Complex DC Circuits

Goals:

1. Design, build and characterize a simple 5-bit digital-to-analog circuit based on resistor ratios.
2. Design, build and characterize a more complex 5-bit digital-to-analog circuit based on unit resistors.
3. Model both structures in MATLAB or Python.

Preparation:

1. Carefully review this document.
2. Be sure to understand the analysis of the circuits to be built: covered in Prelab problems 1 and 2.
3. Review lecture notes on node analysis, superposition, and Norton/Thevenin equivalents.
4. Review instrumentation from Lab 1 (you may want to bring it with you).

Experiments:

1. Resistor ratio-based Digital-to-Analog Converter (DAC)

The goal is to build a 5-bit DAC with a Thevenin equivalent Resistance $\mathbf{R_{TH}= 50\ \Omega}$ and a voltage $\mathbf{V_{TH} = (16b_4+8b_3+4b_2+2b_1+ b_0)/32}$, where b_k is the k^{th} bit (a voltage set to either 1 V or 0 V), for k = 0, 1, 2, 3, or 4, of some digital value.

The schematic and expected protoboard design are shown in Fig. 1. **Note** that the "voltage sources" shown in Figure 1a are actually implemented by switches which either connect to a dc voltage (V_{dc} = 1V) or to ground, depending on the bit setting.

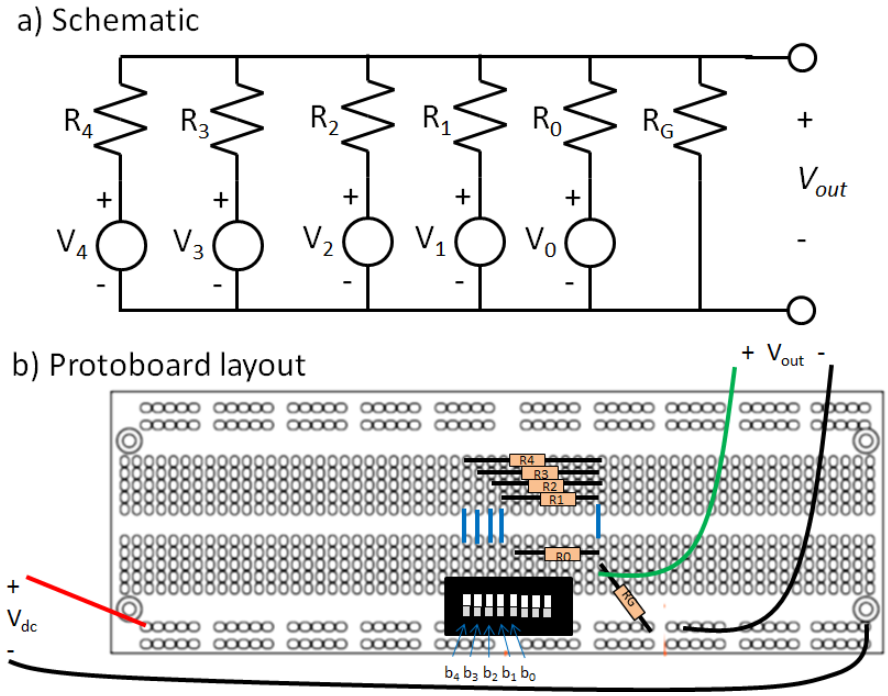


Fig. 1: Simple DAC, schematic and protoboard

(a) Get Parts: Based on prelab problem 1, you will need six resistors, and a protoboard with a switch array already mounted. Compute the resistor values you will need, and then get those whose values most closely approach your design values. Measure and record their actual values using the DMM. Be sure that they are the right order of magnitude (i.e., 50 Ω not 50 kΩ); people have been putting resistors in the wrong bins!

