BIRZEIT UNIVERSITY

Department of Electrical & Computer Engineering
ENCS4110 – Computer Design Laboratory

Experiment 08

Analog-to-Digital Converter (ADC)

Date: May 2026

8 Analog-to-Digital Converter (ADC)

Learning Objectives

After completing this experiment, you will be able to:

- Understand analog-to-digital conversion principles including sampling, quantization, and resolution.
- Recognize the architecture of the TM4C123 ADC modules including dual 12-bit converters and shared input channels.
- Configure ADC sample sequencers (SS0-SS3) with different depths and trigger sources.
- Configure GPIO pins for analog input using AFSEL, AMSEL, and DEN registers.
- Implement software-triggered and hardware-triggered ADC conversions.
- Read and interpret ADC conversion results from FIFO registers.
- Convert digital ADC values to physical quantities such as voltage and temperature.
- Implement both polling and interrupt-driven ADC operation modes.
- Utilize hardware averaging and multi-channel sampling for improved measurement accuracy.

Experiment Overview

This experiment introduces analog-to-digital conversion using the TM4C123's dual 12-bit ADC modules. You will configure sample sequencers for automatic multi-channel data acquisition, implement both polling and interrupt-driven sampling modes, and interface with analog sensors such as potentiometers and the internal temperature sensor. By the end of this lab, you will understand how to convert continuous analog signals to discrete digital values, configure trigger sources for periodic sampling, and implement sensor interfacing applications that bridge the analog and digital domains.

8.1 Theoretical Background

8.1.1 Introduction to Analog-to-Digital Conversion

Analog-to-Digital Converters (ADCs) are essential components in embedded systems, enabling microcontrollers to interface with the analog world by converting continuous analog signals (such as voltage from sensors) into discrete digital values for processing. They are widely used in applications like temperature sensing, light intensity measurement, and audio signal processing.

8.1.1.1 ADC Resolution and Sampling

ADCs are characterized by their **resolution**, which indicates the number of discrete levels they can represent, typically expressed in bits. For instance, a 12-bit ADC can represent $2^{12} = 4096$ discrete levels. The resolution determines the smallest detectable change in the analog input, known as the **least significant bit (LSB)**. The voltage corresponding to one LSB is calculated as:

$$\text{LSB Voltage} = \frac{V_{\text{REFP}} - V_{\text{REFN}}}{2^{\text{Resolution}}} \tag{8.1}$$

where V_{REFP} is the positive reference voltage, V_{REFN} is the negative reference voltage, and Resolution is the number of bits.

8.1.2 TM4C123 ADC Architecture

The TM4C123GH6PM microcontroller features **two identical 12-bit ADC modules** (ADC0 and ADC1) that share 12 analog input channels. These modules provide high-precision conversion of continuous analog voltages to discrete digital numbers.

8.1.2.1 Key Features

The ADC modules offer the following capabilities:

- **Resolution**: 12-bit precision (0 to 4095 discrete levels)
- **Shared Input Channels**: 12 analog input channels (AIN0-AIN11) shared between both modules
- **Internal Temperature Sensor**: On-chip temperature measurement capability
- **Maximum Sample Rate**: 1 million samples per second (1 MSPS)
- **Reference Voltage**: VDDA = 3.3V (separate analog power supply)
- **Input Voltage Range**: 0V to 3.3V for single-ended inputs
- **Sample Sequencers**: Four programmable sequencers per module (SS0-SS3) with depths of 8, 4, 4, and 1 samples
- **Hardware Averaging**: Automatic averaging of up to 64 samples for noise reduction
- **Digital Comparators**: Eight digital comparators per module for threshold detection
- **DMA Support**: Dedicated μDMA channels for efficient data transfer with burst requests
- **Isolated Power**: Separate analog power and ground pins for improved signal integrity

8.1.2.2 ADC Input Channels

The TM4C123 provides 12 analog input channels (AIN0 to AIN11) that can be connected to various GPIO pins, shared between both ADC modules. The mapping is as follows:

Channel	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5	AIN6	AIN7	AIN8	AIN9	AIN10	AIN11
Pin	PE3	PE2	PE1	PE0	PD3	PD2	PD1	PD0	PE5	PE4	PB4	PB5

Table 8.1: ADC Input Channel Pin Mapping

8.1.2.3 Sample Sequencers

Each ADC module has **four independent sample sequencers** that manage sampling control and data capture autonomously. They differ only in the number of samples and FIFO depth.

