Physics 310
Lecture 9b – DAC and ADC

Wed. 3/21 Thurs. 3/22 Fri. 3/23	More of **Ch 14.1, .6-.10; pp 373-374 (**Sampling Frequency**); 12.6:** ADC & DAC **Lab 9:** ADC & DAC More of the same; **Quiz** Ch 14	**HW 9: A*** & Ch 14 Pr 13*, 17* **Lab 9 Notebook**
Mon. 3/26 Wed. 3/28 Thurs. 3/29	**Project:** Component Shopping **Review** ***Sr. Seminar Presentations****(quiz points)* **Exam 2**	**Project Progress Report** (due at the beginning of class)

Announcements:

- **Project Proposals due Today**

Study List for Quiz #9:

- Analog-to-digital conversion (ADC)
  - How analog input and digital output are related
  - Conversion times
  - Nyquist criterion – sampling rate
- Digital-to-analog conversion (DAC)

Equation List:

Nyquist criterion: $f_{\text{sample}} > 2f_{\text{max}}$

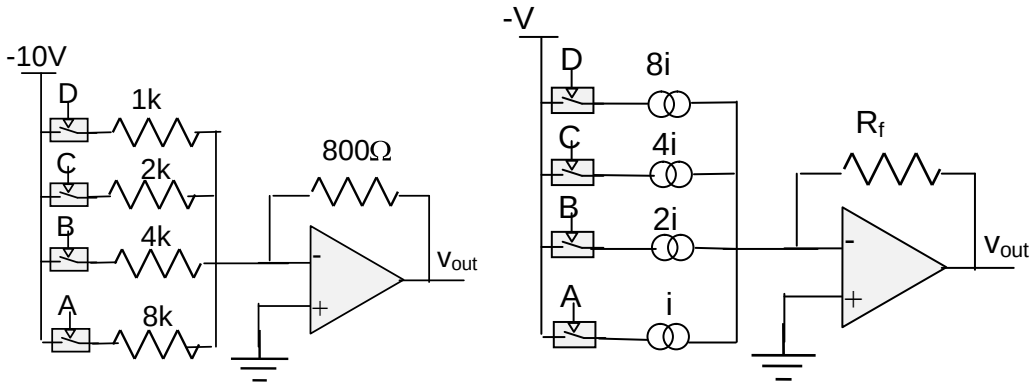
Handout:

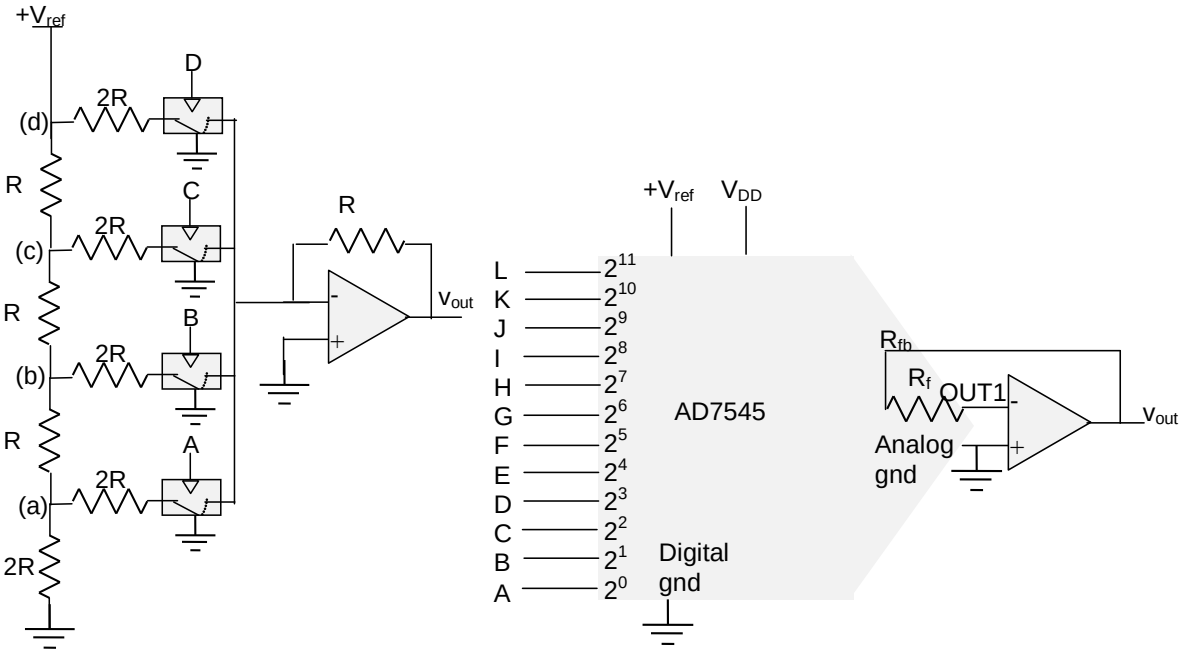
- Lab #9
- DAC circuits
- Flash ADC (2 bit)
- Project order list
-

14.1 Intro:

14.6 Digital –to-Analog Converters: DACs
Weighted Current Source DAC

Physics 310
Lecture 9b – DAC and ADC





Conversion:

$$V_{out} = Integer_{in}\frac{V_{ref}}{2^n}$$

Resolution:

$$\Delta V_{out} = 1\frac{V_{ref}}{2^n}$$

14.7 Analogue to Digital Converters ADC's

If the conversion rule for a DAC is $V_{out} = Integer_{in}\dfrac{V_{ref}}{2^n}$, then we'd guess that the conversion rule for an ADC *should* be $Integer_{out} = V_{in}\left(\dfrac{2^n}{V_{ref}}\right)$.

