Lab5

Analog To Digital Converter Application

MICROPROCESSORS LABORATORY

Ömer Karslıoğlu

Youtube Video Link

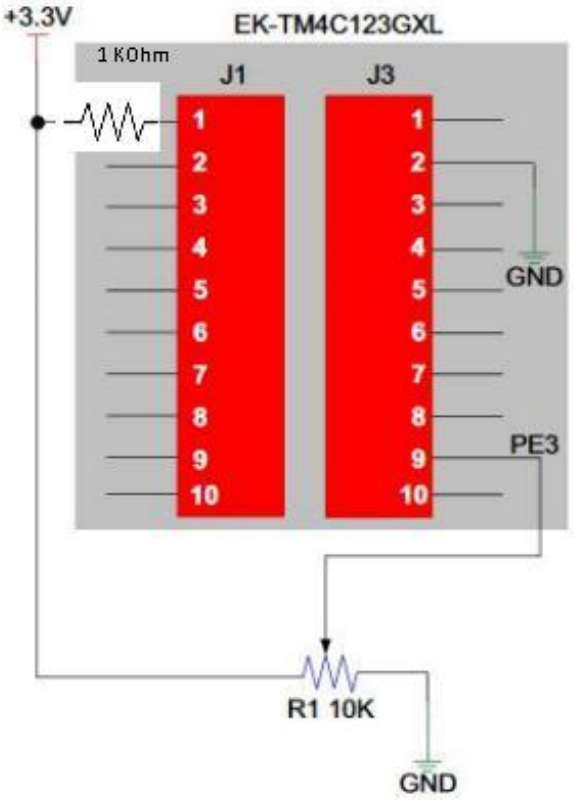
https://youtu.be/fReR-6Vf7V0

Introduction

General Purpose of This Laboratory : The main objective of this experiment is to interface a potentiometer with EKTM4C123G GPIO (PE3) by configuring it as an Analog Input (AN0) and to observe its corresponding 12-bit digital value. In this experiment, we will also understand the Analog-to-Digital Conversion (ADC) and processing of the analog signals in a digital environment.

Pin/Port Selection :

I have made my connections as in the image below.
I connected the middle leg of the potentiometer to PE3.
I put a resistor between +3.3 v and pot leg for safety reasons.



What is the ADC Peripheral ?

First of all, let's answer this what is the duty of the ADC. ADC is the name of the structures that convert an analog signal to a digital signal. I can convert and process analog signals to digital with the ADC structure.

Tiva Launchpad has two 12-bit resolution ADCs. These are ADC0 and ADC1. ADC0 and ADC1 have the same structure. So when choosing the ADC we will use, we can choose any of them.

These two ADC modules provide the following features:

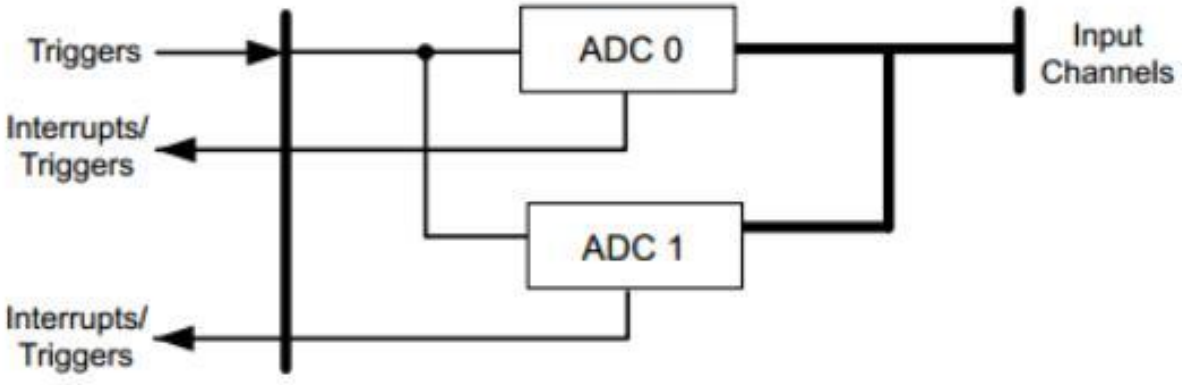
- 12 shared analog input channels
- 12-bit precision ADC
- Single-ended and differential-input configurations
- On-chip internal temperature sensor
- Maximum sample rate of one million samples/second (MSPS)
- Optional phase shift in sample time programmable
- Four programmable sample conversion sequencers from one to eight entries long, with corresponding conversion result FIFOs
- Five flexible trigger controls
  - Software Trigger Controller (default)
  - Timers
  - Analog Comparators
  - PWM
  - GPIO
- Hardware averaging of up to 64 samples
- 2 Analog Comparators

Voltage	2-Bit Digital Representation
0 to 2.5	00
2.5 to 5	01
5 to 7.5	10
7.5 to 10	11

What does 12 bit resolution mean ?
It means that analog values will be expressed as digital values at 2 ^ 12 levels. For example, if the resolution is 2 and the analog value changes between 0 and 10V ,

For ADC in Tiva, the digital range is from 0 to 2^12 -1 (0 to 4095). So it corresponds to 0V = 0 and 3.3V = 4095.

So, why 4095 (2 ^ 12 - 1) ?
It was said that both ADCs have 12 bit resolution. This means that 212 values can be created for 12 bits. It takes value up to 2^12 - 1, that is, 4095, within which we start from 0.



As can be seen from the diagram above, the input channels and triggers of both ADCs are the same. As we can see from the datasheet, which of the input pins are valid for both ADCs, it is also shown below .

Table 13-1. ADC Signals (64LQFP)

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
AIN0	6	PE3	I	Analog	Analog-to-digital converter input 0.
AIN1	7	PE2	I	Analog	Analog-to-digital converter input 1.
AIN2	8	PE1	I	Analog	Analog-to-digital converter input 2.
AIN3	9	PE0	I	Analog	Analog-to-digital converter input 3.
AIN4	64	PD3	I	Analog	Analog-to-digital converter input 4.
AIN5	63	PD2	I	Analog	Analog-to-digital converter input 5.
AIN6	62	PD1	I	Analog	Analog-to-digital converter input 6.
AIN7	61	PD0	I	Analog	Analog-to-digital converter input 7.
AIN8	60	PE5	I	Analog	Analog-to-digital converter input 8.