(b) Assemble the circuit, as shown in Fig. 1. Note that each switch output can be treated as a voltage source whose values is equal to $V_k = b_k \cdot V_{dc}$ (where b_k is the k^{th} bit, for k = 0, 1, 2, 3, or 4), that is, when the k^{th} bit $b_k=0$, $V_k=0$, when $b_k=1$, $V_k = V_{dc}$. V_{dc} is provided externally (next step). Also: **define b4 as the left-most switch, as shown in Fig. 1; ignore the numbers printed on the switch itself.**

(c) Provide power to the circuit: In order for the switch array to actually provide "1's" and "0's" it must have a DC power supply. This will be provided by one of the SMU's on your bench, configured manually (see Fig. 2). Turn on SMU1. To configure it as a voltage supply, push the button marked "V" under "source". To set the voltage push "edit" once and then use the left/right arrows to select the digit to change, and the up/down arrows to select the value. Configure for 1V. Push the "enter" button. Now, to set current compliance, push the "edit" button **twice**. You will need to push the up arrow under "range" three times to get the order of magnitude you need, then use the up/down, left/right "EDIT" to set the compliance to 100 mA. Attach the outputs to the VDC inputs on your board (red-to-red, black-to-black). This will provide the switch array with 0 V and 1V. In order to read back the amount of current being supplied, push the button marked "I" under "MEAS". In order to configure SMU2 to manually read out the output voltage, turn on its power, and push "I" under Source, and check to be sure "Isrc" is 0 (this is the default). Now push "V" under "MEAS". Connect SMU to the output of your circuit, black to the common node, red to the output node.

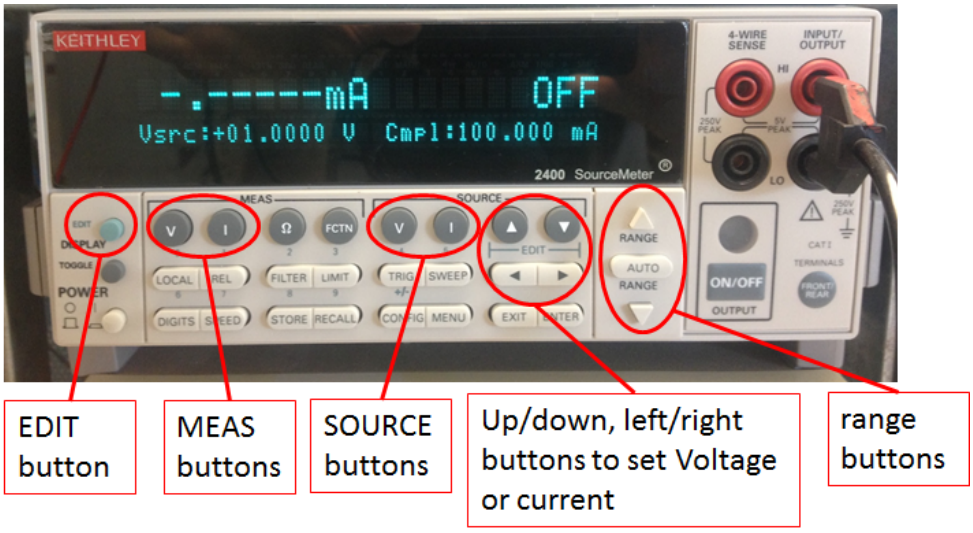


Fig. 2: critical buttons for using an SMU in manual mode

(d) Check basic functionality: Set all of the switches to "0" (all the way "down"), and push the ON/OFF buttons on both SMUs (so that they are ON). SMU1 should show current magnitude less than 1 μA, and SMU2 should show voltage less than 1 mV. Now toggle

b_4 to a "1" (***"1" is all the way "up"***: *switches neither "up" nor "down: are disconnected and look like open circuits*, *which will give weird results*): this should make SMU2 show approximately 0.5 V, and SMU1 show a current of about 5 mA. If you see dramatically different numbers, get help from a TA. *Note: if "Cmpl" starts blinking, your SMU is hitting compliance: if the current shown is much less than compliance, but Cmpl is still blinking, that means your SMU has become stupid and needs to be powered down and set up again.*

(e) Step through the 32 digital numbers by toggling the switches. In each case, record the output voltage displayed on SMU2, and current consumption of the circuit displayed on SMU1. Round off to the nearest 1 mV and 0.01 mA.

