Lab 1: Basic Analog and Digital I/O

Objective

Demonstrate basic digital input and output that works without intervention from the CPU. Additionally, demonstrate the basics of analog input.

Background

Please note that this lab uses terminology that was introduced in Lab 0.

B1: Digital Input Pin

Digital input pins are the base component for reading an external digital input. Digital devices such as pushbuttons are connected to these pins and can be connected to hardware components configured in the PSoC and/or read by the CPU program. In the Cypress component catalog, they can be found under Ports and Pins > Digital Input Pin. Here is what a default input pin looks like:

Both digital and analog input and output pins share the same schematic component; all of the components under Ports and Pins in the catalog are the same component with different initial configurations. Therefore, all of these share the same datasheet, and pins can be configured to change modes, such as making one an output instead of an input.

A pin may be configured as both input and output and both digital and analog if desired (for example: to read an input as both a digital and analog value). Labs will mostly use single mode pins.

There are many ways that an input pin can be configured. The most important options from the configuration window are presented here. For more detailed information, look at the datasheet.

- Name: component name, used to identify the pin in the DWR and API calls in the program
- Pins Tab
  - Number of Pins: sets the number of pins that this component represents. Useful when a component needs multiple pins
  - Pins list: each pin in the component may be individually configured using the Type, General, Input, and Output tabs on the right.
    - Double click a pin: allows a custom alias to be assigned (replaces the index number in the default alias; e.g. Pin_1_0 becomes Pin_1_Foo)
  - Type Tab
    - Digital Input: check this to indicate the pin(s) are a digital input

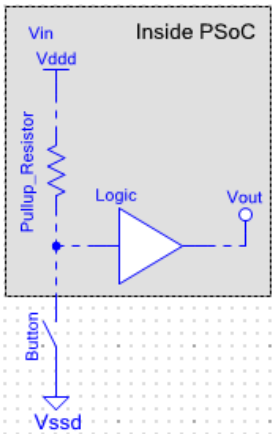
- HW Connection: If the pin(s) are supposed to be connected to other hardware on the schematic, check this box. If the pin(s) will only be read by the software, uncheck it.

▪ Show External Terminal: check this so off-chip components (hardware not part of the PSoC, such as LEDs) can be connected to the pin(s) for annotation purposes. This is optional, but it is useful to keep track of how external hardware connects to the PSoC.

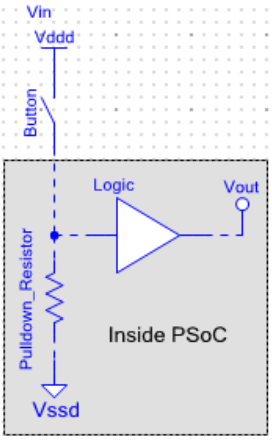
o General Tab

▪ Drive Mode: there are several drive modes, which define the electronic setup for how the pin(s) read or generate a digital signal. The most important ones are listed:

- Resistive Pull Up: used by the pushbuttons SW2 and SW3 on the development board. When the pushbutton is depressed, it connects to ground. See the figure below for a simplified circuit illustrating resistive pull up. While the button is not pressed, logic gate reads logic 1. When the button is pressed, the logic gate reads 0.



- Resistive Pull Down: similar to resistive pull up, except the input connects to power instead of ground. In this case, an open switch is read as logic 0 and a closed switch is read as logic 1.



- High Impedance Digital: the input is a high voltage for 1 or low voltage for 0. Unlike the resistive pull up/down options, the input cannot be disconnected entirely, otherwise the logic reading is unstable.

- Mapping Tab
  - Display as Bus: when this component is configured for multiple pins that have the same properties, this option allows them to be displayed as a single wire (bus) with multiple bits instead of multiple wires with one bit each.
  - Contiguous: when this component is configured for multiple pins, this option forces the pins to be sequentially mapped to the same port, rather than allowing connections to any set of pins. A contiguous configuration is more efficient to implement in both hardware and software, as a port can be read or written in a single call. Additionally, if continuous is unchecked, only aliases for each pin will be generated.