Two qualifiers go along with this. First, the right hand-side doesn't yield an actual *integer* for most values of V_{in}. So how is that handled? Some converters essentially round to the nearest integer, while other models truncate. Second, one scheme we'll look at actually has

Physics 310
Lecture 9b – DAC and ADC

$$Integer_{out} = round\left[V_{in}\left(\frac{2^n - 1}{V_{ref}}\right)\right]$$

while another has

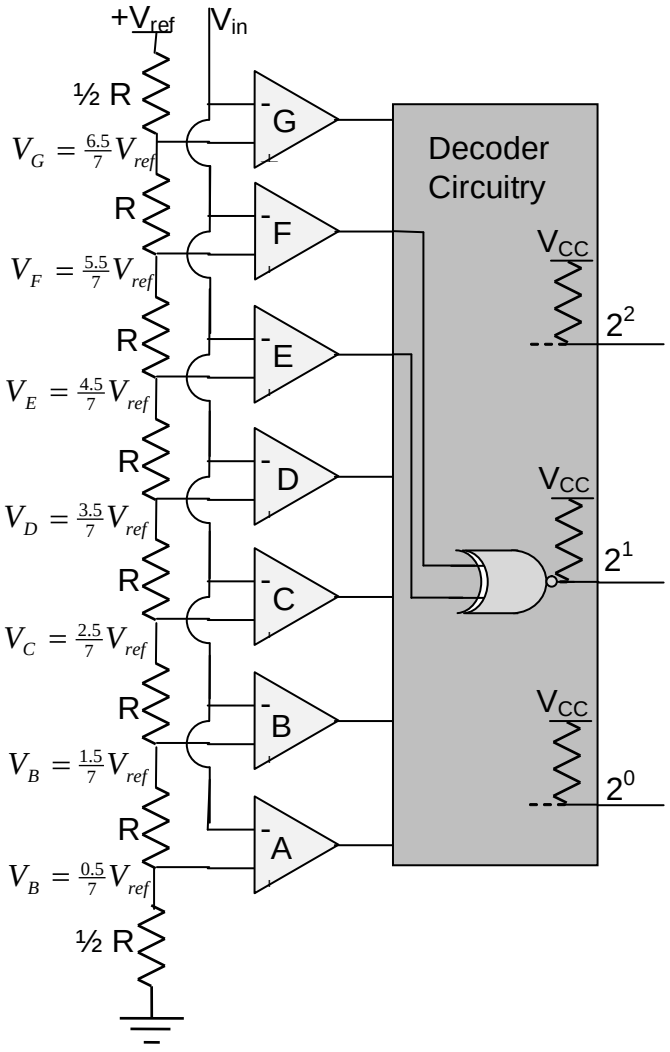
$$Integer_{out} = trunc\left[V_{in}\left(\frac{2^n}{V_{ref}}\right)\right],$$

and still another uses a completely different scheme that doesn't involve a V_{ref} *per se*. When the rubber hits the road, or rather, when the chip hits the socket, you'll want to look at the spec. sheet for you specific chip.

Now for some specific designs.

- **Parallel** (or Flash) these use a number of comparators in parallel.
  - In general, to generate *n* bits, it takes 2^n-1 comparators. So, for something modest like 12 bits, it takes 4,095 comparators! Not cheap. But they are pretty fast; able to operate at 100s of *MHz*, i.e., settling times of *ns*.
  - The comparator reference voltages are necessarily evenly spaced.
  - Here's a 3-bit ACD. It takes 2^3-1=7 comparators

Physics 310
Lecture 9b – DAC and ADC



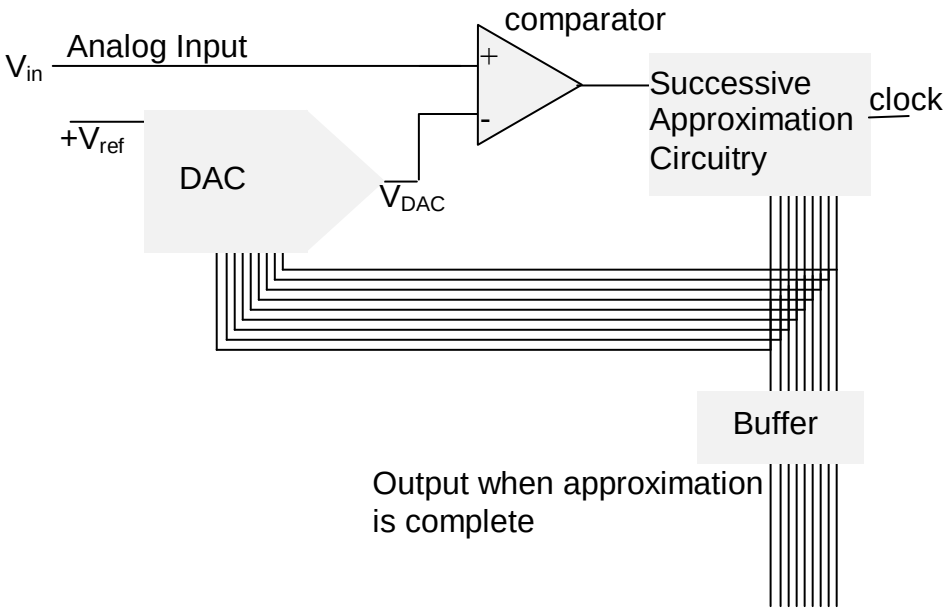
This "decoding" can easily be achieved with a collection of open-collector logic gates. The simplest to imagine would be X-NOR's.

For example, say V_{ref} = 7Volts and V_{in} = 5.2V. That would mean comparators A through E give Hi outputs while F and G give Lo outputs (since V_{in} is in the 4.5-5.5V range.) An X-NOR watching E and F would pull its output Lo only in this case, so wiring its output to the 2^1 output line (and allowing the other two output lines to be pulled up) would generate 1 0 1 (binary for 5.)

Successive Approximation

This ADC actually uses a DAC to successively approximate the input signal, a comparator helps keep track.

Physics 310
Lecture 9b – DAC and ADC



(Note: this is different from the book's schematic in how the comparator is wired up; I'm not sure exactly what the book *meant* to have, but the illustration looks nonsensical. The scheme shown here achieves the basic operation intended.)

