Lab 5: Analog-to-Digital Converter (ADC)

Learning objectives

After completing this lab you will be able to:

- Calculate the voltage divider
- Understand the analog-to-digital conversion process
- Configure and use internal ADC unit

The purpose of the laboratory exercise is to understand analog-to-digital conversion and the use of an internal 8-channel 10-bit analog-to-digital converter.

Table of contents

Components list

- Arduino Uno board, USB cable
- LCD keypad shield

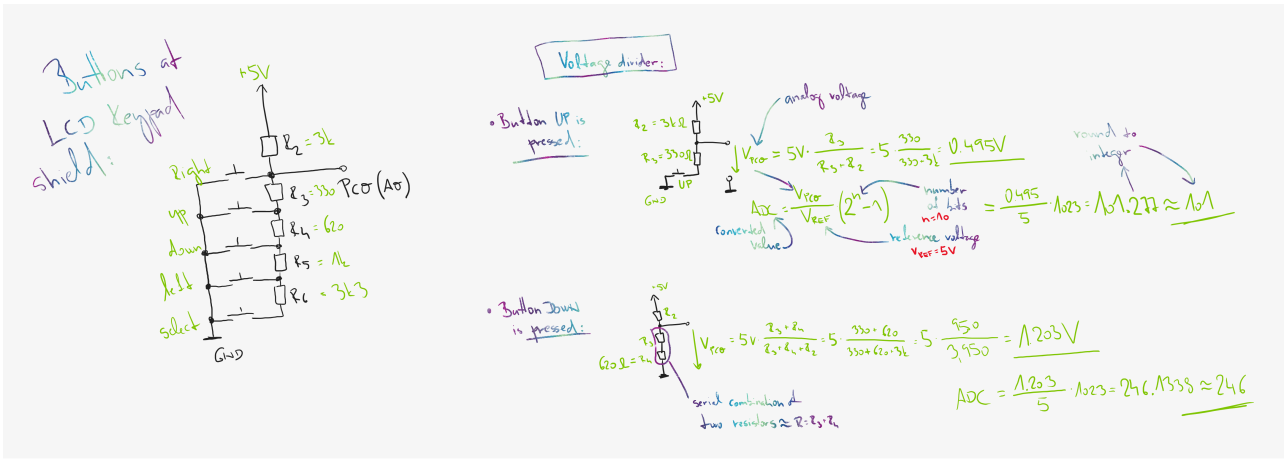
Pre-Lab preparation

1. Use schematic of the LCD keypad shield and find out the connection of five push buttons: Select, Left, Up, Down, and Right.

