3-Wheeled Motorcycle

An Innovative Design with Unparalleled Possibilities

Team 8 – ABIF Design Group

Revision	Description
1	Updated main.c code and flowchart – Ian
2	Updated Overview, added use of UART pins to hardware, added conclusion – Atharv
3	Added test for Autonomous mode, updated footers – Atharv
4	Updated Hardware to include IOT System – Ben
5	Updated System Input Block to include IOT – Ben
6	Added test for IR – Fredy
7	Added IOT Test – Fredy
8	Updated ports.c code. Previous flow chart covers updated ports.c code – Atharv
9	Updated serial_interrupt.c code. Previous flow chart covers. – Fredy
10	Added power analysis results – Ian
11	Added block diagrams for additional functions – Atharv, Ian, Ben
12	Final formatting done – Fredy

Team Members
Atharv Shikarkhane
Fredy Santos
Ian Hellmer
Ben Surrett

Table of Contents

Table of figures entries.

1. Scope

This document describes how the 3-wheeled motorcycles is programmed and designed. The 3-wheeled motorcycle consists of an acrylic chassis, two motors (with wheels connected to them), a castor in the rear, and a collection of boards to control the vehicle. The boards consist of one Texas Instruments MSP430 board, one power and display board connected to the top pins of the TI board and one motor control board connected to the bottom pins of the TI board. The power and display board has an LCD mounted on it which can show relevant information. The TI board also has three buttons which can be programmed to do specific tasks. The vehicle is powered by 4 AA batteries which are mounted to the bottom of the chassis. The TI board has a micro-USB port which is used to transfer programs on to the microcontroller. An IOT is mounted on the power and display board, so that commands can be given to the MSP430 over Wi-Fi.

2. Abbreviations

LCD	Liquid Crystal Display
LED	Light Emitting Diode
MSP	Mixed Signal Processor
USB	Universal Serial Bus
FET	Field Effect Transistor
MCU	Microcontroller Unit
ADC	Analog to Digital Converter
IOT	Internet of Things
TI	Texas Instruments
CPU	Central Processing Unit
IO	Input/Output
IR	Infrared
GND	Ground
RXD	Receive Data
TXD	Transmit Data
PWM	Pulse Width Modulation
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
Wi-Fi	Wireless Fidelity

3. Overview

The system revolves around implementing the MSP430 microcontroller board to serve as the CPU and logic controller for external devices via its pin interface. The external devices include an LCD display, motors, various IO devices, as well as a power regulation device to control voltages. IO devices include LEDs, switches, buttons, potentiometers, and pins. The vehicle has two seral ports, which are used to communicate with a PC, or any other external device, like the analog discovery. An IOT is attached to the display and power board, which is used to connect to the user's local network. The MSP430 and the IOT communicate using the serial communications protocol.

The user also has a TCP Telnet client that runs on a separate device. Through this client commands can be sent wirelessly to the IOT, which get interpreted by the MSP430. The 3-Wheeled Motorcycle is able to navigate a numbed course by receiving commands through the IOT supplied by the user. The 3-Wheeled Motorcycle is also able to go into autonomous mode and navigate a black line course.

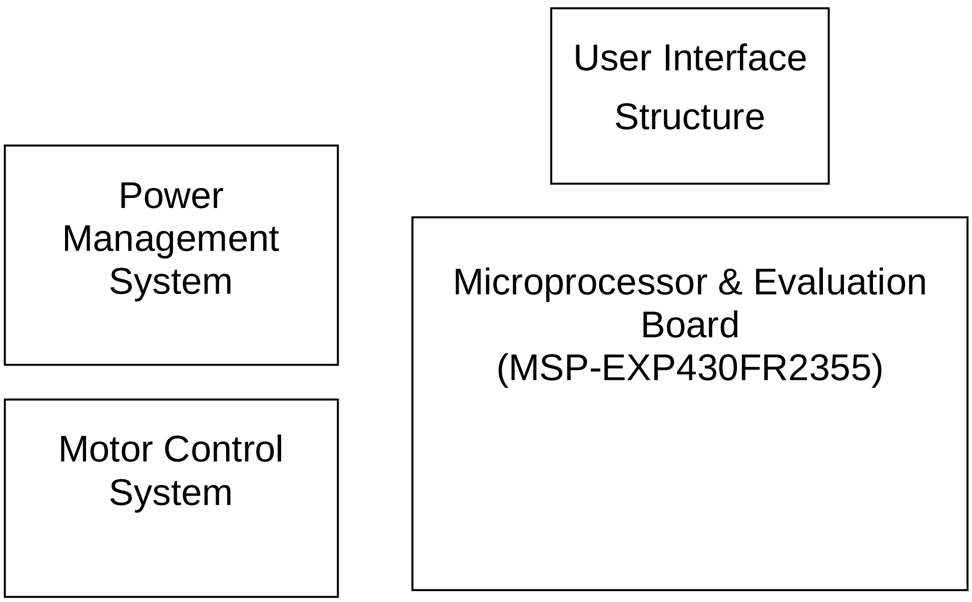

Figure 3.1 Block Diagram Outline

3.1. Microcontroller & Evaluation Board (MSP-EXP430FR2355)

The microcontroller and evaluation board serve the purpose of linking all the components of the device together. It also controls many of the signals that drive the connected components of the system. The board accomplishes this using ports. These ports are further divided into individual pins which send output signals and receive input signals. These input signals can be used by the microcontroller to dictate the actions of the entire system, effectively controlling the entire operation of the completed device.

The board's functional capability can be modified through the micro-USB port which can be connected to a computer where the necessary program can be downloaded onto the board. The push buttons attached to the board allow for manual changes in the operation of the board. LEDs present on the board are also able to provide feedback to the user.

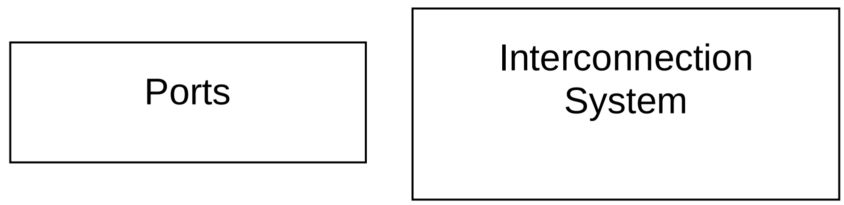

Figure 3.2 Block Diagram Microcontroller & Evaluation Board (MSP-EXP430FR2355)

3.2. Power Management System

The main function of the power management system is to safely deliver the correct amount of power to all boards and peripherals from the battery pack. The power management system is connected to the TI board through the GND, 5V, 3V3, RXD<<, TXD>>, SBWTDIO, and SBWTCK pins. On top of the power board, a display and backlight are connected. The power board also has a connector which connects to the battery pack. To control the flow of power from the battery pack, the board also has a switch. The batteries are replaceable.

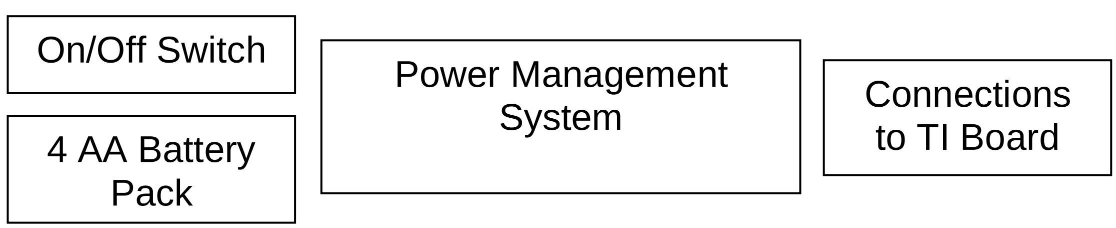

Figure 3.3 Block Diagram Power Management System

3.3. Motor Control System

The motor control system relies upon outputs from the MSP430 to control whether the motors are switched on or off. These motors are powered via the power System board via connector cables. The voltage is regulated by left and right H-Bridges which rely upon the switching nature of the FET's on the H-Bridges

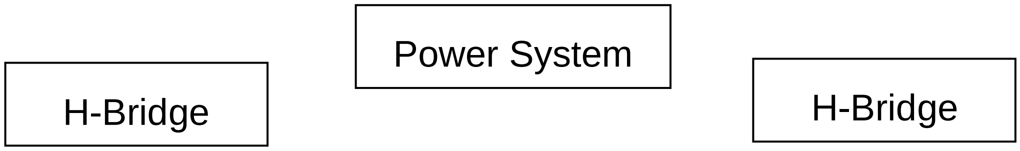

Figure 3.4 Motor Control System

3.4. System Input Block

As far as the user interface system is concerned, most operations are controlled through components scattered across multiple system boards. The two most vital ones are the LEDs and LCD. The LEDs, located on the microcontroller development board, help show the user that the device’s internal system clock is functioning normally. The microcontroller also has two push buttons on both sides that can be configured for user input.

The LCD, located on top of the power system, is used to show the systems current operation. The LCD contains other features, such as a backlight and thumbwheel. Most of these are not enabled by default.

The IOT module accepts wireless RS232 signals for the purpose of executing preprogrammed functions. These inputs allow for the car to perform a variety of tasks once the signal is parsed and the appropriate task has been determined.

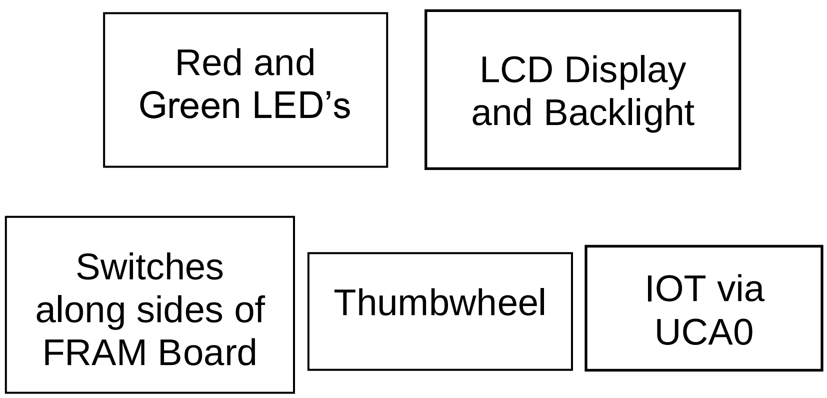

Figure 3.5 System Input Block

4. Hardware

4.1. Microcontroller & Evaluation Board (MSP-EXP430FR2355)

The MSP-EXP430FR2355 links all the components of the device together. It also controls many of the signals that drive the connected components of the system. The board accomplishes this using ports. These ports are further divided into individual pins which send output signals and receive input signals. Figure 4.1 displays the pin layout of the evaluation board. The MSP430 also comes with UART pins. These pins are used when communications using a serial protocol are necessary.

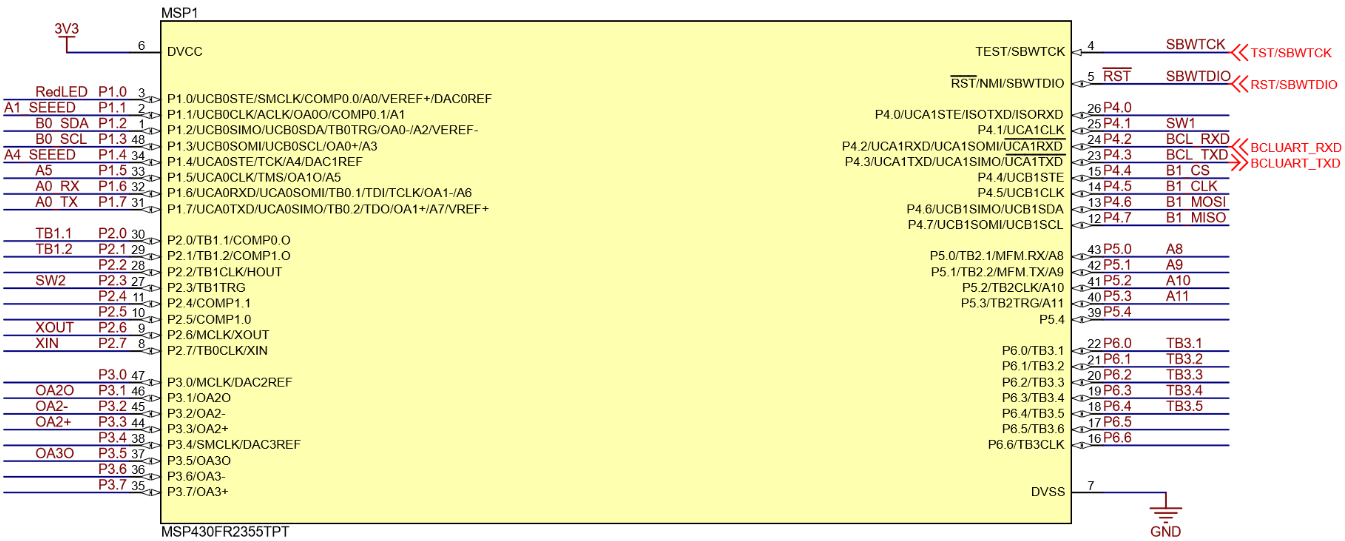

Figure 4.1 Block Diagram Microcontroller & Evaluation Board (MSP-EXP430FR2355)

4.2. Power Management System

The power management system safely delivers power to all the boards and the peripheral components attached to the board. It has a switch that can be used to toggle the power on and off. The power management system gets power from 4 AA batteries. It then puts the power through a buck boost converter before sending the power to the boards and the peripherals.

