



**COLLEGE OF ENGINEERING
& COMPUTER SCIENCE**
Florida Atlantic University

Car Parking Sensor System

**Department of Electrical Engineering
Introduction to Microprocessor System – CDA 3331C**

Tsz Shing Tsoi

**Supervised by
Dr. Shankar
Douglas Athenosy
Caner Mutlu**

Monday, April 22, 2019

Table of Contents

Abstract.....	2
Project Specifications.....	2
Design Methodology	3
Ping Sensor.....	3
Neopixel LED	4
Servo Motor.....	6
PIC16F18855 Microcontroller.....	7
Implementation	7
Pin Connections	7
MCC Settings.....	8
C Codes.....	9
Results.....	10
Discussion	10
Conclusion	11
Acknowledgements	12
References	13
Appendix Page	14
Listing 1: Main.c.....	14
Listing 2: MCC Settings.....	18
Listing 3: Pin Connections	22
Listing 4: Ultrasonic Ranging Module HC - SR04 Data Sheet	24
Listing 5: Servo Motor SG90 Data Sheet	28

Abstract

Modern automobiles are equipped with various sensors to assist drivers in operating automobiles. Advanced driver assistance features such as collision avoidance system and adaptive cruise control require sensors to detect the surrounding environment. Sensors are also used to detect potential hazardous situations and provide warnings to drivers or even take control of the automobiles to avoid collisions under critical conditions.

This project designed a car parking sensor system to provide warnings and intercept the throttle control when a collision is imminent during parking maneuver, and is implemented using the PIC16F18855 microcontroller. Peripherals including pulse width modulation (PWM), Timer, and Master Synchronous Serial Port (MSSP) are used. A Ping sensor, a Neopixel LED and a servo motor are used as object sensor, warning indication and throttle control simulation respectively, and are interfaced with the microcontroller through the digital input and output (I/O) pins on the MPLAB Xpress Evaluation Board.

Project Specifications

The car parking sensor system uses a Ping sensor to detect distance to objects, a Neopixel LED as warning indication, and a servo motor to simulate throttle control. The servo at the -90-degree position (all the way to the left) simulates full throttle control, and the +90-degree position (all the way to the right) simulates no throttle control and the brake is fully applied. The original project specifications and proposed changes are summarized in Table 1.

Table 1 Original Project Specifications and Proposed Changes

State	Object Distance	Servo Motor Position	Neopixel LED	Proposed Changes
Imminent collision	≤ 10 cm	+90-degree	Red	Change from imminent to possible collisions at object distance > 12 cm
Possible collision	> 10 cm and ≤ 30 cm	Move from -90-degree to +90-degree proportional to the object distance	Flashing yellow at an increasing rate as object distance approaches 10 cm	Change from possible to no possible collisions at object distance > 32 cm At object distance of 30 cm, flashing frequency is ~ 1 Hz; at object distance approaching 10 cm, flashing frequency is ~ 8 Hz; at all other object distances, flashing frequency is interpolated
No possible collision	> 30 cm	-90-degree	Green	None

While the state of the car parking sensor system will change from possible to imminent collisions at 10 cm as required in the original specification, the system is proposed to remain at the imminent collision state until the object distance becomes greater than 12 cm. This proposed change will prevent oscillation between the two states with minor fluctuations in the distance measurement at around 10 cm. Similar change is proposed between possible and no possible collisions. Nevertheless, these proposed changes will generate the required or more severe warning indications than in the original specifications at each object distance. In addition, the frequencies of flashing yellow summarized in Table 1 are proposed since the original specifications did not indicate the required flashing frequencies.

Design Methodology

There are four main components used in this project: Ping sensor, Neopixel LED, servo motor, and the MPLAB Xpress Evaluation Board using the PIC16F18855 microcontroller. Ping sensor is used to measure object distance. The microcontroller then interprets the distance measurement, determines the output states, and calculates the parameters to be used to control the Neopixel LED and servo motor. Each of these components is discussed in more details below.

Ping Sensor

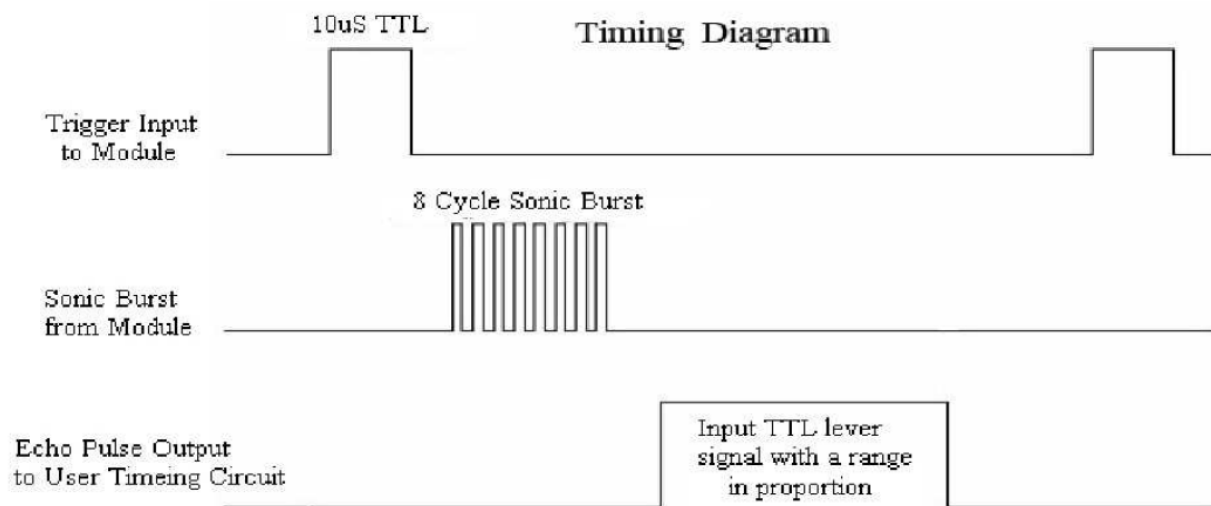
The Ping sensor used in this project is the ultrasonic ranging module HC - SR04. This module has a 4-pin design including the 5V power supply (Vcc), trigger pulse input (Trig), echo pulse output (Echo), and 0V ground (GND) as shown in Figure 1. It provides a range between 2 cm and 4 m with an accuracy of approximately 3 mm, which meets the requirements in the specifications.