Table 13-1. ADC Signals (64LQFP) ***(continued)***

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
AIN9	59	PE4	I	Analog	Analog-to-digital converter input 9.
AIN10	58	PB4	I	Analog	Analog-to-digital converter input 10.
AIN11	57	PB5	I	Analog	Analog-to-digital converter input 11.

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

Another thing to know is Sequencers. I have to choose the sequencers (sequencers) depending on how many samples i want to take.

Sample Sequencers

The sampling control and data capture is handled by the sample sequencers. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO. Table 13-2 on datasheet page 802 shows the maximum number of samples that each sequencer can capture and its corresponding FIFO depth. Each sample that is captured is stored in the FIFO. In this implementation, each FIFO entry is a 32-bit word, with the lower 12 bits containing the conversion result.

Table 13-2. Samples and FIFO Depth of Sequencers

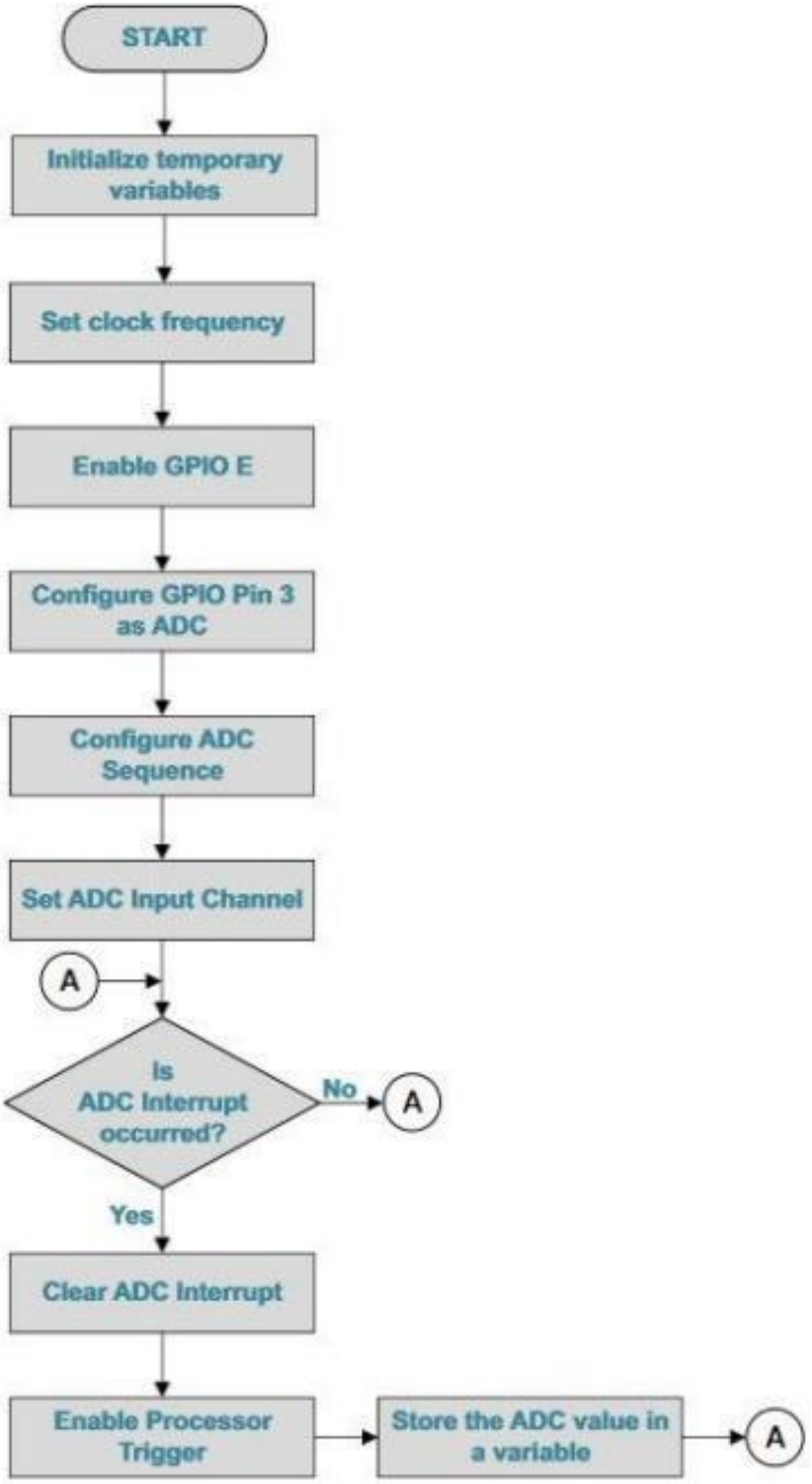
Sequencer	Number of Samples	Depth of FIFO
SS3	1	1
SS2	4	4
SS1	4	4
SS0	8	8

For example, SS3 will be used in this project. Number of samples should be 1 as there will be one value coming from the potentiometer.

These are the signals sent over the control lines in order for the trigger units to operate. In this application, the signals generated by the processor will be used. Trigger signals can also be obtained from external sources (hardware), such as pwm, timer, comparator.

It will be more understandable to explain how I read my ADC configuration settings and the voltage values between the legs of the potentiometer over the ADC peripheral.

