Analog-to-Digital Conversion

Prof. John McLeod

ECE3375, Winter 2022

This lesson introduces analog-to-digital conversion. Some hardware implementations of analog-to-digital converters are discussed, and we describe how to interact with these peripherals in software.

In previous courses this unit covered integrating analog-to-digital converters, but this year we won't. But — I don't like deleting stuff from my notes. So instead, those sections are marked "optional", won't be covered in class, and you don't need to learn it.

Analog-to-Digital Conversion

Most real-world information is analog. An analog signal $v_s(t)$ varies smoothly and continuously in voltage and time. This signal must be converted to digital in order to be processed by a microcontroller. For this reason, almost all microcontrollers have at least one **analog-to-digital converter** peripheral ("A/D" or "ADC") built-in.

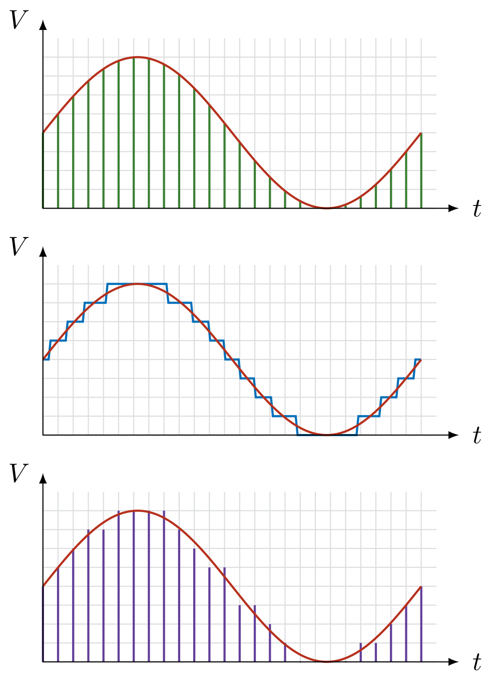

Figure 1: Various representations of an analog signal (a sine wave). Top: The signal is sampled at a given time interval. Middle: The signal is discretized into a given voltage increment. Bottom: The signal is fully digitized.

Definition: An analog signal is **sampled** if it is only measured at discrete time intervals Δt (or measured and averaged over that interval).

Definition: An analog signal is **discretized** if the voltage amplitude is rounded to some multiple of the voltage resolution ΔV.

Definition: An analog signal is **digitized** when it is both sampled in time and discretized in amplitude.

Definition: A digital signal is **binary** if only two discretization levels (equivalent to 0 and 1) are used.

Technically, most ADC are actually analog-to-binary converters (or ABC?), as they take a single analog input and produce an n-bit binary output. But whatever... the name ADC is what everyone uses. A representation of sampled, discretized, and digital signal is shown in Figure 1.

- Time sampling is usually done in multiples of the CPU clock cycle.

- Exactly how the voltage is discretized depends on the hardware implementation of the ADC, we will discuss some implementations later.

A discretized analog signal can only be understood if the **reference voltage**, **bias voltage**, and **data width** are known. A given analog voltage V can be discretized into n bits as:

$$V = \left(\frac{b_{n-1}}{2} + \frac{b_{n-2}}{4} + \cdots + \frac{b_0}{2^n}\right) V_{ref} + V_{bias},$$

where b_i is the value of digital bit i, V_{ref} is the reference voltage (or scale voltage), and V_{bias} is the bias voltage.

- Note that in this definition, the "bias voltage" refers to the *minimum voltage* in the analog signal, as the sum of the bit values is strictly positive. [1]

- Often this bias is removed before ADC, so $V_{bias} = 0\,\mathrm{V}$.

[1] Or the *maximum voltage*, if $V_{ref} < 0$.

From the above expression it should be clear that the **resolution** ΔV of the discretized signal is:

$$\Delta V = \frac{V_{ref}}{2^n}.$$

This is the smallest non-zero analog signal that can be represented in n bits. With a typical reference voltage of $V_{ref} = 5\,\mathrm{V}$, discretizing a signal into $n = 16$ bits gives $\Delta V = 76.3\,\mu\mathrm{V}$, which is often well below the noise threshold for a typical circuit. Consequently, ADC is rarely done with more than $n = 16$ bits of width. [2]

[2] Even that is a bit extreme — if $n = 8$ then $\Delta V = 19.5\,\mathrm{mV}$ which is already pretty small for most circuits.

Digital-to-Analog Conversion

The first circuit we will consider is an ADC turned on its head: a *digital-to-analog converter* (DAC). This actually takes a n-bit binary input and produces a digital output (so a BDC?), but in the microcontroller world anything that is not binary is considered analog.

How does one convert a n-bit binary number into a single analog signal?

- Contrary to what some of your classmates think, simply connecting two or more wires together *does not* add the voltages together.
- However currents *do* add together.
- If bit m with value b_m can be converted into a current proportional to $2^m b_m$, then the sum of all these currents is representative of the signal.

The best way to switch between voltage and current is to use a resistor.
Consider the resistor network shown in Figure 2. The mth input is V_{ref} if the corresponding bit $b_m = 1$, and 0 if $b_m = 0$. Applying

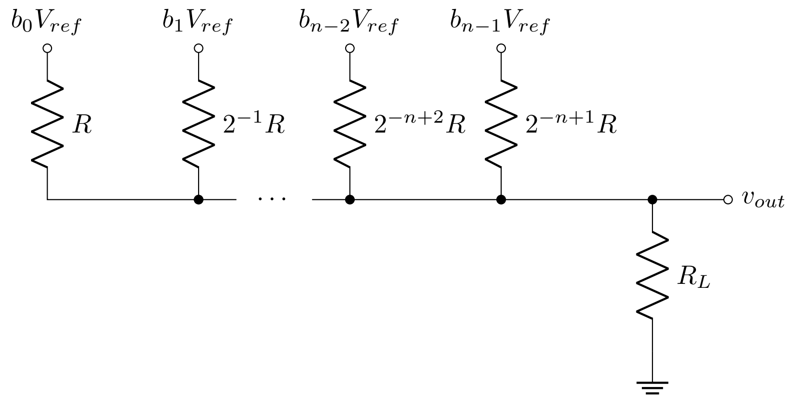

Figure 2: A resistor network for converting a n-bit binary value into a voltage v_{out}.

