EECS 373
Design of Microprocessor-Based Systems

Robert Dick
University of Michigan

Lecture 10: Timer implementation, ADCs, and DACs

7 February 2017

Slides inherited from Mark Brehob.

Outline

- Context and review
- Project schedule
- Timer implementation
- ADCs and DACs
  - Fundamentals
  - Operation

Context and review

- Buses
  - UART: supports various protocols and (potentially high voltage) signaling methods through appropriate drivers.
  - SPI: four-wire, full-duplex, with wire addressing.
  - I2C: two-wire, half-duplex, with seven-bit addresses.
    - Can be a pain to work with when devices have hard-coded addresses.
- ADCs and DACs
  - Deferring review until appropriate part of lecture.

Outline

- ~~Context and review~~
- Project schedule
- Timer implementation
- ADCs and DACs
  - Fundamentals
  - Operation

Team formation meeting

- 7:30-9pm today.
- 1311 EECS.
- Come prepared to briefly summarize your project idea.
- Review terse project proposals first if you are looking for a team to join.
- Not necessary to attend if you have a topic and your team is fully staffed.

Project proposal meeting

- Each team should sign up for a 20-minute project proposal meeting with Matthew and me.
- Only one slot for entire team.
- Use team name.
- Also use team name on project proposal.

Written project proposal

- Most teams will revise, expand the terse project proposal of a team member.
- Matthew and I will review before your presentations, so we need them by Saturday.
- Will post example today (as soon as there is a break in my schedule, probably after team formation meeting).
- These will evolve during the project, but the general direction and components should be settled this week.

- Immediate goal: Have at least two of the components you need to interface ordered by 6 March, or verify that we already have them.

Outline

- ~~Context and review~~
- ~~Project schedule~~
- Timer implementation
- ADCs and DACs
  - Fundamentals
  - Operation

Potential sticking point related to MMIO

- See Lab 5 APB3 Verilog interface code.
  - Put timer registers at specific memory locations for MMIO.
- See Lab 5 mytimer.h.
  - Use same layout within mytimer_t struct.
  - C and/or assembly can use MMIO to those memory locations to read/write timer registers.
  - FPGA can react to changes.

Struct packing 1

```
struct fat {
    char b; // 1 byte
// What goes here?
    char *p; // 4 bytes
    char c; // 1 byte
// What goes here?
    int x; // 4 bytes
};
```

Struct packing 2

```
struct skinny {
    char *p; // 4 bytes
    int x; // 4 bytes
    char b; // 1 byte
    char c; // 1 byte
// What goes here, maybe?
};
```

Deeper dive on ways of implementing PWM

Given

- 0.375 duty cycle → 3/8 → 3-tick on, 5-tick off
- Count-up timer.

Single-register approach

- Set compare register to 3.
- Turn on output.
- LOOP
  - Start timer.
  - When counter hits 3, trigger interrupt.
  - ISR turns off output.
  - Sets compare register to 8
    - Or resets counter and sets compare register to 5.
  - When counter hits 8 (5), trigger interrupt.
  - ISR turn on output.
  - Sets compare register to 3.
  - Resets counter.

Deeper dive on ways of implementing PWM

Given

- 0.375 duty cycle → 3/8 → 3-tick on, 5-tick off
- Count-up timer.

Multi-register approach

- Set compare register to 3.
- Set overflow register to 8.
- Turn on output.
- LOOP
  - Start timer.
  - When counter hits 3, trigger (compare) interrupt.
  - ISR does MMIO read to figure out what the root cause of the (fabric) interrupt was. Turns off output.
  - When counter hits 8, trigger (overflow) interrupt.
  - Also clears counter value in hardware.
  - ISR does MMIO read to figure out cause of (fabric) interrupt. Turns on output.

  - What if the counter was one-shot or didn't clear on overflow?

Definition: bit banging

- Using software to directly set pin values instead of setting parameters in special-purpose hardware.
- Particularly for communication protocols.
- Instruction processor takes responsibility for timing and other aspects of protocol.
- Flexible.
- Wastes power (chainsaw when scissors might be better).
- Keeps processor occupied.

Count-down timer virtual timer example

- Initialize linked list with period, time remaining and function pointer.
- Sort list in order of time left from shortest to longest.
  - Or create data structure that can never be out of order.
- Start hardware timer with shortest time left in hardware counter register.
- Hardware timer interrupts.
- Toggle respective LED (call via function pointer).
- Subtract elapsed time from time left.
- If current timer continuous, add period to time left.
- Sort list in order of time left.
- Start hardware timer with shortest time left in hardware counter register.
- Hardware timer interrupts.
- Repeat.

What if you really need things in sorted order?

- Create data structure that can never be out of order.
  - Elegant.
  - Efficient.
  - May not be flexible enough for some applications.
- Write your own sort?
  - Time consuming to debug, especially for efficient sorts like quick sort and merge sort.
- Reuse C standard library.
  - Make array of list nodes.
  - Write comparison routine that takes list node pointers.
  - Call qsort().
  - Reconstitute list from array.

- Design rule: If there's something close that is already written and debugged, use it even if you need a shim.

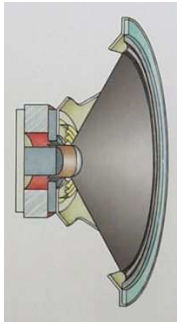
Outline

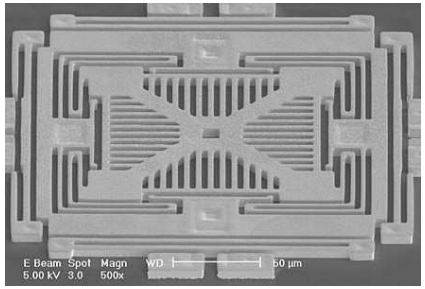
- ~~Context and review~~
- ~~Project schedule~~
- ~~Timer implementation~~
- ADCs and DACs
  - Fundamentals
  - Operation

Review: Many signals effectively analog

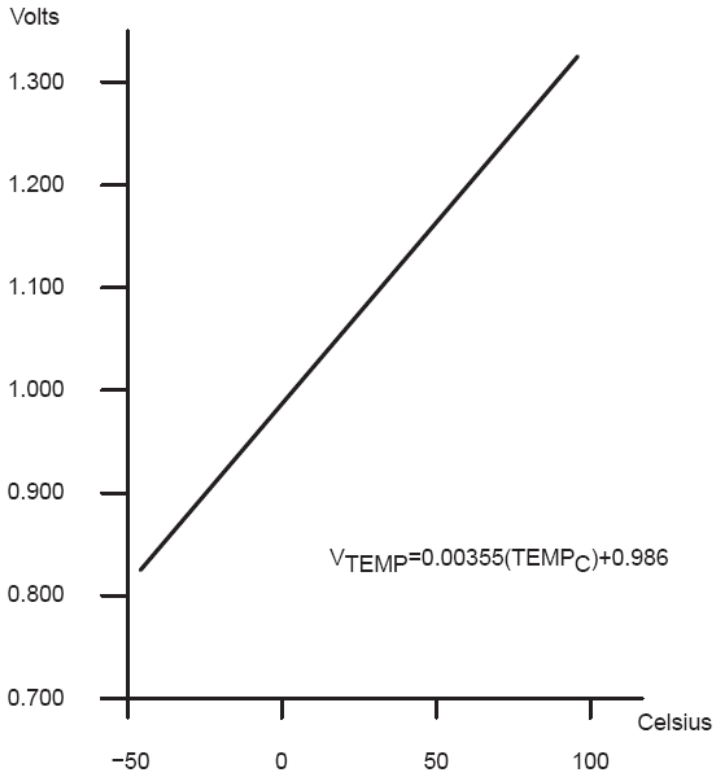
- At the macroscopic level, many signals have so many possible values they are effectively continuous/analog.
  - Sound, light, temperature, pressure, voltage, etc.

- Path to digital system
  - Source → continuous voltage → discrete value.
- Transducers: converts one type of energy to another
  - Electro-mechanical, Photonic, Electrical, …
- Examples
  - Microphone/speaker
  - Thermocouples
  - Accelerometers



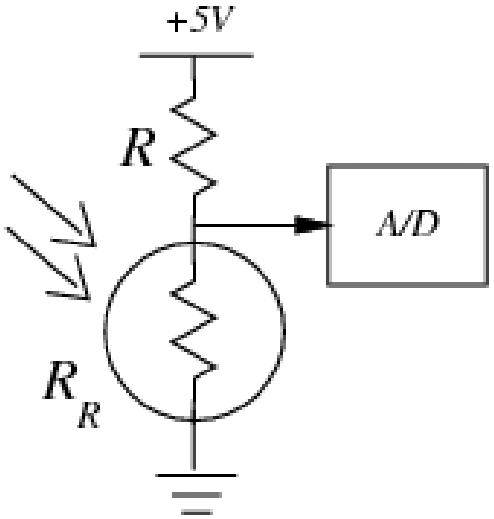


Review: Transducers convert one form of energy into another



Review: Convert light to voltage with a CdS photocell

- $V_{signal} = (+5V)\ R_R/(R + R_R)$
- Choose $R=R_R$ at median of range.
- Cadmium Sulfide (CdS).
- Cheap, low current.
- $T_{RC} = (R+R_R)*C_{l.}$
- Typically R~50-200kW.
- C~20pF.
- So, T_{RC}~20-80uS.
- f_{RC} ~ 10-50kHz.



