ELEX 7820 – Embedded Systems
Final Project Report
Team Moomba

Neil dehoog

Peymon Dadkhah

December 6, 2018

1 Contents

1 Introduction

Team Moomba has created a small autonomous vehicle that is related to, but not exactly the same as, the famous Roomba (for legal purposes). Our vehicle is designed to drive aimlessly throughout a space without running into any objects. To do this, the vehicle will approach to within a safe distance of an object, reverse a set distance, turn a set amount and then continue forward motion. This report outlines the hardware, software and logic that was used to accomplish the above objective.

2 Procedure

This section of the report outlines the two major components of our project. It focuses on the different aspects related to hardware and software of our vehicle to accomplish the autonomous wandering, in a crowded environment, that we desired. This section also goes over the testing of our vehicle and contains a sub-section that covers the mandatory/optional portions of our project with respect to the ELEX 7820 project requirements.

2.1 Hardware

The hardware for the Moomba was straight forward. We required a chassis, wheels, motors, a motor driver, motor encoders, a battery, ultrasonic range finders and the C2000 Piccolo as the required central controller. There are many commercially available options for each of the above. **Table 1** outlines the specific hardware that was used on the Moomba. **Figure 1** shows the high-level layout and wiring used. Note that the final position of the battery and controller might vary slightly but the overall layout and wiring should be the same. The C2000 controller pins and their applicable uses can be viewed in **Appendix A**.

Component	Model	Description
Chassis/Wheels	Robot Shop Smart Car V1	4wd chassis made of laser cut fiber board. 253mm x 148mm with mounts for various instrumentation. Plastic wheels with a diameter of 68mm
Motors	JSDJ 1	6v 1:48 geared motor. 4 motors in total
Ultrasonic Range Finder	HC-SR04	5v ultrasonic sensor with a range of 2cm-500cm. Resolution of 1cm. Effective angle 15°.
Motor Driver	L298	Wire screw mounts for 2 DC motors at 2v-46v. Enable and directional control pins for each motor. 4 stabilized 5v pins and 2 ground pins.
Battery	NiMH	6v supply with 2800mAh capacity. Female Tamagotchi connection type.
Motor Encoder	RB-Wav-100	Photo Interrupter that sends an active low signal. 3.3v
Microcontroller	TI-C2000	28027 model. 60Mhz CLK speed. Extensive capabilities that can be viewed in the C2000 datasheet.

