Faculty of Engineering & Technology Electrical Department

COMPUTER DESIGN LABORATORY - ENCS 4110

Experiment#10: ADC– Measure Analog Voltage Signal with TM4C123

Report#3

Prepared by: Rahaf Naser

ID: 1201319

Instructor: Dr. Abdelsalam sayyad

Teaching Assistant: Eng. Raha Zabadi

Section: 3

Date: 9/6/2024

Abstract

The aim of this experiment to learn how to use the analog to digital module (ADC) of TM4C123GH6PM Microcontroller using TM4C123G Tiva C Launchpad. Firstly, we will learn to configure ADC modules and sample sequencer of TM4C123 using configuration registers. For demonstration purposes, we will measure analog voltage by using one of the analog input pins of TM4C123GH6PM microcontroller. Finally, we will discuss both polling and interrupt-based approach to use TM4C123 MCU ADC.

Table of contents

Abstract……..…….…I

1.Theory……..……1

1.1.TM4C123 Microcontroller ADC Introduction…..……1

1.2.ADC Resolution……..……..1

1.3.Sample Sequencers……..……..2

1.4.Sample Sequencers……..……..2

1.5.Sample Sequencers……..……..2

1.6.Configure PORT as an Analog Pin……..……..2

1.6.1.Activate ADC SS……..……..3

1.6.2. Sampling Option……..……..3

1.6.3. Analog Channel Selection…….……3

1.6.4.ADC Sampling Rate Setting…..……..3

1.7.Schematic Diagram …….…….4

2.Procedure and Results…….…….5

2.1.ADC Code TM4C123G…….…….5

2.1.1.ADC Code TM4C123G steps…….…….6

2.1.2.ADC Code TM4C123G result discussion…….……..7

2.2. Enable TM4C123G AD Interrupt…….……...…..7

2.3. Enable TM4C123 ADC Interrupt Mask…….……7

2.3.1. Enable TM4C123 ADC Interrupt Mask steps…….……..9

2.3.2. Enable TM4C123 ADC Interrupt Mask result discussion……9

3.Lab Works……9

3.1.Lab work 1……9

3.1.1.Lab work 1 steps …….....9

3.1.2.Lab work 1 result……...…….……10

3.2.Lab work2……10

3.2.1.Lab work2 steps…….11
3.2.2.Lab work2 result……..…...12
4.Conclusion……..…13
5.References……..….14
6.Appendices……..……14
6.1.Example1 code…….……15
6.2.Example2 code …….…...16
6.3.Labwork1 code…….……17
6.4.Labwork2 code…….17

List of Figures

Figure 1: ADC modules[1]……..1

Figure2: schematic diagram[1]……4

Figure3: Example1 code……..5

Figure4: Example1 result……6

Figure5 : Example2 code……8

Figure6 : Example2 code……9

Figure7: Lab Work1 result……10

Figure8: Lab Work2 result……11

List of Tables

Table1: GPIO pins[1]…….1

Table2: number of samples and the length of FIFO[1]……..2

1.Theory

1.1.TM4C123 Microcontroller ADC Introduction

It has two identical successive Approximation Register (SAR) architecture based ADC modules such as ADC0 and ADC1 which share 12 analog input channels as shown in the Figure1 below, Each ADC input channel supports 12-bit conversion resolution. Furthermore, it has a built-in temperature sensor and a programmable sequencer to sample multiple input analog sources [1].

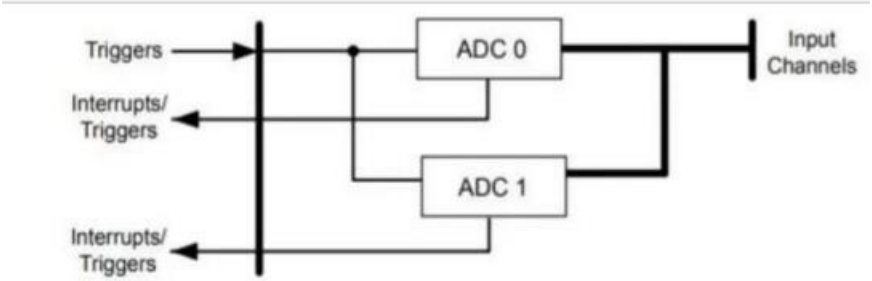

Figure 1: ADC modules[1].

The following table shows the GPIO pins which are used for alternate functions with analog input channels [1].

Analog channel	AN0	AN1	AN2	AN3	AN4	AN5	AN6	AN7	AN8	AN9	AN10	AN11
Pin name	PE3	PE2	PE1	PE0	PD3	PD2	PD1	PD0	PE5	PE4	PB4	PB5
Pin no	6	7	8	9	64	63	62	61	60	59	58	57

Table1: GPIO pins[1].

1.2.ADC Resolution

Digitizers convert the samples of an analog signal into digital values using analog to digital converters (ADCs). The resolution of the ADC is the number of bits it uses to digitize the input samples. For an n bit ADC the number of discrete digital levels that can be produced is 2n. Thus, a 12 bit digitizer can resolve 212 or 4096 levels. The least significant bit (lsb) represents the smallest interval that can be detected and in the case of a 12 bit digitizer is 1/4096 or 2.4 x 10-4. To convert the lsb into a voltage we take the input range of the digitizer and divide by two raised to the resolution of the digitizer [2].

1.3.Sample Sequencers

TM4C123 microcontroller has two ADC modules that are ADC0 and ADC1. Each analog to digital converter has a separate sample sequencer (SS0, SS1, SS2 and SS3) which can sample different input channels. All these sequencers offer different numbers of samples that they can capture and the length of FIFO. M4C123GH6PM microcontroller supports sampling rate from 125 KSPS to 1 MSPS. The following table shows the number of samples and the length of FIFO for each sequencer circuit [1].