Figure 1 Ping Sensor Pin-outs



Per the data sheet, the measurement begins by supplying a short 10 μ s pulse to the Trig input. However, it is found that a 5 μ s pulse is sufficient as the trigger signal. The module then automatically sends out eight 40 kHz ultrasonic pulses at a 15-degree angle and raises its Echo output to HIGH. The Echo output will remain HIGH until after the returning pulse signals are received. The working frequency of this module is 40 Hz or once every 25 ms. However, the data sheet suggested using a measurement cycle of over 60 ms to prevent the trigger signal overlapping with the echo signal. The timing diagram in Figure 2 illustrates this operation. A copy of the data sheet is included in Appendix Listing 4.

Figure 2 Ping Sensor Timing Diagram



Source: ElecFreaks Ultrasonic Ranging Module HC - SR04 Data Sheet

The object distance is then calculated as:

$$\text{Distance} = (\text{High Level Time} \times \text{Velocity of Sound}) / 2$$

where the velocity of sound is approximately 343 m/s at 20-degree Celsius.

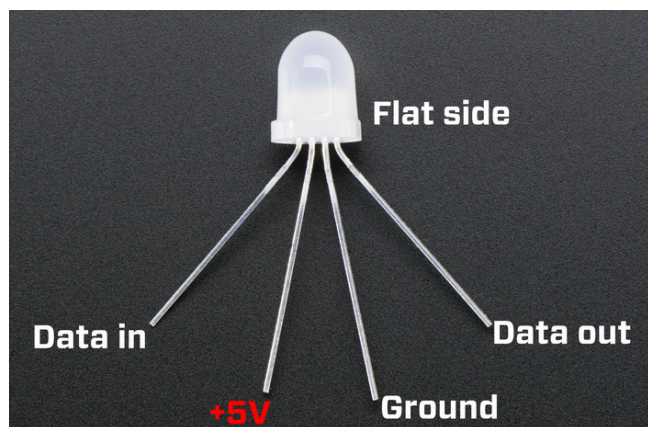
The high level time is measured in the microcontroller using Timer Gate (explained later in this section), and is calculated as:

$$\text{High Level Time} = \text{Timer Count} \times \text{Clock Period}$$

Neopixel LED

Neopixel packs 3 LEDs (Red, Green and Blue) and a small digital logic to control the color with 24-bit resolution. Neopixel LED has a 4-pin design including data input, +5V power, ground and data output as shown in Figure 3. Data is sent to the Neopixel LED via the data input, and the data output is used for daisy-chaining additional Neopixel LEDs. Since the microcontroller uses 3.3V to drive the data input, a 3.3V power supply is used instead of 5V.

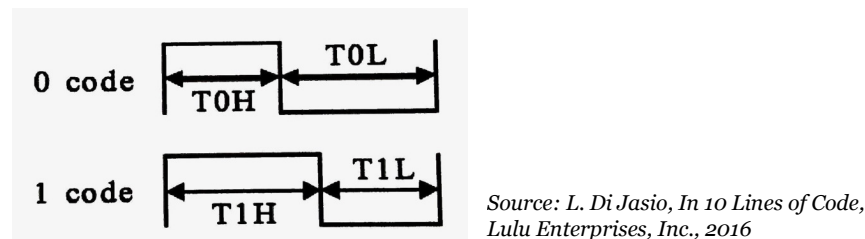
Figure 3 Neopixel LED Pin-outs



Source: Dr Shankar's Github Site
(<https://github.com/RShankar/Intro-to-Microprocessors/tree/master/Lab%20Project%20Examples/NeoPixel%20LED>)

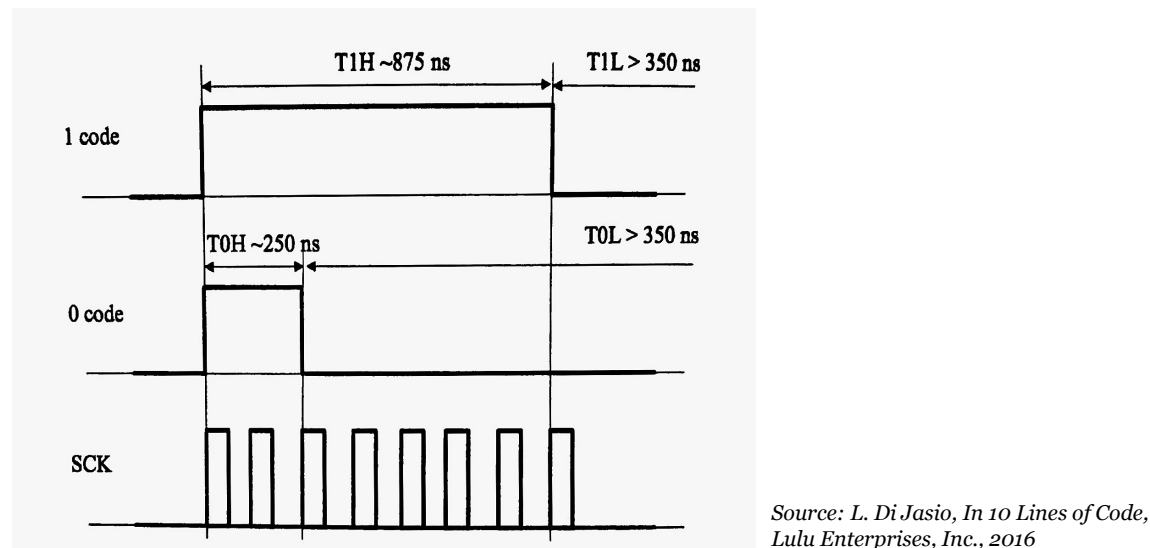
The color is controlled by a 24-bit bitstream with a bit length of 1.25 μ s, which is equivalent to a bit rate of 800 Kbit/s. The 24-bit bitstream is composed of 8 bits of Red, followed by 8 bit of Green, and then 8 bits of Blue information. Each group of 8-bit represents the intensity of the respective color, ranging between 0 and 255. A bit of 0 (0 code) is encoded as a pulse (HIGH) of 350 ns followed by a pause (LOW) of 900 ns, and a bit of 1 (1 code) is encoded as a pulse (HIGH) of 900 ns followed by a pause (LOW) of 350 ns, as illustrated in Figure 4. Each bit is shifted from the microcontroller through the data input and output lines, and the shifting process continues until a pause (LOW) of 50 μ s or longer is encountered in the bitstream. At that point, the Neopixel will update its color based on the last 24 bits of data remained in its shift register.