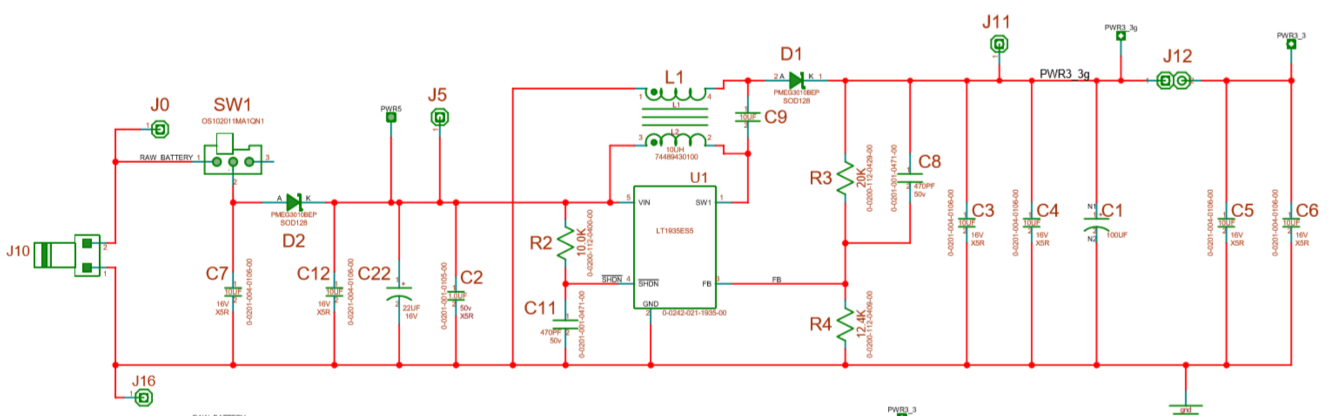

Figure 4.2 Power Management System

4.3. Motor Control System

At this stage, the motors are controlled by two FETs which have one mode, to turn them on when power is on. The FETs accomplish this purpose by increasing the voltage that is supplied to them in order to sufficiently drive the motors. The signal is driven from port 6 as controlled by the msp430. Configuring the outputs from port 6 to this control scheme must be done with great care because if they are accidentally configured to go forward and backwards at the same time it will completely fry the FETs.

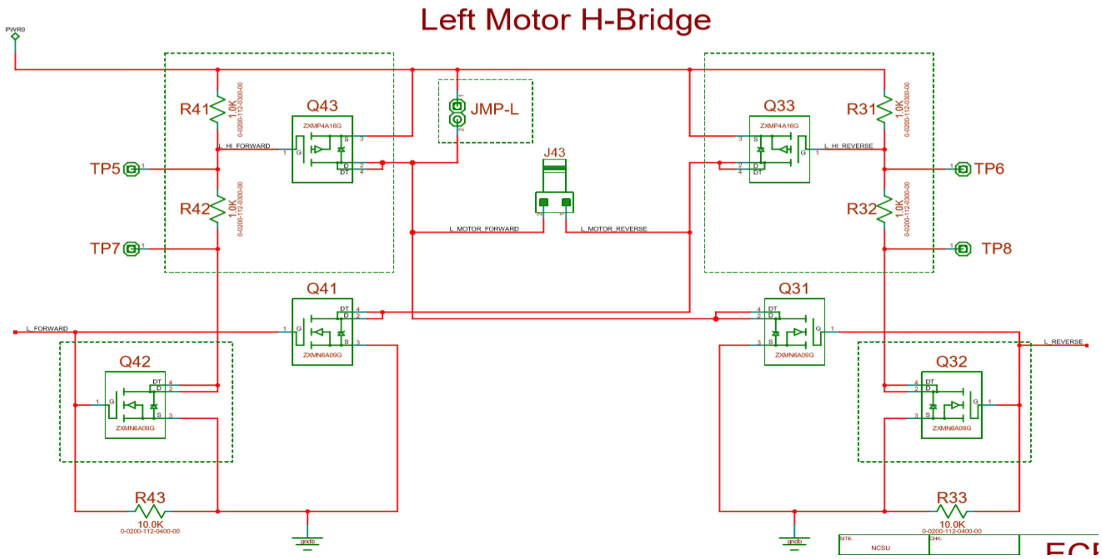

Figure 4.3 Left Motor H-Bridge

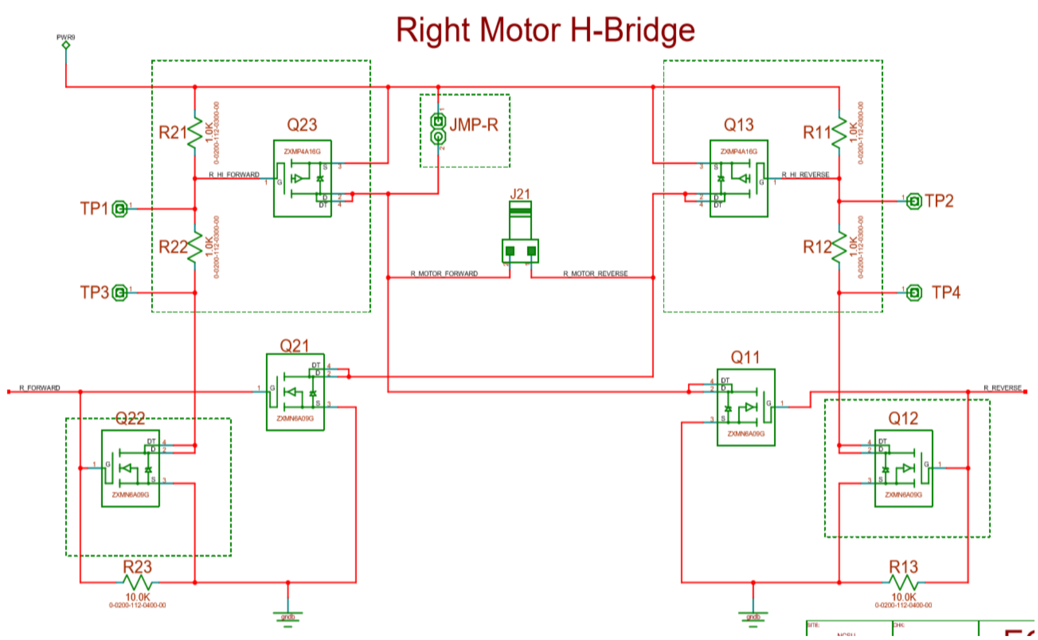

Figure 4.4 Right Motor H-Bridge

4.4. User Interface System

The LCD functions as the device's responsive user interface system and provides a means for debugging and manipulating the device's outputs while in use by giving visual feedback. The LCD can change what it displays by using the two buttons along the sides of the microcontroller, as the LCD is limited to only showing four lines of 10. characters at a time.

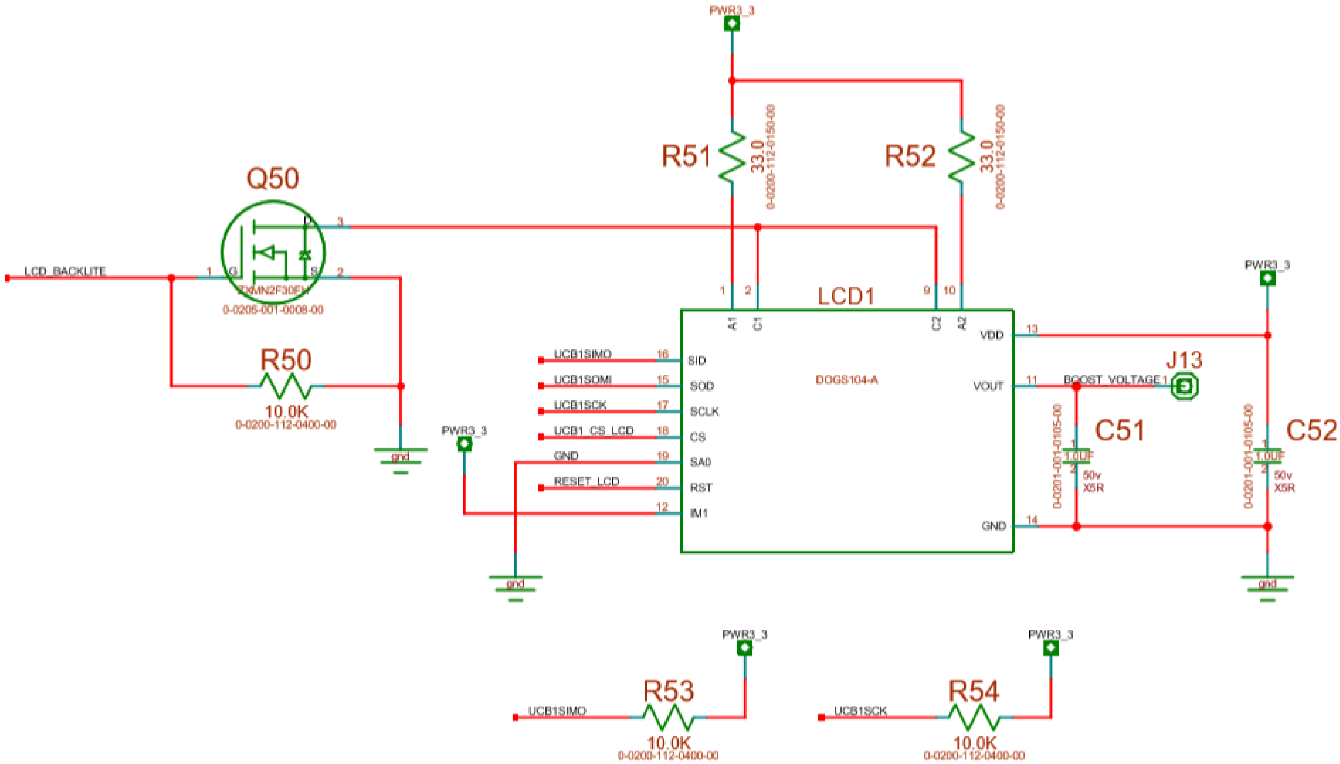

Figure 4.5 LCD

4.5. IOT System

The IOT System utilizes the msp430's UART channel UCA0 in order to transmit and receive serial communications signals following the RS232 protocol. This module allows for communication via wi-fi with other devices using a TCP terminal. This is pivotal for the function of the car as it allows for remote control function of the robot. The date received by the IOT module is passed onto the msp430 for software parsing via a loopback jumper

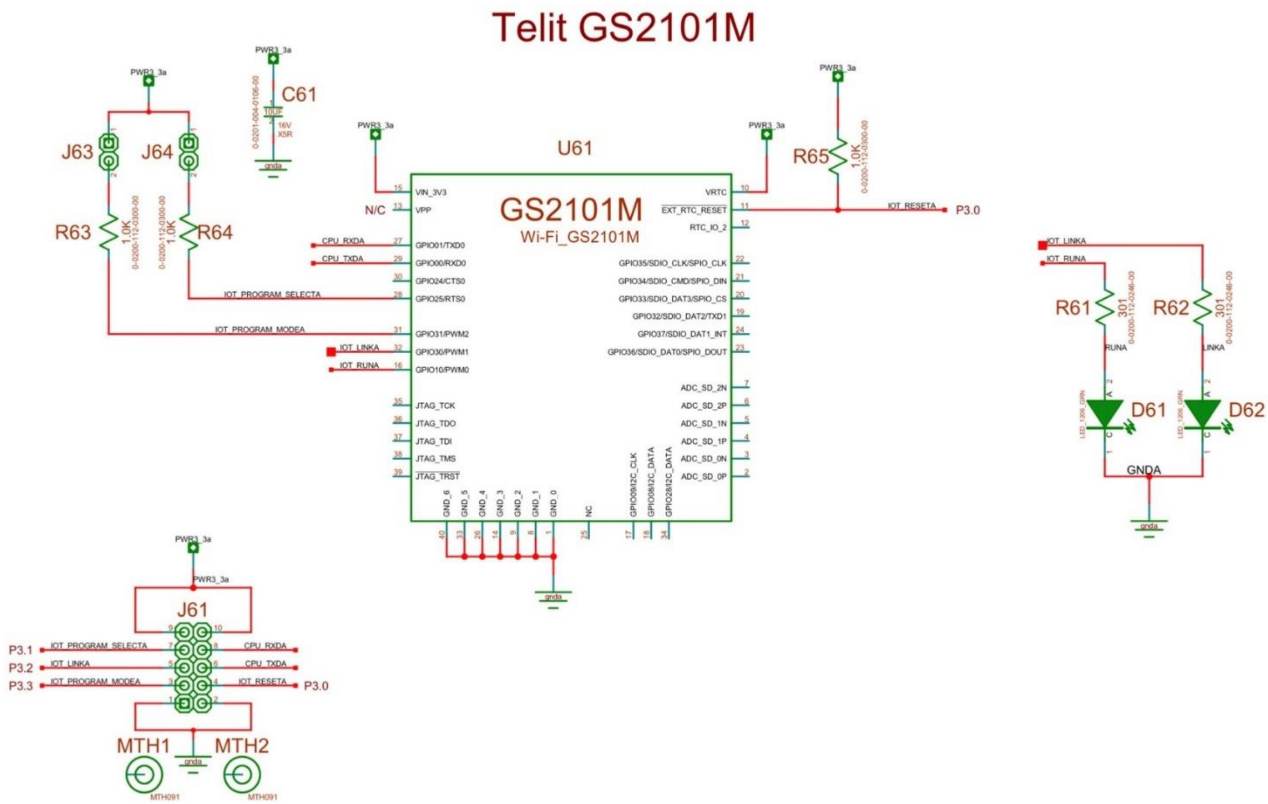

Figure 4.6 IOT

5. Power Analysis

The 3-wheeled motorcycle is powered by a Turnigy 4000mAh 2S 30C XT-60 LiPo battery pack. In an idle state, the motorcycle draws an average of 200 mA of current. When both motors are in use and the IR LED is on, the motorcycle draws an average of 400 mA of current. Using these values, the motorcycle will last 20 hours in an idle state from a freshly charged battery pack. Under full load, the motorcycle will last 10 hours from a freshly charged battery pack. Using a typical use scenario of 70% full load and 30% idle, the motorcycle will last approximately 11.7647 hours from a freshly charged battery pack.