Here is the most important function in the programming interface. "Pin" is replaced with the name given to the component in the schematic view. Because the programming language is C, which is not object-oriented, each component has a uniquely named version of all of these functions.

uint8 Pin_Read()	Returns the current digital value of the pin(s) represented by this component with a single call. Only exists when pins are contiguous.

If this component is chosen to have non-contiguous pins, the function above is not generated. Instead, the alias of the pin needs to be used with the per-pin APIs. These APIs only work on one pin per call, so they are less efficient than having contiguous pins. The pin aliases are #defines that match the aliases assigned in the pins list of the component configuration. These can be found in the project at Generated_Source/PSoC5/Pin/Pin_aliases.h where both instances of Pin are replaced with the name of the component. The aliases file is generated for contiguous pins as well, so the following API works with all pin components. Note that these APIs are actually macros, but they can be treated like functions.

uint8 CyPins_ReadPin(uint16 pinPC)	Reads a single pin. Only pass in aliases defined by a component.

B2: Digital Output Pin

Digital output pins are the base component for sending a digital signal to an external component. Digital devices such as LEDs are connected to these pins. The pin can be driven by hardware configured in the schematic or by the software on the CPU, but not both. In the Cypress component catalog, they can be found under Ports and Pins > Digital Output Pin. Here is what the default output pin looks like:

Note that this is actually the same component as a digital input pin; it's just initially configured as an output pin. The most important options not covered from the input pin section above are presented here. For more detailed information, look at the datasheet.

- Pins Tab
  - Type Tab
    - Digital Output: check this to indicate the pin(s) are a digital output
      - HW Connection: If the pin(s) are supposed to be connected to other hardware on the schematic, check this box. If the pin(s) will be written to by the software, uncheck it.

- Output Enable: adds the ability for the hardware to enable or disable the output pin(s).

- General Tab
  - Drive Mode: there are several drive modes, which define the electronic setup for how the pin(s) read or generate a digital signal. Strong Drive should be used for output pins in most cases.

Here is the most important function in the programming interface. "Pin" is replaced with the name given to the component in the schematic view.

void Pin_Write(uint8 value)	Writes the value to the pin(s) represented by the component with a single call. The last bits (one per pin represented by the component) in the value are used for writing the pins.

Here are the most important per-pin APIs for output pins.

CyPins_SetPin(uint16 pinPC)	Sets a single pin to 1. Only pass in aliases defined by a component.
CyPins_ClearPin(uint16 pinPC)	Sets a single pin to 0. Only pass in aliases defined by a component.

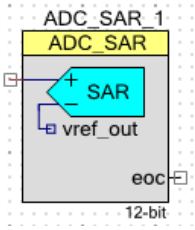
B3: Analog Input: ADC (Analog-to-Digital Converter)

Analog signals are another form of I/O that the PSoC can handle. Some analog applications include audio processing and reading analog sensors such as a potentiometer. Utilizing these signals often requires converting them to a digital representation. Analog-to-Digital Converters, or ADCs, are devices that convert an analog signal to a digital value.

Before introducing ADCs, here are some details about analog input pins. In the Cypress component catalog, they can be found under Ports and Pins > Analog Pin. Note that the wire coming out of the pin is red instead of green, which indicates that it is analog.

The default configuration of this pin does not need to be changed, except perhaps the Show External Terminal option if annotating the schematic is desired. The only analog drive mode is High Impedance Analog, and that is chosen by default. None of the programming interface functions previously introduced should be used with analog pins.

The PSoC includes two different types of ADCs, each with different implementations and capabilities: Delta Sigma and SAR. Because the SAR ADC has a simpler configuration, it will be the one used in this lab. The Delta Sigma ADC generally has more capabilities, such as a higher resolution, so it may be worth looking at later.