Figure 4 Neopixel Bit Encoding



Due to the short and precise timing requirements of each bit, each bit could be represented by a byte sent through the Serial Peripheral Interface (SPI) in the MSSP module of the PIC16F18855 microcontroller, as suggested by L. Di Jasio in *In 10 Lines of Code*. By using a clock source of 8 MHz or period of 125 ns, the 0 code can be represented by sending a byte of 0xC0 or 0b11000000 in binary from the SPI, resulting in a pulse (HIGH) of 250 ns followed by a pause (LOW) of 750 ns; and the 1 code can be represented by sending a byte of 0xFE or 0b11111110 in binary from the SPI, resulting in a pulse (HIGH) of 875 ns followed by a pause (LOW) of 125 ns. Additional pause (LOW) after each byte is sent will be naturally occurring resulting in an overall code duration of approximately 1.25 μ s required by the Neopixel LED. Figure 5 illustrates the Neopixel bit encoding using the SPI.

Figure 5 Neopixel Bit Encoding using Serial Peripheral Interface



Servo Motor

The servo motor used in this project is a SG90 servo motor with 3 pins, including a control pin (PWM), 5V power (Vcc) and ground, as shown in Figure 6.

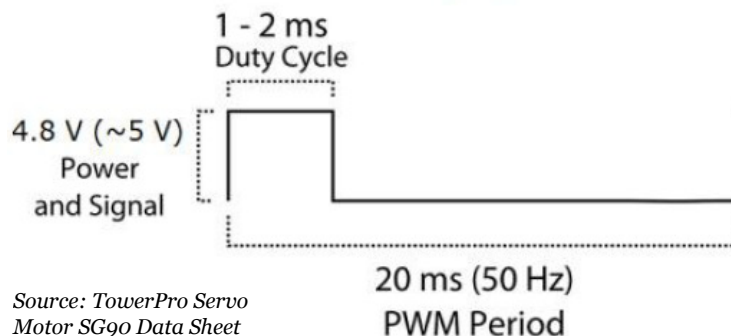
Figure 6 Servo Motor Pin-outs

Source: TowerPro Servo Motor SG90 Data Sheet



The servo motor can rotate approximately 180 degrees and the position is determined by pulse widths generated from the PWM peripheral of the microcontroller. Per the data sheet, the -90-degree position (all the way to the left) is represented by a pulse width of ~1 ms, the 0-degree position (middle) is represented by a pulse width of ~1.5 ms, and the +90-degree position (all the way to the right) is represented by a pulse width of ~2 ms. A PWM period of approximately 20 ms is typically used in the control signal. A typical PWM control signal is illustrated in Figure 7, and a copy of the data sheet is included in Appendix Listing 5.

Figure 7 Motor Pulse Width Modulation Signal



Source: TowerPro Servo Motor SG90 Data Sheet

In practice, the -90-degree position is achieved with a pulse width of ~0.4 ms, and the +90-degree position is achieved with a pulse width of ~2.6 ms. Also, the PWM period has a large degree of flexibility and a PWM period as low as 4 ms could be used to control the servo motor provided the required pulse widths are maintained.

PIC16F18855 Microcontroller

Several peripherals in the PIC16F18855 microcontroller are used in this project, and some of which were mentioned previously. Table 2 summarizes all the peripherals used in each component of the project.

Table 2 Peripherals Used in the PIC16F18855 Microcontroller

Component	Peripherals Used
Ping Sensor	<ul style="list-style-type: none">• Timer1 (Timer Gate)
Neopixel LED	<ul style="list-style-type: none">• MSSP1 (SPI Master)• Timer4 + CLC1 (for flashing operation)
Servo Motor	<ul style="list-style-type: none">• Timer2 + PWM6

The Timer Gate Single-Pulse mode of the 16-bit Timer1 is used to measure the duration of the Echo signal. The Echo signal is used as the signal source for Timer1 Gate, and Timer1 counter will increase by 1 at each rising edge of the clock as long as the Echo signal is HIGH. For example, a clock source of 2 MHz or period of 0.5 μ s and a Timer1 counter value of 10,000 correspond to an Echo pulse width of 5 ms or a distance of approximately 86 cm. With a clock source of 2 MHz and the maximum Timer counter value of 65,535, the maximum measured distance is approximately 5.6m, which is sufficient for measuring all distances within the range of the Ping sensor used in this project.

As discussed previously, the MSSP module is used to control the color of the Neopixel LED. The varying frequencies of flashing yellow operation could be achieved by using Timer4 and Configurable Logic Cell (CLC1) with J-K flip flop to toggle between Yellow and OFF at varying frequencies. The flashing frequency is changed by updating the Timer4 period based on the object distance.

The servo motor is controlled by PWM which requires the use of a Timer. The position of the servo motor is changed by updating the duty cycle of the PWM module based on the object distance and the corresponding desired pulse width. The duty cycle is calculated as:

$$\text{Duty Cycle} = (\text{Desired Pulse Width} / \text{PWM Period}) \times \text{Maximum Duty Cycle}$$

Implementation

This section documents project implementation including the pin connections, MPLAB Code Configurator (MCC) settings and C Code.

Pin Connections

The MPLAB Xpress Evaluation Board provides accessibility to various I/O pins in the PIC16F18855 microcontroller. Table 3 summarizes the pin connections of each component used in this project. All pins are used as digital pins.

Table 3 Pin Connections in the PIC16F18855 Microcontroller

Component	Pin-out	Pin on Microcontroller	Peripheral I/O
Ping Sensor	Vcc	5V	N/A
	Trig	RC6	GPIO output
	Echo	RC5	Timer1 T1G input
	GND	GND	N/A
Neopixel LED	Data in	RB5	MSSP1 SDO1 output
	Power	3.3V	N/A
	Ground	GND	N/A
	Data out	Not connected	N/A
Servo Motor	PWM	RC7	PWM6 output
	Vcc	5V	N/A
	Ground	GND	N/A

Appendix Listing 3 includes photos illustrating the pin connections of each component on the MPLAB Xpress Evaluation Board. Since three Ground connections and two 5V connections are required but only two Ground connections and one 5V connection are available through the socket connectors, the Vcc and Ground connections for the servo motor are loosely connected through the Xpress Evaluation Board's input/output connections.

MCC Settings

A system clock frequency of 32 MHz is used in the microcontroller, which is needed to provide a clock frequency of 8 MHz ($F_{OSC}/4$) for the MSSP module. Table 4 summarizes the key peripheral settings. Various functions for these peripheral are assigned to pins per Table 3.

