eYSIP2016

Around the World of Embedded Systems (TIVA)

Shruti Kothadia
Earnest Vekariya
Parin Chheda
Vishwanathan Iyer
Duration of Internship: 21/05/2016 – 10/07/2016

2016, e-Yantra Publication

Around the World of Embedded Systems (TIVA)

Abstract

TIVA LaunchPad is an inexpensive evaluation platform for ARM CORTEX M4 microcontrollers. Understanding how to use and program the different modules of TIVA Launchpad using Code Composer Studio software. Based on all the modules studied two small projects - A white line following robot and programs for interfacing of switches, LEDs, Joystick, GLCD on the Development Board are completed.

Completion status

All the given tasks have been completed except USB-HID experiment.

1.1 Hardware parts

- TIVA C Series TM4C123G LaunchPad Evaluation Kit
  User's Guide
  Datasheet
  Peripheral Driver Library
- Servo Motor
  SG90 Datasheet
- Joystick
- 16X2 LCD
  JHD162A Datasheet
- 128X64 GLCD
  JHD12864A Datasheet

1.2 Software used

- Code Composer Studio
  Version 6.1.2
  CCS Download Link
  CCS Installation steps
- TivaWare_C_Series-2.1.2.111
  Download Link

Initialization and GPIO

2.1 Lab Objective

1. Understand IO operation in TMS4C123GXL
2. Get acquainted with using on-board RGB LED and User Switches
3. Generating delay using SysCtlDelay()

2.2 Pre-requisites

Creating a new project in CCS and making the required configurations, technique to load and run user written program on the board. Demo code of ledblink is understood and run on the board.

2.3 Problem Statement

In this lab use switch SW1, SW2 and RGB LED present on Tiva C series board.Create a new project and use lab-1.c file.

1. Part 1
   Use switch SW1 to Turn on Red LED on first switch press, Green LED on second switch press and Blue LED on third switch press. Repeat the same cycle next switch press onwards. Note that LED should remain on for the duration switch is kept pressed i.e. LED should turn off when switch is released. Show the result to TA.
2. Part 2
   Use switch SW2 and sw2Status (a variable). The program should increment sw2Status by one, every time switch is pressed. Note how the value of sw2Status changes on each switch press. Use debugger and add sw2Status to Watch Expression window. Does the value of sw2Status

increment by one always? Show the result to TA.
Note: Define sw2Status as a global variable and in debug perspective use continuous refresh option (The Continuous Refresh button is on top of the Expression Window).Step debugging or breakpoints can be used to check the variable value.

Hint:*To add variable to Expression Window, select and right click the variable name and select Add Watch Expression. To view Expression Window, click on View button from CCS menu bar and select Expressions.*

3. Part 3
   Configure SW1 and SW2 such that:
   Every time SW1 is pressed toggle delay of LED should cycle through approximately 0.5s, 1s, 2s (Of any one color).
   Every time SW2 is pressed color of LED should cycle through Red, Green and Blue.

2.4 Relevant Theory

1. Refer *TM4C123G_LaunchPad_Workshop_Workbook* available on course web page.Texas Instrument TM4C123G LaunchPad Workshop - Student Guide and Lab Manual.Refer Chapter-3 Introduction to TivaWare, Initialization and GPIO of the Manual.

2. Use TivaWare Peripheral Driver Library, an API written by Texas Instrument to access different peripherals and functionality of ARM Cortex-M based micro controller.

2.5 Assembly of Hardware

In this Lab external hardware connections are not required.
The PORT pins which will be required are

1. PF0 - SW1

2. PF1 - R of RGB

3. PF2 - B of RGB

4. PF3 - G of RGB

5. PF4 - SW2

All these connections are present on the board.

2.6 Procedure

2.6.1 Procedure for Part 1

1. Include all the required header files. Ensure that the following header file are present:

```
include "inc/hw_types.h"
include "inc/hw_memmap.h"
include "driverlib/sysctl.h"
include "driverlib/gpio.h"
include "inc/hw_ints.h"
include <time.h>
```

2. Declare a variable to monitor the status of switch SW1.
3. Enable all the peripherals which will be used. (PORT F,System Clock)
4. Configure PF4 as input and PF1,PF2 and PF3 as output.
5. When the switch SW1 is pressed the value at PF4 becomes 0. Read the value of PF4.
6. When this value is equal to 0 turn ON R,G or B LED by writing 1 to its corresponding pin

2.6.2 Procedure for Part 2

1. Include all the required header files
2. Enable PORTF and System Clock.
3. Configure PF0as input pin. PF0 is locked by default. Use the following code to unlock PF0

```
HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK)= GPIO_LOCK_KEY;
HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK)= 0;
```

4. Use a variable *sw2status* to record the number of times switch SW2 is pressed
5. In debug perspective double click on variable sw2status. Right click on it and select Add Watch Expression. Select continuous refresh option which is at top in Watch Window.

2.6.3 Procedure for Part 3

1. Include all the required header files.
2. Enable System Clock and PORTF.
3. Configure PF0 and PF4 as input and PF1, PF2 and PF3 as output.
4. Unlock pin PF0.
5. Use SysCtlDelay(6700000) to generate delay of around 0.5 seconds.
6. Monitor the status of SW1 and SW2. Whenever SW1 is pressed the delay should change from 0.5 =>1 =>2 seconds. Whenever SW2 is pressed colour of RGB should change from R =>G =>B.

2.7 Code

1. Github link for solution to Part 1.
2. Github link for solution to Part 2.
3. Github link for solution to Part 3.

2.8 Demonstration Video

Demo video link

Interrupts and Timers

3.1 Lab Objectives

Introduction to the use of Timers and Interrupts on the TM4C123GH6PM.

3.2 Pre-requisites

1. Lab 1: Interfacing RGB LED and switches (SW1 and SW2).
2. Please refer chapter 4 of *TM4C123G_LaunchPad_Workshop_Workbook* available on course page

3.3 Problem Statement

1. Part 1
   Use SW1 to change the color of the RGB LED (R⇒ G⇒ B⇒ R.) where switch is pressed just once instead of long press in Lab 1. Use switch debouncing mentioned below in the procedure to differentiate between switch bounce and actual key press.
2. Part 2
   Use SW2 to increment a global variable once for each button press. Check if the variable always increments by one (adjust the time interval of 10 ms if required)

3.4 Relevant Theory

A timer is used to generate interrupts and a timer interrupt service routine (ISR) that implements a state machine to perform **software button debouncing**.