(f) Confirm the Thevenin equivalency of the circuit with an SMU. This requires extracting output I-V curves. Now configure the SMUs to run an automatic sweep using the lab tracer software. Configure ***SMU1 (connected across "Vdc") as a bias voltage***, with V = 1 V, compliance = 100 mA, also ensure that SMU is set to measure voltage and current (check "readback voltage" and "measure current in the configuration panel). Configure ***SMU2 (connected across V_{out}) to sweep voltage*** from 0 V to 1 V in 101 steps (compliance = 100 mA), and measure voltage and current. Run and ***save data*** for digital input settings: 00000, 01010, 10101, and 11111. Plot current_2 vs voltage_2 in each case. Do the results make sense (especially: does the open circuit voltage, when current = 0, equal what you got in part e)? If not, ask for help from a TA.

2. Ladder Digital-to-Analog Converter (DAC)

In this part you will build a somewhat more advanced DAC structure, based on Prelab, problem 2. Thevenin equivalent resistance R_{TH} = 50 Ω and a voltage $V_{TH} = (16b_4+8b_3+4b_2+2b_1+ b_0)/32$, but to do so using only 100 Ω and 50 Ω resistors.

(a) Get Parts: From prelab problem 2, you will need ten resistors and a protoboard with the switch array already mounted. Compute/decide the resistor values you will need for each resistor, and then get resistors whose values most closely approach your design. Measure and record their actual values using the DMM.

(b) Assemble the circuit, as shown in Fig. 3. As in part 1, each switch output can be treated as a voltage source whose values is equal to $V_k = b_k \cdot V_{dc}$ (where b_k is the k^{th} bit, and k can be 0, 1, 2, 3, or 4), that is, when the k^{th} bit b_k=0, V_k=0, when b_k=1, $V_k = V_{dc}$. V_{dc} is provided externally (next step).

(c) Provide power to the circuit: Use the same arrangement as in part 1c. If the SMU is unresponsive, try pushing the small button on the lower left marked "FRONT/REAR"

(d) Check basic functionality: Set all of the switches to "0", and push the ON/OFF buttons on both SMUs (so that they are ON). SMU1 should show current magnitude less than 1 μA, and SMU2 should show voltage less than 1 mV. Now toggle b_4 to a "1": this should make SMU2 show approximately 0.5 V, and SMU1 show a current of about 5 mA. If you see dramatically different numbers, get help from a TA.

