Computer Systems Design Lab

SYSC 4805 (Fall 2022)

Instructor: Dr. Mostafa Taha
Carleton University
Department of Systems and Computer Engineering

Lab 2: Line Follower, Analog Distance, and Ultrasonic Sensor Sept 15th and 16th, 2022

In this lab, we will conduct the following Tasks:

- Task 1: Testing the Line Follower Sensor
  - Experiment 1, using the Oscilloscope
  - Experiment 2, using the Arduino Due Board.
- Task 2: Testing the Analog Distance Sensor
  - Experiment 1, using Voltmeter
  - Experiment 2, using the Arduino Board
- Task 3: Testing the Ultrasonic Distance Sensor
  - Experiment 1: Generating the trigger signal using the Arduino Due Board
    - Exp. 1.A: Using Delay Functions
    - Exp. 1.B: Using the Analogwrite() Function
    - Exp. 1.C: Using the Timer Counter Unit
    - Exp. 1.D: Using the PWM Macrocell Unit.
  - Experiment 2: Testing the Ultrasonic Distance Module using Oscilloscope.
  - Experiment 3: Testing the Ultrasonic Distance Module using Arduino.
    - Exp.3.A: Using PulseLn() with trigger using Delay(), Timer and PWM.
    - Exp.3.B: Using Timer Unit for both triggering and measuring the Echo.
    - Exp.3.B: Enable and use of Interrupt on the Echo signal.

For every task, we start by introducing the sensor and its technical description. For each experiment, please add a picture or a screenshot to the results on the Oscilloscope or the screen monitor, and complete the provided tables.

Report Instructions

General:

- The report should be submitted before the beginning of the next lab. After that, there will be a penalty of 25% if the report submitted in the following 24 hours. If the report submitted 24 to 48 hours late, it would receive a penalty of 50%. No report will be accepted afterwards.
- The cover page should include your group names, the lab number, and the lab date.
- Organization is extremely important: Divide your report into sections, use captions for figures and headings for tables, …etc.

Lab 2: Your report should include the following

- All the Table completed.
- Answer to all the discussion questions in the experiments of each task.
- The required screenshots and pictures in the experiments of each task.

Task 1: Line Follower Sensor

Prepared By: Mahmoud Sayed

Introduction

A line follower sensor is intended to help the robot to detect and/or follow a path constructed as a dark (preferably black) line on a bright (preferably white) ground or a bright line on a dark ground. The main components in a line follower sensor are Infrared (IR) emitter and IR receiver. Its works by illuminating the surface by IR light and measuring the intensity of the reflected light by the IR receiver to decide the reflectivity of the surface, which, in turn, is translated to a proportional analog voltage. In normal conditions, a white surface/line reflects a large amount of IR radiation which is detected by the receiver, while a black surface/line would reflect much smaller amount of radiation. Hence, the sensor will be able to distinguish between white and black surfaces. Being an analog sensor, the output voltage can be any value between 0 and the power supply voltage depending on the amount of the reflected light; however, in a good setup with high contrast between the ground and the line the sensor output will be almost digital (on/off).

One sensor is enough to detect the line, but it is not enough to follow the line as the robot won't be able to decide to which direction it should turn. At least two sensors are necessary for the "following" functionality. While there are multiple possible configurations depending on the line width and the displacement between the two sensors, we will consider the case when the displacement between the sensors is larger than the line width as shown in Fig. 1. In this case, the black line has to be between the two sensors and unseen by any of them. If the left sensor sees the black line, this means the robot is getting out of track and should turn to the left. Similarly, if the right sensor sees the black line this then the robot should turn to the right.

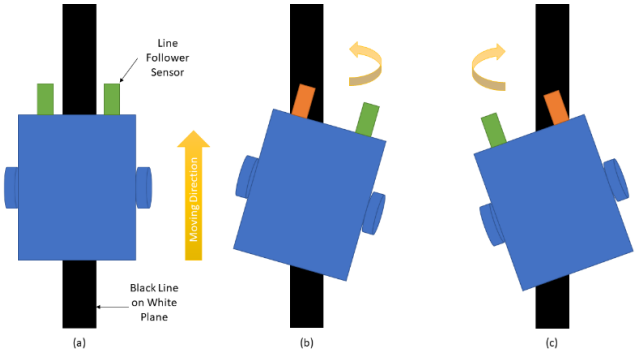

Fig. 1 A vehicle with two line follower sensors moving on black track (a) the vehicle is correctly oriented with the black line undetected by both sensors (b) the left sensor is detecting the line and the vehicle should turn to the left (counter clockwise) (c) the line is detected by the right sensor and the vehicle should turn to the right (clockwise).

Expectedly, more line follower sensors would give better tracking resolution and more info for the robot to make better control decisions for the motors. The line follower sensor from Robotshop has three-line follower sensors combined on the same Printed Circuit Board (PCB). The PCB design allows breaking it into 3 standalone sensors if required. A summary of the module's specifications is provided in table 1. For further information, refer to the datasheet.

The datasheet can be found here:
https://www.robotshop.com/media/files/pdf2/linefollowerdatasheet.pdf

Table 1: VMA330 module specifications.