One attribute that all ADCs share is the resolution of the output. The primary goal of the ADC is to quantize an analog signal to a digital value. An analog signal can take on any voltage between its low value and high value. However, digital values have a set number of bits that are used to represent them, so the set of values is limited by how many bits are used. ADCs offer one or more resolutions for their digital output, where more bits means more resolution. The designs of the ADCs determine what these resolutions are.

Another attribute ADCs share is the sampling rate. Increased sampling allows for more converted analog values to be read per second. Combining higher resolution with higher sampling rates allows for the original analog signal to be represented digitally with high accuracy, which is important for applications like audio. However, with many ADCs, increasing the resolution will decrease the maximum sampling rate. So an appropriate trade-off needs to be made for the application of the ADC. For example, audio needs a high sampling rate and a decent resolution (i.e. 44.1 kHz sampling, 16-bit resolution), while a precision thermometer would require high resolution and would work fine with a low sampling rate (i.e. 100 Hz sampling, 32-bit resolution).

The SAR ADC is a differential type of ADC, comparing an input voltage to a reference voltage. The reference voltage could be a constant value or another input. In this lab, the constant reference voltage mode will be used. The valid input range of this ADC is from 0 volts to the reference voltage, which will be 3.3 volts in this lab. The ADC returns the difference between the input and the reference, which can then be converted to a voltage value using a library function.

Here are the most important options when configuring the SAR ADC:

- Modes
  - Resolution (bits): this ADC can have a resolution of 8, 10, or 12 bits. The lower the resolution, the higher the conversion rate can be.
  - Conversion rate (SPS): how many samples the ADC should take in a second.
- Sample Mode
  - Free running: continuously make conversions after the software requests it to start converting
  - Software/Hardware trigger: do not make conversions until the software or hardware requests a sample (will not be used in this lab)
- Input
  - Input range: selects how the voltage range is determined. Choose “Vssa to Vdda (Single Ended)” for now. Note that the actual analog signals must be between Vssa and Vdda in any case. Definitions for Vssa and Vdda are in the next section.

Here are the most important functions in the programming interface. “ADC” is replaced with the name given to the component in the schematic view.

void ADC_Start()	Powers on the ADC and resets state (use when initializing)
void ADC_Stop()	Powers off the ADC
void ADC_StartConvert()	Initiates conversion on the ADC. If sample mode is Free running, the ADC starts continuously converting. If Software trigger, only one conversion will occur. Not available with Hardware trigger.
void ADC_StopConvert()	Stops conversion on the ADC, halting Free running mode. Not available with Hardware trigger.

uint8 ADC_IsEndConversion (uint8 retMode)	Returns a non-zero value when the last conversion is completed. Pass ADC_RETURN_STATUS for retMode to get an immediate result, or pass ADC_WAIT_FOR_RESULT for this method to block until the last conversion is completed.
int16 ADC_GetResult16()	Returns the result of a conversion in "counts".
float ADC_CountsTo_Volts (int16 adcCounts)	Converts the value from ADC_GetResult16() to a floating point voltage value.
int16 ADC_CountsTo_mVolts (int16 adcCounts)	Converts the value from ADC_GetResult16() to a 16-bit integer millivolt value.
int32 ADC_CountsTo_uVolts (int16 adcCounts)	Converts the value from ADC_GetResult16() to a 32-bit integer microvolt value.

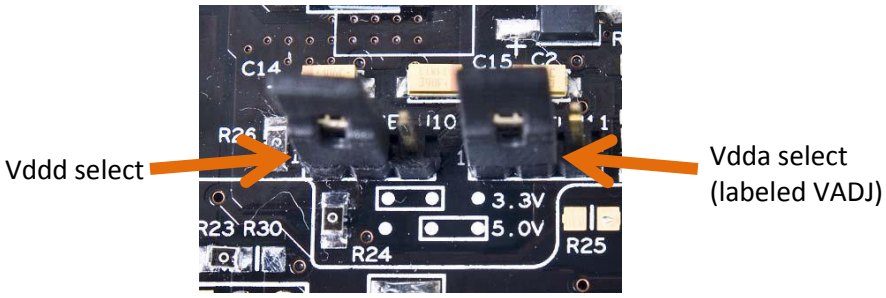
B4: PSoC Development Kit Power System

Because both analog and digital components are used in the PSoC, some special design considerations need to be made to reduce noise in the analog components. One of these special considerations necessitates the separation of power and ground lines between digital and analog components. Here are the names given to the different lines:

Vddd	Digital power
Vssd	Digital ground
Vdda	Analog power
Vssa	Analog ground

When adding electronics to the development kit in the future, be sure to make sure the digital components use the digital power and ground and the analog components us the analog power and ground. Future labs will point this out.

Vddd and Vdda can be set to either 3.3 volts or 5.0 volts. However, when running the development board off of USB power, these two must be set to 3.3 volts because the 5.0 volt source is unstable. If 5.0 volts becomes necessary, a 9V battery or 12V/1A positive center power adapter will need to be used. There are tall jumpers at the top of the development kit below the 9V battery plug that allow the configuration to be changed.



Disconnect all power sources before changing these jumpers, and only change them if necessary. If 5.0 volts is necessary, do NOT use the USB connection to power the PSoC, as it does not have enough power to generate a stable 5.0 volts. Setting Vdda to 3.3 volts and Vddd to 5.0 volts is NOT SUPPORTED. The other three configurations are valid.

PSoC Creator automatically assumes that Vddd and Vdda are 5.0 volts, rather than 3.3 volts. This lab will include instructions for how to change this assumption, since that changes how ADC_CountsTo_Volts and similar functions do their conversions.

B5: Using Hardware over Software

This lab will use the hardware to control the LED instead of software, which was used in the last lab. There are many cases where using hardware is better than using software. In this case, having the software continuously poll the pushbutton pin takes up a lot of CPU time, where in a real application CPU time would be better spent on more complicated tasks. Letting the hardware handle lighting up the LED removes this concern from the CPU. A future lab will introduce another method, called interrupts, of reducing CPU load when the CPU needs to interact with the hardware.

Procedure

Outcome

In this lab, the PSoC will be programmed to turn on an LED (LED4) while a pushbutton (SW2) is pressed without software intervention. Additionally, a SAR ADC will be used to read the voltage off of a potentiometer (variable resistor; rotating it adjusts the resistance, which changes the output voltage). The software performs basic filtering of the readings from the ADC and prints the resulting voltage to the LCD panel.

This lab does not describe in detail how to perform tasks that were introduced in Lab 0. Review the Lab 0 procedure if something was forgotten.

Part 1: Create a New Project

1. Open the workspace from the previous lab.
2. Create a new empty PSoC 5LP project just like in the previous lab, named Lab1. Under Advanced, make sure that “Add to Current Workspace” is selected next to Workspace. See Figure 1 below.

It may be desirable to collapse the Lab0 project in the Workspace Explorer and close any files open from the last lab, since the new lab uses the same filenames in the project. Also, creating a new project sets it as the active project, where commands like “Program” operate on the active project. To mark a project as active, right click it in the Workspace Explorer and select Set As Active Project.