Here's how it operates. In broad strokes, it works its way down from setting the most significant bit to finally setting the least significant bit. Here's a blow-by-blow; to make this concrete, imagine we have an 8-bit system with a 10V reference and an actual Analog Input of 6.8 V.

1. **Determining Most Significant Digit** (2^7)
   a. First the Successive Approximation Circuitry makes the most significant digit Hi and all less-significant ones Lo. Since that's essentially ½ the full range of values that it can represent in binary, the DAC then outputs ½ V_{ref}.
      i. In this specific example, (8-bit, 10V ref) the SAC would produce 10000000 (i.e. $2^7 = 128$) and the DAC would therefore produce

Test 10000000

$$V_{DAC} = 10V\,\frac{2^7}{2^8 - 1} = 5.02V$$

   b. Now, the Comparator compares this value with V_{in}. If V_{in} is higher than this, the Comparator reports Hi and the SAC will ever-after hold the most significant bit Hi; if the input is lower than this, the Comparator reports Lo and the SAC will ever-after hold the most significant bit Lo.

Keep 10000000

      i. In this specific example Vin = 6.8V > 5.02V = V_{DAC}
2. **Determining Next Most Significant Digit** (2^6)
   a. Now the SAC makes the next most significant digit Hi and all less significant ones Lo. That essentially tells the DAC to add another ¼ V_{ref} to its output.
      i. In this specific example, the SAC would be production 11000000 and the

Test 11000000

DAC would therefore produce $V_{DAC} = 10V\,\frac{2^7 + 2^6}{2^8 - 1} = 7.53V$

   b. Now, the Comparator compares this value with V_{in}. If V_{in} is higher than this, the Comparator reports Hi and the SAC will ever-after hold the this bit Hi; if the

Keep 10000000

input is lower than this, the Comparator reports Lo and the SAC will ever-after hold this bit Lo.

i. In this specific example, Vin = 6.8V < 7.53V = V_{DAC}

3. **Determining Next Most Significant Digit** (2^5)
   - **a.** You probably get the picture by now. The SAC sets the next bit Hi, the DAC adds about 1/8 V_{ref} to its previous output. The comparator compares this new value with Vin and tells the SAC whether or not to keep that bit Hi / the DAC whether or not to keep that additional 1/8 V_{ref}.

Test 10100000

Keep 10100000

   - **i.** In this specific example $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5}{2^8 - 1} = 6.27V$ <6.8V=V_{in}, so that new "1" is a keeper.

4. **Determining Next Most Significant Digit** (2^4)
   - **a.** Setting the next bit Hi adds about 1/16 Vref to V_{DAC}. That new value gets compared to Vin to determine whether or not that bit should be kept Hi.

Test 10110000

Keep 10100000

   - **i.** $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5 + 2^4}{2^8 - 1} = 6.90V$ > 6.8 V=V_{in}. So that new "1" is rejected.

5. **Determining Next Most Significant Digit** (2^3)
   - **a.** Setting the next bit Hi adds about 1/32 V_{ref} to V_{DAC}. That new value gets compared to Vin to determine whether or not that bit should be kept Hi.

Test10101000

Keep 10101000

   - **i.** $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5 + 0*2^4 + 2^3}{2^8 - 1} = 6.59V$ < 6.8V=Vin. So that new "1" is a keeper.

6. **Determining Next Most Significant Digit** (2^2)
   - **a.** Setting the next bit Hi adds about 1/64 V_{ref} to V_{DAC}. That new value gets compared to Vin to determine whether or not that bit should be kept Hi.

Test 10101100

Keep 10101100

   - **i.** $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5 + 0*2^4 + 2^3 + 2^2}{2^8 - 1} = 6.75V$ < 6.8V=Vin. So that new "1" is a keeper.

7. **Determining Next Most Significant Digit** (2^1)
   - **a.** Setting the next bit Hi adds about 1/64 V_{ref} to V_{DAC}. That new value gets compared to Vin to determine whether or not that bit should be kept Hi.

Test 10101110

Keep 10101100

   - **i.** $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5 + 0*2^4 + 2^3 + 2^2 + 2^1}{2^8 - 1} = 6.82V$ > 6.8V=Vin. So that new "1" is rejected.

8. **Determining Least Significant Digit** (2^0)
   - **a.** Setting the next bit Hi adds about 1/128 V_{ref} to V_{DAC}. That new value gets compared to Vin to determine whether or not that bit should be kept Hi.

