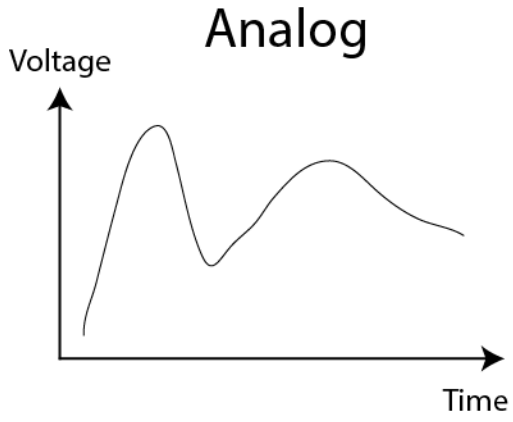
CC2511 Week 8

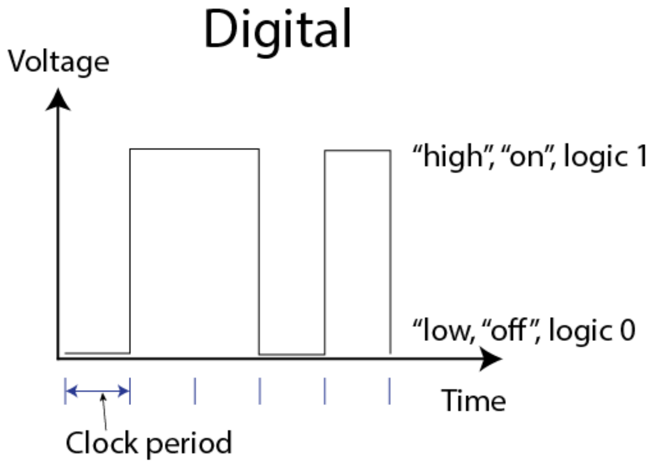
Today: Analog to Digital Conversion

- Sensors
- Analog signals
- Operation of an ADC
- Design tradeoffs and implementation
- Using the ADC in Processor Expert

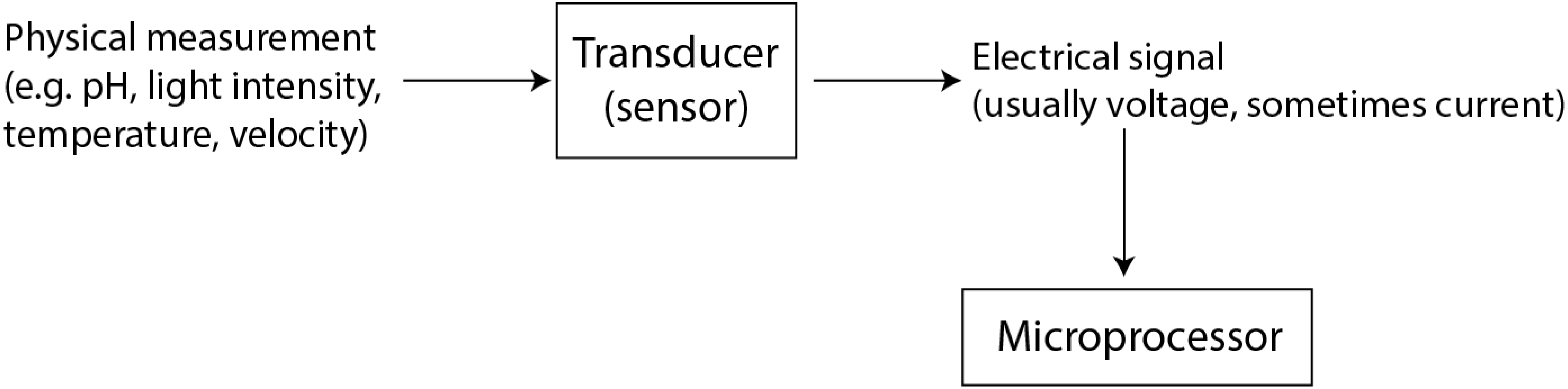
Analog signals vs digital signals

- Up until now we have only used digital inputs such as whether a switch is open or closed.
- Analog inputs allow us to read arbitrary voltages (instead of just logic low vs logic high).
- Examples of continuously varying quantities that might be measured: temperature, pressure, mass, position, velocity, light intensity, pH, …





Analog measurement



Sensors

- The transfer function of a sensor is the relationship between the signal of interest and the voltage.
- It is documented in the sensor's datasheet.
- We want to know a quantity x but the sensor measures
$$\mathrm{y} = V(x).$$
- The quantity of interest is calculated as:
$$x = V^{-1}(y)$$

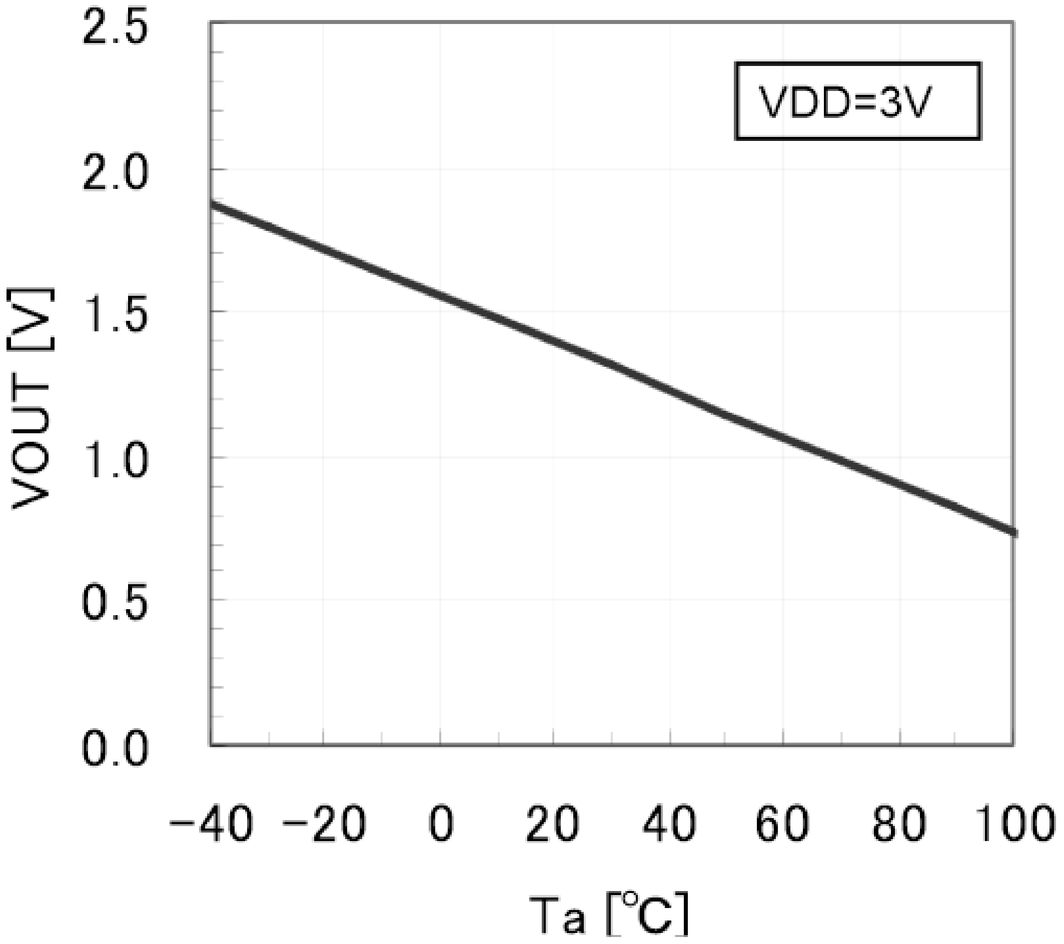
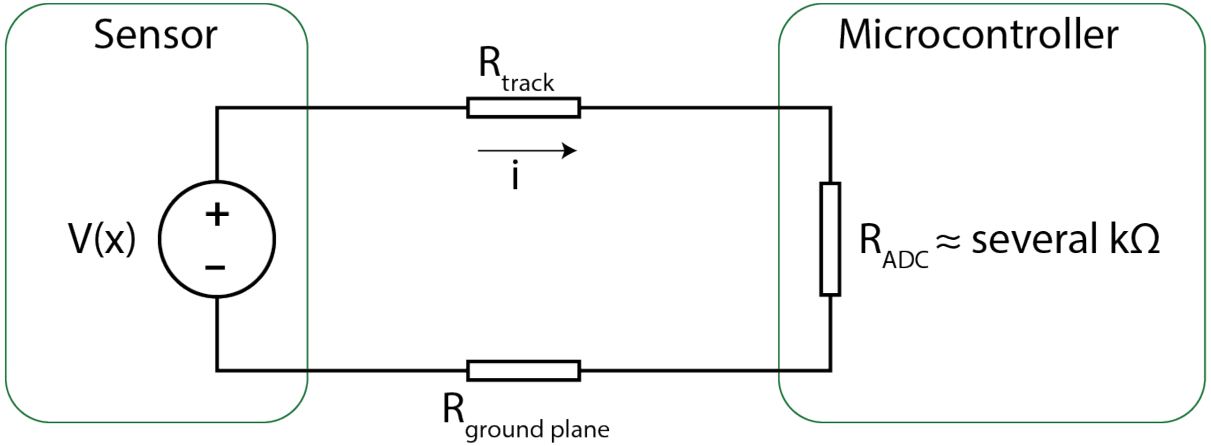
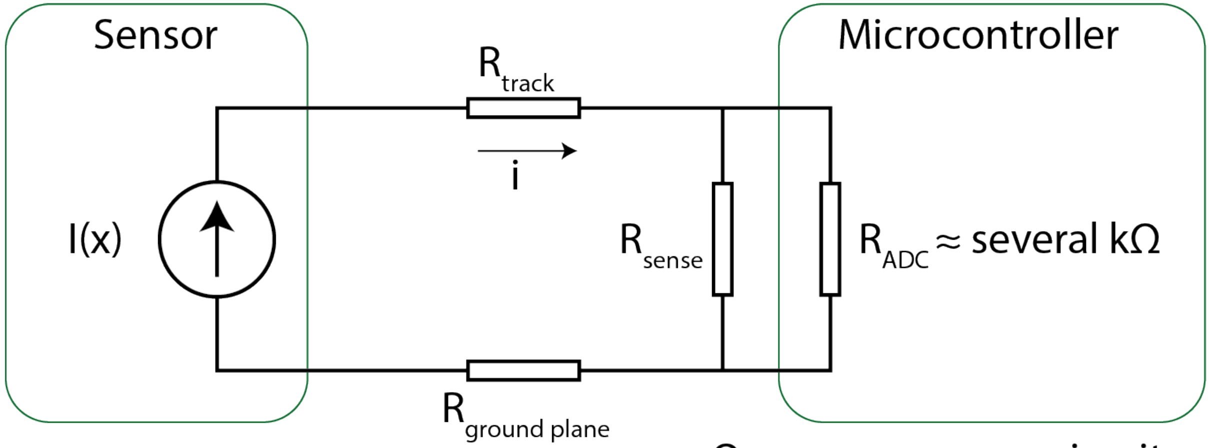


Fig.1 Output Voltage vs. Temperature

Voltage vs current

- How to communicate the signal from sensor to microcontroller?
- An analog signal can be transmitted as a **voltage** or as a **current**.
- Both are used in practice.
- What are some advantages of each?