Operating Voltage (Nominal)	5V
Output Type	Analog voltage (0-Vcc)
Sensing Range	Approx. 3mm to 6 mm (3mm optimal)
Supply Current	75 mA for all 3 sensors.
I/O Interface	VCC, GND, OUT1, OUT2, OUT3
Dimensions	55.0 × 19 × 3.17 mm

Experiment 1: Test using the Oscilloscope

In this experiment, you should test the basic functionality and performance of the line follower sensor module.

Required Equipment:

1- Line follower sensor.
2- DC Power supply (we will use the Arduino itself).
3- Oscilloscope.
4- White paper with printed black tracks.
5- Breadboard.
6- Jumper wires.

Procedure:

1- **Configure the power supply**: we will use the 3.3V pin of Arduino board itself as a power source to the sensor.
2- **Make the connections**: Use a breadboard to connect the VCC and GND pins of the sensor module to the 3.3V and GND terminals of the Arduino Due board.
3- **Connect the scope**: While there is no real time-varying signal to be displayed, using the 2 channels of a scope to monitor the output of two of the sensors would give a simultaneous real-time view for the outputs with response time that is better than using the voltmeters.
   - Connect two probes to the two channels of the scope, if not already connected, and make sure that the multiplication factor on the probe is disabled (×1, not ×10).
   - Turn on the scope, enable both channels on the display, move the ground lines of both channels away from each other i.e. one near the top of the screen and the other near the

bottom, set the voltage division for each of them to a suitable value (1V/div would be suitable), and check the functionality of both channels by connecting them to the 5V square wave terminal in the scope. The time division will not be of a big effect as the monitored signal is essentially DC.
- Connect the ground of the two channels to the circuit ground and connect the channels to two of the sensor's outputs.

4- **Check the operation**: Get the paper with the printed black track and move it (or the sensor itself) below the sensor, alternating the white and black portions of the paper below both of the monitored sensors, and make sure that the paper does not touch the sensors directly, a displacement of 3 mm would be good. There should be a variation in the output voltage of a sensor when the black line moves below it compared to the voltage produced when the white portion is below the sensor. This will be visualized as a shift in the DC line displayed on the scope.

5- **Measure and Record**: Once you make sure the sensor is working as expected, record the following measurements in Table 2
- The output voltage when there is nothing near the sensor.
- The output voltage when the white portion of the paper is below the sensor.
- The output voltage when the black track is below the sensor.

Table 2: Experiment 1 Measurements

Case	Voltage Output
Nothing below the sensor	
White plane below the sensor	
Black track below the sensor	

Experiment 2: Test using the Arduino Due Board

In this experiment, you should interface the line follower sensor with the Arduino Due board. The Arduino board should be able monitor all of the 3 sensors in the module together and display their detection state on the serial monitor.

Required Equipment:

1- Arduino Due board.
2- Line Follower sensors module.
3- White paper with printed black tracks.
4- Breadboard.
5- Jumper Wires.

Procedure:

1- **Make the connections:** Using the breadboard when required
- Connect the GND pins of the Arduino and the module together.
- Connect the output pins of the module to digital IO pins in the Arduino.

- Connect the Vcc pin of the module to the 3.3V pin in the Arduino. Do NOT connect the Vcc pin of the module to the 5V pin of the Arduino (Why?)

2- **Write your code:** Develop an Arduino sketch that does the following:
   - Choose three digital pins on the Arduino, and define them as LFS_L, LFS_M, and LFS_R.
   - The Arduino reads the digital values of the three input pins, once every 500 msec. You can use the digitalRead() function and delay().
   - The Arduino should send the readings to be displayed on the serial monitor, including your section and group number, in a format similar to this one "L#, Group# >> Left: B – Middle: W – Right: W" which means the left sensor sees black, and the middle and the right sensors are seeing white.
3- **Check the operation:** Now move the paper below the sensor, alternating the white and black portions below the sensors and monitor what is displayed on the serial monitor. If everything works as expected, take a screenshot from the serial terminal and include them in your report.

Discussion

1- Does the module work with 3.3V?
2- When there is nothing near the sensor, is the measured voltage equivalent to the case when there is white surface below the sensor or black surface? Why?
3- Why shouldn't the surface be touching the sensor directly?

Task 2: GP2Y0A51SK0F Distance Measuring Sensor

Prepared By: Mahmoud Sayed

Introduction

The GP2Y0A51SK0F is an analog distance measurement sensor that can be used to detect the existence and distance to objects in front of it. Its working principle is based on transmitting Infrared (IR) signal and receiving the reflected signal from on object with an analog IR receiver that measures the intensity of the reflected signal and produces an analog voltage proportional to the distance of the object. The module contains a signal processing circuit that attenuates the effects of object's reflectivity, environmental temperature, and the operation duration by applying the triangulation method to give a reliable distance measurement. This specific model has a short measurement distance of 2 cm to 15 cm. The basic specifications and working conditions of the GP2Y0A51SK0F module are included in Table 1. Please refer to the datasheet for the absolute maximum ratings and other details.