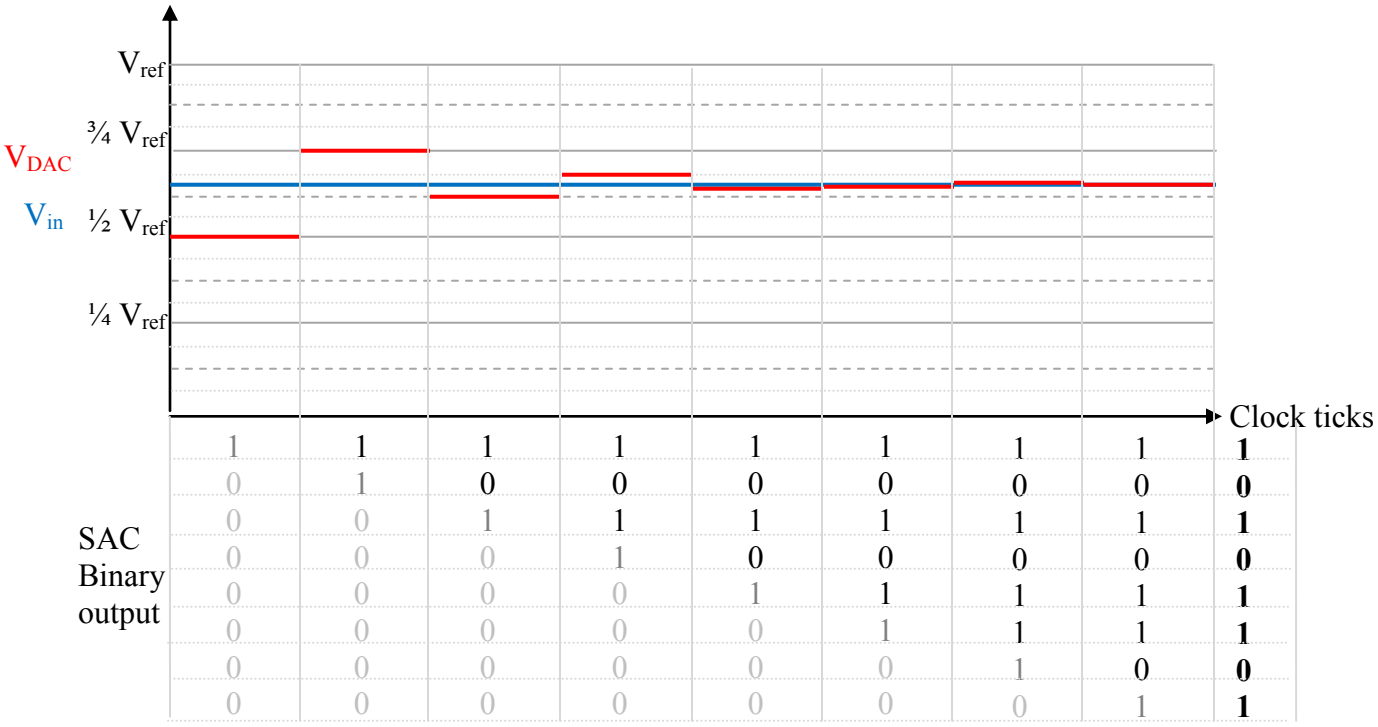
Test 10101101

Keep 10101101

   - **i.** $V_{DAC} = 10V\,\frac{2^7 + 0*2^6 + 2^5 + 0*2^4 + 2^3 + 2^2 + 0*2^1 + 2^0}{2^8 - 1} = 6.78V$ < 6.8V=Vin. So that new "1" is kept.

Tada! Now that each bit has been determined, the buffer outputs these 1 & 0's for everyone to see.

Here's a more visual way of seeing this process play out.



Down sides:

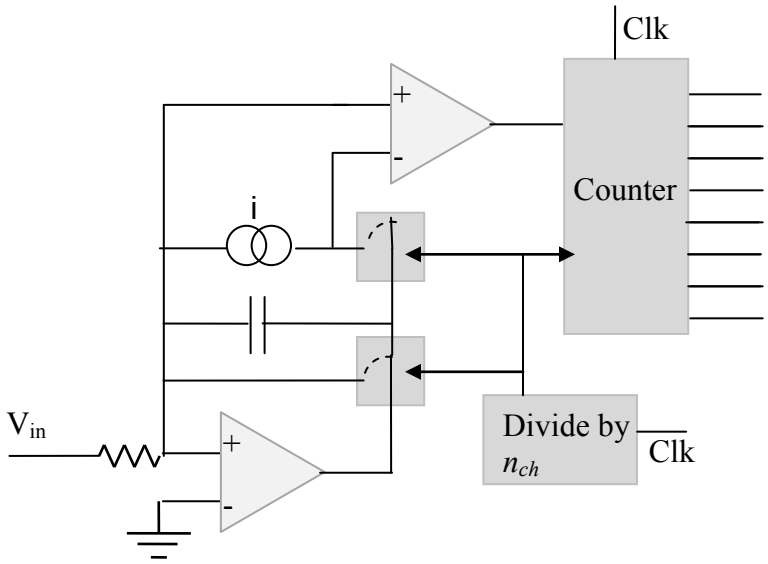
Slow-ish. Unfortunately, this process takes a bit of time. You might imagine that an *n*–bit ADC of this type takes about *n* times as long to process a value as does a Parallel / Flash ADC which does all bits at the same time. A 12 bit ADC of this type can run at about 1 MHz, that is, it can convert a new value about once every μs.

Susceptible to blips. Say that Vin is a little noisy, or maybe there's just a substantial one-time blip. That could really throw off the ADC. If our 6.8V signal suffered a 1V blip early in the conversion process, it could lead to a horribly incorrect binary representation. For that matter, the same would be true of the Parallel / Flash ADC. The only way these circuits could overcome that difficulty would be to employ some error checking – repeatedly converting and just keeping the most consistent, but that takes time and slows down the over-all performance.

→ **Group Problem**

Integrating type: Dual-Slope conversion
Here's conversion process that's still slower, but much less sensitive to blips. You've got a capacitor which you first *charge* for a *set number of clock ticks* and at a rate that is proportional to the input analog voltage, V_{in}. So its final voltage will be proportional to that input voltage. $V_{cap} \propto V_{in}$. You do this using an Op-amp and resistor with the capacitor in an 'integrator' configuration. Next, you *discharge* the capacitor at a *set rate,* so the *number of clock ticks* it takes to discharge will be proportional to the that capacitor voltage, which is itself proportional

to the input voltage. $n_{discharge} \propto V_{cap} \propto V_{in}$. You achieve this by discharging through a constant current source. To come up with a binary representation of the value of V_{in}, the circuit uses a counter to count the number of ticks to discharge (a comparator tells it when the capacitor is fully discharged.)