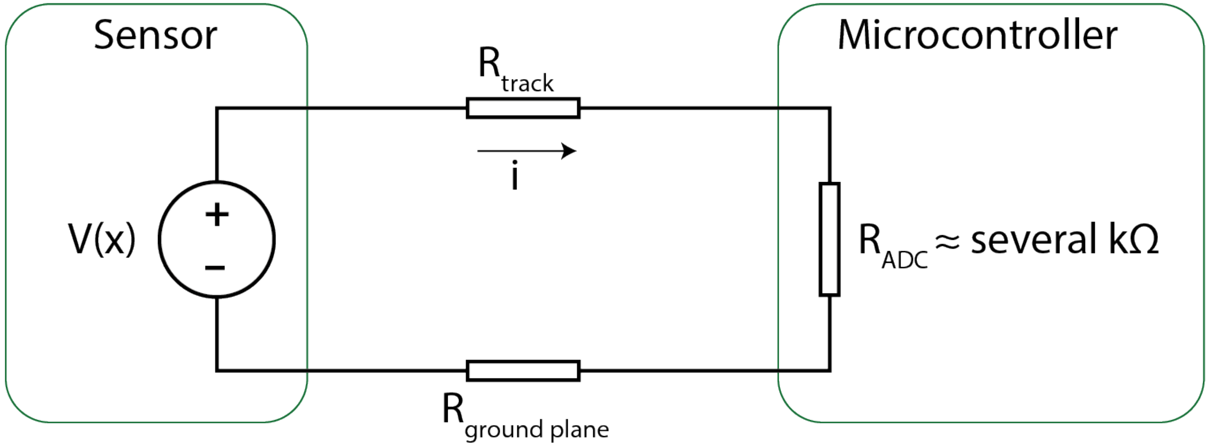




Or use an op-amp circuit to amplify the voltage across R_{sense}.

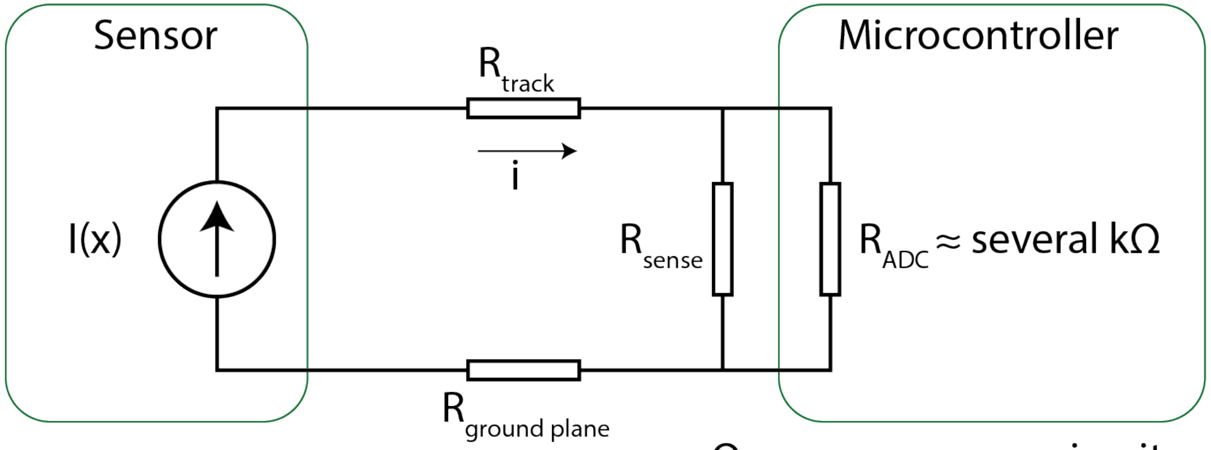
Voltage signals

- **If the sensor is close to the microprocessor, a voltage signal is preferred.**
- Voltage sensors are found in almost all microprocessors. They are cheap and ubiquitous.
- There will be a voltage drop across the wires that connect the sensor and microcontroller.



Current signals

- **If the sensor is far away from the microprocessor, a current signal may be preferred.**
- The current is the same everywhere in that loop.
- Consequently, current signals are more robust across long distances.



Or use an op-amp circuit to amplify the voltage across R_{sense}.

The 4-20 mA current loop

- A common industry standard is the **4-20 mA current loop**.
- The minimum value is transmitted as 4 mA.
- The maximum value is transmitted as 20 mA.
- A “live zero” of 4 mA allows open circuit faults to be immediately detected.

Usage of current loops

- Current loops allow the sensor to be powered using the same lines that transmit the signal back:
  - A power supply injects a voltage into the loop.
  - The sensor modulates the <u>current</u> that flows.
  - The power supply senses the current and thereby reads the signal.
- These current loops are often found in industrial environments.

Analog to digital conversion

- Let’s turn our attention to the microprocessor.
- Assume that a sensor has measured a quantity and now there’s a voltage at the microprocessor that needs to be read.
- This is called **analog to digital conversion**.
- The peripheral that performs this conversion is called an **analog to digital converter (ADC)**.

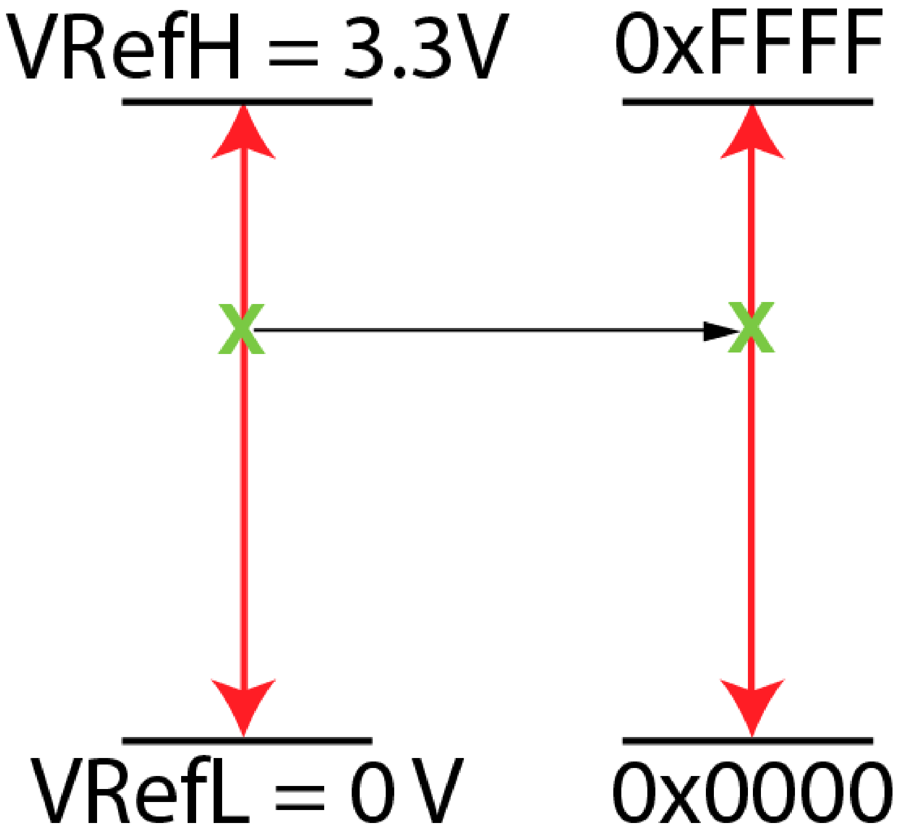
ADC operation

- **Single ended mode** measures a voltage (with respect to ground).
- **Differential mode** measures the <u>difference in voltage</u> between two pins.

- If the signal of interest is of the form $a - b$ then differential mode will be more accurate.