6. Test Process

6.1. Power System Test

The power system board was tested using an oscilloscope set to read 3.3 V and a power supply with an output of 5 V and 0.020 A. The purpose of the power supply is to simulate the voltage output of the battery pack which was attached later. To test the power system, attach the ground of the power supply to one of the GND connections on the power board, attach the 5 V output of the power supply to the J10 connection on the power board, attach the ground of the oscilloscope to one of the GND connections on the power board, and attach the positive probe to the J12 connection on the power board. The connections to the board are illustrated in figure 6.1. A reading of 3.30 V to 3.36 V on the oscilloscope is acceptable for this application.

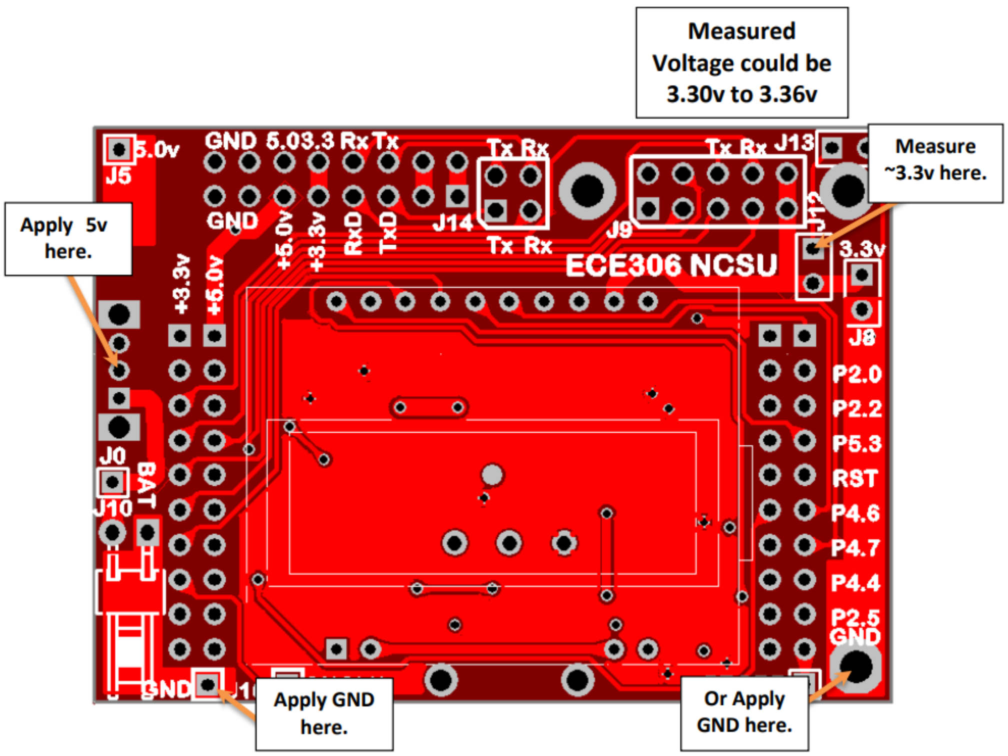

Figure 6.1 Power System Test Diagram

6.2. Battery Pack Test

The operating system of the MSP430 as defined in main.c is written so that when the board is driven with power indicator LED's light up in sequence. A successful installation of the battery pack is evidenced by the LED's lighting up. If the LED's are not lighting up, then the voltage of the battery can be tested across the two leads using a voltmeter to determine whether the battery has enough potential to power the circuit.

6.3. LCD Test

To test the LCD, the board must be first powered on. Once the board is powered on the LCD should display a predefined set of characters. (These are characters are defined in the main.c file). When the board is powered on, the display should start to show the status of the IR emitter. The status of the IR emitter can be changed by a press of a switch. The display should properly show the values read by the IR Detectors. Depending on if the car is on a white surface or a black surface, the values shown on the display will change. In addition to showing the values of the IR emitter, the display should also show if the car is on the right, left, or on top of the black line.

6.4. Motor Connection Test

The motors connection to the MCU was tested by seeing if the motors activated upon receiving power from the MCU. A fraction of code for port six needed to be modified in order to enable the motors to be turned on.

The car first is put into the middle of the 3 ft circle. Then a switch is presses. When this switch is pressed the motors should start spinning forward. They should keep on spinning till the IR detectors see the black line. Then one motor should spin forward and one should spin backwards, depending on direction the car is in when it hits the black line. The motors should then stop spinning when the car is parallel to the black line.

6.5. PWM Test

By changing the configurations of port one from GPIOs to functions, we can access a more reliable method of movement configuration that is much smoother than our previous method. These modifications also paved the way to be able to reliable allow the car to move in reverse as well. This improvement was tested by having the car move forward, backwards, spin clockwise and counterclockwise until it reached the black line. In addition, these different motions were tested at different speeds. This was done by using different PWM values. Higher values make the motors spin faster, while smaller values made them move slower.

6.6. Serial Communications Test

The serial communication was tested using an Analog Discovery 2 to transmit serial signals to and from the MSP430. The UCA0 Tx and Rx pins on J9 were connected to the Tx and Rx cables from the Analog Discovery 2. On the Analog Discovery 2, UART was the protocol selected, the BAUD rate was configured, and stop bits were chosen to match the MSP430. On the motorcycle, “Waiting” was displayed on line 1 to indicate that it is ready to receive a message. The baud rate was also displayed on line 3 of the display. A transmission was initiated from the Analog Discovery 2 and, on the press of a switch, the transmission was sent back to the Analog Discovery 2. This call and response was repeated 6 times, changing the baud rate between 115,200 Hz and 460,800 Hz intermittently and ensuring the correct information was being sent.

6.7. Autonomous

This test checks if the car can travel properly once the autonomous command is given after the car reaches the number 8. The first part of the test is to make sure the car can travel in a proper curve, from eight to the autonomous course. (Tests of different PWM values were always done at or near full batteries) For this part different PWM values were used to see which ones resulted in the most desirable arc. The next part of the test makes sure that the car can follow the course properly. Again, different PWM values were used. In addition to different PWM values, different ADC values for black lines were also used, to check which one yielded the best results.

6.8. IR Test

The IR was tested using a black line course made on white paper. The two emitters attached to the car would output a light source against the black line to be received by the sensor in between both emitters. From there, the code would interpret if the car was within the solid black line or had veered off onto the white paper. If it enters the white section of the test, then the car would self-adjust until it has corrected itself back onto the black line to continue movement forward. This would continue until the car had self-navigated around a circle twice and with use of a timer, would then rotate towards the circle and move inside of it to finish the test.

6.9. IOT Test

The IOT tests consists of moving the car through an obstacle course using commands sent to the car over Wi-Fi. The course was built to have obstacles preventing a straight line between point A and point B of the course. Point A and B were blocked off on two walls to prevent easy access from all directions.

7. Software

7.1. Main Loop

The main loop (main.c) controls the motorcycle by calling functions needed to accomplish the overall goal of operation. First, necessary global variables are declared for use in various functions across multiple files. Next, initialization procedures are ran in order to make sure intended settings are configured properly. Following that, the display is initialized to display whatever information is needed relative to the purpose of operation. Finally, the system processes needed to operate the motorcycle are ran in a loop in order to realize a modular coding approach.

7.1.1. Update_Display_Process() Explanation

This function is responsible for updating global variables necessary for the operation of the LCD. This must be done every 200ms. If desired, this function will also increment and display a time on the display by converting a global time variable from hex to BCD to ADCII. The time is shown on line 4 of the display.

7.1.2. IOT_Process() Explanation

This function is responsible for taking the IOT out of its reset state one second after the motorcycle is powered on. Is desired, the function will also transmit a command to the IOT module to initialize a port to use in order to communicate with the IOT module from a wireless terminal.

7.1.3. Movement_Process() Explanation

This function is responsible for handling the automated movement of the car started by certain commands from the IOT module. When the command to perform the automatic routine is given, the motorcycle turns right and traverses towards a black line course. When the motorcycle arrives at the course, the motorcycle will intercept the line and follow it until an exit command is received. When an exit command is received, the motorcycle will turn away from the course and drive three feet from the course. Throughout the routine process, the progress of the vehicle is shown on the display.

7.1.4. Command_Process() Explanation

This function is responsible for interpreting commands received from the IOT module. Commands available to be received include forward, backward, right, left, auto, exit, position, and IR_LED. These commands are then executed in the order they are received. To handle the reception of multiple commands in a short period of time, a command buffer is utilized to hold multiple commands in a queue at any given time.

7.1.5. Serial_Process() Explanation

This function is responsible for controlling the buffers responsible for serial communication operation and detecting received characters from the serial interrupt. When a character is received from the serial receive interrupt, a buffer is incremented, and this is detected by the serial process. The character is then taken into the process and interpreted in order to take the correct action necessary given the characters received.

7.2. Ports Explanation

The ports configuration file (ports.c), configures the ports to their desired states. These ports will be used in various ways by the MSP430. For example, some ports will be used as GPIO's, while other ports will be used as functions. When the main function starts to run, the Init_Ports() function is called. This function is used to call functions that initialize the ports. These functions go from Init_Ports1() all the way to Init_Ports6().

7.3. ADC Explanation

For the use of the emitters and detectors attached to our car, the code would need to first be accessed through the main function. From there, the main function calls on the Init_ADC function to effectively start the clocks, timers, emitter access and detector access that will be used to manipulate the data gathered. From then forward, all data collected is done so by means of interrupts. Those interrupts place the data received from the detectors and then sends them to HEXtoBCD function, which allows them to be displayed onto the LCD by manipulating the data to be in a suitable state for displaying the numerical value. After doing so, the code directs itself to return to the main function to continue execution prior to the interrupt until another interrupt is triggered. This will then create a cycle of data collection and data manipulation until the system is turned off.

7.4. Timers Explanation

The Init_Timer_B0 and Init_Timer_B3 functions serve to initialize timers which are used for general timer operations throughout the device's software and PWM, respectively. Both were configured to use the SM_CLK as their source clock. Timer B0 is set to work off of a capture compare register set with an interval of 50ms and Timer B3 uses multiple capture compare registers to set multiple intervals at which the vehicle motors can be pulsed on and off to regulate speed.

The capture-compare registers work by a predetermined interval which corresponds to a predetermined amount of time. This interval determines how frequently actions in the corresponding timer interrupts occur.

7.5. Serial Interrupts Explanation

Serial interrupts consist of 4 interrupts. 2 interrupts for port A0 and 2 interrupts for port A1. The two interrupts for each port consist of, an interrupt for receiving characters, and an interrupt for transmitting a character. The transmit interrupt for both ports is blank. The receive interrupt for both ports is populated for both ports. The receive interrupt only activates when a character is received on the UCA*RXBUF.

When the interrupt first starts, the current index for the ring buffer is stored in a temporary variable. Then the character received on the RXBUF is stored into the ring buffer. Before the interrupt ends, a check is made to see if the index for the ring buffer has exceeded the size of the ring buffer. If this is the case, then the index is reset back to 0.

The Init_Serial() function is used to initialize the serial communication properties. It is used to reset the USB_Char ring buffers, read and write integers, the receiving properties, and the initial baud rate that are used within the interrupts associated with serial communication. It also enables the serial interrupt flag so that it may begin receiving messages.

