Florida Atlantic University
Department of Electrical Engineering
Introduction to Microprocessor Systems – CDA 3331C

Car Parking Sensor System

Submitted by:
Joshua Cidoni-Walker
jcidoniwalke2017@fau.edu

Friday, April 26, 2019

Supervised by
Dr. Ravi Shankar
Douglas Athenosy
Caner Mutlu

Disclaimer

The *Car Parking Sensor System* detailed in this report is neither designed nor intended to be used on any motor vehicle accessing public and/or private roadways.

Table of Contents

Acknowledgements

I would like to express my upmost appreciation to all those who have provided me with the opportunity to complete this report. My most profound gratitude is given to Dr. Shankar, who has provided endless assistance, reassurance, and encouragement throughout the semester.

I would also like to mention the two teaching assistants, Douglas Athenosy and Caner Mutlu; they have both indirectly and directly aided me in becoming a more well-rounded Computer Scientist.

Abstract

As a result of increased vehicular and pedestrian traffic [1], new technologies have been integrated into modern-day vehicles in an effort to provide safer travel. Among these technologies are *Car Parking Sensor Systems*. These systems are designed to alert drivers (through various means) of obstacles within their path. The aim of this report is to detail the development and implementation of a mock (yet functional) *Car Parking Sensor System*.

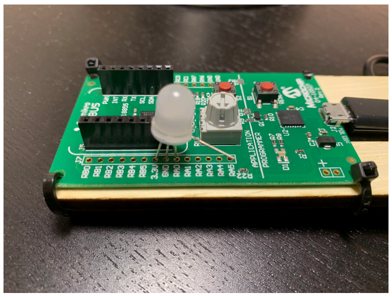
Methodology

The *Car Parking Sensor System* detailed in this report will be developed and implemented on/using the following components:

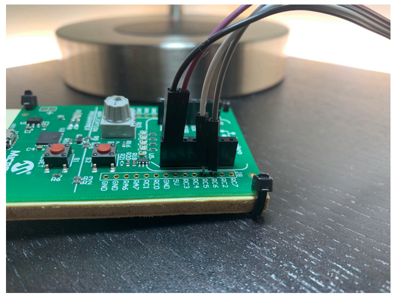
- **Microchip PIC16F18855 Microcontroller** (used to coordinate computation and communication with the several peripheral components)
- **TowerPro SG90 Digital Servomotor** (used to simulate a vehicular breaking mechanism, which will throttle based on an objects distance from the ultrasonic sensor)
- **HCSR04 Ultrasonic Distance Sensor** (used to fetch the distance of objects within its line-of-sight)
- **Adafruit WS2812B RGB "Neopixel"** (used to provide a visual indication of an objects distance from the sensor)

Implementation

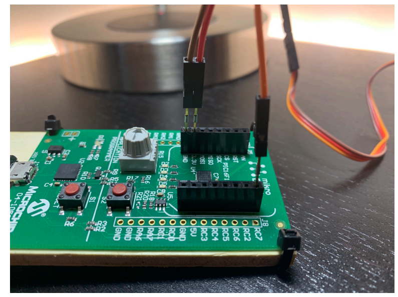
As the *Car Parking Sensor System* detailed in this report uses a microcontroller with external peripherals, we begin by connecting the external peripherals to the microcontroller.



Connect the "**Neopixel**" to "RB5", "3.3V", and "GND". Note: Identify the GND pin (on the "Neopixel") by identifying the side of the "Neopixel" that is flat; the pin on the same side is GND.

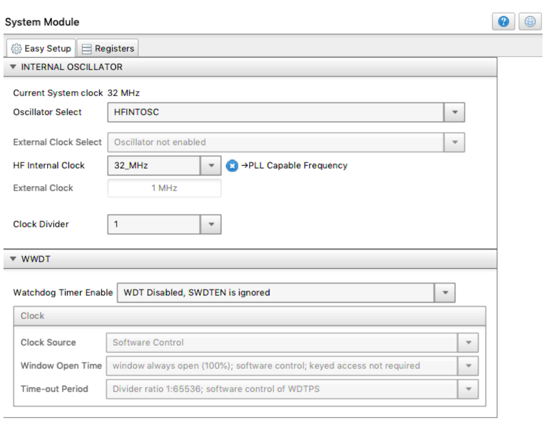


Connect the "**Ultrasonic Sensor**" to "GND", "**5V**", "RC5", and "RC6". Note: Black is "GND", Purple is "5V", White is "Echo", and Gray is "Trig".