Source: Forrest Brewer

Review: Many other common sensors (some digital)

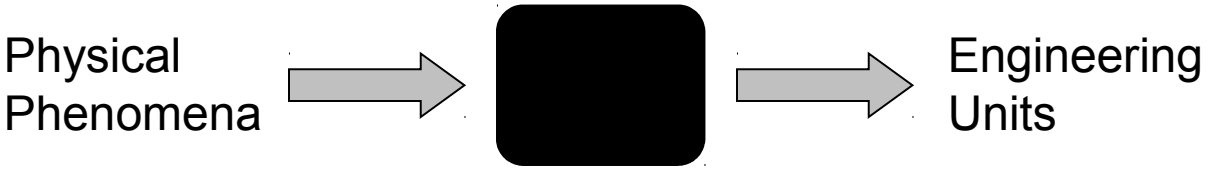
- Force
  - Strain gauges - foil, conductive ink
  - Conductive rubber
  - Rheostatic fluids
    - Piezorestive (needs bridge)
  - Piezoelectric films
  - Capacitive force
- Sound
  - Microphones: current or charge
  - Sonar: usually piezoelectric
- Position
  - Switches
  - Shaft encoders
  - Gyros
- Atmospheric pressure

Review: Many other common sensors (some digital)

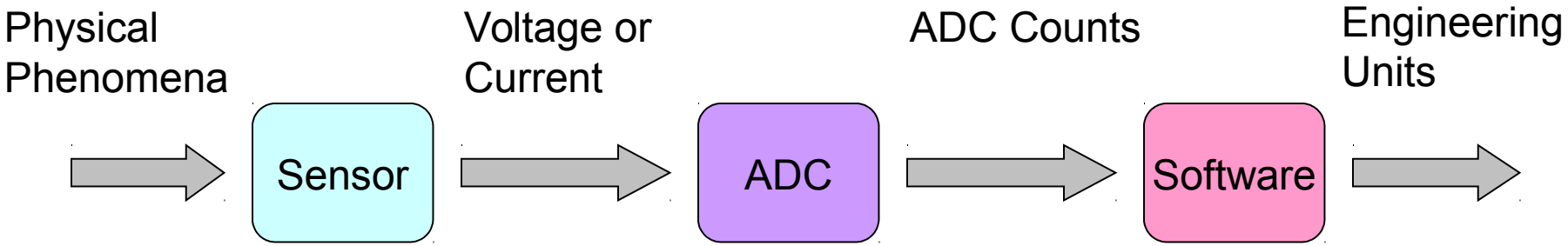
- Acceleration
  - MEMS
  - Pendulum
- Monitoring
  - Battery energy
  - Motor velocity
  - Temperature
- Field
  - Antenna
  - Magnetic
    - Hall effect
    - Flux gate
- Location
  - Permittivity
  - Dielectric
- Conductivity
- Many, many more

Review: Analog to digital

- Goal

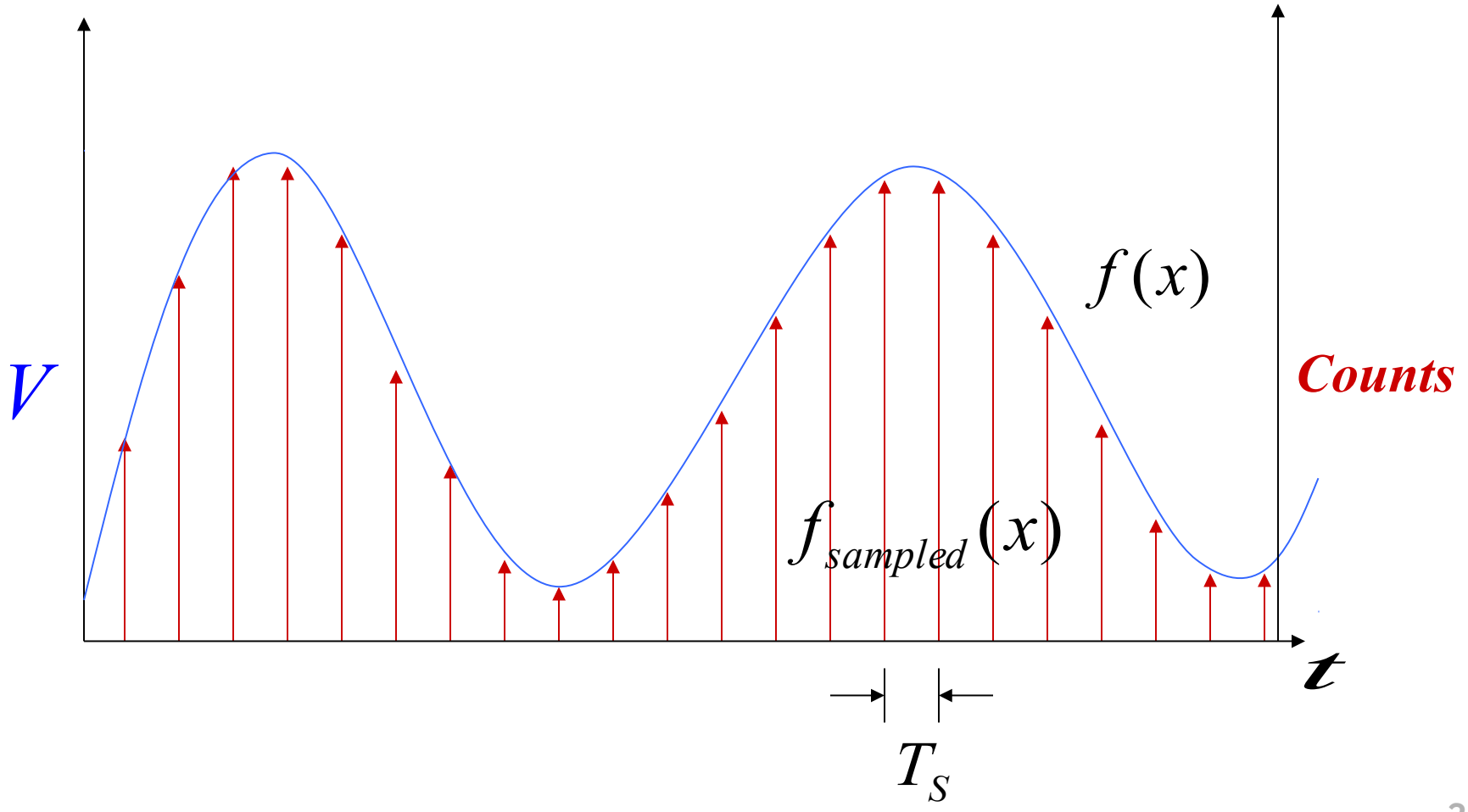


- Process



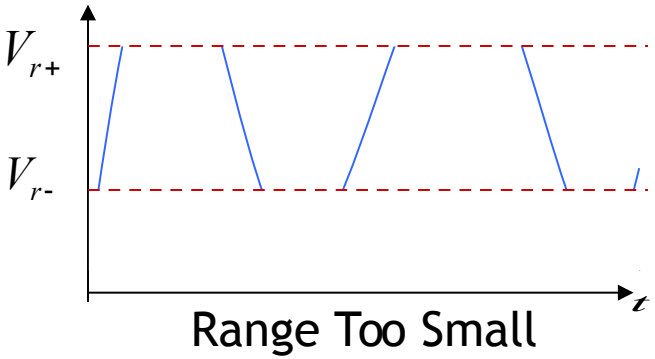
Review: Digital representation of analog signal

- Discretize in time and value.
- Time series of discrete values

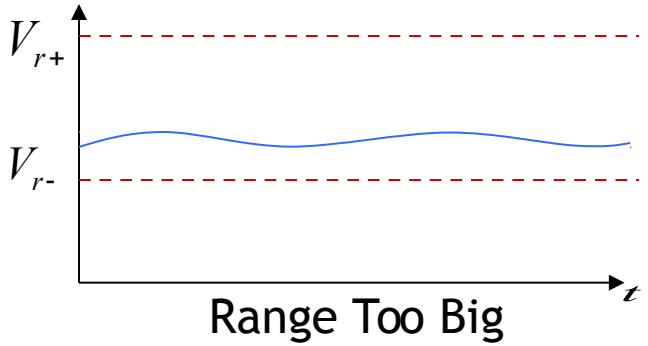


Review: Choosing the value range

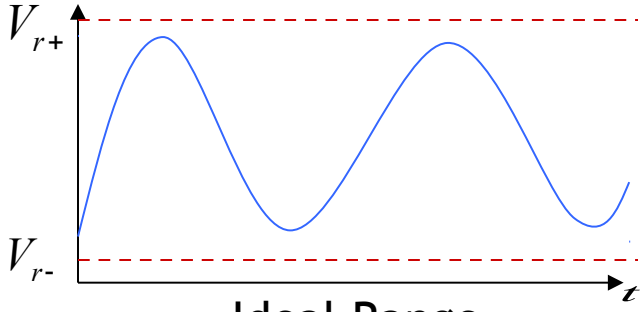
- What do the sample values represent?
  - Some fraction within the range of values



Range Too Small



Range Too Big



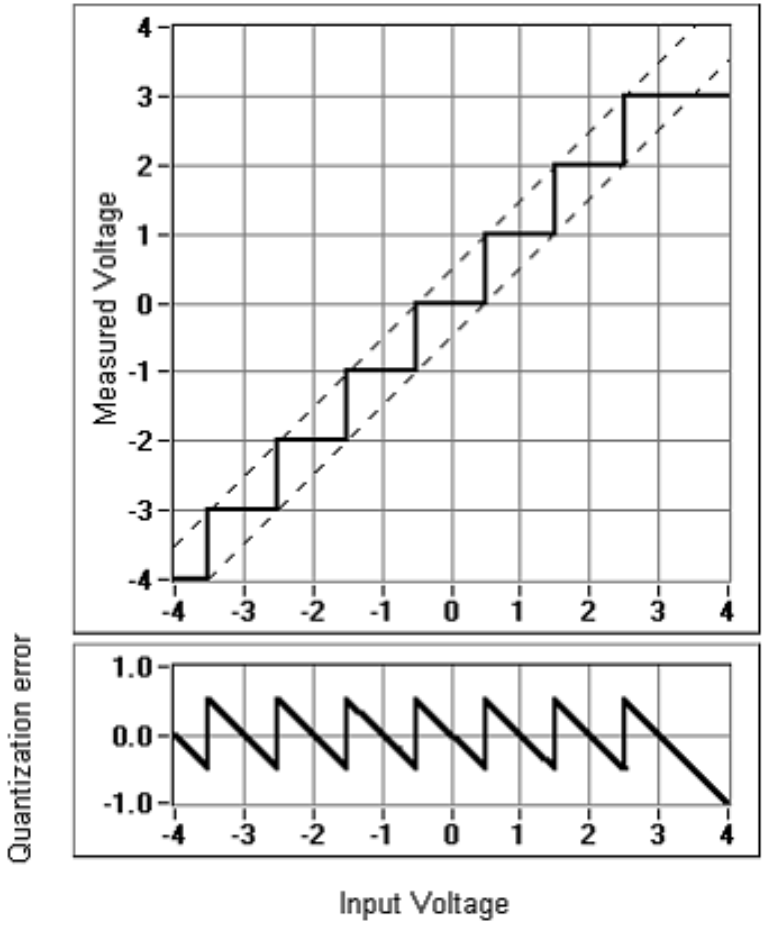
Ideal Range

Review: Choosing the value resolution

- Resolution
  - Number of discrete values that represent a range of analog values.
  - MSP430: 12-bit ADC
    - 4096 values
    - Range / 4096 = Step

