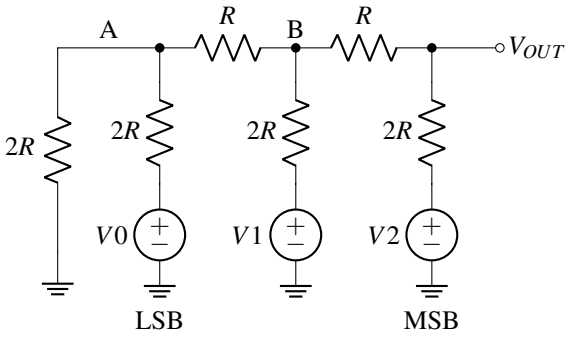
## Lab

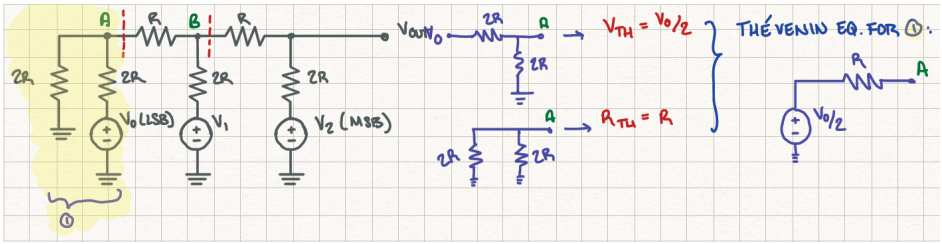
### Part 1: Digital-to-Analog Converters (DACs)

We will first build a 3-bit DAC to convert a **binary** input into an analog voltage, and then we will extend it to 4 bits using the knowledge you gleaned from building the 3-bit DAC. The binary input will come from the Launchpad's digital I/O pins while the analog voltage will be probed with your Launchpad and displayed on the serial monitor.




We can build a DAC using only resistors in a structure called the **R-2R ladder** (shown above). This structure takes an $n$-bit binary input and converts it to an output voltage. The bits here are represented by the voltage sources $V_0$ through $V_2$. You might have seen this structure in EE16A, but we will analyze it again today. Here is a quick review of superposition and Thévenin equivalent circuits to help you get started:

- **Thévenin's Theorem:** *A linear, active, resistive network containing one or more voltage or current sources can be replaced with a single voltage source and a series resistance.*
- To find $R_{TH}$, short out the voltage sources and find the equivalent resistance between output and ground.
- We suggest breaking the circuit up along the red dotted lines and working your way from each bit to the output to determine the contribution of each bit to $V_{\text{OUT}}$. *Hint: this may seem laborious at first, but you may notice a pattern that saves you a lot of work.* We have also done the first equivalent circuit for the LSB for you to help you get started.




**Now, complete Part 1 in the Jupyter notebook, starting with the questions.**

### Part 2: 4-Bit Analog-to-Digital Converter (ADC)

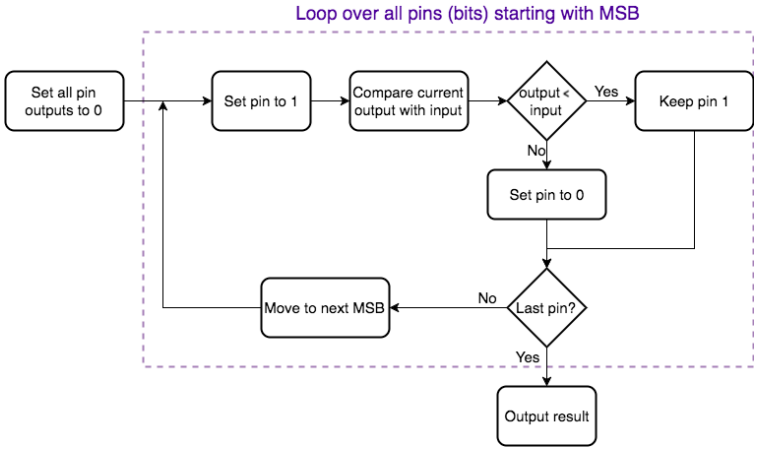
In this part of the lab, we will build a **Successive Approximation Register (SAR) ADC** using the 4-bit DAC you built in the previous part.

Given an analog voltage, an ADC measures it in the digital domain. Digitization of a signal involves both **discretization**, i.e., we restrict its timesteps to being finitely small (infinitely small timesteps ⇒ continuous time), and **quantization**, i.e., we fix a finite "set of states" (in this case, voltage values) that the signal can have at any given moment, which in this case is the set of integer values between 1 and $2^4 - 1$, inclusive. One commonly used circuit

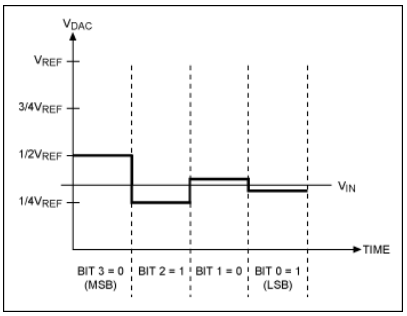
architecture for analog-to-digital converters is the Successive Approximation Register ADC (SAR ADC), which is what we'll be using today.