Single ended ADC

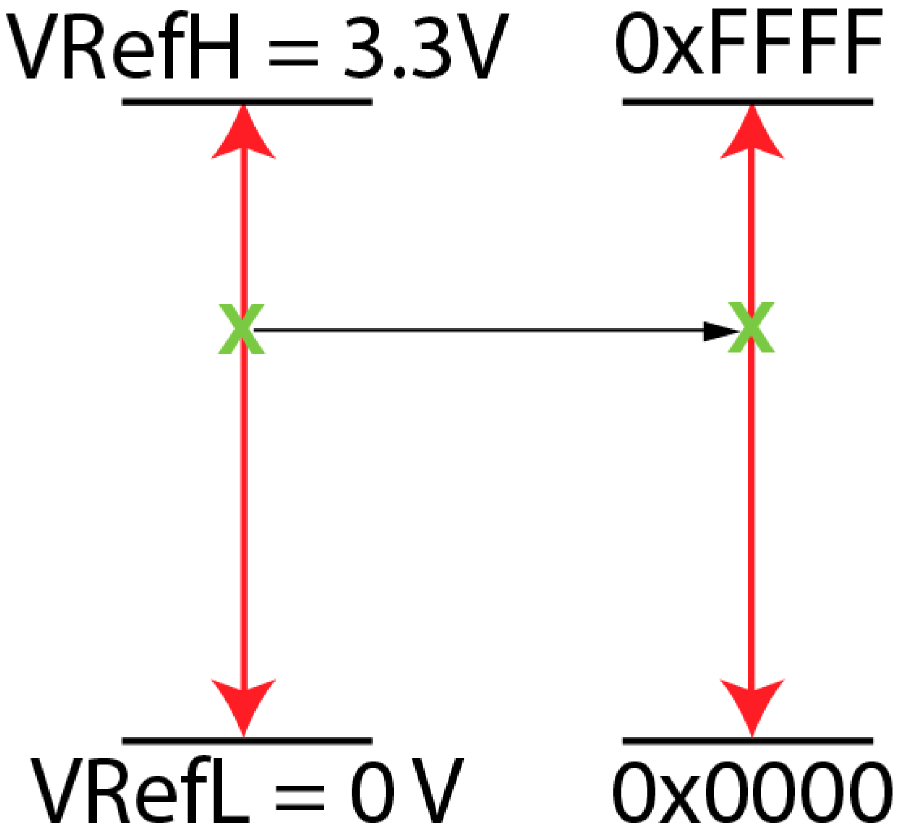
- Given an input voltage, the ADC measures its position between two reference voltages.
- **Vssa** (analog ground) or **VRefL** (voltage reference low) is the zero point.
- **Vdda** (analog supply) or **VRefH** (voltage reference high) is the maximum point.
- On the FRDM board, **VRefH** is connected to the on-board 3.3 V regulator.



Single ended ADC

- An n-bit ADC will produce a value
$$y = V_{input} \times \frac{2^n - 1}{V_{refH}}.$$
- A 16-bit ADC will return a number in the range 0 to $2^{16} - 1$ = 0xFFFF.
- We have $V_{refH} = 3.3$ so:

$$y = V_{input} \times \frac{2^{16} - 1}{3.3}$$



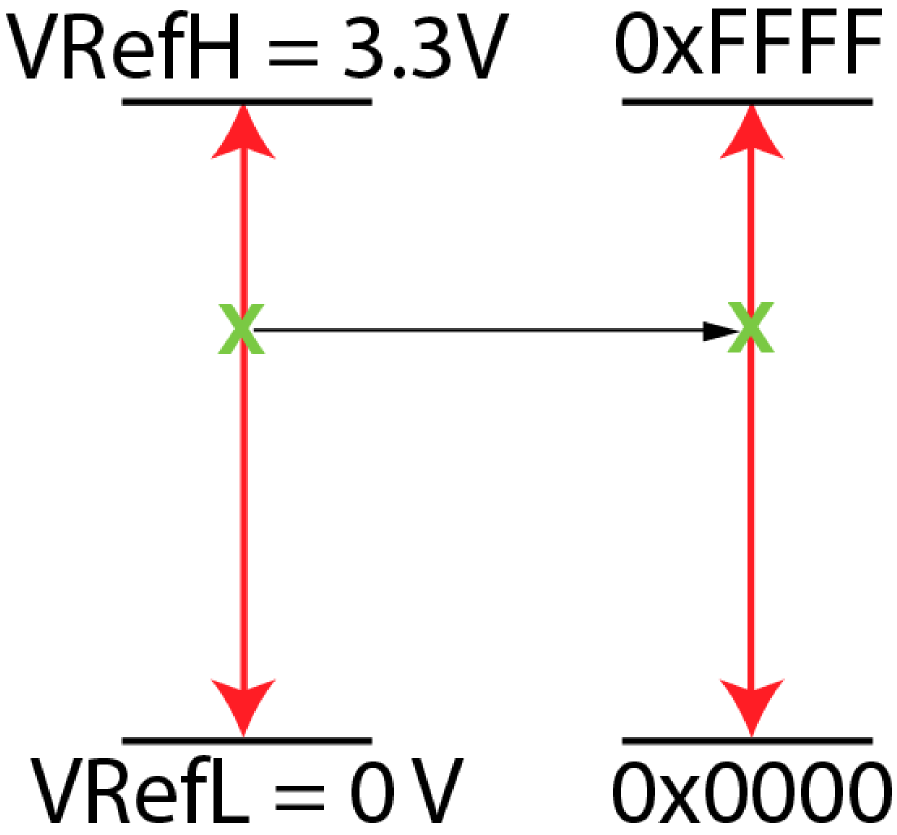
Single ended ADC

- To convert from the ADC value to a voltage,

$$V_{input} = y \times \frac{V_{refH}}{2^n - 1}.$$

- For a 16 bit ADC running on a 3.3V system:

$$V_{input} = y \times \frac{3.3}{2^{16} - 1}$$

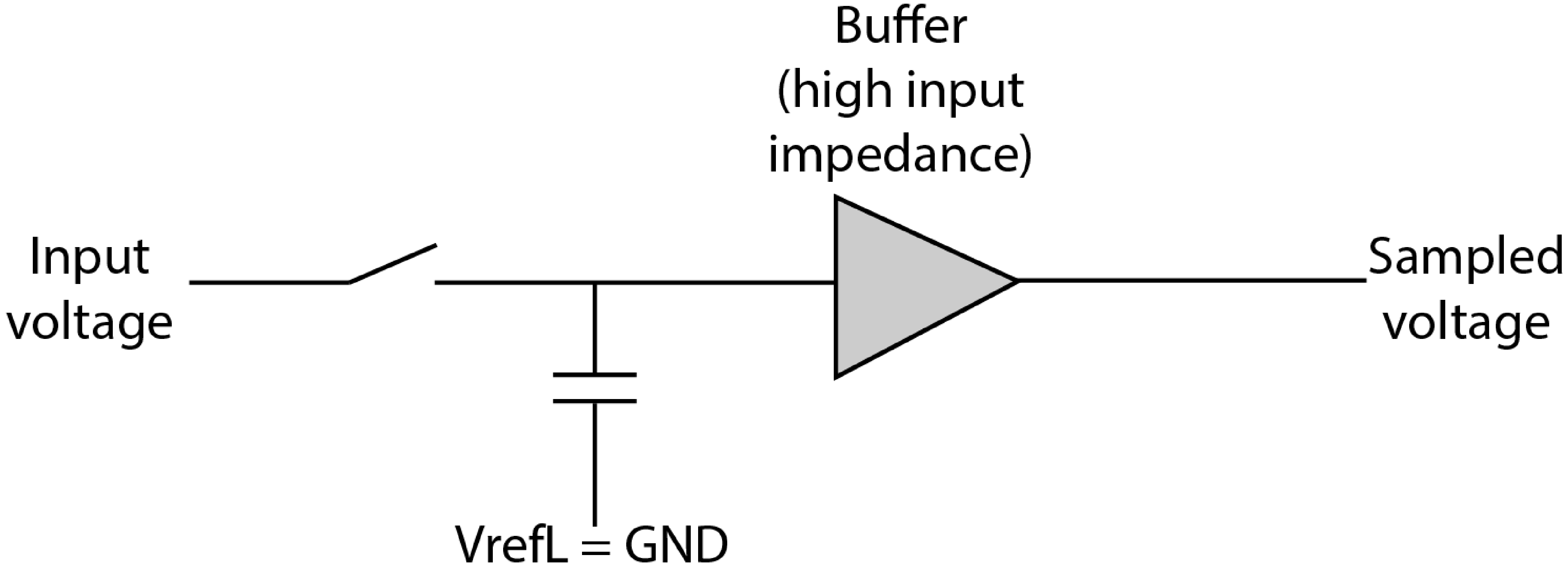


ADC operation

- The K20 and K22 microprocessors implement a **successive approximation (SAR) ADC**.
  - There are also other types of ADC.
- It’s helpful to understand the underlying hardware in order to appreciate the design tradeoffs.