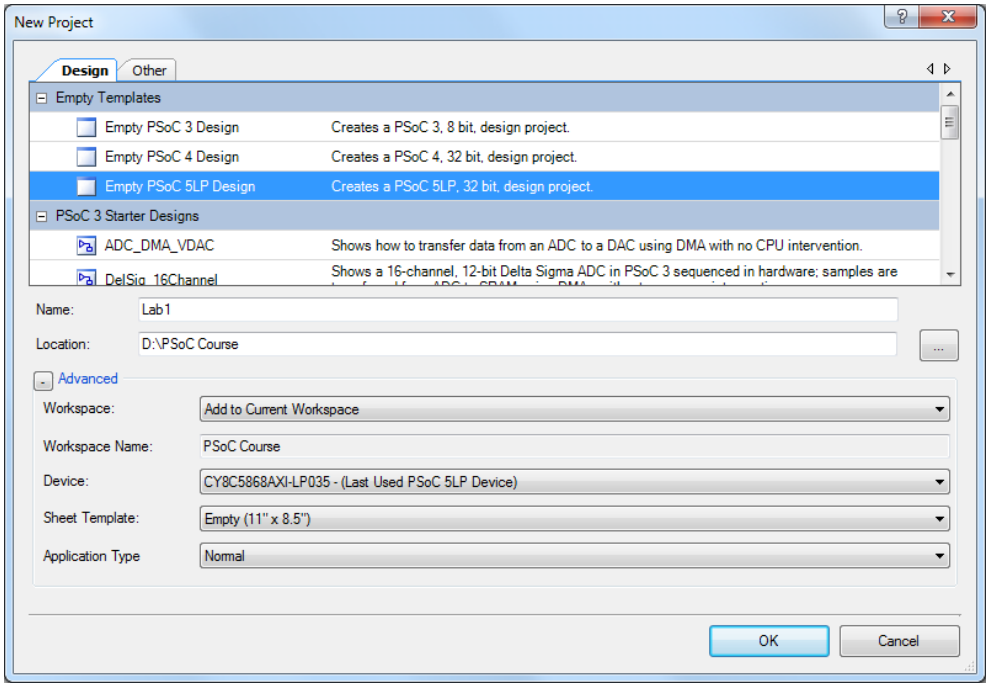
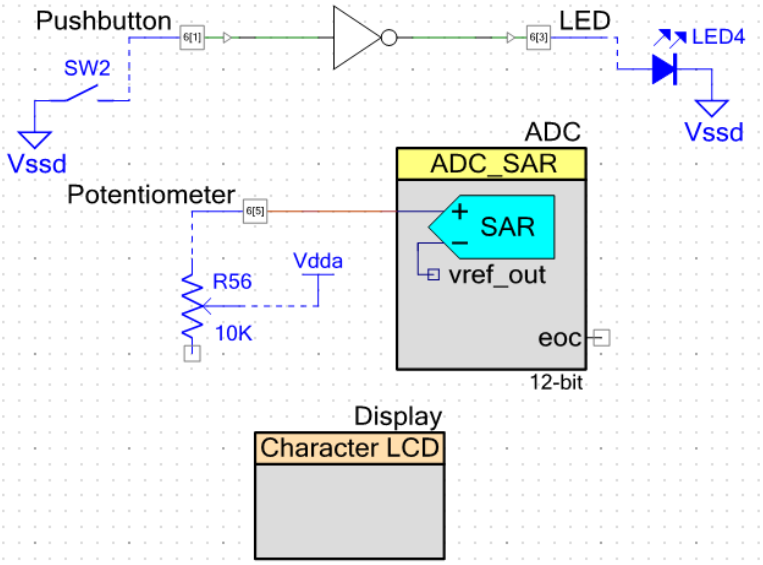


Figure 1: Creating a new project in the same workspace

Part 2: Designing the Schematic

1. Open TopDesign.cysch from Lab1.
2. Add a Ports and Pins>Digital Input Pin to the schematic area, name it Pushbutton, and change its drive mode to Resistive Pull Up. Do not uncheck the HW Connection box this time. Reminder: double clicking a component brings up its configuration window.
3. Add a Ports and Pins>Digital Output Pin to the schematic area and name it LED.
4. Because the pushbutton input pin outputs logic high ("true" or 1) when the button is not pressed, its logic needs to be inverted so the LED is off (receives logic low) when the button is not pressed. Add a Digital>Logic>Not to the schematic area to create a NOT gate.
5. Using the Wire Tool (button on the left edge of the schematic, or press W), connect Pushbutton to the input of the NOT gate, and connect the output of the NOT gate to LED.
6. Add a Ports and Pins>Analog Pin to the schematic area and name it Potentiometer.
7. Add an Analog>ADC>SAR ADC to the schematic area and make the following changes to its configuration:
    a. Name it ADC
    b. Change the Conversion rate (SPS) to 100000
    c. Change the Input range to "Vssa to Vdda (Single Ended)"
    d. Change the Reference to "Internal Vref"
8. Connect the Potentiometer to the input next to the + on ADC.
9. Add a Display>Character LCD to the schematic and name it Display.
10. The final schematic should look similar to what's below. The components in the schematic should look and be connected exactly the same, but not necessarily placed the same way, and the blue parts are optional annotations indicating the external hardware used.