Sequencer	Number of Samples	Depth of FIFO
SS3	1	1
SS2	4	4
SS1	4	4
SS0	8	8

Table2: number of samples and the length of FIFO[1].

1.4.TM4C123G ADC Configuration and Initialization steps

In this section, we will discuss the different registers that are used to configure ADC input channels and sample sequencers of TM4C123 Tiva C Launchpad. Followings are the steps to enable ADC module of TM4C123 Tiva C Launchpad[1]:

1.5.Enable Clock to ADC

First, we enable the clock to ADC module and to the GPIO pin which is used as an analog input. RCGCADC register is used to enable the clock to ADC0 and ADC1 modules. Bit0 of RCGCADC enables ADC0 and bit1 of RCGCADC register enables ADC1. Setting the corresponding bit enables and clearing disables the clock to the corresponding analog to digital converter[1].

RCGCGPIO register is used to enable clock to the related GPIO port pin which will be used to measure analog input. For example, we are using analog channel zero or AN0. As we mentioned earlier, AN0 takes analog signals from the PE3 pin of PORTE. Setting the 5th bit of RCGCGPIO register enables the clock to PORTE of TM4C123 microcontroller [3].

1.6.Configure PORT as an Analog Pin

Sample Sequencer Configuration

The sample sequencer is a component of the TM4C123 microcontroller's ADC modules, as we already discussed. In addition to storing the conversion outcomes in FIFO, it is used to obtain ADC samples [4].

To configure the sample sequencer in the TM4C123 microcontroller's ADC module, follow these steps:

1.6.1.Activate ADC SS

ADCACTSS (active sample sequencer) register is used to enable or disable sample sequences SS0, SS1, SS2 and SS3. For example, we will be using SS3 in this experiment. Setting ASEN3 bit of ADCACTSS register enables the SS3 and clearing it disables the SS3. Before configuring ADC channel, we must disable the sample sequencer by clearing bit3 like this [1]:

ADC0->ACTSS &= ~(1<<3);

1.6.2. Sampling Option

ADCEMUX register selects the trigger option to start sampling of an input signal for each sample sequencer. Trigger event sources are processor, PWM, analog comparators, external interrupts, and software. The default trigger option is by software. This line selects a software event as a start conversion trigger [1].

ADC0->EMUX &= ~0xF000;

1.6.3. Analog Channel Selection

TM4C123G microcontroller provides 12 analog channels. ADC-SSMUXn registers (where n=0, 1,2,3) select analog channels for sample sequencers. For example, if we want to use SS3 and analog channel A0, this line will select AN0 analog channel for sample sequencer 3 or SS3 [1].

ADC0->SSMUX3 = 0;

1.6.4.ADC Sampling Rate Setting

ADCPC register is used to set the sampling rate of ADC. The Analog to digital converter module of the TM4C123G microcontroller supports a sampling rate from 125 KSPS to 1 MSPS. First four bits of the ADCPC register are used to set the sampling rate. This line sets the sampling rate to 250ksps [1].

ADC0->PC = 0x3;

After ADC initialization and configuration, set the SS3 bit of ADCPSSI register to start ADC conversion for sample sequencer 3.

ADC0->PSSI |= (1<<3);

1.7.Schematic Diagram

Schematic diagrams can also differ in their level of abstraction. Although they are typically composed of only abstract symbols and lines, some diagrams can also be **semi-schematic** and contain more realistic elements. Some diagrams can also contain words, such as when a process contains multiple elements that have not been standardized [5].

More simply, a schematic diagram is a simplified drawing that uses symbols and lines to convey important information. For example, if you are taking the subway you may see a “map” showing you all the stations along a subway line, but that map will not show all the roads and buildings you may pass along the way. In this case, the entire subway system can be represented as differently colored lines depicting the different subway routes, with dots indicating the stops along the lines [5].

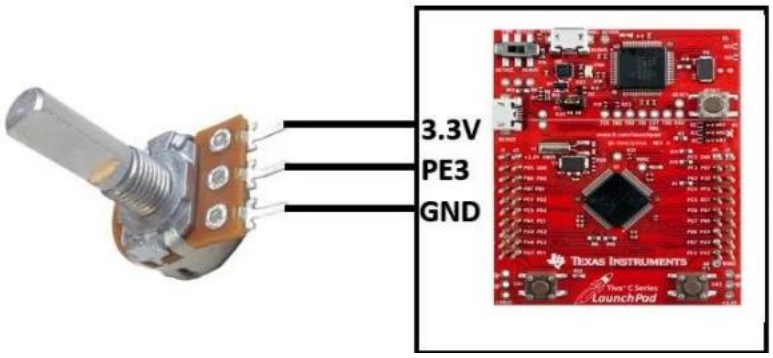

Figure2: schematic diagram[1].