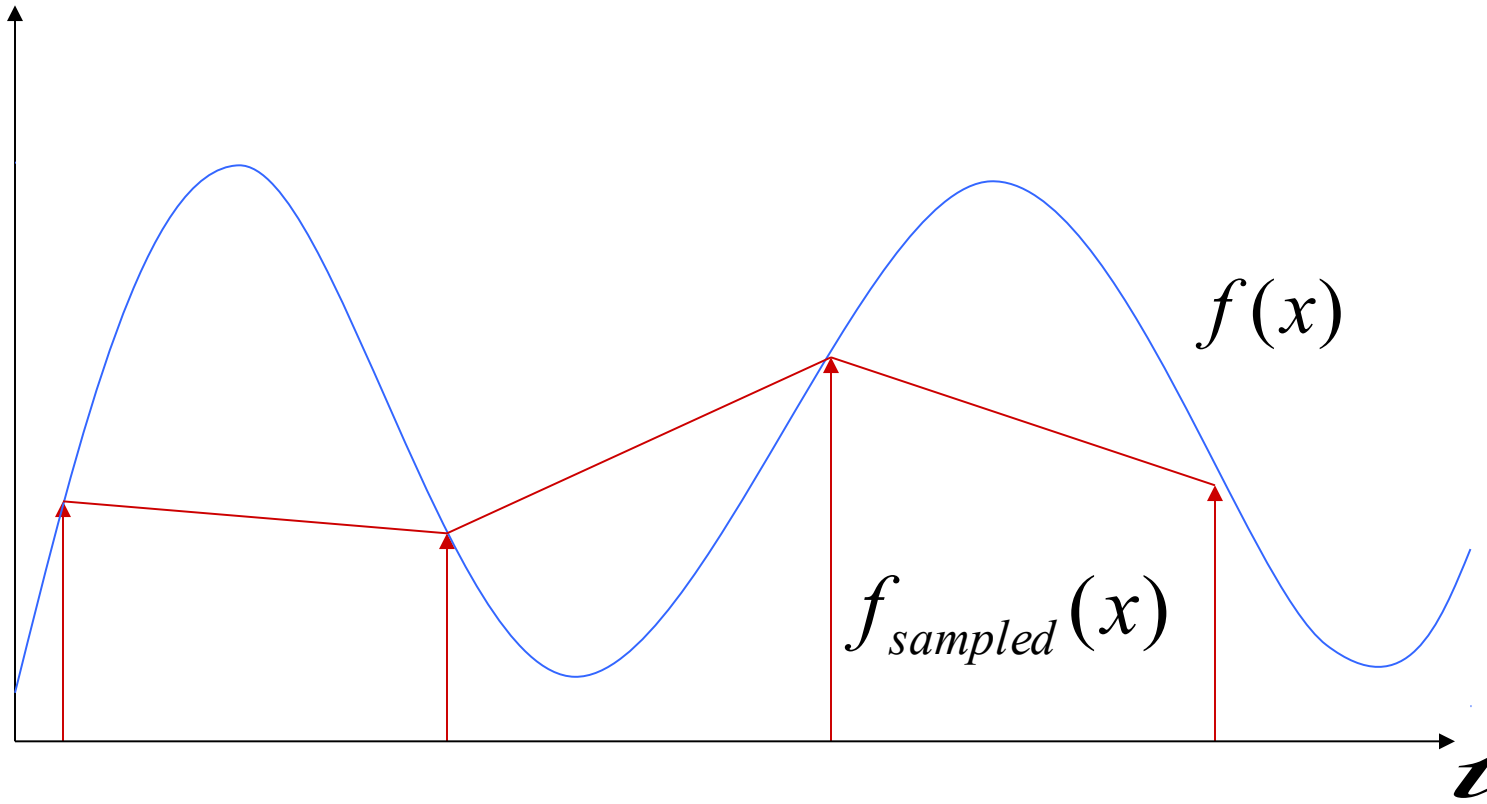
- Quantization Error
  - How far off discrete value is from actual
  - ½ LSB → Range / 8192

Larger range → lower resolution



Review: Choosing the temporal resolution

- Too low: we can't reconstruct the signal.
- Too high: waste computation, energy, resources.



Review: Shannon-Nyquist sampling theorem

- If a continuous-time signal $f(x)$ contains no frequencies higher than $f_{\max}$, it can be completely determined by discrete samples taken at a rate:

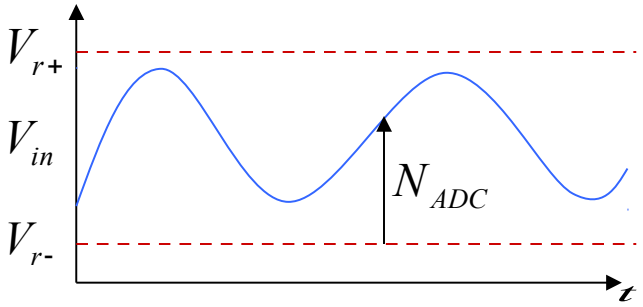
$$f_{\text{samples}} > 2 f_{\max}$$

- Example:
  - Humans can process audio signals 20 Hz – 20 KHz
  - Audio CDs: sampled at 44.1 Khz

- Caveat: additional samples can have value if signal contains high-frequency noise.
  - Allows low-pass filter to improve accuracy at cost of decreased effective samping rate.

Review: Converting between voltages, ADC counts, and engineering units

- Converting: ADC counts → Voltage



$$N_{ADC} = 4095 \times \frac{V_{in} - V_{r-}}{V_{r+} - V_{r-}}$$

$$V_{in} = N_{ADC} \times \frac{V_{r+} - V_{r-}}{4095}$$

- Converting: Voltage → Engineering Units

$$V_{\text{TEMP}} = 0.00355(\text{TEMP}_{\text{C}}) + 0.986$$

$$\text{TEMP}_{\text{C}} = \frac{V_{\text{TEMP}} - 0.986}{0.00355}$$

Review: A note about sampling and arithmetic

- Common error converting values

$$V_{\text{TEMP}} = N_{ADC} \times \frac{V_{r+} - V_{r-}}{4095} \qquad \text{TEMP}_{\text{C}} = \frac{V_{\text{TEMP}} - 0.986}{0.00355}$$

```
float vtemp = adccount/4095 * 1.5;
float tempc = (vtemp-0.986)/0.00355;
```

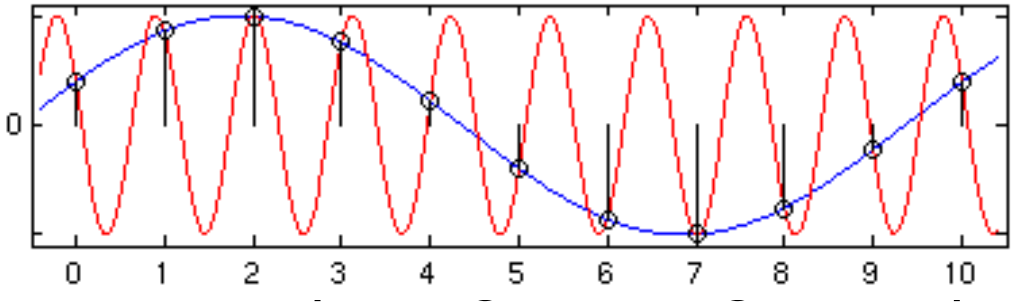
Learn associativity and type conversion rules of ANSI C.

- Fixed point operations
  - Overflow and underflow dangers complicate design.
  - This is deep. Talk to me if you want to try it.

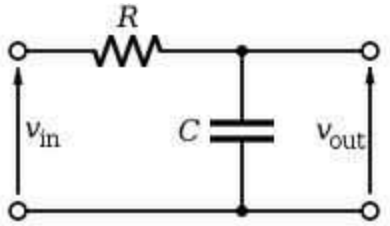
- Floating point operations
  - Often software emulated.
  - Often slow, power-hungry on embedded processors.

Use anti-aliasing filters on ADC inputs to ensure that Shannon-Nyquist is satisfied

- Aliasing
  - Different frequencies are indistinguishable when they are sampled.



