CECS 347 Spring 2022 Project # 2

An Autonomous Wall Follower Robot Car

By

Abhishek Jasti, Anand Jasti

March 6, 2022

Design an autonomous wall follower robot car with GPIO, edge-triggered interrupts, hardware PWM, power supply circuit, IR sensors, ADC, DC motors, and motor drivers using TM4C123G Launchpad Microcontroller.

CECS 347 Project 2 Report

Introduction

Building an autonomous robot car than can navigate through any prebuilt tracks with walls on both sides. Using two IR sensor to detect left and right distances to the walls. This detection will allow us to determine the amount of power given to each wheel to keep the robot car on the track. We are using basic hardware components in our embedded systems, while including GPIO, hardware timer, interrupt, analog to digital conversion (ADC), and hardware PWM. Our goal was to be able to use GPIO to interface basic input and output devices, switches, LEDs, motors, sensors, and input/output drivers.

Operation

In this beginning we implemented hardware PWM to drive the two DC motors at different speeds. We used two IR sensors on the front of the car to determine the direction of the robot car. To display direction and action taken we used LEDs:

- YELLOW LED:
  - Should display for 2 seconds before the robot car starts to move.
- PURPLE LED:
  - Should display when the robot car reaches end of the track
- RED LED:
  - Should display when the robot car is too close to a wall
- GREEN LED:
  - Should display when the robot car is closer to the left wall
- BLUE LED:
  - Should display when the robot car is closer to the right wall
- WHITE LED:
  - Should display when the robot car is changing in speed cycles
  -

No LED should display when the car is in the middle of the track. For this project we also used the two onboard switch buttons.

SWITCH 1:

- Toggle car on/off and toggle reverse/forward direction.

- Turn wheels off.
- Show yellow LED for 2 seconds.
- Start moving from one end of the track and navigate by itself toward the other end of the track.
- Stop at the end of the track, and show purple LED.

SWITCH 2:

- Toggle speed setting mode on/off.
- Check potentiometer value to set robot speed.

Link to Demonstration videos:

Forward in the middle:
https://drive.google.com/file/d/1fAIQaFzEt25InrQ1jtdM3Aa2dBDHU0qu/view?usp=sharing

Forward 20cm to the left wall:
https://drive.google.com/file/d/10gVMWSqpvYfpY3bGPPP2jZvi5TY2i3Tj/view?usp=sharing

Forward 20cm to the right wall: https://drive.google.com/file/d/1-pqo6zZy_n0lxN7lO8MPlM4ZZM_nMYVI/view?usp=sharing

Reverse in the middle:
https://drive.google.com/file/d/1NH1eZ6dgRBY0rsilYsrmZOckWh4X2Fyd/view?usp=sharing

Reverse 20cm to the left wall:
https://drive.google.com/file/d/1HjY5L2w5W1tQeFkzIl9BgcMAF3Wiy7SO/view?usp=sharing

Reverse 20cm to the right wall:
https://drive.google.com/file/d/1mbyDITCCr6dYre1cbwVV89UjX4r2NHHF/view?usp=sharing

Live Demo:
https://drive.google.com/file/d/1vTv4ONJz2DTVVRbfhmeUwLhzmh2mgO7M/view?usp=sharing