Sequencer	Number of Samples	FIFO Depth	Priority
SS0	8	8 entries	Highest
SS1	4	4 entries	High
SS2	4	4 entries	Low
SS3	1	1 entry	Lowest

Table 8.2: Sample Sequencer Capabilities and FIFO Depths

8.1.2.4 Trigger Sources

ADC conversions can be initiated by various trigger sources for flexibility:

- **Processor (Software)**: Manual trigger via ADCPSSI register
- **Analog Comparators**: Trigger on comparator output events
- **GPIO**: External signal on GPIO pins
- **General-Purpose Timers**: Periodic sampling synchronized with timer events
- **PWM Generators**: Synchronize sampling with PWM signals for motor control
- **Continuous**: Free-running continuous sampling mode

Both ADC modules can use independent or shared triggers. When shared, a programmable phase shifter allows delaying sampling by a phase angle for time-interleaved sampling to achieve higher effective rates.

8.1.3 Internal Temperature Sensor

The internal temperature sensor can be sampled by setting the TSn bit in the ADCSSCTLn register. Temperature (TEMP in °C) is calculated from the ADC reading (ADCCODE, 0 to 4095) as:

$$\text{TEMP} = 147.5 - \left(\frac{75 \times (V_{\text{REFP}} - V_{\text{REFN}}) \times \text{ADCCODE}}{4096} \right) \tag{8.2}$$

Note: This measures the microcontroller's die temperature, not ambient, and is useful for thermal management but may require calibration.

8.1.4 ADC Registers

The TM4C123 ADC modules are configured via registers. Refer to the datasheet for details. Key registers include:

RCGCADC — ADC Run Mode Clock Gating Control

Enables/disables clock for ADC modules. Bits correspond to modules (1 = enable, 0 = disable).

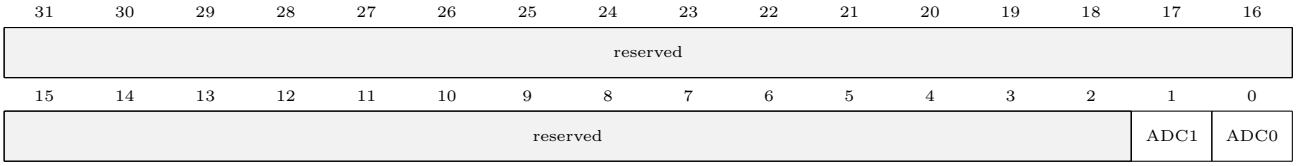

Figure 8.1: RCGCADC Register — ADC Run Mode Clock Gating Control

ADCACTSS — ADC Active Sample Sequencer

Enables/disables sample sequencers. Bits correspond to sequencers (1 = enable, 0 = disable). BUSY indicates ADC status (1 = busy, 0 = idle).

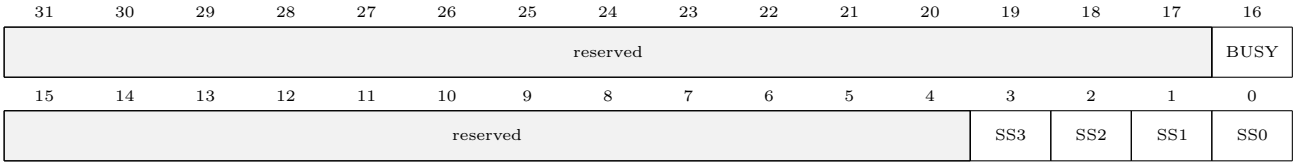

Figure 8.2: ADCACTSS Register — ADC Active Sample Sequencer

ADCSSMUXn — ADC Sample Sequence Input Multiplexer Select

Selects input channels for each sample in a sequencer. Each 4-bit field (MUX0-MUX7) specifies the channel (0-11 for AIN0-AIN11).

