

Lab 8

Microprocessor Systems Lab

Universal Asynchronous

Analog-to-Digital Converter (ADC) December 6th, 2024

California State University of Northridge

Edgar Gutierrez

Git Repository

<https://github.com/edgarg18/Lab-7>

**Introduction**

This lab introduces the Analog-to-Digital Converter (ADC) peripheral of the TM4C123GH6PM microcontroller, focusing on configuring ADC Module 0 to sample the potentiometer and the analog light sensor on the EduBase board. Students explored the principles of ADC operation, learning how to sample and convert analog signals into digital values for interpretation. The potentiometer allowed for manual experimentation with input levels, while the light sensor demonstrated environmental sensing. The lab also included implementing drivers for these components and displaying the measured voltage values on the LCD.

**Prelab Questions**

* 1. 1.
  2. How many ADC modules are within the TM4C123GH6PM microcontroller? How many sample sequencers does each ADC module have? Refer to the Analog-to-Digital Converter (ADC) section of the Tiva TM4C123GH6PM Microcontroller Datasheet.

The TM4C123GH6PM microcontroller contains two ADC modules (ADC0 and ADC1). Each ADC module is equipped with four sample sequencers (SS0, SS1, SS2, and SS3).

* 1. 2.
  2. How many bits of conversion resolution does each ADC module have? How many input channels are supported by the ADC modules?
  3. It ranges from 0 to 4095, the ADC modules support a total of 12 input channels.
  4. 3.

List the address offsets in hexadecimal format and the descriptions for the following ADC registers. Provide the register access type (read-write, write-only, or read-only). Refer to the Register Map section (Table 13-4, ADC Register Map) of the Tiva TM4C123GH6PM Microcontroller Datasheet.

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Address Offset | Access Type | Description |
| ADCACTSS | 0x000 | RW | ADC Active Sample Sequencer |
| ADCRIS | 0x004 | RO | ADC Raw Interrupt Status |
| ADCISC | 0x00C | RW1C | ADC Interrupt Status and Clear |
| ADCEMUX | 0x014 | RW | ADC Event Multiplexer Select |
| ADCTSSEL | 0x01C | RW | ADC Trigger Source Select |
| ADCPSSI | 0x028 | RW | ADC Processor Sample Sequence Initiate |
| ADCCTL | 0x038 | RW | ADC Control |
| ADCSSMUX0 | 0x040 | RW | ADC Sample Sequence Input Multiplexer Select 0 |
| ADCSSCTL0 | 0x044 | RW | ADC Sample Sequence Control 0 |
| ADCSSFIFO0 | 0x048 | RO | ADC Sample Sequence Result FIFO 0 |

4.

What is the ADCACTSS register used for? What is the purpose of the ASENn bits as described in the ADCACTSS register? Refer to pages 821-822 of the Tiva TM4C123GH6PM Microcontroller Datasheet.

This register controls the activation of the sample sequencers. Each sample sequencer can be enabled or disabled independently. The ASENn bits in the ADCACTSS register (where n ranges from 0 to 3) are used to enable or disable the corresponding sample sequencers:

Tools and Materials

Tiva C Series TM4C123G LaunchPad made by Texas Instruments

USB-A to Micro-USB Cable

EduBase Board

Small Flathead Screwdriver

Methods

After opening the ADC GitHub repository and downloading the files from the zip folder. We developed a driver to sample the potentiometer and the analog light sensor on the EduBase board. The ADC.c source file was updated to implement the ADC\_Init and ADC\_Sample functions as described in the Pre-Lab section. Initial implementation focused on sampling the potentiometer, with plans to add a second channel for the light sensor in the Tasks section. The code documentation was updated to include the required details, including author information, to ensure clarity and traceability.

The ADC\_Init and ADC\_Sample functions were implemented, and their corresponding code documentation was added in the ADC.h header file. The documentation included function descriptions and the @author section to credit the contributors.

To test the ADC driver, the main.c file was updated with the necessary header files: TM4C123GH6PM.h, SysTick\_Delay.h, EduBase\_LCD.h, and ADC.h. A static double array named adc\_buffer with eight elements was declared to store the measured voltage values from the potentiometer and analog light sensor. The @author section in the main.c documentation was also updated to include the contributors' names, ensuring proper attribution throughout the code.

The main program was updated to initialize the necessary components and display real-time potentiometer voltage readings on the LCD. The SysTick\_Delay\_Init, EduBase\_LCD\_Init, and ADC\_Init functions were called at the beginning of the main function to set up the SysTick timer, LCD, and ADC module, respectively.

In the while loop, the message “POTENTIOMETER” was displayed in the first row of the LCD, while the measured voltage value from the potentiometer was displayed in the second row. The voltage value was continuously updated every second by sampling the potentiometer using the ADC\_Sample function, storing the result in the adc\_buffer array, and displaying it on the LCD. This implementation ensured dynamic updates of the potentiometer's voltage in a clear and user-friendly manner.

After resetting the TM4C123G LaunchPad, the LCD displayed the potentiometer's measured voltage, with "POTENTIOMETER" on the first row and the voltage value on the second. Rotating the potentiometer with a flathead screwdriver caused the displayed voltage to change, confirming the ADC driver and LCD integration worked correctly.

For task 1 the ADC driver was updated to include the analog light sensor as a second input source, connected to the PE1 pin (Channel 2). The ADC was configured to sample the potentiometer first, followed by the light sensor, with the interrupt signal (INR0) triggered at the end of the second sample's conversion. The ADC\_Sample function was modified to read the Sample Sequence Result FIFO 0 register twice, storing the potentiometer's voltage in the first element and the light sensor's voltage in the second element of the analog\_value\_buffer array.