Kirchoff's current law at the node v_{out}, we have:

$$\begin{aligned}
\frac{v_{out}}{R_L} &= \sum_{m=0}^{n-1} \frac{2^m}{R}\left(b_m V_{ref} - v_{out}\right) \\
\left(\frac{1}{R_L} + \frac{1}{R}\sum_{m=0}^{n-1} 2^m\right) v_{out} &= \frac{V_{ref}}{R}\sum_{m=0}^{n-1} 2^m b_m \\
v_{out} &= \left(\frac{R}{R_L} + 2^n - 1\right)^{-1} V_{ref} \sum_{m=0}^{n-1} 2^m b_m
\end{aligned}$$

This is kind of a mess, but it works: the output voltage v_{out} is proportional to the binary number $b_{n-1}b_n \cdots b_1 b_0$. This isn't a practical DAC, however, because the output voltage scales with the load resistance. Ideally we want the output voltage to depend only on the binary value, not on whatever load peripheral we have connected to the DAC.

The simplest practical n-bit DAC uses an operational amplifier (op amp) to sum a set of n binary signals and act as a buffer to keep the output voltage stable, independent of any load. The signal from bit m resistor $2^{-m}R$ — as the resistance decreases with increasing bit, the current is that much larger. Those of you who paid close attention in your electronics classes will recognize this as a particular implementation of a *weighted inverting summer*. [3]

[3] The word "summer" means a "circuit that performs summations," not the season.

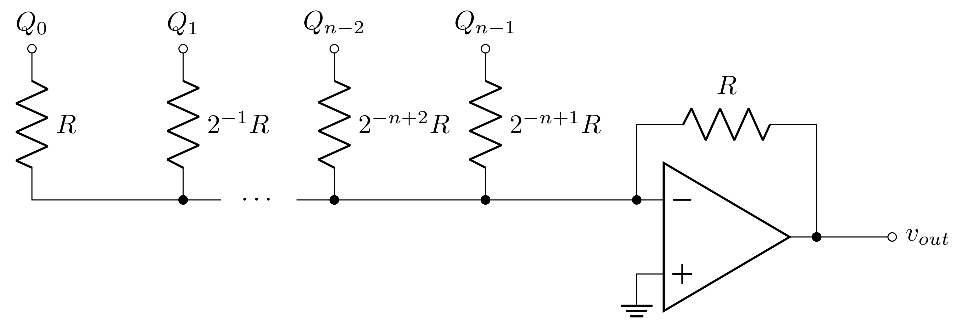

Figure 3: An n-bit weighted-resistor circuit for DAC. The digital signal $Q[n-1,...,0]$ is converted to the analog output v_{out}.

Analyzing this circuit is straightforward, assuming you remember the principle of superposition.

- Because of the operating principles of the op amp, the $-$'ve terminal is at essentially zero potential — as it is tied to the $+$'ve terminal by a virtual short circuit.
- The current that flows from signal $Q_m = 1$ is $i_m = 2^m v_{ref}/R$.
- Current cannot flow into the op amp, so all of the currents add and flow across the feedback resistor.

The output is therefore:

$$\begin{aligned} v_{out} &= -Ri_{tot} \\ &= -Rv_{ref} \sum_{m=0}^{m=n-1} \left(\frac{2^m Q_m}{R} \right) \\ &= -v_{ref} \sum_{m=0}^{m=n-1} 2^m Q_m \end{aligned}$$

Note that this is an inverting circuit, this is typically fixed by using a negative reference voltage v_{ref}.
The weighted-resistor DAC circuit above can sometimes be problematic because so many different resistances are needed. A simpler design uses only standard resistances R, $2R$, and one resistance $3R$. [4] This circuit is typically based on an op amp, as shown in Figure 4.

[4] The $2R$ and $3R$ resistances of course can be made from three R in series, of course.

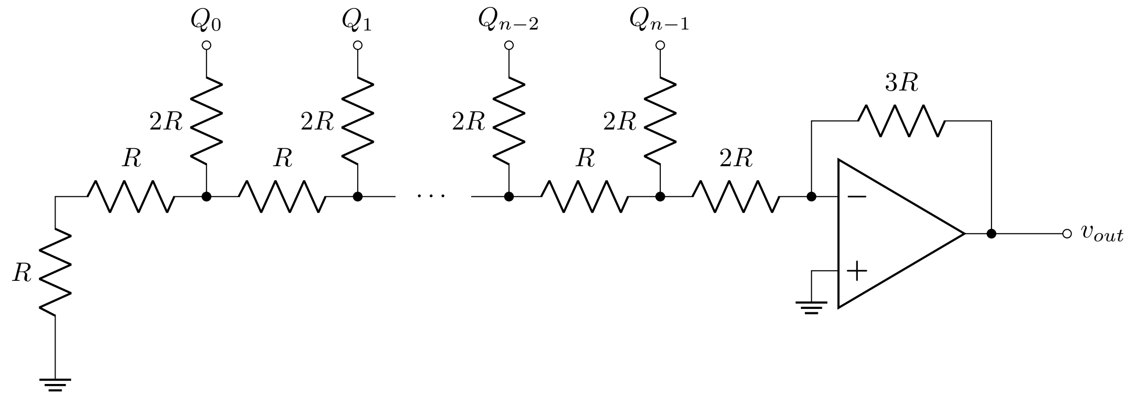

Figure 4: An n-bit ladder converter circuit for DAC. The digital signal $Q[n-1, ..., 0]$ is converted to the analog output v_{out}.

Again, analyzing this circuit is straightforward.

- If only one input Q_m is considered, and all other inputs are set to ground, the equivalent circuit for that input is shown in Figure 5. This occurs because the resistances were carefully chosen: Note that $2R \parallel 2R = R$.

- The equivalent resistance seen by Q_m is $2R + 2R \parallel 2R = 3R$, so the current is:

$$i_m = \left(\frac{Q_m}{3R}\right) V_{ref},$$

  where V_{ref} is the voltage of binary signal 1.

- The equivalent circuit for input Q_m also makes it clear that any current which flows from signal Q_m gets split in half at the node.

- The half current that flows right towards the op amp will further be split in half each time it reaches a node for Q_p (where $m < p < n - 1$). The current from signal Q_m that reaches the $-$'ve terminal of the op amp is therefore:

$$i_{m,-} = \left(\frac{Q_m}{2^{n-m}}\right) \frac{V_{ref}}{3R}.$$

- By Kirchoff's current law, the total current reaching the op amp is just the sum of all the signals:

$$i_{tot,-} = \frac{V_{ref}}{3R} \sum_{m=0}^{m=n-1} \frac{Q_m}{2^{n-m}}$$

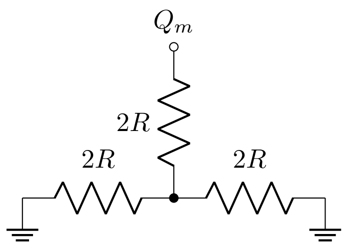

Figure 5: Equivalent circuit by the principle of superposition for any input Q_m in the ladder converter from Figure 4, when all other signal sources are replaced with grounds.