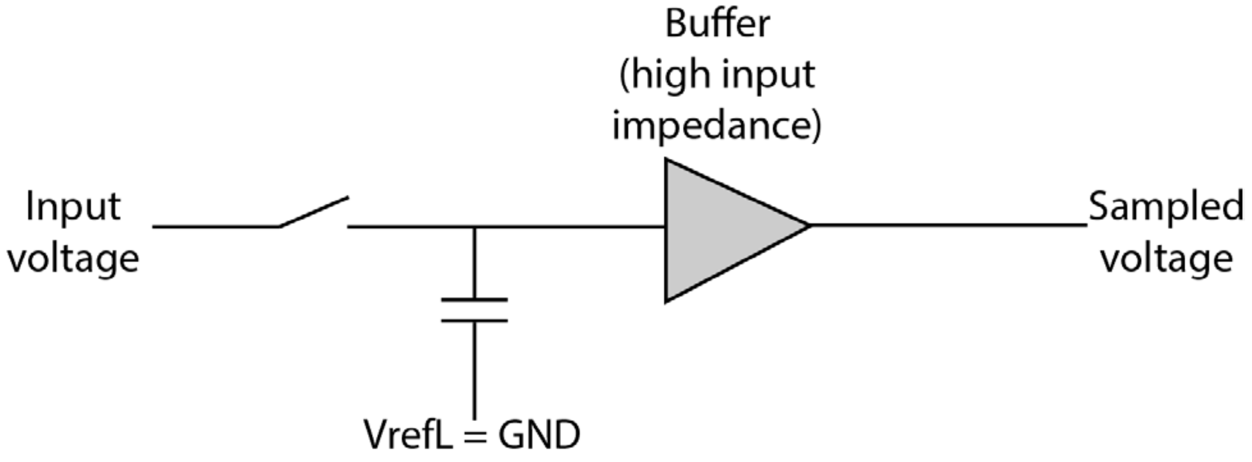
Stage 1: Sample and hold

- A **sample and hold** circuit captures the input voltage



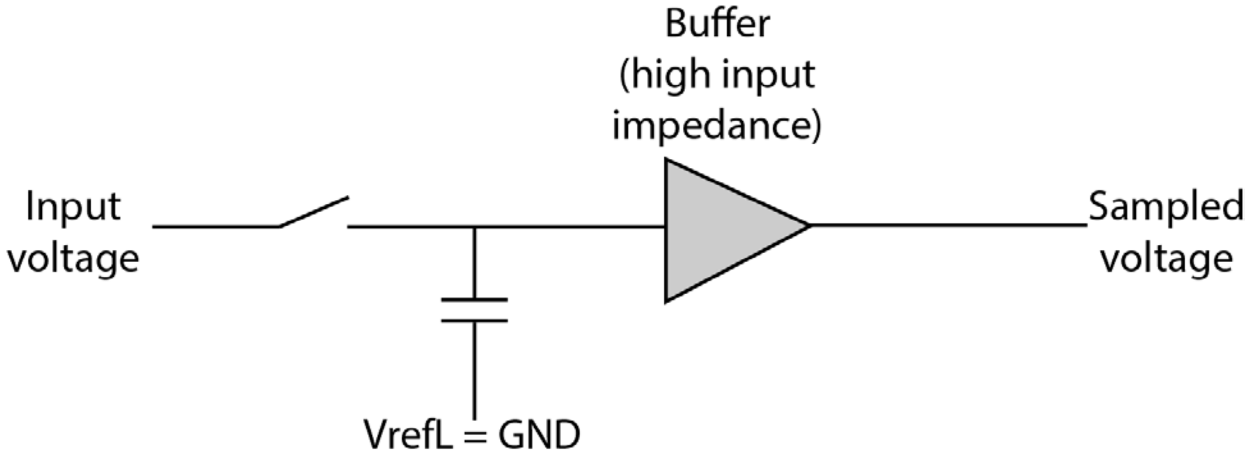
Sample and hold tradeoffs

- Want a small capacitor to not load the analog source (and change its voltage)
- Want a larger capacitor so that the buffer doesn't discharge it during analog-to-digital conversion.



Characteristics of the analog source

- For best results, need the analog source to have a small output impedance. Otherwise the RC time to charge the sample-and-hold capacitor may be too long.
- In practice, use active drive in front of the sample capacitor.



Sample and hold: realistic design

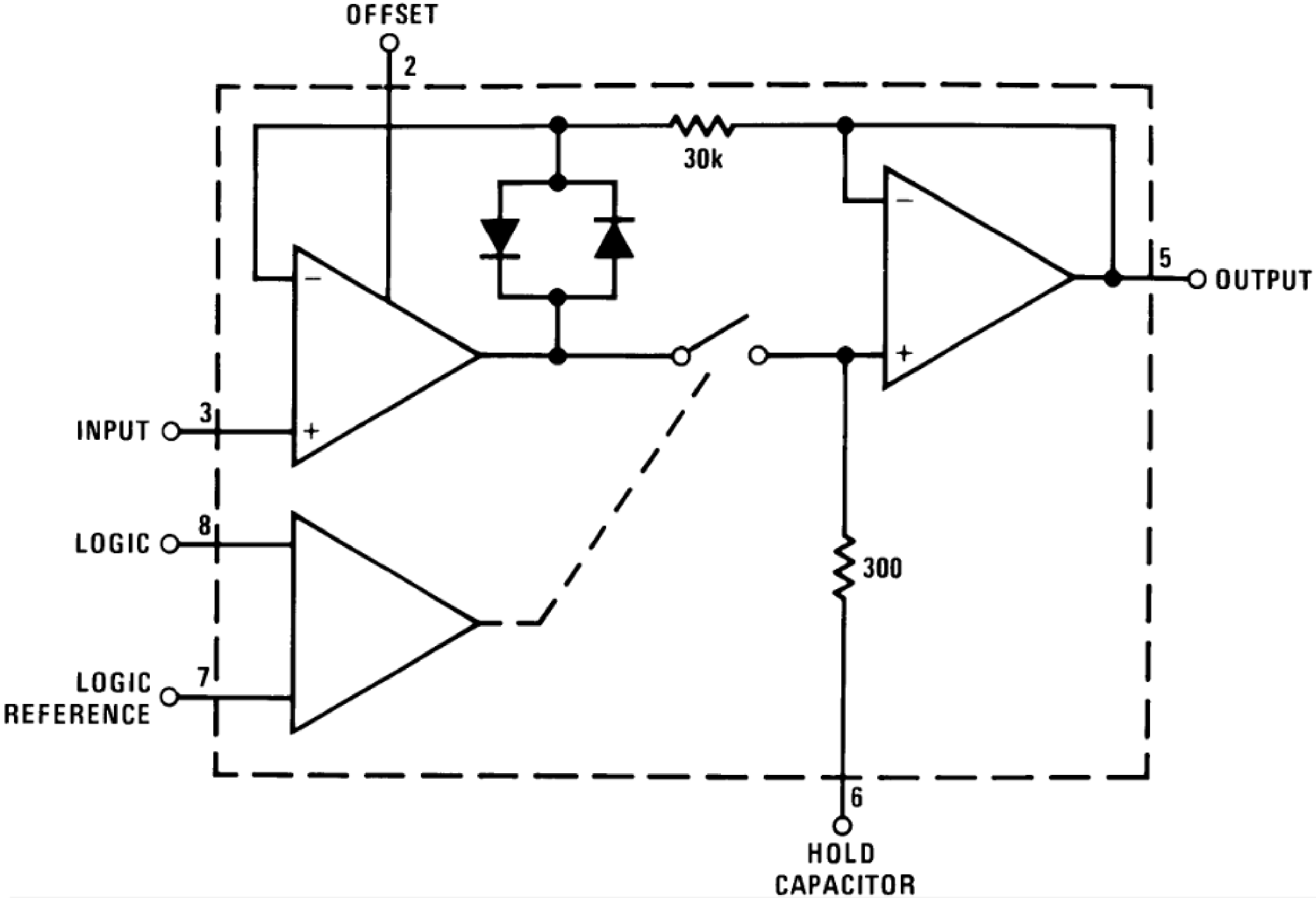


Image from Texas Instruments LF198 datasheet

Sample and hold: realistic design

- Drive the hold capacitor with the first op-amp (to avoid loading the input).
- The output stage is a simple voltage follower.
- The back-to-back diodes provide a feedback path to keep the first op-amp stable when the switch is open.