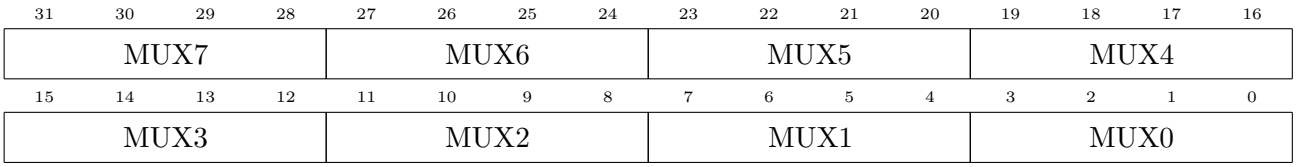

Figure 8.3: ADCSSMUXn Register — ADC Sample Sequence Input Multiplexer Select

ADCSSCTLn — ADC Sample Sequence Control

Configures each sample in a sequence. The END bit must be set for the final sample. This 32-bit register handles up to eight samples.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TS7	IE7	END7	D7	TS6	IE6	END6	D6	TS5	IE5	END5	D5	TS4	IE4	END4	D4
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS3	IE3	END3	D3	TS2	IE2	END2	D2	TS1	IE1	END1	D1	TS0	IE0	END0	D0

Figure 8.4: ADCSSCTLn Register — ADC Sample Sequence Control

Where:
- TSx: Temperature sensor select (1 = sample temperature sensor)
- IEx: Interrupt enable (1 = enable interrupt on sample completion)
- ENDx: End of sequence (1 = last sample in sequence)
- Dx: Differential select (1 = differential input, 0 = single-ended)
- x: Sample index (0-7)

ADCEMUX — ADC Event Multiplexer Select

Selects trigger source for each sample sequencer. Each 4-bit field (EM0-EM3) specifies the trigger source.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EM3				EM2				EM1				EM0			

Figure 8.5: ADCEMUX Register — ADC Event Multiplexer Select

The EMx field specifies the trigger source for sample sequencer x:

Value	Trigger Source
0x0	Processor (software trigger via ADCPSSI register)
0x1	Analog Comparator 0
0x2	Analog Comparator 1
0x3	Reserved
0x4	External GPIO (connected to GPIO interrupt)
0x5	Timer (requires TnOTE bit set in GPTMCTL register)
0x6	PWM Generator 0
0x7	PWM Generator 1
0x8	PWM Generator 2
0x9	PWM Generator 3
0xA–0xE	Reserved
0xF	Continuous (always sampling)

Table 8.3: ADC Event Multiplexer Trigger Sources

ADCPSSI — ADC Processor Sample Sequence Initiate

Initiates sampling for the specified sample sequencer when triggered by the processor.

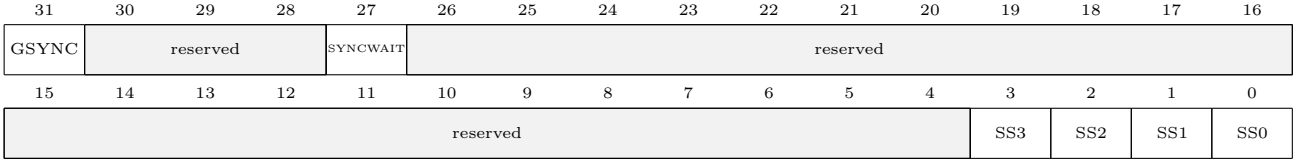

Figure 8.6: ADCPSSI Register — ADC Processor Sample Sequence Initiate

Where:

- **SSx**: Sample Sequencer x Initiate (write 1 to trigger conversion, self-clearing)
- **SYNCWAIT**: Synchronize Wait (1 = wait for synchronization signal before initiating)
- **GSYNC**: Global Synchronize (1 = generate synchronization signal for multiple ADC modules)

Note: The SSx bits are write-only and automatically clear after initiating the conversion. For synchronized sampling across modules, set GSYNC in one module to trigger all modules with SYNCWAIT enabled.

