CSE190 Fall 2023

Lecture 17

Interfacing with The Analog World

Wireless Embedded Systems

Aaron Schulman
What does digital mean?

• Digital computer / microcontroller
• Digital watch
• Digital accelerometer
• Digital TV
• Digital cell phone?

• Operates on discrete signal “e.g., 0s and 1s”
• Not continuous!
We live in an analog world

• Everything in the physical world is an analog signal
  – Continuous!
  – Sound, light, temperature, pressure

• Need to convert into electrical signals
  – Transducers: converts one type of energy to another
    • Electro-mechanical, Photonic, Electrical, ...
  – Examples
    • Microphone/speaker
    • Thermocouples
    • Accelerometers
Going from analog to digital

• What we want...
  Physical Phenomena (Continuous) → Engineering Units (Discrete)

• How we get there?
  Physical Phenomena (Continuous) → Sensor → Voltage or Current (Continuous) → ADC → ADC Counts (Discrete) → Software → Engineering Units
Representing an analog signal in a digital microcontroller

- How do we represent an analog signal (e.g. continuous voltage)?
  - As a time series of discrete values (binary integers)
    → On MCU: read ADC data register (counts) periodically ($T_s$)
Choosing the sample rate

- What sample rate do we need?
  - Too little: we can’t reconstruct the signal we care about
  - Too much: waste computation, energy, resources
Shannon-Nyquist sampling theorem

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

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

- Example:
  - Humans can process audio signals 20 Hz – 20 KHz
  - Audio CDs: sampled at 44.1 KHz
Use anti-aliasing filters on ADC inputs to ensure that Shannon-Nyquist is satisfied

• Aliasing (sampling too slow!)
  – Different frequencies are indistinguishable when they are sampled.

• Solution: Condition the input signal using a low-pass filter
  – Removes high-frequency components
  – (a.k.a. anti-aliasing filter)
Do I really need to filter my input signal?

• Short answer: Yes.

• Longer answer: Yes, but sometimes it’s already done for you.
  – Many (most?) ADCs have a pretty good 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.
2nd digital problem: Choosing the ADC’s range (amplitude)

- Fixed (discrete) # of bits (e.g. 8-bit ADC)
- Span a particular continuous input voltage range
- What do the sample values represent?
  - Some fraction within the range of values

→ What range to use?

![Graphs showing range too small, range too big, and ideal range.](image-url)
Choosing the granularity

- Resolution
  - Number of discrete values that represent a range of analog values
  - 12-bit ADC
    - 4096 values
    - Range / 4096 = Step
      \[ \text{Larger range } \rightarrow \text{ less info / bit} \]

- Quantization Error
  - How far off discrete value is from actual
  - \( \frac{1}{2} \) LSB \( \rightarrow \) Range / 8192
    \[ \text{Larger range } \rightarrow \text{ larger error} \]
Converting between voltages, ADC counts, and engineering units

- Converting: ADC counts $\Leftrightarrow$ 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 $\Leftrightarrow$ Engineering Units

$$V_{TEMP} = 0.00355(TEMP_C) + 0.986$$

$$TEMP_C = \frac{V_{TEMP} - 0.986}{0.00355}$$
A note about sampling and arithmetic*

• Converting values in fixed-point MCUs

\[ V_{\text{TEMP}} = N_{ADC} \times \frac{V_{r+} - V_{r-}}{4095} \]

\[ \text{TEMP}_C = \frac{V_{\text{TEMP}} - 0.986}{0.00355} \]

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

\( \Rightarrow \) \textit{vtemp} = 0! **Not what you intended, even when vtemp is a float!**

\( \Rightarrow \) \textit{tempc} = -277 \text{ C}

• Fixed point operations
  – Need to worry about underflow and overflow

• Floating point operations
  – They can be costly on the embedded system
Try it out for yourself...

$ cat arithmetic.c
#include <stdio.h>

int main() {

    int adccount = 2048;
    float vtemp;
    float tempc;

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

    printf("vtemp: %f\n", vtemp);
    printf("tempc: %f\n", tempc);
}

$ gcc arithmetic.c

$ ./a.out
vtemp: 0.000000
tempc: -277.746490
Oversampling
(sampling faster than Nyquist)

One interesting trick is that you can use oversampling to help reduce the impact of quantization error.

– Let’s look at an example of oversampling plus dithering to get a 1-bit converter to do a much better job...
Oversampling a 1-bit ADC w/ noise & dithering (cont)

uniformly distributed random noise ± 250 mV

$V_{\text{thresh}} = 500 \text{ mV}$

$V_{\text{rand}} = 500 \text{ mV}$

$N_1 = 11$

$N_0 = 32$

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

• How to get more than 1-bit out of a 1-bit ADC?
• Add some noise to the input
• Do some math with the output
• Example
  – 1-bit ADC with 500 mV threshold
  – $V_{\text{in}} = 375$ mV $\rightarrow$ ADC count = 0
  – Add $\pm 250$ mV uniformly distributed random noise to $V_{\text{in}}$
  – Now, roughly
    • 25% of samples ($N_1$) ≥ 500 mV $\rightarrow$ ADC count = 1
    • 75% of samples ($N_0$) < 500 mV $\rightarrow$ ADC count = 0
Can use dithering to deal with quantization

- Dithering (introducing noise)
  - Quantization errors can result in large-scale patterns that don’t accurately describe the analog signal
  - Oversample and dither
  - Introduce random (white) noise to randomize the quantization error.
Selection of a DAC (digital to analog converter)

- **Error/Accuracy/Resolution**: Quantizing error represents the difference between an actual analog value and its digital representation. Ideally, the quantizing error should not be greater than $\pm \frac{1}{2}$ LSB.

**Output Voltage Range** -> Input Voltage Range

- **Output Settling Time** -> Conversion Time
- **Output Coding** (usually binary)