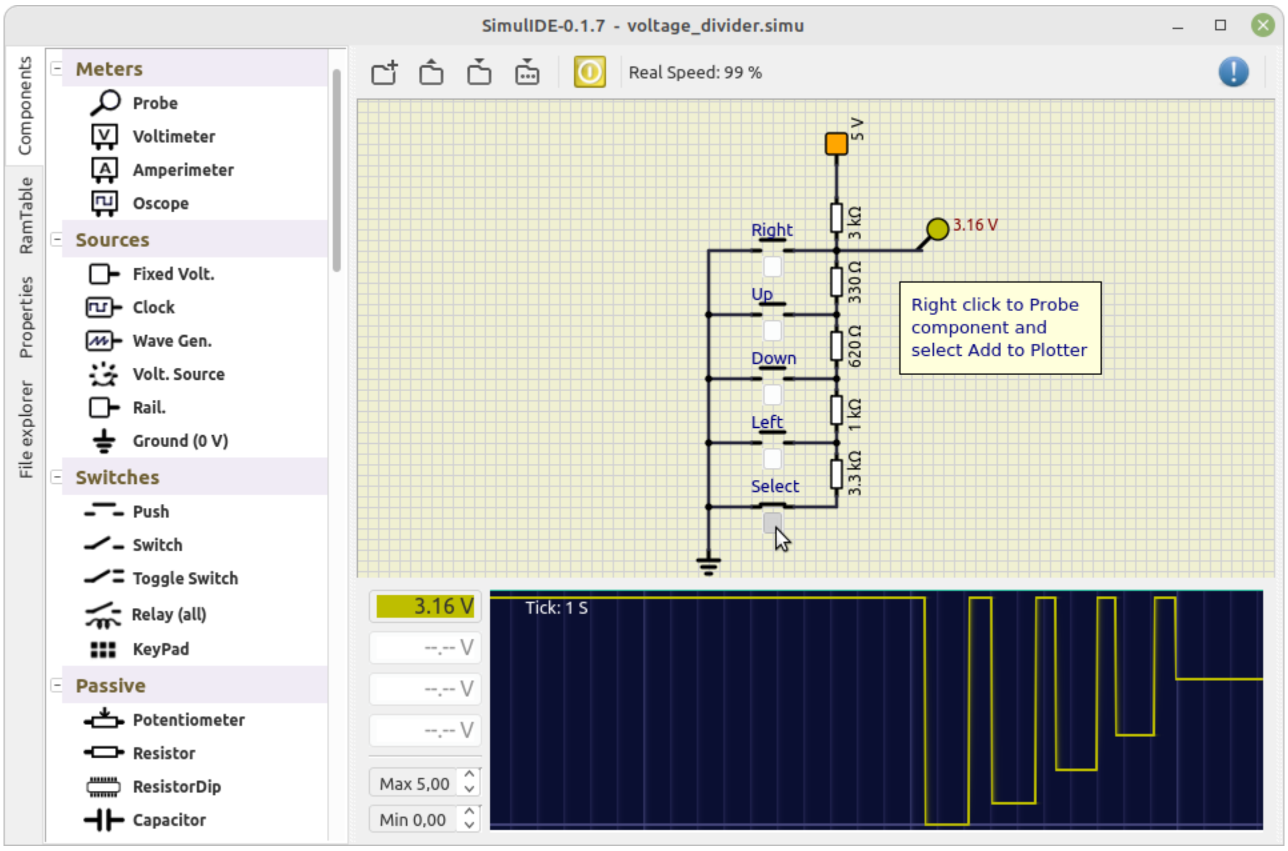
2. Remind yourself, what a voltage divider is.

Part 1: Voltage divider

1. According to the LCD keypad shield connection, calculate the voltage values on pin PC0 for each pressed buttons. In this case, the voltage on the pin is given by the voltage divider, where resistors R3, R4, R5 and R6 are applied successively. Write your values to the **PC0 voltage** column in the following table.


Buttons at LCD Keypad shield:
+5V
$R_2 = 3k$
$R_3 = 330$
PC0 (A0)
$R_4 = 620$
$R_5 = 1k$
$R_6 = 3k3$
Right
up
down
left
select
GND
Voltage divider:
• Button UP is pressed:
$R_2 = 3k\Omega$
$R_3 = 330\Omega$
UP
GND
analog voltage
$V_{PC0} = 5V \cdot \frac{R_3}{R_3 + R_2} = 5 \cdot \frac{330}{330 + 3k} = 0.495V$
$ADC = \frac{V_{PC0}}{V_{REF}}(2^n - 1) = \frac{0.495}{5} \cdot 1023 = 101.277 \approx 101$
converted value
number of bits
$n = 10$
reference voltage
$V_{REF} = 5V$
round to integer
• Button Down is pressed:
R_2
R_3
$620\Omega = R_4$
$V_{PC0} = 5V \cdot \frac{R_3 + R_4}{R_3 + R_4 + R_2} = 5 \cdot \frac{330 + 620}{330 + 620 + 3k} = 5 \cdot \frac{950}{3,950} = 1.203V$
serial combination of two resistors $\approx R = R_3 + R_4$
$ADC = \frac{1.203}{5} \cdot 1023 = 246.1338 \approx 246$