**The SAR ADC Algorithm**

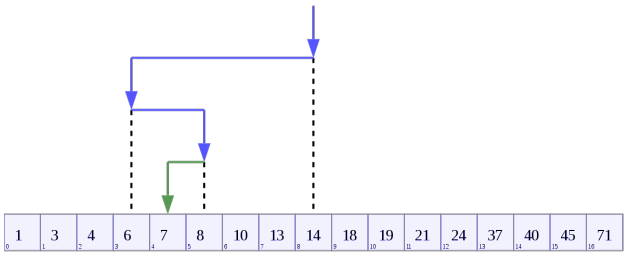
The SAR ADC algorithm tries various binary "trial codes" by feeding them into a DAC to generate voltages and comparing the result with the analog input voltage using a comparator. It then uses feedback to adjust the DAC voltage to get as close as possible to the analog voltage. Here is an illustration of the algorithm. Note that "pin outputs" refers to the MSP outputs connected to the DAC from Part 3, and "current output" means the output voltage of the DAC.



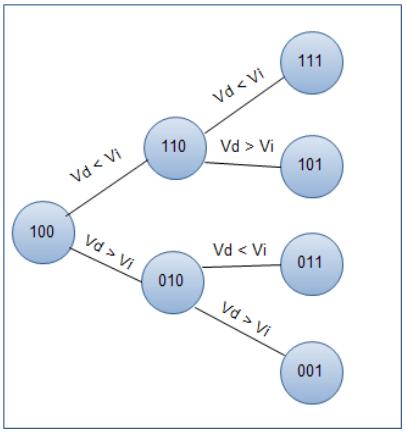

The images below demonstrate how the SAR ADC algorithm is analogous to the binary search algorithm you may have seen in CS 61B.




(a) Output of SAR ADC




(b) Representation of binary search



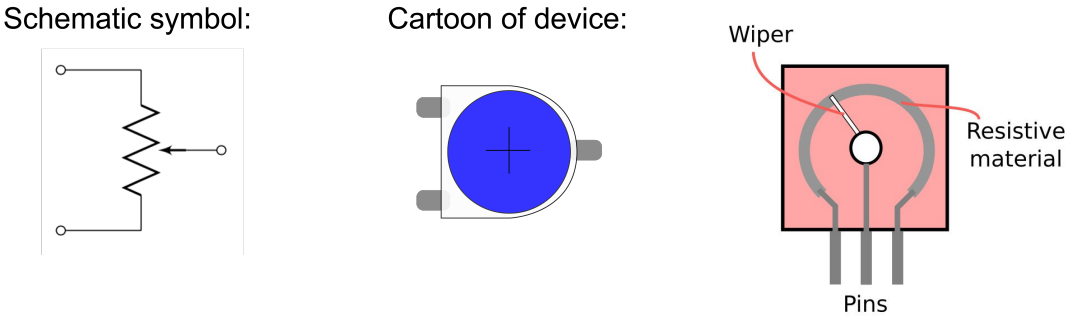

(c) Representation of 3-bit SAR ADC algorithm with binary "trial codes"

Note the similarities between the SAR ADC output in figure (a) and the demonstration of binary search in figure (b). You should be able to see something that looks like a variation of figure (a) when you read the output of your ADC circuit into the serial plotter, though the exact waveform will depend on which analog voltage $V_{IN}$ you send into the ADC.

In figure (c), $V_d$ is the DAC output voltage that corresponds to each given binary trial code, and $V_i$ is the input voltage the ADC is trying to approximate digitally.

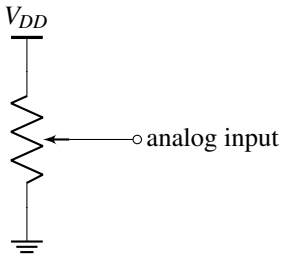
**The Potentiometer: Building your Analog Input**

To build your analog input, you are using a new device. The resistor with an arrow through it is called a potentiometer. The potentiometer has 3 terminals. Think of the top two terminals as one resistor and the bottom two as a a second resistor. The two resistors must add up to 50kΩ, but by turning the knob, you can make the first resistor large and the second small, or vice versa.



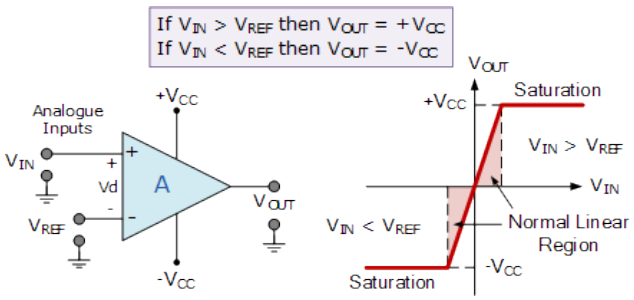

Inside the potentiometer, a wiper (which moves as you turn the knob) that is connected to the output pin sweeps over a ring of resistive material. When you turn the knob, you are changing the ratio between the two "resistors" to which the output is connected. We will use the potentiometer as a voltage divider (by connecting the input terminals to different voltage sources), but if only one input terminal and the output are connected, you can use the potentimeter as a variable resistor.