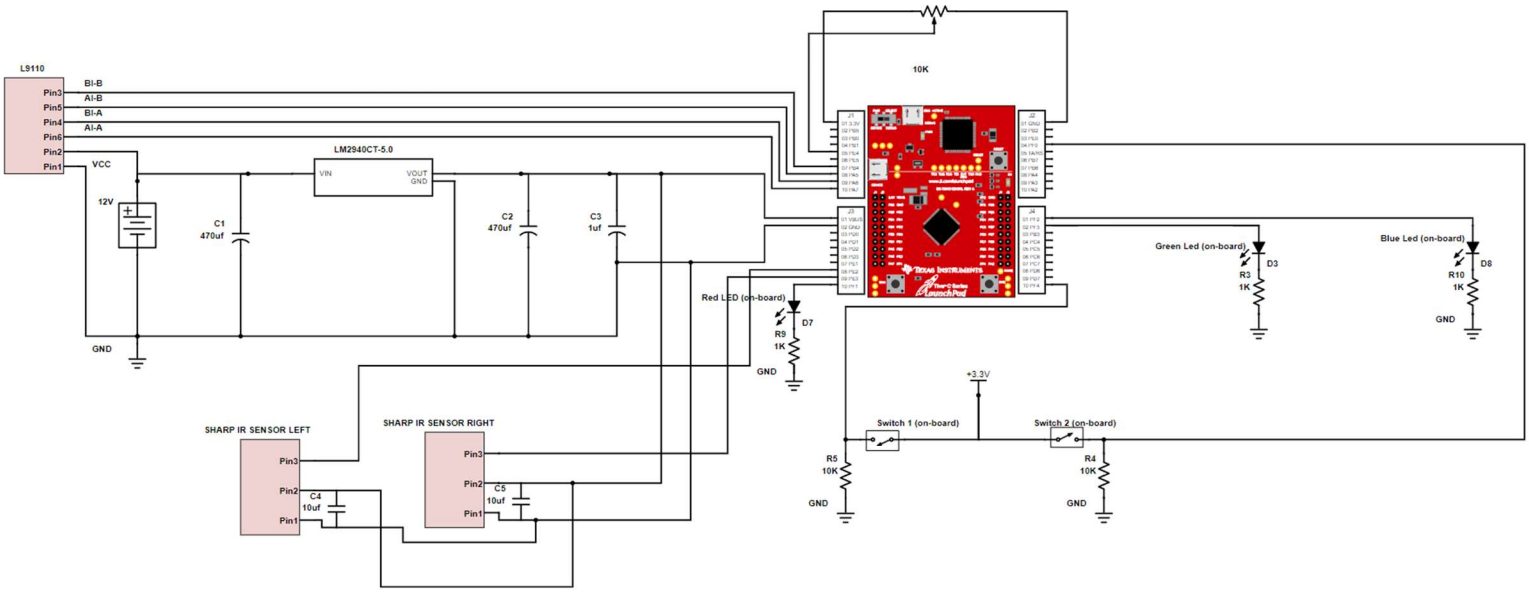
---

Theory

This project uses ARM Cortex TM4C123GH6PM Microcontroller, more specifically we used two of the six General-Purpose I/O ports (PF, PA). In port A, we used two pins for direction (PA4, PA5) and two pins for PWM signal (PA6, PA7). In port F we used the onboard buttons pin 0 and pin 4 for toggling on/off the car and toggling on/off the speed mode (more specifically defined in Operation and Hardware design). In this project we used basic hardware components, GPIO pins, edge-triggered interrupts, hardware PWM, power supply circuits, DC motors, motor drivers, IR sensors, and ADC to implement this autonomous wall follower robot car. We used hardware PWM on pins PA6 and PA7 to generate a 1KHz frequency to control the speed of the robot car. We also used GPIO pins PA4 and PA5 to control the direction (forward and backward) of the robot car. We used the two IR sensors to guide our robot car. We also used the perineometer to set the speed of the car (more specifically defined in Software design).

Hardware design

Schematic:



Outputs:

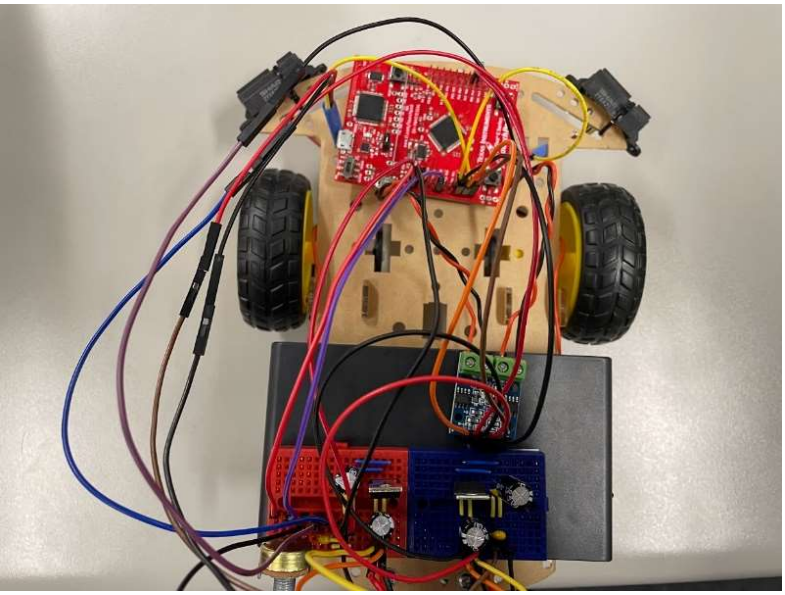
Red LED	PF1
Blue LED	PF2
Green LED	PF3
Direction signal for Left motor (BI-B pin 3 on L9110)	PA4
Direction signal for Right motor (AI-B pin 5 on L9110)	PA5
Speed signal for Left motor (B-IA pin 4 on L9110)	PA6
Speed signal for Right motor (A-IA pin 6 on L9110)	PA7