The datasheet can be found here: https://www.robotshop.com/media/files/pdf/GP2Y0A41SK0F.pdf.pdf

Table 1: VMA330 module specifications.

Operating Voltage	4.5V to 5.5V
Output Type	Analog voltage
Distance Range	2 to 15 cm
Operating Temperature	-10° C to +60° C
I/O Interface	VCC, GND, and OUT
Dimensions	27.0 × 10.8 × 12.0 mm
Weight	2.7g

Experiment 1: Test using Voltmeter

In this experiment, you should test the basic functionality and performance of the GP2Y0A51SK0F module.

Required Equipment:

1- VMA330 module.
2- DC Power supply (we will use the Arduino itself).
3- Voltmeter.
4- Breadboard.
5- Jumper wires.

Procedure:

1- **Configure the power supply**: we will use the 3.3V pin of Arduino board itself as a power source.
2- **Make the connections**: Use a breadboard to connect the VCC and GND pins of the sensor module to the 3.3V and GND terminals of the Arduino Due board.
3- **Check the operation**: Move a white paper in front of the module and change its distance from the module to check its functionality. Use a voltmeter to measure the output voltage of the module, you should notice a variation in the measured voltage with the distance of the paper.
4- **Measure the distance-voltage relationship**: Preform systematic measurements to precisely define its characteristics and find the relation between the distance and the output voltage. Use a ruler to measure the distance between the sensor and the white paper (should in a perpendicular direction from the sensor plane) and measure the output voltage in the range from 0 cm to 25 cm with a distance resolution of 2 cm.
5- **Report the measurements**: Write down the measurements you got in table 2
6- **Change the paper:** Now repeat the experiment with a black paper or a dim object and report the results in table 2.
7- **Change the Vcc voltage to 3.3V**: Is the sensor still producing reliable results? If so, fill down the results in Table 2.
8- **Draw The Data**: Plot a figure that contains the data in table 2 with the distance on x-axis and voltage on y-axis. Make sure to indicate accurate labels on the axes and a legend if the figure contains multiple graphs.

Table 2: Experiment 1 Measurements

Distance (cm)	White Paper, Voltmeter	Black Paper, Voltmeter	White Paper, Arduino
0			
2			
4			
6			
8			
10			
12			
14			
16			
18			
20			
22			
24			
∞			

Experiment 2: Using the Arduino Due Board

In this experiment, you should interface the GP2Y0A51SK0F module with the Arduino Due board. The Arduino board should be able to measure the output voltage of the module and display it as a message on the serial monitor.

Required Equipment:

1- Arduino Due board.
2- GP2Y0A51SK0F module.
3- Breadboard.
4- Jumper Wires.

Procedure:

1- **Make the connections:** Using the breadboard when required
   - Connect the GND pins of the Arduino and the VMA330 together.
   - Connect the output pin of the module to an analog input pin in the Arduino.
   - Connect the Vcc pin of the module to the 3.3V pin in the Arduino. Do NOT connect the Vcc pin of the module to the 5V pin of the Arduino (Why?)
   - Connect the 3.3V pin of the Arduino to the Analog Reference (AREF) pin in the Arduino.
2- **Write your code to read Analog Voltage:** Develop an Arduino sketch that does the following:
   - The Arduino reads the analog value of the analog input pin, at which the output pin of the GP2Y0A51SK0F is connected, once every 500 msec. You can use the analogRead() function.
   - Convert the Analog read value to a voltage using the following equation:

$$V = \frac{Analog\ Read}{1023} * V_{AREF}$$

   - The Arduino should send the read voltage value V to the computer using the serial terminal to be displayed on the serial monitor. Please include your Section # and Group # .
3- **Compare with table 2:** Now change the distance of the white paper similar to what you did in Experiment 1 and compare the Arduino measurements to the measurements in table 2.
4- **Write your code to directly read the distance:** Modify the Arduino sketch so that the Arduino sends not only the voltage value to the serial monitor, but also the distance to the detected object. Use the data you got in table 2 as your reference.

Discussion

1- What is the IR wavelength emitted by the LED in the GP2Y0A51SK0F module?
2- What are the minimum and maximum obstacle distances that can be detected by this module?
3- What are the corresponding maximum and minimum output voltages? Are the voltage boundaries safe to the analog input pins of the Arduino Due?
4- For an object distance of 6 cm, what is the difference in the output voltage when using the white paper and black as obstacles?
5- Using curve fitting techniques, get an equation that describes the distance as a function of the output voltage in the operational range (using the white paper case as a reference).

Task 3: Testing the Ultrasonic Distance Sensor

Prepared By: Mahmoud Sayed

Introduction

The Ultrasonic distance sensor is used by Remotely Operated Vehicles (ROVs) or Autonomous vehicles to measure distances to obstacles for the purposes of detection and avoidance. Its working principle is similar to Radars: Emit a signal and measure the time it takes to come back again after reflection from the object or obstacle to calculate the distance. The main difference between the Radar and the ultrasonic sensors, from its name, is that the former uses electromagnetic waves while the later uses mechanical (acoustic) waves with 40 KHz in the ultrasonic range (cannot be heard by human ears).