ADCSSFIFOn — ADC Sample Sequence FIFO

Holds the conversion results for the specified sample sequencer. Each read retrieves the next sample from the FIFO.

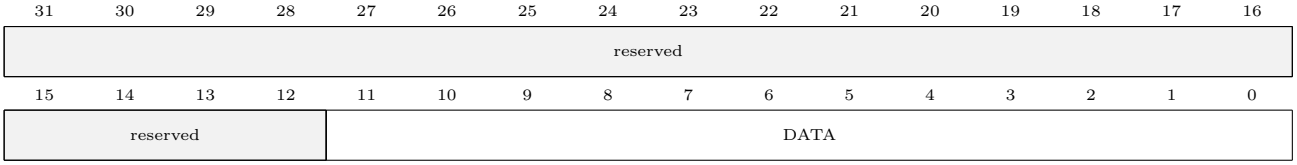

Figure 8.7: ADCSSFIFOn Register — ADC Sample Sequence FIFO

Where:

- DATA: 12-bit conversion result (0 to 4095)
- reserved: Unused bits

You can read from the ADCSSFIFOn register to retrieve conversion results. Each read operation removes the oldest sample from the FIFO. Ensure that the FIFO is not empty before reading to avoid invalid data. You can check the FIFO status using the ADCSSSTAT register (refer to the datasheet for details).

ADCRIS — ADC Raw Interrupt Status

Holds the raw interrupt status for each sample sequencer.

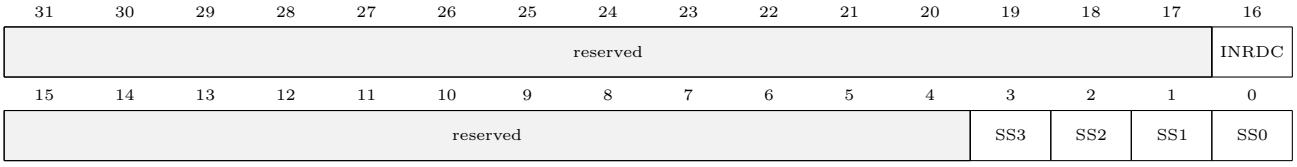

Figure 8.8: ADCRIS Register — ADC Raw Interrupt Status

Where:

- SSx: Sample Sequencer x Interrupt Status (1 = interrupt pending, 0 = no interrupt)
- INRDC: Digital Comparator Raw Interrupt Status
- reserved: Unused bits

ADCIM — ADC Interrupt Mask

Masks/unmasks interrupts for each sample sequencer.

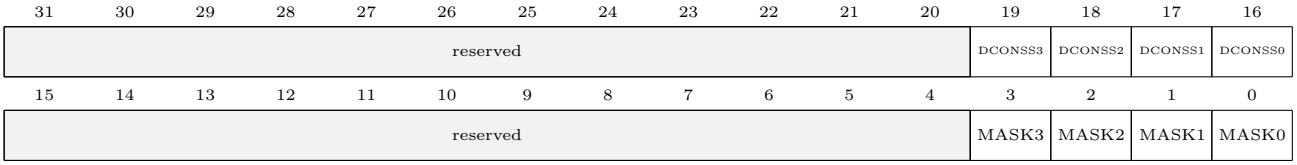

Figure 8.9: ADCIM Register — ADC Interrupt Mask

Where:

- MASKx: Sample Sequencer x Interrupt Mask (1 = unmask interrupt, 0 = mask interrupt)
- DCONSSx: Digital Comparator Interrupt Mask for Sample Sequencer x (1 = unmask comparator interrupt, 0 = mask comparator interrupt)
- x: Sample sequencer index (0-3)
- reserved: Unused bits

ADCISC — ADC Interrupt Status and Clear

Holds and clears the interrupt status for each sample sequencer.