2.Procedure and Results

2.1.ADC Code TM4C123G

This ADC code of TM4C123GH6PM microcontroller reads analog input from AN0 pin and based on digital value turns on or turns off the onboard green LED of TM4C123G Tiva C launchpad. If measured digital value is less than 2048 LED will remain off. If ADC digital value is greater than or equal to 2048, LED turns on.

```
/* TM4C123G Tiva C Series ADC Example */
/* This Program controls the onboard green LED based on
/* If AN0 channel value is less 2048 digital value than
#include "TM4C123GH6PM.h"
#include <stdio.h>
//Functions Declaration
volatile unsigned int adc_value;
void ADC0SS3_Handler(void){
adc_value = ADC0->SSFIFO3; /* read adc coversion result
ADC0->ISC = 8; /* clear coversion clear flag bit*/
 ADC0->PSSI |= (1<<3); /* Enable SS3 conversion or start
}
int main(void)
{
 volatile float voltage;
 /* Enable Clock to ADC0 and GPIO pins*/
 SYSCTL->RCGCGPIO |= (1<<4); /* Enable Clock to GPIOE or
 SYSCTL->RCGCADC |= (1<<0); /* AD0 clock enable*/

 /* initialize PE3 for AIN0 input */
 GPIOE->AFSEL |= (1<<3); /* enable alternate function */
 GPIOE->DEN &= ~(1<<3); /* disable digital function */
 GPIOE->AMSEL |= (1<<3); /* enable analog function */

 /* initialize sample sequencer3 */
 ADC0->ACTSS &= ~(1<<3); /* disable SS3 during configura
 ADC0->EMUX &= ~0xF000; /* software trigger conversion *
 ADC0->SSMUX3 = 0; /* get input from channel 0 */
 ADC0->SSCTL3 |= (1<<1)|(1<<2); /* take one sample at a
 ADC0->ACTSS |= (1<<3); /* enable ADC0 sequencer 3 */

 /*Iniitialize PF3 as a digital output pin */

 SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIO
 GPIOF->DIR |= 0x08; //set GREEN pin as a digital output
 GPIOF->DEN |= 0x08; // Enable PF3 pin as a digital pin

 while(1)
 {

 ADC0->PSSI |= (1<<3); /* Enable SS3 conversi
 while((ADC0->RIS & 8) == 0) ; /* Wait untill
 adc_value = ADC0->SSFIFO3; /* read adc cover
 ADC0->ISC = 8; /* clear coversion clear flag
 /* convert digital value back into voltage *
 voltage = (adc_value * 0.0008);
 if(adc_value >= 2048)
   GPIOF->DATA = 0x08; /* turn on green LED*/
 else if(adc_value < 2048)
 GPIOF->DATA = 0x00; /* turn off green LED*/
 }
}
```

Figure3: Example1 code

2.1.1.ADC Code TM4C123G steps

To accomplish the task of reading an analog input from the AN0 pin and controlling the onboard green LED of the TM4C123GH6PM microcontroller based on the ADC value, we follow these steps:

1.**Setup and Configuration: Enable Clocks:** Enable the clock for GPIO Port E (where AN0 is located) and the ADC module (ADC0).**Then Configure GPIO Pin:** Configure the PE3 pin for analog input by enabling its alternate function, disabling its digital function, and enabling its analog function. Then **Initialize ADC:** Disable the sample sequencer 3 (SS3) during configuration. Configure it for software-triggered conversion. Set it to read from channel 0 (AN0). Enable the sample sequencer after configuration.

2.Configure LED:Enable Clocks: Enable the clock for GPIO Port F (where the green LED is located). Then **Set Pin as Output:** Set PF3 as a digital output pin and enable it.

3.Read ADC Value and Control LED: Start Conversion: Trigger the ADC conversion on SS3.

Then Wait for Conversion to Complete: Poll the RIS register to wait until the conversion is complete. Then **Read ADC Value:** Read the conversion result from the SSFIFO3 register.

Then Clear Interrupt: Clear the conversion complete flag by writing to the ISC register.

Then Control LED: Compare the ADC value to 2048. If the value is greater than or equal to 2048, turn on the green LED (set PF3). Otherwise, turn off the green LED (clear PF3).

2.1.2.ADC Code TM4C123G result discussion

This code configures the TM4C123GH6PM microcontroller to read an analog input from pin PE3 (AN0) using the ADC module. It continuously monitors the ADC value, and if this value is greater than or equal to 2048, it turns on the onboard green LED. If the value is less than 2048, the LED remains off. This demonstrates a basic use of the ADC in conjunction with digital output control on the Tiva C series launchpad.

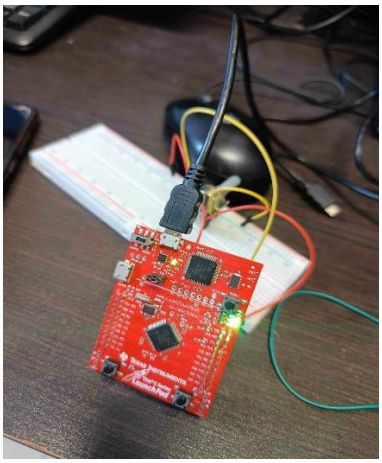

Figure4: Example1 result

2.2. Enable TM4C123G ADC Interrupt

In the previous example, we used a polling method to measure analog signal value with TM4C123 microcontroller ADC. However, the polling method is not an efficient method and it's a wastage of microcontroller processing time and resources. Instead of using a polling method where we keep polling for ADC conversion to complete using bit3 of ADCRIS register, we can use the ADC interrupt handler function to read the ADC conversion value. As we have seen in the previous example, we keep polling for ADC conversion to complete using this line:
while((ADC0->RIS & 8);

TM4C123 microcontroller keeps executing this loop and will not perform any useful function until ADC conversion is not completed. Hence, to avoid this wastage of microcontroller processing time, we can enable ADC data reception using ADC0 handler function.

2.3. Enable TM4C123 ADC Interrupt Mask

In order to enable ADC0 interrupt, ADC Interrupt Mask (ADCIM) is used. Setting bit3 of the ADCIM register enables the SS3 Interrupt Mask. As you know that in ARM Cortex-M4 microcontroller, NVIC or nested vectored interrupt controller manages all interrupts.

TM4C123 microcontroller supports 78 peripheral interrupts which are also known as IRQs. Before using each peripheral interrupt handler, we should enable each peripheral interrupt request to each NVIC using NVIC interrupt control register. The interrupt number of ADC0SS3_IRQn is 17.

This line enables ADC0 sequencer 3 interrupt in NVIC.