- This current can't enter the op amp, as the impedance at the input is almost infinite. Instead it must flow across the $3R$ resistor. The output voltage is therefore:

$$\begin{aligned} v_{out} = -3Ri_{tot,-} &= -V_{ref} \sum_{m=0}^{m=n-1} \frac{Q_m}{2^{n-m}} \\ &= -V_{ref} \left(\frac{Q_0}{2^n} + \frac{Q_1}{2^{n-1}} + ... + \frac{Q_{n-2}}{4} + \frac{Q_{n-1}}{2} \right). \end{aligned}$$

This is a digital signal, as it has 2^n discrete voltage levels, and can only change in value when the inputs Q_m change — and they are controlled by the system clock. However for a sufficiently fast clock (as is usually the case) and reasonably large n (as mentioned above, $n = 8$ is usually plenty), this signal is often indistinguishable from a true analog signal.

Successive-Approximation Converters

A successive-approximation converter is a robust circuit that does not require precision components, and that has a fixed conversion time regardless of signal magnitude. [5] A schematic of this device is shown in Figure 7. This circuit implements a *binary search* for the correct value that digitizes $v_s(t)$, by comparing the current estimate of that value $(Q_7, Q_6, ..., Q_0)$ to $v_s(t)$ using a DAC.

[5] Actually this device is usually called a **successive-approximation register**, or SAR.

- The first stage of this circuit allows the signal $v_s(t)$ to charge a capacitor. This is the "sampling and hold amplifier". A timer control signal (`tmr` in Figure 7) is used to turn on and off the transistor switch.

- From circuits class, you hopefully recognize that the voltage across capacitor C_{ADC} is:

$$v_{ADC} = v_s \left[1 - \exp\left(-\frac{t}{R_{ADC}C_{ADC}}\right)\right],$$

  as long as the signal voltage $v_s(t)$ is fairly constant over the sampling time.

- As $t \to \infty$, $v_{ADC} \to v_s$, so a sample time is selected that is large enough such that v_{ADC} settles within the resolution limit of the signal voltage.

- The next stage of this circuit is an open-loop op amp comparator. Because there is no feedback loop, the output saturates to

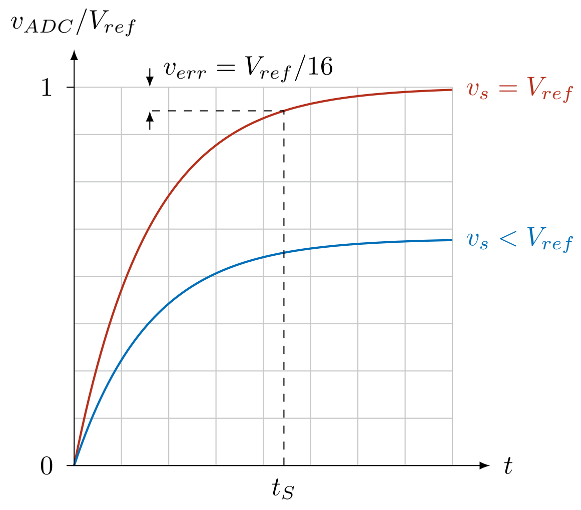

Figure 6: Example of using the sampling and hold amplifier from a 3-bit SAR ADC. If the signal is sampled for a time t_s, then the error in the maximum sample ($v_s = V_{ref}$) is half the resolution. Any signal with lower voltage ($v_s < V_{ref}$) sampled for this same time t_s will therefore have an even smaller error.

V_{ref} or 0 depending on whether the positive or negative input is larger.

- Therefore, if $v_{ADC} > v_A$, the output saturates to ground. If $v_{ADC} \leq v_A$, the output saturates to V_{ref}.

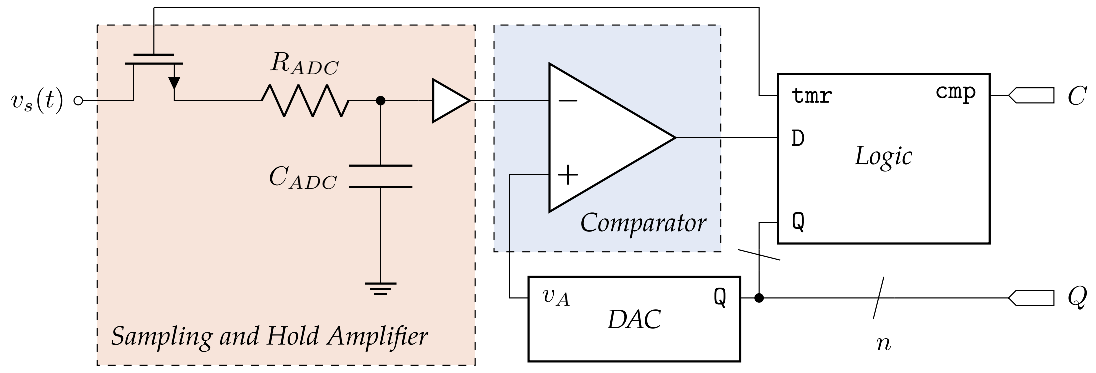

Figure 7: A rough schematic of a n-bit successive-approximation ADC.

The "logic" part of the circuit is more complicated than a simple counter (and that is why I drew it as a block), it is responsible for performing the binary search. The logic circuit processes the output bit-by-bit, starting with the MSb and working down to the LSb. It starts with $Q_n, Q_{n-1}, ..., Q_0 =$ `0b00...0`. It then iterates through this algorithm for all bits m in $n > m \geq 0$.

- Assume bit m is high, so the value is $Q_nQ_{n-1}...Q_{m+1}10...0$.
- Check the output from the comparator.
- If this guess is $\leq v_s(t)$, keep bit m high. Otherwise, flip bit m back to low.
- Repeat the process for bit $m-1$.