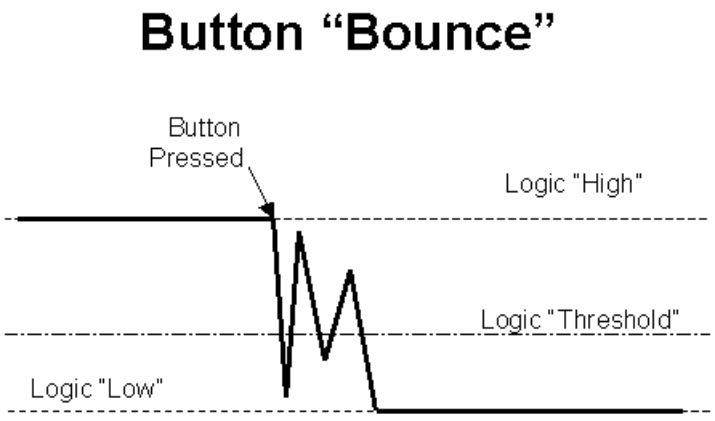


Figure 3.1: Button Bounce Waveform
[1]

Button Bounce: From the above diagram it is clear that when switch is pressed the signal bounces for a finite amount of time before settling down to its final logic state. This is due to the tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open. The bounce period is 10 to 20 ms.

Solution: There are several ways to do button debouncing which includes modifications in both hardware and software. In this lab we will explore software debouncing. In software debouncing too, there are a couple of ways to solve this problem viz.

1. **Polling the switch plus introducing finite delays:** In polling method the controller is busy polling the switch hence it will not be able to perform any other tasks.
2. **Using interrupts and timers:** If we want to use this time to do other tasks, then we can use interrupts and timers.

Switch debouncing using interrupts and timers: Check for the status of the switch at finite intervals (say 10 ms). This time interval will be gener-

ated using timers. When the 10 ms interval is over the timer should generate an interrupt and the code in the interrupt service routine (ISR) will be executed. In the interrupt service routine, a state machine is implemented. Here the idea is that if the switch pressed for two successive intervals (of 10 ms) then we confirm that the switch is pressed and the next key press is detected only if it is confirmed that switch is released. This will ensure that if a switch press is only detected once.

A state machine is described below for implementing software debouncing. In this state machine there are three states viz. Idle, Press and Release for a switch. There are two transition paths in each state. Any state transition condition is checked after a fixed interval of time which is set by a timer (in this case it is 10 ms).
Note: As the bounce period is 10 ms to 20 ms you can change this time interval based on your experimental trials. It is possible that different switches have different bounce periods.

IDLE
If the key is pressed, enter press state
else remain in IDLE state

PRESS
If key is still pressed, then enter release state and make flag 1 indicating that key was pressed.
else return to idle state (de-bouncing period)

RELEASE
If the key is released, then enter an idle state
else remain in release state until key is released.

3.5 Assembly of Hardware

In this Lab external hardware connections are not required.
The PORT pins which will be required are

1. PF0 - SW1
2. PF1 - R of RGB
3. PF2 - B of RGB
4. PF3 - G of RGB

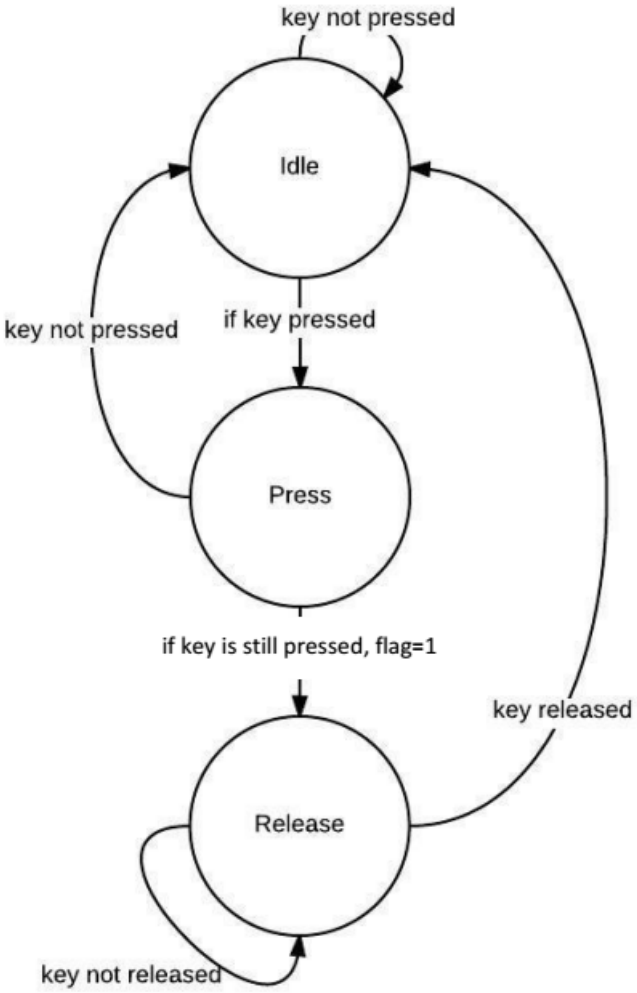


Figure 3.2: Switch debouncing state diagram

5. PF4 - SW2

All these connections are present on the board.

3.6 Procedure

3.6.1 Procedure for Part 1

1. Include all the required header files.
   For using timer and interrupt include :

```
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/pwm.h"
```

2. Enable the peripherals (System Clock, PORTF, Timer).
3. Configure PF4 as input and PF1, PF2 and PF3 as output.
4. Configure the Timer 0 as a 32-bit timer in periodic mode.
5. Calculations :
   To toggle a GPIO at 10Hz and 50% Duty Cycle, interrupt is generated at half value of required period.
   To calculate the number of clock cycles required for 10Hz call *SysCtlClockGet()* and divide it by desired frequency. As interrupt is generated at half value of required frequency, this result is divided by 2.
   period = (SysCtlClockGet()/10)/2.
6. This calculated period is stored in Timer's Interval Load Register.
7. Enable the interrupt in timer module and NVIC.
8. Enable the Timer.
9. After the calculated period interrupt is generated and a ISR is called.
10. ISR :
    In ISR check whether the switch is pressed after an interval of 10ms twice.
    If this condition is satisfied then change the color of RGB.

3.6.2 Procedure for Part 2

1. Include all the required header files.
   For using timer and interrupt include :

   ```
   #include "driverlib/interrupt.h"
   #include "driverlib/timer.h"
   #include "driverlib/pwm.h"
   ```

2. Declare a global variable *sw2status* to monitor the switch press.

3. Enable the peripherals (System Clock, PORTF, Timer) and unlock PF0.

4. Configure PF0 as input.

5. Configure the Timer 0 as a 32-bit timer in periodic mode.

6. Calculations :
   To toggle a GPIO at 10Hz and 50% Duty Cycle, interrupt is generated at half value of required period.
   To calculate the number of clock cycles required for 10Hz call *SysCtlClockGet()* and divide it by desired frequency. As interrupt is generated at half value of required frequency, this result is divided by 2.
   period = (SysCtlClockGet()/10)/2.

7. This calculated period is stored in Timer's Interval Load Register.

8. Enable the interrupt in timer module and NVIC.

9. Enable the Timer.

10. After the calculated period interrupt is generated and a ISR is called.

11. ISR :
    In ISR check whether the switch is pressed after an interval of 10ms twice.
    If this condition is satisfied then increment the value of *sw2status*.