- Cond s filter
  - Removes high-frequency components
  - (a.k.a. anti-aliasing filter)

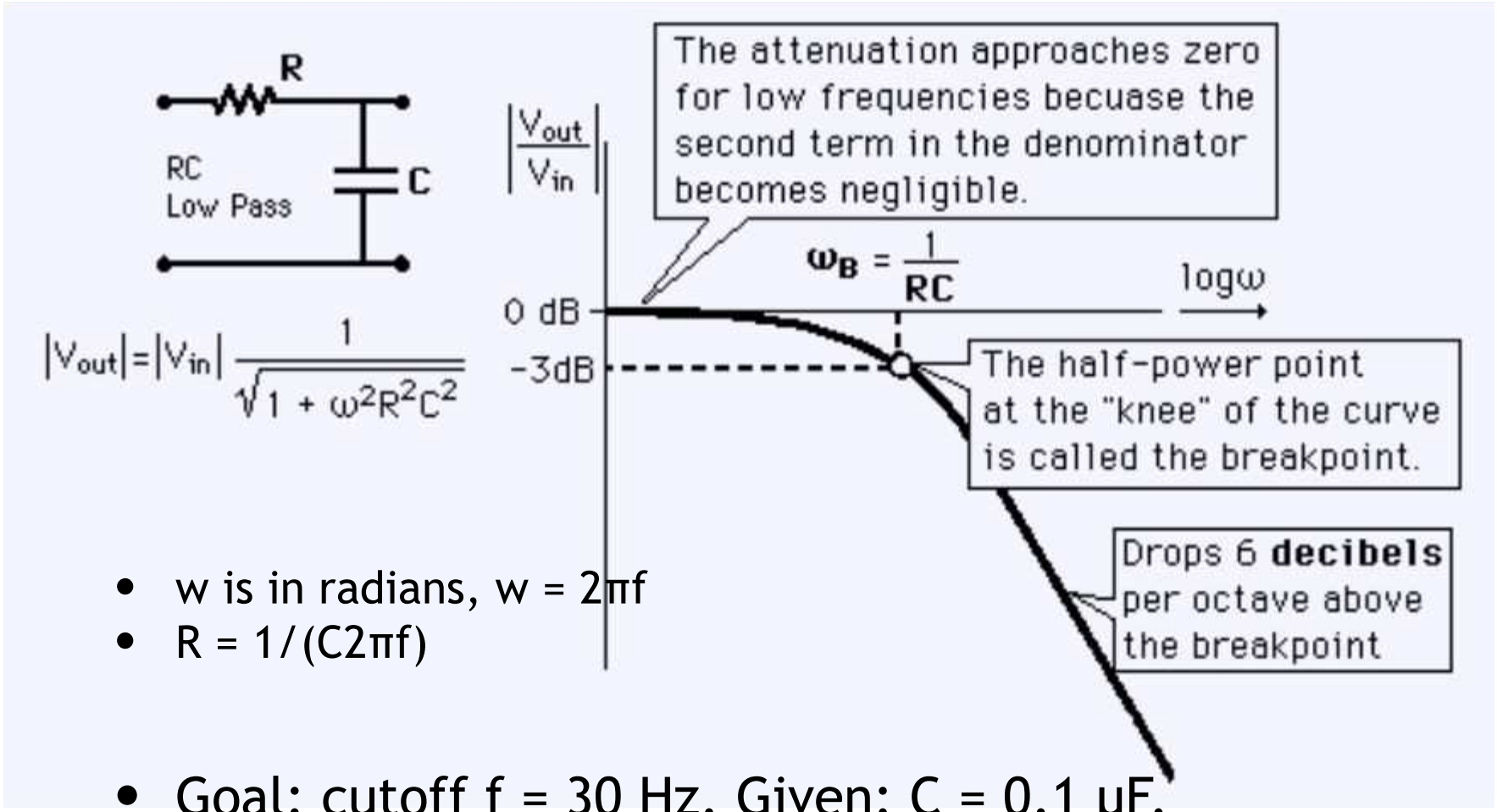


Do I really need to condition my input signal?

- Often.

- Many ADCs have analog filter built in.
- Those filters typically have a cut-off frequency just above ½ their ***maximum*** sampling rate.
- Which is great if you are using the maximum sampling rate, less useful if you are sampling at a slower rate.

Designing the anti-aliasing filter



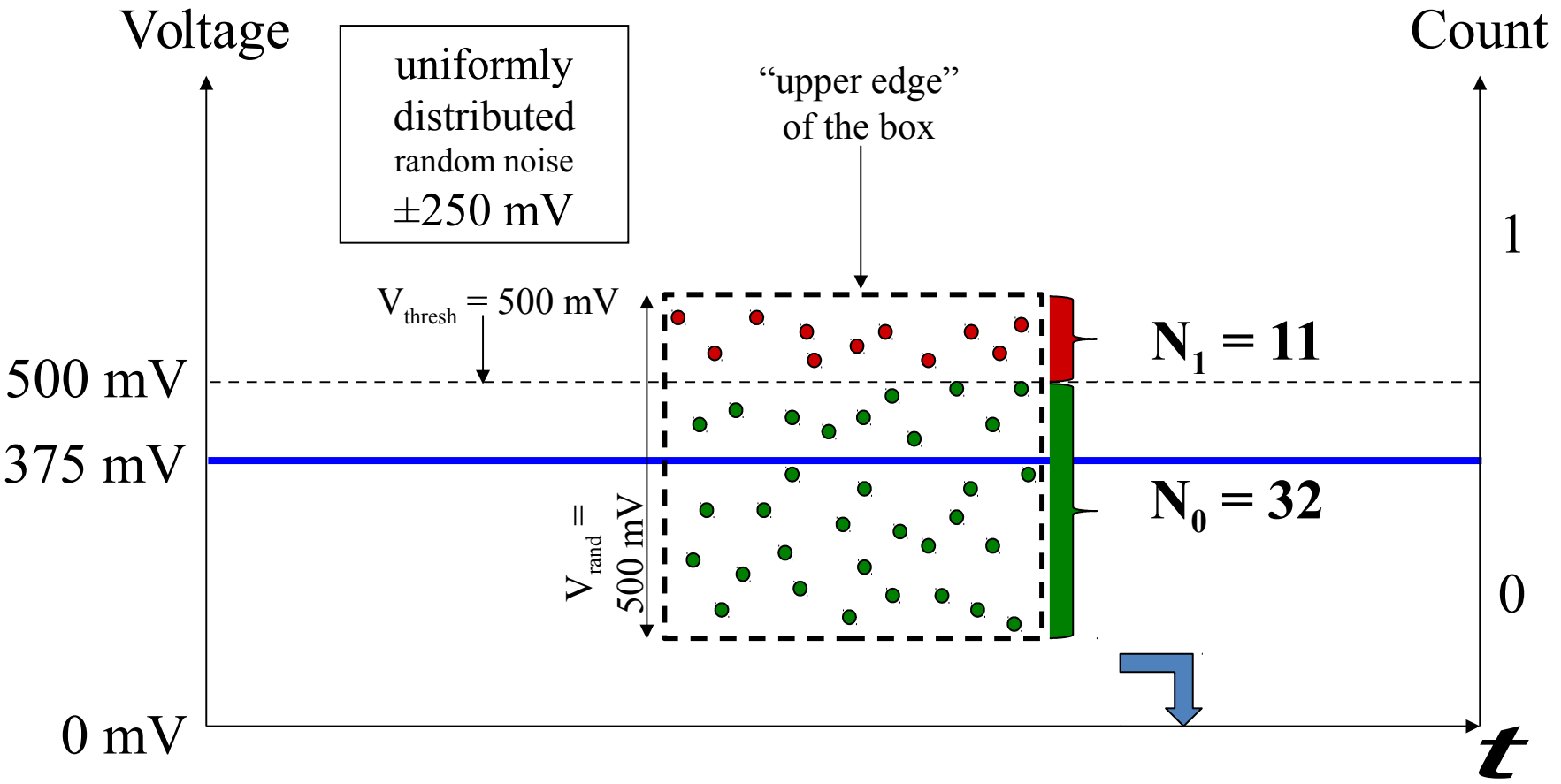
- w is in radians, w = 2πf
- R = 1/(C2πf)

- Goal: cutoff f = 30 Hz. Given: C = 0.1 μF.
- Question: R = ?

Oversampling

- Can intentionally introduce or leave high-frequency noise and oversample.
- Reduces quantization error.

Oversampling a 1-bit ADC w/ noise & dithering (cont)



Note:

N_1 is the # of ADC counts that = 1 over the sampling window

N_0 is the # of ADC counts that = 0 over the sampling window

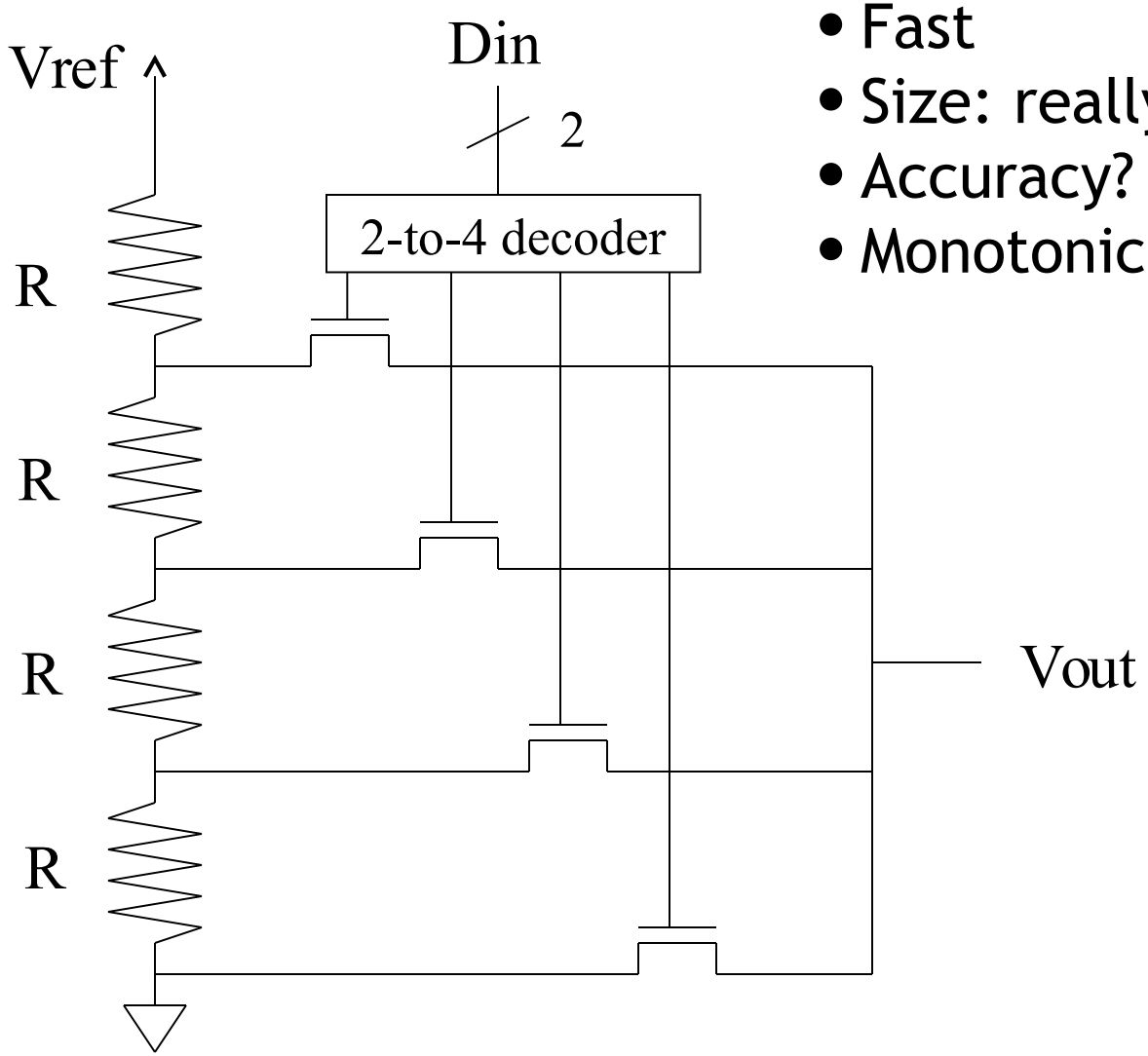
Lots of other issues

- Might need anti-imaging filter.
  - Especially during analog signal reconstruction.
  - Remove high-f components from stair stepping.
- Cost and power play a role.
- Might be able to avoid ADCs/DACs.
  - E.g., PWM.

Outline

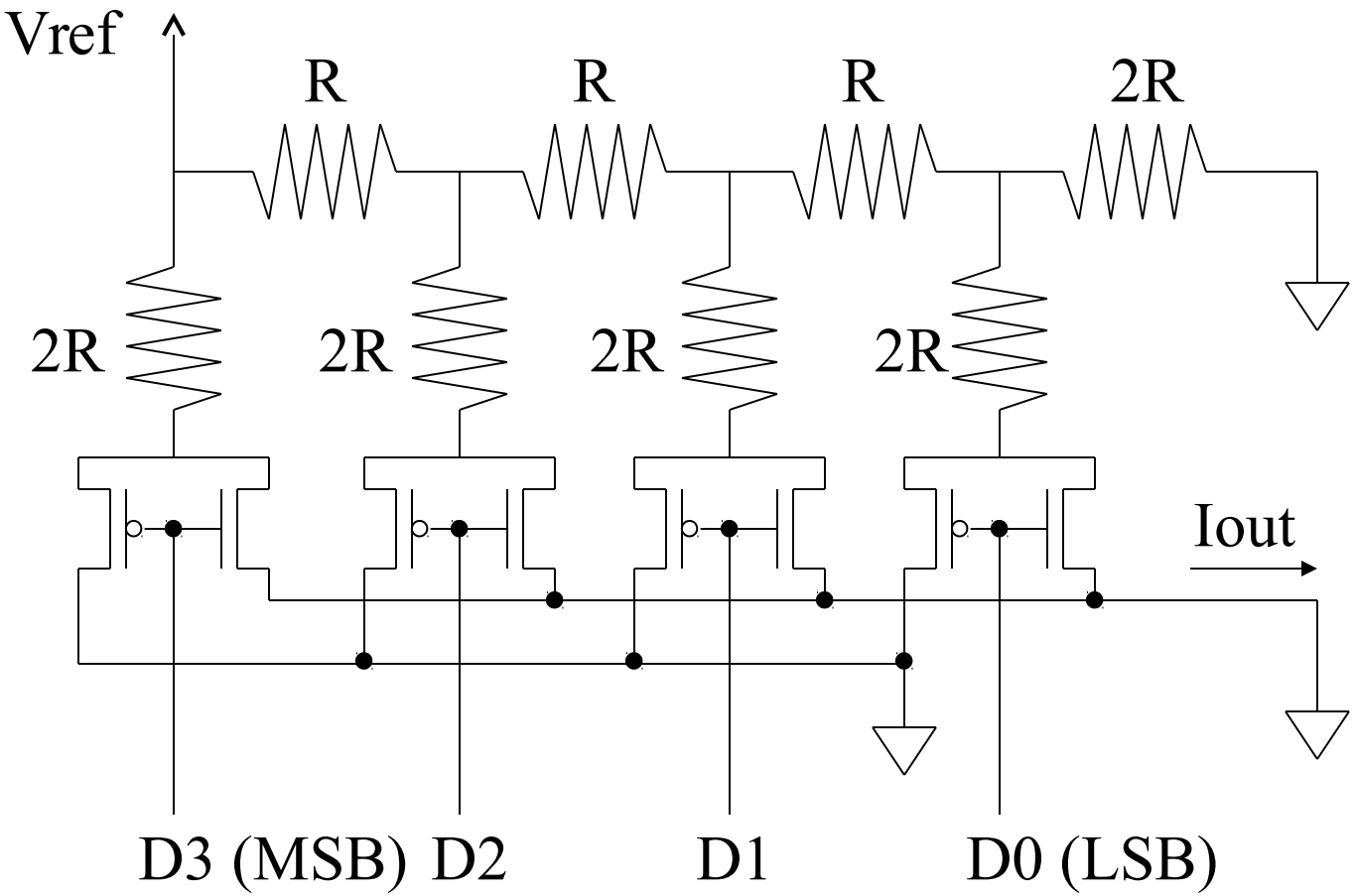
- ~~Context and review~~
- ~~Project schedule~~
- ~~Timer implementation~~
- ADCs and DACs
  - ~~Fundamentals~~
  - Operation

DAC #1: voltage divider



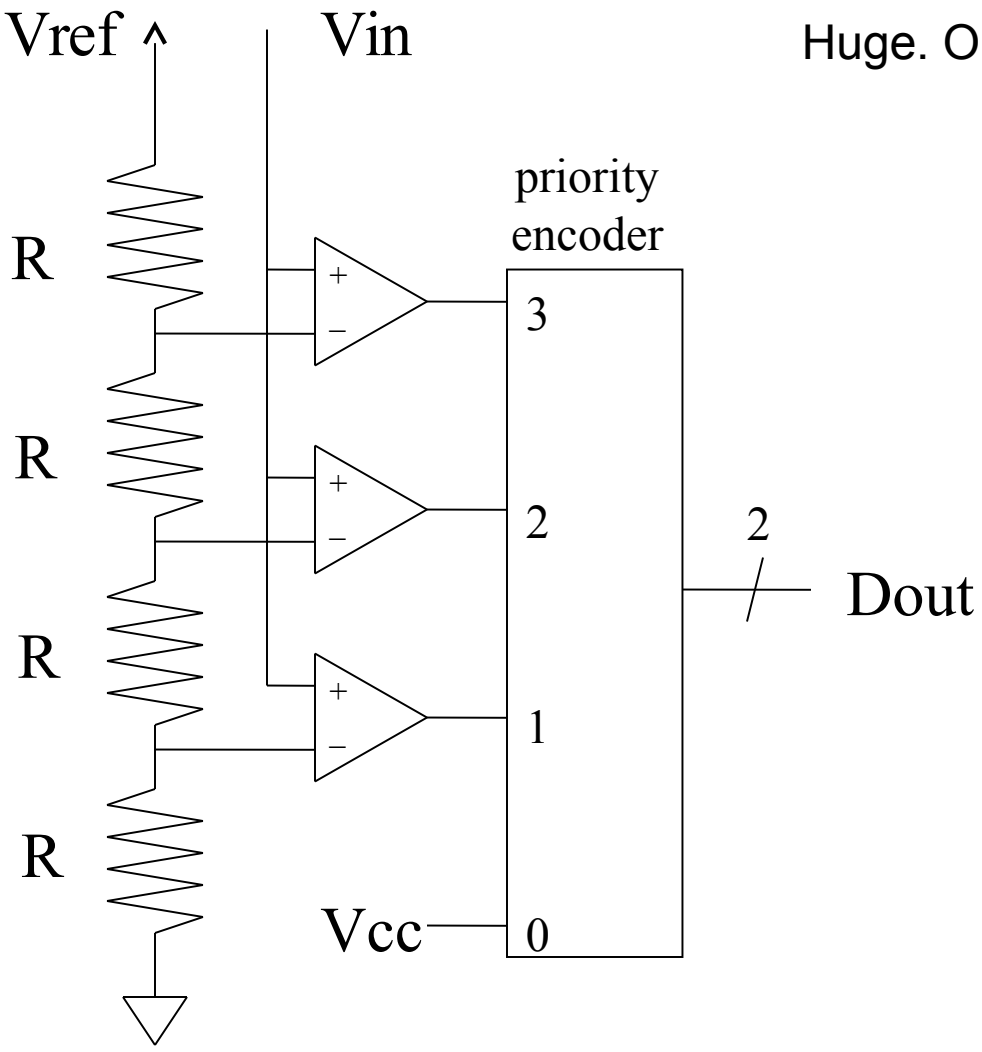
- Fast
- Size: really big: $O(2^n)$
- Accuracy?
- Monotonicity?

