to discriminate between noise and a proper echo. The pulse does spread as it travels away, so the first reflection may come from an object that is not directly in front of the sensor. Clutter **does** affect the usefulness of the sensor.

SONAR (**SO**und **N**avigation **A**nd **R**anging) uses an ultrasonic burst (well above human hearing range) to determine the presence and distance of objects. SONAR is a term that is valid for both air and water. The frequency of the pulses emitted are different for these two applications. We will be implementing a primitive SONAR. The sensor emits 40 KHz pulses and estimates distance by using the fact that objects reflect sound. The distance is calculated by determining the time between emitting a sound pulse and receiving an echo. The echo is translated into distance given the speed of sound in a particular medium. In our case, the medium is air. While air temperature does affect the speed of sound, for our purposes we will assume the speed of sound to be constant at 340 m/s or 1130 ft/sec.



The composition and shape of objects affects the amount of sound that gets reflected back to the sensor. A soft material like fabric will absorb more sound than a hard material like steel. It is possible to angle a box so that sound waves will reflect away from the sensor, so the box may "disappear" like the B2 or F117 US Air Force airplanes.

General Purpose Timers

The General-Purpose Timer Module (**GPTM**) contains **six** 16/32-bit GPTM blocks. Each 16/32-bit GPTM block can provide two 16-bit (half-width) timers/counters. 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. All timers have interrupt controls and separate interrupt vectors as well as separate interrupt handlers.

Each GPTM block can work in one of the following modes:

1. One-Shot Timer Mode
2. Periodic Timer Mode
3. Periodic Snapshot Timer Mode
4. Wait-for-Trigger Mode
5. Real-Time Clock Timer Mode
6. Input Edge-Count Mode
7. Input Edge-Time Mode
8. PWM Mode
9. DMA Mode
10. Synchronizing GP Timer Blocks
11. Concatenated Modes.

In this lab we will be making use of the Input Edge-Time Mode, in order to capture the time a pulse is sent and received back on the PING))) sensor.

Reference files

The following reference files will be used in successful completion of this lab:

1. Lab8_Evaluation_Form
2. lcd.c, a program file containing various LCD functions
3. lcd.h, the header file for lcd.c
4. Timer.c, A program file containing various wait commands
5. Timer.h, the header file for timer.c
6. TM4C123GH6PM Datasheet
7. PING))) Sensor Datasheet
8. Picoscope Guide

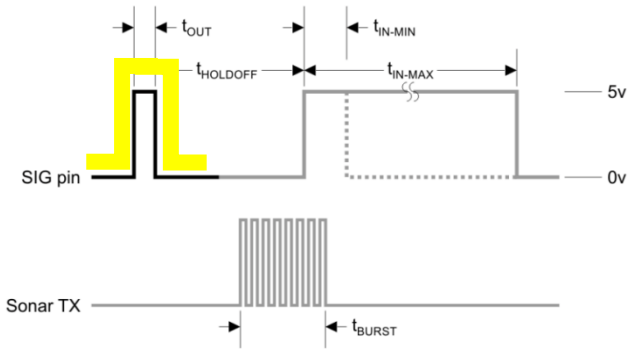
In addition to the files that have already been included for you, you will need to write your own **ping.c** file and **ping.h** file and the associated functions for setting up and using the input-edge time mode on the GPTM module. Separate functionalities should be in separate functions for good coding quality and reusability purposes. This means that in your ping.c file, you should write separate functions for initializing/configuring the PING))) sensor and taking a distance measurements. Remember to use good naming conventions for function names and variables. For example, you may want to name your PING))) initialization function **ping_init** and name your function to pulse the sensor **ping_read** as there will be other initialization functions you will write in later labs that will eventually have to be used together. Minimally, we recommend defining the following functions:

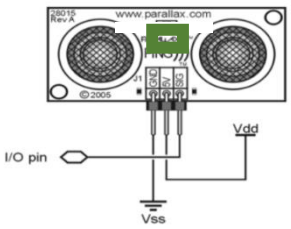
```
void ping_init(void);
int ping_read(void);
```

Part 1: Activating the sensor

Your first task is to trigger the PING))) sensor to emit a sonar burst by sending a short start pulse (low-high-low signal) out of GPIO Port B pin 3, PB3. The green LED on the sensor should blink at regular intervals when you are successful. **Use an oscilloscope** to verify the highlighted pulse occurs as expected, and to verify as you move an object to different distances in front of the Cybot the response pulse changes in a reasonable manner.