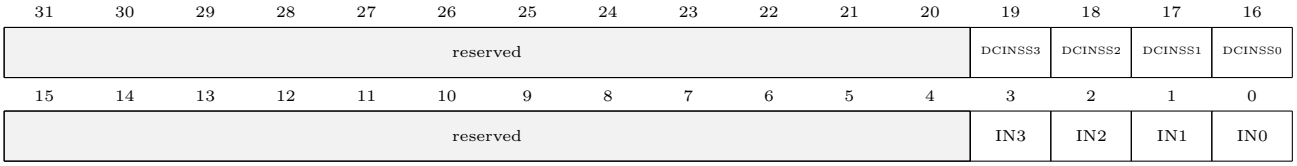

Figure 8.10: ADCISC Register — ADC Interrupt Status and Clear

Where:

- SSx: Sample Sequencer x Interrupt Clear (write 1 to clear interrupt)
- DCINSSx: Digital Comparator Interrupt Clear for Sample Sequencer x (write 1 to clear comparator interrupt)
- reserved: Unused bits

8.1.5 GPIO Registers

To configure GPIO pins for ADC input, you will need to set the appropriate GPIO registers. Below are the key registers involved in configuring GPIO pins for analog input:

GPIODEN — GPIO Digital Enable

Enables/disables digital functionality for GPIO pins. To use a pin as an analog input, its corresponding bit in this register must be cleared (0 = disable digital function).

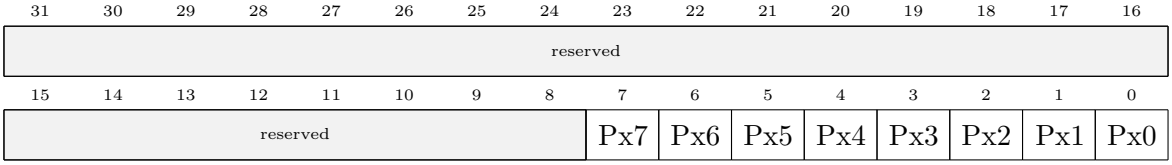

Figure 8.11: GPIODEN Register (Port x) — Digital Enable

Where:

- Pxn: Digital Enable for Pin n of Port x (1 = enable digital function, 0 = disable digital function)
- reserved: Unused bits
- x: Port letter (A-F)
- n: Pin number (0-7)

Note: To configure a pin for analog input, clear its corresponding bit in the GPIODEN register.

GPIOAFSEL — GPIO Alternate Function Select

Selects alternate functions for GPIO pins. To use a pin for ADC input, its corresponding bit in this register must be set (1 = alternate function enabled).

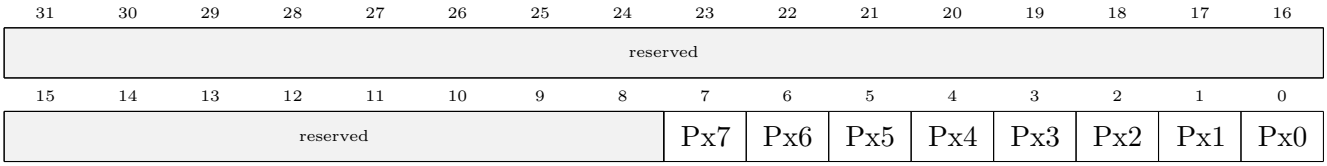

Figure 8.12: GPIOAFSEL Register (Port x) — Alternate Function Select

Where:

- Pxn: Alternate Function Select for Pin n of Port x (1 = enable alternate function, 0 = GPIO function)
- reserved: Unused bits
- x: Port letter (A-F)
- n: Pin number (0-7)

Note: To configure a pin for ADC input, set its corresponding bit in the GPIOAFSEL register.

GPIOAMSEL — GPIO Analog Mode Select

Enables/disables analog functionality for GPIO pins. To use a pin as an analog input, its corresponding bit in this register must be set (1 = enable analog function).