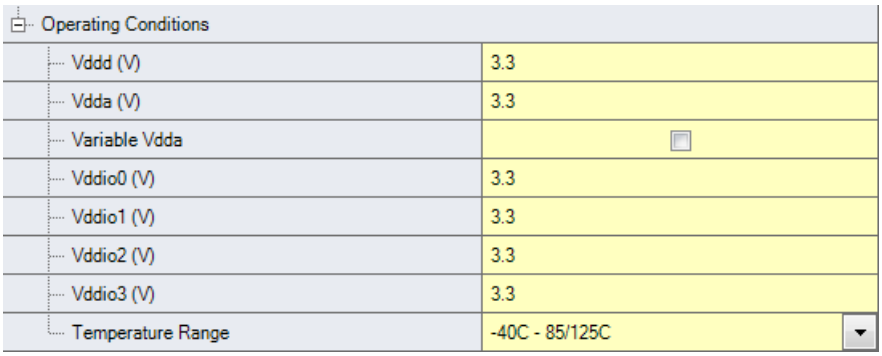
11. Save the schematic.

Part 3: Configure Design-Wide Resources

1. Open Lab1.cydwr.
2. Configure Display, LED, and Pushbutton to use the same ports as the previous lab: P2[6:0], P6[3], and P6[1] respectively.
3. For some reason, the port that the potentiometer is connected to is not labeled on the development kit's board. It is connected to P6[5], so configure Potentiometer to use that port.
4. See the image below for how the pin configuration should look.

Alias	Name	Port	Pin
	\Display:LCDPort[6:0]\	P2[6:0] Trace, Trace, Trace, Trace	95..99,1..2
	LED	P6[3]	92
	Potentiometer	P6[5]	7
	Pushbutton	P6[1]	90

5. Go to the System tab at the bottom of the DWR view.
6. Under Operating Conditions, change all of the voltages from 5.0 to 3.3. It is important to do this in every project that uses analog components from now on so the API functions that convert counts from the ADC to volts are correct.



7. Save the DWR file.

Part 4: Programming the Firmware

1. Generate the application.
2. Open main.c.
3. Use the code snippet at the end of this section to initialize the hardware.
4. Write a program to continuously read the ADC, filter the readings, and print the result to the LCD.
5. To read the ADC, call ADC_IsEndConversion(ADC_WAIT_FOR_RESULT) to wait for a reading (it takes time to do conversions) and get the result in counts using ADC_GetResult16(). Convert counts to mV using ADC_CountsTo_mVolts(uint16).
6. There tends to be jitter in readings from ADCs, so filter the input by taking several readings and average them, or have a rolling average.
7. When printing the result to the LCD, it may be desirable to utilize sprintf to properly pad the number that is printed. See the code snippet for an example. Please note that it is necessary to include “stdio.h” to use sprintf. Remember: because the programming language is C, the string type is a char array.
8. The software does not need to do anything with the pushbutton and LED because their functionality is entirely implemented in hardware.

```
#include <project.h>
//Needed for sprintf
#include "stdio.h"

int main()
{
    //Put variables here

    //Temporarily holds the string representation of the filtered
    //mV reading before printing it to the display
    char displayStr[5] = {'\0'};

    //Initialize the display with the strings that don't change
    Display_Start();
    Display_Position(0, 1);
    Display_PrintString("Potentiometer");
    Display_Position(1, 9);
    Display_PrintString("mV");
```

```
    //Initialize the ADC
    ADC_Start();
    ADC_StartConvert();

    //Loop forever
    for(;;)
    {
        //Read the ADC, filter the readings, and print the result to the LCD
        //Use the code below to print the mV reading to the LCD

        //Convert the number to a string with leading spaces
        sprintf(displayStr, "%4d", filteredReading);

        //Print the number to the display
        Display_Position(1, 4);
        Display_PrintString(displayStr);
    }
}
```