8. Flow Chart

8.1. Main Blocks

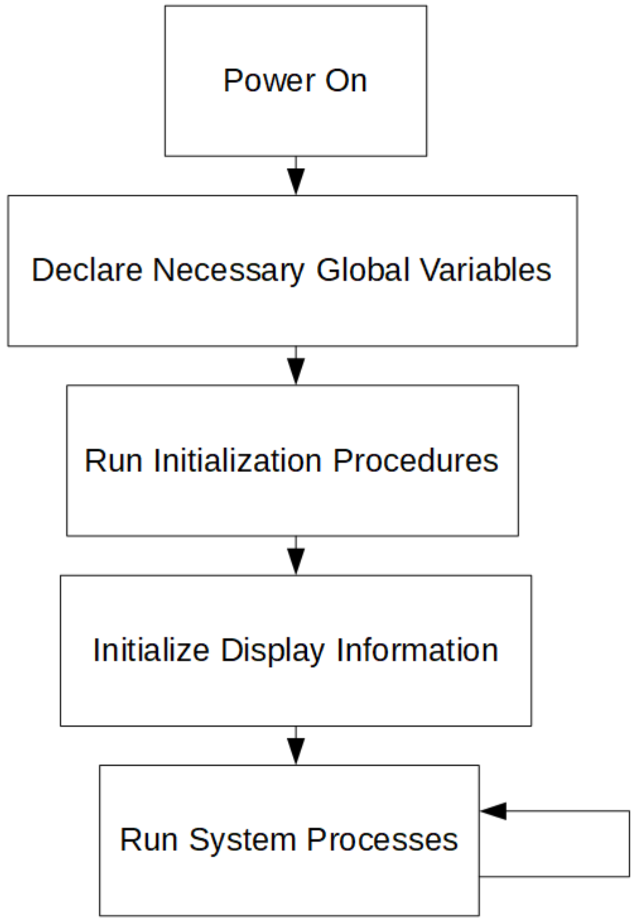

Figure 8.1 Main Loop Flowchart

8.1.1. Update_Display_Process()

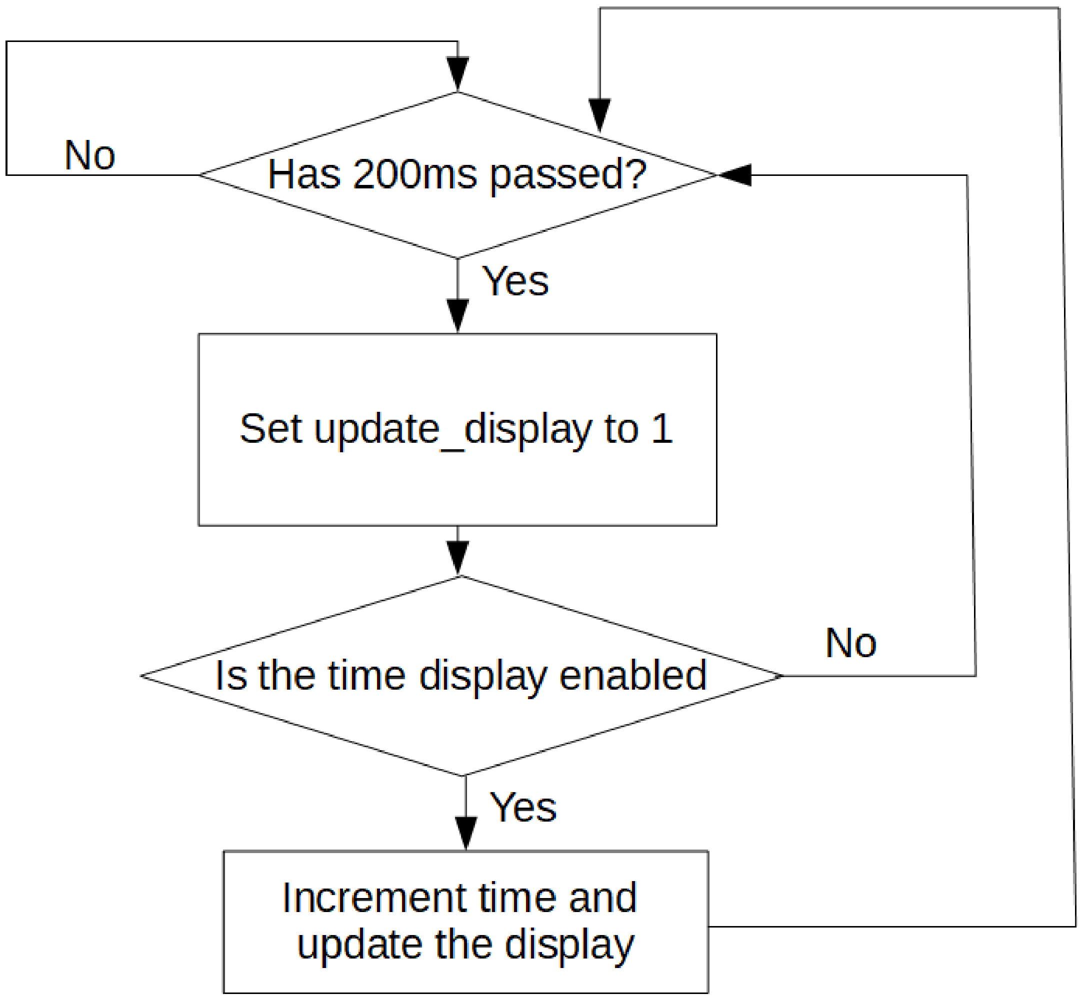

Figure 8.2 Update_Display_Process() Flowchart

8.1.2. IOT_Process()

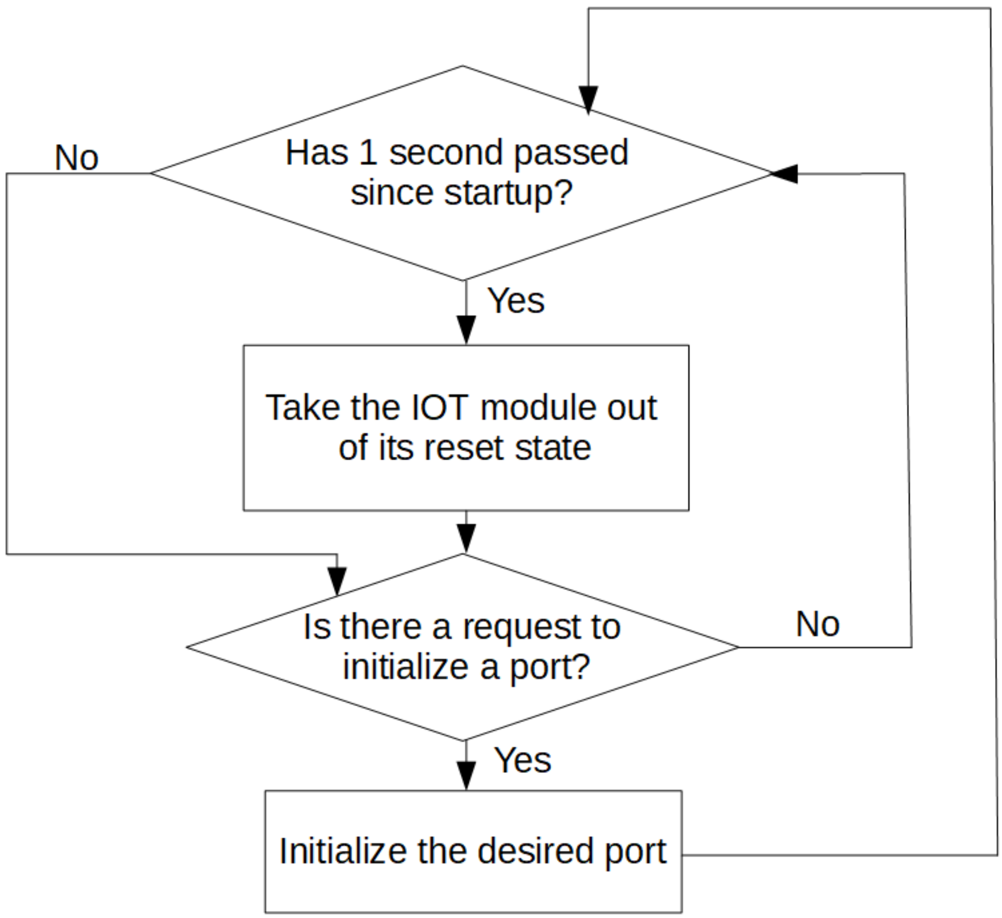

Figure 8.3 IOT_Process() Flowchart

8.1.3. Movement_Process()

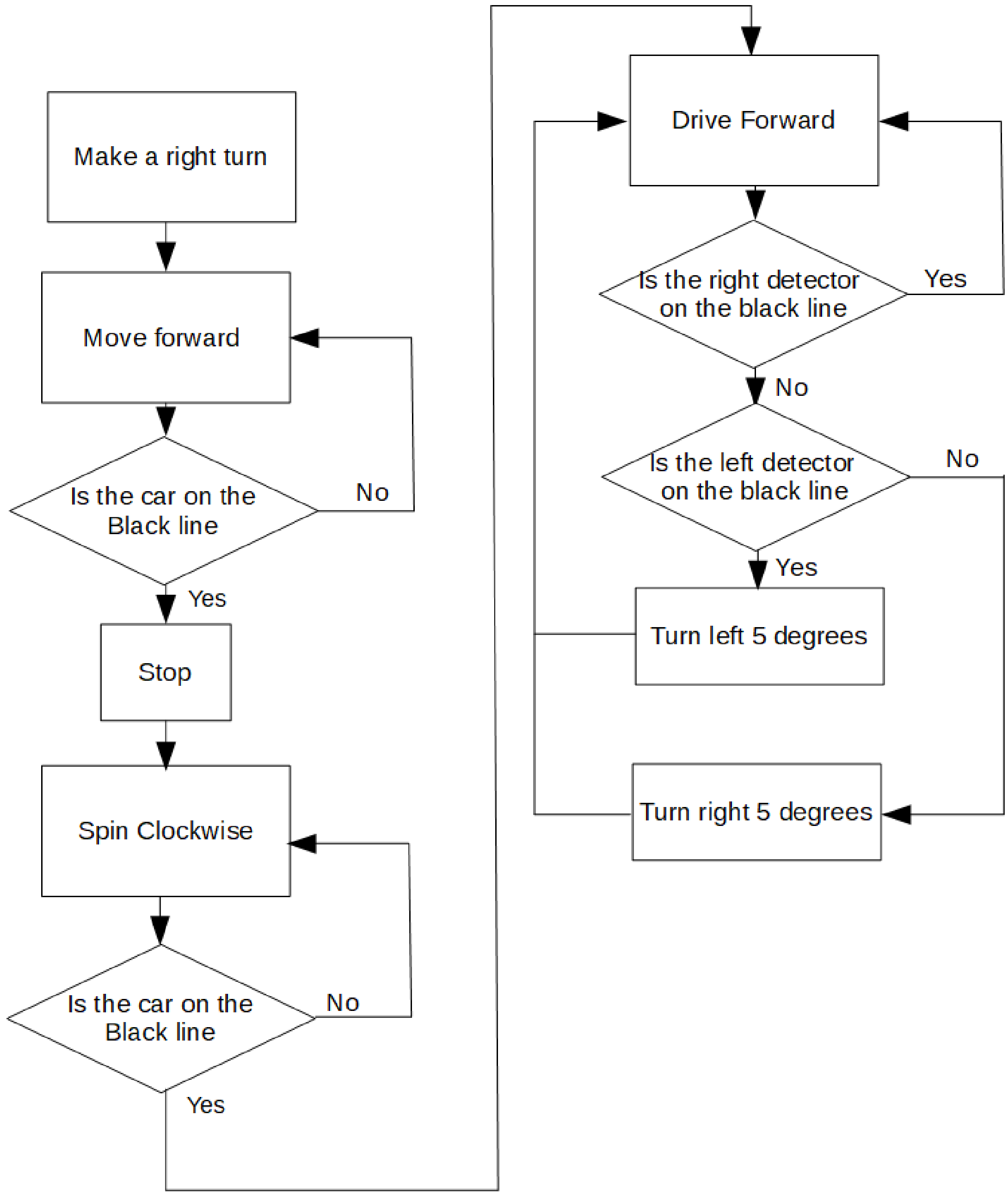

Figure 8.4 Movement_Process() Flowchart

8.1.4. Command_Process()

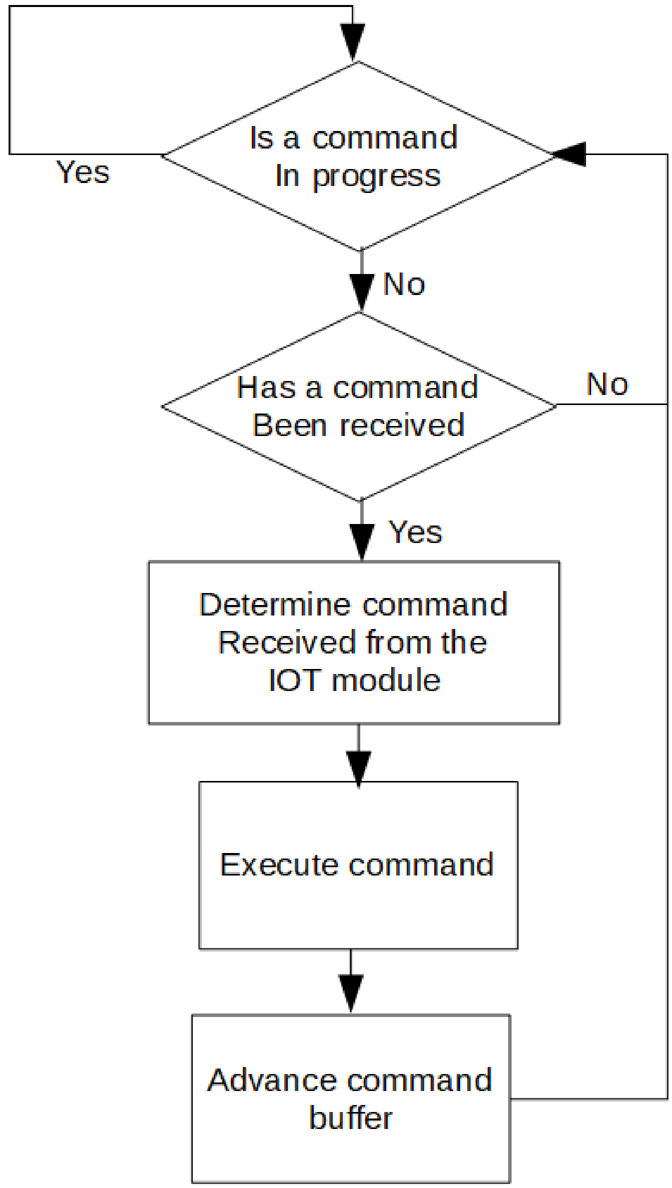

Figure 8.5 Command_Process() Flowchart

8.1.5. Serial_Process()

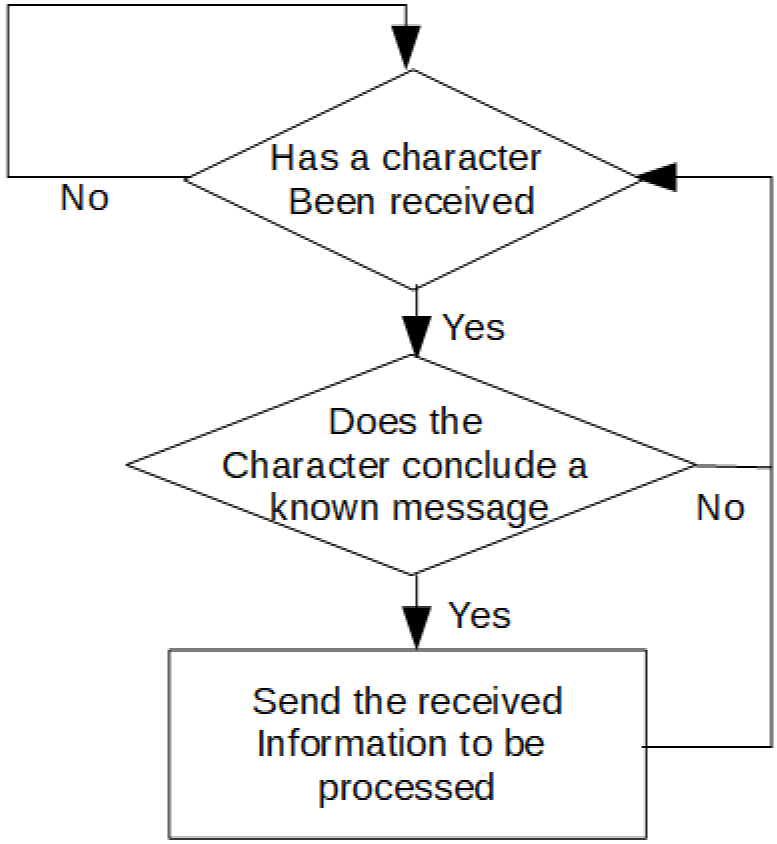

Figure 8.6 Serial_Process() Flowchart

9. Software Listing

9.1. Main.c

```
//------------------------------------------------------------------------------
//
//  Description: This file contains the Main Operating System
//
//
//  Ian Hellmer
//  Mar 2020
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.11.2)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

  // Global Variables
volatile char slow_input_down;
extern char display_line[LINES][CHAR_SPACES];
extern char *display[LINES];
unsigned char display_mode;
extern volatile unsigned char display_changed;
unsigned int test_value;
char chosen_direction;
char change;
unsigned volatile int display_state = RESET_STATE;

void main(void){
//------------------------------------------------------------------------------
// Main Program
// This is the main routine for the program. Execution of code starts here.
// The operating system is Back Ground Fore Ground.
//
//------------------------------------------------------------------------------
// Disable the GPIO power-on default high-impedance mode to activate
// previously configured port settings
  PM5CTL0 &= ~LOCKLPM5;
  Init_Ports();                            // Initialize Ports
  Init_Clocks();                           // Initialize Clock System
  Init_Conditions();                      // Initialize Variables and Initial Conditions
  Init_Timers();                           // Initialize Timers
  Init_LCD();                              // Initialize LCD
  Init_ADC();                              // Initialize ADC
  Init_Serial_UCA0(BAUD_115200);           // Initialize Serial Port for USB
  Init_Serial_UCA1(BAUD_115200);

  change_IOT_reset = RESET_STATE;
  set_display(" WAITING  ", " STARTUP  ", "          ", "FOR INPUT ");

//------------------------------------------------------------------------------
// Begining of the "While" Operating System
//------------------------------------------------------------------------------

  while(ALWAYS) {
      Movement_Process();
```

```
      Update_Display_Process();
      Display_Process();
      Serial_Process();
      IOT_Process();
      Command_Process();
  }

//--------------------------------------------------------------------------------
}
```

9.1.1. Update_Display_Process()

```
void Update_Display_Process(void) {
//--------------------------------------------------------------------------------
// Updates display every 200ms and updates display after a change
//--------------------------------------------------------------------------------

  unsigned int tmp, tmp1;

  if(display_state > CHANGE) {
    update_display = YES;
    display_state = RESET_STATE;
    if(time_display_enable) {
      current_time = current_time + TWO_HUNDRED_MS;
      tmp = HEXtoBCD(current_time);

      tmp1 = tmp & FIRST_NIBBLE_MASK;
      time[TIME_DECIMAL_DIGIT] = ASCII_START + tmp1;
      tmp1 = tmp & SECOND_NIBBLE_MASK;
      tmp1 = tmp1 >> SECOND_NIBBLE_SHIFT;
      time[TIME_ONES_DIGIT] = ASCII_START + tmp1;
      tmp1 = tmp & THIRD_NIBBLE_MASK;
      tmp1 = tmp1 >> THIRD_NIBBLE_SHIFT;
      time[TIME_TENS_DIGIT] = ASCII_START + tmp1;
      tmp1 = tmp & FOURTH_NIBBLE_MASK;
      tmp1 = tmp1 >> FOURTH_NIBBLE_SHIFT;
      time[TIME_HUNDREDS_DIGIT] = ASCII_START + tmp1;

      strcpy(display_line[LINE_FOUR], time);           // Update Time on Display
      update_string(display_line[LINE_FOUR], STRING_FOUR);
      display_changed = YES;
    }
  }
//--------------------------------------------------------------------------------
}
```

9.1.2. IOT_Process()

```
void IOT_Process(void) {
//------------------------------------------------------------------------------
// Controls IOT reset process and sends a command to the IOT to initialize port
// 3141 when requested.
//------------------------------------------------------------------------------
  if(!IOT_reset_changed && (change_IOT_reset > ONE_SECOND_RESET)) {
    P5OUT |= IOT_RESET;                  // Set IOT_RESET Off [Low]
    IOT_reset_changed = YES;
  }

  if(initialize_port) {
    strcpy(UCA0_transmit_message, "AT+NSTCP=3141,1");