Table 1 - Hardware components used on the Moomba autonomous vehicle

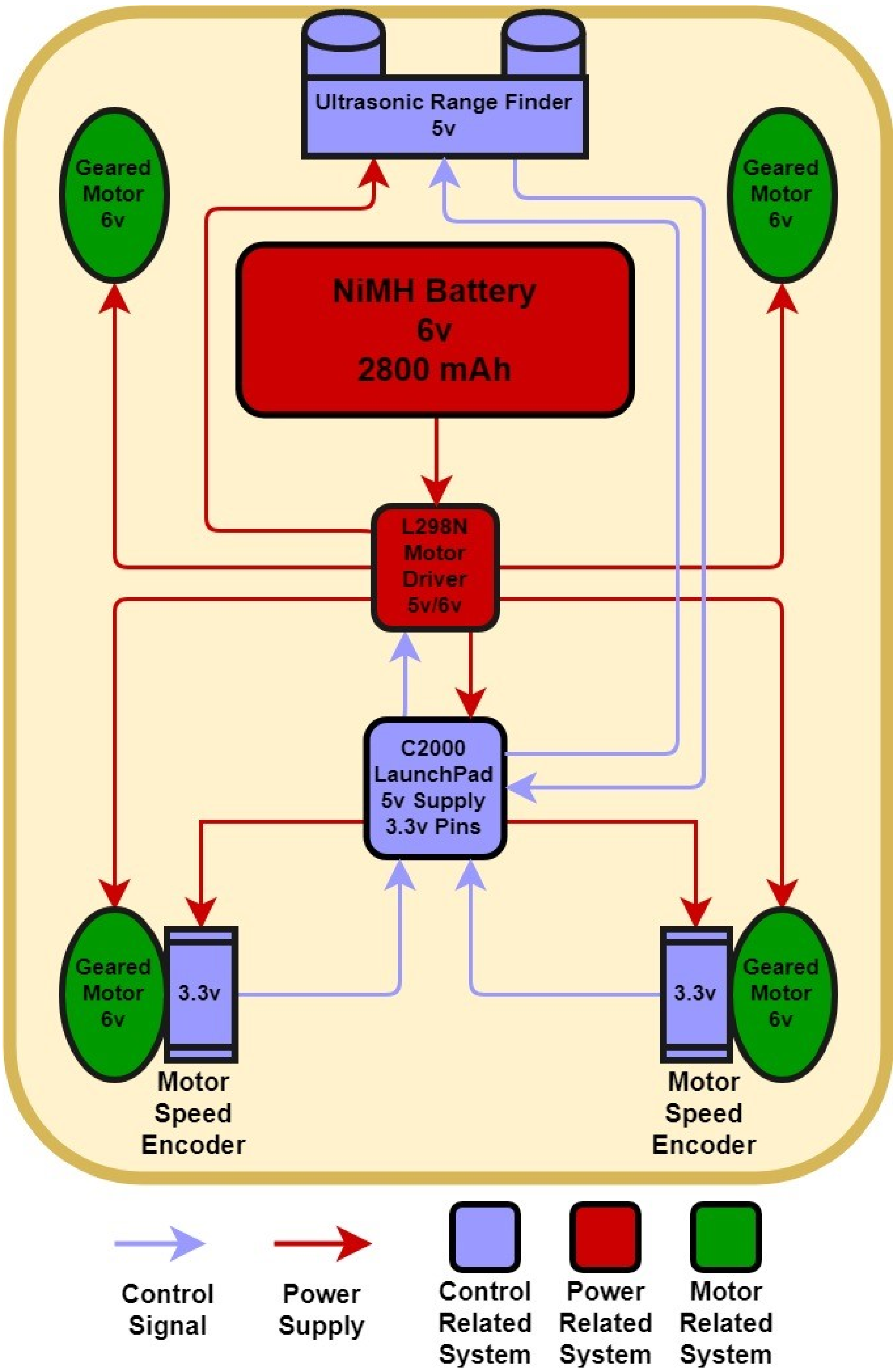

Figure 1 - Moomba hardware layout including high level power and control wiring

2.2 Software

This section contains a high-level priority diagram (**Figure 2**) and description of our various software threads for the C2000 controller. For the full Moomba C2000controller code, please see **Appendix C**.

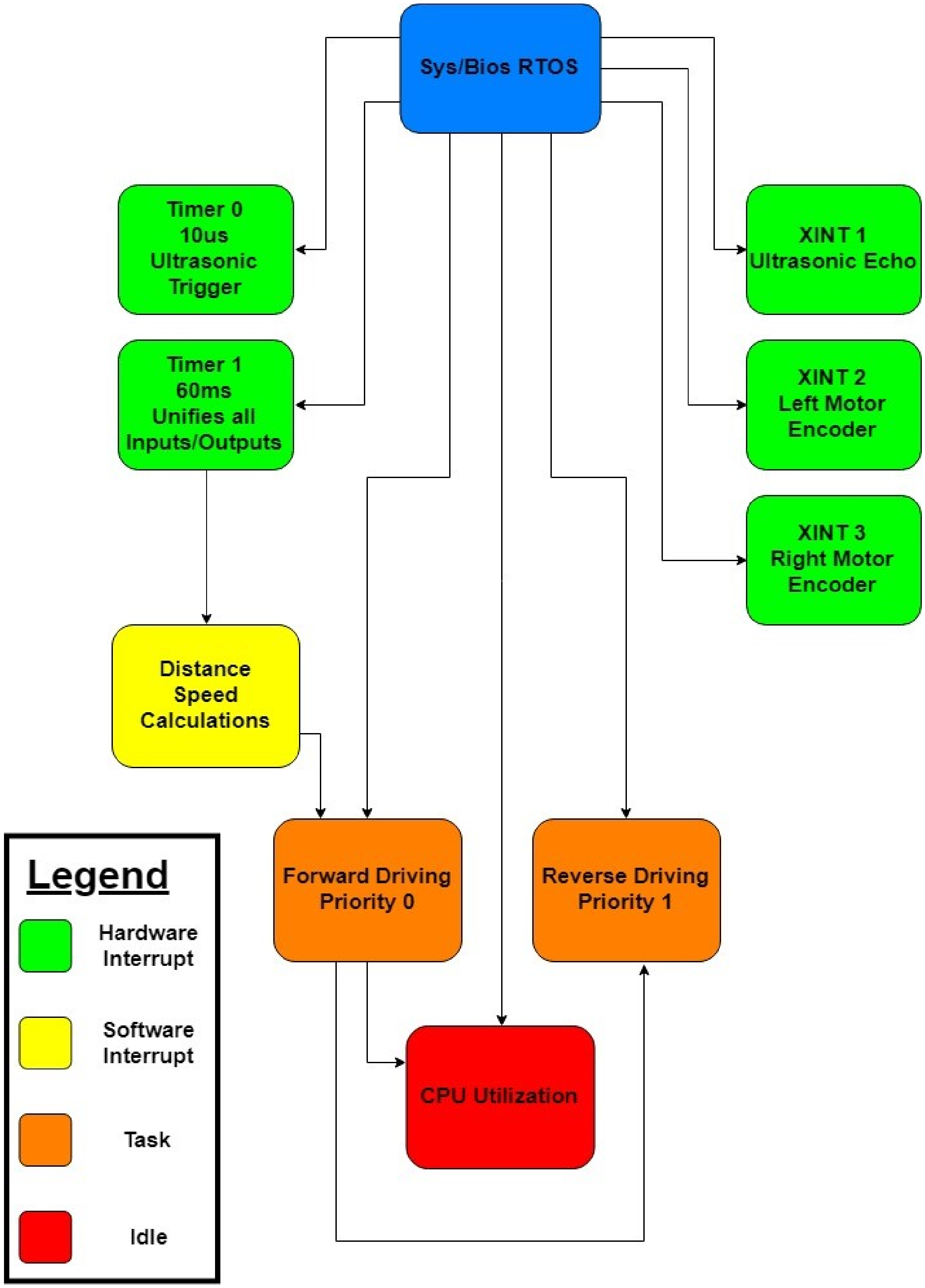

Figure 2 - High level chart of Moomba's C2000 controller's software threads

2.2.1 Hardware Interrupts

- Timer 0 – 10us timer used to trigger the ultrasonic sensor to become active.
- Timer 1 – 60ms timer used to ensure that the ultrasonic sensor has enough time to emit its sonar signals and then receive/transmit the response on the echo pin of the ultrasonic sensor to the C2000. This timer also gives a basis for the number of motor encoder pings per set amount of time. The encoder emits 20 pulses per rotation of the tire. Posts the SWI Speed.
- XINT1 – Catches the rising and falling edge of the ultrasonic echo signal.
- XINT2 – Catches Left motor encoder pulses and increments a count for each pulse.
- XINT3 – Catches Right motor encoder pulses and increments a count for each pulse.

