# EE 209 Lab – Range Finder

## 1 Introduction

In this lab you will build a digital controller for an ultrasonic range finder that will be able to determine the distance between the range finder and an object in units of inches or centimeters.

## 2 What you will learn

This lab is intentionally will provide less guidance on actual datapath and control unit design and allow you to take high-level requirements and derive your own design and implementation. Along the way you will also learn about interfacing devices that use different signaling levels as well as how to emulate floating point operations on integer values.

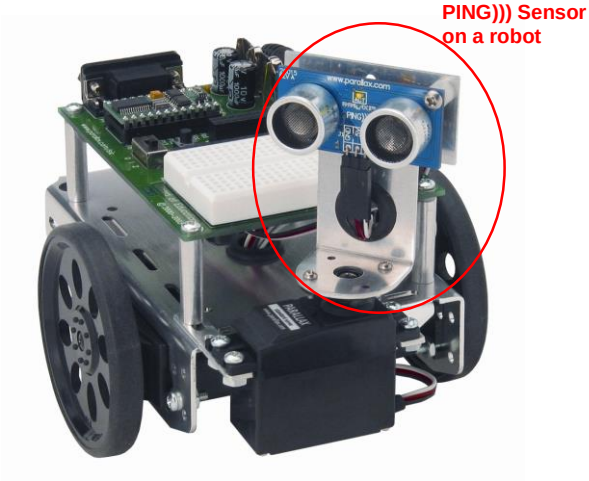
## 3 Background Information and Notes

### 1. Overview

Your design will send and receive signals to an ultrasonic range finder called the PING))) sensor. The sensor is designed by a company called Parallax and when told to start, emits an ultrasonic signal and determines the distance to an object by listening for the echo. The documentation for the sensor can be found here (https://www.parallax.com/sites/default/files/downloads/28015-PING-Documentation-v1.6.pdf). Spend some time reading about it and understanding the digital interface it requires. In particular, find the following information:

- How many signals (non PWR or GND) are exchanged between the digital controller and the PING sensor?
- What voltage levels does it expect for logic 0 and logic 1?
- How long a pulse must we send to initiate a sensing?
- How long do we need to wait before expecting a return signal?
- What range of time will the return pulse last?
- How long do we need to wait before starting our next sensing?

Answer these questions in the prelab (place your answers in a file named "prelab.txt"