The sensor uses a single signal pin which is connected to GPIO **Port B pin 3 (PB3)**. In order to use the sensor, you must first trigger it. This trigger should be in the form of a short low-high-low digital pulse as shown in yellow. This means that you need to set PB3 as a **digital output** (disable the alternate function) and then set the pin low-high-low, as pictured. Refer to the PING))) datasheet for more information about the duration of the trigger pulse (how long it should be high) to meet the specifications of the PING))) sensor.





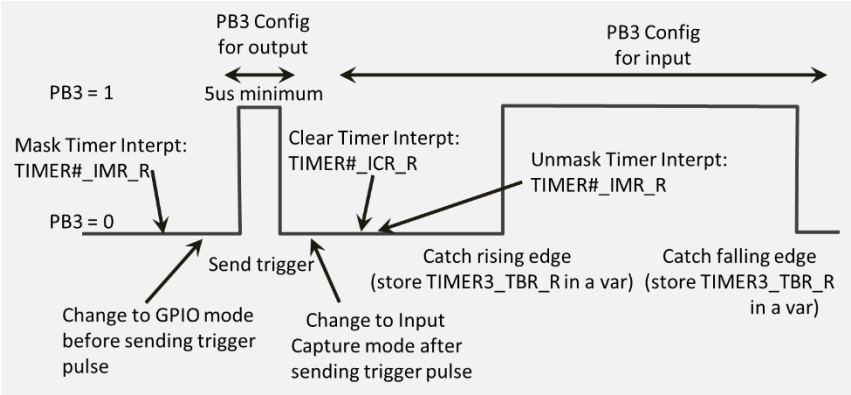
There is a green LED located between the two cones that will blink when the sensor is active.

Checkpoint: Demonstrate the PING))) sensor being triggered, and responding in a reasonable manner your mentor **based on an oscilloscope**. For this part, you are not yet required to read the sensor or calculate the distance. You will do that later.

Part 2: Determine the sensor echo pulse width

Now that the PING))) sensor is working, it will be sending an echo pulse to the microcontroller, and your program needs to determine the time (pulse width between the rising and falling edges) of the echo pulse. You will use the Input Edge-Time Mode of the GPTM timer, more generally known as "input capture." Be sure to initialize/configure Timer 3B for input edge-time mode. The timer should be in 16-bit **count-down mode** (**See Errata note, GPTM#11, at the end of Part 2***). The 8-bit prescaler will be used as an extension to get a 24-bit timer by initializing the ILR and PR registers to 0xFFFF and 0xFF respectively). You also need to configure which edges of the signal on the CCP pin will be detected. You should also configure and enable interrupts so that an edge generates an interrupt. You will need to write interrupt handler code.

Important note: You need to re-configure the GPIO Port B pin (PB3) to serve as the input FROM the PING))) sensor for this part of the lab. More specifically, PB3 is initially configured as a digital output to activate the sensor (as in Part 1), and is then dynamically reconfigured by your program as a timer input to detect the PING))) edges (this part of the lab). In this part of the lab, PB3 is used for both purposes to fully operate the sensor. This reconfiguration is done in pairs, as illustrated in the lecture slide copied at right. Every PING))) sensor reading requires first using PB3 as GPIO digital output to activate the sensor, then as timer input to get the pulse.



Note in the sensor datasheet timing specification, there is a guaranteed holdoff delay after the sensor receives the trigger pulse from the microcontroller and before it will respond with the rising edge of the echo pulse. **During this time, your microcontroller program must get ready to read the echo pulse. You will reconfigure the PB3 pin from digital output to timer input.** Since the sensor is using the same physical pin of the microcontroller for both input and output, you need to 1) change the data direction on PB3 from output to input (it's now going to receive the echo pulse signal), and 2) change to the alternate function for PB3 (it's now being used to detect the edges of the echo pulse, i.e., as an input capture pin for Timer 3B (T3CCP1)).