The main program was updated to display "Light Sensor" on the first row of the LCD and the measured voltage from the light sensor on the second row. Testing involved shining a flashlight on the light sensor to observe an increase in the displayed voltage value, verifying the correct functionality of the updated ADC driver and its integration with the LCD.

Task 2 was to implement the Analog\_Voltage\_to\_Digital functionto map an analog voltage value to a digital value between 0 and 255. The function takes the measured voltage (voltage), input range (in\_min = 0.0V, in\_max = 3.3V), and output range (out\_min = 0, out\_max = 255) as parameters. The function checks if the voltage is out of bounds:

* Returns 0 if the voltage is less than in\_min.
* Returns 255 if the voltage is greater than in\_max.
* Otherwise, maps the voltage linearly using the formula: digital\_value = (voltage - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min.

Main Program Updates:

The main program was updated to use the Analog\_Voltage\_to\_Digital function for mapping the light sensor's measured voltage to a digital value. This digital value was displayed in the second row of the LCD, starting at column 13. Additionally, if the digital value was 200 or greater, all LEDs on the EduBase board were turned on, indicating high light intensity detected by the sensor.

These updates allowed dynamic visualization of the light sensor's voltage as both an analog and digital value on the LCD, with LEDs providing a visual indicator for high light levels.

Results

In this lab, we successfully developed a driver for the ADC peripheral to sample the potentiometer and the analog light sensor on the EduBase board. The potentiometer and light sensor voltage values were measured and displayed dynamically on the LCD. The ADC driver was updated to handle multiple channels, with the potentiometer sampled first, followed by the light sensor, and the results stored in a buffer.

A function, Analog\_Voltage\_to\_Digital, was implemented to map the light sensor's voltage to a digital value between 0 and 255. The digital value was displayed on the LCD alongside the voltage reading. Additionally, if the digital value exceeded 200, all LEDs on the EduBase board were turned on, demonstrating an effective response to high light intensity. These results confirmed the successful integration of the ADC, LCD, and EduBase peripherals to process and visualize analog signals effectively.

I was not able to test the code for the second task due to not having the EduBase board, but I should have a light sensor at home and I have a screen. I will complete the task and update my GitHub

Discussion

This lab was one of the ones I was looking forward to the most and wish I would’ve had more time or attention span to spend working on it. I had some issues when it came to implementing the code for the first task. The reading from the light sensor was not changing as fast as I expected it to, and I ran out of time to trouble shoot the issue and verify if the sensor was operating as desired.

I will definitely be spending more time tinkering with the ADC peripheral in the Tiva board. One project that I have in mind is building a garden control system and implement several sensors such as temperature, moisture, and EC sensors. I would like to be able to collect data and also automate feeding. I am sure I will be needing to implement the last labs to complete the project.

Post Lab Questions

1.

In this lab, the potentiometer is sampled first, and the analog light sensor is sampled

after. Which ADC register was used to determine the order of the sampling

sequence? Specify the bits that were modified.

The ADCSSMUX0 register has fields (MUXn) that define the ADC input channel for each sample in the sequence.

ADC0->SSMUX0 |= 0x01;

ADC0->SSMUX0 |= 0x20;

2.

How were the analog values of the potentiometer and the light sensor calculated in

this lab? Briefly describe the sampling process as specified in the implementation of

the *ADC\_Sample* function in the ADC.csource file.

Analog Voltage (V)= ADC Digital Value×Reference Voltage (3.3V)​/ Resolution (4096)

Sampling begins by setting the appropriate bit in the PSSI register. The ADC converted analog signals to digital values, read sequentially from the SSFIFO0 register. The interrupt was then cleared via the ISC register, and the digital values were converted to voltages and stored, ensuring accurate sampling of both inputs.

3.

Which ADC register is used to indicate which channel is the end of the sampling

sequence? In this lab, which ADC channel was configured to be the last sample?

ADC0->SSCTL0.

This configuration ensured that the light sensor (Channel 2, PE1) was the final sample after the potentiometer (Channel 1, PE2).

4.

Briefly describe the changes that need to be made to the *ADC* driver in order to add

a third channel to the sampling sequence. Assume that there is another analog

sensor connected to the PE5 pin.

We would change the bit mask from 0x06 to 0x26, and update ADCSSMUX0. Configure the third sample (MUX2) in the sequence. Update the ADC\_Sample function to read three values from the SSFIFO0 register, storing the third result in the appropriate location (e.g., analog\_value\_buffer[2]). Add a conversion step for the new channel (PE5) to map its ADC value to a voltage.

5.

Which pins are available to use as analog input sources? List the information in the

table below. Refer to the Signal Description (Table 13-1, ADC Signals) section on

pages 801-802 of the Tiva TM4C123GH6PM Microcontroller Datasheet.

|  |  |  |
| --- | --- | --- |
| **Pin Name** | **Pin Mux / Pin Assignment** | **Description** |
| AIN0 | PE3 | Analog-to-digital converter input 0. |
| AIN1 | PE2 | Analog-to-digital converter input 1. |
| AIN2 | PE1 | Analog-to-digital converter input 2. |
| AIN3 | PE0 | Analog-to-digital converter input 3 |
| AIN4 | PD3 | Analog-to-digital converter input 4. |
| AIN5 | PD2 | Analog-to-digital converter input 5. |
| AIN6 | PD1 | Analog-to-digital converter input 6. |
| AIN7 | PD0 | Analog-to-digital converter input 7. |
| AIN8 | PE5 | Analog-to-digital converter input 8. |
| AIN9 | PE4 | Analog-to-digital converter input 9. |
| AIN10 | PB4 | Analog-to-digital converter input 10. |
| AIN11 | PB5 | Analog-to-digital converter input 11. |