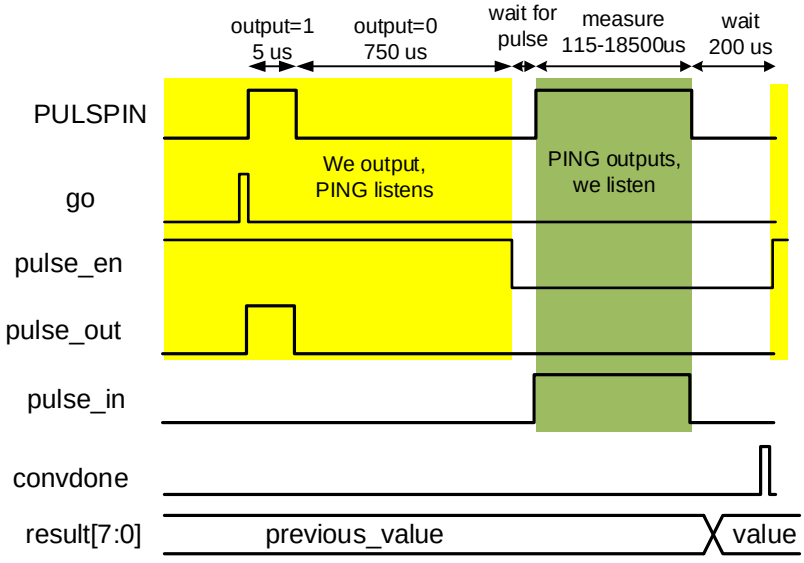



### 2. Bidirectional signals

Because only one signal is exchanged between our circuit and the sensor we will

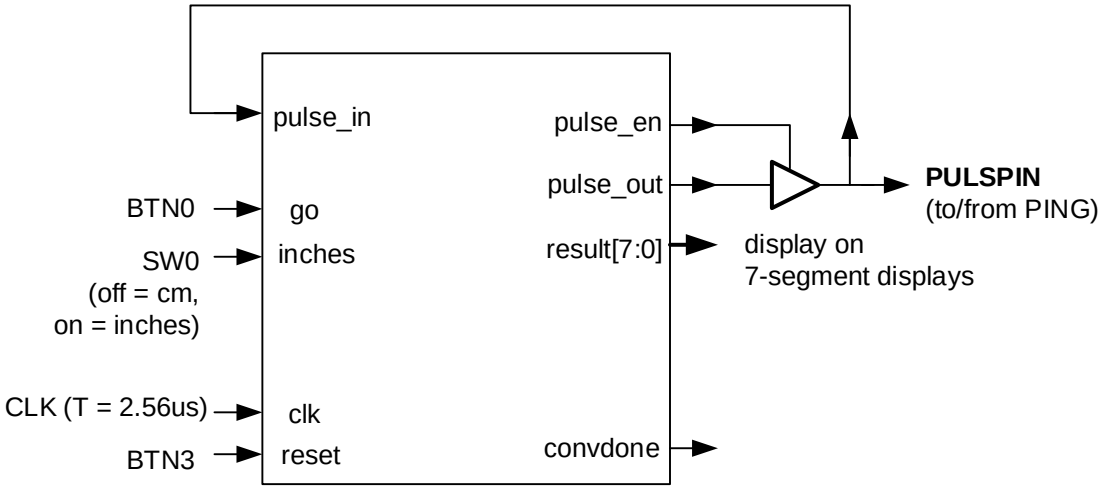
need to takes turns sending and receiving information. In almost all of our digital designs thus far bits always get produced by one circuit (gate) and consumed as an input by another. But in this case the same wire/bit will sometimes need to be output by your circuit and received as an input by the sensor, but later the sensor will output a value and your circuit must receive that as an input. To do this we use **tri-state gates**, but we will take care of that outside your design in our provided top-level module. You will receive an input signal (**pulse_in**) representing what the sensor is sending you, an output signal (**pulse_out**) that you are sending the sensor, and an enable signal (pulse_en) that you will also output. **This enable signal MUST be a 1 if you want to SEND your output to the sensor and 0 if you want to be receiving information from the sensor**. In the image below, the yellow area represents time where our controller sends a signal to the PING sensor. This is indicated by pulse_en = 1. Your design must ensure pulse_en=1 during the times you want to send a value to the PING by outputting the desired value on pulse_out. By changing pulse_en=0, the PING sensor can send data to you and you can look at the pulse_in input. **Be very sure (by checking in simulation) that your pulse_en = 0 when you should be receiving the pulse from the PING.**

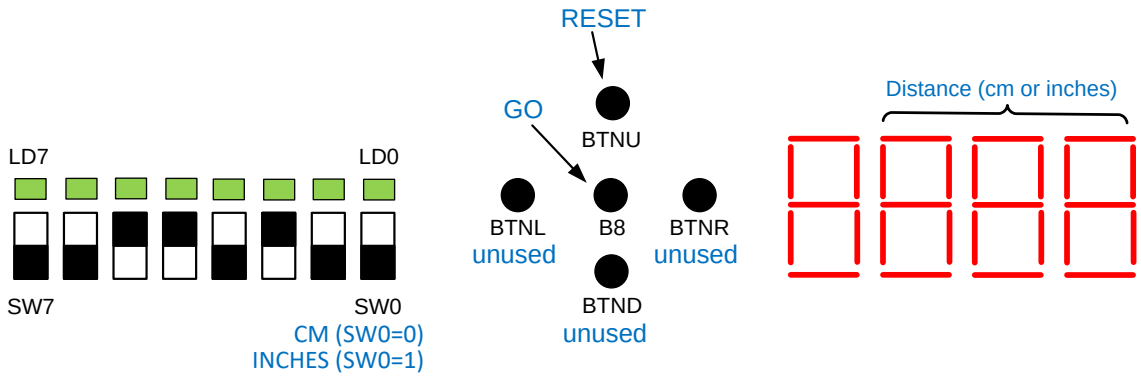
**Note**: In the diagram below **PULSPIN** represents *the actual bidirectional signal between your circuit and the PING sensor*. It is created from pulse_en, pulse_out and the PING sensor itself. In addition, our top-level module will use PULSPIN to produce pulse_in so that you can use its value to measure how long the pulse is high.




3. **Starting our Design:** The major task in this design is to produce a signal for a given period time. To do this we must know our clock frequency. In the block

diagram below, you will see that our top-level module provides your design a clock with T = 2.56 microseconds. Using this value and appropriately sized counters you can count the necessary clock cycles to wait for a specified duration before taking some action. A state machine controlling various counters and causing certain outputs to be generated for various durations will likely be sufficient for this design.