Table 4 Key Peripheral Settings

Component	Peripheral	Clock Frequency	Key Settings
Ping Sensor	Timer1	$F_{OSC}/4$ with prescaler of 1:4 = 2 MHz	<ul style="list-style-type: none"> • Enable Gate Single-Pulse mode • Gate Signal Source = T1G_pin • Gate Polarity = HIGH
Neopixel LED	MSSP1	$F_{OSC}/4 = 8$ MHz	<ul style="list-style-type: none"> • SPI Master Mode • Clock Polarity = active HIGH • Clock Edge = rising
	Timer4	LFINTOSC with prescaler of 1:64 = 484.375 Hz	<ul style="list-style-type: none"> • Control mode = roll over pulse (free running period mode)
	CLC1	Varies (Timer4 = PR4)	<ul style="list-style-type: none"> • Mode = J-K flip flop with R • J and K inputs are HIGH (toggle mode) • Clock source is Timer4 = PR4
Servo Motor	Timer2	$F_{OSC}/4$ with prescaler of 1:128 = 62.5 kHz	<ul style="list-style-type: none"> • Timer Period = 4.096 ms • Control mode = roll over pulse (free running period mode)
	PWM6	244.14 Hz (Timer2 = PR2)	<ul style="list-style-type: none"> • PWM Polarity = active HIGH

Appendix Listing 2 includes screen captures of various MCC settings. Pin assignments for several functions such as CLC1's CLCINO and MSSP1's SCK1 and SDI1 are required but these functions are not used in this project. Therefore, these functions are assigned to any unused pin with no particular preference. Also, any reasonable custom pin name could be used in the Pin Module setting.

C Codes

C codes in the microcontroller were developed to integrate the peripherals and all external components to implement the car parking sensor system. The C codes are implemented to read the object distance and then calculate the appropriate output settings for the Neopixel LED and servo motor.

The duty cycles corresponding to the -90-degree position and +90-degree position of the servo motor are pre-calculated as follows:

$$\text{Duty Cycle} = (\text{Desired Pulse Width} / \text{PWM Period}) \times \text{Maximum Duty Cycle}$$

$$\text{DC_LOW} = (\sim 0.4 \text{ ms} / 4.096 \text{ ms}) \times 1,023 \approx 95$$

$$\text{DC_HIGH} = (\sim 2.6 \text{ ms} / 4.096 \text{ ms}) \times 1,023 \approx 645$$

Two functions discussed below are implemented in the C codes for various tasks executed in the main function.

- `float getDistance()` – This function sends a 5us pulse to the Trig output to trigger the Ping sensor measurement. It then resets Timer1 to zero and increments Timer1 for the duration of the Echo input, and returns the measured distance in cm. The measured distance is calculated as:

$$\begin{aligned} \text{Distance} &= (\text{Timer Count} \times \text{Clock Period} \times \text{Velocity of Sound}) / 2 \\ &= \text{Timer Count} \times (5 \text{ us} \times 343 \text{ m/s}) / 2 \\ &\approx \text{Timer Count} / 116 \end{aligned}$$

- `void NeoPixel_Stream(uint8_t *p, uint8_t count)` – This function constructs and sends a 24-bit bitstream using the SPI to set the color of the Neopixel LED. It reads the 8-bit representation of each color of Red followed by Green and Blue, and processes the data bit-by-bit. It then sends a byte of 0xFE or 0b11111110 in binary through the SPI if the bit is 1; otherwise a byte of 0xC0 or 0b11000000 in binary is sent.

In the main function, the following logic and calculations are implemented:

- If object distance is within 10 cm, the Neopixel LED color is set to Red and the PWM6 duty cycle to DC_HIGH. This is repeated until object distance is greater than 12 cm.
- If object distance is greater than 30 cm, the Neopixel LED color is set to Green and the PWM6 duty cycle to DC_LOW.
- If object distance is greater than 10 cm but within 30 cm, the Timer4 period value is first interpolated linearly between 31 and 255 based on the object distance, which correspond to the Timer periods of approximately 64 ms and 526 ms respectively, and the CLC1 toggling frequencies of approximately 8 Hz and 1 Hz respectively. The Timer4 period value is calculated as:

$$\text{T4PR} = (\text{Distance} - 10) \times 224 / 20 + 31$$

The Neopixel LED color is then set to Yellow when the CLC1 output is LOW and OFF when the CLC1 output is HIGH. The PWM6 duty cycle is similarly interpolated between DC_LOW and DC_HIGH based on the object distance, and is calculated as:

$$\text{Duty Cycle} = (30 - \text{Distance}) \times (\text{DC_HIGH} - \text{DC_LOW}) / 20 + \text{DC_LOW}$$

This is repeated until the object distance is within 10 cm or greater than 32 cm.

The C codes from Main.c are included in Appendix Listing 1.

Results

The project was implemented and tested with an object, such as a textbook, placed in front of the Ping sensor. The object was moved gradually from a distance greater than 30 cm to a distance less than 10 cm to simulate car parking maneuver. This process was then reversed and repeated several times, and the Neopixel LED and servo motor position were visually inspected for compliance with the specifications. The transition between various states were also tested several times and inspected visually, and changes to the C codes were made as appropriate based on the observations. The final implementation fully complies with the specifications, and no glitches were observed during state transition and the servo movement was smooth.

A video illustrating the operation of the car parking sensor system can be found in this link <https://youtu.be/oC9GonOZldQ>.

Discussion

This project was designed and implemented largely based on the book *In 10 Lines of Code* by L. Di Jasio as well as previous homework and lab exercises in this class. It integrates various components with specific calculations to achieve the desired outcomes. Several lessons learned were noted during the project development process, and are summarized below.

- If the color of the Neopixel LED is updated too frequently, such as less than 5 ms between updates, the Neopixel LED would fail to neither respond nor display the color properly.
- Care must be taken to calculate the Timer period and PWM duty cycle. The calculated values should be within the allowable ranges, i.e. 0 to 255 for Timer4 period and 0 to 1,023 for PWM6, given all possible values of the object distance. The calculated values exceeded these allowable ranges in earlier versions of the project implementation, and resulted in glitches and unexpected behavior under certain circumstances.
- An earlier version of the project implementation attempted to directly use the Timer1 counter value in the calculations to achieve the same resolutions in Neopixel flashing frequency and servo movement using integer arithmetic instead of real number arithmetic. However, the Timer1 counter value is too large that the interim values calculated may have resulted in integer overflow, resulting in improper system operations.
- Since the MPLAB Xpress Evaluation Board does not have in-circuit debugger, the Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART) was used to display values of local variables in CoolTerm for debugging purposes.

Conclusion

This project demonstrates that a car parking sensor system could be implemented with off-the-shelf components and a mid-range microcontroller. Although this project is specific to a car parking sensor system implemented with a Ping sensor, Neopixel LED and servo motor, it simply reads and interprets inputs from the sensor, calculates the desired output states, and sets the appropriate parameters to each connected output devices. This concept could be expanded to other types of systems using different types of sensors and devices.

Acknowledgements

I would like to personally thank Dr. Shankar, Mr. Douglas Athenosy and Mr. Caner Mutlu for developing this design project and their teaching and assistance throughout this CDA 3331C course. I would also like to thank Mr. Perry Weinthal for preparing all the project components used throughout this class.