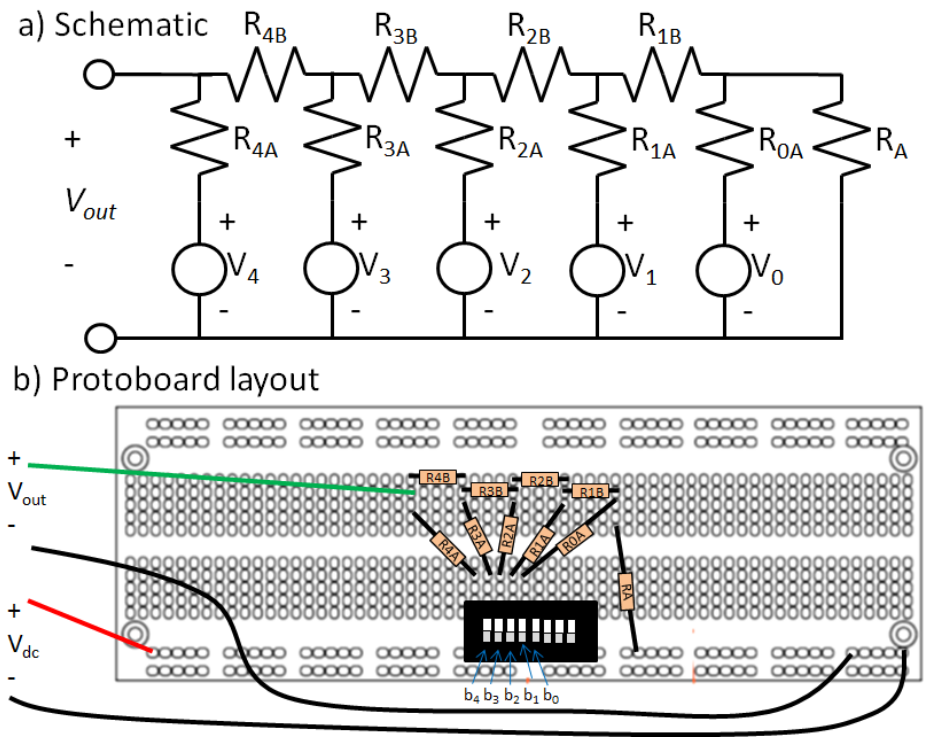


Fig. 3: Ladder DAC, schematic and protoboard

(e) Step through the 32 digital numbers by toggling the switches. In each case, record the output voltage from SMU2, and current consumption of the circuit from SMU1. Round off to two significant digits.

(f) Confirm the Thevenin equivalency of the circuit with an SMU. This requires extracting output I-V curves. Now configure the SMUs to run an automatic sweep using the lab tracer software. Configure ***SMU1 (across V_{dc}) as bias voltage***, with V = 1 V, compliance = 100 mA and measure voltage and current. Configure ***SMU2 (across V_{out}) to sweep voltage*** from 0 V to 1 V in 101 steps, and measure voltage and current. Run and ***save data*** for digital input settings: 00000, 01010, 10101, and 11111. Also plot current_2 vs voltage_2 in each case. Do the results make sense? If not, ask for help from a TA.

Wind down:

Clean up around your bench and return any components back to their storage bins. **Please DO NOT disassemble the switch array-protoboard combination!** Be sure all data is collected and placed on your own storage media. Delete all files on your desktop or at least organize them in a folder. ECE makes no guarantee that these files left on your desktop will remain over time.

Analysis:

1. For the simple DAC, plot the measured output voltage vs the binary number N (that is 00000 = 0, 00001 = 1, 00010 = 2, etc.).
2. Also, plot the current consumed vs the binary number N. Explain its shape.
3. Plot the difference (error) between the measured output voltage and the ideal voltage $(16V_4+8V_3+4V_2+2V_1+V_0)/32$, vs the binary number N. Is there an obvious pattern to the

error, as a function of N? What is the largest error? What is the mean-square-error (MSE: square the error for each setting, take the mean)? Describe/try to explain the shape of this error curve.

4. Plot the I-V curves obtained from the SMU for the binary number N=0, N=10, N=21, and N=31 and extract the output resistance from their slopes.
5. For the ladder DAC, plot the measured output voltage vs the binary number N (that is 00000 = 0, 00001 = 1, 00010 = 2, etc).
6. Also, plot the current consumed vs the binary number N. Explain its shape.
7. Plot the difference (error) between the measured output voltage and the ideal voltage $(16V_4+8V_3+4V_2+2V_1+V_0)/32$, vs the binary number N. Is there an obvious pattern to the error, as a function of N? What is the largest error? What is the mean-square-error (MSE: square the error for each setting, take the mean)? Describe/try to explain the shape of this error curve.
8. Plot the I-V curves obtained from the SMU for the binary number N=0, N=10, N=21, and N=31 and extract the output resistance from their slopes.
9. **Model the two DAC Circuits in MATLAB/Python** as described in "Appendix A: Numerical Circuit Modeling" below.