    transmit_UCA0 = YES;
    initialize_port = RESET_STATE;
  }
//------------------------------------------------------------------------------
}
```

9.1.3. Movement_Process()

```
void Movement_Process(void) {
//------------------------------------------------------------------------------
// State machine for movement routine
//------------------------------------------------------------------------------
  switch(ROUTINE_STATE) {
      case(RESET_STATE):
        break;
      case(INITIAL_TURN):
          Spin_CW();
          movement_state = RESET_STATE;
          ROUTINE_STATE = INITIAL_TURN_END;
          set_display_line(" BL START ", LINE_ONE);
        break;
      case(INITIAL_TURN_END):
        if(movement_state > TURN_90_TIME_CW) {
          Stop_Movement();
          movement_state = RESET_STATE;
          ROUTINE_STATE = WAIT_BUFFER_INITIAL;
        }
        break;
      case(WAIT_BUFFER_INITIAL):
        if(movement_state > ONE_SECOND) {
          Forward_Movement();
          movement_state = RESET_STATE;
          ROUTINE_STATE = WAIT_BUFFER;
        }
        break;
      case(WAIT_BUFFER):
          if(movement_state > TWO_SECONDS) ROUTINE_STATE = FORWARD_STOP;
        break;
      case(FORWARD_STOP):
        if(((ADC_Left_Detect    >    line_threshold)    ||    (ADC_Right_Detect    >
line_threshold))) {
          Stop_Movement();
          ROUTINE_STATE = WAIT_BUFFER_1;
          movement_state = RESET_STATE;
          set_display_line("INTERCEPT ", LINE_ONE);
```

```
        }
        break;
      case(WAIT_BUFFER_1):
        if(movement_state >= TWO_SECONDS) {
          ROUTINE_STATE = SPIN_AND_SEARCH;
          movement_state = RESET_STATE;
        }
        break;
      case(SPIN_AND_SEARCH):
        Spin_CCW_SLOW();
        if(((ADC_Left_Detect    >    line_threshold)    ||    (ADC_Right_Detect    >
line_threshold))) {
          Stop_Movement();
          ROUTINE_STATE = WAIT_BUFFER_2;
          movement_state = RESET_STATE;
        }
        break;
      case(WAIT_BUFFER_2):
        Stop_Movement();
        if(movement_state >= TWO_SECONDS) {
          set_display_line("BL TRAVEL ", LINE_ONE);
          ROUTINE_STATE = MOVE_TO_EDGE;
          movement_state = RESET_STATE;
        }
        break;
      case(MOVE_TO_EDGE):
        Forward_Movement_Slow();
        if(ADC_Left_Detect < line_threshold) {
          ROUTINE_STATE = TRAVERSE_CIRCLE;
          movement_state = RESET_STATE;
        }
        break;
      case(TRAVERSE_CIRCLE):
        if((ADC_Right_Detect < line_threshold)) {
          if(ADC_Left_Detect > SENSITIVE_THRESHOLD) {
            Turn_L_Five_Degrees();
            ROUTINE_STATE = WAIT_FOR_L_TURN;
          }
          else {
            Turn_R_Five_Degrees();
            ROUTINE_STATE = WAIT_FOR_R_TURN;
          }
        }
        break;
      case(WAIT_FOR_R_TURN):
        Turn_R_Five_Degrees();
        if(done_turning_R) {
          if(movement_state > TEN_SECONDS) {
            set_display_line("BL CIRCLE ", LINE_ONE);
          }
          Forward_Movement_Slow();
          done_turning_R = RESET_STATE;
          ROUTINE_STATE = TRAVERSE_CIRCLE;
        }
        break;
      case(WAIT_FOR_L_TURN):
        Turn_L_Five_Degrees();
        if(done_turning_L) {
```

```
          if(movement_state > TEN_SECONDS) {
            set_display_line("BL CIRCLE ", LINE_ONE);
          }
          Forward_Movement_Slow();
          done_turning_L = RESET_STATE;
          ROUTINE_STATE = TRAVERSE_CIRCLE;
        }
        break;
      case(WAIT_BUFFER_3):
        Stop_Movement();
        set_display_line(" BL EXIT  ", LINE_ONE);
        if(movement_state > TWO_SECONDS) {
          Spin_CCW_SLOW();
          movement_state = RESET_STATE;
          ROUTINE_STATE = TURN_BACK;
        }
        break;
      case(TURN_BACK):
        if(movement_state > TURN_90_TIME_CCW) {
          Stop_Movement();
          movement_state = RESET_STATE;
          ROUTINE_STATE = WAIT_BUFFER_4;
        }
        break;
      case(WAIT_BUFFER_4):
        if(movement_state > TWO_SECONDS) {
          Forward_Movement();
          movement_state = RESET_STATE;
          ROUTINE_STATE = FIND_CENTER;
        }
        break;
      case(FIND_CENTER):
        if(movement_state > THREE_SECONDS) {
          Stop_Movement();
          TB0CCTL2 &= ~CCIE;                    // CCR2 disable interrupt
          ROUTINE_STATE = RESET_STATE;
          routine_engaged = RESET_STATE;
          time_display_enable = RESET_STATE;
          set_display_line(" BL STOP  ", LINE_ONE);
        }
        break;
      default:
        break;
  }
//-------------------------------------------------------------------------------
}
```