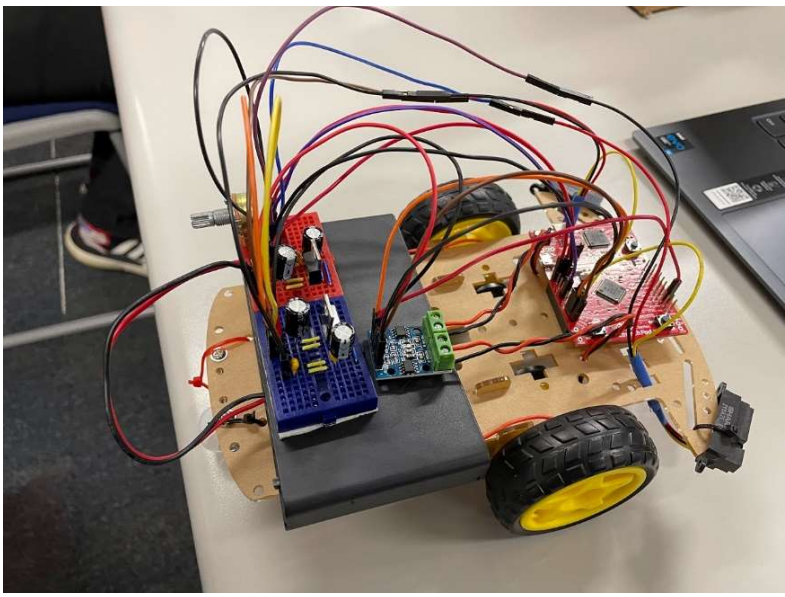
Inputs:

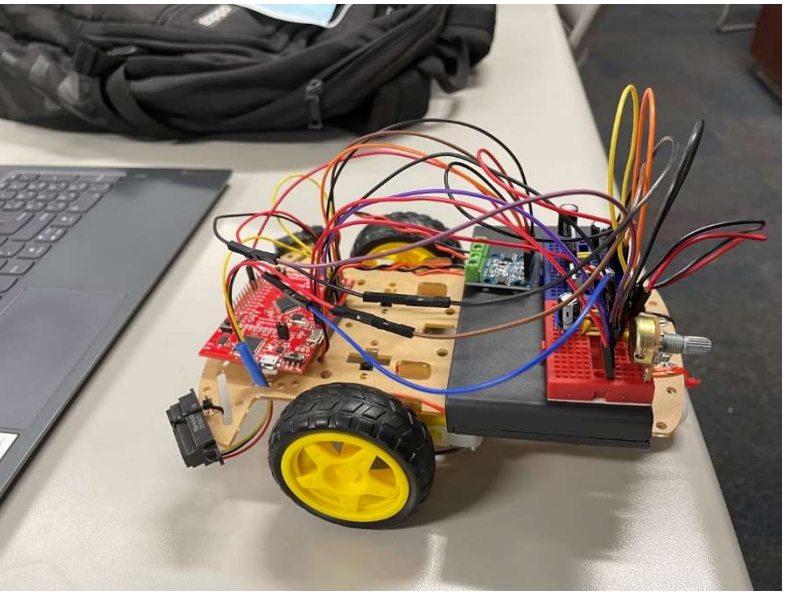
Switch 1	**PF4**
Switch 2	**PF0**
Left Sharp IR Sensor	**PE2**
Right Sharp IR Sensor	**PE3**
Potentiometer for speed control	**PE4**