2.2.2 Software Interrupts

- Speed – Calculates and updates the current speed and distance from nearest approaching object for the Moomba.

2.2.3 Tasks

- Forward – Task priority 0. Dynamically adjusts the Moomba's speed as it approaches an object. Also posts the Reverse task when an approaching object has fallen below a critical distance. Posts Idle if the Reverse task is not needed.
- Reverse – Task priority 1. Reverses the Moomba away from the object that was being approached by a set number of motor encoder pulses and then initiates a set turn to the left. Pends once the required turn has been completed.

2.2.4 Idle

- Toggles an idle pin that can be read through the use of an oscilloscope to calculate CPU usage.

2.3 Testing

To test the Moomba, we first assembled, coded and tested each individual hardware component. This was done through the use of the Code Composer Studio debugger to monitor the changes in variables in tandem with an oscilloscope to monitor various signal pulses. Once each individual component was working correctly, we constructed the real time operating system and began debugging the full working system. Debugging the final system involved setting the Moomba loose in a real environment and seeing how it performed. We used this performance to dial in the following criteria:

- Stopping distance
- Appropriate distance to reverse when an object has been approached to within safe limits
- Number of motor encoder pulses per desired turning arc
- Distance from an object at which the dynamic slow down should be engaged.

Through testing, these parameters were solidified, and the final version of the rover could be assembled. The final step completed was securing the hardware to avoid any unnecessary wires and components from catching on external objects.

2.4 ELEX 7820 Project Requirements

This section outlines how our project covers the mandatory and some of the optional requirements for the ELEX 7820 final project.

2.4.1 Mandatory Requirements

- **SYS/BIOS RTOS** – The Moomba utilizes HWI, SWI, TSK and IDLE threads as outlined in **section 2.2**
- **No floating-point datatypes** – Our controller code scales and then truncates integer values appropriately to get the desired resolution, without having to use floating point data types. This can be confirmed in **Appendix C**.
- **CPU Utilization** – All HWI, SWI and TASKS set the idle pin high while the idle function sets the idle pin low. By measuring the percent time high of the idle pin in a given period, we can determine the CPU utilization. Through this process we determined that the CPU utilization was 65.8% (**Figure 3**).
- **Digital Signal Processing** – The Moomba utilizes several digital signal processing algorithms to operate. These algorithms are further explained in **section 2.5**.
- **Real-Time-Processing** – The Moomba would function erratically if it missed a sample or event. Its operational performance and CPU utilization test both support the fact that it is processing in real time.

Figure 3 - Percent CPU utilization for the C2000 under normal operating conditions

2.4.2 Optional Requirements

- **Standalone Operation** – The Moomba operates from a portable battery pack and wanders autonomously, without being connected to a computer USB port.
- **Reliable** – The Moomba operates reliably and will self correct if an unknown state somehow occurs in the environment.
- **Add-on Board** – The Moomba utilizes a motor driver, 2 motor encoders and an ultrasonic sensor to operate successfully.
    - **Motor Encoders** – Require a GPIO pin to read a falling edge from the motor encoder signal.
    - **Motor Driver** – Requires 3 GPIO pins per left and right side of the Moomba. Two pins with PWM capability to control speed and direction, and one GPIO output to enable motor functionality.
    - **Ultrasonic Sensor** – The ultrasonic sensor requires one output GPIO pin to send a 10us pulse to the sensor to activate it. After 8ms of sampling, the ultrasonic sensor sends an echo pulse to a separate GPIO pin where the length of the pulse needs to be interpreted to calculate distance to the nearest object.

2.5 Digital Signal Processing

The Moomba has two digital signal processing algorithms that allow it to function properly. The first relates to synchronization of the left and right motors. The second relates to dynamically scaling motor power as the Moomba approaches an object.

Cheaper DC motors often have variability in their output power for a given PWM signal. To account for this, the encoder from the left and right-side motors are constantly tracked and compared over time. If one encoder is pulsing faster than the other, the faster encoders motor will have its PWM duty cycle reduced to bring it in line with the slower motors output.

We found that the Moomba would have to initiate a stop at a very long distance from an object if full motor power was used at all times. To correct for this, we created a dynamic reduction in motor duty cycle as the Moomba approaches an object. The following formula correctly implements this effect (DC = Duty Cycle):

$$DC = \frac{(Max\ DC - Min\ DC) * \left(\frac{(Object\ Range - Min\ Safe\ Object\ Range) * 100}{Dynamic\ Slow\ Down\ Initiation\ Range}\right) + Min\ DC}{100}$$

3 Results

The following video shows the Moomba operating in all its glory:

https://www.youtube.com/watch?v=ZOBH44eo8TY

4 Conclusions

Overall, the Moomba project was a complete success. The team performed well together, and we learned a great deal about implementing a real time operating system in an embedded system. Our hardware selection was appropriate, except for the fact that we over estimated the number of external interrupts available on the C2000 controller. This limited us to a single ultrasonic range finder. However, the Moomba achieved its design objective despite the single range finder limitation.