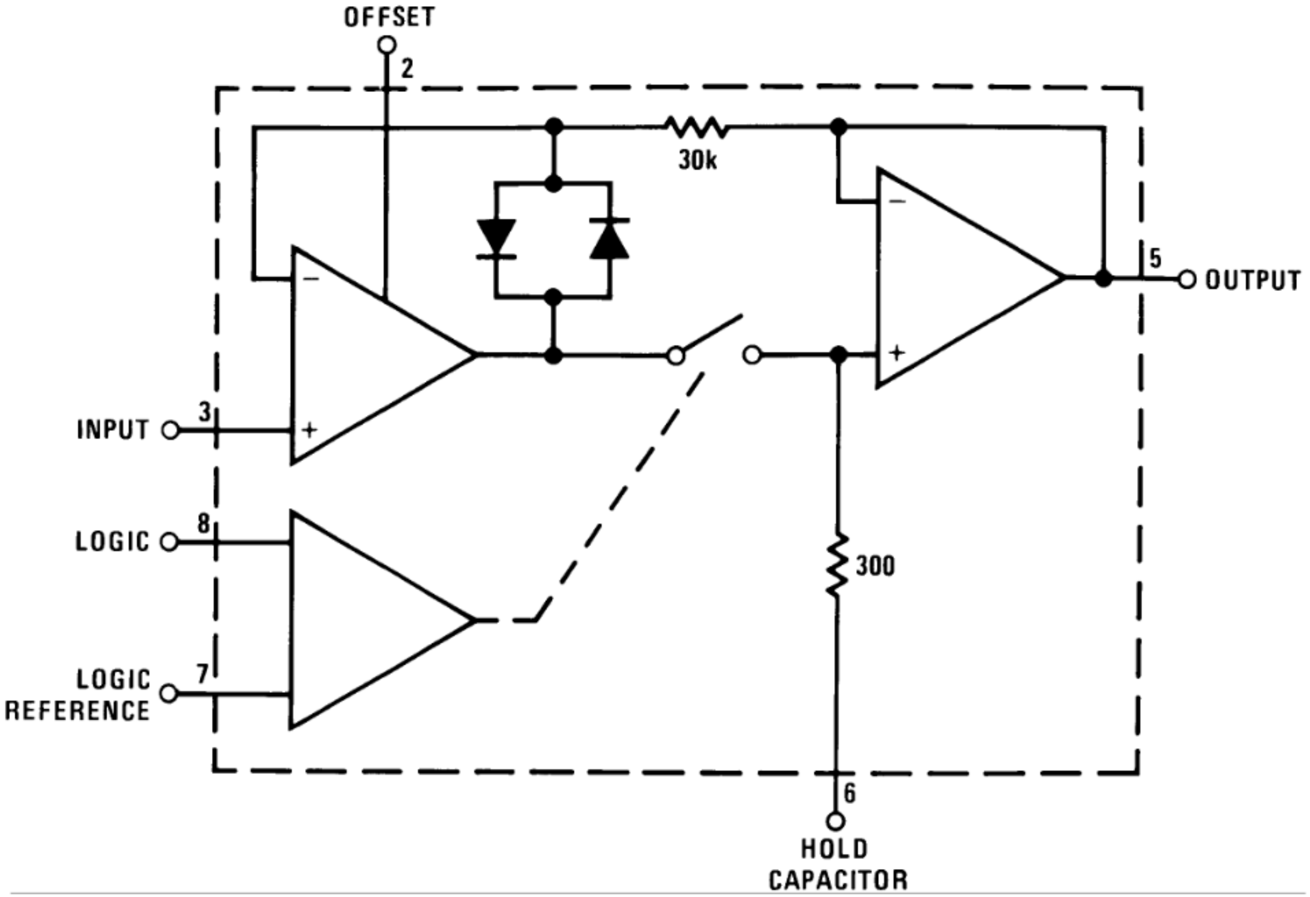


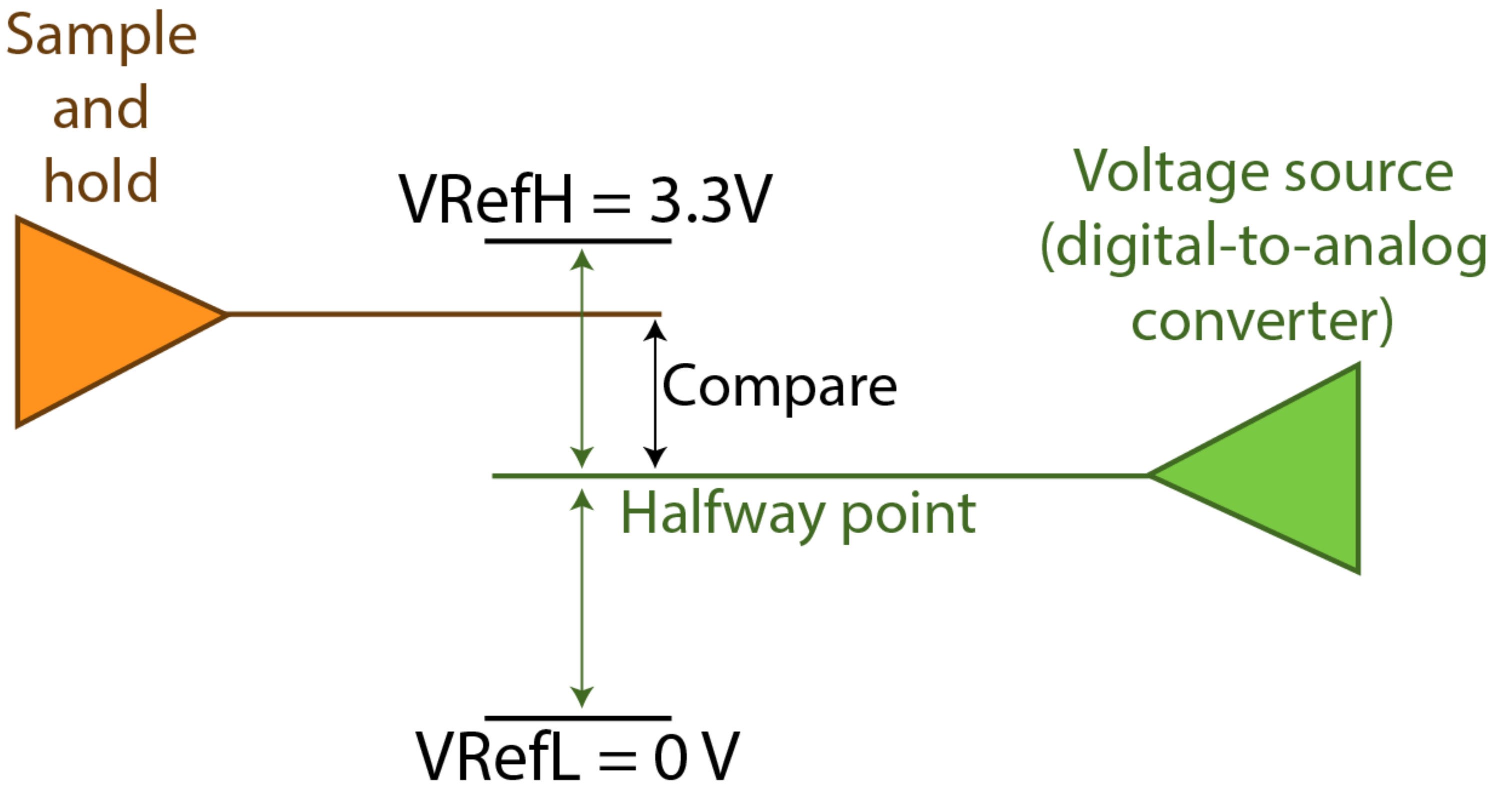
Image from Texas Instruments LF198 datasheet

Stage 2: Successive approximation

- The sample and hold circuit provides a stable voltage.
- How to measure this voltage?
- **Successive approximation:**
  1. Guess the voltage
  2. Check whether it’s too low or too high with a comparator circuit.
  3. Adjust the guess and repeat.

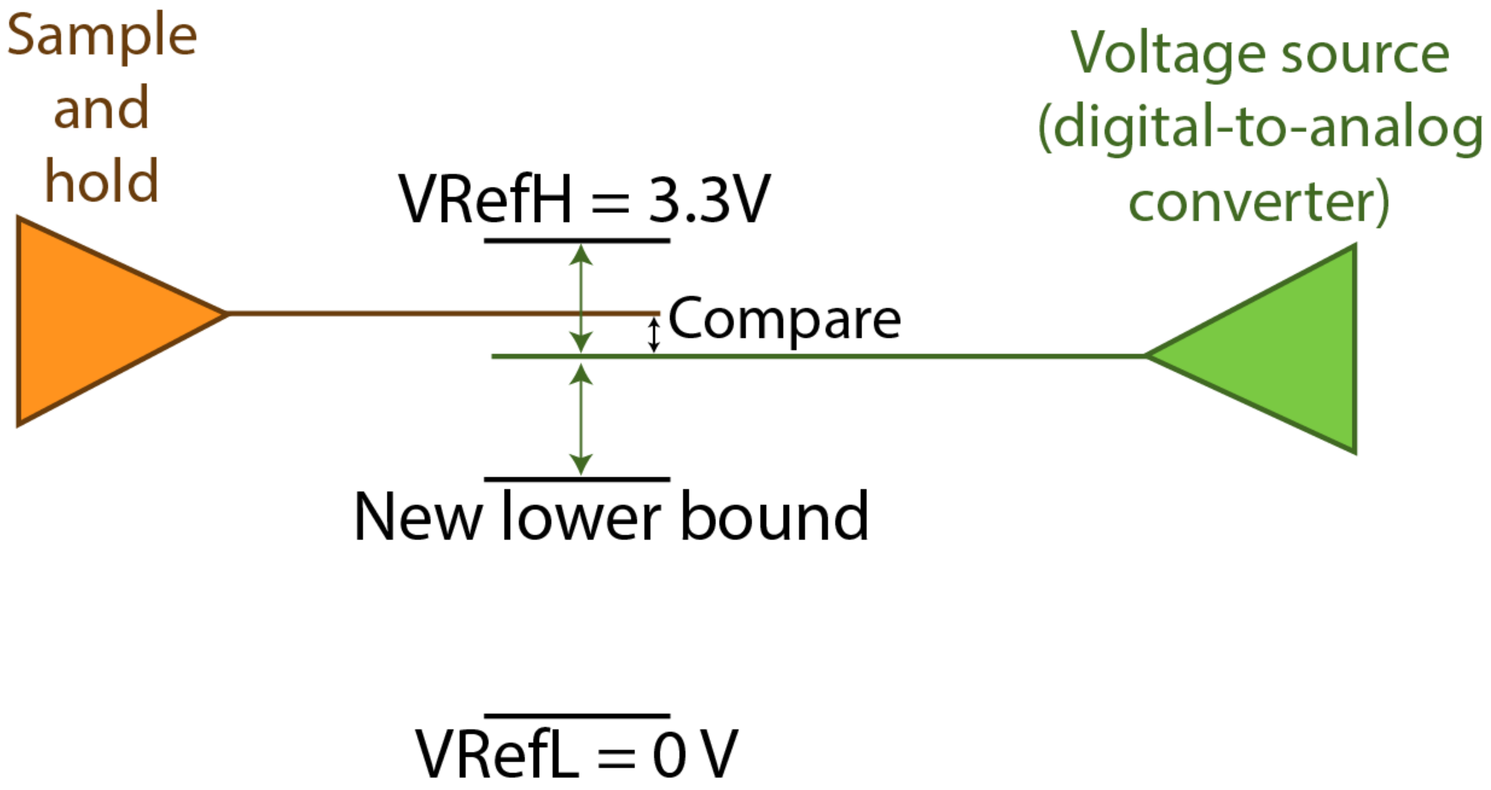
Stage 2: Successive approximation

- Divide the reference voltages in half.
- **Is the signal in the top half or the bottom half?**



Stage 2: Successive approximation

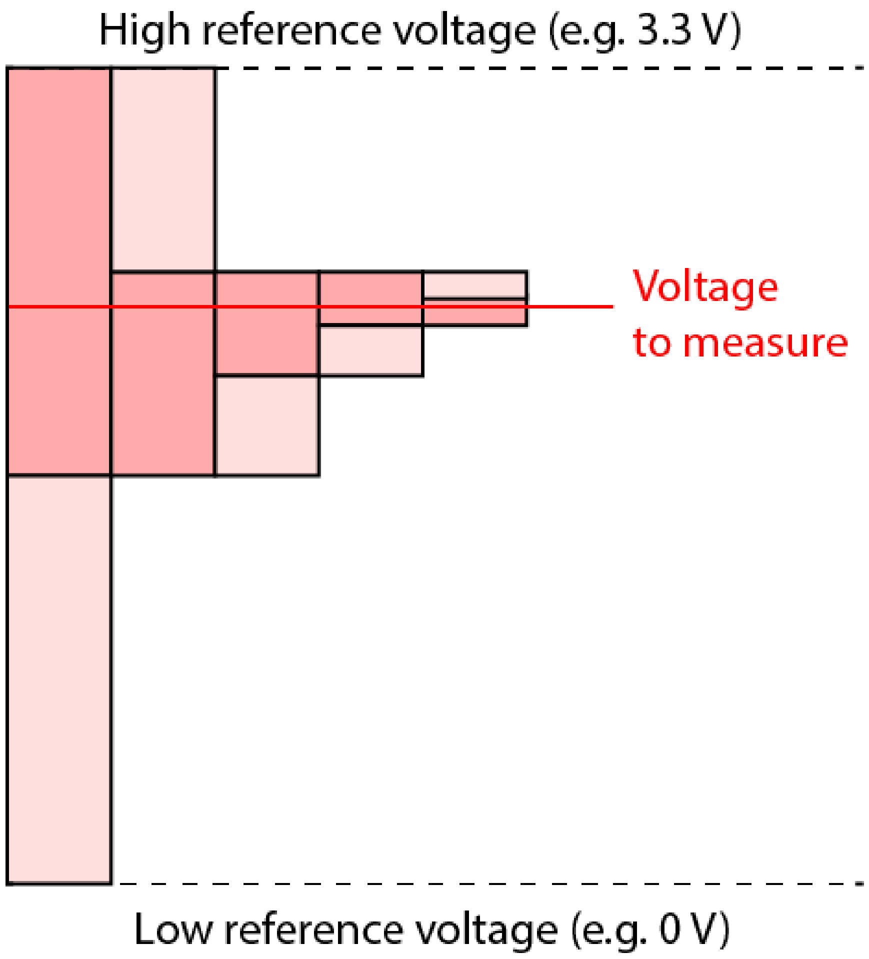
- The previous comparison reduced the range by half.
- Test against a new point halfway between the upper and lower bounds.