4. **Converting to Centimeters or Inches:** Once we generate the start pulse and then turn the output off for the needed time, we will look for a high-value on **pulse_in.** The duration of that pulse tells us the time it took for an echo to return to the PING sensor. Using the formula provided in the PING datasheet (page 3) for the speed of sound in air (note **Cair** corresponds to speed (m/s) and **Tc** represents temperature in degrees Celsius). Using a room temperature of around 78 degrees Fahrenheit, our clock period of T = 2.56 microseconds and the fact that the time we measure must be divided by 2 since we are waiting for the sound to travel **to and from** (i.e. round trip) the object, apply some arithmetic to arrive at a conversion factor (constant value) for computing centimeters per clock cycle). Put your value in your prelab.txt file.

Once you have this constant value (cm / cycle) we will multiply it times the actual clock cycles we count during the pulse_in high time. Suppose the constant you found is .02461 cm/cycle (just made up), if the pulse lasted 200 cycles then an object must be 200*0.02461 cm away. The problem is we only have integer-based hardware components and we'd rather not create floating point hardware. How can we then multiply by 0.02461? A commonly used trick is to scale the fractional value up by multiply by some power of 2 to make it an integer and then multiply the resulting integer as needed and finally divide by the power of 2 at the end, keeping only the integer value. We use a power of 2 because dividing by a power of 2 is as easy as shifting or just dropping bits. Essentially we'll perform:

$$= \left\lfloor \frac{(\boldsymbol{cycles} * \lfloor (\mathbf{0.02461} * \mathbf{2^{16}}) \rfloor)}{\mathbf{2^{16}}} \right\rfloor$$

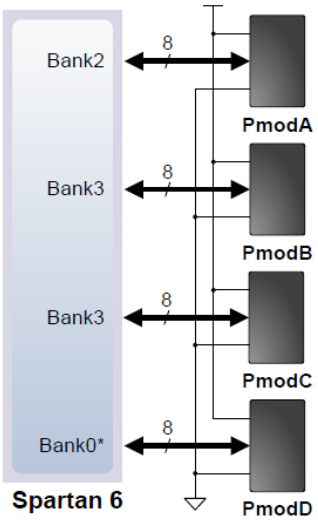
As an example, let us take 0.02461 * $2^{16}$ = 1612. So if we now multiply this scaled up value by our clock cycles (i.e. 200 * 1612 = 322,400) then divide out the $2^{16}$ factor we get *floor*(4.919) = 4 cm. Compare that to the actual value of (200*.02461 = 4.922).

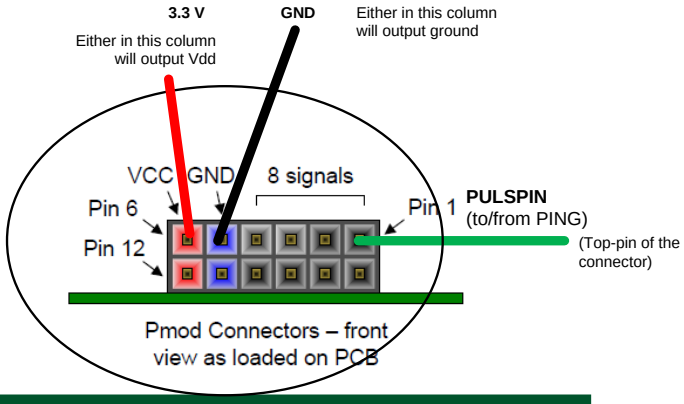
In hardware we can precompute the constant term (i.e. 0.02461 * $2^{16}$ = 1612) and just build a multiplier to multiply 1612*clock_cycles. To divide by 16 we just drop the lower 16-bits and only take the upper bits. Our ping sensor has trouble detecting values past 1 meter so we can safely limit the centimeter result to be between 0-255 (i.e. 8-bits). So from the multiplier we can take bits 23-16 of the result (because we will drop bits 15-0 to divide by $2^{16}$. Since 1 inch = 2.54 cm, we can also output our distance in units of inches by multiply by 1612/2.54 = 634.

Take some time now and compute the integer conversion factor for both centimeters and inches by first ensure you've calculated the correct conversion factor (cm/cycle) from above and then apply the technique just described to arrive at an integer constant that you will multiply by the cycle count.