3.7 Code

1. Github link for solution to Part 1.

2. Github link for solution to Part 2.

3.8 Demonstration Video

Demo video link

PWM and interfacing Servo Motor

4.1 Lab Objectives

1. Understanding PWM operation in TMS4C123GXL.
2. Interfacing servo motor with the LaunchPad.

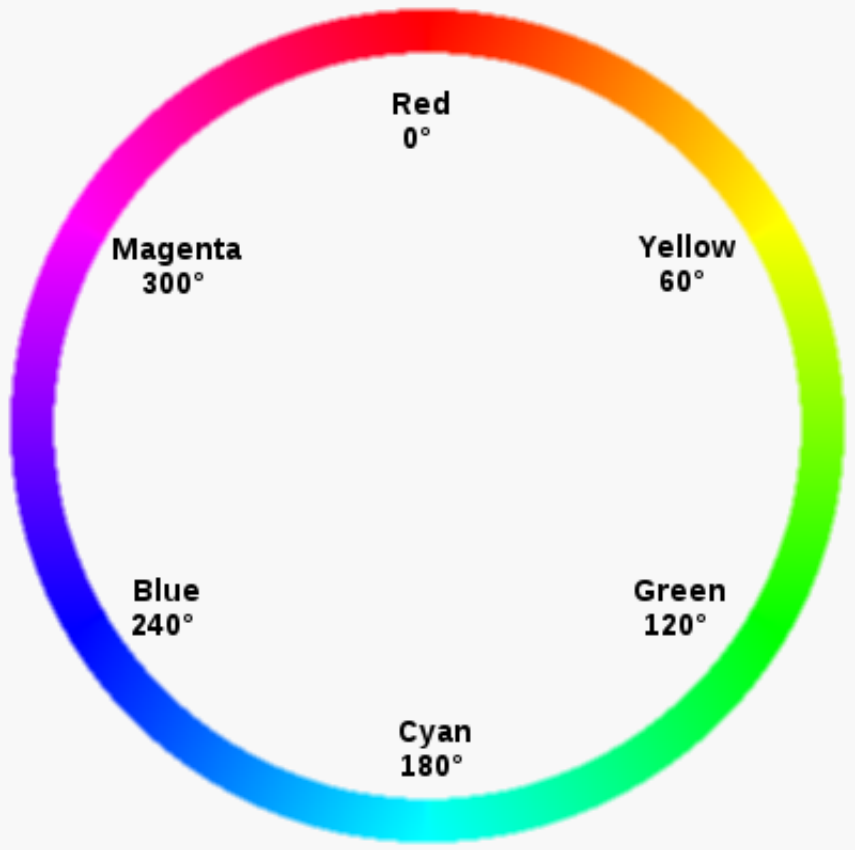
4.2 Pre-requisites

1. Lab 1 and Lab 2: Interfacing RGB LED and both the switches.
2. PWM theory

4.3 Problem Statement

Part 1: Design a RGB LED controller using SW1 and SW2 present in Launchpad board.RGB LED controller has two modes of operation. Auto mode and Manual mode At initial, when program is loaded controller will be in Auto mode. Combination of SW1 an SW2 has to be pressed to go to Manual mode.When Reset button is pressed, controller will go to Auto mode.

1. Auto mode
    - In Auto mode color of the RGB LED follows a pattern in a cycle.
    - The pattern must follow the color circle as shown in Figure 1.
    - In Auto mode SW1 will increase the speed of color transition and SW2 will decrease the speed.



RGB Color Wheel [1]

2. Manual mode

   - In Manual mode, user must be able to select any one of the color from the color circle. For this intensity of any of the 3 LEDs must be controlled independently.
   - Mode 1 (Red LED control) - When SW2 is pressed continuously(long press) and SW1 is pressed once controller goes to Manual Mode 1. In this mode, intensity of Red LED can be controlled using SW1 and SW2.
   - Mode 2 (Blue LED control) - When SW2 is pressed continuously(long press) and SW1 is pressed twice controller goes to Manual Mode 2. In this mode, intensity of Blue LED can be controlled using SW1 and SW2.

[1] Image Courtesy: https://en.wikibooks.org/wiki/File:RGB_color_circle.png accessed on 05/06/2016

- Mode 3 (Green LED control) - When SW1 and SW2 are pressed continuously controller goes to Manual Mode 3. In this mode, intensity of Green LED can be controlled using SW1 and SW2.

Part 2: Interfacing a servo motor and controlling it using a switch.

1. When switch 1 is pressed the motor should rotate by ten degrees clockwise.
2. When switch 2 is pressed the motor should rotate by ten degrees anticlockwise.
3. While doing the above two actions check for limits of the servo motor. It should no move beyond the operating range.
4. Use the switch debouncing method for interfacing the switch.

4.4 Relevant Theory

1. Refer *TM4C123G_LaunchPad_Workshop_Workbook* available on course web page.Texas.Refer Chapter-15.

4.5 Assembly of Hardware

For part 1 external hardware connections are not required.
For part 2 connect the pin PD0 as PWM input to Servo Motor.

1. PF0 - SW1
2. PF1 - R of RGB
3. PF2 - B of RGB
4. PF3 - G of RGB
5. PF4 - SW2
6. PD0 - PWM Output

4.6 Procedure

1. Include all the header files. Ensure that the following header files are present.
   include "driverlib/pwm.h"
   include "driverlib/timer.h"
   include "driverlib/interrupt.h"

2. Configure the PORT Pins as PWM outputs and configure the PWM generator depending on the PWM pin used.

3. Enable the PWM output state.

4. Enable the PWM generator.

5. For Part 1 -
   Depending on the color wheel switch on the respective LEDs. Do not turn off the LED completely by assigning 0 value to it as it causes it to turn ON with a high intensity. Instead assign a minimum non-zero value to it so as to reduce the intensity.

6. For Part 2 -
   Calculate the value of PWM period.
   eg. In this code PWM base frequency is taken as 50Hz. The total period of Servo Motor is 1/50 = 20 ms.
   The oscillator frequency is 40MHz. This value is divided by 64. The answer is further divided by the PWM base frequency(50Hz). Thus Period=12500.

4.7 Code

1. Github link for solution to Part 1.

2. Github link for solution to Part 2.

4.8 Demonstration Video

Demo video link

ADC and Serial Communication

5.1 Lab Objective

Use of the analog to digital conversion (ADC) peripheral and Universal Asynchronous Receiver/Transmitter (UART) on the TM4C123GH6PM.

5.2 Pre-requisite

Go through the documents in relevant theory section.

5.3 Problem Statement

1. Using the inbuilt ADC to interface a joystick with Tiva Board. The analog values read from the joystick have to be converted to digital and displayed on the watch window of the CCS IDE.For information about the joystick pinout refer to the figure below:

2. The inbuilt UART is used to communicate the digital values (i.e. both the X axis and the Y axis) to the computer.A terminal emulation software like Serial Terminal or Real Term can be used to view the data being sent by the Tiva Board to computer.
   The data sent should have the following syntax :
   "X: Digital equivalent of the read value"
   "Y: Digital equivalent of the read value"

3. Once you complete the above mentioned problem statements, you have to depict the values received in a graphical representation. In short, create a GUI which tracks the real time movements of the joystick.
   Refer to the Figure 1, the circle marker should move to the left if the joystick button is tilted towards left and right if tilted towards right. The marker should remain at the centre when joystick is stationary.