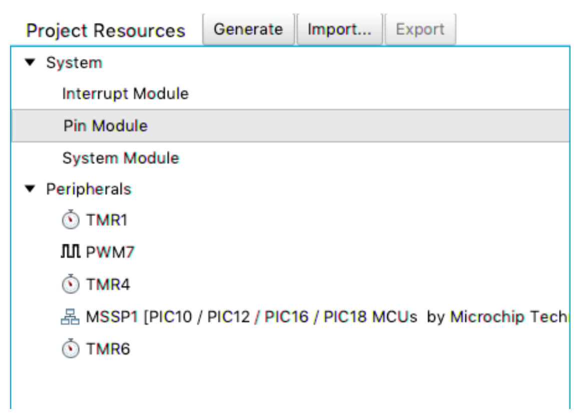


Connect the "**Servomotor**" to "GND", "3.3V", and "RC7". Note: Brown is GND, Red is VCC, and orange is PWM.

Now, we configure the hardware through Microchip's MCC (MPLAB Code Configurator) within MPLAB X IDE. Begin by creating an empty project and adding an empty "main.c"; then, enter MCC.

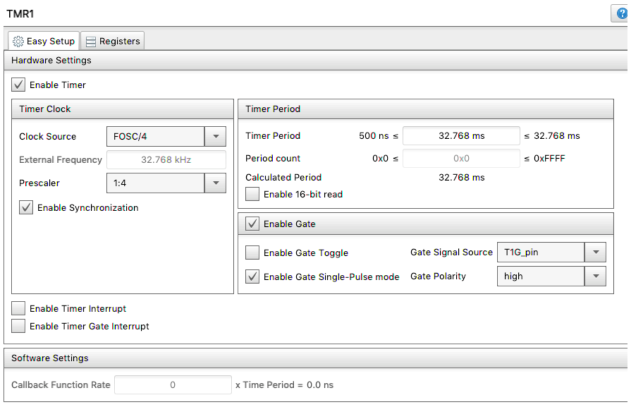


Configure "System Module" to reflect the options chosen in the example above.

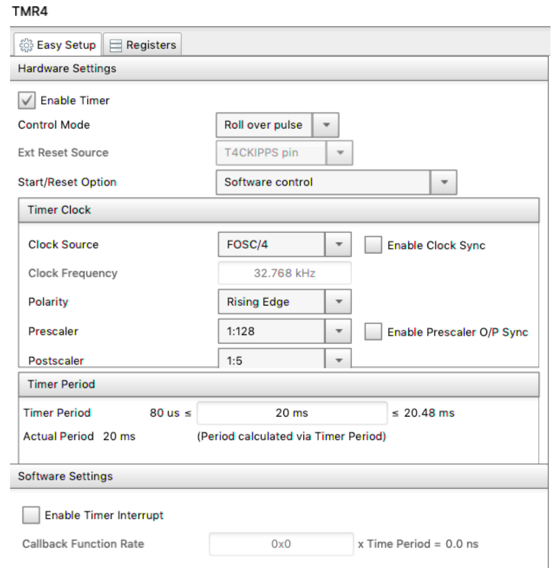


Add the Peripherals seen above (TMR1, PWM7, TMR4, MSSP1, TMR6). Note: These peripherals are added through the "Device Resources" pane.

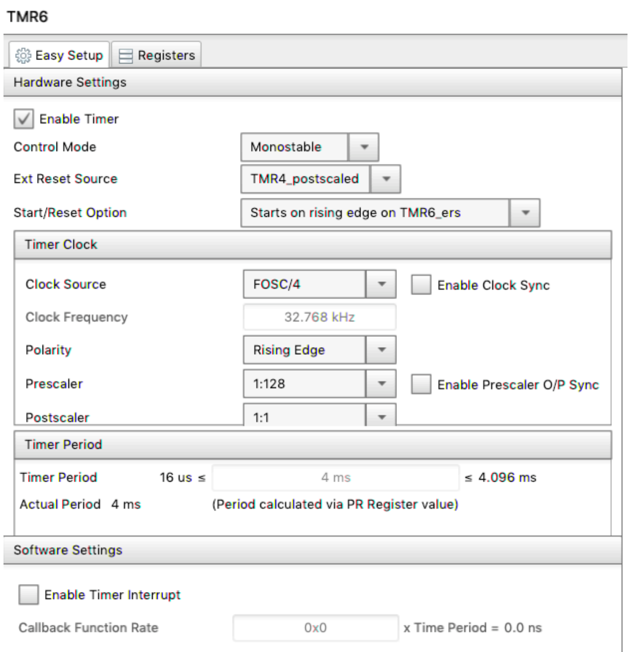
We must configure each peripheral individually (as shown below).