Working Principle

The module does not have a processing unit, which means it does not output the distance measurement as a direct quantity. Instead, it outputs an electric signal that is proportional to the measured distance. To understand how it works, let's have a look at its 4 pins.

- **Vcc**: The power pin. Should be connected to the DC supply.
- **Trig**: The trigger pin is an input to the sensor that is driven by the microcontroller to initiate a measurement process.
- **Echo**: The echo pin is an output from the sensor. The sensor outputs a signal on this pin that is proportional to the measured distance.
- **Gnd**: The ground pin. Should be connected to the circuit ground.

The timing diagram of the sensor signals is illustrated in Fig. 1.

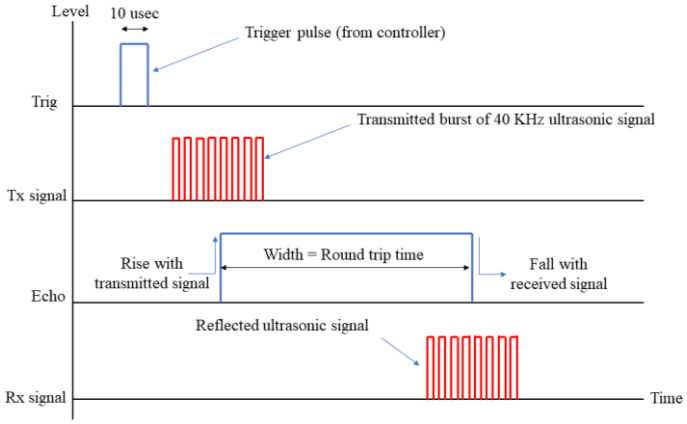

Fig. 2 Timing diagram of the ultrasonic sensor's signals

As can be seen from Fig.1, the measurement process starts by sending a 10 usec pulse from the microcontroller to the Trig pin. Then, the sensor begins transmitting a burst of 40 KHz ultrasonic pulses from its Tx terminal and it also raises the voltage on the Echo pin at the same time. When the reflected ultrasonic is received at the Rx terminal, the sensor pulls the voltage on the Echo pin to the Gnd voltage. Hence, the width of the pule generated on the Echo pin represents the time taken by the ultrasonic signal to travel from the Tx to an obstacle and back to the sensor again i.e., the roundtrip time. Knowing that the speed of sound in air is approximately 343 m/sec, if the echo pulse width is T_p then the distance to the obstacle D can be calculated as:

$$D = 343 \times \frac{T}{2} \ (m) \tag{1}$$

The division by 2 is required since T is the round-trip time. Table 1 gives a summary of the sensor specifications:

Table 1: VMA330 module specifications.

Operating Voltage	3.3V to 5V
Acoustic frequency	40 KHz
Measurement Angle	15°
Range (claimed)	5.5 m
Trigger format	10 usec pulse.
Output type	Pulse width modulated output
Working temperature	-10 to +60 °C

Some Practical Considerations

For the sensor to operate properly, the following points should be carefully considered.

1- **Trigger signal:** The trigger signal should be a rectangular pulse with minimum recommended width of 10 usec. Each trigger pulse will initiate a single measurement cycle. So, repeated pulses are required for repeated measurement cycles i.e. continuous measurement. In this case, the trigger signal will be then as a short duration periodic pulse or, equivalently, a pulse width modulated signal with low duty cycle. The minimum width of the pulse is specified by the datasheet to be 10 usec, but what about the period? What should be the distance between two consecutive pulses? To answer this question, we can assume that the sensor will work at its full range of 5.5m. Before triggering the sensor for the cycle n, we should wait to make sure that cycle $n-1$ is completely finished by giving the ultrasonic signal enough time to reach the maximum distance and come back again. Assuming the maximum distance is 5.5 m, then the total round-trip distance is 11 m and the maximum round-trip time T_{max} would be:

$$T_{max} = \frac{11m}{340m/sec} = 32.35\ msec \tag{2}$$

This means that the time spacing between two consecutive pulses should not be less than 32.35 msec, or the trigger pulses frequency should be no more than 30.91 Hz. You may need to increase the period more than that to accommodate for another practical issues, it is advised for some ultrasonic sensors that the period be 60 msec at least.

2- **Sensor Orientation**: The sensor has a measurement angle of 15°, which can cause erroneous readings if the sensor is placed very close to the ground because of unwanted reflection from the ground surface.
3- **Echo Signal**: The width of the echo signal should be measured in order to calculate the distance. Measuring time depending on the microcontroller's CPU solely will cause blocking behavior: The CPU will be stuck while measuring the time and won't be able to do anything else, including monitoring other sensors for example, till the measurement process ends. Usually, this is unwanted behavior. Most microcontrollers have modules called "timers" that are dedicated to the timing tasks. The SAM3X8E microcontroller of the Arduino Due is indeed full of resources and has 9 timer modules with massive number of options. You may need to utilize these timers to measure the time to avoid blocking the CPU.