It might be helpful to sketch the pulse diagram above and think about the configuration code and other code for each section of the diagram. What does the code need to do for each section? Where is the code written in your program (e.g., main, functions, interrupt handler)? What registers are used? Take it one section at a time. You already did the first section (trigger pulse) in Part 1.

Use available resources to set up the GPIO Alternate Function Select (GPIOAFSEL) register (page 671 of the datasheet) and the GPIO Port Control (GPIOPCTL) register (page 688), which selects one of several peripheral functions for each GPIO pin. For information on the configuration options, refer to Table 23-5 on page 1351.

The echo pulse from the sensor will have a rising edge and then falling edge. The elapsed time between these two edges is directly proportional to the roundtrip distance between the sensor and the object. Input capture in edge-time mode will detect the edges of the signal and store the current counter value when an edge is detected into the GPTMTBR (TIMER3_TBR_R) register. The difference in counter values represents the pulse width in clock cycles. The time in seconds can be calculated based on the system clock frequency, which determines the amount of time for each counter value. The counter value for an edge is illustrated in datasheet Figure 11-4 (16-Bit Input Edge-Time Mode Example), which shows a 16-bit counter in count-down mode.

Set up the associated ports and pins to use the GPIO and timer modules. Refer to section 11.4.4 Input Edge Time Mode in the Tiva Datasheet; see also examples in chapter 9.2.8.2 Input Edge-Time Implementations in the textbook, the Timer ICE, and lecture slides.

Note: Do not forget to include "driverlib/interrupt.h" so that you can call IntMasterEnable() to globally enable interrupts. Also do not forget enable interrupts in the NVIC appropriately for Timer 3B.

Calculate the pulse width in clock cycles. Display this echo pulse time on the LCD in number of clock cycles. Vary the distance between an object and the sensor and observe the changing pulse widths.

Select a pulse-width value written to the LCD and verify that it makes sense with respect to the echo pulse timing range for the PING))) sensor given in the datasheet. **And check that it matches the time measured with an oscilloscope**

***Errata for our Microcontroller:** It is common for a complicated device, such as a microcontroller, to have known bugs (i.e. behaviors that do not match datasheet specifications). When companies discover these issues, they add them to what is called an Errata document. This document will typically describe: i) the conditions that cause a given bug, ii) the behavior caused by the bug, and iii) how to work around the bug. Below is from page 34 of our microcontroller Errata. It specifies a configuration of the Timers that should be avoided. The full Errata can be found as one of the reference documents for this Lab.

GPTM#11	***The Prescalar Does not Work Properly When Counting up in Input Edge-Time Mode When the GPTM Timer n Interval Load (GPTMTnILR) Register is Written With 0xFFFF***
Revision(s) Affected:	6 and 7.
Description:	If the GPTM is configured in Input Edge-Time count-up mode with the GPTM Timer n Interval Load (GPTMTnILR) register equal to 0xFFFF, the prescaler does not work properly.
Workaround(s):	Do not load 0xFFFF into the GPTMTnILR register when counting up in Input Edge-Time mode.

Checkpoint: Display and demonstrate to your mentor the PING))) echo pulse width in clock cycles. Your TA will also ask you to also demonstrate measuring and verifying the signal timing with the oscilloscope.

Part 3: Continuous distance measurement

From part 2, you now have a program that calculates the pulse width using input capture. You will now use this pulse width to estimate distance to the object. First calculate the pulse width in milliseconds, and then calculate the distance in centimeters. Try to be accurate with your distance calculations, as you will be using them for navigation later on.

Tip: Remember, for our purposes we will assume the speed of sound to be constant at 340 m/s or 1130 ft/sec.

Although the range of the sensor is expected to be between 2 cm and 3 meters, experiments have shown the practical range of the mounted PING))) on the floor of the lab to be roughly 5 cm to 275 cm. Reflections from the floor are the likely cause for the maximum range being limited compared to the datasheet.