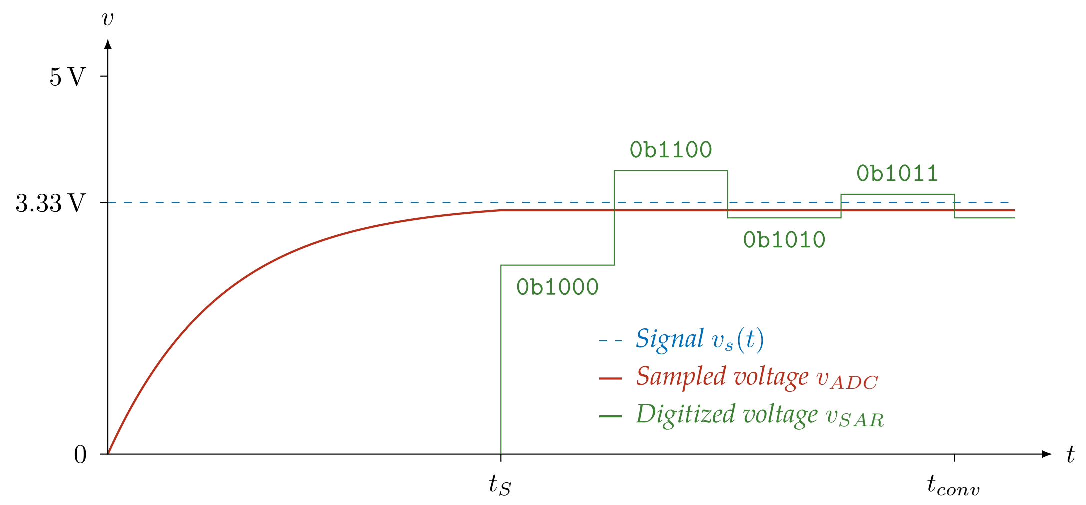

Figure 8: Visual representation of SAR conversion process. First the signal is sampled for a time t_S until the sample voltage is within error of the signal. Then the signal is cut out of the circuit, so any changes in the signal do not affect the remaining part of the conversion. A binary search is initiated to find the signal. This search always takes n iterations for a n-bit conversion. The entire process takes a time t_{conv}.

An example of this search for a 4-bit successive approximation ADC is shown in Table 1. Starting with the MSb, each bit is successively tested (shown in red). If the resulting voltage exceeds the analog signal, that bit is cleared back to 0, otherwise it is retained as 1.

- The successive approximation ADC will always take n iterations for n-bit conversion, regardless of the magnitude of the input signal.

Guess	Voltage	Result
`0b1000`	2.5 V	$v_A < v_s$
`0b1100`	3.75 V	$v_A > v_s$
`0b1010`	3.125 V	$v_A < v_s$
`0b1011`	3.3475 V	$v_A > v_s$

Table 1: Example of using a binary search to find the 4-bit binary equivalent of $v_s = 3.33\,\text{V}$ when $V_{ref} = 5\,\text{V}$ The final result is `0b1010`.

Successive approximation ADCs are quite common, as the constant conversion time makes it easier to synchronize analog signal processing with other processes in the microcontroller.

Sampling Errors and Sampling Frequency

What is the error between the true analog signal and the digital conversion?

- At best, the maximum error is half the voltage resolution:

$$\Delta v_s = \pm\frac{\Delta V}{2} = \pm\frac{V_{ref}}{2^{n+1}}$$

- This occurs when the digitized signal is a true representation of the analog signal.

Unfortunately, when the analog signal is unknown, the error can be considerably larger. This is because for any given sine wave, there are other sine waves with higher frequencies that provides the same digital signal when sampled at the same fixed interval.

Definition: The **Nyquist frequency** f_S is twice the highest frequency component of the signal.

$$f_S = 2f_{max}$$

As long as a signal is sampled at the **Nyquist frequency** or higher, the resulting digitized signal is a true representation of the analog signal. This is the **Nyquist-Shannon Sampling Theorem**, visualized in Figure 10.

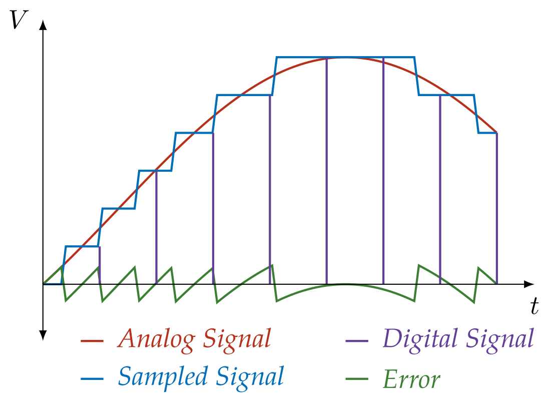

Figure 9: The error is the difference between the true analog signal and the digital signal. Depending on how the sampling time aligns with the signal, the error falls within ± half the voltage resolution.

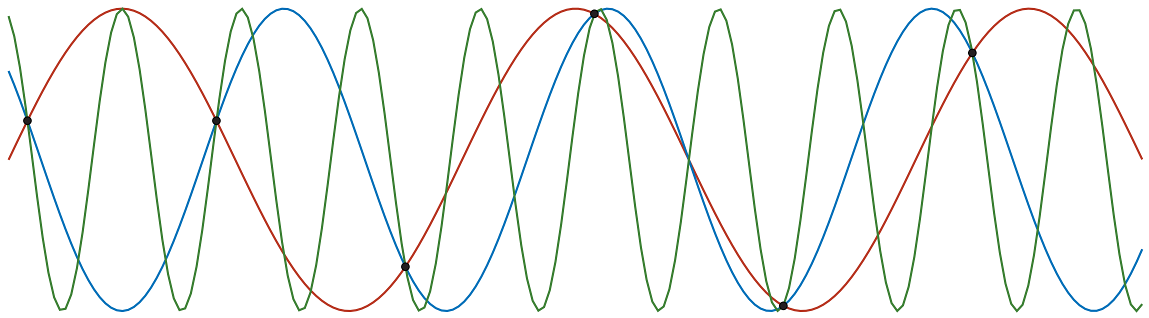

Figure 10: Visualization of the Nyquist-Shannon Sampling Theorem. The sampled data (dots) match all three different sine waves — without further information there is no way to tell which of these is the true signal.

The **Nyquist frequency** required for sampling is a major factor in deciding which type of ADC to use. Obviously a microcontroller-driven ADC cannot sample faster than the clock speed of the CPU, but most ADC cannot operate anywhere near that fast.

- Sigma-Delta $(\Sigma - \Delta)$ ADC is a very popular piece of hardware for microcontrollers. These have an error-correction loop, and are very accurate. [6] They also tend to be relatively slow, often slower than $10\,\mathrm{kHz}$. The trade-off is that they are high resolution (even up to 24 bits).

- SAR ADC, as discussed here, is the other main type of ADC used in microcontrollers. These tend to be faster, operating in the $100\,\mathrm{kHz}$ to $1\,\mathrm{MHz}$ range. The trade-off is that they are lower resolution.

- Flash ADC is the fastest, and can operate up to the clock speed of the microcontroller, so often approximately $1\,\mathrm{GHz}$. These are very specialized circuits, and require 2^n stages for a n-bit conversion. Consequently they are probitively expensive and power-hungry even at 8 bits. [7]

[6] They are based on dual-slope integrating ADCs, you can read the optional sections at the end of this note if you are interested.

[7] You can read the optional section at the end of this note if you are interested in this kind of ADC.