References

- [1] L. Di Jasio, *In 10 Lines of Code*, Lulu Enterprises, Inc., 2016.
- [2] R. Shankar, D. Athenosy, C., *Course materials in CDA 3331C*, Florida Atlantic University, 2019.
- [3] Microchip Technology Inc., *PIC16(L)F18855/75 Full-Featured 28/40/44-Pin Microcontrollers Data Sheet*, 2017.

Appendix Page

Listing 1: Main.c

```
/*
 * Made by Tsz Shing Tsoi
 * For FAU CDA3331C - Design Project: Car Parking Sensor System
 * Date: 04-21-2019
 *
 * This programs implements the car parking sensor system on
PIC16F18855 microcontroller
 * connected with a HC-SR04 Ping sensor, a NeoPixel LED, and a
SG90 servo motor.
 *
 * Pin connections for 4-pin Ping HC-SR04 sensor:
 * Vcc=5V
 * Echo=RC5
 * Trig=RC6
 * Ground = GND
 *
 * Pin connections for NeoPixel LED:
 * Data in = RB5
 * Vcc = 3.3V
 * Ground = GND
 * Data out is not connected
 *
 * Pin connections for SG90 servo motor:
 * Control pin = RC7
 * Vcc = 5V
 * Ground = GND
 */

#include "mcc_generated_files/mcc.h"
#define RED 0 //index of Red color in the color[] array
#define GREEN 3 //index of Green color in the color[] array
#define YELLOW 6 //index of Yellow color in the color[] array
#define DC_LOW 95 //duty cycle corresponding to the -90-degree
// position
#define DC_HIGH 645 //duty cycle corresponding to the
// +90-degree position
#define DELAY 60 //delay in ms between updating object distance
// and Neopixel LED color
// and servo motor position

/*
                                     Main application
 */
```

```

/**
 * Trigger Ping sensor and read Echo duration, and return the
 * object distance in cm.
 * @return
 * Return object distance in cm.
 */
float getDistance() {
    TRIG_SetHigh(); // sends trigger pulse
    __delay_us(5);
    TRIG_SetLow();
    TMR1GIF = 0; //resets timer
    TMR1_WriteTimer(0);
    TMR1_StartSinglePulseAcquisition();
    while (!TMR1GIF); // waits for return pulse to end
    return TMR1_ReadTimer() / 116.0; //math for cm
}

/**
 * Send the 24-bit bitstream representing RGB color through the
 * SPI module for Neopixel LED.
 * @param p
 * Pointer to an array containing values for RGB. Length of the
 * array should be multiples of 3.
 * @param count
 * Number of Neopixel LED colors to be sent through the SPI
 * module.
 */
void NeoPixel_Stream(uint8_t *p, uint8_t count) {
    // sends count x RGB data packets (24-bit each)
    uint8_t bitCount, data;
    while (count--) {
        bitCount = 24;
        do {
            if ((bitCount & 7) == 0)
                data = *p++;
            SSP1BUF = ((data & 0x80) ? 0xFE : 0xC0);
            // WS2812B 900ns - 350ns
            data <<= 1;
        } while (--bitCount);
    }
}

/**
 * Return smaller of the two input values.
 * @param val
 * Input value 1 for comparison.
 * @param otherVal

```

```

    * Input value 2 for comparison.
    * @return
    * Smaller of val and otherVal.
    */
inline float min(float val, float otherVal) {
    return (val <= otherVal) ? val : otherVal;
}

/**
 * Return larger of the two input values.
 * @param val
 * Input value 1 for comparison.
 * @param otherVal
 * Input value 2 for comparison.
 * @return
 * Larger of val and otherVal.
 */
inline float max(float val, float otherVal) {
    return (val >= otherVal) ? val : otherVal;
}

void main(void) {
    // initialize the device
    SYSTEM_Initialize();
    TRIG_SetLow(); //set trigger low to being pulse

    float distance = getDistance(); //object distance in cm
    uint8_t color[] = {0xFF, 0, 0, 0, 0xFF, 0, 0xFF, 0xFF, 0, 0,
0, 0}; //color matrix: Red, Green, Yellow and OFF

    while (1) {
        if (distance <= 10) { //imminent collision
            do {
                NeoPixel_Stream(&color[RED], 1);
                PWM6_LoadDutyValue(DC_HIGH);
                __delay_ms(DELAY);
                distance = getDistance();
            } while (distance <= 12); //change from red to
                // yellow at 12cm
        } else if (distance > 30) { //no possible collision
            NeoPixel_Stream(&color[GREEN], 1);
            PWM6_LoadDutyValue(DC_LOW);
            __delay_ms(DELAY);
            distance = getDistance();
        } else { // 10 < distance <=30, possible collision
            do {
                //limit the distance to be between 10 and 30

```

```

        distance = max(min(distance, 30), 10);
        //interpolate between T4PR values of 0 and 255
        //resulting toggling frequency is ~1Hz at 30cm,
        //and ~8Hz at 11cm
        T4PR = (uint8_t)((distance - 10) * 224) / 20 +
        31);
        //output Yellow if toggle OFF,
        //otherwise Blank if toggle ON
        NeoPixel_Stream(&color[YELLOW +
        CLC1CONbits.LC1OUT * 3], 1);
        //interpolate duty value between DC_LOW and
        // DC_HIGH
        PWM6_LoadDutyValue((uint16_t)((30 - distance)*
        (DC_HIGH - DC_LOW) / 20 + DC_LOW));
        __delay_ms(DELAY);
        distance = getDistance();
    } while (distance > 10 && distance <= 32); //change
    // from yellow to red at 10cm, yellow to green at 32cm
}

}

/**

End of File

*/

```

Listing 2: MCC Settings

Pin Module

Pin Module									
<div> <div>Easy Setup</div> <div>Registers</div> </div> <div>Selected Package : UQFN28</div>									
Pin Name	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RA0	CLC1	CLCIN0		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RB5	MSSP1	SDO1	SDO1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RC3	MSSP1	SCK1	SCK1	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RC4	MSSP1	SDI1	SDI1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RC5	TMR1	T1G	T1G	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RC6	Pin Module	GPIO	TRIG	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none
RC7	PWM6	PWM6OUT		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	none

Pin Manager

Pin Manager: Grid View																														
Package:	UQFN28		Pin No:		27	28	1	2	3	4	7	6	18	19	20	21	22	23	24	25	8	9	10	11	12	13	14	15	26	
			Port A ▼								Port B ▼								Port C ▼								E			
Module	Function	Direction	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	3			
CLC1	CLC1OUT	output																												
CLCx ▼	CLCIN0	input																												
	CLCIN1	input																												
	CLCIN2	input																												
	CLCIN3	input																												
MSSP1 ▼	SCK1	in/out																												
	SDI1	input																												
	SDO1	output																												
OSC	CLKOUT	output																												
PWM6	PWM6OUT	output																												
Pin Module ▼	GPIO	input																												
	GPIO	output																												
RESET	MCLR	input																												
TMR1 ▼	T1CKI	input																												
	T1G	input																												
TMR2	T2IN	input																												
TMR4	T4IN	input																												