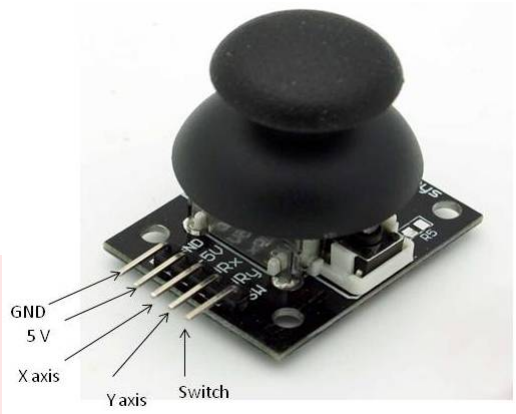


Figure 5.1: Joystick Pinout

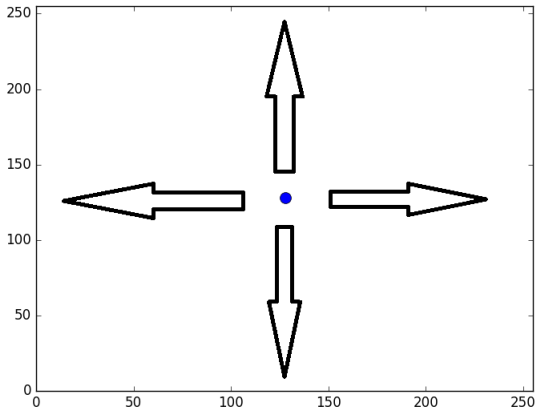


Figure 5.2: Joystick Graphical interface

5.4 Relevant Theory

Some general terms related to ADC. [1]

1. **Sample Sequencer** The sampling control and data capture is handled by the sample sequencers. The sample sequencers can be assumed to be a module with memory, which can sample different analog sources with a single trigger event without processor intervention. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO.

2. **Trigger Source** ADC can be triggered by various sources like processor, external GPIO, PWM, internal comparators, timer etc.

3. **Relative Priority** There are 4 sample sequencers. Hence, we need to specify the relative priority of a sequencer in case we are using multiple sample sequencers.

4. **GPIO Pins** There are 12 analog input channels shared by two Analog-to-Digital Converter modules. Certain GPIO pins can act as input channels for these ADC modules. For information about these pins, refer to section 12, pages 799 - 801 of *Datasheet* . Use GPIOPinTypeADC() function to configure these pins.Refer to page 266 of *Peripheral Driver Library* for detailed description of this function.

Detailed description of theory and code examples are available in the following links:

1. Refer *TM4C123G_LaunchPad_Workshop_Workbook* chapter 5 for ADC12.

2. Refer *TM4C123G_LaunchPad_Workshop_Workbook* chapter 7 for UART.

Joystick Graphical User Interface:
You can use any open source programming language to build your GUI. Below, we have listed down some pointers to assist you, if you choose python

1. The first step is to read serial data coming on your COM port. Use pyserial library to do so.

2. If you notice carefully, GUI only requires five distinct values, you can avoid sending other values in your embedded C code. Also the rate at which Joystick data is sampled can also be reduced.

[1] EE712 Embedded Systems, presentation from WEL Lab

3. Once you start receiving your required data, you can use matplotlib and drawnow library to get a real time Graphical representation. There are several other libraries that can be used to get a real time representation, you are free to use any of them.

5.5 Procedure

5.5.1 For ADC and UART code :

1. Include "adc.h" and "uart.h" header files.
2. Enable ADC0 and ADC1 peripherals. Enable and configure the UART pins. Set the required Baud rate.
3. Wait till ADC conversion is complete. Take average of the data obtained. This is then converted according to requirements.
4. Send the ASCII value of this data to Computer using UART.

5.5.2 To create GUI in Python

1. Import serial library, matplotlib, numpy.
2. Read the serial data.
3. Enable the interactive plot using plt.ion() function.
4. Split the received co-ordinates at comma and convert it into float.
5. Plot these co-ordinates using "drawnow()" function.

5.6 Code

1. Github link to use ADC and UART.
2. Github link for GUI in Python.

5.7 Demonstration Video

Demo video link.

Temperature Sensor(LM35) and LCD

6.1 Lab Objective

Interfacing a 16x2 LCD with the TIVA board and LM35 temperature sensor.

6.2 Pre-requisite

1. Lab 4 : ADC and UART
2. Working of 16x2 LCD and interfacing it in 4 bit mode.

6.3 Problem Statement

1. Interface a 16x2 LCD in 4 bit mode with the board. There is an onboard temperature sensor, display the temperature sensed by this sensor on the LCD; use the internal ADC to convert analog readings to digital ones.
2. Next, interface an external LM35 temperature sensor with the board and display the temperature on the LCD. Use the switch "sw1" to switch between internal and external temperature readings. The display should show external sensor readings by default and switch to internal when"sw1" is pressed once, on the next press the display should return to default (show external temperature readings). Figure 2 below will clear things out.

Follow the display format as shown in the figure 1 below:

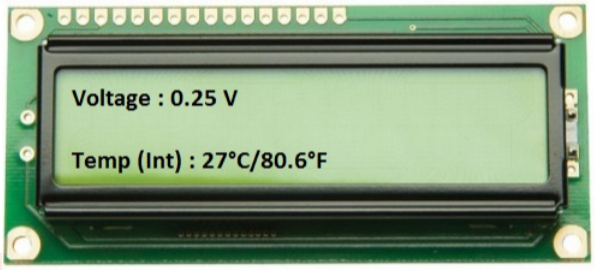


Figure 6.1: LCD Display Format

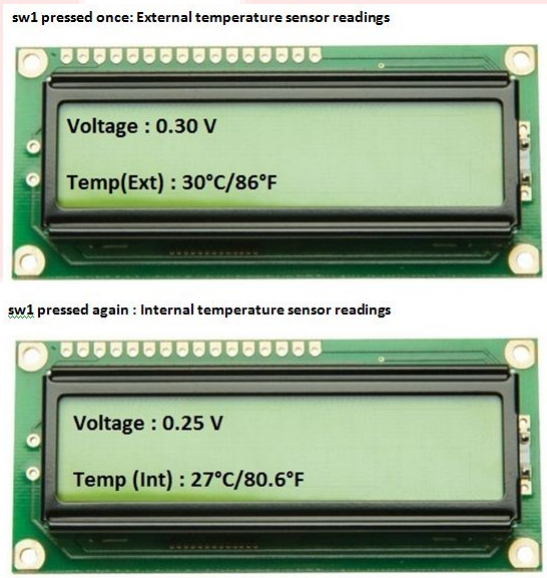


Figure 6.2: Alternating sensor display values

6.4 Relevant Theory

1. This lab will use LM35 temperature sensor, you can refer to its datasheet from the following link:**Datasheet**.

2. LCD has to be used in 4 bit mode. Go through the commands required to interface LCD in 4 bit mode.

3. The problem statement requires you display decimal values upto two places on the LCD. You can shift the values to the left by appropriate places and then use the ascii value of "." to display decimal values.