- Finally, pipeline ADC combines a low-resolution flash ADC to subdivide the signal, then a conventional SAR ADC stage refines the conversion to an acceptable resolution. By passing the signal through a coarse and fast flash ADC stage, the SAR ADC requires fewer steps in the binary search to converge to the correct value. This trade-off allows the device to operate around $100\,\mathrm{MHz}$ while still having 12 or 16 bits of resolution.

ADC on the DE1-SoC/DE10-Standard

The DE1-SoC/DE10-Standard development board has an ADC peripheral. This is an 8-channel, 12-bit ADC which uses the successive-approximation method. [8]

[8] The peripheral is a commercially-available LTC2308 IC, if you want full technical details of the ADC look up the datasheet for that chip. (Just Googling "LTC2308" should find it.)

- The ADC is memory mapped to the base address `0xFF204000`.
- The **data register** for each of the eight channels is mapped to a sequential word, starting with the base address for channel 0.
- As the ADC is 12-bit, only the 12 LSbs for each register are used for data.
- The register for channel 0 does double duty: it also acts as a **control register** for the ADC. Writing a value — *any* value — to this register will cause *all* channels to update. This means all channels will start converting whatever input is present to a digital value.
- The register for channel 1 also does double duty: it also acts as a **control register** for the ADC. Writing a 1 to this register will cause *all* channels to auto-update.
- For all channels, bit 15 is used as a "status bit" to indicate when the conversion is complete. This bit gets set to 1 when the conversion for that channel is complete, and it gets cleared to 0 after that channel is read by the CPU.

The structure of the ADC is shown schematically in Figure 11.

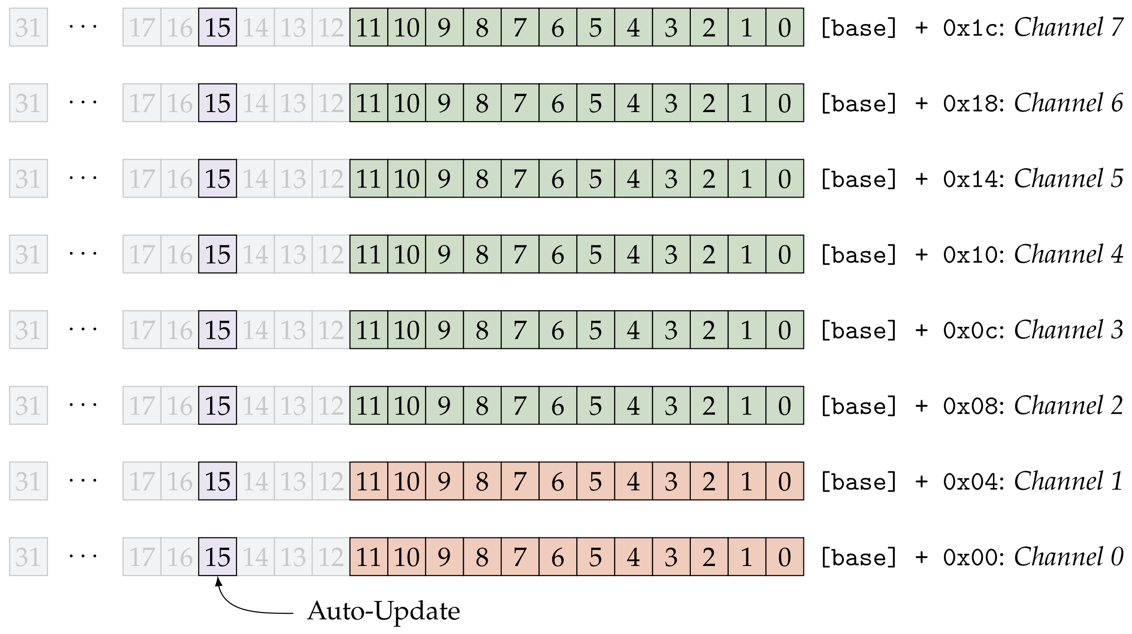

Figure 11: Structure of the LTC2308 ADC chip on the DE1-SoC board. The base address is `0xFF204000`. The registers for channels 0 and 1 do double duty as data and control registers. Bit 15 for all channels acts as a "status bit", informing whether the conversion is complete for that channel.

Interacting with the ADC peripheral in Assembly is similar to using the *interval timers*: write the contents of whatever register is handy to channel 0 or 1 to set the operating mode, then use `ldr` with the appropriate address to read from whichever channel is set up.

Example: If we want to read a single value from channels 0 and 3, the following assembly code will do the trick.

```
ldr r0, =0xff204000 @ ADC base

str r1, [r0] @ write to ch0 to update ADC
/* do some things */
ldr r1, [r0]      @ read ch0
ldr r1, [r0,#12] @ read ch3
```

Example: The following Assembly code reads a continuous stream of data from channel 4 and stores it to memory.

```
ldr r0, =0xff204000 @ ADC base
ldr r1, adr_array_base @ where to store data

mov r4, #1
str r4, [r0,#4] @ set ADC to auto-update

lsl r4, #15  @ bit mask for bit 15

adc_loop:
  ldr r2, [r0,#16] @ read ch4
```

```
and r3, r2, r4 @ check bit 15
cmp r3, r4
bne adc_loop @ conversion not done yet

sub r2, r4 @ remove bit 15 from data
str r2, [r1], #4 @ save data to memory

/* some other code to decide when
  enough data has been read */
b adc_loop
```

On the physical DE10-Standard board, there is a standard 10-pin connector for the ADC (8 channels, V_{ref}, and ground) so some external analog device can be connected. Obviously this cannot be done in the simulator.

- The simulator simply increments each channel in the ADC each time they are updated.

Consequently, the ADC is of limited value in simulator-only labs. [9]

[9] **Important Note:** It seems there might be a bug in the simulator. The code above does *not* work as intended, because the simulator uses **bit 16** (not bit 15) as the auto-update flag. However the actual DE10-Standard hardware (and the manual for the DE1-SoC) suggest that bit 15 should be the status flag.So if you want the code to work in the simulator, use a shift of 16 bits for the bitmask (or a mask of `0x0000 8000`). But for the code to work in hardware, use a shift of 15 bits (or a mask of `0x0001 0000`).

Model ADC Peripheral

For sample problems that we may ask to you solve "by hand", we will use a simple model ADC rather than the somewhat clumsy ADC structure on the DE1-SoC. This is similar to the model timer discussed last lesson.