Experiment 1: Generating the trigger signal using the Arduino Due Board

Although we could use a function generator to generate the required triggering signal, we will use the Arduino board itself to complete this task. In this experiment, we need to generate a pwm signal (square wave) with a period of 60 msec and a pulse width of at least 10 usec, similar to the following figure:

Fig. 3 Timing diagram of the ultrasonic sensor's signals

Required Equipment:

1- Arduino Due Board
2- Oscilloscope.
3- Breadboard.
4- Jumper wires.

There are three methods that could be used to generate the required triggering signal on the Arduino Due Board: 1- using delay functions, 2- using the AnalogWrite function, 3- using the Timer-Counter (TC) modules, 4- using PWM cell.

Procedure for Exp. 1.A: Using Delay Functions

1- Define any pin as TRIG_PIN
2- Initialize the TRIG_PIN as a digital output using pinMode().
3- Write HIGH on this digital pin, using digitalWrite(), then delay for 10 usec (microsecond) using delayMicroseconds().
4- Write LOW on this digital pin, using, then delay for 50 msec (millisecond) using delay().
5- Connect the Oscilloscope CH1 probe to the selected TRIG_PIN, and the group to the GND pin in the Arduino.
6- Change the Trigger source to be CH1 itself.
7- Watch the generated signal, take a picture to include it in your report

8- Measure the period of the generated signal. Although we mandated 10+50 = 60usec period, the period on the Oscilloscope will show a slightly higher value. Record this value and explain the reason for this difference.

Although this method is very easy, we can't use it in our project since we are blocking the processor from doing any other task.

Procedure for Exp. 1.B: Using the AnalogWrite() **Functions.**

1- Define any pin as TRIG_PIN
2- Just use the function `analogWrite`(TRIG_PIN, `value`). The `int` value is the duty cycle: between 0 (always off) and 255 (always on) in its default precision. We need a duty cycle that is so small, 10usec high / 60msec * 255 the default scaling of Arduino = 0.0425. Set the duty cycle to the minimum available value of 1 using `analogWrite`(TRIG_PIN, `1`).
3- Connect the Oscilloscope CH1 probe to the selected TRIG_PIN, and the group to the GND pin in the Arduino.
4- Change the Trigger source to be CH1 itself.
5- Watch the generated signal, take a picture to include it in your report
6- Measure the period of the generated signal.

Although the signal will be generated with a small duty cycle, the period will too small. The frequency is much higher than required. Once again, we can't use this function since we can't control the period (only the duty cycle is adjustable).

Procedure for Exp. 1.C: Using the Timer Counter Unit.

From Page 4 in the datasheet, we have 3 Timer Counter Units. Details about these units are available Pages: 856 to 908.

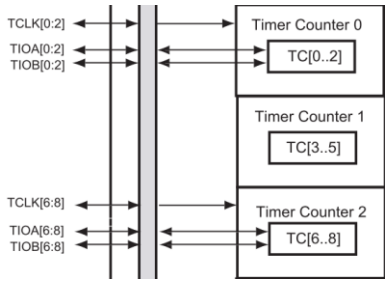

Fig. 4 Timer Control units onboard the SAM3X8E microcontroller

- Connect the Oscilloscope CH1 to read the trigger signal from pin 2 on the Arduino Due board.
- Write the following code. This code generates the trigger signal on pin 2. The code goes as follows:
  1- Enable the Timer Control TC0, power ON. This is Timer Counter 0 channel 0.
  2- Set the multiplexer of this pin to be used as Peripheral B (Timer Control) instead of A (GPIO).
  3- Select frequency of the source clock and its properties. Modify the register TC_CMR of TC0 Channel 0 to selects the master clock divided by 2 (Mck/2), with Clk on rising edge, in Waveform mode.
  4- Select the frequency and duty cycle. Since we need a period of 60 msec, we require a frequency of 16.667 Hz. Since ($TC\_RC = Mck/2 \ / \ Freq$), then assign TC_RC = 2520000.

Note that the value written in the code is 2520000-1 since the counter starts counting from zero.

5- Also, we need the duty cycle to be 10/60,000 = TC_RA/TC_RC. Given TC_RC = 2520000, then TC_RA should be 420.
Note that the value written in the code is 420-1 since the counter starts from zero.

6- Finally, enable the required interrupts to let the time run.

```
/*Sample Code to generate PWM signal with period 60 msec and pulse duration 10 usec
  From Timer 0 Channel 0.
  The output pin is B25, which is Arduino Pin 2.
  Using Register level access
*/
void setup() {
  PMC->PMC_PCER0 |= PMC_PCER0_PID27;//TC0 power ON - Timer Counter 0 channel 0
  PIOB->PIO_PDR |= PIO_PDR_P25;     // The pin is no more driven by GPIO
  PIOB->PIO_ABSR |= PIO_PB25B_TIOA0;// B Assign B25 to alternative periph_B (TIOA0):

  TC0->TC_CHANNEL[0].TC_CMR = TC_CMR_TCCLKS_TIMER_CLOCK1 //MCK/2 = 42 MHz,
                              | TC_CMR_WAVE              //Waveform mode
                              | TC_CMR_WAVSEL_UP_RC      //Count UP mode till RC
                              | TC_CMR_ACPA_CLEAR //Clear TIOA0 on RA compare match
                              | TC_CMR_ACPC_SET;  // Set TIOA0 on RC compare match

  TC0->TC_CHANNEL[0].TC_RC = 2520000-1;  //Set the frequency to 66.667Hz (Period 60 ms)
  TC0->TC_CHANNEL[0].TC_RA = 420-1;  //Set the duty cycle (Pulse of 10 usec)

  TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_SWTRG //Software trigger TC0 channel 0 counter
                               | TC_CCR_CLKEN;//and enable
}

void loop() {
}
```

- Change the Trigger source to be CH1 itself.
- Watch the generated signal, take a picture to include it in your report
- Measure the period of the generated signal.