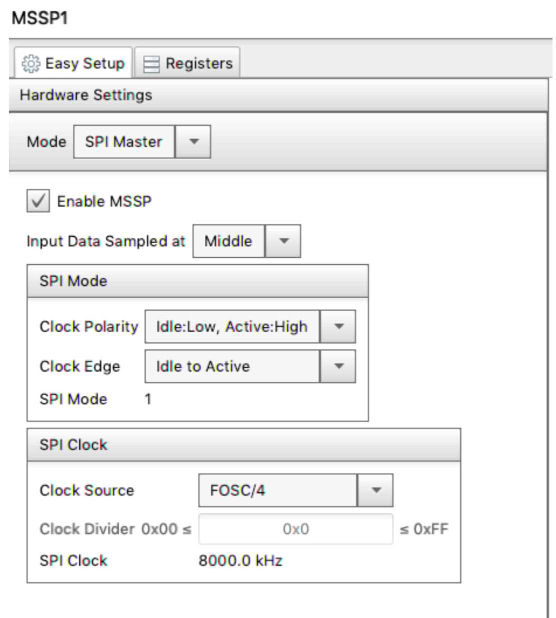
Configuration of **TMR1**



Configuration of **TMR4**

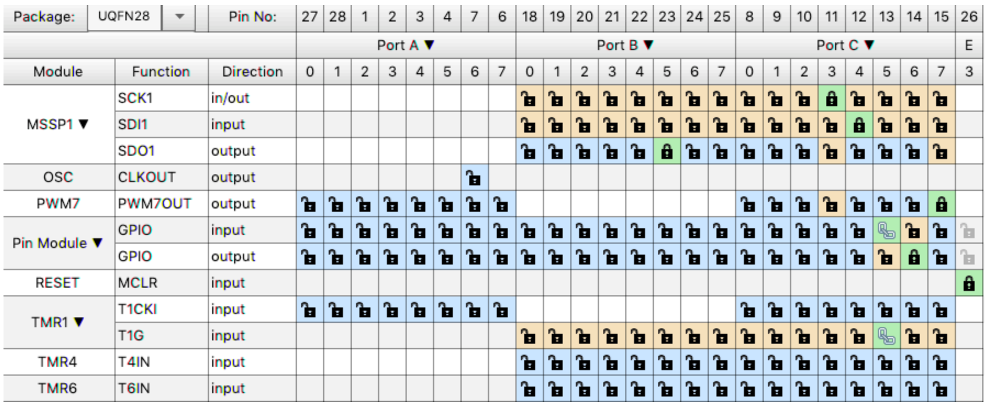


Configuration of **TMR6**



Configuration of **MSSP1**

We must also arrange the pin configuration.



Configuration of **Pin Manager** (in Grid View)

Pin Module

Easy Setup | Registers

Selected Package : UQFN28

Pin Name ▲	Module	Function	Custom Name	Start High	Analog	Output	WPU	OD	IOC
RB5	MSSP1	SDO1	SDO1	☐	☑	☑	☐	☐	none
RC3	MSSP1	SCK1	SCK1	☐	☐	☑	☐	☐	none
RC4	MSSP1	SDI1	SDI1	☐	☐	☐	☐	☐	none
RC5	Pin Module	GPIO	IO_RC5	☐	☐	☐	☐	☐	none
RC5	TMR1	T1G	IO_RC5	☐	☐	☐	☐	☐	none
RC6	Pin Module	GPIO	TRIG	☐	☑	☑	☐	☐	none
RC7	PWM7	PWM7OUT		☐	☑	☑	☐	☐	none

Configuration of **Pin Module**

At this point, we have chosen to *Generate* the updated MCC files (with our new configurations), saved, and exited. Now, we use the code provided (see Appendix Listing 1) as the contents of "main.c".

Results

The *Car Parking Sensor System* detailed in this report is functional. Objects are detected and distance in measured (within the ultrasonic sensors line-of-sight); events are triggered based on configured conditions resulting from such distance.