4. Please follow the timings given in the LCD data sheet during initialization.

6.5 Procedure

6.5.1 For using LCD in 4 bit mode :

1. First initialize LCD in 4 bit mode

2. Give commands for clear screen,display on and cursor location

3. Command mode: In command mode RS=0 and RW=0.
   Send higher 4 bit.
   After sending data to the port;give a enable pulse.
   Send lower 4 bit.
   After sending data to the port;give a enable pulse.
   Give delay of microsecond at end.

4. Write mode: In write mode RS=1 and RW=0
   Send higher 4 bit
   After sending data to the PORT generate high to low enable pulse
   Send lower 4 bit
   After sending data to the PORT generate a high to low enable pulse
   Give delay of microsecond at end

6.5.2 For external temperature conversion:

1. Configure ADC1 chip.

2. Set ch1 to get analog data.
3. Enable ADC1.
4. Wait until interrupt has occurred,
5. Store the data in the variable and convert it into Celsius by dividing it with 12.41.
6. Convert the stored data into character and store it in array.
7. Send the character array to LCD screen.

6.5.3 For internal temperature

1. Configure ADC0 chip.
2. Set the temperature given on chip.
3. Enable ADC0.
4. Wait until interrupt has occurred.
5. Store the data in variable and convert data into Celsius.
6. Convert the stored data into character and store it in array.
7. Send the character array to LCD screen.

6.6 Code

Github link of solution.

6.7 Demonstration

1. LCD Interfacing Tutorial
2. Demo video link

Project 1:Line follower robot

Abstract:

White line following robot using tiva board.It has autocalibration mode which calibrate the sensor and store the value in eeprom.If we again restart the board then it regain the value from eeprom and follow the white line.

Completion Status:

It follows white line and goes to auto mode and stores value in eeprom.

7.1 Software used

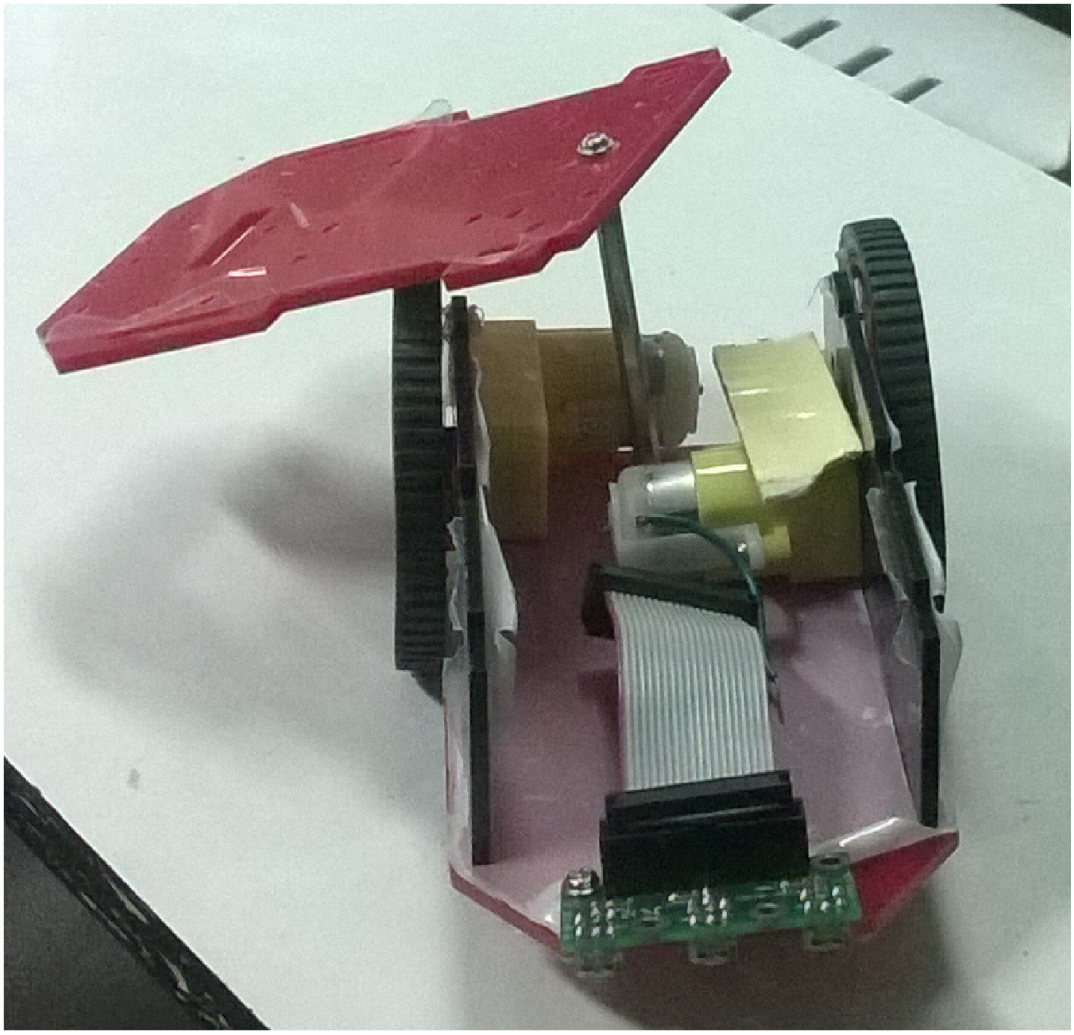
1. Code Composer Studio
   Version 6.1.2
   CCS Download Link
   CCS Installation steps
2. TivaWare_C_Series-2.1.2.111
   Download Link