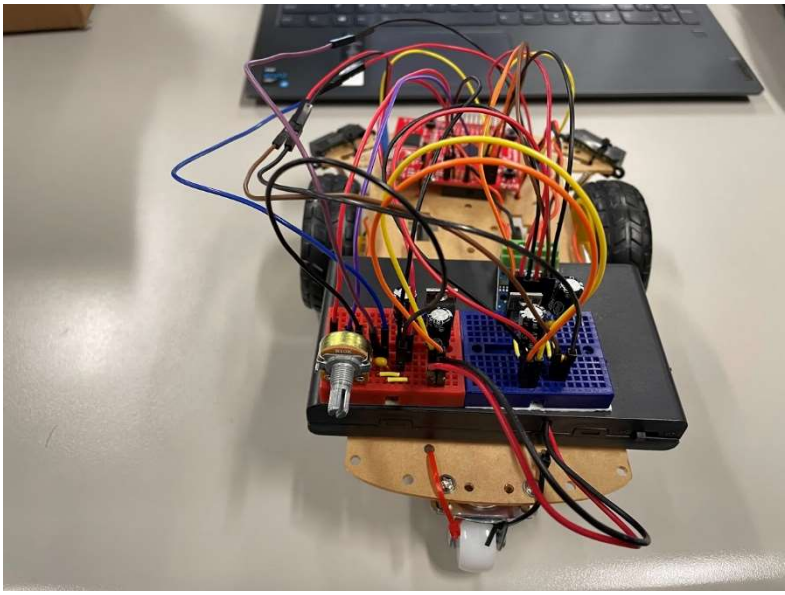
The left and right motors are being driven by L9110. The pins connected to the L9110 from our launchpad are listed above. The Red, Green, and Blue LEDs are on-board LEDs provided by the launchpad.

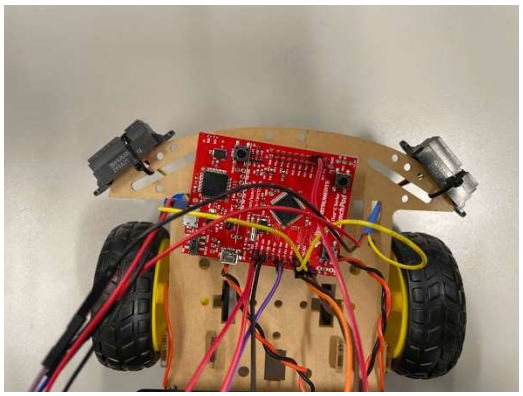
Pictures of Hardware System:











Software design

Software Source Code:

```
// PA6_PA7_PWM.c
// Documentation
// Description: Initialize Port A bit 6 and 7 for Hardware PWM
// Student Name: Abhishek Jasti, Anand Jasti

#include <stdint.h> // C99 data types
#include "tm4c123gh6pm.h"

// period is a 16-bit number of PWM clock cycles in one period
// Output on PA6/M1PWM2, PA7/M1PWM3
void PWM1G1AB_Init(uint16_t period){
  SYSCTL_RCGCPWM_R |= SYSCTL_RCGCPWM_R1;      // activate PWM1
  //if ((SYSCTL_RCGC2_R &= SYSCTL_RCGC2_GPIOA) != SYSCTL_RCGC2_GPIOA){
      //SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA;   // activate port A
      //while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOA)!=SYSCTL_RCGC2_GPIOA){} // wait for the clock to be ready
  //}

  GPIO_PORTA_AFSEL_R |= 0xC0;         // enable alternate function on PA6, PA7
  GPIO_PORTA_PCTL_R &= ~0xFF000000;   // configure PA6, PA7 as PWM1
  GPIO_PORTA_PCTL_R |= 0x55000000;
  GPIO_PORTA_AMSEL_R &= ~0xC0;        // disable analof functionality on PA6, PA7
  GPIO_PORTA_DEN_R |= 0xC0;           // enable digitial I/O on PA6, PA7

  //SYSCTL_RCC_R &= ~0x00100000;        // system clock is the source for PWM clock 16MHz
  PWM1_1_CTL_R = 0;                   // re-loading down-counting mode
  PWM1_1_GENA_R = (PWM_1_GENA_ACTCMPAD_INV|PWM_1_GENA_ACTLOAD_ZERO);  // low on LOAD, high on CMPA down
  PWM1_1_GENB_R = (PWM_1_GENB_ACTCMPBD_INV|PWM_1_GENB_ACTLOAD_ZERO);  // low on LOAD, high on CMPB down
  PWM1_1_LOAD_R = period -1;          // cycles needed to count down to 0
  PWM1_1_CMPA_R = 0;                  // count value when output rises
  PWM1_1_CMPB_R = 0;                  // count value when output rises
  PWM1_1_CTL_R |= 0x00000001;         // Start PWM1
  PWM1_ENABLE_R |=  0x0000000C;
}

// change duty cycle of PA6, PA7
// duty is number of PWM clock cycles output is high
void PWM1AB_Duty(uint16_t duty_L, uint16_t duty_R){
  PWM1_1_CMPA_R = duty_L - 1;           // count value when output rises
  PWM1_1_CMPB_R = duty_R - 1;           // count value when output rises
}