9.1.4. Command_Process()

```
void Command_Process(void) {
//------------------------------------------------------------------------------
// State machine for commands routine
//------------------------------------------------------------------------------

  if(current_commands) {
    if(Command_Buffer[current_command][DIRECTION_CHAR] == 'T') {
      movement_state = RESET_STATE;
      ROUTINE_STATE = WAIT_BUFFER_3;
      TB0CCTL2 |= CCIE;                   // CCR2 enable interrupt
      routine_engaged = YES;
      current_command++;
      if(current_command >= COMMAND_LINES) {
        current_command = BEGINNING;     // Circular buffer back to beginning
      }
      current_commands--;
    }
    if(!(forward_command_engaged        ||        backwards_command_engaged        ||
left_command_engaged || right_command_engaged || routine_engaged)) {
      switch(Command_Buffer[current_command][DIRECTION_CHAR]) {
        case('F'):
          movement_value              =              movement_value              +
((Command_Buffer[current_command][HUNDREDS_CHAR] - ASCII_START)*HUNDRED);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][TENS_CHAR] - ASCII_START)*TEN);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][ONES_CHAR] - ASCII_START));
          forward_command_engaged = YES;
          break;
        case('B'):
          movement_value              =              movement_value              +
((Command_Buffer[current_command][HUNDREDS_CHAR] - ASCII_START)*HUNDRED);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][TENS_CHAR] - ASCII_START)*TEN);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][ONES_CHAR] - ASCII_START));
          backwards_command_engaged = YES;
          break;
        case('R'):
          movement_value              =              movement_value              +
((Command_Buffer[current_command][HUNDREDS_CHAR] - ASCII_START)*HUNDRED);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][TENS_CHAR] - ASCII_START)*TEN);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][ONES_CHAR] - ASCII_START));
          right_command_engaged = YES;
          break;
        case('L'):
          movement_value              =              movement_value              +
((Command_Buffer[current_command][HUNDREDS_CHAR] - ASCII_START)*HUNDRED);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][TENS_CHAR] - ASCII_START)*TEN);
          movement_value              =              movement_value              +
((Command_Buffer[current_command][ONES_CHAR] - ASCII_START));
          left_command_engaged = YES;
          break;
        case('A'):
```

```
          ROUTINE_STATE = INITIAL_TURN;
          TB0CCTL2 |= CCIE;                  // CCR2 enable interrupt
          routine_engaged = YES;
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          break;
        case('K'):
          on_number[NUMBER_POSITION] = Command_Buffer[current_command][ONES_CHAR];
          set_display_line(on_number, LINE_ONE);
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          break;
        case('I'):
          if(Command_Buffer[current_command][ONES_CHAR] - ASCII_START) {
            set_display_line("IR_LED ON ", LINE_ONE);
            P3OUT |= IR_LED;
          }
          else {
            set_display_line("IR_LED OFF", LINE_ONE);
            P3OUT &= ~IR_LED;
          }
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          break;
        case('T'):
          break;
        default: break;
      }
      time_display_enable = YES;
      current_commands--;
    }
  }

  if(forward_command_engaged) {
    switch(forward_movement_state) {
      case(RESET_STATE):
        set_display_line(" FORWARD  ", LINE_ONE);
        Forward_Movement();
        movement_time = RESET_STATE;
        forward_movement_state++;
        break;
      case(STATE_1):
        if(movement_time > movement_value) {
          Stop_Movement();
          forward_movement_state = STATE_2;
        }
        break;
      case(STATE_2):
        if(movement_time > movement_value + FIFTY_MSEC) {
          forward_movement_state = RESET_STATE;
          forward_command_engaged = RESET_STATE;
          current_command++;
```

```
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          movement_value = RESET_STATE;
        }
        break;
      default:
        break;
    }
  }

  if(backwards_command_engaged) {
    switch(backwards_movement_state) {
      case(RESET_STATE):
        set_display_line(" BACKWARD ", LINE_ONE);
        Reverse_Movement();
        movement_time = RESET_STATE;
        backwards_movement_state++;
        break;
      case(STATE_1):
        if(movement_time > movement_value) {
          Stop_Movement();
          backwards_movement_state = STATE_2;
        }
        break;
      case(STATE_2):
        if(movement_time > movement_value + FIFTY_MSEC) {
          backwards_movement_state = RESET_STATE;
          backwards_command_engaged = RESET_STATE;
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          movement_value = RESET_STATE;
        }
        break;
      default:
        break;
    }
  }

  if(left_command_engaged) {
    switch(left_movement_state) {
      case(RESET_STATE):
        set_display_line("   LEFT   ", LINE_ONE);
        Spin_CCW();
        movement_time = RESET_STATE;
        left_movement_state++;
        break;
      case(STATE_1):
        if(movement_time > movement_value) {
          Stop_Movement();
          left_movement_state = STATE_2;
        }
        break;
      case(STATE_2):
        if(movement_time > movement_value + FIFTY_MSEC) {
          left_movement_state = RESET_STATE;
```

```
          left_command_engaged = RESET_STATE;
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          movement_value = RESET_STATE;
        }
        break;
      default:
        break;
    }
  }

  if(right_command_engaged) {
    switch(right_movement_state) {
      case(RESET_STATE):
        set_display_line("  RIGHT   ", LINE_ONE);
        Spin_CW();
        movement_time = RESET_STATE;
        right_movement_state++;
        break;
      case(STATE_1):
        if(movement_time > movement_value) {
          Stop_Movement();
          right_movement_state = STATE_2;
        }
        break;
      case(STATE_2):
        if(movement_time > movement_value + FIFTY_MSEC) {
          right_movement_state = RESET_STATE;
          right_command_engaged = RESET_STATE;
          current_command++;
          if(current_command >= COMMAND_LINES) {
            current_command = BEGINNING;     // Circular buffer back to beginning
          }
          movement_value = RESET_STATE;
        }
        break;
      default:
        break;
    }
  }

//------------------------------------------------------------------------------
}
```

9.1.5. Serial_Process()

```
void Serial_Process(void) {
//------------------------------------------------------------------------------
// Handles serial processes such as receiving and transmitting messages
//------------------------------------------------------------------------------
  unsigned int write_point_UCA0 = usb_rx_ring_wr_UCA0;
  unsigned int read_point_UCA0 = usb_rx_ring_rd_UCA0;
  unsigned int write_point_UCA1 = usb_rx_ring_wr_UCA1;
  unsigned int read_point_UCA1 = usb_rx_ring_rd_UCA1;

  int i = RESET_STATE;

  if(write_point_UCA1 != read_point_UCA1) {
  }

  if(UCA1_messaged_received) {
    UCA1_messaged_received = RESET_STATE;
  }

  if(write_point_UCA0 != read_point_UCA0) {
    Process_Char_Rx_UCA0[process_ring_rd_UCA0] = USB_Char_Rx_UCA0[read_point_UCA0];
    usb_rx_ring_rd_UCA0++;
    if(usb_rx_ring_rd_UCA0 >= (SMALL_RING_SIZE)) {
      usb_rx_ring_rd_UCA0 = BEGINNING;      // Circular buffer back to beginning
    }
    UCA1TXBUF = Process_Char_Rx_UCA0[process_ring_rd_UCA0];
    switch(UCA0_state_0) {
      case(RESET_STATE):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'a') {
          UCA0_state_0 = STATE_1;
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        }
        else if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'v') {
          UCA0_state_0 = DISCONNECT_CASE;
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        }
        else if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'C') {
          UCA0_state_0 = RECONNECT_CASE;
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        }
        else if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '^') {
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;     // Circular buffer back to beginning
          }
          UCA0_state_0 = STATE_6;
        }
        else {
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        }
        break;
      case(RECONNECT_CASE):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'C') {
```

```
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
          strcpy(UCA0_transmit_message, "AT+NSTAT=?");
          transmit_UCA0 = YES;
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        break;
      case(DISCONNECT_CASE):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'e') {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
          set_display_line("DISCONNECT", LINE_TWO);
          set_display_line("DISCONNECT", LINE_THREE);
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        break;
      case(STATE_1):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'd') {
          UCA0_state_0 = STATE_2;
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        break;
      case(STATE_2):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'd') {
          UCA0_state_0 = STATE_3;
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
```

```
          process_ring_rd_UCA0 = BEGINNING;
        break;
      case(STATE_3):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'r') {
          UCA0_state_0 = STATE_4;
        }
        else {
          for(i = RESET_STATE; i < (UCA0_state_0 + BUFFER_BUFFER); i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
          Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
          process_ring_rd_UCA0 = BEGINNING;
        break;
      case(STATE_4):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '=') {
          UCA0_state_0 = STATE_5;
        }
        else {
          for(i = RESET_STATE; i < (UCA0_state_0 + BUFFER_BUFFER); i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        Process_Char_Rx_UCA0[process_ring_rd_UCA0] = NULL;
        process_ring_rd_UCA0 = BEGINNING;
        break;
      case(STATE_5):
        if((Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '.') ||
(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == ' ') ||
((Process_Char_Rx_UCA0[process_ring_rd_UCA0] >= '0') &&
(Process_Char_Rx_UCA0[process_ring_rd_UCA0] <= '9'))) {
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == 'G') {
          process_the_buffer_1 = YES;
          use_process_2 = YES;
          UCA0_state_0 = RESET_STATE;
          initialize_port = YES;
          found_ip = YES;
        }
        break;
      case(STATE_6):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '^') {
          process_the_buffer_1 = YES;
          use_process_2 = YES;
          UCA0_state_0 = RESET_STATE;
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '1') {
          UCA0_state_0 = STATE_7;
```

```
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        break;
      case(STATE_7):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '3') {
          UCA0_state_0 = STATE_8;
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        break;
      case(STATE_8):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '6') {
          UCA0_state_0 = STATE_9;
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        break;
      case(STATE_9):
        if(Process_Char_Rx_UCA0[process_ring_rd_UCA0] == '8') {
          UCA0_state_0 = STATE_10;
          process_ring_rd_UCA0++;
          if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
            process_ring_rd_UCA0 = BEGINNING;    // Circular buffer back to beginning
          }
        }
        else {
          for(i   =   RESET_STATE;   i   <   (UCA0_state_0   +   BUFFER_BUFFER);   i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
          process_ring_rd_UCA0 = BEGINNING;
          UCA0_state_0 = RESET_STATE;
        }
        break;
```

```
      case(STATE_10):
        UCA0_state_0 = STATE_11;
        process_ring_rd_UCA0++;
        if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
          process_ring_rd_UCA0 = BEGINNING;     // Circular buffer back to beginning
        }
        break;
      case(STATE_11):
        UCA0_state_0 = STATE_12;
        process_ring_rd_UCA0++;
        if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
          process_ring_rd_UCA0 = BEGINNING;     // Circular buffer back to beginning
        }
        break;
      case(STATE_12):
        UCA0_state_0 = STATE_13;
        process_ring_rd_UCA0++;
        if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
          process_ring_rd_UCA0 = BEGINNING;     // Circular buffer back to beginning
        }
        break;
      case(STATE_13):
        process_the_buffer_1 = YES;
        use_process_2 = YES;
        UCA0_state_0 = RESET_STATE;
        process_ring_rd_UCA0++;
        if(process_ring_rd_UCA0 >= PROC_RING_SIZE) {
          process_ring_rd_UCA0 = BEGINNING;     // Circular buffer back to beginning
        }
        break;
      default:
        break;
    }
  }


  if(process_the_buffer_1) {
    if(found_ip) {
      find_ip_address(Process_Char_Rx_UCA0);
      found_ip = RESET_STATE;
    }
    else {
      switch(Process_Char_Rx_UCA0[FIRST_CHAR_RECEIVED]) {
        case('^'):
          switch(Process_Char_Rx_UCA0[PASSWORD_CHAR_1]) {
            case('^'):
              strcpy(UCA1_transmit_message, "\nI'm here");
              transmit_UCA1 = YES;
              break;
            case('F'):
              UCA0BRW = BRW_115200;                          // 115,200 Baud
              UCA0MCTLW = MCTLW_115200;
              set_display_line("  115200  ", LINE_ONE);
              break;
            case('S'):
              UCA0BRW = BRW_9600;                            // 9,600 Baud
              UCA0MCTLW = MCTLW_9600;
              set_display_line("   9600   ", LINE_ONE);
```

```
              break;
            default:
              if((Process_Char_Rx_UCA0[PASSWORD_CHAR_1]       ==       '1')       &&
(Process_Char_Rx_UCA0[PASSWORD_CHAR_2]            ==            '3')            &&
(Process_Char_Rx_UCA0[PASSWORD_CHAR_3]            ==            '6')            &&
(Process_Char_Rx_UCA0[PASSWORD_CHAR_4] == '8')) {
                if(current_commands < COMMAND_LINES) {
                  Command_Buffer[next_empty_command][COMMAND]                     =
Process_Char_Rx_UCA0[COMMAND_CHAR];
                  Command_Buffer[next_empty_command][COMMAND_VALUE_HUNDRED]       =
Process_Char_Rx_UCA0[COMMAND_VALUE_1];
                  Command_Buffer[next_empty_command][COMMAND_VALUE_TEN]           =
Process_Char_Rx_UCA0[COMMAND_VALUE_2];
                  Command_Buffer[next_empty_command][COMMAND_VALUE_ONE]           =
Process_Char_Rx_UCA0[COMMAND_VALUE_3];
                  next_empty_command++;
                  if(next_empty_command >= COMMAND_LINES) {
                    next_empty_command = BEGINNING;      // Circular buffer back to
beginning
                  }
                  current_commands++;
                }
              }
              break;
          }
          break;

        default:
          break;
      }
    }
    process_the_buffer_1 = RESET_STATE;
    for(i    =    RESET_STATE;    i    <    sizeof(Process_Char_Rx_UCA0);    i++)
Process_Char_Rx_UCA0[i] = NULL_HEX;
    process_ring_rd_UCA0 = RESET_STATE;
  }


  if(transmit_UCA0) {
    UCA0TXBUF = UCA0_transmit_message[RESET_STATE];
    UCA0IE |= UCTXIE;                // Enable TX interrupt
    UCA0_index = RESET_STATE;
    transmit_UCA0 = RESET_STATE;
  }


  if(transmit_UCA1) {
    UCA1TXBUF = UCA1_transmit_message[RESET_STATE];
    UCA1IE |= UCTXIE;                // Enable TX interrupt
    UCA1_index = RESET_STATE;
    transmit_UCA1 = RESET_STATE;
  }

//------------------------------------------------------------------------------
}
```