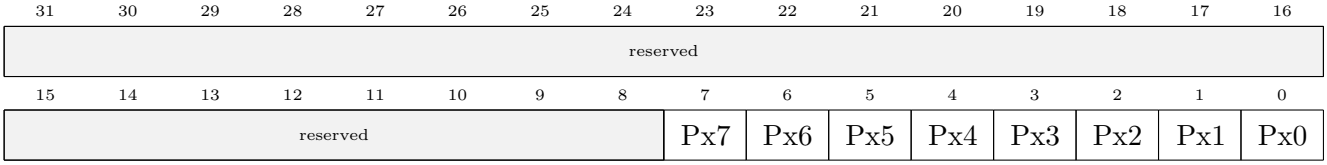

Figure 8.13: GPIOAMSEL Register (Port x) — Analog Mode Select

Where:

- Pxn: Analog Mode Select for Pin n of Port x (1 = enable analog function, 0 = disable analog function)
- reserved: Unused bits
- x: Port letter (A-F)
- n: Pin number (0-7)

Note: To configure a pin for ADC input, set its corresponding bit in the GPIOAMSEL register.

8.1.6 Configuration Steps

Configuring a GPIO pin for ADC input on the TM4C123 microcontroller involves three stages:

8.1.6.1 GPIO Pin Configuration

To prepare a GPIO pin for analog input:

1. Enable the clock for the corresponding GPIO port by setting the appropriate bit in the `RCGCGPIO` register.
2. Enable the pin's alternate function using the `GPIOAFSEL` register.
3. Disable its digital function by clearing the pin's bit in the `GPIODEN` register.
4. Enable the analog function by setting the pin's bit in the `GPIOAMSEL` register.
5. Set the pin direction as input by clearing its bit in the `GPIODIR` register.

8.1.6.2 ADC Sampling Sequencer Setup

After the pin is configured, the ADC module and sampling sequencer must be set up:

1. Enable the ADC clock by setting the correct bit in the `RCGCADC` register.
2. Disable the chosen sample sequencer by clearing its bit in the `ADCACTSS` register.
3. Select the input channel(s) by configuring the `ADCSSMUXn` register.
4. Set the control options for each sample (e.g., END, IE) in the `ADCSSCTLn` register, ensuring the END bit is set for the final sample.
5. Choose the trigger source by configuring the `ADCEMUX` register.
6. Re-enable the sequencer by setting its bit in `ADCACTSS`.
7. If processor-triggered sampling is used, start a conversion using the `ADCPSSI` register.

8.1.6.3 Interrupt Configuration

If interrupts are required:

1. Enable interrupts for the selected sequencer by setting its bit in the `ADCIM` register.
2. Enable the corresponding interrupt in the NVIC by setting the appropriate bit in the `NVIC_ISER` register.
3. In the ISR, read the conversion result from the `ADCSSFIFO` register.
4. Clear the interrupt flag by writing to the appropriate bit in the `ADCISC` register.

8.2 Procedure

This section demonstrates how to configure and use the ADC module on the TM4C123 microcontroller using both polling and interrupt-driven sampling methods. A potentiometer is used as a controllable analog input source by wiring its terminals to 3.3 V and ground, with the adjustable rotor connected to an ADC input such as AIN0 (PE3). Rotating the potentiometer varies the rotor voltage between 0 V and 3.3 V, allowing you to generate different analog levels and observe the corresponding ADC conversion results.

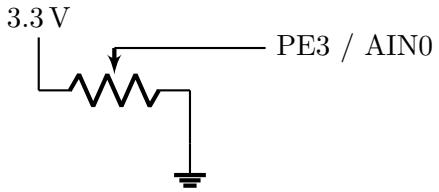

Figure 8.14: Potentiometer Voltage Divider for ADC Input

8.2.1 Example 1: Single ADC Channel Reading (Polling Method)

In this example, you will configure the ADC to read a single analog input channel (AIN0) using sample sequencer 3 (SS3) and turn on an LED if the voltage exceeds a certain threshold.

```
#include "TM4C123.h"
#define GREEN_LED 1 << 3
volatile unsigned int adc_value;
int main(void)
{
    SYSCTL->RCGCGPIO |= (1 << 4); // Enable Clock to GPIOE or PE3/AN0
    SYSCTL->RCGCADC |= (1 << 0);  // AD0 clock enable

    GPIOE->AFSEL |= (1 << 3);  // enable alternate function
    GPIOE->DEN &= ~(1 << 3);   // disable digital function
    GPIOE->AMSEL |= (1 << 3);  // enable analog function

    ADC0->ACTSS &= ~(1 << 3);            // disable SS3 during configuration
    ADC0->EMUX &= ~0xF000;                // Software Trigger
    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

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

    while (1)
    {
        ADC0->PSSI |= (1 << 3);            // initiate SS3 conversion
        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

        if (adc_value >= 2048)
            GPIOF->DATA |= GREEN_LED; // turn on green LED
        else if (adc_value < 2048)
            GPIOF->DATA &= ~GREEN_LED; // turn off green LED
    }
}
```