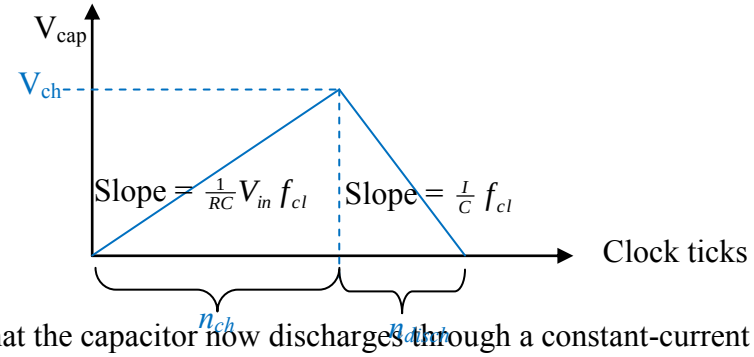
To walk through it in some detail, this rough schematic doesn't show all the details, but it gives the basic idea. A capacitor in an Integrator configuration charges up linearly for a set number of clock ticks n_{ch}; given the clocks frequency, that means a set period of time, $t_{ch} = n_{ch} f_{cl}$, with the voltage across it proportional to Vin, more specifically,

$$V_{cap}(t) = \tfrac{1}{RC} V_{in} t \,.$$

So after the given number of ticks, the capacitor is charged to

$$V_{ch} = V_{cap}(t_{ch}) = \tfrac{1}{RC} V_{in} t_{ch}$$

$$V_{ch} = \tfrac{1}{RC} V_{in} n_{ch} f_{cl}$$



Next, the switches flip so that the capacitor now discharges through a constant-current source of set current, I (and the op-amp's output is reset to 0.) Meanwhile, the voltage across the capacitor is monitored by a comparator, and the Counter counts off, in binary, the number of clock ticks until the comparator's sign flips / the capacitor's voltage crosses zero. If t_{ch}, R, and I are chosen appropriately for the clocks rate, then the number of clock ticks *equals* V_{in}. Thus V_{in} is now represented in Binary.

$$n_{disch} = t_{disch} f_{clock}$$

$$I = \frac{Q_{charged}}{t_{disch}} = \frac{CV_{charged}}{t_{disch}} = \frac{C\frac{1}{RC}V_{in}n_{ch}f_{cl}}{t_{disch}} = \frac{\frac{1}{R}V_{in}n_{ch}f_{cl}}{t_{disch}}$$

so

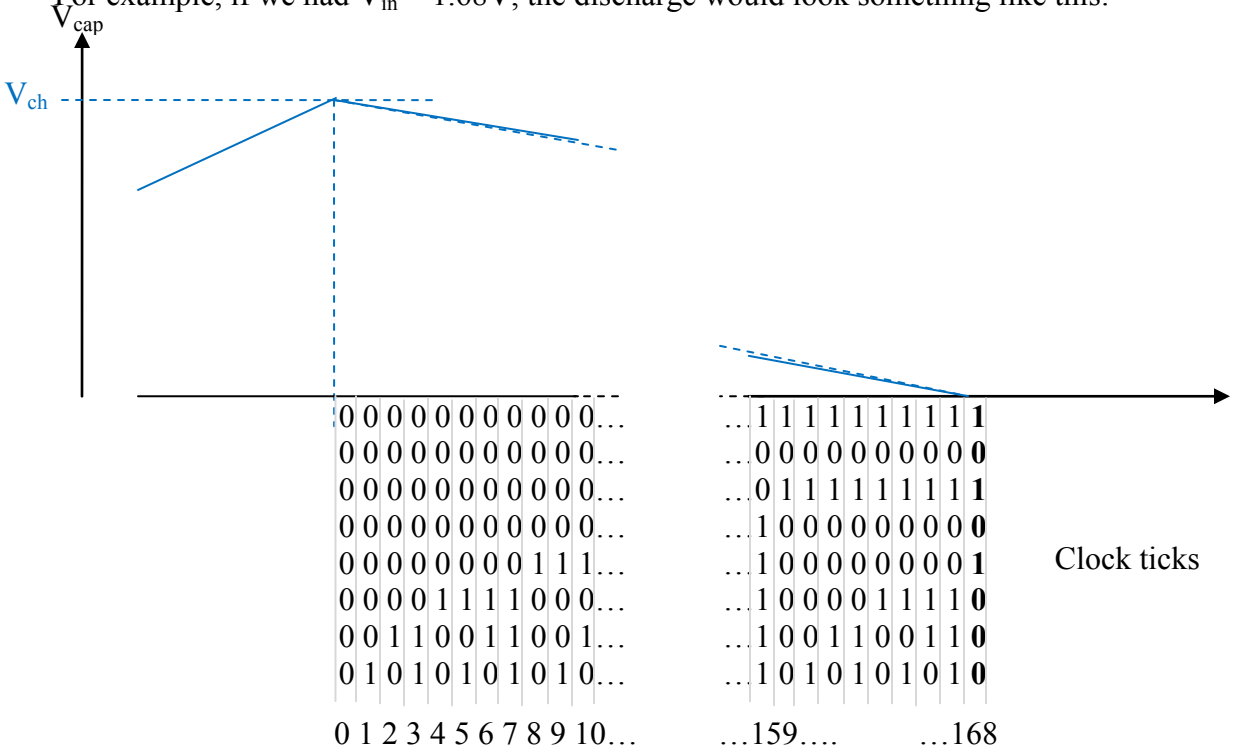
$$t_{disch} = \frac{V_{in} n_{ch} f_{cl}}{RI}$$

Thus,

$$\#ticks = \frac{V_{in} n_{ch}}{RI}$$

So, if the resistance and current are chosen such that $RI = n_{ch}$, then $\#ticks = V_{in}$. Alternatively, if RI are chosen to be $1/100^{th}$ of n_{ch}, then $\#ticks = 100 * V_{in}$

For example, if we had V_{in} = 1.68V, the discharge would look something like this:



This method is fairly insensitive to blips on Vin and the effect of noise tends to average out through the integrating / charging up process.

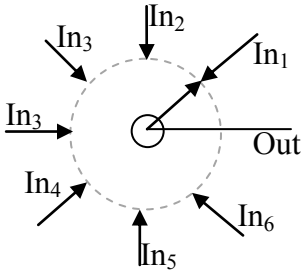
14.8 Support circuitry

Along with ADC's, often go multiplexers and sample-holds.