9.2. Ports.c

```
//------------------------------------------------------------------------------
//
//  Description: This file initializes ports
//
//
//  Jan 2020
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

void Init_Ports(void) {
//------------------------------------------------------------------------------
  Init_Port1();
  Init_Port2();
  Init_Port3(USE_GPIO);
  Init_Port4();
  Init_Port5();
  Init_Port6();

}

void Init_Port1(void){
//------------------------------------------------------------------------------
// Configure PORT 1
  P1SEL0 = SET_GPIO;                    // Set P1 operation to GPIO
  P1SEL1 = SET_GPIO;                    // Set P1 operation to GPIO
  P1DIR = SET_OUTPUTS;                      // Set P1 direction to output
  P1OUT = SET_LOW;                          // P1 set Low

// PIN 0
  P1SEL0 &= ~RED_LED;                       // RED_LED GPIO operation
  P1SEL1 &= ~RED_LED;                       // RED_LED GPIO operation
  P1DIR |= RED_LED;                         // Set RED_LED direction to output
  P1OUT &= ~RED_LED;                        // Set RED_LED Off [Low]

// PIN 1
  P1SEL0 |= A1_SEEED;                       // A1_SEEED operation
  P1SEL1 |= A1_SEEED;                       // A1_SEEED operation

// PIN 2
  P1SEL0 |= V_DETECT_L;                     // V_DETECT_L operation
  P1SEL1 |= V_DETECT_L;                     // V_DETECT_L operation

// PIN 3
  P1SEL0 |= V_DETECT_R;                     // V_DETECT_R operation
  P1SEL1 |= V_DETECT_R;                     // V_DETECT_R operation

// PIN 4
  P1SEL0 |= A4_SEEED;                       // A4_SEEED operation
  P1SEL1 |= A4_SEEED;                       // A4_SEEED operation

// PIN 5
```

```
  P1SEL0 |= V_THUMB;                    // V_THUMB operation
  P1SEL1 |= V_THUMB;                    // V_THUMB operation

// PIN 6
  P1SEL0 |= UCA0RXD;                    // UCA0RXD operation
  P1SEL1 &= ~UCA0RXD;                   // UCA0RXD operation

// PIN 7
  P1SEL0 |= UCA0TXD;                    // UCA0TXD operation
  P1SEL1 &= ~UCA0TXD;                   // UCA0TXD operation
//------------------------------------------------------------------------------
}

void Init_Port2(void){
//------------------------------------------------------------------------------
// Configure PORT 2
  P2SEL0 = SET_GPIO;                // Set P2 operation to GPIO
  P2SEL1 = SET_GPIO;                // Set P2 operation to GPIO
  P2DIR = SET_OUTPUTS;                  // Set P2 direction to output
  P2OUT = SET_LOW;                      // P2 set Low

// PIN 0
  P2SEL0 &= ~P2_0;                      // P2_0 GPIO operation
  P2SEL1 &= ~P2_0;                      // P2_0 GPIO operation
  P2DIR &= ~P2_0;                       // Direction = input

// PIN 1
  P2SEL0 &= ~P2_1;                      // P2_1 GPIO operation
  P2SEL1 &= ~P2_1;                      // P2_1 GPIO operation
  P2DIR &= ~P2_1;                       // Direction = input

// PIN 2
  P2SEL0 &= ~P2_2;                      // P2_2 GPIO operation
  P2SEL1 &= ~P2_2;                      // P2_2 GPIO operation
  P2DIR &= ~P2_2;                       // Direction = input

// PIN 3
  P2SEL0 &= ~SW2;                       // SW2 Operation
  P2SEL1 &= ~SW2;                       // SW2 Operation
  P2DIR &= ~SW2;                        // Direction = input
  P2OUT |= SW2;                         // Configure pullup resistor
  P2REN |= SW2;                         // Enable pullup resistor
  P2IES |= SW2;                         // P2.0 Hi/Lo edge interrupt
  P2IFG &= ~SW2;                        // Clear all P2.6 interrupt flags
  P2IE |= SW2;                          // P2.6 interrupt enabled

// PIN 4
  P2SEL0 &= ~P2_4;                      // P2_4 GPIO operation
  P2SEL1 &= ~P2_4;                      // P2_4 GPIO operation
  P2DIR &= ~P2_4;                       // Direction = input

// PIN 5
  P2SEL0 &= ~P2_5;                      // P2_5 GPIO operation
  P2SEL1 &= ~P2_5;                      // P2_5 GPIO operation
  P2DIR &= ~P2_5;                       // Direction = input

// PIN 6
  P2SEL0 &= ~LFXOUT;                    // LFXOUT Clock operation
```

```
  P2SEL1 |= LFXOUT;                        // LFXOUT Clock operation

// PIN 7
  P2SEL0 &= ~LFXIN;                        // LFXIN Clock operation
  P2SEL1 |= LFXIN;                         // LFXIN Clock operation
//------------------------------------------------------------------------------
}

void Init_Port3(char MODE){
//------------------------------------------------------------------------------
// Configure PORT 3
  P3SEL0 = SET_GPIO;                   // Set P3 operation to GPIO
  P3SEL1 = SET_GPIO;                   // Set P3 operation to GPIO
  P3DIR = SET_OUTPUTS;                     // Set P3 direction to output
  P3OUT = SET_LOW;                         // P3 set Low

// PIN 0
  P3SEL0 &= ~TEST_PROBE;                   // TEST_PROBE GPIO Operation
  P3SEL1 &= ~TEST_PROBE;                   // TEST_PROBE GPIO Operation
  P3DIR |= TEST_PROBE;                     // Direction = input
  P3OUT &= ~TEST_PROBE;

// PIN 1
  P3SEL0 |= CHECK_BAT;                     // CHECK_BAT Operation
  P3SEL1 |= CHECK_BAT;                     // CHECK_BAT Operation

// PIN 2
  P3SEL0 |= OA2N;                          // OA2N Operation
  P3SEL1 |= OA2N;                          // OA2N Operation

// PIN 3
  P3SEL0 |= OA2P;                          // OA2P Operation
  P3SEL1 |= OA2P;                          // OA2P Operation

// PIN 4
  if(MODE == USE_GPIO) {
    P3SEL0 &= ~P3_4;                       // P3_4 GPIO Operation
    P3SEL1 &= ~P3_4;                       // P3_4 GPIO Operation
    P3DIR &= ~P3_4;                        // Direction = input
  }
  else if(MODE == USE_SMCLK) {
    P3SEL0 |= SMCLK_OUT;                   // SMCLK_OUT Operation
    P3SEL1 &= ~SMCLK_OUT;                  // SMCLK_OUT Operation
    P3DIR |= SMCLK_OUT;                    // SMCLK_OUT Operation
  }

// PIN 5
  P3SEL0 &= ~IR_LED;                       // IR_LED GPIO Operation
  P3SEL1 &= ~IR_LED;                       // IR_LED GPIO Operation
  P3DIR |= IR_LED;                         // Direction = output
  P3OUT &= ~IR_LED;                        // Set IR_LED Off [Low]

// PIN 6
  P3SEL0 &= ~IOT_LINK;                     // IOT_LINK GPIO Operation
  P3SEL1 &= ~IOT_LINK;                     // IOT_LINK GPIO Operation
  P3DIR &= ~IOT_LINK;                      // Direction = input
  P3OUT &= ~IOT_LINK;                      // Condition = 0
```

```
// PIN 7
  P3SEL0 &= ~P3_7;                      // P3_7 GPIO Operation
  P3SEL1 &= ~P3_7;                      // P3_7 GPIO Operation
  P3DIR &= ~P3_7;                       // Direction = input
//------------------------------------------------------------------------------
}

void Init_Port4(void){
//------------------------------------------------------------------------------
// Configure PORT 4
  P4SEL0 = SET_GPIO;                // Set P4 operation to GPIO
  P4SEL1 = SET_GPIO;                // Set P4 operation to GPIO
  P4DIR = SET_OUTPUTS;                  // Set P4 direction to output
  P4OUT = SET_LOW;                      // P4 set Low

// PIN 0
  P4SEL0 &= ~RESET_LCD;                 // RESET_LCD GPIO operation
  P4SEL1 &= ~RESET_LCD;                 // RESET_LCD GPIO operation
  P4DIR |= RESET_LCD;                   // Set RESET_LCD direction to output
  P4OUT |= RESET_LCD;                   // Set RESET_LCD Off [High]

// PIN 1
  P4SEL0 &= ~SW1;                       // SW1 GPIO operation
  P4SEL1 &= ~SW1;                       // SW1 GPIO operation
  P4DIR &= ~SW1;                        // Direction = input
  P4OUT |= SW1;                         // Configure pullup resistor
  P4REN |= SW1;                         // Enable pullup resistor
  P4IES |= SW1;                      // P2.0 Hi/Lo edge interrupt
  P4IFG &= ~SW1;                     // Clear all P2.6 interrupt flags
  P4IE |= SW1;                       // P2.6 interrupt enabled

// PIN 2
  P4SEL0 |= UCA1TXD;                    // USCI_A1 UART operation
  P4SEL1 &= ~UCA1TXD;                   // USCI_A1 UART operation

// PIN 3
  P4SEL0 |= UCA1RXD;                    // USCI_A1 UART operation
  P4SEL1 &= ~UCA1RXD;                   // USCI_A1 UART operation

// PIN 4
  P4SEL0 &= ~UCB1_CS_LCD;               // UCB1_CS_LCD GPIO operation
  P4SEL1 &= ~UCB1_CS_LCD;               // UCB1_CS_LCD GPIO operation
  P4DIR |= UCB1_CS_LCD;                 // Set SPI_CS_LCD direction to output
  P4OUT |= UCB1_CS_LCD;                 // Set SPI_CS_LCD Off [High]

// PIN 5
  P4SEL0 |= UCB1CLK;                    // UCB1CLK SPI BUS operation
  P4SEL1 &= ~UCB1CLK;                   // UCB1CLK SPI BUS operation

// PIN 6
  P4SEL0 |= UCB1SIMO;                   // UCB1SIMO SPI BUS operation
  P4SEL1 &= ~UCB1SIMO;                  // UCB1SIMO SPI BUS operation

// PIN 7
  P4SEL0 |= UCB1SOMI;                   // UCB1SOMI SPI BUS operation
  P4SEL1 &= ~UCB1SOMI;                  // UCB1SOMI SPI BUS operation
//------------------------------------------------------------------------------
}
```

```
void Init_Port5(void){
//------------------------------------------------------------------------------
// Configure PORT 5
  P5SEL0 = SET_GPIO;                    // Set P5 operation to GPIO
  P5SEL1 = SET_GPIO;                    // Set P5 operation to GPIO
  P5DIR = SET_INPUTS;                       // Set P5 direction to input
  P5OUT = SET_LOW;                          // P5 set Low

// PIN 0
  P5SEL0 &= ~IOT_RESET;                     // IOT_RESET GPIO operation
  P5SEL1 &= ~IOT_RESET;                     // IOT_RESET GPIO operation
  P5DIR |= IOT_RESET;                       // Direction = output
  P5OUT &= ~IOT_RESET;                      // Set IOT_RESET Off [Low]

// PIN 1
  P5SEL0 &= ~V_BAT;                         // V_BAT GPIO operation
  P5SEL1 &= ~V_BAT;                         // V_BAT GPIO operation
  P5DIR &= ~V_BAT;                          // Direction = input
  // P5OUT &= ~V_BAT;                          // Set V_BAT Off [Low]

// PIN 2
  P5SEL0 &= ~IOT_PROG_SEL;                  // IOT_PROG_SEL GPIO operation
  P5SEL1 &= ~IOT_PROG_SEL;                  // IOT_PROG_SEL GPIO operation
  P5DIR &= ~IOT_PROG_SEL;                   // Direction = input
  // P5OUT &= ~IOT_PROG_SEL;                   // Set IOT_PROG_SEL Off [Low]

// PIN 3
  P5SEL0 &= ~V_3_3;                         // V_3_3 GPIO operation
  P5SEL1 &= ~V_3_3;                         // V_3_3 GPIO operation
  P5DIR &= ~V_3_3;                          // Direction = input
  // P5OUT &= ~V_3_3;                          // Set V_3_3 Off [Low]

// PIN 4
  P5SEL0 &= ~IOT_PROG_MODE;                 // IOT_PROG_MODE GPIO operation
  P5SEL1 &= ~IOT_PROG_MODE;                 // IOT_PROG_MODE GPIO operation
  P5DIR &= ~IOT_PROG_MODE;                  // Direction = input
  // P5OUT &= ~IOT_PROG_MODE;                  // Set IOT_PROG_MODE Off [Low]
//------------------------------------------------------------------------------
}

void Init_Port6(void){
//------------------------------------------------------------------------------
// Configure PORT 6
  P6SEL0 = SET_GPIO;                    // Set P6 operation to GPIO
  P6SEL1 = SET_GPIO;                    // Set P6 operation to GPIO
  P6DIR = SET_OUTPUTS;                      // Set P6 direction to output
  P6OUT = SET_LOW;                          // P6 set Low

// PIN 0
  P6SEL0 |= R_FORWARD;                      // R_FORWARD PWM operation
  P6SEL1 &= ~R_FORWARD;                     // R_FORWARD PWM operation
  P6DIR |= R_FORWARD;                       // Set R_FORWARD direction to output

// PIN 1
  P6SEL0 |= L_FORWARD;                      // L_FORWARD PWM operation
  P6SEL1 &= ~L_FORWARD;                     // L_FORWARD PWM operation
  P6DIR |= L_FORWARD;                       // Set L_FORWARD direction to output
```

```
// PIN 2
  P6SEL0 |= R_REVERSE;                   // R_REVERSE PWM operation
  P6SEL1 &= ~R_REVERSE;                  // R_REVERSE PWM operation
  P6DIR |= R_REVERSE;                    // Set R_REVERSE direction to output

// PIN 3
  P6SEL0 |= L_REVERSE;                   // L_REVERSE PWM operation
  P6SEL1 &= ~L_REVERSE;                  // L_REVERSE PWM operation
  P6DIR |= L_REVERSE;                    // Set L_REVERSE direction to output

// PIN 4
  P6SEL0 &= ~LCD_BACKLITE;               // LCD_BACKLITE GPIO operation
  P6SEL1 &= ~LCD_BACKLITE;               // LCD_BACKLITE GPIO operation
  P6DIR |= LCD_BACKLITE;                 // Set LCD_BACKLITE direction to output
  P6OUT |= LCD_BACKLITE;                 // Set LCD_BACKLITE On [High]

// PIN 5
  P6SEL0 &= ~P6_5;                       // P6_5 GPIO operation
  P6SEL1 &= ~P6_5;                       // P6_5 GPIO operation
  P6DIR &= ~P6_5;                        // Direction = input

// PIN 6
  P6SEL0 &= ~GRN_LED;                    // GRN_LED GPIO operation
  P6SEL1 &= ~GRN_LED;                    // GRN_LED GPIO operation
  P6DIR |= GRN_LED;                      // Set GRN_LED direction to output
  P6OUT &= ~GRN_LED;                     // Set GRN_LED Off [Low]
//------------------------------------------------------------------------------
}
```