Stage 2: Successive approximation

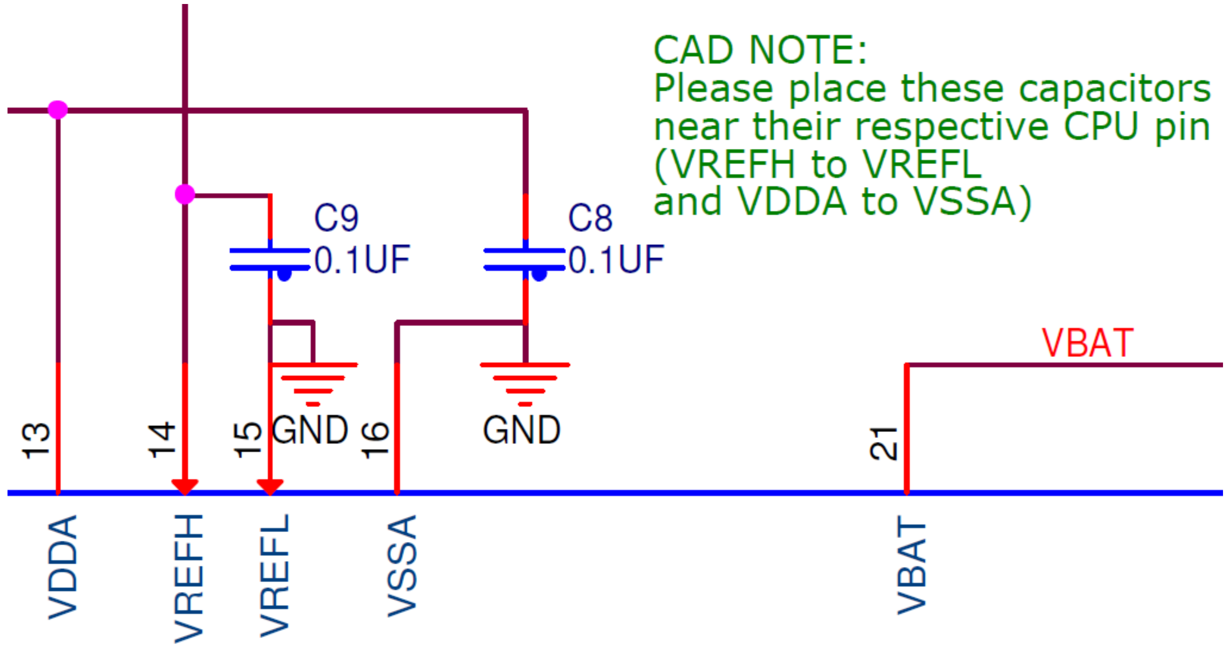
Each comparison:

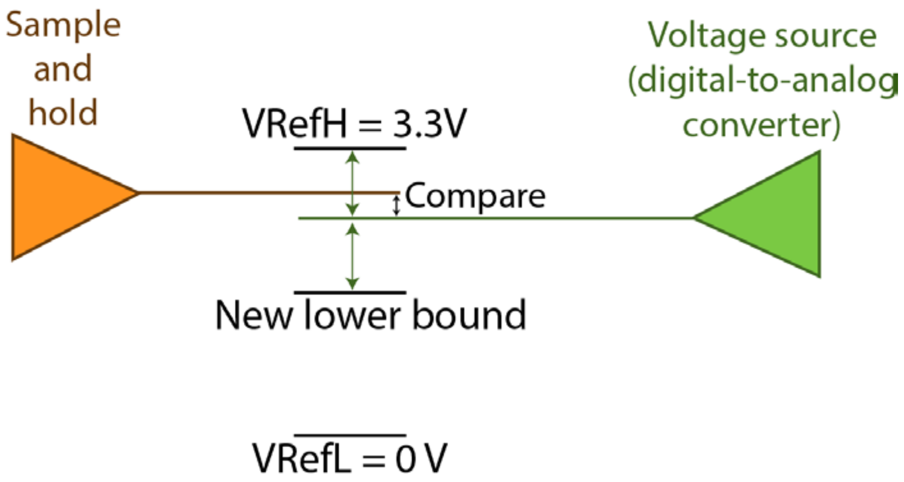
- Halves the search space.
- Supplies one bit of information.



Design considerations

- The voltage references must be **absolutely stable** during conversion.