DAC #2: R/2R ladder



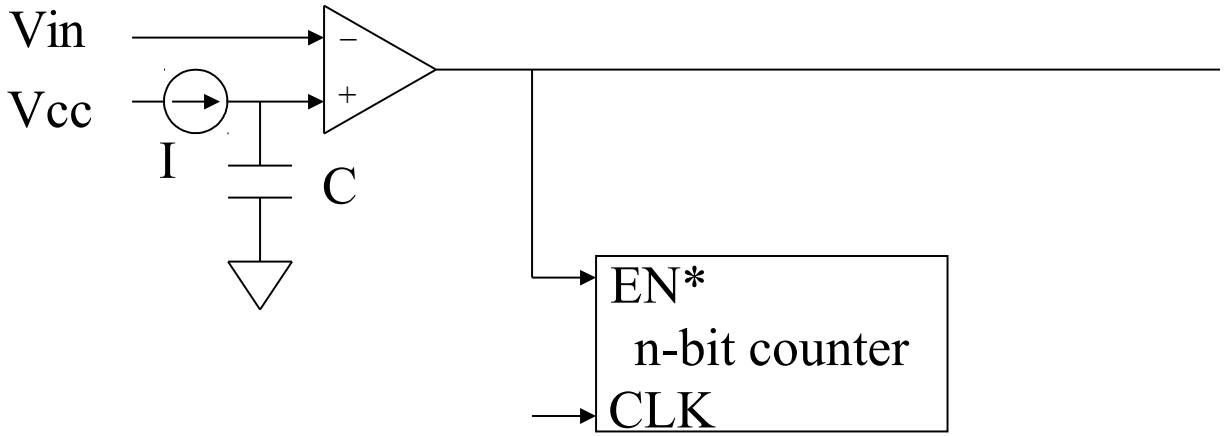
- Size: small, O(n)
- Accuracy?
- Monotonicity? (Consider 0111 -> 1000)

ADC #1: flash



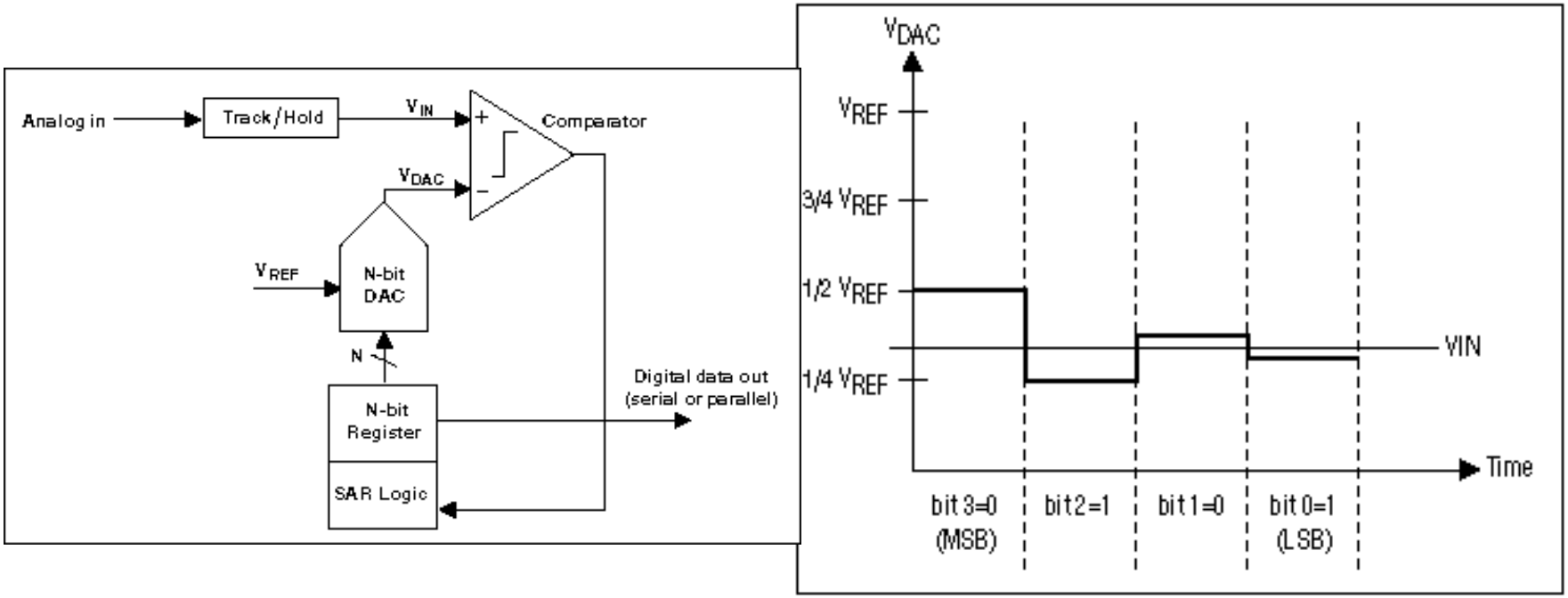
Huge. $O(2^n)$.

ADC #2: single-slope integration



- Start: Reset counter, discharge C.
- Charge C at fixed current I until Vc > Vin . How should C, I, n, and CLK be related?
- Final counter value is Dout.
- Slow: conversion may take several milliseconds.
  - O(2n)
- Good differential linearity (dI/dO)
- Absolute linearity depends on precision of C, I, and clock.