Part 5: Running the Project

1. Build and load the project onto the PSoC development kit.
2. Test the program to make sure it works:
    - Pressing the SW2 should turn on LED4, and releasing the button should turn it off.
    - Rotate the potentiometer, which is to the left of the prototyping area (see the images below). When it is rotated all to the left, the display should show 0 mV. When rotated all to the right, the display should show 3299 mV. The readings should smoothly change between those values as the potentiometer is rotated.

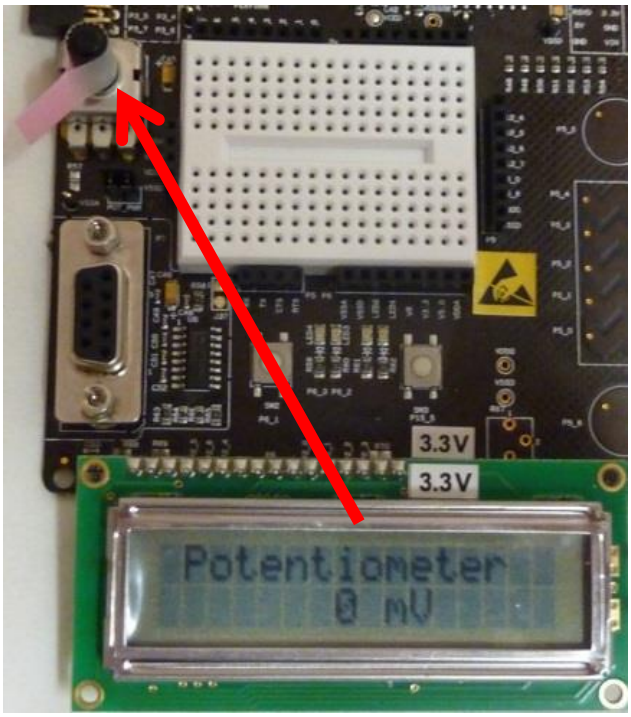


Figure 2: Potentiometer (upper left) at far left position. The sticky note shows the position of the potentiometer.

Figure 3: Potentiometer near a middle position

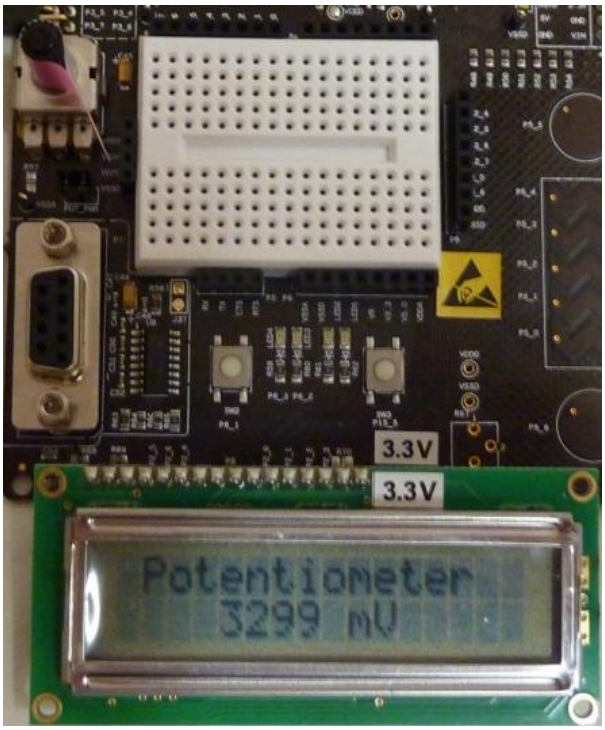


Figure 4: Potentiometer at far right position