Appendix A: Numerical Circuit Modeling

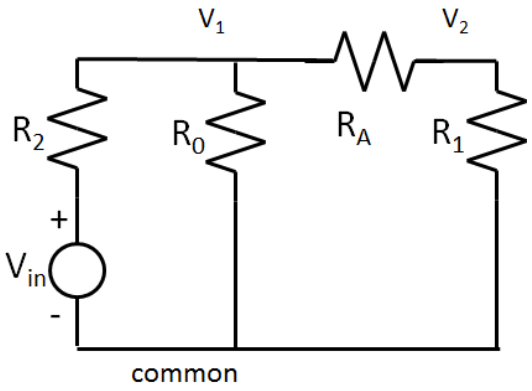
Goals:

1. Practice using matrix-formulations of node-voltage equations for fast solutions.
2. Predict Behavior of two Digital-to-Analog Converter circuits based on ideal and real resistor values.
3. Compare simulated results to formal analysis (Prelab 2) and measured results (Lab 2).

Many circuits are simply too large to realistically solve by hand (that is they have too many nodes). Even in cases where other approaches work (e.g., source transforms) it can still be cumbersome in cases where the components are not regular values. In such cases, it often makes sense to use a computer to solve the system of equations for us.

1. Example Circuit

Here is a brief, worked example of how to use MATLAB to solve node voltage analysis: (If you prefer to use python, the same approach should work, with slightly different syntax).



Start with the circuit shown.
Let R1 = 100 Ω, R2 = 200 Ω, R0 = 100 Ω, RA = 50 Ω, R1 = 50Ω, Vin = 1V.

Write node voltage equations:

$$0 = \frac{V_{in}-V_1}{R_2} + \frac{V_2-V_1}{R_A} - \frac{V_1}{R_0},$$
$$0 = \frac{V_1-V_2}{R_A} - \frac{V_2}{R_1}$$

Now re-arrange to isolate the independent terms:

$$\frac{V_{in}}{R_2} = V_1\left(\frac{1}{R_2}+\frac{1}{R_A}+\frac{1}{R_0}\right) - V_2\frac{1}{R_A},$$
$$0 = -V_1\frac{1}{R_A} + V_2\left(\frac{1}{R_A}+\frac{1}{R_1}\right)$$

Write as a matrix:

$$\begin{pmatrix}\frac{V_{in}}{R_2}\\ 0\end{pmatrix} = \begin{bmatrix}\left(\frac{1}{R_2}+\frac{1}{R_A}+\frac{1}{R_0}\right) & -\frac{1}{R_A}\\ -\frac{1}{R_A} & \left(\frac{1}{R_A}+\frac{1}{R_1}\right)\end{bmatrix}\begin{pmatrix}V_1\\V_2\end{pmatrix}$$ which we can also write as $\vec{\imath_n} = G\vec{v}$

Now open MATLAB, and in the command window (or in a new .m-file) Type:

```
R1=100;
R2=200;
R0=100;
RA=50;
Vin=1;
in=[Vin/R2; 0]
```

```
G=[1/R2+1/RA+1/R0, -1/RA;
      -1/RA, 1/RA+1/R1]
```

Now, to solve, just compute $\vec{v} = G^{-1}\vec{in}$ by typing:

```
v=inv(G)*in
```

which should return:

v =

0.2308

0.1538

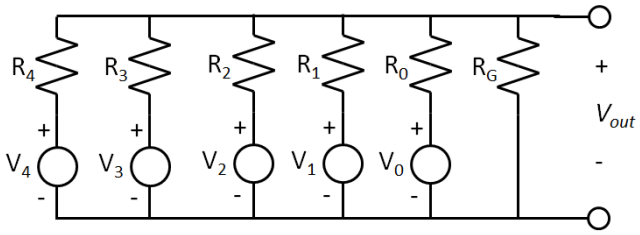
That is, V_1=0.2308V, and V_2 = 0.1539V

If you also want to find the power supplied by Vin:

```
Pdiss=Vin*(Vin-v(1))/R2
```

You can confirm these results with hand calculation using whichever analysis method you prefer.