```

In this bit of code above we are initializing Port A bit 6 and 7 for Hardware PWM, and we are also providing a function to change the duty cycle of PA6 and PA7. We are using these two pins to control the speed of the left and right DC motors using the L9110 motor driver.

```c
// PA4_PA5_GPIO.c
// Documentation
// Description: Initialize Port A bit 4 and 5 as output
// Student Name: Abhishek Jasti, Anand Jasti

#include "tm4c123gh6pm.h"

// Initialize Port A bit 4 and 5 as output
void PortA45_Init(void){
  //if ((SYSCTL_RCGC2_R &= SYSCTL_RCGC2_GPIOA) != SYSCTL_RCGC2_GPIOA){
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOA;   // activate port A
  while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOA)!=SYSCTL_RCGC2_GPIOA){} // wait for the clock to be ready
  //}

  GPIO_PORTA_AMSEL_R &= ~0x30;        // disable analog function
  GPIO_PORTA_PCTL_R &= ~0x00FF0000;   // GPIO clear bit PCTL
  GPIO_PORTA_DIR_R |= 0x30;           // make PA4,PA5 output
  GPIO_PORTA_AFSEL_R &= ~0x30;        // disable alternate function on PA4,PA5
  GPIO_PORTA_DEN_R |= 0x30;           // enable digital pins PA6,PA5
}
```

In this code above we are initializing Port A pin 4 and 5 to be outputs. We are using these two pins to control the direction of the left and right DC motors using the L9110 motor driver.

```c
// SysTickInterrupt.c
// Documentation
// Description: Initialize SysTick timer for 1ms delay with
//    interrupt enabled and priority 1 assuming 16MHz clock
// Student Name: Abhishek Jasti, Anand Jasti

#include "tm4c123gh6pm.h"

// Initialize SysTick timer for 0.05s delay with interrupt enabled
void SysTickInterrupt_Init(void){
  NVIC_ST_CTRL_R = 0;                 // disable SysTick during setup
  //NVIC_ST_RELOAD_R = 800000 - 1;    // number of counts to wait 0.05 seconds (assuming 16MHz clock)
  NVIC_ST_RELOAD_R = 16000 - 1;       // number of counts to wait 1ms (assuming 16MHz clock)
  NVIC_ST_CURRENT_R = 0;              // any write to CURRENT clears
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x1FFFFFFF)|0x20000000;  // priority 1
  NVIC_ST_CTRL_R = 0x07;              // enable SysTick with core and interrupts
}
```