Push button	PC0 voltage	ADC value (calculated)	ADC value (measured)	ADC value (measured, hex)
Right	0 V	0	0	0
Up	0.495 V	101		
Down	1.203 V	246		
Left				
Select				
none				


SimulIDE-0.1.7 - voltage_divider.simu

Right click to Probe component and select Add to Plotter

Part 2: Synchronize repositories and create a new project

1. In your working directory, use **Source Control (Ctrl+Shift+G)** in Visual Studio Code or Git Bash (on Windows) or Terminal (on Linux) to update the local repository.

   > **Help:** Useful bash and git commands are `cd` - Change working directory. `mkdir` - Create directory. `ls` - List information about files in the current directory. `pwd` - Print the name of the current working directory. `git status` - Get state of working directory and staging area. `git pull` - Update local repository and working folder.

2. In Visual Studio Code create a new PlatformIO project `lab5-adc` for `Arduino Uno` board and change project location to your local repository folder `Documents/digital-electronics-2`.

3. IMPORTANT: Rename `LAB5-ADC > src > main.cpp` file to `main.c`, ie change the extension to `.c`.

Part 3: Analog-to-Digital Conversion

We live in an analog world, but surrounded by digital devices. Everything we see, feel or measure is analog in nature such as light, temperature, speed, pressure etc. It is obvious that we need something that could convert these analog parameters to digital value for a microcontroller or micro-processor to understand it.

An Analog to Digital Converter (ADC) is a circuit that converts a continuous voltage value (analog) to a binary value (digital). These ADC circuits can be found as an individual ADC ICs by themselves or embedded into a modern microcontroller.

The internal ADC module of ATmega328P can be used in relatively slow and not extremely accurate data acquisitions. But it is a good choice in most situations, like reading sensor data or push button signals.

AVR's ADC module has 10-bit resolution with +/-2LSB accuracy. It means it returns a 10-bit integer value, i.e. a range of 0 to 1023. It can convert data at up to 76.9 kSPS, which goes down when higher resolution is used. We mentioned that there are 8 ADC channels available on pins, but there are also three internal channels that can be selected with the multiplexer decoder. These are temperature sensor (channel 8), bandgap reference (1.1V) and GND (0V) [4].

1. Convert the voltages from the previous part according to the following equation. Note that reference is Vref=5V and number of bits for analog to digital conversion is n=10. Write the values to **ADC value (calculated)** column in the table from Part 2.1.

$$ADC = \left\lfloor \frac{V_{PC0}}{V_{REF}} \cdot (2^{nbit} - 1) \right\rfloor$$

2. The operation with the AD converter is performed through ADMUX, ADCSRA, ADCL+ADCH, ADCSRB, and DIDR0 registers. See ATmega328P datasheet (**Analog-to-Digital Converter > Register Description**) and complete the following table.

Operation	**Register(s)**	**Bit(s)**	**Description**
Voltage reference	ADMUX	REFS1:0	00: ..., 01: AVcc voltage reference (5V), ...
Input channel	ADMUX	MUX3:0	0000: ADC0, 0001: ADC1, ...
ADC enable	ADCSRA		

Operation	Register(s)	Bit(s)	Description
Start conversion			
ADC interrupt enable			
ADC clock prescaler		ADPS2:0	000: Division factor 2, 001: 2, 010: 4, ...
ADC 10-bit result			