Procedure for Exp. 1.C: Using the PWM Macrocell Unit.

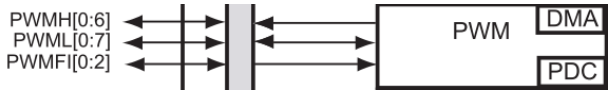

Fig. 5 PWM macrocell unit onboard the SAM3X8E microcontroller

SAM3X8E has a dedicated PWM generation module. This module is independent of the timers and can generate 8 independent PWM waves with high precision and broad synchronization options compared to timer modules. Details about the PWM macrocell are available in the datasheet, pages: 970 to 1052. The PWM macrocell controls 8 channels independently. The following code uses the 14-bit resolution on channel 0, pin 6.

- Connect the Oscilloscope CH1 to read the trigger signal from pin 35 on the Arduino Due board.
- Write the following code. This code generates the trigger signal on pin 35. The code goes as follows:
  1- Enable the PWM controller using the `PMC->PMC_PCER0` register.
  2- Set the PWM controller's output to Pin 60 on the SAM3X8E chip, which is pin PC3 (pin3 in port C), which is connected to pin 35 of the Arduino Due Board, using the `PIOC->PIO_ABSR` and `PIOC->PIO_PDR` registers.
  3- Set the PWM clock relative to the microcontroller's master clock (which is 84MHz), using the `PWM->PWM_CLK` register.
  4- Set the PWM mode of operation, using the `PWM->PWM_CH_NUM[0].PWM_CMR` register, to be clocked from clkA and be left aligned.
  5- Finally, set the frequency and duty cycle using the `PWM->PWM_CH_NUM[0].PWM_CPRD` and `PWM->PWM_CH_NUM[0].PWM_CDTY` registers respectively. It's also necessary to enable channel 0 with the `PWM->PWM_ENA` register.

```
/*Sample code to configure the PWM module to generate a PWM signal on pin 35,
which is PC3 - Arduino physical Pin 60.
 * Period: 60 msec, Pulse duration: 10 usec.
 * Register Level Access
 */
void setup() {
  PMC->PMC_PCER1 |=  PMC_PCER1_PID36;   // Enable Clock to PWM module
  PIOC->PIO_ABSR |= PIO_PC3B_PWMH0;     // Assign C3 to PWM module (Periph_B)
  PIOC->PIO_PDR |= PIO_PDR_P3;          // Release C3 from the PIO module

  REG_PWM_CLK = PWM_CLK_PREA(0) | PWM_CLK_DIVA(84);//Set PWM clock 1MHz (Mck/84)
  PWM->PWM_CH_NUM[0].PWM_CMR |= PWM_CMR_CPRE_CLKA // Set the clock source as CLKA
                              | PWM_CMR_CPOL;      //Set output polarity be high.

  PWM->PWM_CH_NUM[0].PWM_CPRD = 60000-1;   //Set PWM freq 1MHz/(60000) = 66.667Hz
  PWM->PWM_CH_NUM[0].PWM_CDTY = 10-1;      // Set PWM duty cycle

  PWM->PWM_ENA = PWM_ENA_CHID0;            // Enable the PWM channel
}

void loop() {
}
```

- Change the Trigger source to be CH1 itself.
- Watch the generated signal, take a picture to include it in your report
- Measure the period of the generated signal.

Note that, for experiments 1.A and 1.B, we used some code inside the loop function which means the CPU is always busy executing these commands. However, in experiments 1.C and 1.D, we don't have a single line of code inside the loop function. The CPU executes some code in the setup function once, to program some control registers, then is completely idle and available for any other task, e.g., monitoring other sensors or controlling the motors. The trigger signal is being generated and managed by other

units onboard the SAM3X8E, namely the Timer Counter unit and the PWM unit, that are running independent from the CPU.

Experiment 2: Test the Ultrasonic Distance Module using Oscilloscope.

In this experiment, you should test the basic functionality and performance of the ultrasonic distance sensor module.

Required Equipment:

1- Ultrasonic distance sensor module.
2- The Arduino Due board to generate the trigger signal.
3- Oscilloscope.
4- Breadboard.
5- Jumper wires.