In this bit of code we are initializing SysTick timer for 1ms delay with interrupt enabled and priority 1. We are using the SysTick timer to sample ADC values. The three ADC values we are sampling are left sensor, right sensor, and the potentiometer.

```c
// PortF.c
// Documentation
// Description: Initialize Port F LEDs
//    Initialize edge trigger interrupt for
//    PF0 (SW2) and PF4 (SW1) falling edge
// Student Name: Abhishek Jasti, Anand Jasti

#include <stdint.h> // C99 data types
#include "tm4c123gh6pm.h"

// Initialize Port F LEDs
void PortF_LEDInit(void){
  if ((SYSCTL_RCGC2_R &= SYSCTL_RCGC2_GPIOF) != SYSCTL_RCGC2_GPIOF){
      SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF;   // activate port F
      while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOF)!=SYSCTL_RCGC2_GPIOF){} // wait for the clock to be ready
  }

  GPIO_PORTF_AMSEL_R &= ~0x0E;         // disable analog function
  GPIO_PORTF_PCTL_R &= ~0x0000FFF0;    // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R |= 0x0E;            // make PF1,PF2,PF3 output (built-in LED)
  GPIO_PORTF_AFSEL_R &= ~0x0E;         // disable alternate function on PF1,PF2,PF3
  GPIO_PORTF_DEN_R |= 0x0E;            // enable digital pins PF1,PF2,PF4
}
```

In this code we are initializing Port F pins 1,2, and 3. We are using these three pins as outputs (which are built-in LEDs). In this project we are using the built-in LEDs to indicate car status. Here is the color information: yellow light: two seconds yellow light before robot start to move; purple light: when reaching end of the track, turn on purple light; red light: the car is too close to a wall and stopped before restart; green light: the car is closer to the left wall; blue light: the car is closer to the right wall; white light: the car is in speed setting mode. No light indicates the car is moving in the middle of the track.

```c
// Initialize edge trigger interrupt for PF0 (SW2) and PF4 (SW1) falling edge
void PortF_EdgeTriggerInit(void){
  if ((SYSCTL_RCGC2_R &= SYSCTL_RCGC2_GPIOF) != SYSCTL_RCGC2_GPIOF){
      SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF;   // activate port F
      while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOF)!=SYSCTL_RCGC2_GPIOF){} // wait for the clock to be ready
  }

  GPIO_PORTF_LOCK_R = 0x4C4F434B;     // unlock PortF PF0
  GPIO_PORTF_CR_R |= 0x01;            // allow changes to PF0
  GPIO_PORTF_AMSEL_R &= ~0x11;        // disable analog function
  GPIO_PORTF_PCTL_R &= ~0x000F000F;   // GPIO clear bit PCTL
  GPIO_PORTF_DIR_R &= ~0x11;          // make PF0, PF4 input (built-in buttons)
  GPIO_PORTF_AFSEL_R &= ~0x11;        // disable alternate function on PF0, PF4
  GPIO_PORTF_PUR_R |= 0x11;           // enable pullup resistors on PF0, PF4
  GPIO_PORTF_DEN_R |= 0x11;           // enable digital pins PF0, PF4
  GPIO_PORTF_IS_R &= ~0x11;           // PF0, PF4 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x11;          // PF0, PF4 is not both edges
  GPIO_PORTF_IEV_R &= ~0x11;          // PF0, PF4 Falling edge event
  GPIO_PORTF_ICR_R = 0x11;            // clear flag0, flag4
  GPIO_PORTF_IM_R |= 0x11;            // arm interrupt on PF0, PF4
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00400000;  // priority 2
  NVIC_EN0_R |= 0x40000000;           // enable interrupt 30 in NVIC
}
```

In this code above we are initializing Port F pins 0 and 4 which are built-in push buttons to be inputs, using falling edge trigger interrupt. The push buttons are used as follows: sw1 will toggle on/off the car, sw2 will toggle speed setting mode on/off.

```
// Subroutine to wait 0.5 sec assuming 16MHz clock
// Inputs: None
// Outputs: None
// Notes: ...
void Delay_50ms(uint8_t n){
  unsigned long volatile time;
  time = 727240*10*n/91;  // 0.05 sec assuming 16MHz clock
  while(time){
    time--;
  }
  for (time=0;time<1000;time=time+10) {
  }
}
```