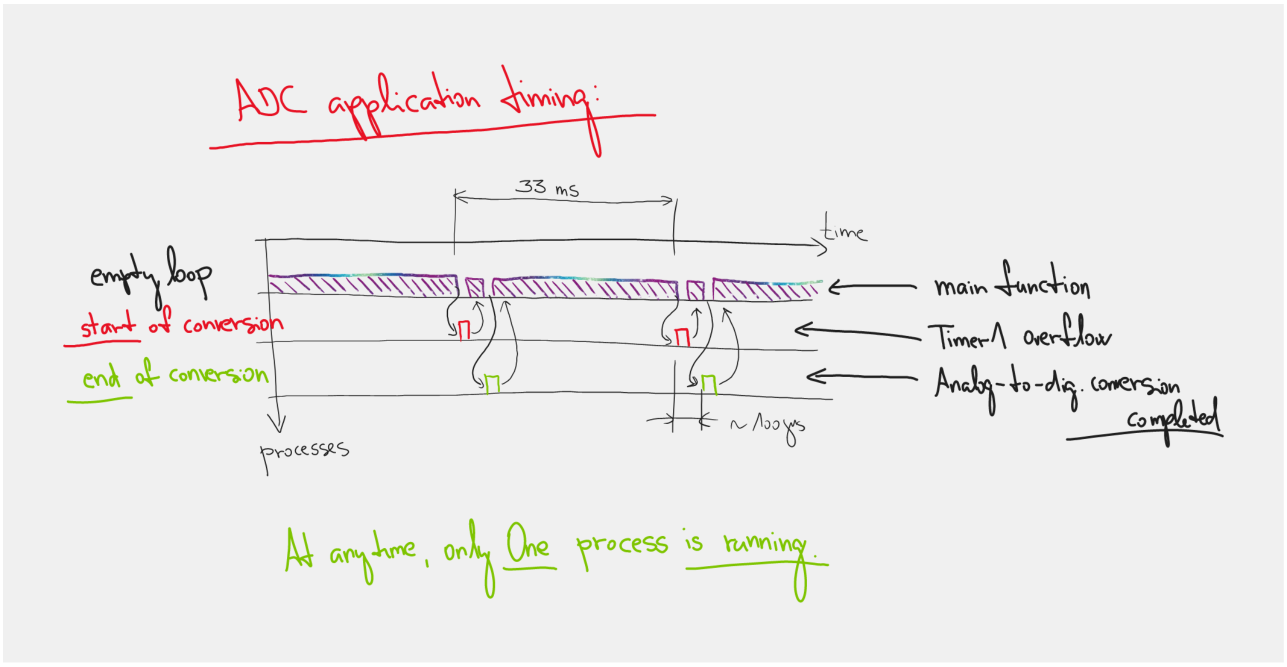
3. Copy/paste template code to `LAB5-ADC > src > main.c` source file.

4. Use your favorite file manager and copy `timer` and `lcd` libraries from the previous lab to the proper locations within the `LAB5-ADC` project. The final project structure should look like this:

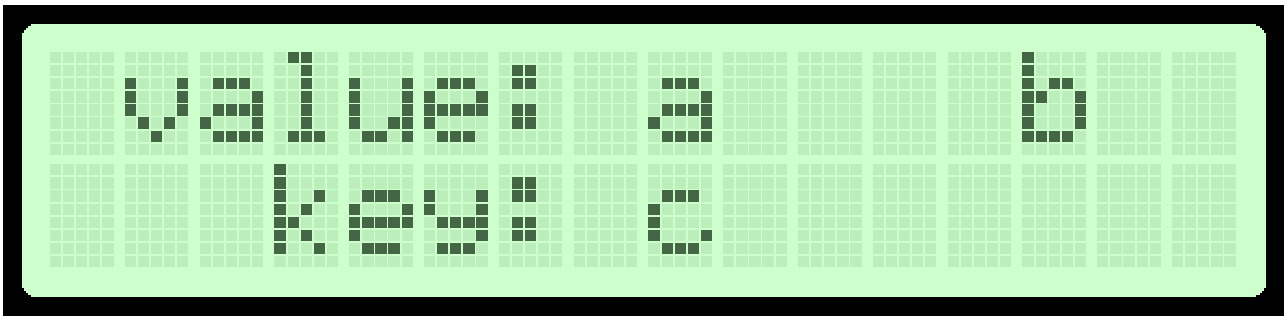
```
LAB5-ADC             // PlatfomIO project
├── include          // Included file(s)
│   └── timer.h
├── lib              // Libraries
│   └── lcd          // Peter Fleury's LCD library
│       ├── lcd.c
│       ├── lcd.h
│       └── lcd_definitions.h
├── src              // Source file(s)
│   └── main.c
├── test             // No need this
└── platformio.ini   // Project Configuration File
```