5 Appendix A – C2000 Table of Pin Use

C2000 Physical Pin	Pin Function Selected	Pin Use	Hardware Connected to Pin
Row J6 – Pin 1	GPIO 0	High/Low Output	CPU Utilization
Row J6 – Pin 3	EPWM2A	PWM	Left Motor Reverse
Row J6 – Pin 4	EPWM2B	PWM	Left Motor Forward
Row J6 – Pin 5	EPWM3A	PWM	Right Motor Reverse
Row J6 – Pin 6	EPWM3B	PWM	Right Motor Forward
Row J2 – Pin 2	GPIO 19	High/Low Output	Enable Left Motor
Row J2 – Pin 3	GPIO 12	High/Low Output	Enable Right Motor
Row J2 – Pin 9	GPIO 7	High/Low Input	Ultrasonic Echo
Row J2 – Pin 8	GPIO 6	High/Low Output	Ultrasonic Trigger
Row J1 – Pin 7	GPIO 18	High/Low Input	Left Motor Encoder
Row J1 – Pin 3	GPIO 28	High/Low Input	Right Motor Encoder

Table 2 - C2000 pin selection

6 Appendix B – Compartmentalization of Project Workload

Completed by Neil:

- Project Final Report

- Device initialization, control register set up and pin functionality including GPIO, PWM, peripheral interrupts and timers

Completed by Peymon:

- SYS/BIOS Thread set up
- Thread programming
- Final RTOS implementation

Completed Together:

- Thread selection and priorities
- Debugging troublesome code
- DSP Algorithms
- Final testing and implementation of the Moomba Rover