Procedure for Experiment 2:

1- **Configure the power supply**: we use the 3.3V pin of Arduino board itself.
2- **Configure the Trigger signal**: Generate the required trigger signal using the Arduino Due Board, as introduced in Experiment 1.
3- **Make the connections**: We need to connect the power and signal pins appropriately for the sensor to function as required.
   - Use a breadboard to connect the VCC and GND pins of the sensor to the 3.3V and GND terminals of the Arduino board. Also connect the GND of the Oscilloscope to GND of circuit.
   - Connect the sensor's trigger pin to the trigger signal output pin of the Arduino Due board.
   - Connect the oscilloscope channels: Ch1 to monitor the triggering signal and Ch2 to monitor the echo signal. Adjust the trigger settings to be with Ch1 at a rising edge, move the triggering point to the left of the screen so you can see both the trigger and echo signal next to each other. Adjust the Voltage/Div and Time/Div knops to 2V for both CH1 and CH2 and to 5ms, respectively.
4- **Check the operation**: Move an obstacle in front of the module to check its functionality. You should notice changing in the width of the echo pulse; as the obstacle gets further from the sensor, the pulse echo pulse width increases and vice versa.
   For testing, note that, if there is no obstacle at all, the echo signal should still show a rectangular square wave with around 32.35 msec of HIGH. If you can't see a rectangular waveform, please recheck the connection, and/or inform your TA.
   For testing, note that, only for testing the operation of the ultrasound sensor, and without using the trigger signal from the Arduino Board, you can move the trigger signal of the module to quickly touch the 3.3 V supply from the Arduino. This will supply a one-time trigger signal to the module.
5- **Measure the distance**: With the sensor functioning as expected, it is time to take detailed measurements. Adjust the obstacle distance from the sensor according to Table 2 and measure the pulse width at each distance, using the "Measure" functionality of your Oscilloscope. Compare it to the actual distance and calculate the error. What is the minimum and maximum detectable distances?

Table 2: Experiment 2 Measurements

Actual Distance (cm)	Using Oscilloscope			Arduino Due		
	Pulse width	Measured Distance	Distance Error	Pulse width	Measured Distance	Distance Error
Min						
6						
8						
10						
15						
20						
25						
30						
40						
50						
60						
70						
80						
100						
Max						

Experiment 3: Testing the Ultrasonic Distance Module using Arduino

In this experiment, you should interface the ultrasonic module with the Arduino Due board to provide trigger and to measure the echo. The Arduino board should utilize the ultrasonic module to measure the distance to an obstacle and display it on the serial monitor. In this exercise, we will start using Experiment 1.A to generate the trigger signal and wait to measure the echo pulse.

Required Equipment:

1- Arduino Due board.
2- Ultrasonic sensor module.
3- Breadboard.
4- Jumper Wires.

Procedure for Exp.3.A: Using PulseLn() with trigger using Delay(), Timer Counter and PWM.

1- **Make the connections:**
   - Connect the Vcc and GND pins of the sensor to the 3.3V power and GND pins of the Arduino due board. Connect the Trig pin of the sensor to any digital IO pin in the Arduino, to be called TRIG_PIN in the Arduino sketch. Finally, connect the Echo pin of the sensor to another digital IO pin the Arduino Due board, to be called ECHO_PIN in the Arduino sketch.
2- **Write your code:** Develop an Arduino sketch that does the following in a loop:
   - The Arduino applies the Trigger signal by deriving the TRIG_PIN to HIGH.
   - Wait for 10 usec using delayMicroseconds().

- Turn off the TRIG_PIN by deriving it to LOW.
- Poll the ECHO_PIN using the Arduino function "pulseln()" to measure its pulse width, use `timeout` of 50000 (50 msec). Search online to know its parameters and how it works.
- Calculate the distance using the measured pulse width time according to equation (1).
- The Arduino should send the reading value to the computer using the serial terminal to be displayed on the serial monitor, along with your section # and Group #.
- Wait for 60 msecs using delay() then repeat the loop.

3- **Take measurements**:
   - As done in Experiment 1, move the obstacle according to the distance values in Table 2 and fill in the pulse width, calculate the distance, and calculate the error.

4- Replace the trigger source to used Timer Counter and PWM:
   - If step 2 of this experiment went as expected, replace the trigger source code from using delay functions to either using the Timer Counter unit, or the PWM macrocell Unit.

Procedure for Exp.3.B: Using Timer Unit for both triggering and measuring the Echo.

In this experiment, we will replace the pulseln() function which blocks the CPU from doing any other task to use the on-board Timer in capture mode to capture the time of the echo signal being high. This is very similar to the functionality of pulseln(), however we are going to use the available Timer module instead of the CPU.

- Connect the trigger signal of the Ultrasound Sensor module to pin 2 on the Arduino Due board.
- Connect the Echo signal of the Ultrasound Sensor module to pin A7 on the Arduino Due board.