Flowchart


START
Initialize temporary variables
Set clock frequency
Enable GPIO E
Configure GPIO Pin 3 as ADC
Configure ADC Sequence
Set ADC Input Channel
A
Is ADC Interrupt occurred?
No
A
Yes
Clear ADC Interrupt
Enable Processor Trigger
Store the ADC value in a variable
A

Explanations of The Code

Earlier in my laboratory reports, I explained in detail what the libraries I use here do, and how to initialize the GPIO.After defining the necessary libraries , I started to writed SysClock_80MHz() function for clock settings .

```
37  void SysClock_80MHz (void)
38  {
39     SYSCTL_RCC2_R |=    (0x80000800);
40     SYSCTL_RCC_R    = (SYSCTL_RCC_R & ~ (0X000007C0))+(0x15<<6);
41     SYSCTL_RCC2_R &=  ~(0x00000070);
42     SYSCTL_RCC2_R &=  ~(0x00002000);
43     SYSCTL_RCC2_R |=    (0x40000000);
44     SYSCTL_RCC2_R  = (SYSCTL_RCC2_R & ~(0x1FC00000))+(0x04<<22);
45     while ((SYSCTL_RIS_R & 0x00000040)==0){}
46     SYSCTL_RCC2_R &=  ~(0x00000800);
47  }
```

Here I did the following operations respectively :

1-)31th bit must be equal to 1 (for enabling) and 11th bit must be equal to 1 (for bypass)
2-)Clearing RCC[10:6] then setting XTAL=RCC[10:6]=15 (16 MHz External Osc.)
3-)Clearing RCC2[6:4] OSCSRRC2 =000=MOSC (Configuration for main oscillator source)
4-)Clearing RCC2[13]=PWRDN=0 (Clearing PWRDN to active PLL)
5-)Setting RCC2[30]=1 (selecting DIV400=400MHz)

6-)SYSCTL_RCC2_R = (SYSCTL_RCC2_R & ~(0x1FC00000))+(0x04<<22) : It means that giving N=4 , N+1=5 , 400/5=80 (Setting system clock divider RCC[28:22])
7-)Waiting for PLL to lock by polling PLLRIS
8-)RCC2[11]=0 (Setting BYPASS to 0, select PLL as the source of system clock)

In summary, after doing these steps, I have achieved 80 MHz with PLL.

Initializing ADC Peripheral

First, I enabled the clock peripheral for PORTRE.

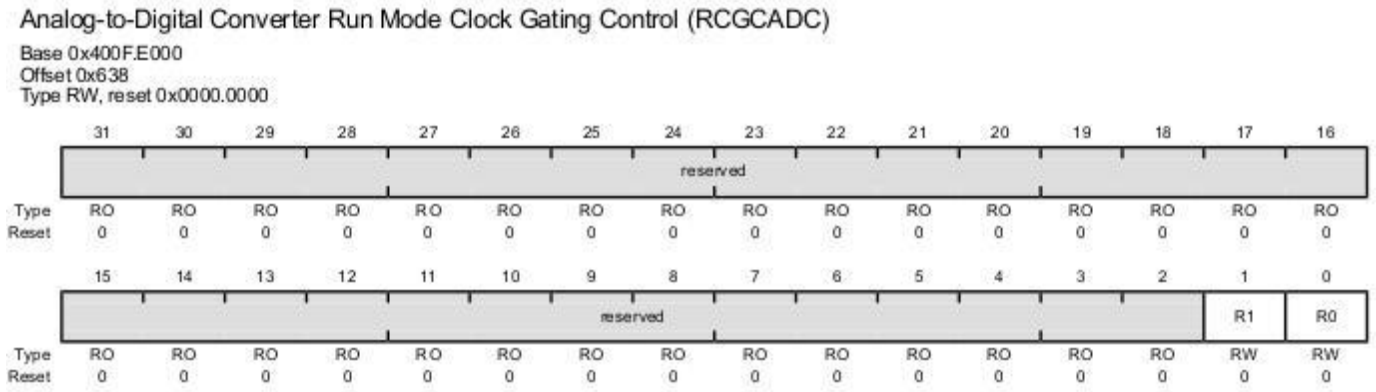
```
50  void init_Clock(void)
51  {
52    SYSCTL_RCGC2_R |= 0x10;                // to initialize clock to Port E
53    while(!( SYSCTL_PRGPIO_R & 0x10)){}    // waiting to enable clk
54  }
```

Then I made the ADC0_init(void) function definition.
Before activating the ADC peripheral, I made my GPIO configuration for analog reading by setting PE3 input and AFSEL, AMSEL registers.

```
56  // Definition : Initialize ADC0
57  void ADC0_init(void)
58  {
59    GPIO_PORTE_DIR_R    = 0x00;              // PE3 input
60    GPIO_PORTE_AFSEL_R  = 0x08;              // Enable alternate function for PE3
61    GPIO_PORTE_DEN_R    = 0x00;              // Disable "digital enable" for PE3 ,
62                                             // because I am gonna use periph. of adc .
63    GPIO_PORTE_AMSEL_R  = 0x08;              // Make PE3 analog
64
```

```
65    SYSCTL_RCGC0_R = 0x00010000;              // Activate ADC0 (16th bit corresponds to ADC0)
66
67    SYSCTL_RCGC0_R &= ~(0x00000300);          // Set sampling rate to 125K
68                                              // 9th and 8th bits correspond to ADC0 sampling rate
69                                              // 11th and 10th bits correspond to ADC1 sampling rate
70                                              // 00 means 125K
```

First thing I need to do is to enable the clock to the ADC0 or ADC1. Bit 0 and bit 1 of RCGCADC register are used to enable the clock to ADC0 and ADC1 modules, respectively. The RCGCADC is part of the System Control register and is located at base address of 0x400F.E000 with offset 0x638. That means, the RCGCADC is located at physical address of 0x400FE638 (0x400FE000 + 0x638 = 0x400FE638) in memory map. See the figures below.



Name	Description
R0	0: ADC Module 0 is disabled, 1: Enable and provide a clock to ADC module 0
R1	0: ADC Module 1 is disabled, 1: Enable and provide a clock to ADC module 1

By the way , If required by the application , reconfigure the ssample sequencer prioties in the ADCSSPRI register .

The default configuration has Sample Sequencer 0 with the highest priority and Sample Sequencer 3 as the lowest priority. So didn't use this register .