7 Appendix C – C2000 Controller Implementation Code

7.1 Task.c

```
#define xdc__strict // gets rid of #303-D typedef warning re Uint16, Uint32
#include <xdc/std.h>
#include <ti/sysbios/BIOS.h>
#include <xdc/runtime/System.h>
#include <ti/sysbios/knl/Swi.h>
#include <ti/sysbios/knl/Task.h>
#include <ti/sysbios/knl/Semaphore.h>
#include <Peripheral_Headers/F2802x_Device.h>

//function prototypes
extern void DeviceInit(void);       // Initializes device peripherals

// Global Constant variables
#define      ZERO   0
#define T2DIS      4524               // Value t oconvert time to distance in cm
#define      D2SPEED 834                          // Convert distance by # tach
ticks to speed
#define MAXPWM      550
#define      MINPWM 200
#define MIDDIST     30
#define      MINDIST      10
#define REVCNT      30
#define      TURNCNT      45
```

```
// Global variables (GV)
volatile UInt FpwmDC1 = 0;          // Pwm duty cycle between ZERO - MAXPWM
volatile UInt FpwmDC2 = 0;          // Pwm duty cycle between ZERO - MAXPWM
volatile UInt RpwmDC = 0;           // Pwm duty cycle between ZERO - MAXPWM

volatile UInt carState = 0;         // status of car ==> 0=Stopped,  1=Slow,
2=Fast,  3=Turning
//=======GVs for UltraSonic Sensor===========
volatile UInt range = 0;            // Range from ultrasonic sensor in cm
volatile UInt riseE = 0;            // Rising edge detector of ultrasonic echo
volatile UInt trig = 0;             // Tracks triggering of the ultrasonic sensor
volatile ULong uStartT = 0;            // Start of ultrasonic echo pulse
volatile ULong uStopT = 0;             // End of ultrasonic echo pulse
volatile long uPulseT = 0;             // Total time of ultrasonic echo pulse in
timer counts
volatile ULong period = 0;             // time period for Ultrasonic range sensor
to calculate distance
volatile ULong dist = 0;               // distance in cm to an object in front
//=======GVs for tachometer Sensors===========
volatile ULong tGetSpeed = 0;       // Count number of 6mS interrupts and get
speed after 5 interrupts
volatile UInt tachCntr1 = 0;           // number of tachometer interrupts in 30mS
volatile UInt speed1 = 0;              // Speed of motors on side 1
volatile UInt tachCntr2 = 0;           // number of tachometer interrupts in 0.3
seconds
volatile UInt speed2 = 0;              // Speed of motors on side 2

//=======Semaphores declarations===========
extern const Semaphore_Handle ReverseTsk;
extern const Semaphore_Handle ForwardTsk;
extern const Swi_Handle SpeedSWI;
//===============================================================================

/*=========================================================
 * Main functions just initializes everything and then runs
 * the SYS/BIOS
 =========================================================*/
void main(void)

{
    DeviceInit();                           //Initialize peripherals
    CpuTimer0Regs.TCR.bit.TIE = 1;          // Enable timer0 ISR
    System_printf("Initialized\n");         //Confirm initialization occurred
    BIOS_start(); // Initiate real time operating system, does not return
}

//===============================================================================
/*=========================================================
 *ReverseTskFxn is priority 2 and backs up the car for REVCNT
 * distance then turn for (TURNCNT-REVCNT) distance. Then hands the car
 * back to ForwardTskFxn & Pends.
 =========================================================*/
```

```
Void ReverseTskFxn(Void)
{
    for(;;){
        GpioDataRegs.GPASET.bit.GPIO0 = 1;    // set GPIO 1 to High to get CPU
utilization

             //First back up the car for REVCNT/20 revelotion
        if (tachCntr1 < REVCNT && carState==3 ){
            FpwmDC2 = FpwmDC1 = ZERO;
            RpwmDC = MINPWM;
            GpioDataRegs.GPASET.bit.GPIO12 = 1;
            GpioDataRegs.GPASET.bit.GPIO19 = 1;
            EPwm2Regs.CMPA.half.CMPA = FpwmDC1;          //Forward speed set for
motors1

            EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                 //Forward speed
set for motors2
            EPwm2Regs.CMPB = RpwmDC;                            //Reverse speed
set for motors1
            EPwm3Regs.CMPB = RpwmDC;                            //Reverse speed
set for motors2
        }
             //Then turn the car left for (TURNCNT-REVCNT)/20 revelotion
        else if (tachCntr1 < TURNCNT && carState==3){
            FpwmDC1 = MINPWM+50;
            FpwmDC2 = ZERO;
            RpwmDC = MINPWM+50;
            GpioDataRegs.GPASET.bit.GPIO12 = 1;
            GpioDataRegs.GPASET.bit.GPIO19 = 1;
                    EPwm2Regs.CMPA.half.CMPA = FpwmDC1;         //Forward speed
set for motors1
            EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                 //Forward speed
set for motors2
            EPwm2Regs.CMPB = FpwmDC2;                           //Reverse speed
set for motors1
            EPwm3Regs.CMPB = FpwmDC1;                           //Reverse speed
set for motors2

        }
             // when done stop the car and change car state to 1. post ForwardTsk
and pend  ReverseTsk
        else{
            FpwmDC2 = FpwmDC1 = ZERO;
            RpwmDC = ZERO;
            GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
                    EPwm2Regs.CMPA.half.CMPA = FpwmDC1;         //Forward speed
set for motors1
            EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                 //Forward speed
set for motors2
            EPwm2Regs.CMPB = FpwmDC2;                           //Reverse speed
set for motors1
            EPwm3Regs.CMPB = FpwmDC1;                           //Reverse speed
set for motors2
```

```
                    tachCntr1 = 0;
                    tachCntr2 = 0;
            carState = 2;
            Semaphore_post(ForwardTsk);             // Post ForwardTsk task
            Semaphore_pend(ReverseTsk, BIOS_WAIT_FOREVER);
        }
    }
}
/*==========================================================
 *ForwardTskFxn is priority 1 and sets the car speed based
 * on distance to obsticle posts ReverseTskFxn when closer
 * that MINDIST cm to wall. This task gets pended everytime
 ==========================================================*/
Void ForwardTskFxn(Void)
{
    for(;;){
        GpioDataRegs.GPASET.bit.GPIO0 = 1;    // set GPIO 1 to High to get CPU
utilization
             // when distannce less than MINDIST stop car and post ReverseTsk
        if (dist < MINDIST && dist!= 1){ //NOTE: dist == 1 is special case of out
of range
            FpwmDC1 = FpwmDC2 = ZERO;
            RpwmDC = ZERO;
            GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;
            GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;
                    EPwm2Regs.CMPA.half.CMPA = FpwmDC1;           //Forward speed