Timer1

TMR1

Easy Setup

Registers

Hardware Settings

☒ Enable Timer

Timer Clock

Clock Source

FOSC/4

External Frequency

32.768 kHz

Prescaler

1:4

☒ Enable Synchronization

Timer Period

Timer Period

500 ns ≤ 32.768 ms ≤ 32.768 ms

Period count

0x0 ≤ 0x0 ≤ 0xFFFF

Calculated Period

32.768 ms

☐ Enable 16-bit read

☒ Enable Gate

☐ Enable Gate Toggle

Gate Signal Source

T1G_pin

☒ Enable Gate Single-Pulse mode

Gate Polarity

high

☐ Enable Timer Interrupt

☐ Enable Timer Gate Interrupt

Software Settings

Callback Function Rate

0

 x Time Period = 0 s

MSSP1

MSSP1

Easy Setup

Registers

Hardware Settings

Mode

SPI Master

☒ Enable MSSP

Input Data Sampled at

Middle

SPI Mode

Clock Polarity

Idle:Low, Active:High

Clock Edge

Idle to Active

SPI Mode

1

SPI Clock

Clock Source

FOSC/4

Clock Divider

0x00 ≤ 0x0 ≤ 0xFF



SPI Clock

8000.0 kHz

19

Timer4

TMR4

 Easy Setup  Registers

Hardware Settings

☒ Enable Timer

Timer Clock

Clock Source LFINTOSC

Clock Frequency 32.768 kHz

Postscaler 1:1

Prescaler 1:64

Polarity Rising Edge

☐ Enable Prescaler O/P Sync

☐ Enable Clock Sync

Timer Period

Timer Period 2.064516 ms ≤ ≤ 528.516129 ms

Actual Period 528.516129 ms (Period calculated via PR Register value)

Ext Reset Source T4CKIPPS pin

Control Mode Roll over pulse



Start/Reset Option Software control

☐ Enable Timer Interrupt

Software SettingsCallback Function Rate 0x0 x Time Period = 0.0 ns

CLC1

CLC1

 Easy Setup  Registers

Hardware Settings

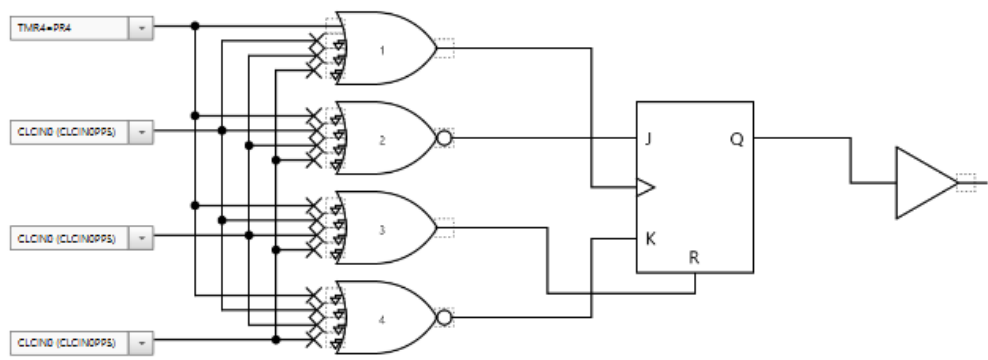
☒ Enable CLC

☐ Enable CLC Interrupt

☐ Enable Rising Interrupt ☐ Enable Falling Interrupt



Export CLC image

Mode JK flip-flop with R



Timer2

TMR2

 Easy Setup  Registers

Hardware Settings

☒ Enable Timer

Timer Clock

Clock Source FOSC/4

Clock Frequency 32.768 kHz

Postscaler 1:1

Prescaler 1:128

Polarity Rising Edge

☐ Enable Prescaler O/P Sync

☐ Enable Clock Sync

Timer Period

Timer Period 16 us ≤ 4.096 ms ≤ 4.096 ms

Actual Period 4.096 ms (Period calculated via PR Register value)