7.2 Hardware Parts:

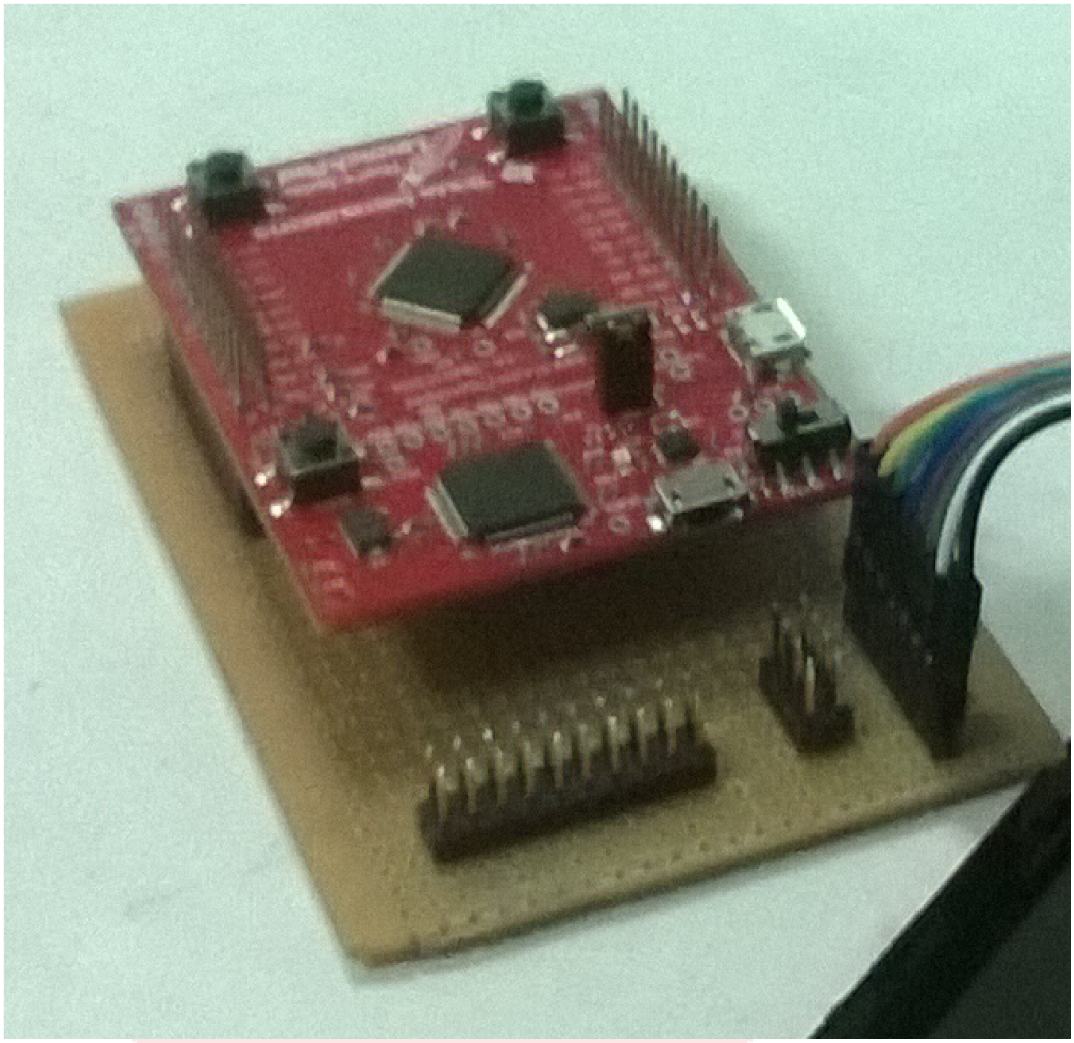
1. Two dc motor
2. White line sensor
3. Motor driver ic
4. Power Bank
5. Two wheel

7.3 Assembly of hardware:

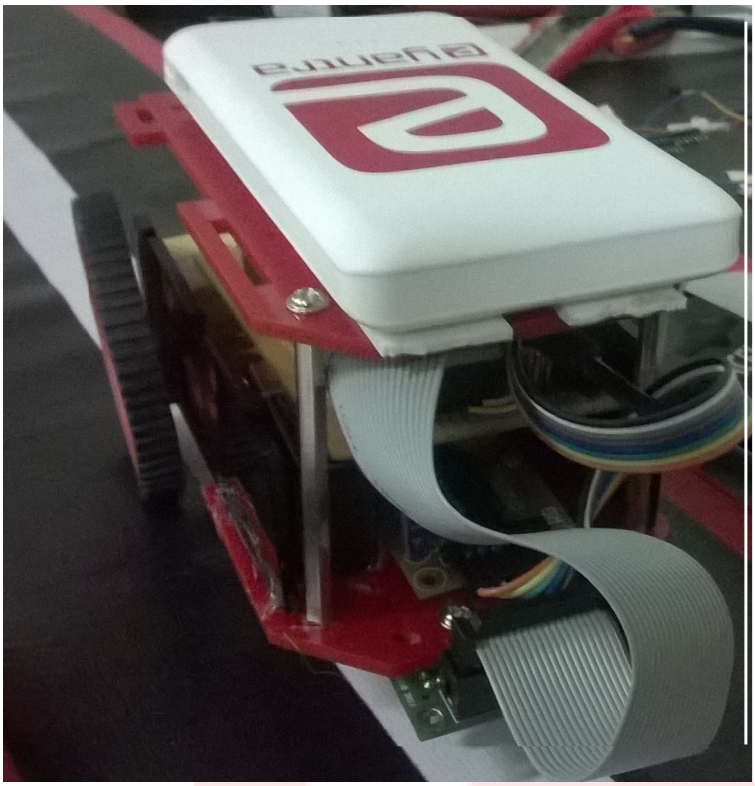
1. First attach the wheel with dc motor through screw.
2. White line sensor to be attached at the bottam of the chassey.
3. Increase the height of top through the support as shown in figure



4. Now on genral pcb make the connection as shown in figure.

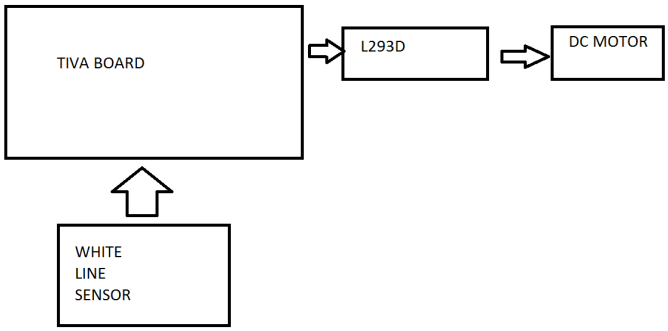


5. Connect the motor driver ic, white line sensor with tiva board



7.4 Block Diagram:

Step 1:TIVA board converts analog value of white line sensor into digital value.



7.5 PROCEDURE

1. First intialize eeprom and read the stored value previously written by the code
2. Set the timer which varify the sw1 status at every 10 millisecond and when the swutch is pressed it goes to auto mode.
3. It do the calibration and stores the value in eeprom.
4. Now after it switch from calibration mode,continous to follow on white line.
5. It takes right turn if both sensor on left side have value greater than their threshold and it takes left turn if both sensor on right side are greater than their threshold
6. If above both condition are not satisfeid than it continous forward journey

7.6 Challanges

1. Autocalibration of white line sensor using eeprom

7.7 Future plan

1. Sensor such as distance sensor,servo motor,lcd can be interfaced.
2. EEPROM concept can be used to solve the problem such as if robot forgets it's path than it can retraced back.

7.8 Code

Github link for all the codes required for this project.

Project 2 : ARM CORTEX M4 based Development Board

Abstract

Development board is a development platform based on ARM CORTEX M4 microcontroller. It uses TIVA LaunchPad TM4C123GXL. Other peripherals like push buttons, LEDs, Analog Thumb Joystick, 16X2 LCD, 128X64 GLCD, Xbee, SPI, MPU and Buzzer are connected on the development board. User can start programming these peripherals directly with minimal or no external hardware connections.