NVIC->ISER[0] |= 0x00010001;

/* enable IRQ17 for ADC0SS3*/ Exception numbers are defined inside the header file of TM4C123GH6PM microcontroller and interrupt vector table. The next step is to find the name of the interrupt handler function of ADC0SS3_IRQn inside the startup file of TM4C123G microcontroller. If we check startup file:

ADC0Seq3_Handler () Now receive data from the FIFO register of SS3 inside this function. This ADC0Seq3_Handler () function wi ll execute whenever interrupt occurs due to sample sequencer 3.

2.3.1. Enable TM4C123 ADC Interrupt Mask steps

The below code sets up the TM4C123GH6PM microcontroller to read analog input from the AN0 pin using the ADC module, process it via an interrupt service routine (ISR), and control an onboard green LED based on the ADC value. Initially, clocks for GPIO Port E and the ADC module are enabled. The PE3 pin is configured for analog input. Sample sequencer 3 (SS3) is configured for software-triggered ADC conversion, reading from channel 0 (AN0), and is set to generate an interrupt upon completion. The LED on PF3 is configured as a digital output. The

ADC interrupt is enabled, allowing the ADC0SS3_Handler ISR to execute upon conversion completion, where it reads the conversion result, clears the interrupt flag, and restarts the conversion. In the main loop, the ADC value is continuously monitored and compared to a threshold. If the value is 2048 or higher, the green LED is turned on; otherwise, it is turned off. This setup demonstrates efficient ADC usage with interrupt handling for real-time analog input monitoring and digital output control.

```
/* TM4C123G Tiva C Series ADC Example */
/* This Program controls the onboard green LED based
/* If AN0 channel value is less 2048 digital value tha
#include "TM4C123GH6PM.h"
#include <stdio.h>
//Functions Declaration
void delayUs(int); //Delay in Micro Seconds
volatile unsigned int adc_value;
void ADC0SS3_Handler(void){
adc_value = ADC0->SSFIFO3; /* read adc coversion resul
ADC0->ISC = 8; /* clear coversion clear flag bit*/
 ADC0->PSSI |= (1<<3); /* Enable SS3 conversion or sta
}
int main(void)
{
char* str = "Tiva C starting"; //Write any string you
char s[20];
volatile float voltage;
 /* Enable Clock to ADC0 and GPIO pins*/
 SYSCTL->RCGCGPIO |= (1<<4); /* Enable Clock to GPIOE
 SYSCTL->RCGCADC |= (1<<0); /* AD0 clock enable*/

 /* initialize PE3 for AIN0 input */
 GPIOE->AFSEL |= (1<<3); /* enable alternate function
 GPIOE->DEN &= ~(1<<3); /* disable digital function */
 GPIOE->AMSEL |= (1<<3); /* enable analog function */

 /* initialize sample sequencer3 */
 ADC0->ACTSS &= ~(1<<3); /* disable SS3 during configu
 ADC0->EMUX &= ~0xF000; /* software trigger conversion
 ADC0->SSMUX3 = 0; /* get input from channel 0 */
 ADC0->SSCTL3 |= (1<<1)|(1<<2); /* take one sample at
 ADC0->ACTSS |= (1<<3); /* enable ADC0 sequencer 3 */

 SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GP
 GPIOF->DIR |= 0x08; //set GREEN pin as a digital outp
 GPIOF->DEN |= 0x08; // Enable PF3 pin as a digital pi
 /* Enable ADC Interrupt */
 ADC0->IM |= (1<<3); /* Unmask ADC0 sequence 3 interru
 NVIC->ISER[0] |= 0x00020000; /* enable IRQ17 for ADC0
 NVIC->ISER[0] |= 0x00020000; /* enable IRQ17 f
 ADC0->ACTSS |= (1<<3); /* enable ADC0 sequence
 ADC0->PSSI |= (1<<3); /* Enable SS3 conversion
 while(1)
 {
/*control Green PF3->LED */
 /* convert digital value back into voltage */
 voltage = (adc_value * 0.0008);
 // sprintf(s, "\r\nVoltage = %f", voltage);
 if(adc_value >= 2048)
 GPIOF->DATA = 0x08; /* turn on green LED*/
 else if(adc_value < 2048)
 GPIOF->DATA = 0x00; /* turn off green LED*/
 }
}
}
```

Figure5 : Example2 code

2.3.2. Enable TM4C123 ADC Interrupt Mask result discussion

This code sets up the ADC to continuously sample analog input from the AN0 pin and process the result using an interrupt-driven approach. The ISR (ADC0SS3_Handler) reads the ADC value, restarts the ADC conversion, and clears the interrupt flag. The main loop checks the ADC value and controls the onboard green LED based on the threshold value of 2048. If the ADC value is 2048 or higher, the green LED is turned on; otherwise, it is turned off. This demonstrates how to use ADC interrupts for efficient analog-to-digital conversion and digital output control on the Tiva C series launchpad.

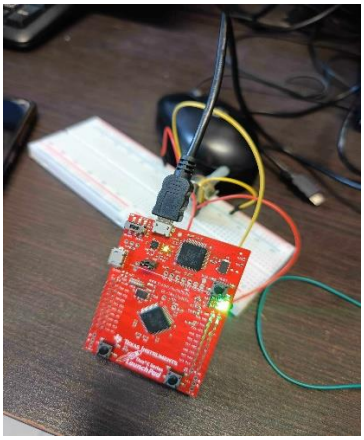

Figure6 : Example2 code

3.Lab Works

3.1.Lab work 1

3.1.1.Lab work 1 steps

The modified code includes changes in the main function and the ADC interrupt handler :

- Inside the ADC0SS3_Handler function The if-else statement has been modified to control the state of the LEDs based on the ADC value.