You should perform a distance estimation every 200 - 500 milliseconds, and continuously display the pulse width (in both clock cycle counts and milliseconds) and distance in centimeters.

What if you calculate a negative pulse width? Note that the timer is running continuously, so the counter values may span across the 2^{24}-1 boundary (i.e., In countdown mode the first value is smaller than the second value read from the timer, means the time got to 0x00000, and then wrapped back to 0xFFFFF). When this happens, a timer "overflow" occurs (the timer rolls over from all 0x0's to all F's). Assuming count-down mode and at most one overflow for any pulse width measurement, a negative pulse width (i.e., new TIMER3_TBR_R is greater than the previous) indicates under/overflow. Is it okay to assume at most one overflow? Since the maximum echo pulse width is much shorter than the maximum 24-bit timer period of about 1 second in our configuration, there will be at most one overflow possible during a PING))) echo pulse. The overflow situation is more complicated if the pulse (or time between edge events) is longer than the maximum timer range or if there is more than one overflow. Your program can assume the simpler situation.

Remember, a negative pulse width doesn't make sense. You should use the overflow condition to appropriately adjust the pulse width calculation. Your program should also display a running count of overflows that occur.

Checkpoint: Display and demonstrate to your mentor the echo pulse width in clock cycles and milliseconds, the distance to the object in centimeters, and a running count of the number of timer overflows.

Part 4: Graphical User Interface (GUI)

Embedded systems typically have an application-specific interface. Such interfaces are often composed of: 1) physical components (e.g., buttons, knobs, switches, leavers, etc.) for providing input, 2) displays for the user to visualize aspects of the system such as status and/or information collected from sensors, and 3) graphical entities for display or collecting inputs in the form of a **Graphical User Interface (GUI)**.

In this part of lab, you will be introduced to a GUI development framework called "**Tkinter**" that comes packaged with most Python installations. This is a lightweight GUI framework for making simple GUIs quickly. While Tkinter can also be used for developing intricate GUIs, other GUI frameworks such as QT (C++-based), and pyQT (Python-based) are often instead used for developing more advanced GUIs. Once you learn a bit about Tkinter, other GUI frameworks will be easier to pick-up.

- **Making your first GUI**

Based on these resources:

  - RealPython tutorial: https://realpython.com/python-gui-tkinter/
  - Hello world: https://www.youtube.com/watch?v=yQSEXcf6s2I
  - Positioning objects: https://www.youtube.com/watch?v=BSfbjrqIw20
  - Interactive Push button: https://www.youtube.com/watch?v=yuuDJ3-EdNQ

1. Create a simple GUI that take some action when a Graphical Button is pressed
2. Explain to your TA conceptionally what the `.mainloop()` function does. (See RealPython tutorial).

- **Modify your first GUI**:

Read through the remainder of the RealPython tutorial, and **add one feature** to your GUI based on the information in the tutorial.

***Note**: Here are a couple nice Youtube Tkinter resources, if you want to learn more:

- 5-hour Tkinter course: https://www.youtube.com/watch?v=YXPyB4XeYLA
- Play list with videos covering a wide range of Tkinter GUI features: https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV

Checkpoint: Demonstrate your GUI program to your mentor, and explain what the code is doing, and walk them through conceptually the purpose/functionality of the `.mainloop()` function.

Bonus: IR Calibration using the PING))) Sensor

Because the PING))) sensor is fairly accurate in its distance measurement (i.e., it does not need to be rigorously calibrated similar to the IR sensor), it can be used to provide a known distance to calibrate the IR sensor. For example, move the CyBot backwards away from a wall in the test field, using a foam board provided in the lab as a wall. Use your movement API, UART, and IR code.

You should send data back via Putty using the distance from the PING))) sensor and also the quantized value of the IR sensor. Use this information to calibrate the IR sensor. Graph your results. The goal is that this will provide you with a quick calibration method for the final project.

Checkpoint: Demonstrate your calibration program to your mentor. You should demonstrate your program, as well as a graph showing your results.