Ext Reset Source T2CKIPPS pin

Control Mode Roll over pulse

Start/Reset Option Software control



☐ Enable Timer Interrupt

Software Settings

Callback Function Rate 0x0 x Time Period = 0.0 ns

PWM6

PWM6

 Easy Setup  Registers

Hardware Settings

☒ Enable PWM

Select a Timer : Timer2

Duty Cycle

Duty Cycle 100.0 %

PWMDC Value 1023

PWM Parameters

PWM Polarity active_hi

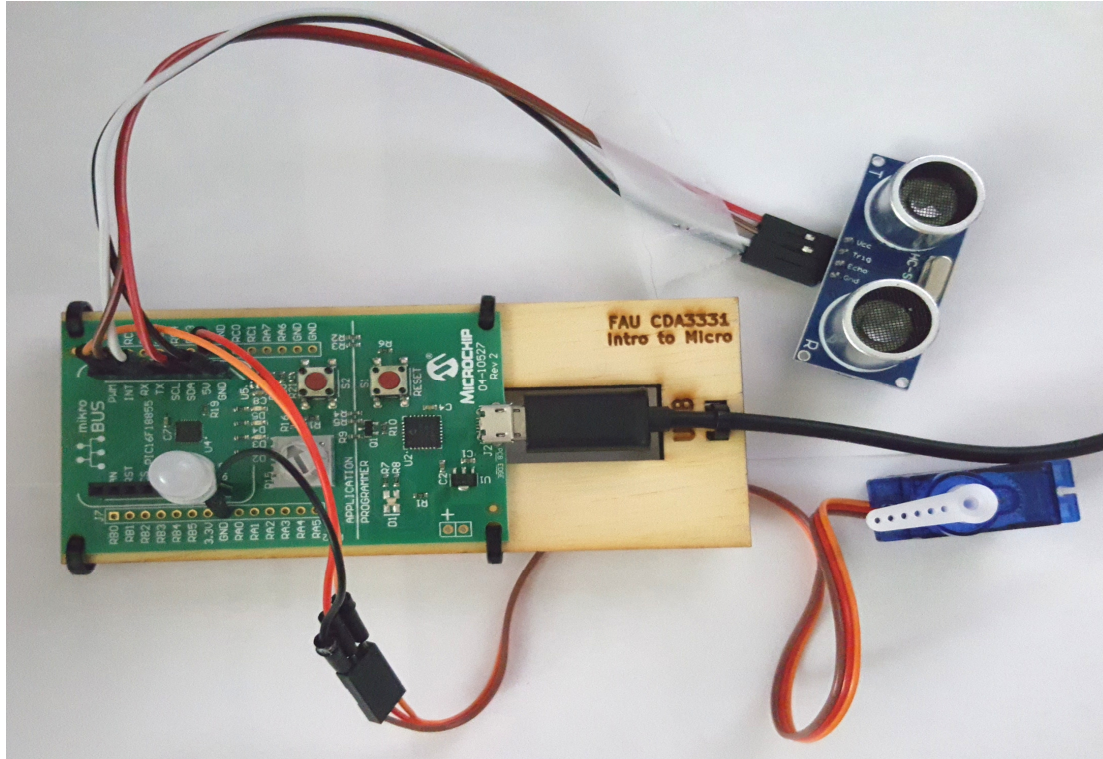
PWM Period 4.096 ms

PWM Frequency 244.14 Hz

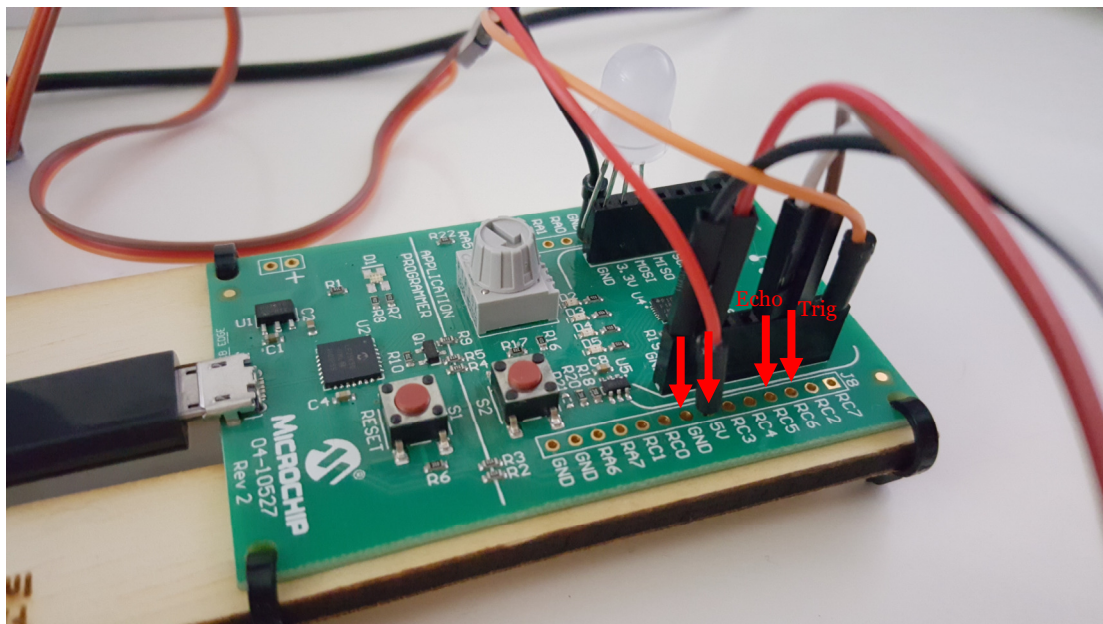
PWM Resolution 10 Bits

Listing 3: Pin Connections

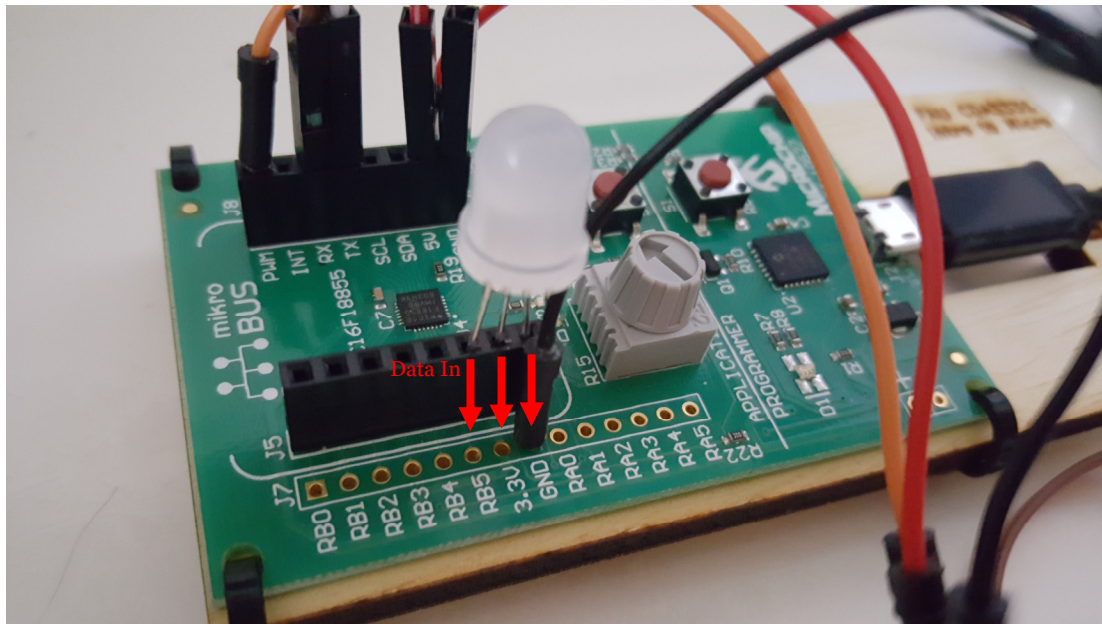
Overall Wiring



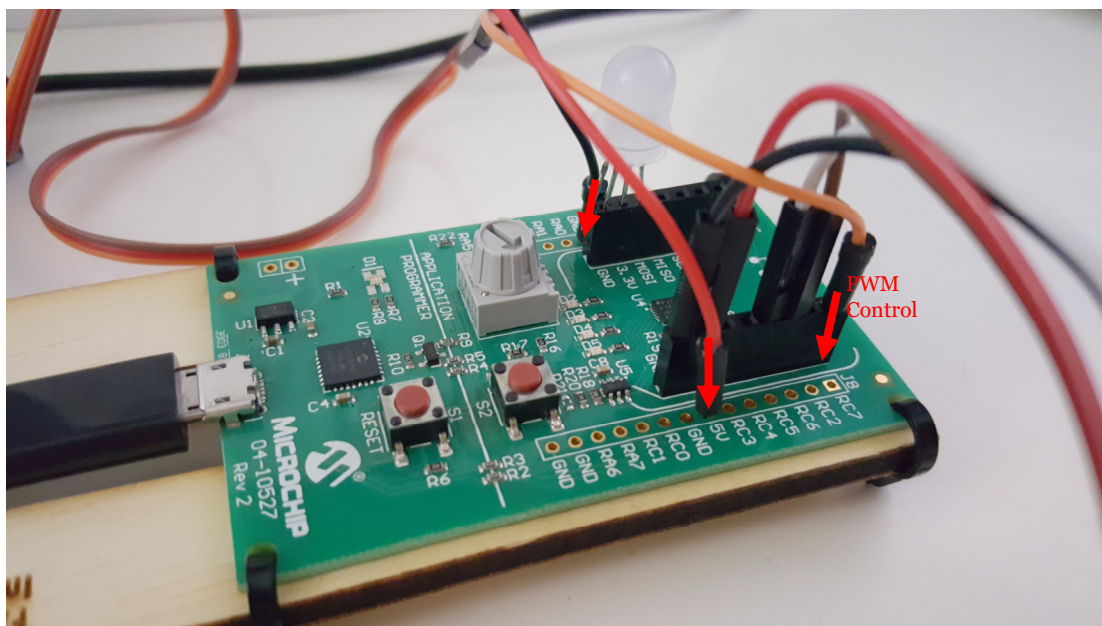
Pin Placement for Ping Sensor



Pin Placement for Neopixel LED



Pin Placement for Servo Motor



Listing 4: Ultrasonic Ranging Module HC - SR04 Data Sheet



Tech Support: services@elecfreaks.com

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

- (1) Using IO trigger for at least 10us high level signal,
- (2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