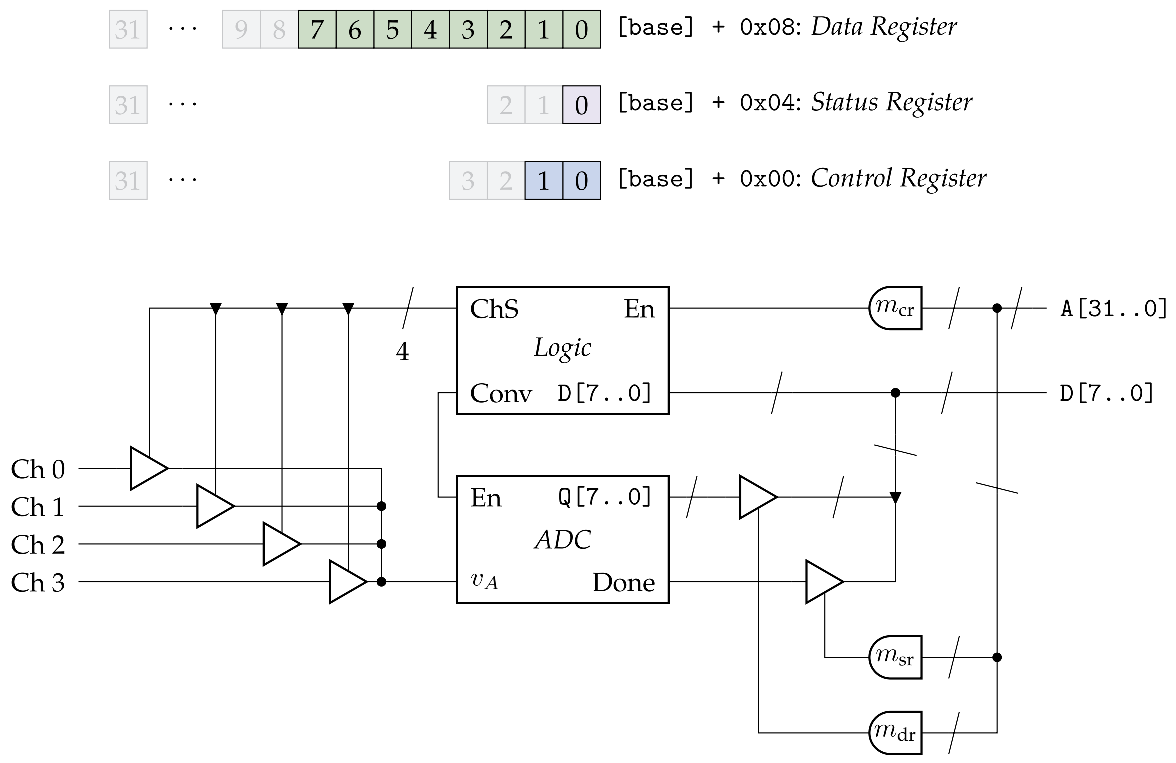

Figure 12: Memory-mapped structure and simplified circuit diagram of the model ADC that may be used for problems in this course.

Our model ADC will be 4-channel and 8-bit, with the following memory-mapped registers:

- A **control register** at the ADC base address. Writing a value between 0 and 3 to this register tells the ADC to start sampling the corresponding channel. Note that it is the "act of writing" that starts the ADC — at the start, this register may be filled with zeros, but if you want to read from channel 0 you still need to write `0x00` to this register.

- A **status register** at the ADC base address offset by `#4`. The LSb in this register is normally 1, when it is cleared to 0 it means the conversion has finished.

- A **data register** at the ADC base address offset by `#8`. This holds the 8-bit converted value from then channel selected by the **control register**.

Note that this ADC structure only allows one of the four channels to be sampled at a time, as there is only one data register to hold the result.

Optional: Single-Slope Integrating ADCs

Converting an analog signal into a binary sequence is slightly more complicated than converting binary to analog. One method for ADC is a **single-slope integrator**, as shown in Figure 13.

- The first stage of this circuit is an *inverting integrator*, where charge accumulates on the capacitor C.
- The input "signal" for this integrator is just the negative of the reference voltage V_{ref},[10] so the output of this integrator is a straight line sloping up from 0 to V_{ref} as a function of time.

[10] Because op amp integrators are always inverting, using $-V_{ref}$ provides a positive output.

$$v_{int}(t) = \left(\frac{V_{ref}}{RC}\right) t$$

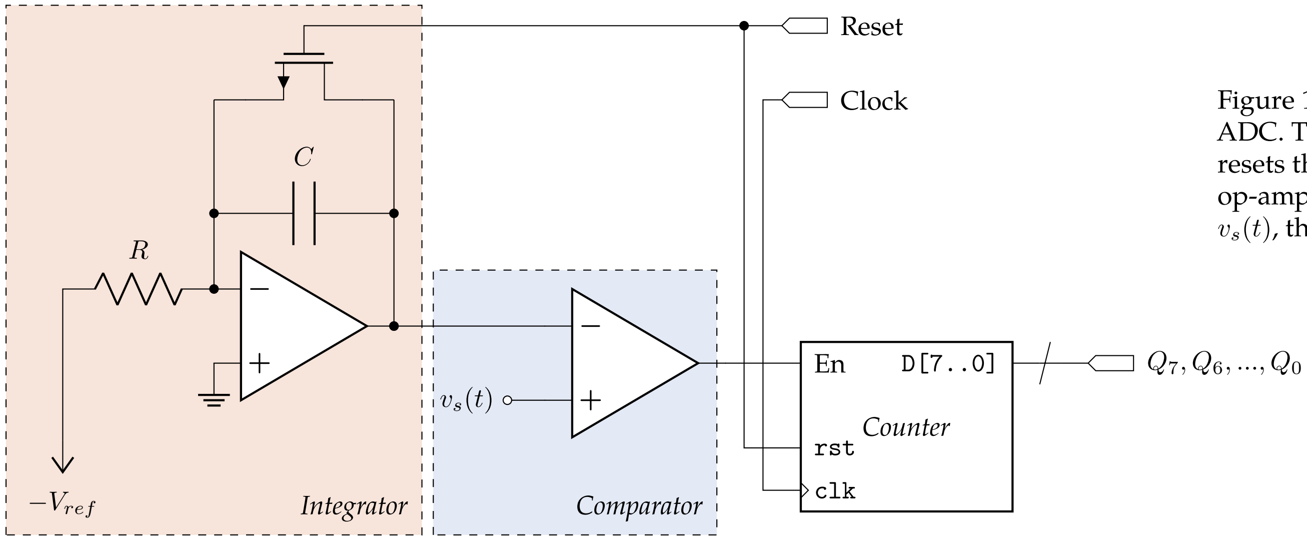

Figure 13: An 8-bit single-slope integrator for ADC. The reset signal from the microcontroller resets the counter and shorts the capacitor in the op-amp integrator circuit. The analog input is $v_s(t)$, the digital output is $Q_7, Q_6, ..., Q_0$.