12-6 Multiplexers

To motivate the utility of a multiplexer, think about your computer keyboard. One way or another, it needs to report to the computer the status of each of umpteen keys, mine has 122 keys. Then again, it's plugged into your computer with a USB cable which only has a couple of data lines (I'd guess just one for sending info out and one for taking info in). Presumably, when you press an individual key it sets an individual data line inside the keyboard 'Hi'. How does the

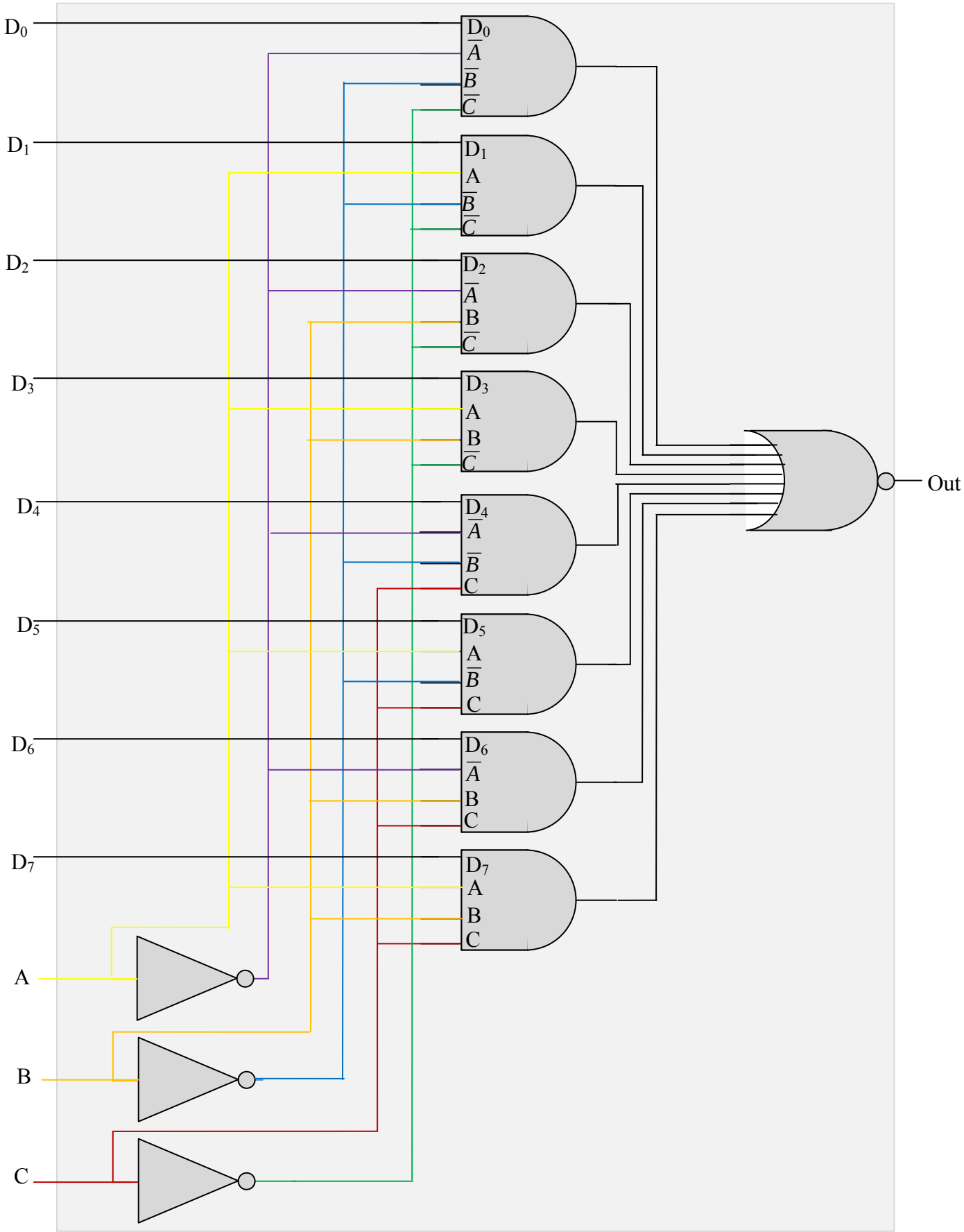
status of each key's data line get transmitted over just one wire to the computer? There's got to be something inside the keyboard which monitors each key's value (Hi or Lo) in turn, and then passes that on. I don't the exact circuitry, but *something* must be doing this. A multiplexer is such a 'something.' As the book says, this is the logic-circuit equivalent of a rotary switch. A rotary switch:



By rotating the switch between the six different input lines, the output can get its signal from each one in turn. Back in the olden days, this is how you changed TV channels.

On the next page is the logic-circuit equivalent. Notice that not only is the information that is passed going to be a 'logic' signal, i.e., Hi or Lo, but the 'dial is turned' based on command logic.

Physics 310
Lecture 9b – DAC and ADC



Note that, while this doesn't change the logic of the device, for the sake of simplicity, I'm showing only one, rather than two inverters on the ABC inputs. The practical advantage of using the two may be that two inverters in a row act as a follower.

Reasoning through this, say ABC = 111, then all but the bottom AND gate has at least one 0 input, and so, regardless of what their respective D lines say, the ANDs output 0. Now, the bottom AND gate though will output 1 if D_7=1 and will output 0 if D_7=0; in short, it passes D_7's value. Now, the NOR gate is handed a whole bunch of 0's and then D_7's value. If D_7 =1, then the NOR gate passes 0, if D7=0, the NOR gate passes 1. In short, the NOR gate passes the opposite of D_7.

Similarly, if ABC = 000, then all but the top AND gate has at least one 0 input and so outputs 0 regardless of their D values. From there, similar reasoning as above tells us that the NOR gate passes the opposite of D_0.