- If the adc_value is greater than 1000, the RED LED (connected to pin PF1) is turned on by setting GPIOF->DATA to 0x02.

- If the adc_value is less than or equal to 1000, the GREEN LED is turned on by setting

GPIOF->DATA to 0x08. These changes ensure that the RED LED is only on when the ADC value is greater than 1000, and the GREEN LED is only on when the ADC value is below 1000.

3.1.2.Lab work 1 result

The Modified code configures the TM4C123GH6PM microcontroller to control two onboard LEDs (red and green) based on the ADC value from the AN0 pin. Initially, clocks are enabled for the necessary GPIO ports and the ADC module. The PE3 pin is configured for analog input.

Sample sequencer 3 (SS3) is set up for software-triggered ADC conversion with an interrupt upon completion. PF1 (red LED) and PF3 (green LED) are configured as digital output pins.

The ADC interrupt is enabled, allowing the ADC0SS3_Handler ISR to handle conversion results. In the ISR, the ADC value is read, the interrupt flag is cleared, and the conversion is restarted. In the main loop, the ADC value is continuously monitored: if the value is greater than 1000, the red LED (PF1) is turned on; if the value is below 1000, the green LED (PF3) is turned on. This demonstrates real-time analog input monitoring with conditional digital output control for two LEDs.

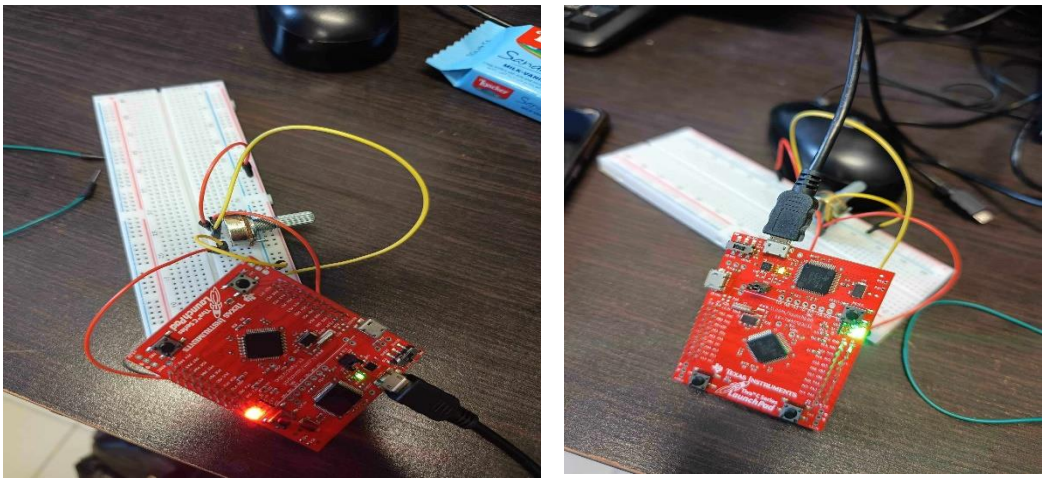

Figure7: Lab Work1 result

3.2.Lab work2

3.2.1.Lab work2 steps

The modified code incorporates an LCD to display the ADC input value and the corresponding voltage, in addition to controlling two LEDs based on the ADC value.**Initialization and Configuration**:

- Functions for initializing the LCD (LCD_init), sending commands to the LCD (LCD_command), sending data to the LCD (LCD_data), and printing strings to the LCD (LCD_print) are added.

- Clocks for GPIO ports A and B are enabled. Port A pins 5, 6, and 7 are used for LCD control signals (RS, RW, and E), and all pins of Port B are used for LCD data signals.

- The ADC configuration remains similar, initializing PE3 for analog input and configuring the ADC to use sample sequencer 3 (SS3) for conversion.

- The ADC0SS3_Handler function reads the ADC value from the SS3 FIFO and restarts the ADC conversion. This is where the ADC value is updated upon each conversion.

- The ADC value is converted to a voltage using the formula voltage = (adc_value * 3.3) / 4096.0.

-The LCD is cleared, and the ADC input value is displayed on the first line.The voltage is displayed on the second line.

- The red LED (PF1) is turned on if the ADC value is greater than 1000, and the green LED (PF3) is turned on if the ADC value is below 1000.

3.2.2.Lab work2 result

This part sets up the TM4C123GH6PM microcontroller to read an analog input, display the ADC value and corresponding voltage on an LCD, and control two LEDs based on the ADC value. The ADC interrupt handler processes conversion results, while the main loop updates the LCD and manages LED states. The red LED is on when the ADC value exceeds 1000, and the green LED is on when it is below 1000. This setup provides real-time feedback on both the LCD and LEDs, showcasing analog input monitoring and digital output control.