set for motors1
            EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                   //Forward speed
set for motors2
            EPwm2Regs.CMPB = FpwmDC2;                             //Reverse speed
set for motors1
            EPwm3Regs.CMPB = FpwmDC1;                             //Reverse speed
set for motors2
            carState = 3;
            Semaphore_post(ReverseTsk);                    // Post Reverse task
        }
             // When distance b/w MIDDIST and MINDIST change speed dynamicly
slowing down
        else if (dist > MINDIST && dist < MIDDIST){
            FpwmDC2 = FpwmDC1 = (MAXPWM-MINPWM)*(((dist-MINDIST)*100)/(MIDDIST-
MINDIST))/100 + MINPWM;
            RpwmDC = ZERO;
                    EPwm2Regs.CMPA.half.CMPA = FpwmDC1;           //Forward speed
set for motors1
            EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                   //Forward speed
set for motors2
            EPwm2Regs.CMPB = FpwmDC2;                             //Reverse speed
set for motors1
            EPwm3Regs.CMPB = FpwmDC1;                             //Reverse speed
set for motors2
            GpioDataRegs.GPASET.bit.GPIO12 = 1;
            GpioDataRegs.GPASET.bit.GPIO19 = 1;
```

```
carState = 1;
        }
              //When distance greater than MINDIST ensure the car is going
straight even when
              // there is friction on one side
        else if(dist >= MIDDIST || dist == 1){ //NOTE: dist == 1 is special case
of out of range

                 if (speed1 >= (speed2+100)){
                    FpwmDC1 = MAXPWM - 50;
                    FpwmDC2 = MAXPWM;
                }
                else if (speed2 >= (speed1+100)){
                    FpwmDC1 = MAXPWM;
                    FpwmDC2 = MAXPWM - 50;
                }
                else
                    FpwmDC1 = FpwmDC2 = MAXPWM;
                 RpwmDC = ZERO;
                        EPwm2Regs.CMPA.half.CMPA = FpwmDC1;           //Forward speed
set for motors1
                EPwm3Regs.CMPA.half.CMPA = FpwmDC2;                   //Forward speed
set for motors2
                EPwm2Regs.CMPB = FpwmDC2;                             //Reverse speed
set for motors1
                EPwm3Regs.CMPB = FpwmDC1;                             //Reverse speed
set for motors2
                GpioDataRegs.GPASET.bit.GPIO12 = 1;
                GpioDataRegs.GPASET.bit.GPIO19 = 1;
                carState = 2;
        }
        Semaphore_pend(ForwardTsk, BIOS_WAIT_FOREVER);     //pend task everytime
(this is for easy measurement of CPU utilization
    }
}

/*=========================================================
 * Timer 0 and 1 create a 10us pulse every 60ms to activate
 * the ultraSonic sensor, and wait for the response to be
 * measured
 * Timer 0 is only 10us for the pulse
 * Timer 1 is 60ms to wait for the UltraSonic response and
 * Get the speed of the wheels
 =========================================================*/
void timer0Fxn(void)
{
       GpioDataRegs.GPASET.bit.GPIO0 = 1;      // set GPIO 1 to High to get CPU
utilization
       //make a 10us trigger and then interrupt is disabled
    if (!trig){
        trig++;
        GpioDataRegs.GPASET.bit.GPIO6 = 1;      // Set Trigger Pin
        Semaphore_post(ForwardTsk);
    }
```

```
    else if (trig == 1){
        GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;    // Clear Trigger Pin
        XIntruptRegs.XINT1CR.bit.ENABLE = 1;    // Enable Peripheral Interrupt 1
        CpuTimer0Regs.TCR.bit.TIE = 0;          // Disable timer0 ISR
        trig++;
    }
}
// 1 periodif 60mS interrupt which pulses the ultraSonic sensor and counts
// the time for distance traveled
void timer1Fxn(void)
{
      GpioDataRegs.GPASET.bit.GPIO0 = 1;      // set GPIO 1 to High to get CPU
utilization
    trig = 0;                                   // Reset Trigger tracker
    riseE = 0;                                  // Clear rising edge count
    CpuTimer0Regs.TCR.bit.TIE = 1;              // Enable timer0 ISR
    XIntruptRegs.XINT1CR.bit.ENABLE = 0;        // Disable Peripheral Interrupt 1
      // counts to 5 before speed is calculated (get speed every 0.3 seconds)
    tGetSpeed++;
}
//================================================================================
/*========================================================
 * UltraSonicHWIFxn uses an external interrupt to find the
 * time elapsed between the rising edge and falling edge
 * this time elapsed then gives us the distance to obsticle.
 ========================================================*/
void ultraSonicHWIFxn(void)
{
      GpioDataRegs.GPASET.bit.GPIO0 = 1;      // set GPIO 1 to High to get CPU
utilization
    if (riseE == 0){ //at rising edge get cpu time
        riseE++;
        uStartT = ReadCpuTimer1Counter();
        period = ReadCpuTimer1Period();
    }

    else if (riseE == 1){ //at falling edge get cpu time and find elapsed time
        uStopT = ReadCpuTimer1Counter();
        uPulseT = (uStartT - uStopT);
        XIntruptRegs.XINT1CR.bit.ENABLE = 0;    // Disable Peripheral Interrupt 1
        riseE++;
        Swi_post(SpeedSWI);             //post SWI semaphor to get car speed and
obsticle distance
    }
}
```

```
/*=========================================================
 * Increments the tachometer counter 1 at negative edge.
 * Then calcualte distance traveled & car speed
 =========================================================*/
void Tach1HWIFxn(void)
{
	GpioDataRegs.GPASET.bit.GPIO0 = 1;	// set GPIO 1 to High to get CPU
utilization
    tachCntr1++;
}
/*=========================================================
 * Increments the tachometer counter 2 at negative edge.
 * Then calcualte distance traveled & car speed
 =========================================================*/
void Tach2HWIFxn(void)
{
	GpioDataRegs.GPASET.bit.GPIO0 = 1;	// set GPIO 1 to High to get CPU
utilization
    tachCntr2++;
}
//===============================================================================
/*=========================================================
 *Speed SWI gets the speed of travel and distance to object
 =========================================================*/
void SpeedSWIFxn(UArg arg)
{
	GpioDataRegs.GPASET.bit.GPIO0 = 1;	// set GPIO 1 to High to get CPU
utilization
	// get speed from distance traveled every 0.3 seconds
    if (tGetSpeed>4 && carState!=3){	//get speed every 0.3 seconds
        tGetSpeed = ZERO;
        speed1 = tachCntr1 * D2SPEED;
        tachCntr1 = ZERO;
             speed2 = tachCntr2 * D2SPEED;;
        tachCntr2 = ZERO;
    }
	// Calculate the distance and update the register
	dist = uPulseT/(T2DIS);
}


void myIdleFxn(void)
{
	GpioDataRegs.GPACLEAR.bit.GPIO0 = 1;	// set GPIO 1 to Low to get CPU
utilization
}
```