The circuit below will provide your analog input. The voltage at the node labeled "analog input" should change if you turn the potentiometer.




**The Comparator**

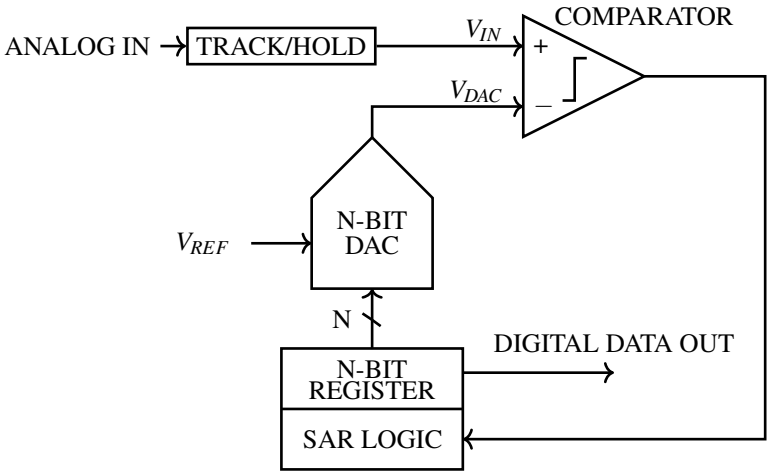
A comparator is simply an op-amp configured in open-loop, so that the output saturates at either the positive or the negative rail depending on whether the voltage at the noninverting terminal of the op-amp is greater or less than the voltage at the inverting terminal.




You will use a comparator to compare the output of the DAC to the analog input voltage you get from the potentiometer.

**Bringing It All Together**

The image below shows a block diagram of an N-bit SAR ADC. Let's walk through it.

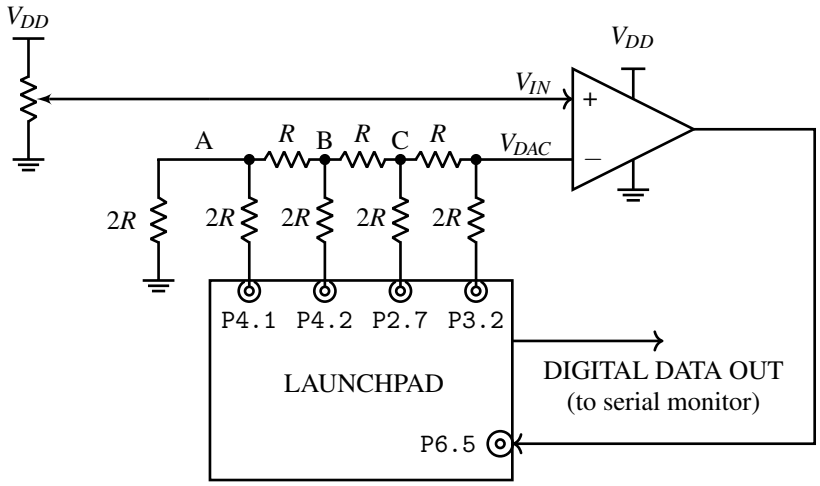



The voltage we are trying to read is at the analog input. The track/hold block samples this voltage and holds it steady at its output long enough for the SAR ADC algorithm to completely determine all N bits of the digital output; i.e., the voltage at the comparator's noninverting input won't change until all N cycles of the algorithm are complete. At that point, the track/hold block will resample the analog input and the ADC will read the new input voltage, and the whole algorithm will repeat.

At the start of the algorithm, the SAR logic sets all bits in the register to zero, thereby setting all the DAC bit voltages to zero. At this point, $V_{DAC} = 0$.

Then, we begin trying DAC voltages. The SAR logic sets the MSB to 1, which turns on the corresponding bit in the DAC. $V_{DAC}$ is then compared with $V_{IN}$. If $V_{DAC} < V_{IN}$, then the comparator's output is low, and the SAR logic moves on to trying the next bit. If $V_{DAC} > V_{IN}$, then the comparator's output is high, and the SAR logic sets the MSB back to zero before trying the next bit. This cycle repeats until all bits in the register are set, at which point the digital data is sent out.

Now, let's see how we will implement this ADC in the lab. Our ADC will have 4 bits of resolution, as shown in the image below. The Launchpad replaces the SAR logic (executed in code) and the registers (memory).




**Now, proceed to Part 2 of the Jupyter notebook and follow the directions there.**

*Written by Mia Mirkovic (2019). Version 2.0, 2020*