5. **Converting Voltage Levels:** As you read in the PING))) datasheet, it requires 5V signals and uses TTL logic (a predecessor of CMOS), while our FPGA uses CMOS logic using 3.3V power (i.e. Vdd = 3.3V). In addition, CMOS current output capability is often smaller than TTL is able to source or sink. In essence, CMOS and TTL speak a slightly different language (a different dialect) from each other and thus need to be translated. For that purpose we have provided a separate circuit board (https://www.sparkfun.com/products/11771 breakout). By connecting 5V and GND from the PING sensor side, 3.3V and GND from the FPGA side, and then the actual signal going to and from each side the circuit will change the voltage levels for us (i.e. when our FPGA outputs 3.3V, the circuit

board will raise that signal to 5V. When the PING outputs 5V, the circuit board will lower it to 3.3V.




| Pmod Pinouts | | | |
|---|---|---|---|
| JA1: T12 | JB1: K2 | JC1: H3 | JD1: G11 |
| JA2: V12 | JB2: K1 | JC2: L7 | JD2: F10 |
| JA3: N10 | JB3: L4 | JC3: K6 | JD3: F11 |
| JA4: P11 | JB4: L3 | JC4: G3 | JD4: E11 |
| JA7: M10 | JB7: J3 | JC7: G1 | JD7: D12 |
| JA8: N9 | JB8: J1 | JC8: J7 | JD8: C12 |
| JA9: U11 | JB9: K3 | JC9: J6 | JD9: F12 |
| JA10: V11 | JB10: K5 | JC10: F2 | JD10: E12 |

# 4 Prelab

Write down your answers to the question regarding the PING))) data sheet in part 1 of the Background information. Also write down the cm/cycle conversion constant and the integer scaling constant for both centimeters and inches you found in part 4. Submit them in a file 'prelab.txt'

# 5 Procedure

Be sure you have read the Background Notes and Information before you start this project.

1. Download the skeleton project: **ping.zip** and extract it to a folder. In the folder you'll find:
    a. 16x16 bit multiplier with 32-bit output (mult16x16.v)
    b. 16-bit counter with reset and enable (cntr16ce.v)
    c. 16-bit equality comparator (compeq16.v)
    d. A D-FF with set and clear (dff1s.v)
2. Consider the sequence of states (steps) you need to go through for each use of the PING sensor and consider how many cycles you need to be in each state. Use counter(s) to track the duration (how many cycles) you have been in a state and use logic to determine when you hit the specific count where you want to move on to the next step.

3. Based on your thoughts and values from the previous step, design the state machine needed to generate the outputs: pulse_out and pulse_en as well as waiting the appropriate durations.
4. After the return pulse from the PING))) sensor finishes (coming in via **pulse_in** signal), your circuit should perform the conversion from raw cycle count to cm or inches (use the **inches** input to determine whether to convert to cm or inches). This will require performing the multiplication of the cycle count times the integer conversion constants you found. Remember you will only need an 8-bit answer but you must drop the lower 16-bits to perform the division by $2^{16}$. The 8-bit integer result should be saved in a register whose output is then fed as the output: **result**. This register should retain its value until the next system takes another sample and converts it (i.e. don't reset it to 0).
5. Finally, you should ensure the output **convdone** goes high for 1 clock cycle when the sensor is ready to take another sample. This is important because in our top-level file we will capture and display the result you produce when you make convdone go high for a cycle. Thus if **convdone** does not work correctly you will not see correct values on the FPGA 7-segment displays.
6. We have provided a small testbench. Use it to simulate your design and see if things work as expected. But we'd recommend adding another case or two with varying return pulse widths representing objects closer or shorter.
7. **Demonstrate your simulation to your TA so that they can ensure you pulse_en and pulse_out signals look correct.**
8. Once your TA has looked at your simulation you can synthesize, implement, and generate the programming file and go to one of the test stations with an FPGA and robot and download your design to see it working. Press the 'go' button slowly and leave some time in between. If you get a reading of 0, wait a second and try again. The FPGA connections are a bit temperamental.
9. Show your TA and get signed off.
10. Submit your ping.v and prelab.txt on the website.

# 6 EE 209 Lab PING Grading Rubric

Student Name: _____

TA sign off: _____

| Item | Outcome | Score | Max. |
|---|---|---|---|
| Datapath Correctness<br>• Correct cm and inch integer conversion values in prelab<br>• Correct PULSE_OUT period, holdoff period, and rest period<br>• Correctly counts duration of PULSE_IN.<br>• Conversion to inches and cm is done (correct multiplication and use of appropriate 8 output bits)<br>• Produces CONVDONE for a cycle | Yes / No<br>All / Some / None<br>Yes / No<br>Yes / No<br>Yes / No | | 2<br>3<br>1<br>1<br>1 |
| Simulation & Testing<br>• Simulation exhibits correct behavior | Yes / No | | 2 |
| SubTotal | | | 10 |
| Late Deductions (-1 pts. per day) | | | |
| Total | | | 10 |
| Open Ended Comments: | | | |