Configutation of Sample Sequencer :
I must be sure that the sample sequencer is disabled by clearing the corresponding ASEn bit in the ADCACTSS register .

```
82 |   ADC0_ACTSS_R  &= ~(1<<3);              // Disable sample sequencer 3 before changes (0)
```

Then , I configured the trigger event for th sample sequencer in the ADCEMUX register (I set software trigger for SS3).

```
87 |   ADC0_EMUX_R   &= (0xF<<12);
```

3-0 bits correspond to SS3
7-4 bits correspond to SS2
11-8 bits correspond to SS1
15-12 bits correspond to SS0
(from datasheet of Tiva C Series TM4C123G Launch Pad)

Note : SS3 trigger event = continously sample (continous means that no trigger is required , the ADC will start as soon as it is enabled .)

When using a PWM generator as the trigger source,I can use the ADC Trigger Source Select (ADCTSSEL) register to specify in which PWM module the generator is located. The default register reset selects PWM module 0 for all generators . So , I don't need to configure ADCTSSEL register as ADCSSPRI register.

For each sample in the sample sequence , I must configure the corresponding input source in the ADCSSMUXn register .
I choosed Ain0 (PE3) as input source (0) (page 801 on Datasheet)

```
94 |   ADC0_SSMUX3_R &= 0xFFFFFFF0;
```

Differential versus Single-Ended

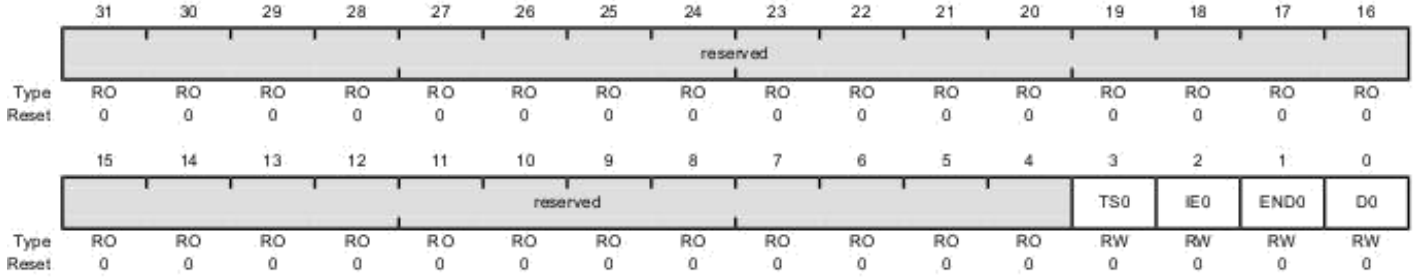
In some applications, our interest is in the differences between two analog signal voltages (the differential voltages). Rather than converting two channels and calculate the differences between them, the Tiva TM4C123G has the option of converting the differential voltages of two analog channels. The bit 0 of ADCSSCTL3 (ADC Sample sequence Control 3) register allows us to enable the differential option. Upon Reset, the default is the single-ended input. The pairing of the analog inputs for differential is hardwired.The table below shows the pairing of the ADC input channels for differential option.

Differential Pair	Analog Inputs
0	0 and 1
1	2 and 3
2	4 and 5
3	6 and 7
4	8 and 9
5	10 and 11

The other bits in ADCSSCTL3 register are bit 1 (END0), bit 2 (IE0), and bit 3 (TS) bits. On some of the ADC inputs, we have an internal temperature sensor embedded into the chip. Making bit 3 = 1, reads the temperature sensor value inside the chip itself. Since we are using one sample in the SSE3, we must enable the bit 1 (END0) to let ADC know that first sample is the only sample and there is no sample coming after that. Bit 2 (IE0) is the Interrupt enable. It causes the raw interrupt flag to set when this sample conversion is completed. However, in order to redirect the end-of-conversion to the NVIC interrupt controller, we must also enable the bit in ADCIM register, as we will see soon. It must be noted, even if we do not want to use interrupt, we must still set IE0 = 1, in order to post the raw interrupt flag for polling the conversion complete.

ADC Sample Sequence Control 3 (ADCSSCTL3)

ADC0 base: 0x4003.8000
ADC1 base: 0x4003.9000
Offset 0x0A4
Type RW, reset 0x0000.0000



3rd bit is for temperature sensor
2nd bit is for raw interrupt enable
1st bit is for end of sequence (needed to start sampling)
0th bit is for sampler differential input
(from datasheet of Tiva C Series TM4C123G Launch Pad)

```
106 |   ADC0_ACTSS_R |= 0x00000008;
```

If interrupt are to be used , I must set the corresponding MASK bit in the ADCIM register :

```
103 |   ADC0_IM_R = (1<<3);
```

A sample has completed conv. and the respective ADCSSCTL3 IEn bit is set , enabling raw interrupt.

```
106 |   ADC0_ACTSS_R |= 0x00000008;
```

Then , I activated SS3 and cleared interrupt status for SS3 to be able to start sampling again .

```
109     ADC0_ISC_R = (1<<3);
```

Short Informations About Bits of ADCISC register :
3rd bit corresponds to Interrupt(IN)3
2nd bit corresponds to IN2
1st bit corresponds to IN1
0th bit corresponds to IN0
(from datasheet of Tiva C Series TM4C123G Launch Pad)