9.3. timers.c

```
//------------------------------------------------------------------------------
//
//  Description: This file contains timer configurations
//
//
//  Feb 2020
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

void Init_Timers(void) {
  Init_Timer_B0();
  Init_Timer_B1();
  Init_Timer_B3();
}

// Timer B0 initialization sets up both B0_0, B0_1, B0_2, and overflow
void Init_Timer_B0(void) {
  TB0CTL = TBSSEL__SMCLK;                 // SMCLK source
  TB0CTL |= TBCLR ;                     // Resets TB0R , clock divider, count direction
  TB0CTL |= MC__CONTINOUS;                // Continuous up
  TB0CTL |= ID__2;                        // Divide clock by 2

  TB0EX0 = TBIDEX__8;                     // Divide clock by an additional 8

  TB0CCR0 = TB0CCR0_INTERVAL;             // CCR0
  TB0CCTL0 |= CCIE;                       // CCR0 enable interrupt

  TB0CCR1 = TB0CCR1_INTERVAL;             // CCR1
  TB0CCTL1 &= ~CCIE;                      // CCR1 disable interrupt

  TB0CCR2 = TB0CCR2_INTERVAL;             // CCR2
  TB0CCTL2 &= ~CCIE;                      // CCR2 disable interrupt

  TB0CTL &= ~TBIE ;                       // Disable Overflow Interrupt
  TB0CTL &= ~TBIFG ;                      // Clear Overflow Interrupt flag
}
//------------------------------------------------------------------------------
```

9.4. init_ADC.c

```
//------------------------------------------------------------------------------
//
//  Description: This file initializes the ADC
//
//
//  Feb 2020
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

void Init_ADC(void){
//------------------------------------------------------------------------------
// V_DETECT_L (0x04) // Pin 2 A2
// V_DETECT_R (0x08) // Pin 3 A3
// V_THUMB (0x20) // Pin 5 A5
//------------------------------------------------------------------------------  //
ADCCTL0 Register
  ADCCTL0 = RESET_STATE;        // Reset
  ADCCTL0 |= ADCSHT_2;          // 16 ADC clocks
  ADCCTL0 |= ADCMSC;            // MSC
  ADCCTL0 |= ADCON;             // ADC ON

// ADCCTL1 Register
  ADCCTL2 = RESET_STATE;        // Reset
  ADCCTL1 |= ADCSHS_0;          // 00b = ADCSC bit
  ADCCTL1 |= ADCSHP;             // ADC sample-and-hold SAMPCON signal from sampling
timer.
  ADCCTL1 &= ~ADCISSH;          // ADC invert signal sample-and-hold.
  ADCCTL1 |= ADCDIV_0;          // ADC clock divider - 000b = Divide by 1
  ADCCTL1 |= ADCSSEL_0;         // ADC clock MODCLK
  ADCCTL1 |= ADCCONSEQ_0;          // ADC conversion sequence 00b = Single-channel
single-conversion
// ADCCTL1 & ADCBUSY identifies a conversion is in process

// ADCCTL2 Register
  ADCCTL2 = RESET_STATE;        // Reset
  ADCCTL2 |= ADCPDIV0;          // ADC pre-divider 00b = Pre-divide by 1
  ADCCTL2 |= ADCRES_2;             // ADC resolution 10b = 12 bit (14 clock cycle
conversion time)
  ADCCTL2 &= ~ADCDF;            // ADC data read-back format 0b = Binary unsigned.
  ADCCTL2 &= ~ADCSR;             // ADC sampling rate 0b = ADC buffer supports up to
200 ksps

// ADCMCTL0 Register
  ADCMCTL0 |= ADCSREF_0;        // VREF - 000b = {VR+ = AVCC and VR- = AVSS }
  ADCMCTL0 |= ADCINCH_5;        // V_THUMB (0x20) Pin 5 A5
  ADCIE |= ADCIE0;              // Enable ADC conv complete interrupt
  ADCCTL0 |= ADCENC;            // ADC enable conversion.
  ADCCTL0 |= ADCSC;             // ADC start conversion.
}
```

9.5. init_serial.c

```
//------------------------------------------------------------------------------
//
//  Description: This file initializes the Serial Communications
//
//
//  Feb 2020
//  Built with IAR Embedded Workbench Version: V4.10A/W32 (7.12.4)
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

#include  "functions.h"
#include  "msp430.h"
#include  "macros.h"
#include <string.h>

extern volatile unsigned int usb_rx_ring_wr_UCA0;
extern volatile unsigned int usb_rx_ring_rd_UCA0;
extern volatile char USB_Char_Rx_UCA0[SMALL_RING_SIZE];

extern volatile unsigned int usb_rx_ring_wr_UCA1;
extern volatile unsigned int usb_rx_ring_rd_UCA1;
extern volatile char USB_Char_Rx_UCA1[SMALL_RING_SIZE];

volatile extern char test_command_UCA0[SMALL_RING_SIZE];
volatile extern char test_command_UCA1[SMALL_RING_SIZE];

volatile unsigned int UCA0_index;
volatile unsigned int UCA1_index;

unsigned int baudrate = BAUD_115200;

void Init_Serial_UCA0(unsigned int baud){
//------------------------------------------------------------------------------
// Initializes Serial Port UCA0
//------------------------------------------------------------------------------
  int i;
  for(i = RESET_STATE; i < SMALL_RING_SIZE; i++) {
    USB_Char_Rx_UCA0[i] = NULL_HEX;              // USB Rx Buffer
  }
  usb_rx_ring_wr_UCA0 = BEGINNING;
  usb_rx_ring_rd_UCA0 = BEGINNING;

  // Configure UART 0
  UCA0CTLW0 = RESET_STATE;              // Use word register
  UCA0CTLW0 |= UCSWRST;                 // Set Software reset enable
  UCA0CTLW0 |= UCSSEL__SMCLK;           // Set SMCLK as fBRCLK

  switch(baud) {
    case(BAUD_115200):
      UCA0BRW = BRW_115200;                          // 115,200 Baud
      UCA0MCTLW = MCTLW_115200;
      UCA0CTLW0 &= ~UCSWRST;              // Set Software reset enable
      UCA0IE |= UCRXIE;                   // Enable RX interrupt
      break;
    case(BAUD_460800):
      UCA0BRW = BRW_460800;                        // 460,800 Baud
```

```
      UCA0MCTLW = MCTLW_460800;
      UCA0CTLW0 &= ~UCSWRST;                // Set Software reset enable
      UCA0IE |= UCRXIE;                     // Enable RX interrupt
      break;
    default:
      break;
  }
  UCA0IFG &= ~UCTXIFG;
  UCA0_index = RESET_STATE;
  UCA0IE |= UCRXIE;
}

void Init_Serial_UCA1(unsigned int baud){
//------------------------------------------------------------------------------
// Initializes Serial Port UCA1
//------------------------------------------------------------------------------
  int i;
  for(i = RESET_STATE; i < SMALL_RING_SIZE; i++) {
    USB_Char_Rx_UCA1[i] = NULL_HEX;              // USB Rx Buffer
  }
  usb_rx_ring_wr_UCA1 = BEGINNING;
  usb_rx_ring_rd_UCA1 = BEGINNING;

  // Configure UART 1
  UCA1CTLW0 = RESET_STATE;                         // Use word register
  UCA1CTLW0 |= UCSWRST;                  // Set Software reset enable
  UCA1CTLW0 |= UCSSEL__SMCLK;            // Set SMCLK as fBRCLK

  switch(baud) {
    case(BAUD_115200):
      UCA1BRW = BRW_115200;                         // 115,200 Baud
      UCA1MCTLW = MCTLW_115200;
      UCA1CTLW0 &= ~UCSWRST;                // Set Software reset enable
      UCA1IE |= UCRXIE;                     // Enable RX interrupt
      break;
    case(BAUD_460800):
      UCA1BRW = BRW_460800;                         // 460,800 Baud
      UCA1MCTLW = MCTLW_460800;
      UCA1CTLW0 &= ~UCSWRST;                // Set Software reset enable
      UCA1IE |= UCRXIE;                     // Enable RX interrupt
      break;
    default:
      break;
  }
  UCA1IFG &= ~UCTXIFG;
  UCA1_index = RESET_STATE;
  UCA1IE |= UCRXIE;
}
```

10. Conclusion

Our whole team worked hard on this project. We went from not knowing how to program a microcontroller to programming our microcontroller and controlling a car to follow a black line. Throughout this project, we learned many things.

We first learned the concept of non-blocking code vs blocking code. Knowing this difference really helped us to make our code more efficient. The second thing we learned was port configuration. We also learned that if the ports are not configured properly then it can lead to hardware failure. One of the most important things we learned was the concept of interrupts. The use of interrupts really helped to make the code even more efficient, by only running when specific criteria are met. Other important things we learned were how Analog to Digital converters worked, and how they could be used to detect a black line. Another important topic we learned about was serial communications. Before this class we had no idea as to what it exactly meant when devices were using serial communications to talk to each other.

The most fascinating part about this project was being able to connect our car to our network using an IOT module and being able to send commands to it. This was our first time programming an IOT module and it was a very novel experience.

Overall, this project was very enlightening and very fun, and really helped us to understand many complex concepts in embedded systems.