5. Go through the `main.c` file and make sure you understand each line. Use ATmega328P datasheet and complete the ADC configuration as follows:

   - voltage reference: AVcc with external capacitor at AREF pin
   - input channel: ADC0
   - enable ADC module
   - enable interrupt
   - clock prescaler: 128



Use single conversion mode and start the AD conversion every 33 ms by Timer/Counter1 overflow. When analog to digital conversion is finished, read the voltage level on push buttons and display it in decimal at LCD display position a. Display the same value in hexadecimal at position b. Note that you can use the 16-bit ADC variable (declared in the AVR library) to read the value from both converter registers ADCH:L.

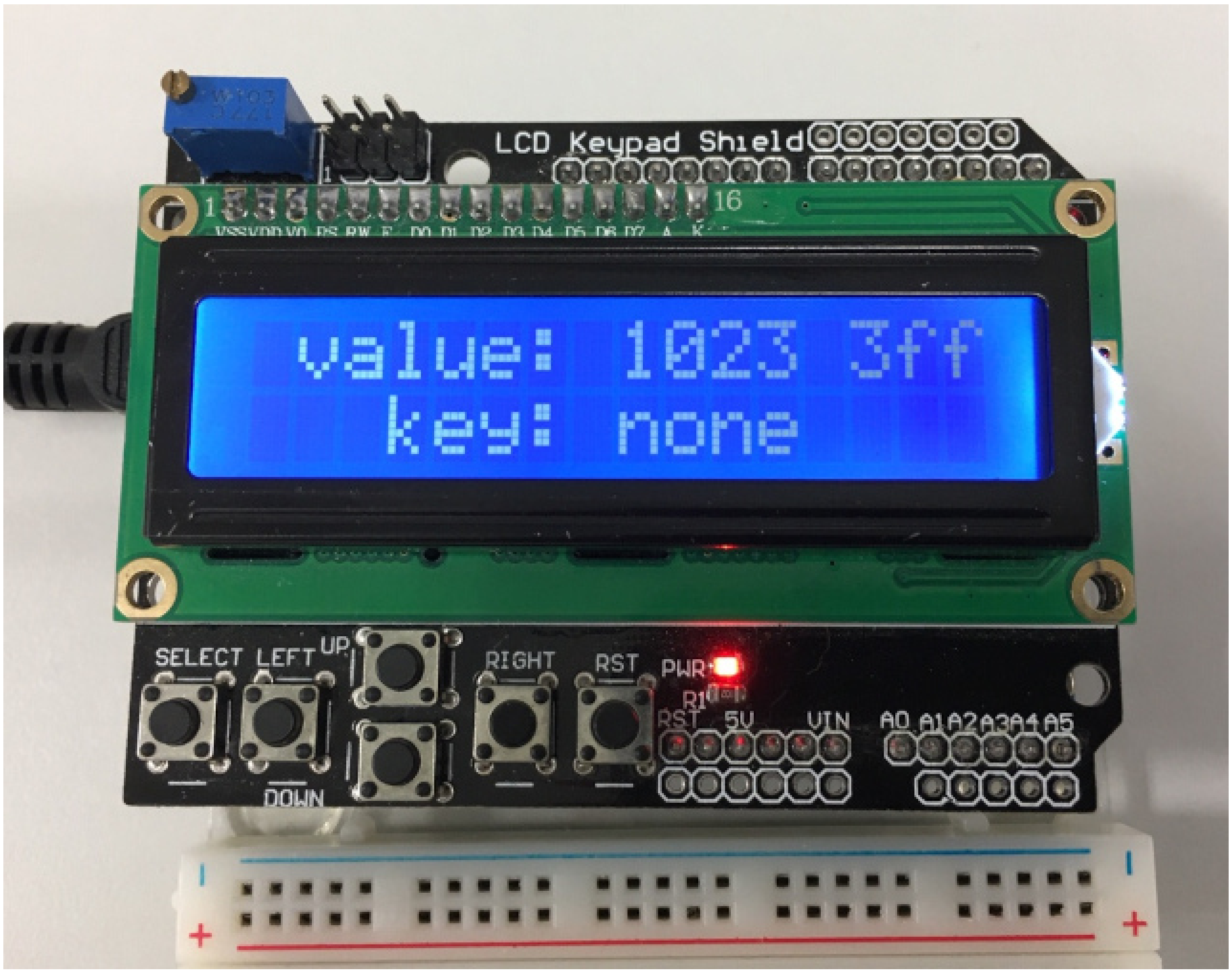


```
/**********************************************************************
*
 * Function: ADC complete interrupt
 * Purpose:  Display converted value on LCD screen.

**********************************************************************
/
ISR(ADC_vect)
{
    uint16_t value;
    char string[4];  // String for converted numbers by itoa()

    // Read converted value. Note that, register pair ADCH and ADCL
    // can be read as a 16-bit value ADC
    value = ADC;
    // Convert "value" to "string" and display it
}
```