2. Binary-Weighted Resistor DAC



(a) Before writing any code, start by using KCL on the output node to write a single equation for V_{out} as a function of V_0-V_4, R_G, and R_0-R_4. Also write an equation for the power supplied by each voltage source (V_0-V_4) as a function of that voltage, V_{out}, and the relevant resistance, as well as an equation for the total power consumed (which is just the sum of all the individual powers supplied).

(b) Now download and open the MATLAB file "code_for_DACs". Define your resistors' values at the beginning of the script (after the definition of "Codes" but before the for-loop) based on your analysis from Prelab problem 1. That is, type:

```
RG=1500;
R0=1500;
```

etc… (don't actually use these values!)

(c) Now, inside the for-loop include your equation for V_{out} in terms of these resistances and V_0, V_1, V_2, V_3, and V_4 (which are already defined) after the lines where the voltages were defined. You will want to make V_{out} an indexed value (V_{out}(n)) so that you can plot it vs n later. Also compute the total power supplied by the voltage sources, and index that as well (that is, write an equation for Pdissg(n)). Be sure to end each line of code with semicolon or MATLAB will print everything as it goes.

(d) Finally, add some code at the end of the script to plot V_{out} vs N, and P vs N.

(e) Run your code (F5 is the hot key to save and run). Resolve your inevitable bugs. And look at your plots. Do your results make sense? If you want to save this data, type

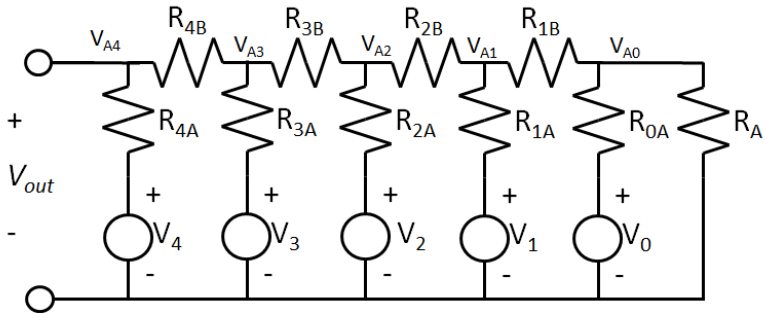
```
save 'filename' N Vout Pdiss
```

Where you should replace filename with the name of the file.

(f) Now, change your resistor definitions to reflect the *actual* resistor values you used in lab and re-run.

(g) Overlay your simulated and measured results for V_{out} vs N and P_{supp} vs N. Comment on similarities/discrepencies.

3. Resistor-Ladder DAC



(a) Before writing any code, write the node voltage equations for V_{A0} through V_{A4} (where $V_{out} = V_{A4}$), and formulate them in vector-matrix form. Also write an equation for the power from each voltage source (V_0-V_4) as a function of that voltage, of the internal nodes (V_{A0}-V_{A4}), and the relevant resistance, as well as an equation for the total power consumed.

(b) Now make a copy of your earlier code, and based on your analysis from Prelab problem 1, define R_A, R_{0A}-R_{4A} and R_{0B}-R_{4B}.

(c) Define your matrix in terms of these resistors (it should be a 5x5 matrix) and compute its inverse (that is write code to define the matrix and its inverse). Note once you have defined a matrix G, its inverse in MATLAB is just R=inv(G).

(d) Now, inside the for-loop define the input vector based on V_0-V_4, and compute the node-voltage vector based on the input vector and matrix.

(e) Based on this vector, extract Vout(n) and P(n). (remember, Px for each input is (V_x-V_{Ax})/R_{XA}, total power is the sum of Px across x=0-4)

(f) Finally, add some code at the end of the script to plot V_{out} vs N, and P vs n.

(g) Run your code, resolve your inevitable bugs. And look at your plots. Save them. Do your results make sense?

(h) Now, change your resistor definitions to reflect the *actual* resistor values you used in lab and re-run.

(i) Overlay your simulated results from h. and measured data for V_{out} and P_{supp}. Comment.