- The integrator output goes into a second op amp. The signal voltage $v_s(t)$ is used as the input to the $+$'ve terminal. This op amp has no *feedback loop*, so the output is always **saturated** to the limits set by the op amp power supplies — usually V_{ref} and ground.

- Because an op amp is a *differential amplifier*, whenever $v_s(t) - v_{int}(t) > 0$, the comparator output will saturate to V_{ref}. Whenever $v_s(t) - v_{int}(t) < 0$, the comparator output will saturate to ground.

- The comparator output is used as the enable for a n-bit counter, it will stop counting as soon as $v_{int}(t) > v_s(t)$.

- The value after the count has stopped is used as binary equivalent of the analog signal at the given time interval.

- To read the next time interval, the CPU sends a reset signal to clear the capacitor charge and reset the counter.

It is important to recognize that the entire process described above must occur to read the signal for a *single* time interval: the operation of an integrating ADC is necessarily significantly slower than the system clock. This is usually not a huge problem, but is something to be aware of.
The operation of a single-slope integrator is shown in Figure 14. Two example input signal voltages are given as v_1 and v_2.

- At $t = 0$, $v_{int}(t)$ is less than the signal, so the counter is enabled.

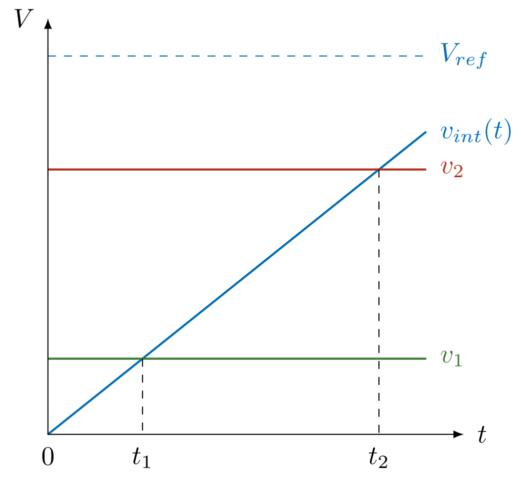

Figure 14: Example of the operation of a single-slope integrator with two different signal voltages (v_1 and v_2) given. The integrator output $v_{int}(t)$ cannot exceed V_{ref}.

- The counter stops when $v_{int}(t) > v_s(t)$, so the time spent counting (t_1 and t_2) is proportional to the magnitude of the signal.
- If the single-slope integrator circuit was designed appropriately such that $RC = 2^n\tau$, where τ is the clock frequency for the counter, then the counter stops at a time:

$$v_1 = v_{int}(t_1) = \left(\frac{V_{ref}}{RC}\right) t_1 = \frac{V_{ref}}{2^n}\left(\frac{t_1}{\tau}\right)$$
$$t_1 = 2^n\tau\left(\frac{v_1}{V_{ref}}\right),$$

- This means the final value on the counter is:

$$D = 2^n\left(\frac{v_1}{V_{ref}}\right).$$

This circuit works in principle, but it suffers from two flaws:

1. The conversion time depends on the magnitude of the signal voltage (the smaller input v_1 is converted in a time t_1 that is shorter than the time t_2 for the larger input v_2).
2. If RC is not exactly equal to $2^n\tau$ (which is likely) then conversion errors start creeping in.

The first flaw is endemic to integrating ADCs, but the second flaw can be addressed using a **dual-slope integrator**.

Optional: Dual-Slope Integrating ADCs

A dual-slope integrator switches between the reference voltage and the signal for the integration input. An example of a dual-slope integrator is shown in Figure 15. In this circuit, a switch control bit is used to control whether the signal $v_s(t)$ or the constant $-V_{ref}$ is fed into the integrator.

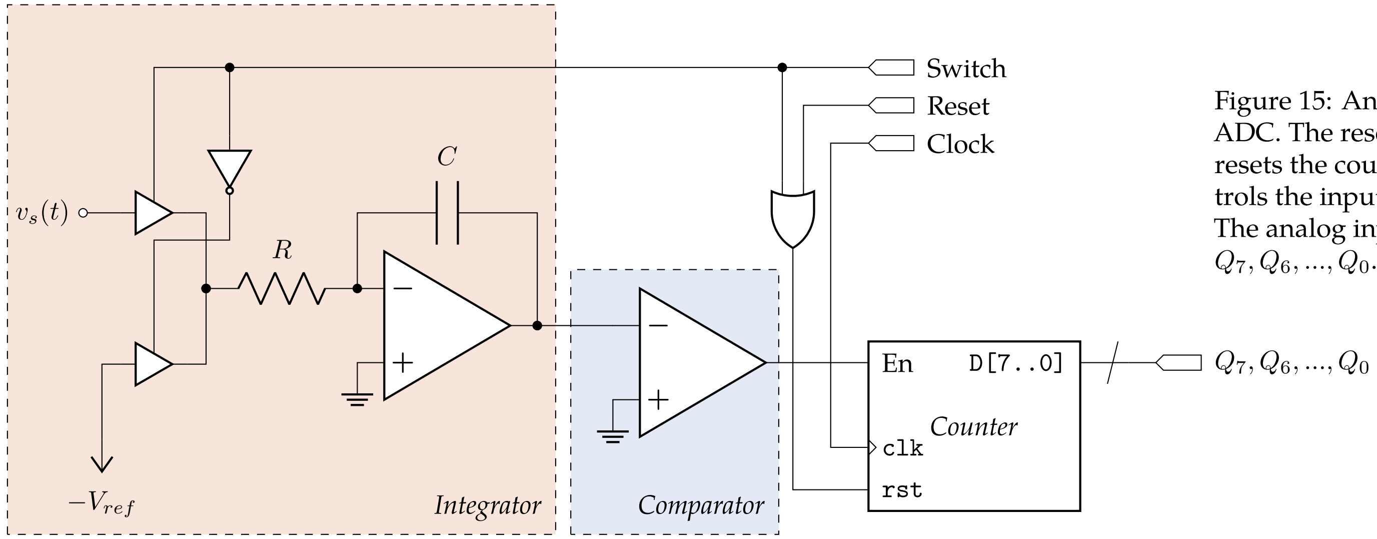

Figure 15: An 8-bt dual-slope integrator for ADC. The reset signal from the microcontroller resets the counter, while the switch signal controls the input to the op-amp integrator circuit. The analog input is $v_s(t)$, the digital output is $Q_7, Q_6, ..., Q_0$.