Listing 8.1: Single ADC Channel Reading Example

8.2.2 Example 2: Reading Internal Temperature Sensor

In this example, you will configure the ADC to read the internal temperature sensor using sample sequencer 3 (SS3) and display the temperature value on the debugger using the Watch window.

1. **Start a debug session:** After building the program successfully, start a debug session in Keil uVision IDE (Ctrl + F5).
2. **Add variables to the Watch window:** by selecting the variable name in the code, right-clicking, and choosing "Add to Watch" or manually entering the variable name in the Watch window.
3. **Run the program:** Use the "Run" button (F5) to execute the program. The Watch window will update with the current values of the monitored variables.

```c
#include "TM4C123.h"

volatile unsigned int adc_value;
volatile float temp;

void ADC0SS3_Handler(void)
{
    adc_value = ADC0->SSFIFO3 & 0xFFF; // Read 12-bit ADC value from FIFO
    temp = 147.5 - (75 * 3.3f * adc_value)/ 4096; // Convert to temperature in
    Celsius
    ADC0->ISC = 8; // Clear SS3 interrupt flag
}

int main(void)
{
    SYSCTL->RCGCGPIO |= (1 << 4); // Enable clock to Port E
    SYSCTL->RCGCADC |= (1 << 0);  // Enable clock to ADC0

    GPIOE->AFSEL |= (1 << 3);  // Enable alternate function on PE3
    GPIOE->DEN &= ~(1 << 3);   // Disable digital I/O on PE3
    GPIOE->AMSEL |= (1 << 3);  // Enable analog input on PE3

    ADC0->ACTSS &= ~(1 << 3);             // Disable SS3 during configuration
    ADC0->EMUX |= 0xF000;                 // Set SS3 to continuous sampling mode
    ADC0->SSMUX3 = 0;                     // Configure SS3 to sample AIN0 (PE3)
    ADC0->SSCTL3 |= (1 << 1) | (1 << 2) | (1 << 3);   // Set end of sequence, enable
    interrupt, enable temperature sensor

    ADC0->IM |= (1 << 3);        // Enable SS3 interrupt mask
    NVIC_EnableIRQ(ADC0SS3_IRQn); // Enable ADC0 SS3 interrupt in NVIC

    ADC0->ACTSS |= (1 << 3);  // Enable SS3

    while (1) {} // Wait for interrupts
}
```

Listing 8.2: Single ADC Channel Reading with Interrupts Example

8.2.3 Tasks

8.2.3.1 Task 1: Timer-Based Temperature Measurement

Modify Example 2 so that the ADC sampling is triggered by a timer every second, instead of using continuous sampling mode. Connect an analog temperature sensor (e.g., TMP36) to an ADC input channel (such as AIN0 on PE3). Configure the ADC to read the sensor voltage on each timer trigger and calculate the corresponding temperature. Monitor the temperature values in real time by adding the variable `temp` to the Watch window in the debugger.

8.2.3.2 Task 2: Comparing Two ADC Channels Using Dual Modules

Extend Example 2 to read two different analog inputs using separate ADC modules: use **ADC0 module** to sample AIN0 (PE3) and **ADC1 module** to sample AIN1 (PE2), both with sample sequencer 3 (SS3). Configure both ADC modules to use interrupts so that each conversion triggers its respective ISR. In the interrupt handlers, compare the two ADC values and use the result to control LEDs on GPIOF: for example, turn on the green LED if ADC0's value is higher, or turn on the red LED if ADC1's value is higher. This demonstrates simultaneous sampling using both ADC modules. Display both ADC values in the debugger Watch window to observe the comparison in real time.