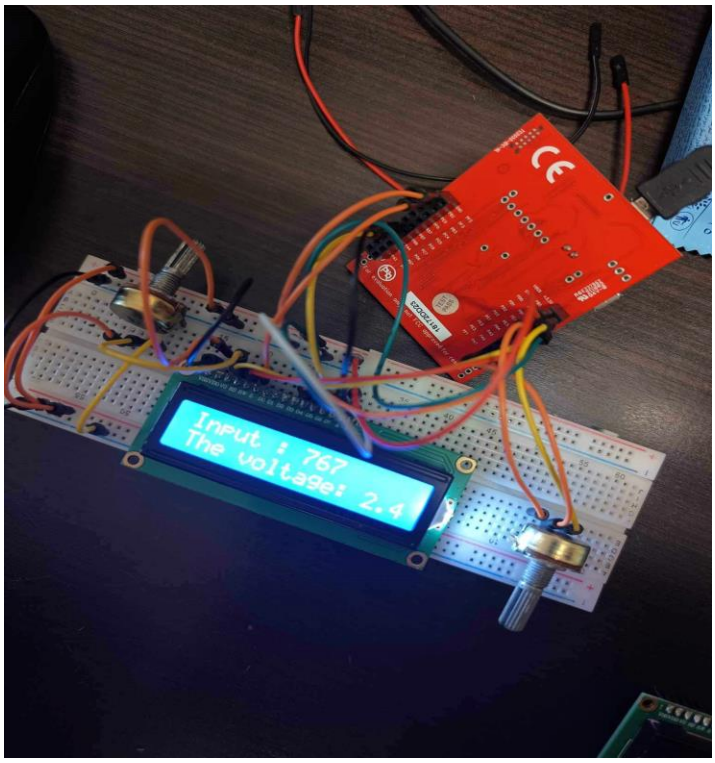

Figure8: Lab Work2 result

4.Conclusion

In conclusion, after finishing the experiment we became know how to use the analog to digital module (ADC) of TM4C123GH6PM Microcontroller using TM4C123G Tiva C Launchpad. And we became know how to configure ADC modules and sample sequencer of TM4C123 using configuration registers. For demonstration purposes, we became able to measure analog voltage by using one of the analog input pins of TM4C123GH6PM microcontroller. We studied both polling and interrupt-based approach to use TM4C123 MCU ADC.

5.References

[1] Lab manual. [Accessed on 12 june at 5:15]

[2]https://spectruminstrumentation.com/support/knowledgebase/hardware_features/ADC_and_Resolution.php , [Accessed on 12 june at 6:15]

[3] https://www.nxp.com/docs/en/application-note/AN5250.pdf , [Accessed on 12 june at 5:30]

[4] https://learn.sparkfun.com/tutorials/analog-to-digital-conversion/all, [Accessed on 11 june at 5:15]

[5] https://www.thoughtco.com/what-is-a-schematic-diagram-4584811 , [Accessed on 10 june at 10:1]

6.Appendices

6.1.Example1

```
/* TM4C123G Tiva C Series ADC Example */

/* This Program controls the onboard green LED based on discrete digital value of ADC */
/* If AN0 channel value is less 2048 digital value than LED turns off and otherwise remain on */
#include "TM4C123GH6PM.h"
#include <stdio.h>

//Functions Declaration
volatile unsigned int adc_value;

void ADC0SS3_Handler(void){
        adc_value = ADC0->SSFIFO3; /* read adc coversion result from SS3 FIFO*/
        ADC0->ISC = 8;                  /* clear coversion clear flag bit*/
       ADC0->PSSI |= (1<<3);      /* Enable SS3 conversion or start sampling data from AN0 */

}

int main(void)
{
   volatile float voltage;
  /* Enable Clock to ADC0 and GPIO pins*/
  SYSCTL->RCGCGPIO |= (1<<4);  /* Enable Clock to GPIOE or PE3/AN0 */
  SYSCTL->RCGCADC |= (1<<0);   /* AD0 clock enable*/

  /* initialize PE3 for AIN0 input  */
  GPIOE->AFSEL |= (1<<3);     /* enable alternate function */
  GPIOE->DEN &= ~(1<<3);      /* disable digital function */
  GPIOE->AMSEL |= (1<<3);     /* enable analog function */

  /* initialize sample sequencer3 */
  ADC0->ACTSS &= ~(1<<3);      /* disable SS3 during configuration */
  ADC0->EMUX &= ~0xF000;   /* software trigger conversion */
  ADC0->SSMUX3 = 0;       /* get input from channel 0 */
  ADC0->SSCTL3 |= (1<<1)|(1<<2);      /* take one sample at a time, set flag at 1st sample */
  ADC0->ACTSS |= (1<<3);       /* enable ADC0 sequencer 3 */

        /*Iniitialize PF3 as a digital output pin */

   SYSCTL->RCGCGPIO |= 0x20;  // turn on bus clock for GPIOF
   GPIOF->DIR      |= 0x08;        //set GREEN pin as a digital output pin
   GPIOF->DEN      |= 0x08;      // Enable PF3 pin as a digital pin


  while(1)
  {

       ADC0->PSSI |= (1<<3); /* Enable SS3 conversion or start sampling data from AN0 */
       while((ADC0->RIS & 8) == 0) ; /* Wait untill sample conversion completed*/
      adc_value = ADC0->SSFIFO3; /* read adc coversion result from SS3 FIFO*/
      ADC0->ISC = 8; /* clear coversion clear flag bit*/

        /* convert digital value back into voltage */
         voltage = (adc_value * 0.0008);

         if(adc_value >= 2048)

            GPIOF->DATA  = 0x08; /* turn on green LED*/
        else if(adc_value < 2048)
            GPIOF->DATA  = 0x00; /* turn off green LED*/
  }
}
```