The code goes as follows:

1- First configure the Timer Counter to generate the Trigger signal, similar to Experiment 1.C
2- Power on the Counter 0, Channel 1.
3- Select the course clock for this counter as the Master Clk / 2 = 84 MHz / 2 = 42 MHz
4- Select which input pin to trigger on, set to start the counter on the rising edge and stop the counter on the falling edge.
5- Finally, reset the counter itself and enable.
6- In the loop function, read the TC_RA register of Counter 0, Channel 1, and do the math to convert the time read into distance in CM, as follow. 42 MHz is the timer input clock, 2 to get time of half of the two-way trip, 100 to convert to CM. 340 m/sec is ~speed of sound in air. Then send it over the serial monitor.

```
/*Sample Code to generate the Trigger signal from Arduino Pin 2.
  And getting the Echo signal to pin A7 in the Arduino.
*/
void setup() {
  Serial.begin(115200); //Enable Serial connection to report the time
  //-----------Setting Registers for the Trigger Signal--------------------
  PMC->PMC_PCER0 |= PMC_PCER0_PID27;//TC0 power ON - Timer Counter 0 channel 0
  PIOB->PIO_PDR |= PIO_PDR_P25;     // The pin is no more driven by GPIO
  PIOB->PIO_ABSR |= PIO_PB25B_TIOA0;// B Assign B25 to alternative periph_B (TIOA0):

  TC0->TC_CHANNEL[0].TC_CMR = TC_CMR_TCCLKS_TIMER_CLOCK1 //MCK/2 = 42 MHz,
                            | TC_CMR_WAVE          //Waveform mode
                            | TC_CMR_WAVSEL_UP_RC      //Count UP mode till RC
                            | TC_CMR_ACPA_CLEAR //Clear TIOA0 on RA compare match
                            | TC_CMR_ACPC_SET;  // Set TIOA0 on RC compare match

  TC0->TC_CHANNEL[0].TC_RC = 2520000-1;  //Set the frequency to 66.667Hz (Period 60 ms)
  TC0->TC_CHANNEL[0].TC_RA = 420-1;  //Set the duty cycle (Pulse of 10 usec)

  TC0->TC_CHANNEL[0].TC_CCR = TC_CCR_SWTRG //Software trigger TC0 channel 0 counter
                           | TC_CCR_CLKEN;//and enable

//-----------Setting Registers for the Echo Signal--------------------
  PMC->PMC_PCER0 |= PMC_PCER0_PID28;     // Timer Counter 0 channel 1 IS TC1, TC1 power ON
  TC0->TC_CHANNEL[1].TC_CMR = TC_CMR_TCCLKS_TIMER_CLOCK1 // capture mode, MCK/2 = 42 MHz
                            | TC_CMR_ABETRG      // TIOA is used as the external trigger
                            | TC_CMR_LDRA_FALLING// load RA on falling edge of TIOA
                            | TC_CMR_ETRGEDG_RISING; // Trigger on rising edge

  TC0->TC_CHANNEL[1].TC_CCR = TC_CCR_SWTRG | TC_CCR_CLKEN; // Reset TC counter and enable
}

void loop() {
  volatile uint32_t CaptureCountA;
  CaptureCountA = TC0->TC_CHANNEL[1].TC_RA;
  printf("L#, Group#: \r %f cm \n", 340.0*CaptureCountA/(42000000.0)/2*100);
  delay(500);
}
```

Note that, using this code, the CPU is not waiting for the Echo signal. The distance in the Echo signal is being read, and recorded by the Timer unit, completely independent from the CPU. We added a delay of 500 msec to allow the TC_RA register to update.

Replace the L# and Group # by your section and group numbers and take a screenshot.

Procedure for Exp.3.B: Enable and use of Interrupt on the Echo signal.

The last trick is to let the falling edge of the sensor Echo signal to itself initiate an interrupt to that the CPU will have a look and read the TC_RA value.

1- Copy the previous code.
2- Define two global variables, before the setup function, to be accessible from the interrupt function as well as the main loop function

```
volatile uint32_t CaptureCountA;
volatile boolean CaptureFlag;
```

3- Add these two lines at the end of the setup function to enable the interrupt.

```
  TC0->TC_CHANNEL[1].TC_IER |= TC_IER_LDRAS; // Trigger interrupt on Load RA
  NVIC_EnableIRQ(TC1_IRQn);                  // Enable TC1 interrupts
```

4- Modify the loop function and add a new interrupt handler function, outside the main loop, as follows.

```
void loop()
{
  if (CaptureFlag) {
    CaptureFlag = 0;        //Reset the flag,
    printf("L#, Group#: \r %f cm \n", 340.0*CaptureCountA/(42000000.0)/2*100);}
}

void TC1_Handler() {
  uint32_t status = TC0->TC_CHANNEL[1].TC_SR; //Read status register, Clear status
  if (status & TC_SR_LDRAS) {                 // If ISR is fired by LDRAS then ....
    CaptureCountA = TC0->TC_CHANNEL[1].TC_RA; //read TC_RA
    CaptureFlag = 1;          //Inform the main loop of an update.
  }
}
```

Note that, using this code, the CPU is, once again, not waiting for the Echo signal. The distance in the Echo signal is being read, and recorded by the Timer unit, completely independent from the CPU. Once new data is read, an interrupt is raised, forcing the CPU to perform the interrupt handling routine. Here, we don't need to add any delay functions in the main loop.

Replace the L# and Group # by your section and group numbers and take a screenshot.

Discussion

1- Draw a figure that plots 2 measured distance curves + 2 error curves (4 in total) against the actual distance.
   - What is the measurement conditions that corresponds to minimum error?
   - For the Arduino measured curve, what do you suggest to compensate the error?

End of Lab 2.