The truth table for the whole thing is

C	B	A		Out
0	0	0	=0	$\overline{D}_0$
0	0	1	=1	$\overline{D}_1$
0	1	0	=2	$\overline{D}_2$
0	1	1	=3	$\overline{D}_3$
1	0	0	=4	$\overline{D}_4$
1	0	1	=5	$\overline{D}_5$
1	1	0	=6	$\overline{D}_6$
1	1	1	=7	$\overline{D}_7$

Here are a few practical imperfections of multiplexers.

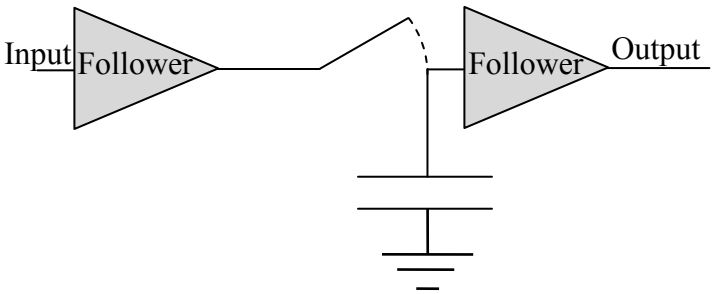
Transfer error: due to switch resistances on order of 50Ω to 2kΩ, it's good to use a follower after a multiplexer so as to minimize the effect (by having high input impedance / low output impedance).

Settling Time: As the name suggests, time for the switch to turn on/off. You can't jump between input signals more frequently than $1/ t_{settle}$.

Cross-talk: When a given switch is off, it can still effect the output. Think of it as having not-quite infinite resistance.

Sample-and-hold: Recall the "buffer" in my version of the Successive Approximation ADC. The use was that it held off on reporting the new output until it had actually figured out what that output should be. This is desirable for any ADC or DAC for that matter – you don't often want

to be confusing the circuits downstream with half-baked values. For digital outputs, you may remember that some of the Flip-Flops that we've met could be used as 'memory', that is, they can be set into modes in which they simply hold their output regardless of how the input changes. Here's another version of memory that's good for remembering either an analog or a digital value, and it's even simpler to understand, if difficulty to create in practice.



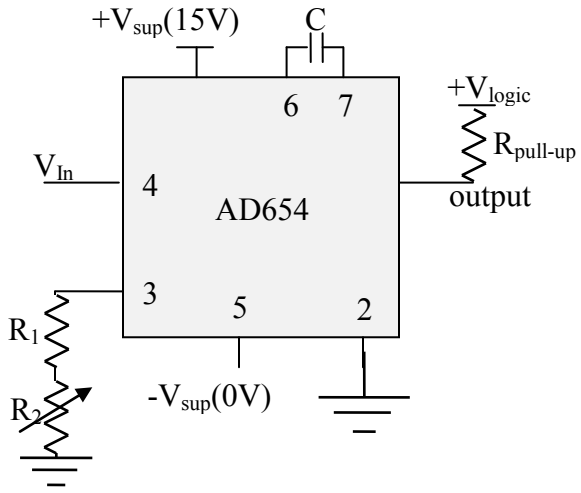
Say you have a DAC off-stage left and it sets the value of the input here. When we're ready to pass the value, the switch is thrown and so the left follower simply passes the value and so does the right follower, right on to the output. Notice that, in the process, the capacitor in the middle gets charged up to that input/output voltage. Now, when the DAC's going to calculate a new value, we break the switch. That way, while the left follower's value may fluctuate with the DAC's output, it goes no further. Meanwhile, the capacitor continues to sit there at the appropriate output value and the right follower continues to feed that value on downstream. Simple.
Now, in reality, a *real* switch isn't infinitely resistive when broken, so the left follower's output slowly chares up/down the capacitor. Similarly, a *real* follower doesn't have infinite input impedance, so the capacitor can get slowly discharged through it.

Commercially manufactured sample&hold's perform this job fairly well.

14-9 Voltage-to-Frequency and Frequency-to-voltage converters V/F & F/V

Recall that a 555 can be set up so that a capacitor charges between two frequencies that the user can set. With a given RC then, the frequency of the 555's output is determined by a user-selected voltage. So in that way, the 555 can be a 'voltage-to-frequency converter.' Of course, the 555 has many uses, and that's just one possibility, other circuits are designed specifically to have this behavior.
These have some handy applications. One is solving analog voltage transmission issues – a voltage can easily degrade over a long transmission, but a frequency wont, so generate a frequency proportional to the voltage and transmit a signal at that frequency, and then convert it back at the other end.

Another use is if you want a voltage to be represented by a pitch that is heard, or vice versa.



$$f_{out} = \frac{V_{in}}{10V\,(R_1 + R_2)\,C}$$

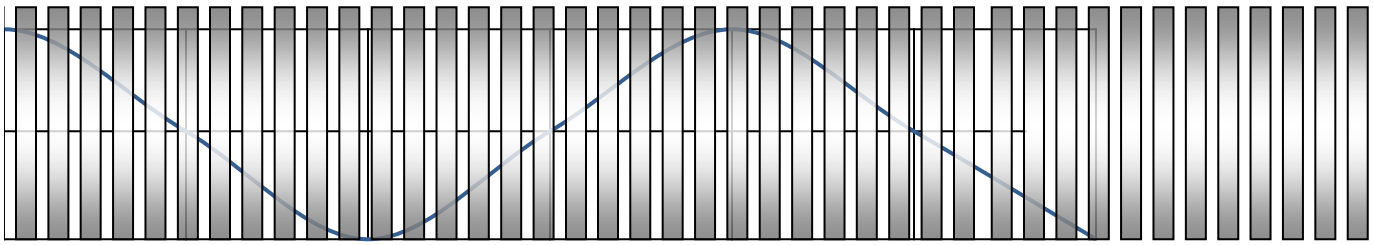
14.10 Commercial Data Acquisition Systems/ Conclusion

You probably won't actually design and build your own ADC's, DAC's,… but as the text, rather quotably, says "only the feeblest experimentalist does not understand the instrument he or she is using, particularly its limitations."