7.2 DeviceInit_18Nov2018.c

```
//=================================================================
// FILE:   DeviceInit_18Nov2018.c
//
// DESC:   Peripheral Initialization for F2802x
//
// Version:     1.2
//
// Modified by: ND & PD 06Dec2018 for using SYS/BIOS in WonBot car Project
//=================================================================
#include "Peripheral_Headers/F2802x_Device.h"

void DeviceInit(void);
//-----------------------------------------------------------------
//  Configure Device for target Application Here
//-----------------------------------------------------------------
void DeviceInit(void)
{
   EALLOW; // temporarily unprotect registers

// LOW SPEED CLOCKS prescale register settings
   SysCtrlRegs.LOSPCP.all = 0x0002; // Sysclk / 4 (15 MHz)
   SysCtrlRegs.XCLK.bit.XCLKOUTDIV=2;
// PERIPHERAL CLOCK ENABLES
//---------------------------------------------------
// If you are not using a peripheral you may want to switch
// the clock off to save power, i.e., set to =0
//
// Note: not all peripherals are available on all 280x derivates.
// Refer to the datasheet for your particular device.
   SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 0;    // ADC
   //------------------------------------------------
   SysCtrlRegs.PCLKCR3.bit.COMP1ENCLK = 0;  // COMP1
   SysCtrlRegs.PCLKCR3.bit.COMP2ENCLK = 0;  // COMP2
   //------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.I2CAENCLK = 0;   // I2C
   //------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.SPIAENCLK = 0;   // SPI-A
   //------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.SCIAENCLK = 0;   // SCI-A
   //------------------------------------------------
   SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 0;  //eCAP1
   //------------------------------------------------
   SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  // ePWM1
   SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;  // ePWM2
   SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;  // ePWM3
   SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;  // ePWM4
   //------------------------------------------------
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;   // Enable TBCLK
   //------------------------------------------------
```

```
//------------------------------------------------------------------
    /*
     * Setting up all the GPIO Pins needed for the project
     * 4 EPWM setup (2A, 2B, 3A, 3B)
        * GPIO0 output for CPU utilization
     * GPIO_6 set for Ultrasonic trigger, GPIO7 set for XInt1
     * GPIO_19 set for EnA, and GPIO_12 set for EnB
     * Written by ND & edited by PD*/
    //------------------------------------------------------------------
    //==================================================================
    //Customization of original DeviceInit file for Car Project
//------------------------------------------------------------------
    // Initialize test pin. LED 0       pin. Row J6 - Pin 1
    //  GPIO-00 - PIN FUNCTION = blue LED D2 (rightmost on LaunchPad)
        GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 0;     // 0=GPIO,  1=EPWM1A,  2=Resv,
3=Resv
        GpioCtrlRegs.GPADIR.bit.GPIO0 = 1;      // 1=OUTput,  0=INput
        GpioDataRegs.GPACLEAR.bit.GPIO0 = 1;    // uncomment if --> Set Low
initially (Active low LED)
    //  GpioDataRegs.GPASET.bit.GPIO0 = 1;      // uncomment if --> Set High
initially
//------------------------------------------------------------------
    //  GPIO-02 - PIN FUNCTION = set up for EPWM2A use in forward mode for motor
1
        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;     // 0=GPIO,  1=EPWM2A,  2=Resv,
3=Resv
        GpioCtrlRegs.GPADIR.bit.GPIO2 = 1;      // 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO2 = 1;    // uncomment if --> Set Low
initially
        GpioDataRegs.GPASET.bit.GPIO2 = 1;      // uncomment if --> Set High
initially
//------------------------------------------------------------------
    //  GPIO-03 - PIN FUNCTION = et up for EPWM2B use in reverse mode for motor 1
        GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;     // 0=GPIO,  1=EPWM2B,  2=Resv,
3=COMP2OUT
        GpioCtrlRegs.GPADIR.bit.GPIO3 = 1;      // 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO3 = 1;    // uncomment if --> Set Low
initially
        GpioDataRegs.GPASET.bit.GPIO3 = 1;      // uncomment if --> Set High
initially
//------------------------------------------------------------------
    // GPIO-04 - PIN FUNCTION = set up for EPWM3A use in forward mode for motor 2
        GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;// 0=GPIO,  1=EPWM3A, 2=Resv,   3=Resv
        GpioCtrlRegs.GPADIR.bit.GPIO4 = 1; // 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;// uncomment if --> Set Low initially
        GpioDataRegs.GPASET.bit.GPIO4 = 1;// uncomment if --> Set High initially
//------------------------------------------------------------------
    // GPIO-05 - PIN FUNCTION = set up for EPWM3B use in reverse mode for motor 2
        GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;// 0=GPIO,  1=EPWM3B, 2=Resv,  3=ECAP1
        GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;// 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;// uncomment if --> Set Low initially
        GpioDataRegs.GPASET.bit.GPIO5 = 1;// uncomment if --> Set High initially
```

```
//-----------------------------------------------------------------
    // Initialize Ultrasonic Trigger    pin. Row J2 - Pin 8.
    //  GPIO-06 - PIN FUNCTION = --Spare--
        GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0; // 0=GPIO,  1=EPWM4A,  2=SYNCI,
3=SYNCO
        GpioCtrlRegs.GPADIR.bit.GPIO6 = 1; // 1=OUTput,  0=INput
        GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;// uncomment if --> Set Low initially
    //  GpioDataRegs.GPASET.bit.GPIO6 = 1;// uncomment if --> Set High initially
//-----------------------------------------------------------------
    // Initialize Ultrasonic interrupt     pin. Row J2 - Pin 9
        GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;//0=GPIO,  1=EPWM3A
        GpioCtrlRegs.GPADIR.bit.GPIO7 = 0;// 1=OUTput,  0=INput
        GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1;// disable internal pull-up resistor
        GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 7; // Select Interrupt GPIO
        XIntruptRegs.XINT1CR.bit.POLARITY = 3;// Select Interrupt on pos/neg edge
        XIntruptRegs.XINT1CR.bit.ENABLE = 0;// Disable Peripheral Interrupt XINT1
//-----------------------------------------------------------------
    // EPWM ENB (enable pin 'B')        pin. Row J2 - Pin 3
    //  GPIO-12 - PIN FUNCTION = --Spare--
        GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;// 0=GPIO, 1=TZ1,  2=SCITX-A,  3=Resv
        GpioCtrlRegs.GPADIR.bit.GPIO12 = 1; // 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;//uncomment if --> Set Low initially
        GpioDataRegs.GPASET.bit.GPIO12 = 1;// uncomment if --> Set High initially
//-----------------------------------------------------------------
        // External Interupt         pin. Row J1 - Pin 7
        //  GPIO-18 - PIN FUNCTION = GPIO set for input interrupt XINT2 for
tachometer 1
        GpioCtrlRegs.GPAMUX2.bit.GPIO18 = 0; // 0=GPIO,  1=SPICLK-A,  2=SCITX-A,
3=XCLKOUT
        GpioCtrlRegs.GPADIR.bit.GPIO18 = 0; // 1=OUTput,  0=INput
        GpioCtrlRegs.GPAPUD.bit.GPIO18 = 1; // disable internal pull-up resistor
        GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 18;// Select Interrupt GPIO
        XIntruptRegs.XINT2CR.bit.POLARITY = 0;// Select Interrupt on pos/neg edge
        XIntruptRegs.XINT2CR.bit.ENABLE = 1;// Enable Peripheral Interrupt XINT2
//-----------------------------------------------------------------
    // EPWM ENA (enable pin 'A')        pin. Row J2 - Pin 2
    //  GPIO-19 - PIN FUNCTION = GPIO output for encoder Enable A
    GpioCtrlRegs.GPAMUX2.bit.GPIO19 = 0;//0=GPIO, 1=SPISTE-A, 2=SCIRX-A,  3=ECAP1
        GpioCtrlRegs.GPADIR.bit.GPIO19 = 1;// 1=OUTput,  0=INput
    //  GpioDataRegs.GPACLEAR.bit.GPIO19 = 1;//uncomment if --> Set Low initially
        GpioDataRegs.GPASET.bit.GPIO19 = 1;// uncomment if --> Set High initially
//-----------------------------------------------------------------
    //  GPIO-28 - PIN FUNCTION = GPIO set for input interrupt XINT3 for
tachometer 2
        GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 0;//0=GPIO, 1=SCIRX-A, 2=I2C-SDA, 3=TZ2
      GpioCtrlRegs.GPADIR.bit.GPIO28 = 0; // 1=OUTput,  0=INput
      GpioCtrlRegs.GPAPUD.bit.GPIO28 = 1;// disable internal pull-up resistor
      GpioIntRegs.GPIOXINT2SEL.bit.GPIOSEL = 28;//Select Interrupt GPIO
      XIntruptRegs.XINT3CR.bit.POLARITY = 0;// Select Interrupt on pos/neg edge
      XIntruptRegs.XINT3CR.bit.ENABLE = 1;// Enable Peripheral Interrupt XINT2
```

```
    // EPWM2 Initialization
        EPwm2Regs.TBPRD = 600; // Period in TBCLK counts, sets period based on
SYSCLK(60MHz) / (CLKDIV*HSPCLKDIV)
        EPwm2Regs.CMPA.half.CMPA = 300;// Compare counter for EPWMA, sets duty
cycle
        EPwm2Regs.CMPB = 200; // Compare counter for EPWMB, sets duty cycle
      EPwm2Regs.TBPHS.all = 0;            // Set Phase register to zero
      EPwm2Regs.TBCTR = 0;                // clear TBCLK counter
      EPwm2Regs.TBCTL.bit.CTRMODE = 0;    // Set mode to up-count
      EPwm2Regs.TBCTL.bit.PHSEN = 0;      // Phase loading disabled
      EPwm2Regs.TBCTL.bit.PRDLD = 0;      // Select Shadow Register Loading
      EPwm2Regs.TBCTL.bit.SYNCOSEL = 3;   // Disables the EPWM Synchronization
      EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1;  //Multiplies SYSCLKDIV divisor CLKDIV
      EPwm2Regs.TBCTL.bit.CLKDIV = 7;//Determine SYSCLK Divisor to create TBCLK
      EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0;//Select Counter Compare
      EPwm2Regs.CMPCTL.bit.SHDWBMODE = 0;//Select Counter Compare
      EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;//Set timer to zero initially for EPWMA
      EPwm2Regs.CMPCTL.bit.LOADBMODE = 0;// Set timer to zero initially for EPWMB
      EPwm2Regs.AQCTLA.bit.ZRO = 2;//Set output high when counter = 0 for EPWMA
      EPwm2Regs.AQCTLA.bit.CAU = 1;//Set output low when counter = Compare
Counter for EPWMA
      EPwm2Regs.AQCTLB.bit.ZRO = 2;       // Set output high when counter = 0
      EPwm2Regs.AQCTLB.bit.CBU = 1;       // Set output low when counter =
Compare Counter for EPWMB
//-----------------------------------------------------------------
    // EPWM3 Initialization
        EPwm3Regs.TBPRD = 600;// Period in TBCLK counts, sets period based on
SYSCLK(60MHz) / (CLKDIV*HSPCLKDIV)
      EPwm3Regs.CMPA.half.CMPA = 300;// Compare counter for EPWMA, sets duty
      EPwm3Regs.CMPB = 200; // Compare counter for EPWMB, sets duty cycle
      EPwm3Regs.TBPHS.all = 0;            // Set Phase register to zero
      EPwm3Regs.TBCTR = 0;                // clear TBCLK counter
      EPwm3Regs.TBCTL.bit.CTRMODE = 0;    // Set mode to up-count
      EPwm3Regs.TBCTL.bit.PHSEN = 0;      // Phase loading disabled
      EPwm3Regs.TBCTL.bit.PRDLD = 0;      // Select Shadow Register Loading
      EPwm3Regs.TBCTL.bit.SYNCOSEL = 3;   // Disables the EPWM Synchronization
      EPwm3Regs.TBCTL.bit.HSPCLKDIV = 1;  //Multiplies SYSCLKDIV divisor CLKDIV
      EPwm3Regs.TBCTL.bit.CLKDIV = 7;// Determines SYSCLK Divisor to create TBCLK
      EPwm3Regs.CMPCTL.bit.SHDWAMODE = 0;//Select Counter Compare Shadow Mode
      EPwm3Regs.CMPCTL.bit.SHDWBMODE = 0;//Select Counter Compare Shadow Mode
      EPwm3Regs.CMPCTL.bit.LOADAMODE = 0;//Set timer to zero initially for EPWMA
      EPwm3Regs.CMPCTL.bit.LOADBMODE = 0;//Set timer to zero initially for EPWMB
      EPwm3Regs.AQCTLA.bit.ZRO = 2; // Set output high when counter = 0 for EPWMA
      EPwm3Regs.AQCTLA.bit.CAU = 1;//Set output low when counter = Compare
Counter for EPWMA
      EPwm3Regs.AQCTLB.bit.ZRO = 2;// Set output high when counter = 0 for
      EPwm3Regs.AQCTLB.bit.CBU = 1;       // Set output low when counter =
Compare Counter for EPWMB
//-----------------------------------------------------------------
    CpuTimer0Regs.TCR.bit.TIE = 0; // Disable timer0 ISR
//-----------------------------------------------------------------
    EDIS;   // restore protection of registers
} // end DeviceInit()
```