The system operates within the following states:

- **Imminent Collision (an object within 10cm of the ultrasonic sensor):** The servomotor will fully turn right, and the "Neopixel" will illuminate red.
- **Possible Collision (an object beyond 10cm, but within 30cm of the ultrasonic sensor):** The servomotor will begin turning right (proportionate to the distance of an object from the sensor), and the "Neopixel" will blink yellow (at a speed proportionate to the distance of an object from the sensor).
- **No Possible Collision (an object farther than 30cm from the ultrasonic sensor):** The servomotor will fully turn left, and the "Neopixel" will illuminate green.

Note: It is expected that *this* system does not perform to the standards of a production-ready parking proximity sensing system. There are many reasons for this; one being, in this report we design a system using a single ultrasonic sensor, whereas in *production* systems multiple sensors (ultrasonic, and others) are used to increase the range of detection.

Please visit: https://youtu.be/wyIV9HABg-4 to see a working example.

Discussion

Throughout the development of this *Car Parking Sensor System* only one major challenge presented itself; appropriately configuring the system oscillator. The TowerPro Servomotor typically functions using a low-clocked oscillator; whereas, the Adafruit Neopixel typically operates using a high-clocked oscillator. Thus, the problem arose such that two devices, that typically operate at different clock rates, must function using the same clock rate.

To compensate, we configure the system to operate off the highest clock rate needed among the required components (the Neopixel). Then, we use several timers to (essentially) act as scalers to achieve a suitable configuration for the servomotor to operate [2]. At this point, both components (that typically require different clock rates) function using the same clock rate.

Conclusion

Development and implementation of a *Car Parking Sensor System* has been shown using a microcontroller, ultrasonic sensor, servo motor, and Neopixel. Events are triggered based on the distance of an object from a sensor.

The *system* detailed in this report can be extended to increase usefulness. For example, by adding multiple ultrasonic sensors to the system, we can expand the range of view monitored; an aspect needed in real-world applications of a parking system.

References

[1] Traffic Congestion: Why It's Increasing And How To Reduce It. 2015 July 07. [Online]. Viewed 2019 April 20. Available: https://www.livablestreets.info/. traffic_congestion_why_its_increasing_and_how_to_reduce_it

[2] [Untitled]. [Undated]. [Online]. Viewed 2019 April 21. Available: https://github.com/RShankar/Intro-to-Microprocessors/blobmasterLab%20Project%20ExamplesUsing%20The%20Servo%20with%20a%20Neopixel/readme.md

Appendix

Listing 1: *main.c*

```c
#include "mcc_generated_files/mcc.h"

void send_pulse() {
    TRIG_SetHigh();// Sends trigger pulse
    __delay_us(5);
    TRIG_SetLow();
}

void NeoPixel_Stream(uint8_t *p, uint8_t count)
{   // sends count x GRB 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);
    }
}

void trigger_danger() {
    PWM7_LoadDutyValue(600);
    uint8_t color[] = {0xFF, 0, 0};
    NeoPixel_Stream(color, sizeof(color)/3);
}

void trigger_warning(int* yellow_warning, uint16_t distance) {

    PWM7_LoadDutyValue( (int)-(distance - 30) * 30 );
    uint8_t color[] = {0xFF, 0xFF, 0x00};
    *yellow_warning = ~*yellow_warning;
    if(*yellow_warning == -1) {
        color[0] = 0;
        color[1] = 0;
        color[2] = 0;
    }

    for(int i = 0; i < distance*11; i++)
        __delay_ms(1);

    NeoPixel_Stream(color, sizeof(color)/3);
}

void trigger_clear() {
    PWM7_LoadDutyValue(100);
    uint8_t color[] = {0, 0xFF, 0};
    NeoPixel_Stream(color, sizeof(color)/3);
}

void main(void)
{

    // initialize the device
    SYSTEM_Initialize();
    // initialize the PWM_6
    PWM7_Initialize();

    TRIG_SetLow();//set trigger low to being pulse
    uint16_t distance;

    int yellow_warning = 0;

    while (1)
    {
        send_pulse();

        TMR1GIF=0;//resets timer
        TMR1_WriteTimer(0);
        TMR1_StartSinglePulseAcquisition();
        while(!TMR1GIF);// waits for return pulse to end

        distance = (int)(TMR1_ReadTimer()/116);

        if(distance <= 10)
            trigger_danger();
        else if(distance <= 30)
            trigger_warning(&yellow_warning, distance);
        else
            trigger_clear();


        __delay_ms(5);//rate of read
    }
}
```