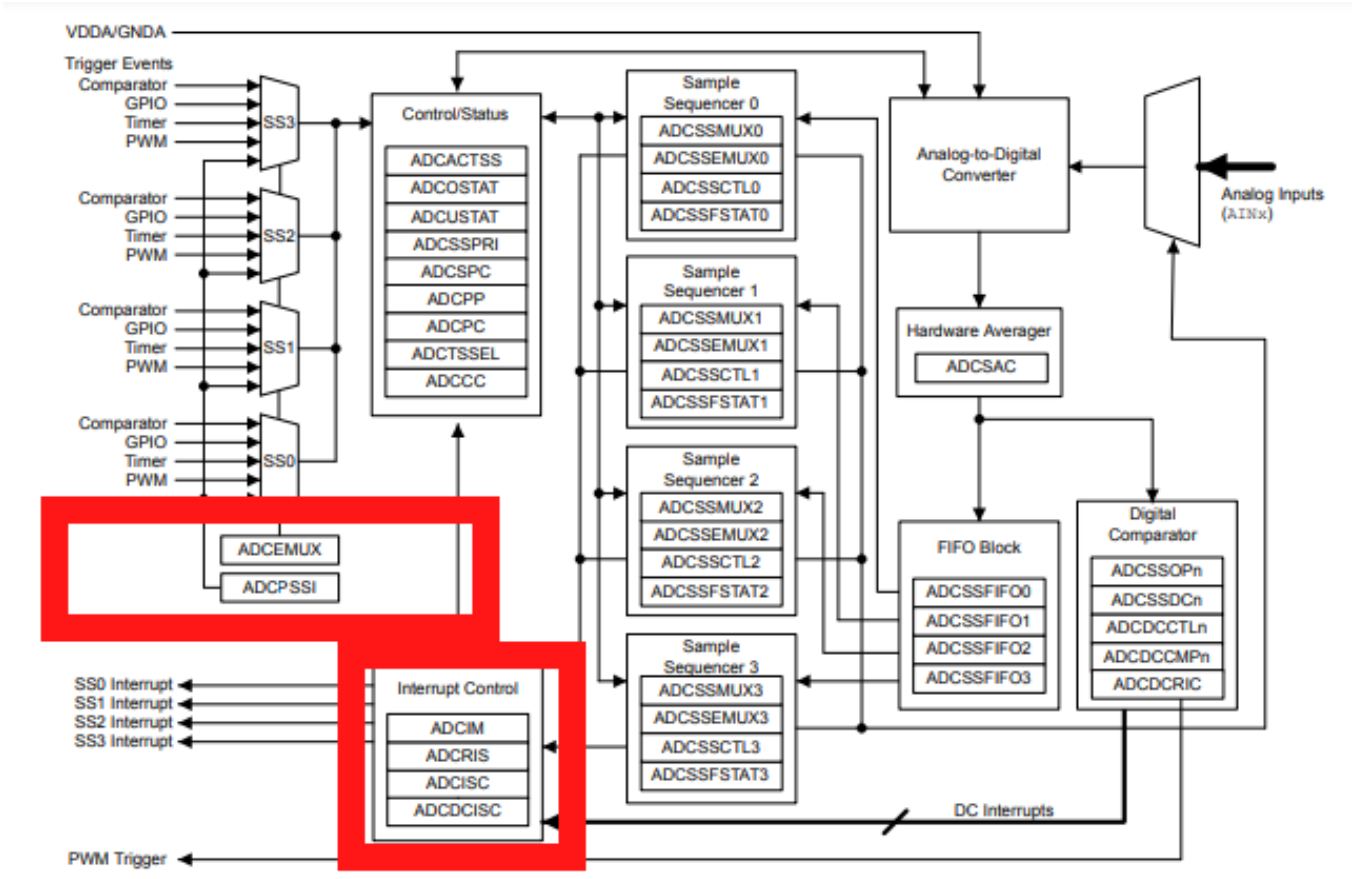
Here , ADC0_init(void) function finished.

Now , I gonna mention about reading analog voltage values .

voltageRead(void) Function

```
115  unsigned long voltageRead (void)
116  {
117      unsigned long ADC_result;
118      ADC0_PSSI_R = 0x0008;
119      while((ADC0_RIS_R&0x08)==0){}
120
121      ADC_result = ADC0_SSFIFO3_R&0xFFF;
122
123      ADC0_ISC_R = 0x0008;
124      return ADC_result;
125  }
```

To understand the code here, it would be useful to examine the diagram below.



Especially if we look at the registers I marked above, we will see which registers I will use for interrupt.

First I initialized SS3 by setting the PSSI registier.

By reading data from the RIS register, which is the interrupt register, I made it wait for the analog digital conversion process to finish.

Then I assigned the analog value assigned to FIFO to the ADC_result variable.

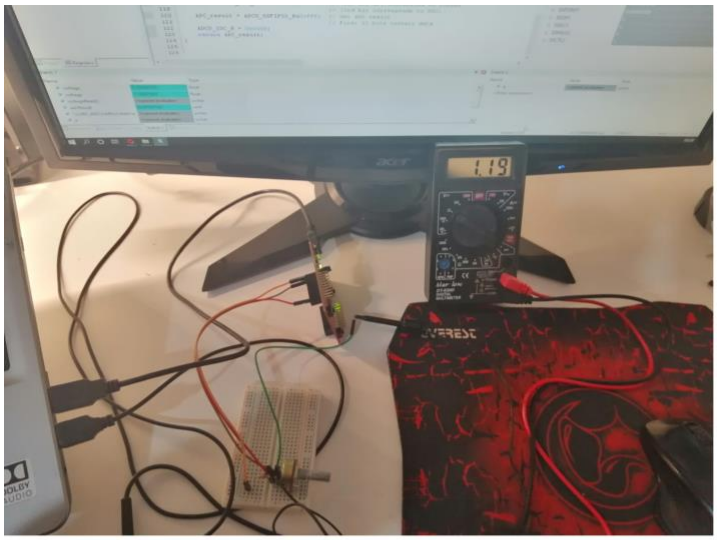
After that , I cleared interrupt status for SS3 to be able to start sampling again .