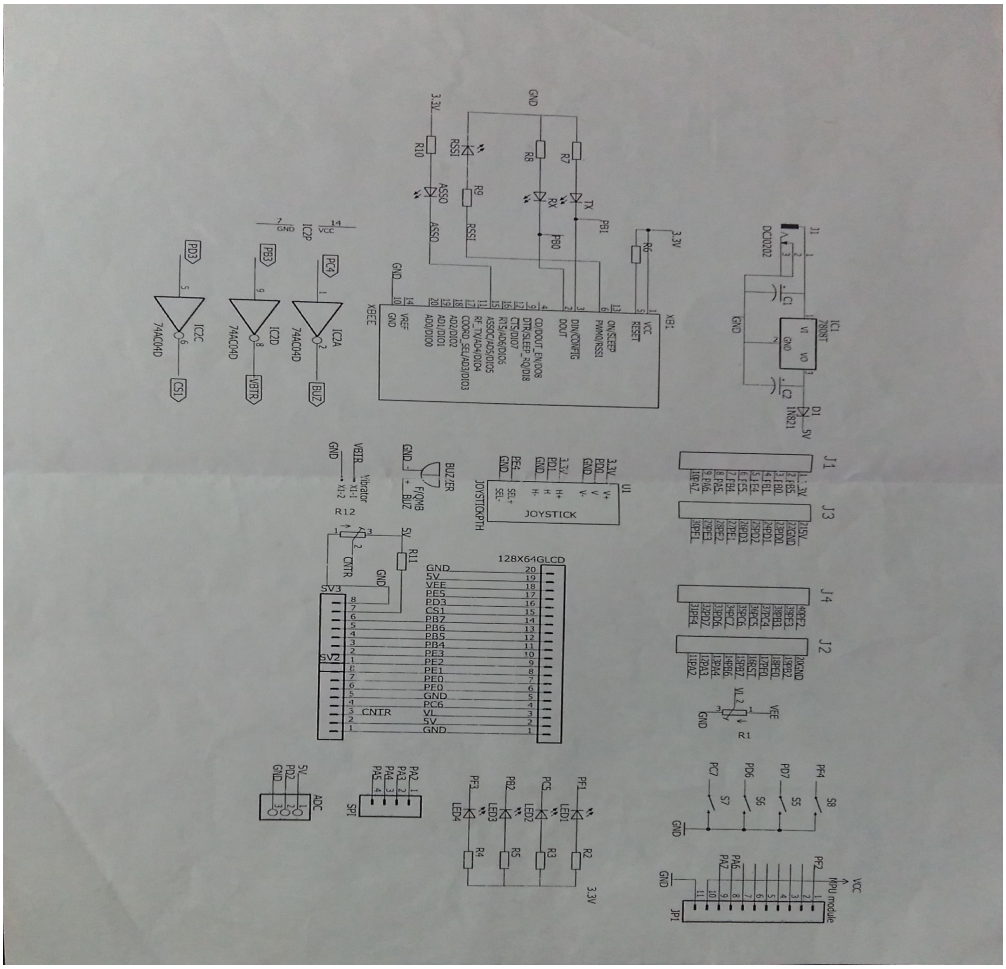
Completion status

The codes for interfacing push buttons, LEDs, 16X2 LCD, 128X64 GLCD and Analog Thumb Joystick have been completed.

8.1 Hardware parts

- TIVA C Series TM4C123G LaunchPad Evaluation Kit
  User's Guide
  Datasheet
  Peripheral Driver Library
- Joystick
- 16X2 LCD
  JHD162A Datasheet
- 128X64 GLCD
  JHD12864A Datasheet

- Schematic of Development Board



8.2 Software used

- Code Composer Studio
  Version 6.1.2
  CCS Download Link
  CCS Installation steps

- TivaWare_C_Series-2.1.2.111
  Download Link

8.3 Assembly of hardware

1. PC7 - SW1
2. PD6 - SW2
3. PD7 - SW3
4. PF4 - SW4
5. PF1 - LED1
6. PC5 - LED2
7. PB2 - LED3
8. PF3 - LED4
9. PD0 - V of Joystick
10. PD1 - H of Joystick
11. PE0 to PE3 - D0 to D3 of LCD and GLCD
12. PB4 to PB7 - D4 to D7 of LCD and GLCD
13. PC6 - RS for LCD and GLCD.
14. PF0 - Enable of LCD and GLCD.
15. PD3 - CS2 for GLCD.
16. Not PD3 - CS1 for GLCD.
17. PE5 - RST of GLCD.

8.4 Procedure

8.4.1 To program push buttons and LEDs

1. Include all the required header files

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"
```

2. Enable the System Clock, PORT B, PORT C, PORT D, PORT F. (*Assembly of harware* section gives details about the PORT pins used for different peripherals)

3. Configure the port pins connected to LEDs as output and those connected to push buttons as input.

4. Unlock pin PF0.

5. Read the status of switch and turn on the corresponding LED if switch press is detected.

6. Switch debouncing can also be added if required.

8.4.2 To program 16X2 LCD

1. Include all the necessary header files.

2. Enable System Clock, PORT B, PORT C, PORT E and PORT F.Unlock pin PF0.

3. Refer section *Assembly of hardware* and configure all the required PORT pins as output. (Only LCD write function is implemented hence all pins are output pins)

4. Initialize the LCD in 8 bit mode.

5. Pass the data to be written on LCD to the data lines.

8.4.3 Programming Joystick using ADC

1. Inlcude all the required header files. Ensure that the following header file is included.