SAR architecture

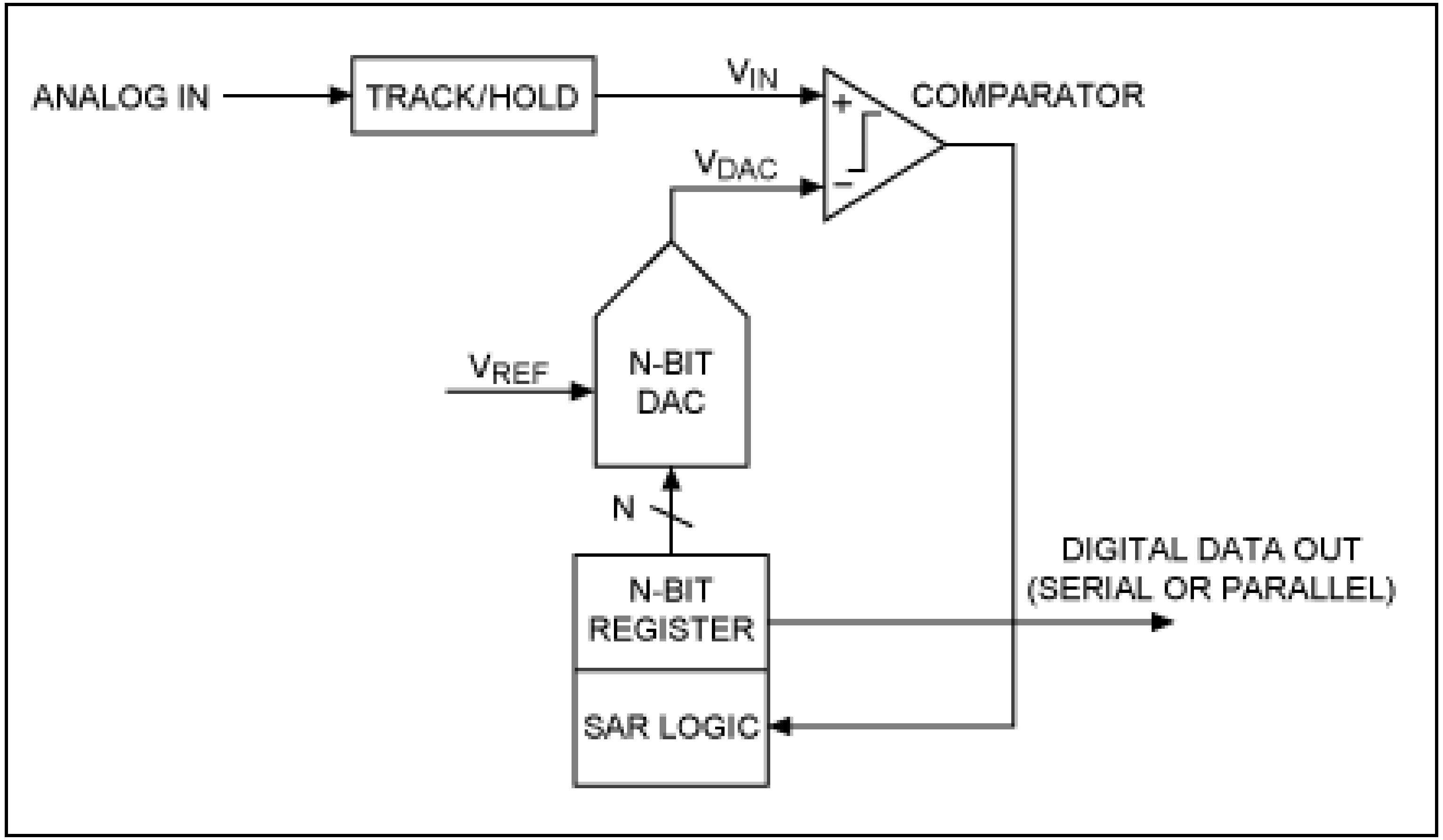


Image from Maxim Application Notes 1080

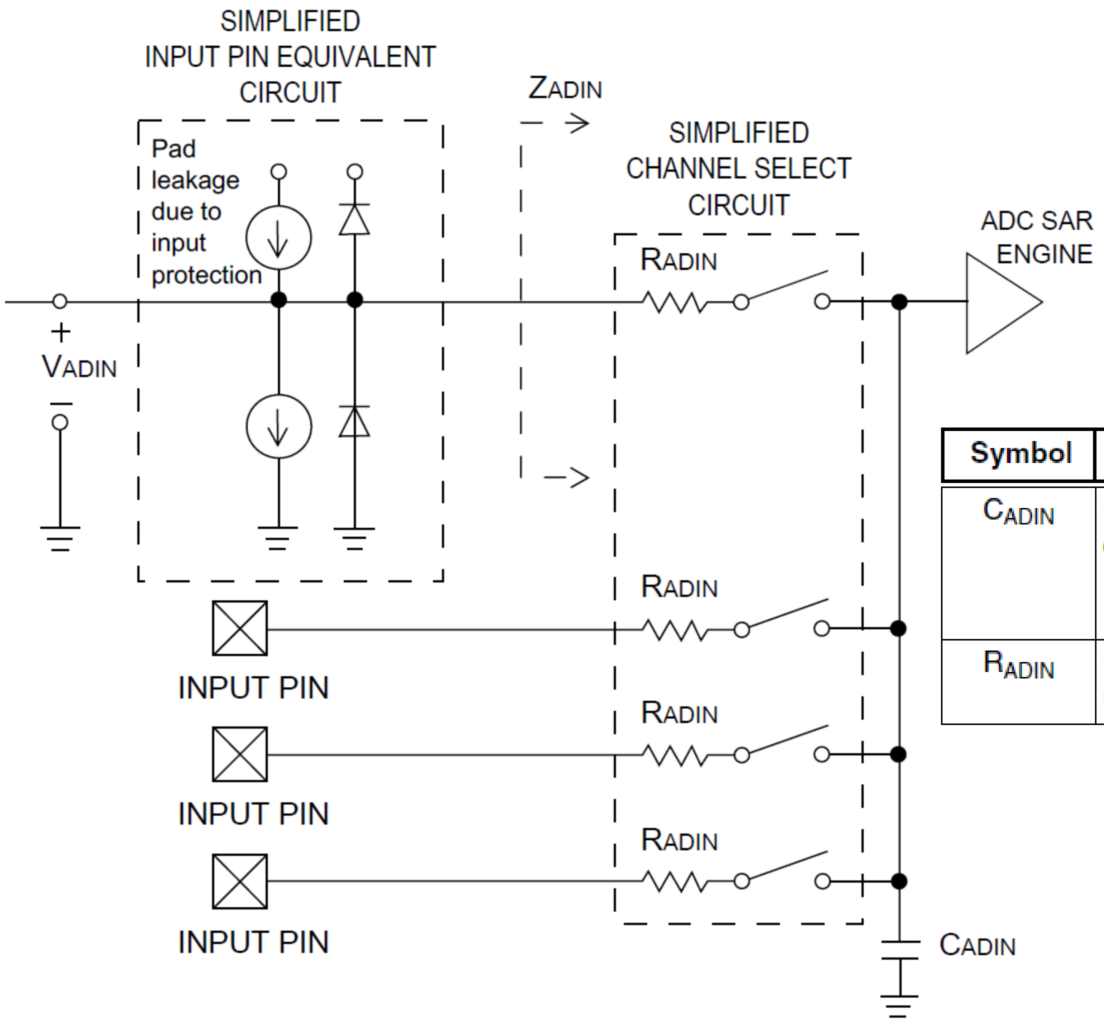
Limiting factors

- The voltage source (digital-to-analog converter) must set a voltage quickly and accurately.
- The comparator must resolve a small voltage difference very quickly.
- There’s a **speed-accuracy tradeoff** for the overall system.
- More accurate conversions necessarily take longer.

The ADC in the Kinetis family

- Can be configured to use multiple channels.
- When conversion is triggered, the ADC reads each channel nearly simultaneously.

Kinetis ADC equivalent circuit



Symbol	Description	Conditions	Min.	Typ.[1]	Max.	Unit	Notes
C_{ADIN}	Input capacitance	• 16-bit mode • 8-bit / 10-bit / 12-bit modes	— —	8 4	10 5	pF	
R_{ADIN}	Input series resistance		—	2	5	kΩ	

Image from Kinetis K22F datasheet

Requirements on the analog source

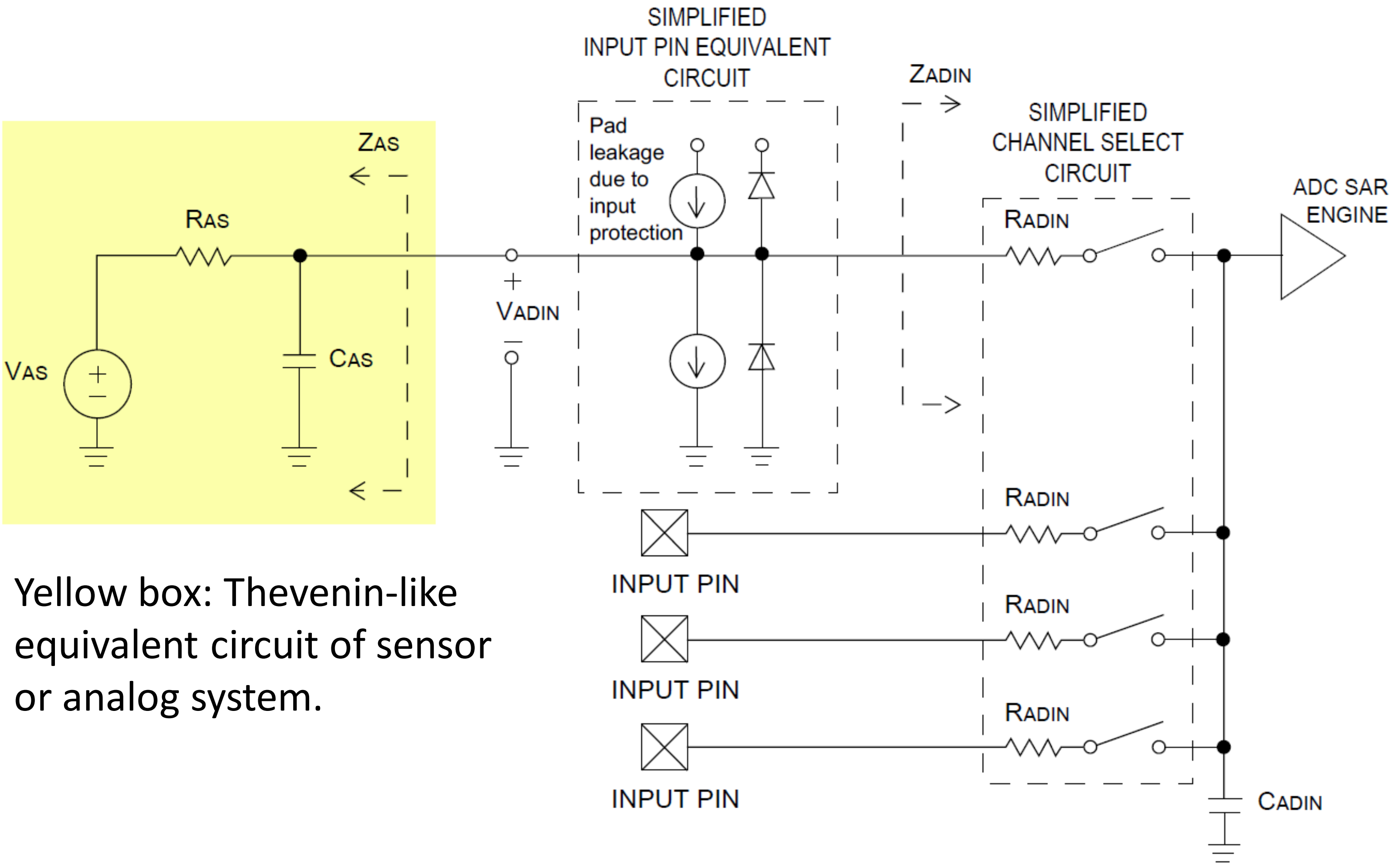


Image from Kinetis K22F datasheet

Yellow box: Thevenin-like equivalent circuit of sensor or analog system.

For K22F:

$$R_{AS} < 5\text{k}\Omega$$
$$R_{AS}C_{AS} < 1\text{ns}$$

If your analog source does not meet these requirements, use an op-amp buffer, e.g. voltage follower or non-inverting amplifier.

ADC calibration

- The ADC must be calibrated before it is used.
- There’s a self-calibration function that should be run when the microprocessor is powered on.
- There is a discussion in the Reference Manual about sources of error. Refer to this as needed.

Temperature effects

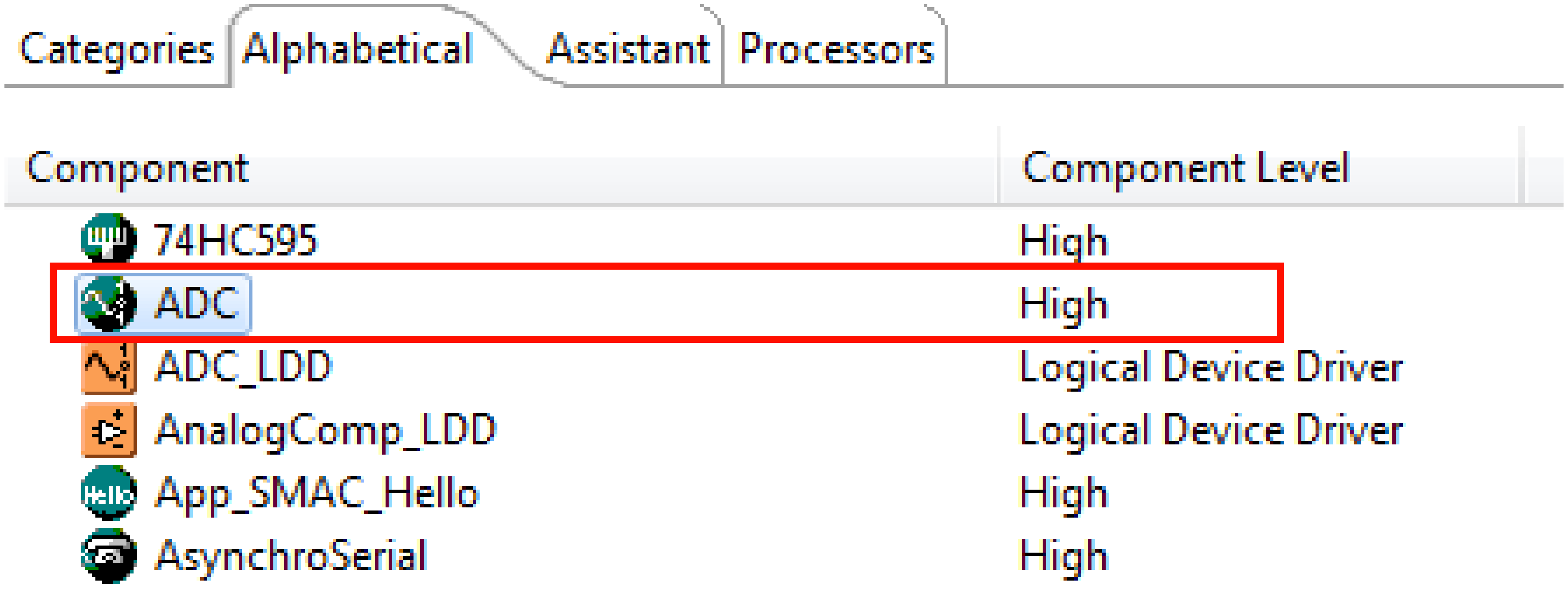
- ADC calibration depends upon the temperature.
- There’s a built-in temperature sensor that’s used for calibration.
- The temperature can be read directly if desired.
  - There’s also a separate external temperature sensor on the FRDM K20D50M board.