Sampling Frequency (pp 373-374)
Speaking of "limitation", here's one. One important property of an ADC, DAC, or any bit of circuitry involved in the whole conversion process is how *quickly it can process a signal.* This is particularly important if you want to process a time-varying input signal since this determines how quickly you can 'sample' that signal. The book has spoken about the frequencies at which these devices can operate. But that isn't the whole story. Say you want to pretty accurately convert a 10 kHz sine wave, then you need a device that processes 'snapshots' of the input a hundred or maybe a thousand times faster, 1MHz-10MHz, so that the information pretty smoothly varies. If, on the other hand, you're going to have some smoothing hardware process this, you might get away with sampling much less frequently. Here's the question: what is the *minimum* frequency you can sample at and still preserve the signal's frequency?

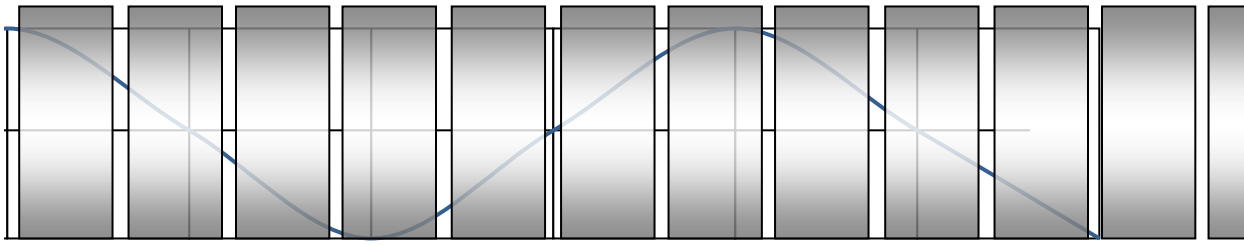
Q. How fast must you sample a time-varying signal to capture its frequency?

A. Twice the highest frequency component in the signal. The reason for the factor of two is quite simple: you need to see the signal both hi and lo. A picture makes this really easy to appreciate.

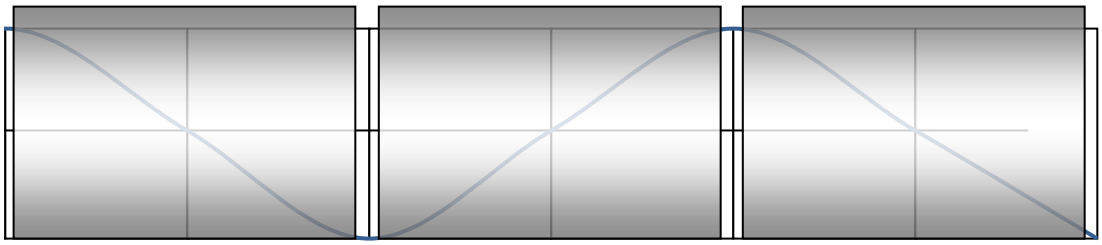


Physics 310
Lecture 9b – DAC and ADC

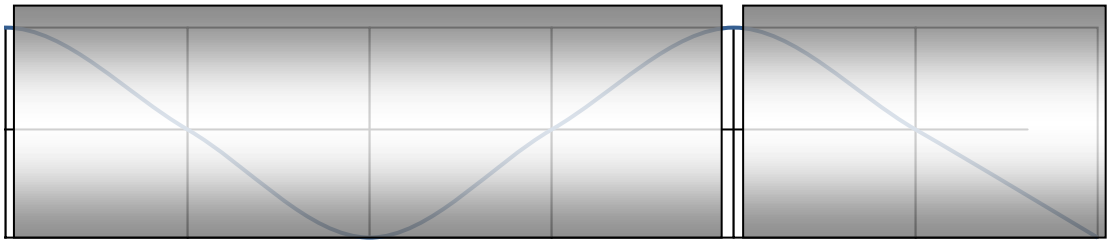
Sampled 23 times per period / $f_{sample} = 23 f_{input}$



Sampled 9 times per period / $f_{sample} = 9 f_{input}$



Sampled 2 times per period / $f_{sample} = 2 f_{input}$



Sampled 1 times per period / $f_{sample} = f_{input}$

Physics 310
Lecture 9b – DAC and ADC

Summary

DAC: never get closer to Vref than Vref*$(2^n - 1)/(2^n)$

- explain that switches are controlled by binary inputs
- cover - resistor network (Fig. 14.10), R-2R ladder (Fig. 14.13) w/ Faissler's explanation (also good for negative voltages!)
- skip - current-to-voltage (Fig. 14.11) and weighted current source (Fig.14.12)
- mention "glitching" - output dips as switches close

ADC

- Parallel Conversion ("flash" converter) - fast, but lots of comparators $(2^n - 1)$
- Successive Approximation - takes n clock pulses
- Dual-slope Conversion - conversion time varies

Nyquist Criterion - must sample at twice the bandwidth (this is essentially the maximum frequency of the input

Multiplexing (12-6) - alternately look at different inputs

For ADC and DAC's talk about their doing linear conversions, so

$$V_{out} = \frac{Votage.Range}{Number.Range} * N_{in}$$

$$V_{out} = \frac{V_{\max} - 0V}{N_{\max} - 0} * N_{in} \qquad \text{for a DAC} \quad \text{where} \quad \frac{V_{\max}}{(2^{bits} - 1)} = \frac{V_{ref}}{(2^{bits})}$$

$$V_{out} = \frac{V_{\max} - 0V}{(2^{bits} - 1) - 0} * N_{in}$$

Or the other way around for an ADC

$$N_{out} = \frac{Number.Range}{Voltage.Range} * V_{in}$$

$$V_{out} = \frac{N_{\max} - 0}{V_{\max} - 0V} * N_{in}$$

$$V_{out} = \frac{(2^{bits} - 1) - 0}{V_{\max} - 0V} * N_{in}$$