- (3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.

Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

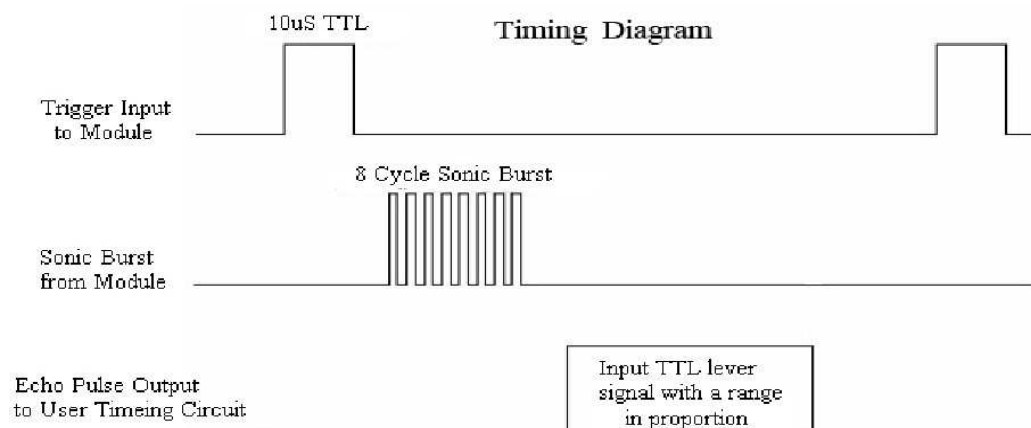
Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm



Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion. You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: $\mu\text{S} / 58 = \text{centimeters}$ or $\mu\text{S} / 148 = \text{inch}$; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



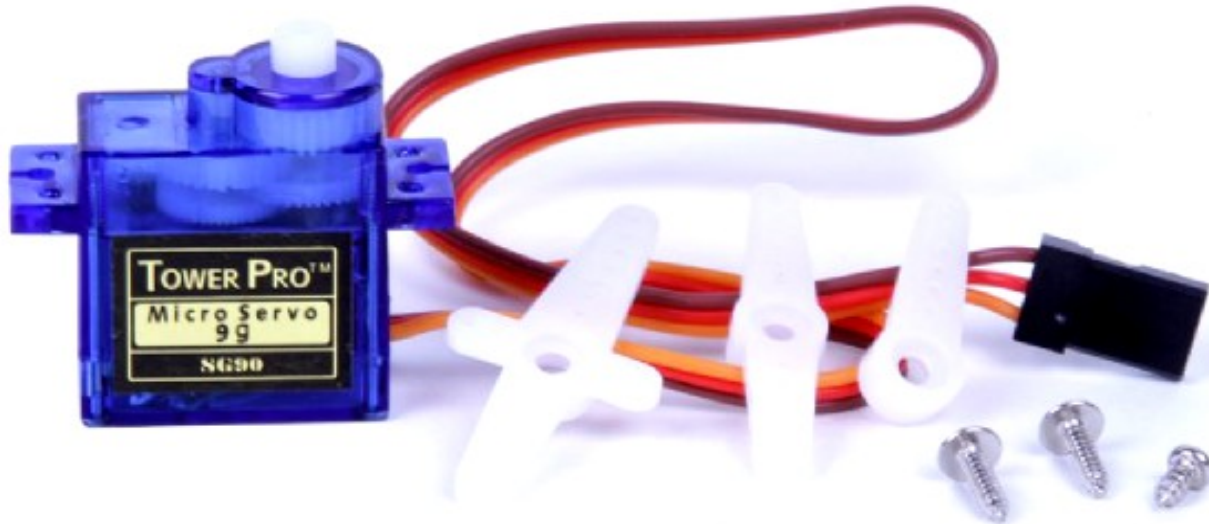
Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

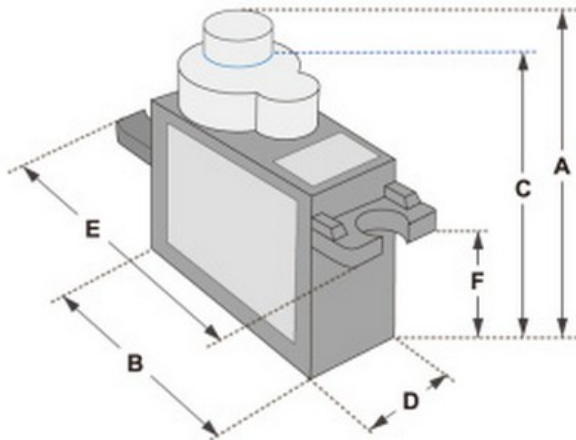
www.ElecFreaks.com



Listing 5: Servo Motor SG90 Data Sheet



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.



Dimensions & Specifications	
A (mm) :	32
B (mm) :	23
C (mm) :	28.5
D (mm) :	12
E (mm) :	32
F (mm) :	19.5
Speed (sec) :	0.1
Torque (kg-cm) :	2.5
Weight (g) :	14.7
Voltage :	4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

PWM=Orange (⏏)
Vcc=Red (+)
Ground=Brown (-)