ADC #3: successive approximation

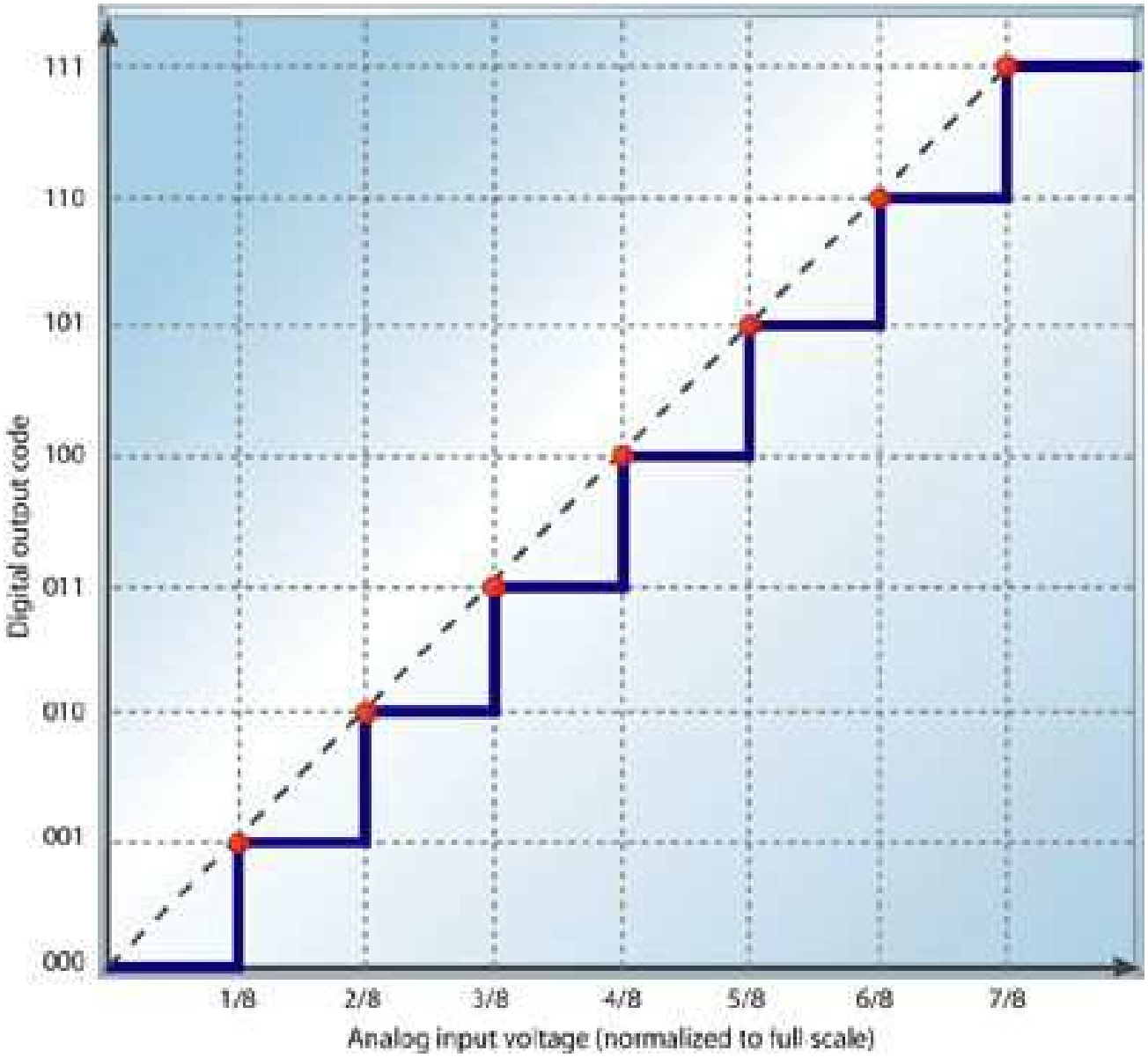


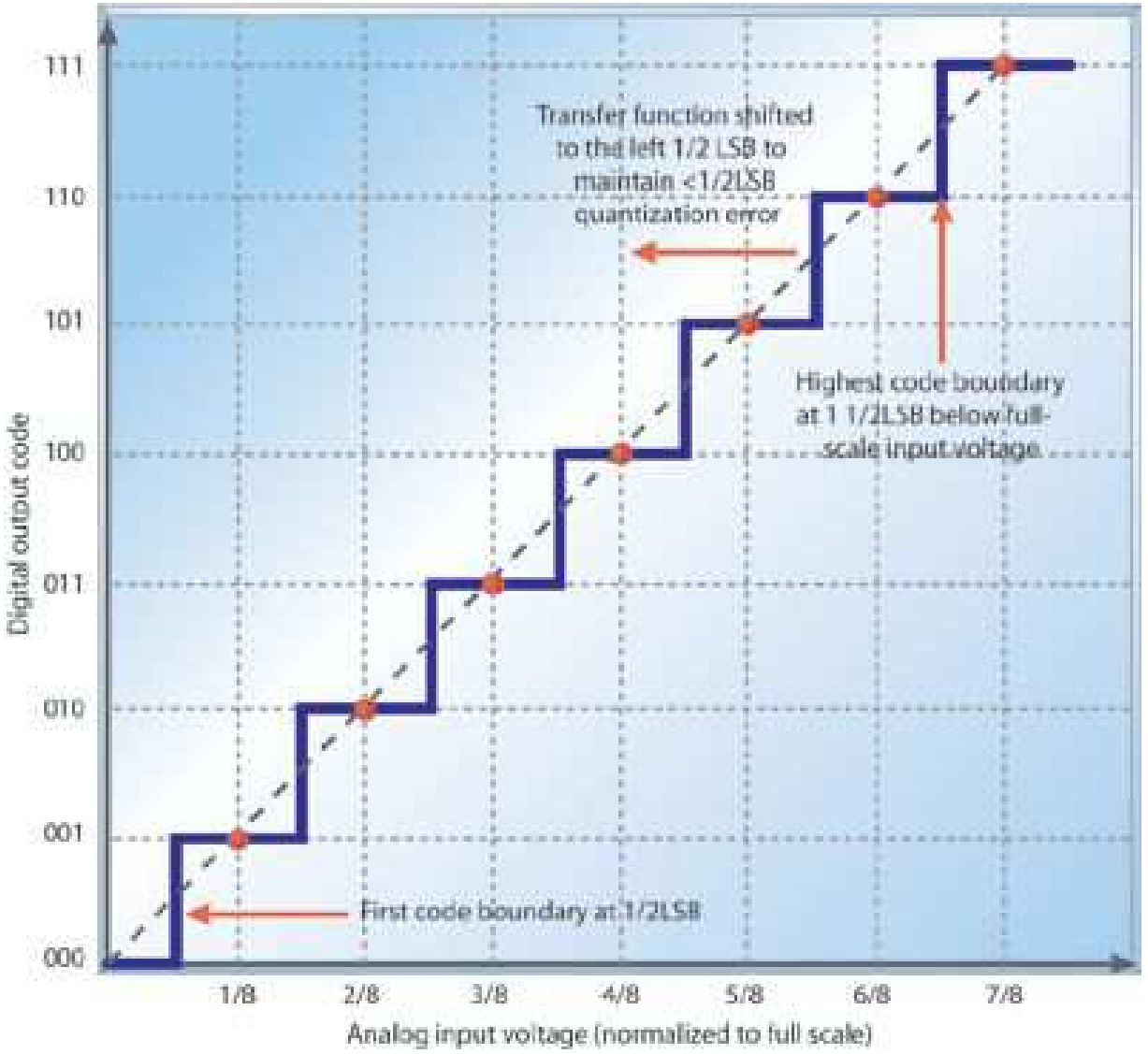
1 Sample → Multiple cycles

- Uses DAC for guessing.
- Faster: O(n)
- Goes from MSB to LSB.
- Not good for high-speed ADCs.

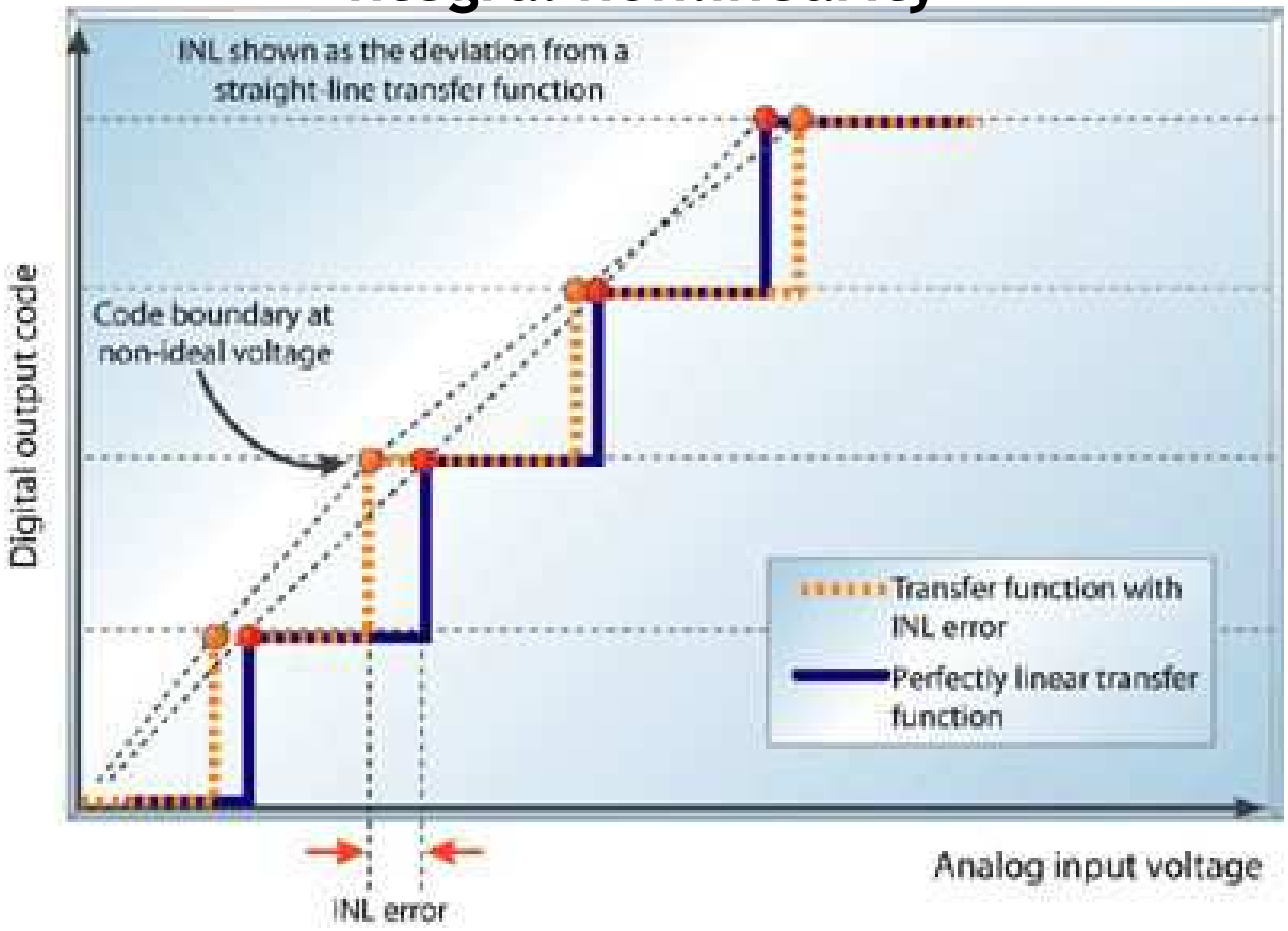
Errors and ADCs

- Figures and some text from:
  - Understanding analog to digital converter specifications. By Len Staller
  - http://www.embedded.com/showArticle.jhtml?articleID=60403334
- Key concept here is that the specification provides *worst case* values.


111
110
101
100
011
010
001
000
Digital output code
1/8
2/8
3/8
4/8
5/8
6/8
7/8
Analog input voltage (normalized to full scale)

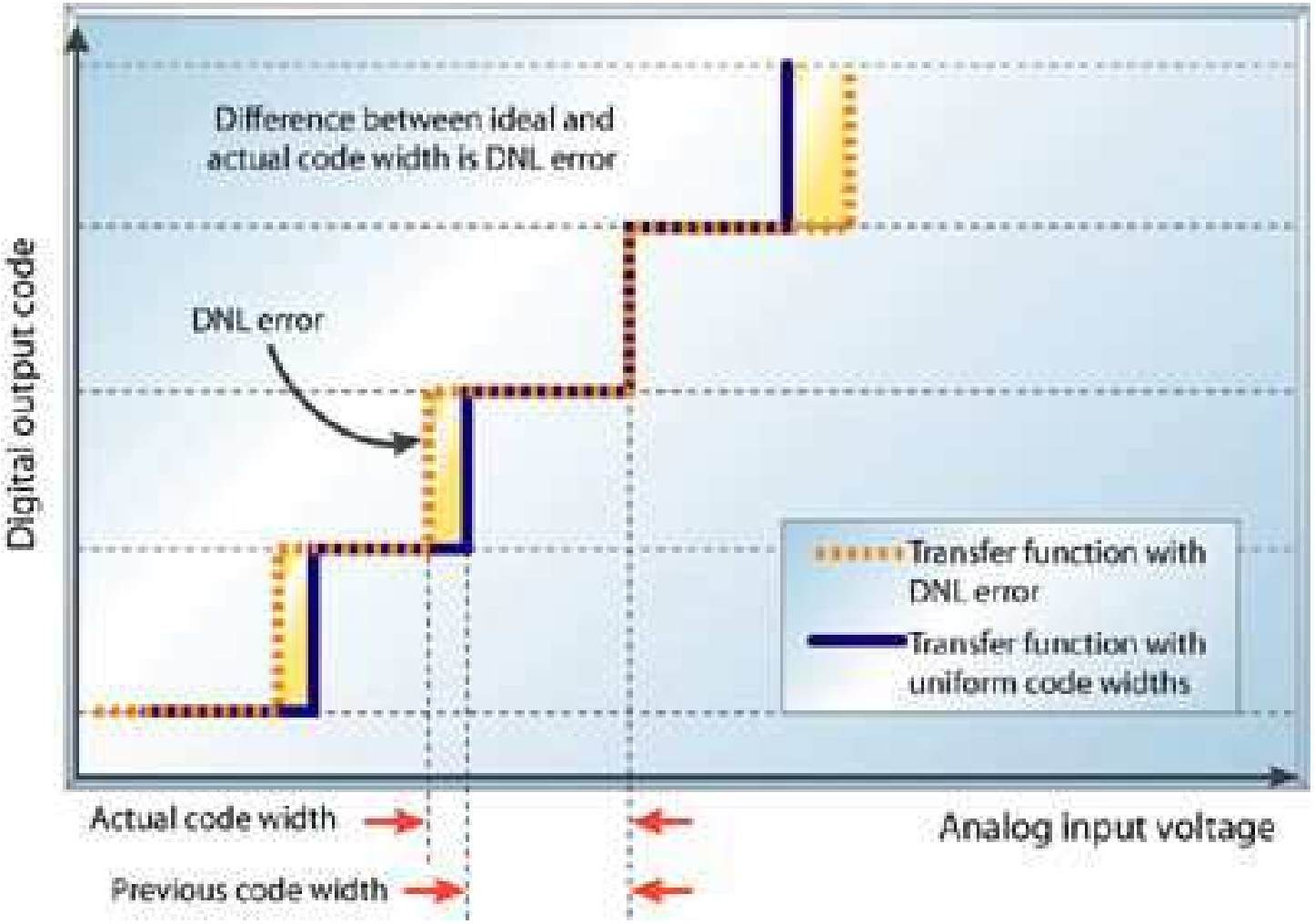

111
110
101
100
011
010
001
000
Digital output code
Transfer function shifted to the left 1/2 LSB to maintain <1/2LSB quantization error
Highest code boundary at 1 1/2LSB below full-scale input voltage
First code boundary at 1/2LSB
1/8
2/8
3/8
4/8
5/8
6/8
7/8
Analog input voltage (normalized to full scale)