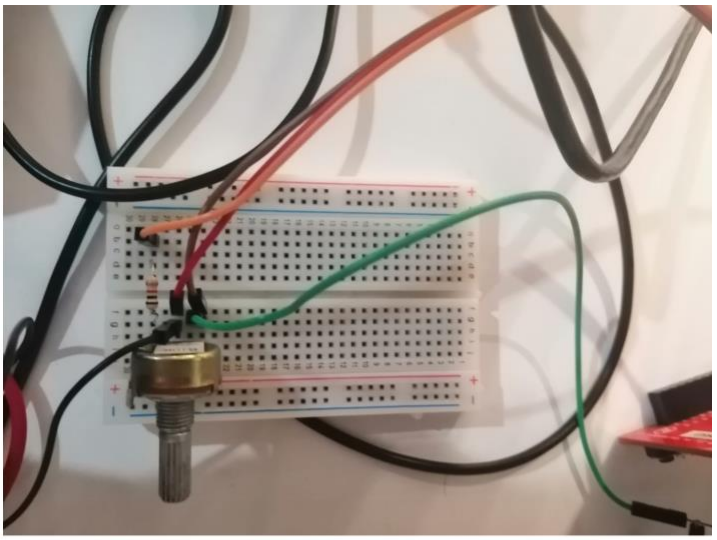
```
13  int main (void)
14  {
15    SysClock_80MHz();
16    init_Clock();
17    ADC0_init();
18
19    while(1)
20    {
21      adcResult=voltageRead();
22      voltage=adcResult*3.3/4095 ;  // Convert ADC value to voltage
23    }
24  return 0;
25  }
```

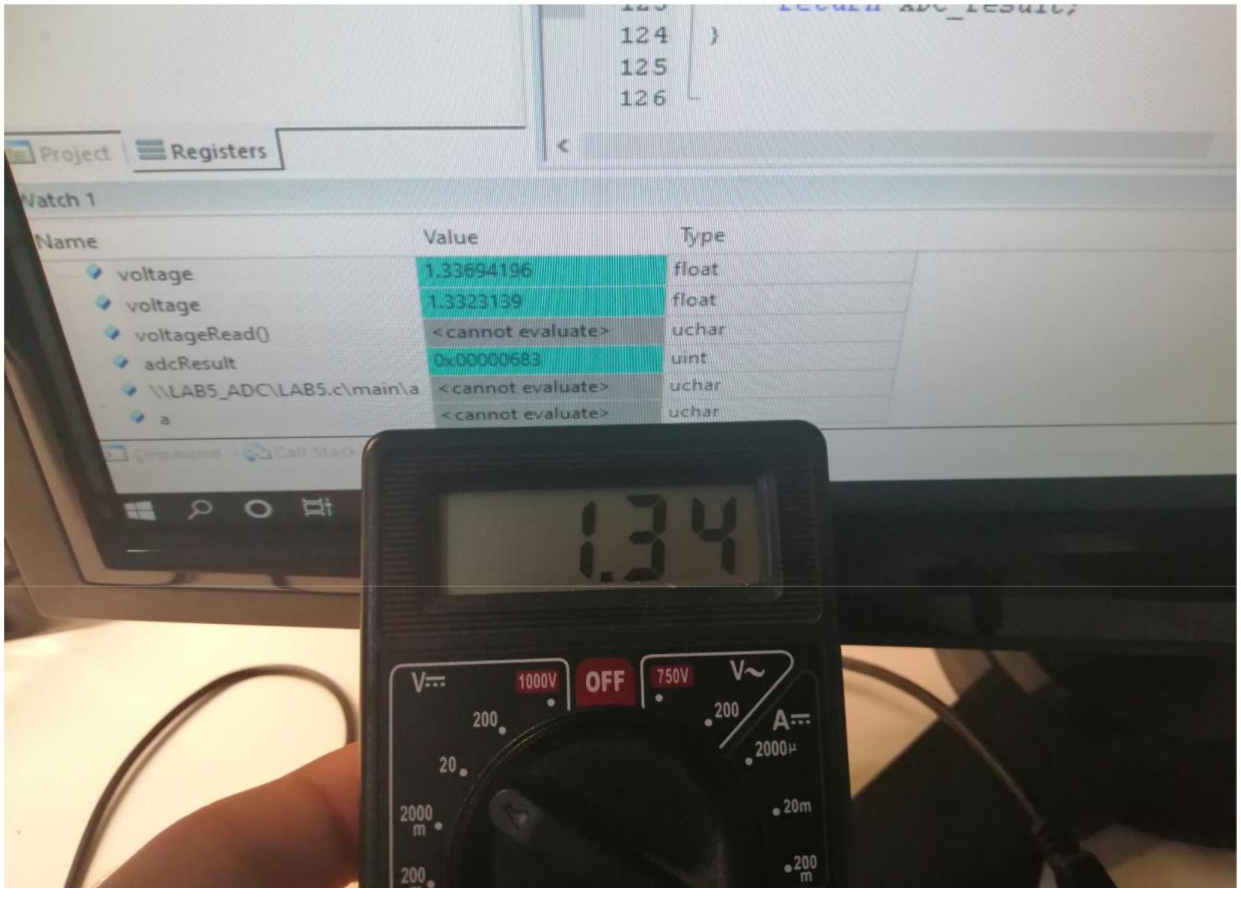
After I wrote all the initializing functions in their places, I wrote the main loop codes.I assigned the read values between 0 and 4095 to the adcResult variable.Next, I calculated the value of voltaged and assign it to the variable voltage. Since there is an increase value of 4095, the voltage value will increase by 3.3 / 4095 in each increase.

This is all I wanted to talk about the code I wrote. While writing the code, I often made use of the Tiva C Series TM4C123G datasheet. In addition, I added annotations to the code to make it easier to understand.