6.2.Example2

```
/* TM4C123G Tiva C Series ADC Example */

/* This Program controls the onboard green LED based on discrete digital value of ADC */
/* If AN0 channel value is less 2048 digital value than LED turns off and otherwise remain on */
#include "TM4C123GH6PM.h"
#include <stdio.h>

//Functions Declaration
void delayUs(int);      //Delay in Micro Seconds
volatile unsigned int adc_value;

void ADC0SS3_Handler(void){
        adc_value = ADC0->SSFIFO3; /* read adc coversion result from SS3 FIFO*/
        ADC0->ISC = 8;                  /* clear coversion clear flag bit*/
       ADC0->PSSI |= (1<<3);      /* Enable SS3 conversion or start sampling data from AN0 */

}

int main(void)
{
        char* str = "Tiva C starting";   //Write any string you want to display on LCD
        char s[20];
        volatile float voltage;

  /* Enable Clock to ADC0 and GPIO pins*/
  SYSCTL->RCGCGPIO |= (1<<4);  /* Enable Clock to GPIOE or PE3/AN0 */
  SYSCTL->RCGCADC |= (1<<0);   /* AD0 clock enable*/

  /* initialize PE3 for AIN0 input  */
  GPIOE->AFSEL |= (1<<3);      /* enable alternate function */
  GPIOE->DEN &= ~(1<<3);       /* disable digital function */
  GPIOE->AMSEL |= (1<<3);      /* enable analog function */

  /* initialize sample sequencer3 */
  ADC0->ACTSS &= ~(1<<3);      /* disable SS3 during configuration */
  ADC0->EMUX &= ~0xF000;   /* software trigger conversion */
  ADC0->SSMUX3 = 0;       /* get input from channel 0 */
  ADC0->SSCTL3 |= (1<<1)|(1<<2);       /* take one sample at a time, set flag at 1st sample */
  ADC0->ACTSS |= (1<<3);       /* enable ADC0 sequencer 3 */
```

```
        /*Iniitialize PF3 as a digital output pin */

   SYSCTL->RCGCGPIO |= 0x20;  // turn on bus clock for GPIOF
   GPIOF->DIR      |= 0x08;        //set GREEN pin as a digital output pin
   GPIOF->DEN      |= 0x08;      // Enable PF3 pin as a digital pin

        /* Enable ADC Interrupt */
  ADC0->IM |= (1<<3);                 /* Unmask ADC0 sequence 3 interrupt*/
  NVIC->ISER[0] |= 0x00020000;   /* enable IRQ17 for ADC0SS3*/
  ADC0->ACTSS |= (1<<3);             /* enable ADC0 sequencer 3 */
  ADC0->PSSI |= (1<<3);              /* Enable SS3 conversion or start sampling data from AN0 */


  while(1)
  {

       /*control Green PF3->LED */
        /* convert digital value back into voltage */
        voltage = (adc_value * 0.0008);
        // sprintf(s, "\r\nVoltage = %f", voltage);

        if(adc_value >= 2048)
        GPIOF->DATA  = 0x08; /* turn on green LED*/
        else if(adc_value < 2048)
        GPIOF->DATA  = 0x00; /* turn off green LED*/
  }
}

}
```

6.3.Lab Work1

```
#include "TM4C123GH6PM.h"
#include <stdio.h>
//Functions Declaration
volatile unsigned int adc_value;
void ADC0SS3_Handler(void){
adc_value = ADC0->SSFIFO3; /* read adc coversi
ADC0->ISC = 8; /* clear coversion clear flag b
ADC0->PSSI |= (1<<3); /* Enable SS3 conversion
}
int main(void)
{
volatile float voltage;
/* Enable Clock to ADC0 and GPIO pins*/
SYSCTL->RCGCGPIO |= (1<<4); /* Enable Clock to
SYSCTL->RCGCADC |= (1<<0); /* AD0 clock enable

/* initialize PE3 for AIN0 input */
GPIOE->AFSEL |= (1<<3); /* enable alternate fu
GPIOE->DEN &= ~(1<<3); /* disable digital func
GPIOE->AMSEL |= (1<<3); /* enable analog funct

/* initialize sample sequencer3 */
ADC0->ACTSS &= ~(1<<3); /* disable SS3 during
ADC0->EMUX &= ~0xF000; /* software trigger con
ADC0->SSMUX3 = 0; /* get input from channel 0
ADC0->SSCTL3 |= (1<<1)|(1<<2); /* take one sam
ADC0->ACTSS |= (1<<3); /* enable ADC0 sequence

/*Iniitialize PF3 as a digital output pin */

SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock
GPIOF->DIR |= 0x02; //set GREEN pin as a digit
GPIOF->DEN |= 0x02; // Enable PF3 pin as a dig
GPIOF->DIR |= 0x08; //set GREEN pin as a digit
```

```
while(1)
{

ADC0->PSSI |= (1<<3); /* Enable SS3 convers
while(ADC0->RIS & ADC0->ISC == 0) ;/* Wait
adc_value = ADC0->SSFIFO3; /* read adc cove
ADC0->ISC = 8; /* clear coversion clear fla
/* convert digital value back into voltage
voltage = (adc_value * 0.0008);
if(adc_value >= 1000)
GPIOF->DATA = 0x02; /* turn on green LED*/
else if(adc_value < 1000)
GPIOF->DATA = 0x08; /* turn off green LED*/
}
}
```