Integral nonlinearity



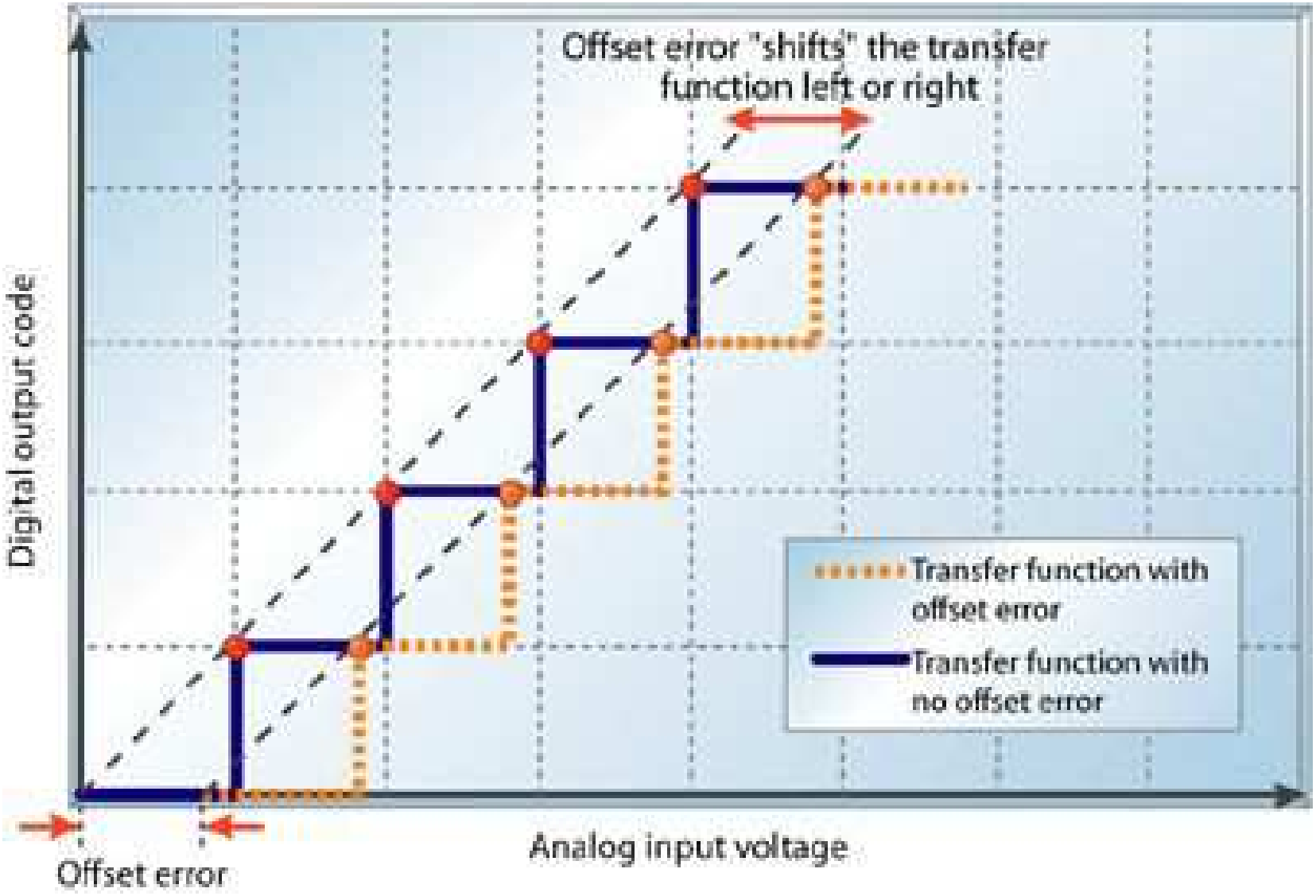
The *integral nonlinearity (INL) is the deviation of an ADC's transfer function from a* ***straight line***. This line is often a best-fit line among the points in the plot but can also be a line that connects the highest and lowest data points, or endpoints. INL is determined by measuring the voltage at which all code transitions occur and comparing them to the ideal. The difference between the ideal voltage levels at which code transitions occur and the actual voltage is the INL error, expressed in LSBs. INL error at any given point in an ADC's transfer function is the accumulation of all DNL errors of all previous (or lower) ADC codes, hence it's called *integral* nonlinearity.

Differential nonlinearity

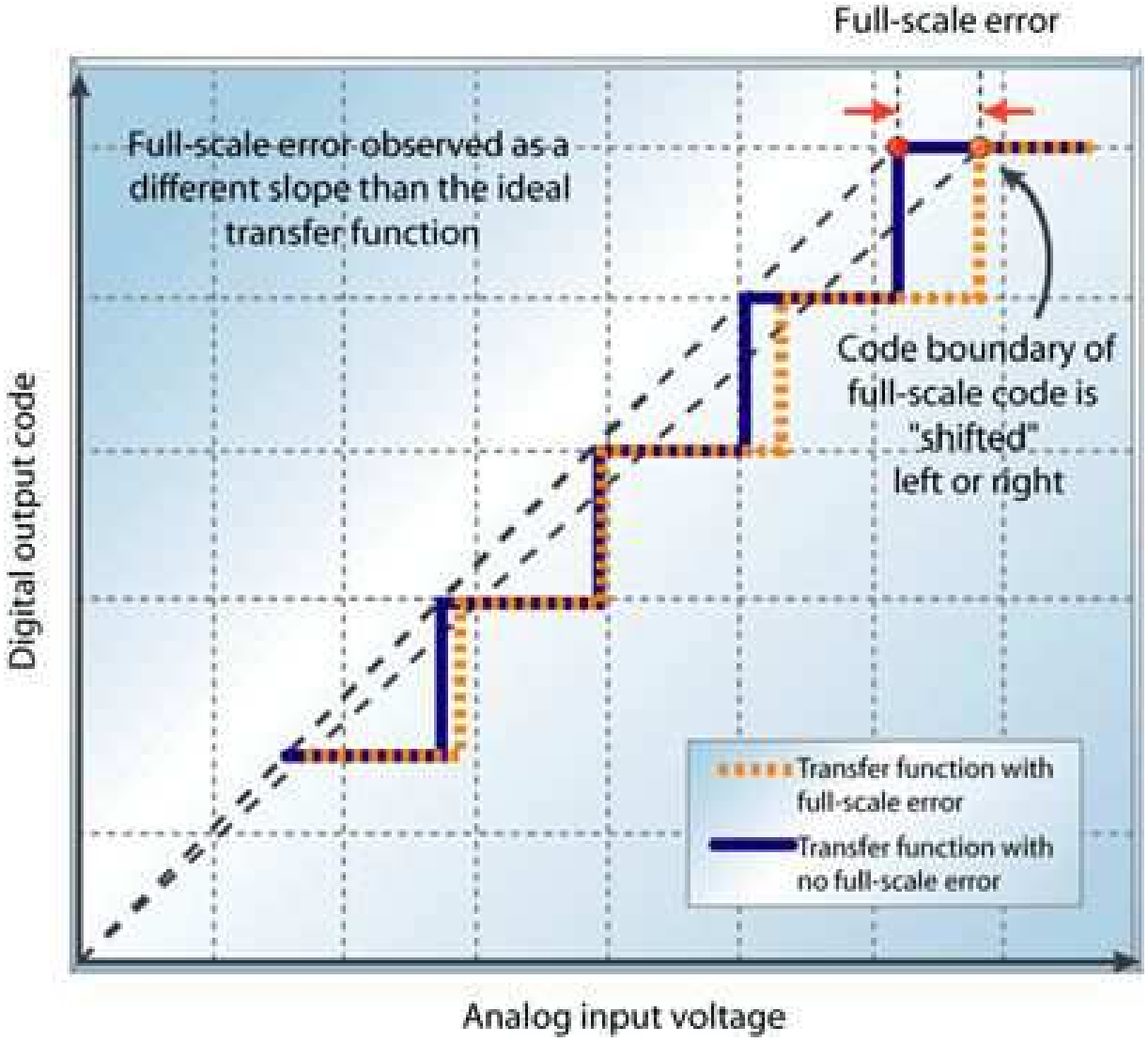


DNL is the worst cases variation of actual step size vs. ideal step size.

It's a promise it won't be worse than X.


Offset error "shifts" the transfer function left or right
Digital output code
Transfer function with offset error
Transfer function with no offset error
Analog input voltage
Offset error

Full-scale error is also sometimes called “gain error”



Full-scale error is the difference between the ideal code transition to the highest output code and the actual transition to the output code when the offset error is zero.

Errors

- Errors in a specification are bad.
  - So if you have an INL of ±0.25 LSB, you "know" that the device will never have more than 0.25 LSB error from its ideal value.
  - That of course assumes you are operating within the specification.
    - Temperature, input voltage, input current available, etc.
- Integral nonlinearity and differential nonlinearity are important.
  - Should know what full-scale error is.
  - Can compensate in software.
    - Where is best place to compensate?