Photos About Working







All Codes

```c
#include <stdint.h>
#include <stdlib.h>
#include <tm4c123gh6pm.h>

void init_Clock(void);
void SysClock_80MHz (void);
void ADC0_init(void);
unsigned long voltageRead (void);

volatile static uint32_t adcResult = 0 ;
volatile static float    voltage   = 0 ;

int main (void)
{
  SysClock_80MHz();
  init_Clock();
  ADC0_init();

  while(1)
  {
    adcResult=voltageRead();
    voltage=adcResult*3.3/4095 ;  // Convert ADC value to voltage
  }
return 0;
}


// Definition : Initialize System Clock Function
// Exp. :
// Fout = N x Fref
// We must configure RCC and RCC2 register TO TAKE 80 MHz
void SysClock_80MHz (void)
{
  SYSCTL_RCC2_R |=   (0x80000800);
  // 31th bit must be equal to 1 (for enabling) and 11th bit must be equal to 1 (for bypass)

  SYSCTL_RCC_R   = (SYSCTL_RCC_R & ~ (0X000007C0))+(0x15<<6);
  // Clearing RCC[10:6] then setting XTAL=RCC[10:6]=15 (16 MHz External Osc.)

  SYSCTL_RCC2_R &=  ~(0x00000070);                            // Clearing RCC2[6:4] OSCSRRC2 =000=MOSC (Configuration for main oscillator source
  SYSCTL_RCC2_R &=  ~(0x00002000);                            // Clearing RCC2[13]=PERDN=0 (Clearing PWRDN to active PLL)
  SYSCTL_RCC2_R |=   (0x40000000);                            // Setting RCC2[30]=1 (selecting DIV400=400MHz)
  SYSCTL_RCC2_R  = (SYSCTL_RCC2_R & ~(0x1FC00000))+(0x04<<22); // Gives N=4 , N+1=5 , 400/5=80 (Setting system clock divider RCC[28:22])
  while ((SYSCTL_RIS_R & 0x00000040)==0){}                    // Wait for PLL to lock by polling PLLRIS
  SYSCTL_RCC2_R &=  ~(0x00000800);                            //RCC2[11]=0 (Setting BYPASS to 0, select PLL as the source of system clock)
}

// Definition : Initialize ADC0
void init_Clock(void)
{
  SYSCTL_RCGC2_R |= 0x10;               // to initialize clock to Port E
  while(!( SYSCTL_PRGPIO_R & 0x10)){}   // waiting to enable clk
}

// Definition : Initialize ADC0
void ADC0_init(void)
{
  GPIO_PORTE_DIR_R    = 0x00;               // PE3 input
  GPIO_PORTE_AFSEL_R  = 0x08;               // Enable alternate function for PE3
  GPIO_PORTE_DEN_R    = 0x00;               // Disable "digital enable" for PE3 ,
                                            // because I am gonna use periph. of adc .
  GPIO_PORTE_AMSEL_R  = 0x08;               // Make PE3 analog

  SYSCTL_RCGC0_R = 0x00010000;              // Activate ADC0 (16th bit corresponds to ADC0)

  SYSCTL_RCGC0_R &= ~(0x00000300);          // Set sampling rate to 125K
                                            // 9th and 8th bits correspond to ADC0 sampling rate
                                            // 11th and 10th bits correspond to ADC1 sampling rate
                                            // 00 means 125K
                                            // 01 means 250K
                                            // 10 means 500K
                                            // 11 means 1M

  // If required by the application , reconfigure the ssample sequencer prioties in the ADCSSPRI register .
  // The default configuration has Sample Sequencer 0 with the highest priority and Sample Sequencer 3 as the lowest priority.
  //ADC0_SSPRI_R  = 0x00000123;             // Set sequencer 3 to highest priority (00)
                                            // 13th and 12th bits correspond to SS3

  // Configutation of Sample Sequencer
  // I must be sure that the sample sequencer is disabled by clearing the corresponding ASEn bit in the ADCACTSS register .
  ADC0_ACTSS_R  &= ~(1<<3);                 // Disable sample sequencer 3 before changes (0)


  // I configured the trigger event for th sample sequencer in the ADCEMUX register .
  // SS3 trigger event = continously sample , continous means that no trigger is required , the ADC will start as soon as it is enabled .
  ADC0_EMUX_R   &= (0xF<<12);               // Set trigger for SS3


  // When using a PWM generator as the trigger source,I can use the ADC Trigger Source Select (ADCTSSEL) register to
  // specify in which PWM module the generator is located. The default register reset selects PWM module 0 for all generators .

  // For each sample in the sample sequence ,
  // I must configure the corresponding input source in the ADCSSMUXn register .
```

```
  ADC0_SSMUX3_R &= 0xFFFFFFF0;              // I choosed Ain0 (PE3) as input source (0) (page 801 on Datasheet)

  ADC0_SSCTL3_R  = 0x0006;                  // Choose control bits
                                            // 3rd bit temperature sensor (0)
                                            // 2nd bit raw interrupt enable (1)
                                            // 1st bit end of sequence (1) (needed to start sampling)
                                            // 0th bit sampler differential input (0)
  // If interrupt are to be used ,
  // I must set the corresponding MASK bit in the ADCIM register .
  ADC0_IM_R = (1<<3); // A sample has completed conv. and the respective ADCSSCTL3 IEn bit is set ,
                      // enabling raw interrupt.

  ADC0_ACTSS_R |= 0x00000008;               // Enable SS3

  ADC0_ISC_R = (1<<3);                      // Clear interrupt status for SS3
                                            // to be able to start sampling again
                                            // 3rd bit corresponds to Interrupt(IN)3
}

// Function convert from ADC value to readale voltage
unsigned long voltageRead (void)
{
    unsigned long ADC_result;
    ADC0_PSSI_R = 0x0008;               // Initiate SS3 (1)
    while((ADC0_RIS_R&0x08)==0){}       // Wait for A/D conversion to finish
                                        // (3rd bit corresponds to SS3)
    ADC_result = ADC0_SSFIFO3_R&0xFFF;  // Get ADC result
                                        // First 12 bits contain data
    ADC0_ISC_R = 0x0008;
    return ADC_result;
}
```

Thanks for reading ...

I added to my video link at page 2 of the my lab5 report and added uVision project file in the directory.