6.4.Lab Work2

```
#include "TM4C123.h" // Device header
#define LCD GPIOB //LCD port with Tiva C
#define RS 0x01 //RS -> PB0 (0x01)
#define RW 0x02 //RW -> PB1 (0x02)
#define EN 0x04 //EN -> PB2 (0x04)
//Functions Declaration
void delayUs(int); //Delay in Micro Seconds
void delayMs(int); //Delay in Milli Seconds
void LCD4bits_Init(void); //Initialization of LCD D
void LCD_Write4bits(unsigned char, unsigned char);
void LCD_WriteString(char*); //Write a string on LC
void LCD4bits_Cmd(unsigned char); //Write command
void LCD4bits_Data(unsigned char); //Write a charac

void delayUs(int); //Delay in Micro Seconds
volatile unsigned int adc_value;
void ADC0SS3_Handler(void){
adc_value = ADC0->SSFIFO3; /* read adc coversion re
ADC0->ISC = 8; /* clear coversion clear flag bit*/
ADC0->PSSI |= (1<<3); /* Enable SS3 conversion or s
}
int main(void)
{
char* str = "Tiva C starting"; //Write any string y
char s[20];
volatile float voltage;
char D[20];
volatile float DC;
LCD4bits_Init(); //Initialization of LCD
LCD4bits_Cmd(0x01); //Clear the display
LCD4bits_Cmd(0x80); //Force the cursor to beginning
delayMs(500); //delay 500ms for LCD (MCU is faster
LCD_WriteString(s); //Write the string on LCD
delayMs(500); //Delay 500 ms to let the LCD displays
```

```
34  delayMs(500); //Delay 500 ms to let the LCD diplays the dat
35  LCD4bits_Cmd(0xC0); //Force the cursor to beginning of 2
36  delayMs(500); //delay 500ms for LCD (MCU is faster than LCD
37  LCD_WriteString(D); //Write the string on LCD
38  delayMs(500); //Delay 500 ms to let the LCD diplays the dat
39  /* Enable Clock to ADC0 and GPIO pins*/
40  SYSCTL->RCGCGPIO |= (1<<4); /* Enable Clock to GPIOE or PE3,
41  SYSCTL->RCGCADC |= (1<<0); /* AD0 clock enable*/
42
43  /* initialize PE3 for AIN0 input */
44  GPIOE->AFSEL |= (1<<3); /* enable alternate function */
45  GPIOE->DEN &= ~(1<<3); /* disable digital function */
46  GPIOE->AMSEL |= (1<<3); /* enable analog function */
47
48  /* initialize sample sequencer3 */
49  ADC0->ACTSS &= ~(1<<3); /* disable SS3 during configuration
50  ADC0->EMUX &= ~0xF000; /* software trigger conversion */
51  ADC0->SSMUX3 = 0; /* get input from channel 0 */
52  ADC0->SSCTL3 |= (1<<1)|(1<<2); /* take one sample at a time,
53  ADC0->ACTSS |= (1<<3); /* enable ADC0 sequencer 3 */
54
55  /*Iniitialize PF3 as a digital output pin */
56
57  SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
58  GPIOF->DIR |= 0x08; //set GREEN pin as a digital output pin
59  GPIOF->DEN |= 0x08; // Enable PF3 pin as a digital pin
60  /* Enable ADC Interrupt */
61  ADC0->IM |= (1<<3); /* Unmask ADC0 sequence 3 interrupt*/
62  NVIC->ISER[0] |= 0x00020000; /* enable IRQ17 for ADC0SS3*/
63  ADC0->ACTSS |= (1<<3); /* enable ADC0 sequencer 3 */
64  ADC0->PSSI |= (1<<3); /* Enable SS3 conversion or start sam
65  while(1)
66  {
67  /*control Green PF3->LED */
```

```
//LCD4bits_Cmd(0x01); //Clear the di
LCD4bits_Cmd(0x80); //Force the curs
delayMs(500); //delay 500ms for LCD
LCD_WriteString(s); //Write the stri
delayMs(500); //Delay 500 ms to let
printf(s, "DC = %d", adc_value);
//LCD4bits_Cmd(0x01); //Clear the di
LCD4bits_Cmd(0xc0); //Force the curs
delayMs(500); //delay 500ms for LCD
LCD_WriteString(s); //Write the stri
delayMs(500); //Delay 500 ms to let
if(adc_value >= 2048)
GPIOF->DATA = 0x08; /* turn on green
else if(adc_value < 2048)
GPIOF->DATA = 0x00; /* turn off gree
}
}
void LCD4bits_Init(void)
{
SYSCTL->RCGCGPIO |= 0x02; //enable c
delayMs(10); //delay 10 ms for enabl
LCD->DIR = 0xFF; //let PORTB as outp
LCD->DEN = 0xFF; //enable PORTB digi
LCD4bits_Cmd(0x28); //2 lines and 5x
LCD4bits_Cmd(0x06); //Automatic Incr
LCD4bits_Cmd(0x01); //Clear display
LCD4bits_Cmd(0x0F); //Display on, cu
```

```
100  void LCD_Write4bits(unsigned char data, unsigned char control)
101  {
102  data &= 0xF0; //clear lower nibble for control
103  control &= 0x0F; //clear upper nibble for data
104  LCD->DATA = data | control; //Include RS value (command or data
105  LCD->DATA = data | control | EN; //pulse EN
106  delayUs(0); //delay for pulsing EN
107  LCD->DATA = data | control; //Turn off the pulse EN
108  LCD->DATA = 0; //Clear the Data
109  }
110  void LCD_WriteString(char * str)
111  {
112  volatile int i = 0; //volatile is important
113  while(*(str+i) != '\0') //until the end of the string
114  {
115  LCD4bits_Data(*(str+i)); //Write each character of string
116  i++; //increment for next character
117  }
118  }
119  void LCD4bits_Cmd(unsigned char command)
120  {
121  LCD_Write4bits(command & 0xF0 , 0); //upper nibble first
122  LCD_Write4bits(command << 4 , 0); //then lower nibble
123  if(command < 4)
124  delayMs(2); //commands 1 and 2 need up to 1.64ms
125  else
126  delayUs(40); //all others 40 us
127  }
128  void LCD4bits_Data(unsigned char data)
129  {
```

```
128  void LCD4bits_Data(unsigned char data)
129  {
130  LCD_Write4bits(data & 0xF0 , RS); //upper nibble first
131  LCD_Write4bits(data << 4 , RS); //then lower nibble
132  delayUs(40); //delay for LCD (MCU is faster than LCD)
133  }
134  void delayMs(int n)
135  {
136  volatile int i,j; //volatile is important for variables i
137  for(i=0;i<n;i++)
138  for(j=0;j<3180;j++) //delay for 1 msec
139  {}
140  }
141  void delayUs(int n)
142  {
143  volatile int i,j; //volatile is important for variables i
144  for(i=0;i<n;i++)
145  for(j=0;j<3;j++) //delay for 1 micro second
146  {}
147  }
```