In this code we are providing a delay function to be used in Port F Handler for debouncing.

```
// ADCSWTrigger_PE2_PE3_PE4.c
// Documentation
// Description: Initialize ADC0 SS2 on Ain1(PE2) Ain0(PE3) Ain9(PE4) to be
//                          triggered by software and trigger a conversion,
//                          wait for it to finish, and return the result.
// Student Name: Abhishek Jasti, Anand Jasti

#include "tm4c123gh6pm.h"

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain1 (PE2)
// SS3 interrupts: flag set on completion but no interrupt requested
void ADC0_InitSWTriggerSeq2_Ch1_Ch0_Ch9(void){
  //volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x00000010;   // 1) activate clock for Port E
  while((SYSCTL_RCGC2_R&0x00000010) != 0x00000010){} // wait for clock to be ready
  //delay = SYSCTL_RCGC2_R;         //    allow time for clock to stabilize
  GPIO_PORTE_DIR_R &= ~0x1C;      // 2) make PE2, PE3, PE4 input
  GPIO_PORTE_AFSEL_R |= 0x1C;     // 3) enable alternate function on PE2, PE3, PE4
  GPIO_PORTE_DEN_R &= ~0x1C;      // 4) disable digital I/O on PE2, PE3, PE4
  GPIO_PORTE_AMSEL_R |= 0x1C;     // 5) enable analog function on PE2, PE3, PE4
  SYSCTL_RCGC0_R |= 0x00010000;   // 6) activate ADC0
  while((SYSCTL_RCGC0_R&0x00010000) != 0x00010000){} // wait for clock to be ready
  //delay = SYSCTL_RCGC2_R;
  SYSCTL_RCGC0_R &= ~0x00000300;  // 7) configure for 125K
  ADC0_SSPRI_R = 0x0132;          // 8) Sequencer 2 is highest priority
  ADC0_ACTSS_R &= ~0x0004;        // 9) disable sample sequencer 2
  ADC0_EMUX_R &= ~0x0F00;         // 10) seq2 is software trigger
  ADC0_SSMUX2_R = (ADC0_SSMUX2_R&0xFFFFF000)+1+(9<<8); // 11) channel Ain1(PE2), Ain0(PE3), Ain9(PE4)
  ADC0_SSCTL2_R = 0x0600;         // 12) no D0 END0 IE0 TS0, D1 END1 IE1 TS1, D2 TS2, yes IE2 END2
  ADC0_IM_R &= ~0x0004;           // disable SS2 interrupts
  ADC0_ACTSS_R |= 0x0004;         // 13) enable sample sequencer 2
}
```

```
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
void ADC0_InSeq2(unsigned long temp[]){
  unsigned long result1, result2, result3;
  ADC0_PSSI_R = 0x0004;            // 1) initiate SS2
  while((ADC0_RIS_R&0x04)==0){};   // 2) wait for conversion done
  result1 = ADC0_SSFIFO2_R&0xFFF;  // 3) read result
  result2 = ADC0_SSFIFO2_R&0xFFF;  // 3) read result
  result3 = ADC0_SSFIFO2_R&0xFFF;  // 3) read result

  ADC0_ISC_R = 0x0004;             // 4) acknowledge completion
  temp[0] = result1;
  temp[1] = result2;
  temp[2] = result3;
}
```

In the bit of code we are initializing ADC0 sequencer 2 Ain1(PE2), Ain0(PE3), Ain9(PE4) to be software trigger. We are using these three pins to sample ADC values from the left sensor, right sensor, and the potentiometer. We also provider a function to read all three ADC values and put it into an array to be used in the main function.

Conclusion

Implementing this project with the robot car was not as hard as we thought it would be. Building the robot car took the longest time in this project. Creating a power source using a small breadboard was a little tedious trying to fit everything on the breadboard but it was worth the amount of space we saved. We had the most trouble with soldering the wires to the motor, the soldering machine was not the best quality item we bought but we made it work. Fitting everything on the car was not the smartest idea, especially with a huge battery pack that takes eight AA batteries. We decided to use double sided tape to stick everything on the car instead of using zip ties or any other attaching item. Coding of this project for the robot car was not very simple. With the help of our previous labs using PWM and IR sensors helped us tremendously with coding this project for our robot car. Overall, this project took a long time. This project was very helpful to understand and to review the topics of GPIO, edge-triggered interrupts, hardware PWM, power supply circuits, DC Motors, IR sensors, ADC and motor drivers.