All Codes (Text Type) :

```
#include <stdint.h>
#include <stdlib.h>
#include <tm4c123gh6pm.h>

void init_Clock(void);
void SysClock_80MHz (void);
void ADC0_init(void);
unsigned long voltageRead (void);

volatile static uint32_t adcResult = 0 ;
volatile static float    voltage   = 0 ;

int main (void)
{
    SysClock_80MHz();
    init_Clock();
    ADC0_init();

    while(1)
    {
            adcResult=voltageRead();
            voltage=adcResult*3.3/4095 ;  // Convert ADC value to voltage
    }
return 0;
}


// Definition : Initialize System Clock Function
// Exp. :
// Fout = N x Fref
// We must configure RCC and RCC2 register TO TAKE 80 MHz
void SysClock_80MHz (void)
{
    SYSCTL_RCC2_R |=   (0x80000800);
    // 31th bit must be equal to 1 (for enabling) and 11th bit must be equal to 1 (for bypass)

    SYSCTL_RCC_R   = (SYSCTL_RCC_R & ~ (0X000007C0))+(0x15<<6);
    // Clearing RCC[10:6] then setting XTAL=RCC[10:6]=15 (16 MHz External Osc.)

    SYSCTL_RCC2_R &=  ~(0x00000070);  // Clearing RCC2[6:4] OSCSRRC2 =000=MOSC (Configuration for main oscillator source)
    SYSCTL_RCC2_R &=  ~(0x00002000);
     // Clearing RCC2[13]=PERDN=0 (Clearing PWRDN to active PLL)
    SYSCTL_RCC2_R |=   (0x40000000);
     // Setting RCC2[30]=1 (selecting DIV400=400MHz)
    SYSCTL_RCC2_R  = (SYSCTL_RCC2_R & ~(0x1FC00000))+(0x04<<22); // Gives N=4 , N+1=5 , 400/5=80 (Setting system clock divider RCC[28:22])
    while ((SYSCTL_RIS_R & 0x00000040)==0){}                    // Wait for PLL to lock by polling PLLRIS
    SYSCTL_RCC2_R &=  ~(0x00000800);                            //RCC2[11]=0 (Setting BYPASS to 0, select PLL as the source of system clock)
}

// Definition : Initialize ADC0
void init_Clock(void)
{
    SYSCTL_RCGC2_R |= 0x10;                                                 // to initialize clock to Port E
    while(!( SYSCTL_PRGPIO_R & 0x10)){}             // waiting to enable clk
}

// Definition : Initialize ADC0
void ADC0_init(void)
{
  GPIO_PORTE_DIR_R                  = 0x00;                                         // PE3 input
  GPIO_PORTE_AFSEL_R        = 0x08;                                         // Enable alternate function for PE3
  GPIO_PORTE_DEN_R                  = 0x00;                                         // Disable "digital enable" for PE3 ,
                                // because I am gonna use periph. of adc .
  GPIO_PORTE_AMSEL_R        = 0x08;                                                 // Make PE3 analog
```

```
    SYSCTL_RCGC0_R = 0x00010000;                                    // Activate ADC0 (16th bit corresponds to ADC0)

SYSCTL_RCGC0_R &= ~(0x00000300);                                    // Set sampling rate to 125K

                                                                    // 9th and 8th bits correspond to ADC0 sampling rate

                                                                    // 11th and 10th bits correspond to ADC1 sampling rate

                                                                    // 00 means 125K

                                                                    // 01 means 250K

                                                                    // 10 means 500K

                                                                    // 11 means 1M

// If required by the application , reconfigure the ssample sequencer prioties in the ADCSSPRI register .
    // The default configuration has Sample Sequencer 0 with the highest priority and Sample Sequencer 3 as the lowest priority.
    //ADC0_SSPRI_R  = 0x00000123;                                   // Set sequencer 3 to highest priority (00)

                                                                    // 13th and 12th bits correspond to SS3

    // Configutation of Sample Sequencer
    // I must be sure that the sample sequencer is disabled by clearing the corresponding ASEn bit in the ADCACTSS register .
ADC0_ACTSS_R  &= ~(1<<3);                                           // Disable sample sequencer 3 before changes (0)



    // I configured the trigger event for th sample sequencer in the ADCEMUX register .
// SS3 trigger event = continously sample , continous means that no trigger is required , the ADC will start as soon as it is enabled .
ADC0_EMUX_R   &= (0xF<<12);                                         // Set trigger for SS3



    // When using a PWM generator as the trigger source,I can use the ADC Trigger Source Select (ADCTSSEL) register to
    // specify in which PWM module the generator is located. The default register reset selects PWM module 0 for all generators .


    // For each sample in the sample sequence ,
    // I must configure the corresponding input source in the ADCSSMUXn register .
ADC0_SSMUX3_R &= 0xFFFFFFF0;                                        // I choosed Ain0 (PE3) as input source (0) (page 801 on Datasheet)

ADC0_SSCTL3_R  = 0x0006;                                            // Choose control bits

                                                                    // 3rd bit temperature sensor (0)

                                                                    // 2nd bit raw interrupt enable (1)

                                                                    // 1st bit end of sequence (1) (needed to start sampling)

                                                                    // 0th bit sampler differential input (0)
    // If interrupt are to be used ,
    // I must set the corresponding MASK bit in the ADCIM register .

    ADC0_IM_R = (1<<3); // A sample has completed conv. and the respective ADCSSCTL3 IEn bit is set ,
                                                                                        // enabling raw interrupt.


ADC0_ACTSS_R |= 0x00000008;                                         // Enable SS3

    ADC0_ISC_R = (1<<3);                // Clear interrupt status for SS3
```

```
                                        // to be able to start sampling again

                                                // 3rd bit corresponds to Interrupt(IN)3
}


// Function convert from ADC value to readale voltage
unsigned long voltageRead (void)
{
        unsigned long ADC_result;
        ADC0_PSSI_R = 0x0008;                   // Initiate SS3 (1)
        while((ADC0_RIS_R&0x08)==0){}           // Wait for A/D conversion to finish

                        // (3rd bit corresponds to SS3)

        ADC_result = ADC0_SSFIFO3_R&0xFFF;      // Get ADC result

                // First 12 bits contain data
        ADC0_ISC_R = 0x0008;
   return ADC_result;
}
```