Use standard C library and display converted value as string. Build the application and upload it to Arduino Uno board.



6. Test all push buttons and write converted values to the **ADC value (measured)** column in the table from Part 1.1.

7. Apply the "extending" method from past labs and start the ADC conversion not every 33 milliseconds but every 100 milliseconds.

8. Based on the converted values, distinguish which push button was pressed and display the information at LCD position `c`.

9. After completing your work, ensure that you synchronize the contents of your working folder with both the local and remote repository versions. This practice guarantees that none of your changes are lost. You can achieve this by using **Source Control (Ctrl+Shift+G)** in Visual Studio Code or by utilizing Git commands.

> **Help:** Useful git commands are `git status` - Get state of working directory and staging area. `git add` - Add new and modified files to the staging area. `git commit` - Record changes to the local repository. `git push` - Push changes to remote repository. `git pull` - Update local repository and working folder. Note that, a brief description of useful git commands can be found here and detailed description of all commands is here.

(Optional) Experiments on your own

1. In your application, try to recalculate the input voltage values in mV. *Hint: Use integer data types only; the absolute accuracy of the calculation is not important here.*

2. Create a library for Analog-to-Digital Converter. Create new files `adc.h` and `adc.c`, suggest function names, their parameters, and program their bodies.

3. Consider an application for temperature measurement. Use analog temperature sensor TC1046, LCD, and a LED. Every 30 seconds, the temperature is measured and the value is displayed on LCD screen. When the temperature is above the threshold, turn on the LED.

    1. Draw a schematic of temperature meter. The image can be drawn on a computer or by hand. Always name all components, their values and pin names!

    2. Draw two flowcharts of temperature meter: `TIMER1_OVF_vect` (which overflows every 1 sec) and `ADC_vect` interrupt handlers. The image can be drawn on a computer or by hand. Use clear description of individual algorithm steps.

4. Finish all (or several) experiments, upload them to your GitHub repository, and submit the project link via BUT e-learning. The deadline for submitting the assignment is the day prior to the next lab session, which is one week from now.

References

1. Tomas Fryza. Schematic of LCD Keypad shield

2. EETech Media, LLC. Voltage Divider Calculator

3. Components101. Introduction to Analog to Digital Converters (ADC Converters)

4. Embedds. ADC on Atmega328. Part 1

5. Microchip Technology Inc. ATmega328P datasheet

6. Tomas Fryza. Useful Git commands