```
#include "driverlib/adc.h"
```

2. Enable the System Clock and PORT D.
3. PD0 and PD1 are ADC channels CH7 and CH6 respectively. Configure these pins as ADC input.
4. Configure ADC Sequencer as per requirement.
5. Enable the ADC.
6. Store the ADC converted data in a variable.
7. This data can be divided by a suitable value.
8. The final result can be displayed in Watch window, on LCD or on Computer using UART. The movement of Joystick can also be displayed on GLCD.

8.4.4 Programming 128X64 GLCD

1. To dislay image on GLCD the hex values are calculated. These values are stored in a array in a header file. This header file is included in main program. The array elements can be accessed from the program and written on GLCD data lines.
2. Include all the necessary header files.
3. Enable the System Clock, PORT D, PORT E, PORT F, PORT C.
4. Refer the section *Assembly of hardware* for information regarding connection of PORT pins with GLCD.
5. All the PORT pins are configured as output pins.
6. Initialize and clear the GLCD.
7. To set the GLCD page, page number is logically ORed with 0xB8 and the result is sent to GLCD as GLCD command.
8. To set the GLCD column, column number is logically ORed with 0x40 and the result is sent to GLCD as GLCD command.
9. To display a line on GLCD, select any page number from 0 to 7, select columns from 0 to 127 one after other and write data 0xFF to GLCD.

8.4.5 To display movement of Joystick on GLCD

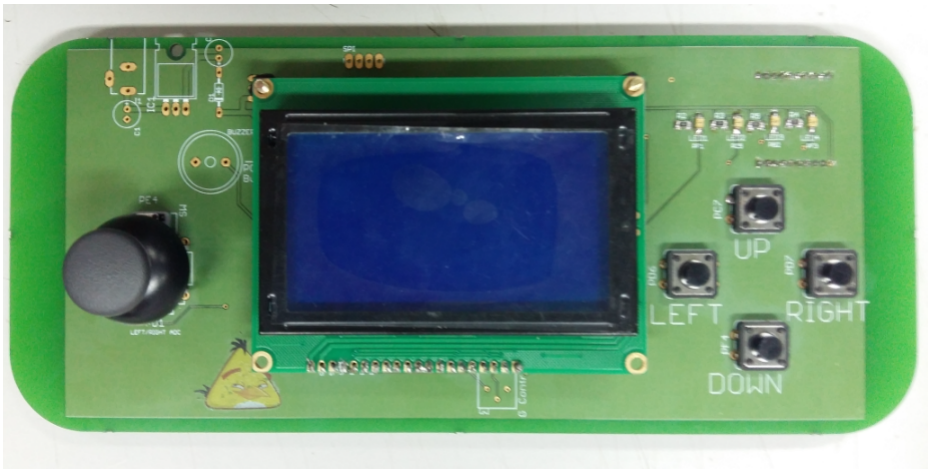
1. Enable and configure System Clock.
2. Configure PORT B, PORT C, PORT D (PD3), PORT E and PORT F as output PORTs.
3. Configure PD0 and PD1 as analog input.(ADC inputs)
4. Configure the ADC.
5. Initialize and clear GLCD.
6. Select a square of 8X8 pixels which will show the position of Joystick on GLCD.
7. To obtain page number the digital value obtained is divided by 512 and for column number the digital value is divided by 32.
8. This value is then sent to GLCD and the movement of Joystick can be observed on GLCD.

8.5 Code

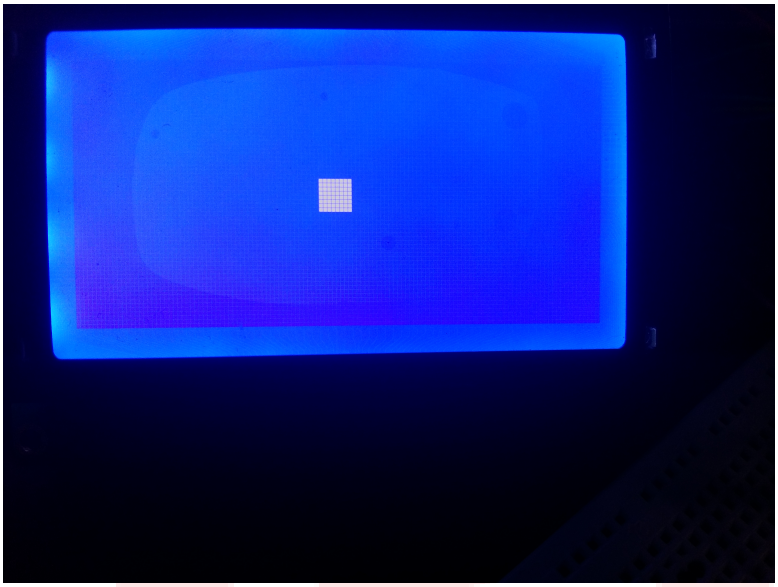
Github link for all the codes required for this project.

8.6 Demonstration

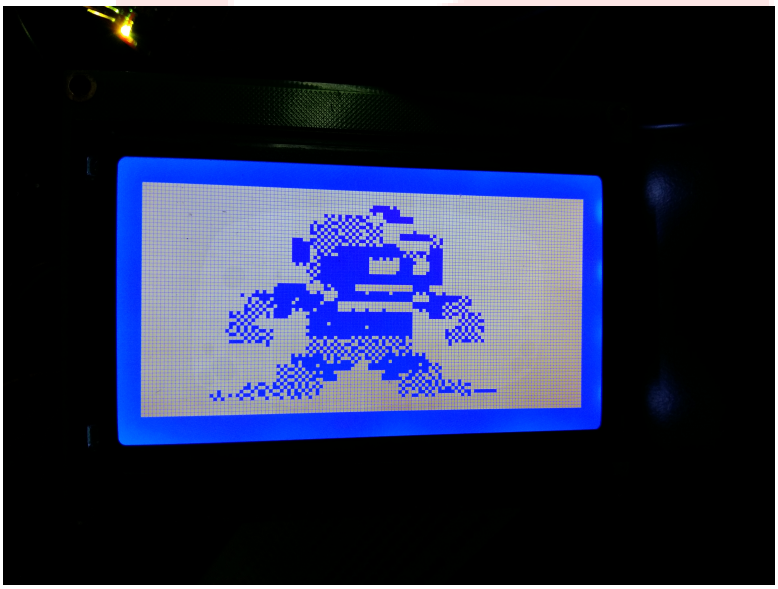
1. Final Setup Image



2. Joystick position displayed on GLCD.



3. Image displayed on GLCD.



8.7 Future Work

A gaming console can be made using Joystick, push button switches and GLCD present on Development Board.

8.8 Challenges

Pin number PB6 and PB7 are internally connected in TIVA Board with PD0 and PD1. PD0 and PD1 are analog input channels of board and PB6, PB7 are data lines D6 and D7 of GLCD. Thus when ADC and GLCD are used simultaneously, expected output is not obtained.
I have replaced data lines PB4 to PB7 of GLCD with PA4 to PA7 and the output is obtained.

Bibliography

[1] 8051 Projects, *LCD Interfacing Tutorial - LCD 4 bit Mode Introduction.* [Online] http://www.8051projects.net/lcd-interfacing/lcd-4-bit.php

[2] Microchip, *16X2 LCD in 4 bit mode.* [Online] http://www.microchip.com/forums/m656930.aspx

[3] TI E2E Community, *Interfacing TIVA C series to 16X2 LCD.* [Online] https://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/t/348669

[4] TI E2E Community,*Interfacing TIVA C series to 16X2 LCD.* [Online] http://www.circuitstoday.com/interfacing-lcd-to-arduino

[5] TI E2E Community, *Connecting Stellaris LaunchPad with 16X2 LCD.* [Online] https://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/t/279206

[6] Circuitstoday,*Interfacing LCD to Arduino - Display text and characters on LCD screen using Arduino.* [Online] https://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/t/385440

[7] Stackoverflow,*Plot data points in python using pylab.* [Online] http://stackoverflow.com/questions/15859534/plot-data-points-in-python-using-pylab

[8] Instructables,*Plotting real time data from Arduino using Python* [Online] http://www.instructables.com/id/Plotting-real-time-data-from-Arduino-using-Python-/

[9] TI E2E Community,*Unlock NMI pin.* [Online] https://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/p/298835/1042023

[10] TI E2E Community,*TM4C123G PORT pins locked* [Online] http://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/p/398941/1410634

[11] TI E2E Community,*Synthesizing a full 8 bit data port - when seemingly unavailable* [Online] http://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/t/279242