Using the ADC with registers

- Configuration via registers is described in the reference manual.
- There are pseudocode examples showing the sequence of register writes to configure and then trigger conversion.

Using ADCs in Processor Expert

- Use the ADC component.



ADC properties

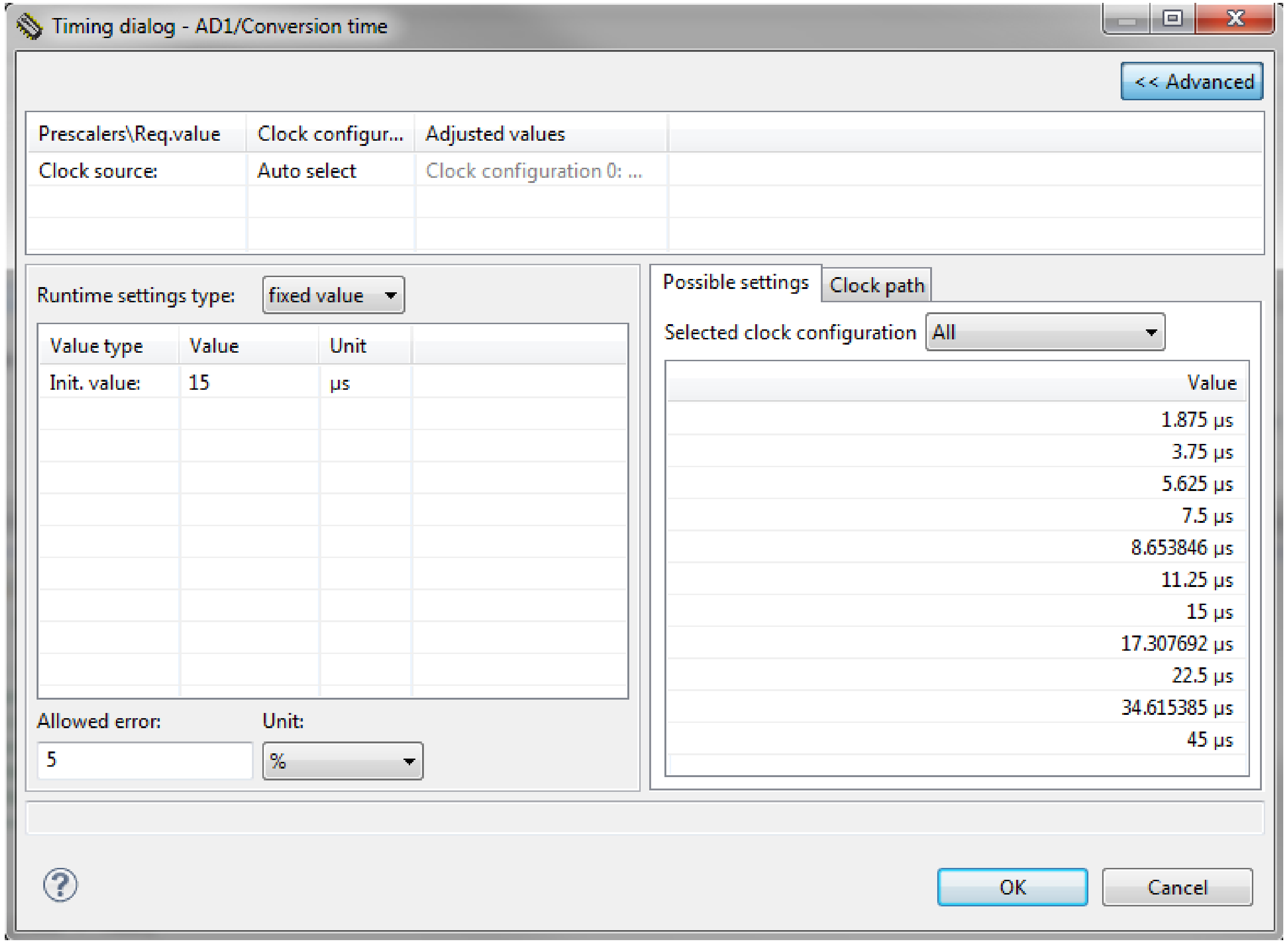
Component Inspector - ADC | Components Library

Properties | Methods | Events

Name	Value	Details
A/D converter	ADC0	ADC0
Sharing	Disabled	
Interrupt service/event	Disabled	
A/D channels	1	
Channel0		
A/D channel (pin)	ADC0_DM0	
Mode select	Single Ended	
A/D resolution	16 bits	16 bits
Conversion time	12.48 µs	12.480 µs
Low-power mode	Disabled	
High-speed conversion mode	Enabled	
Asynchro clock output	Enabled	
Sample time	12	Total conv. time: high: 13.54 us
Initialization		
Enabled in init. code	yes	

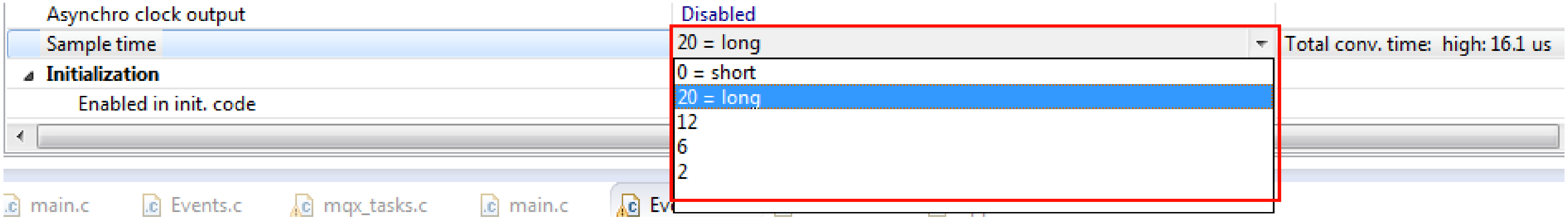
Configuring the conversion time

- The conversion time is controlled by the clock frequency supplied to the ADC.
- There’s a configurable clock divider running off the bus clock.



Extra time

- An extra number of clock cycles can be given for conversion.
- Longer times may be more accurate.



Using the ADC component

- First, calibrate:

```
ADC_Calibrate(TRUE);
```

- The argument TRUE indicates that the function should wait for calibration to complete.
- Otherwise, there’s an interrupt-driven event OnCalibrationEnd.
- Only do this once (unless you expect temperature changes, in which case, consider recalibrating).

Triggering a conversion

- Trigger a conversion using the Measure function:

```
ADC_Measure(TRUE);
```

- The TRUE argument indicates that the function should wait for conversion to finish.
- There’s an interrupt event OnEnd that triggers when conversion is complete.

Extracting a result

- In the case of a **single channel**:

```
uint16 adc_result;
ADC_GetValue16(&adc_result);
```

- Notice that GetValue16 needs a pointer to the variable to receive the result.

Extracting results from multiple channels

- If there are multiple channels, allocate an array:

```
uint16 measurements [NUM_CHANNELS];
ADC_GetValue16(measurements);
```

- Notice that no & is needed because an array is already a pointer (to the first item)

A common mistake

```
uint16 adc_result;
ADC_Measure(TRUE);
for (;;) {
    ADC_GetValue16(&adc_result);
    // do something with adc_result
}
```

- **Measure is only called once!** The same result will be printed over and over.

Using interrupts

- Conversion is slow in comparison with the microprocessor.
- Can use interrupts to be notified when a conversion is complete.

Properties | Methods | Events

Name	Value
A/D converter	ADC0
Sharing	Disabled
Interrupt service/event	Enabled
A/D interrupt priority	high priority
A/D channels	5
Channel0	
A/D channel (pin)	ADC0_SE14/TSI0_CH13/PTC0/SPI0_PCS4/PDB0_EXTRG
Mode select	Single Ended
Channel1	
A/D channel (pin)	ADC0_SE15/TSI0_CH14/PTC1/LLWU_P6/SPI0_PCS3/UART1_RTS_b...
Mode select	Single Ended
Channel2	
A/D channel (pin)	ADC0_SE7b/PTD6/LLWU_P15/SPI0_PCS3/UART0_RX/FTM0_CH6/F...

Event handler

- Call GetValue16 from the event handler.

```
volatile uint16 measurements [NUM_CHANNELS];
void ADC_OnEnd(void)
{
    ADC_GetValue16(measurements);
}
```

Summary

- Many sensors produce analog voltages that are read into a microprocessor by an analog-to-digital converter (ADC).
- ADCs usually have a speed-accuracy tradeoff.
- Implementations must place capacitors on the analog voltage supplies near the microprocessor for accurate readings.
- In Processor Expert, trigger a conversion with the Measure function, and once it’s finished, retrieve the result with the GetValue16 function.