- At the start of the conversion, the switch is set to use $v_s(t)$ as the input. The switch remains in this position for a known interval $\Delta t = K\tau$, where τ is the counter clock pulse and K is some integer.
- Since the integrator is inverting, this drives the integrator

output to the negative voltage:

$$v_{int}(t = K\tau) = -\left(\frac{v_s}{RC}\right) K\tau$$

- The comparator +'ve input is grounded. As $0 - v_{int}(t) > 0$ when the integrator output is negative, the comparator output saturates to V_{ref} and the counter is enabled. However the counter doesn't actually start counting, as the reset signal is held.
- After an interval of $K\tau$, the switch is flipped and $-V_{ref}$ is used as the input. This drives $v_{int}(t)$ linearly back up to zero. Flipping the switch also turns the reset to the counter off. As the counter enable is still set, the counter starts counting.
- After an *unknown* time interval $\Delta t'$, the integrator output goes slightly above zero: $v_{int}(t = K\tau + \Delta t') > 0$. At this point the $-$'ve input to the comparator exceeds ground, so the comparator output saturates to 0 and the counter stops.
- This unknown interval can be expressed in clock pulses: $\Delta t' = L\tau$. We therefore have:

$$v_{int}(t = K\tau + L\tau) = 0 = -\left(\frac{v_s}{RC}\right) K\tau + \left(\frac{V_{ref}}{RC}\right) L\tau$$

$$\frac{v_s}{V_{ref}} = \frac{L}{K}$$

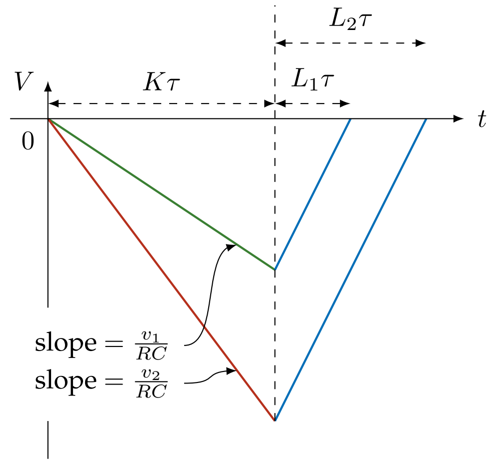

Figure 16: Example of the operation of a dual-slope integrator with two different signal voltages (v_1 and v_2) given, where $V_{ref} > v_2 > v_1$.

If we chose $K = 2^n$, then the value on the counter is:

$$D \approx 2^{\log_2 L}.$$

This circuit still has the problem that the conversion time depends on the magnitude of the signal (which is proportional to the unknown constant L), but the dependence on RC has been removed. These **dual-slope integrators** are not often used as-is for ADC, but rather as the first stage of a **sigma-delta** ADC. Here an additional logic stage controls the switch and reset inputs, and reads the counter output D. This additional logic stage does further comparisons between the input signal and the output to reduce errors. [11]

[11] The dual-slope integrator as presented has a tendency to always "round down" the converted signal.

Optional: Flash ADC

It is important to stress that the analog signal is a function of time: $v_s(t)$, but all of the ADC methods discussed above treat it as constant over the sampling interval.

- ADC needs to be performed for each sample time in the signal.

This is not a problem if the analog signal varies slowly. However, one should remember that the *ADC clock speed* is not the same as the *ADC conversion time* — rather n-bit conversion takes up to 2^n clock cycles. [12]

[12] Always 2^n iterations for successive approximation, for the integrators 2^n is the worst-case.

- You can *never* sample analog signals frequencies that are faster than the clock speed.
- However as accurate ADC usually requires at least $n = 8$ bits, it is quite possible to have signals of interest that have key frequencies that are between $(2^n\tau^{-1})$ and τ^{-1}.

An easy method for rapid ADC is called **flash ADC**. [13]
The operating principle of a flash ADC is straightforward.

[13] This has nothing to do with Flash memory!

- A voltage divider ladder composed of resistances R and $2R$ is used between V_{ref} and ground to obtain all the intermediate voltages for digitization.
- Each of these intermediate voltages is compared with the signal $v_s(t)$. The output of the comparators is V_{ref} if the intermediate voltage is larger than $v_s(t)$, otherwise it is ground.

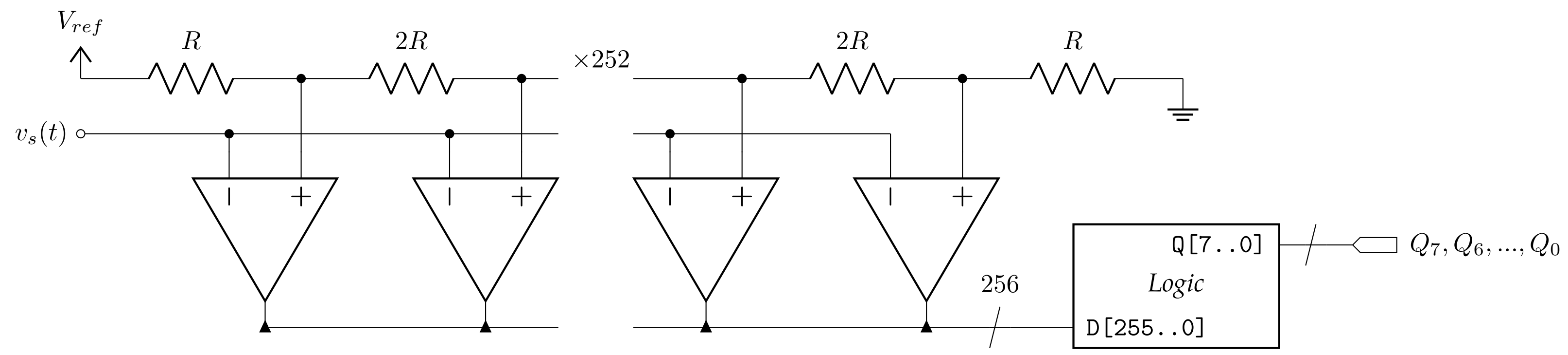

Figure 17: An 8-bit flash ADC. The input signal is compared to a voltage divider ladder with $2^8 = 256$ stages. A combinational logic circuit encodes these 256 inputs into an 8-bit sequence.

- A combinational logic circuit combines the binary output from the comparators into a binary sequence.

Flash ADC does not integrate the signal, and it processes all bits in parallel rather than sequentially, so it is extremely fast. However it is also extremely large: 2^n comparator stages are required for a n-bit ADC. This takes up a lot of physical chip space, and also uses a significant amount of power. Flash ADCs beyond 8-bits are prohibitively expensive, and even 8-bit flash ADCs are too large and power-hungry for many microcontrollers. Flash ADCs are typically only used in specialist applications that require high-speed ADC.