Voltage, A2D, and Piece Wise Linear

Topics

- What form are real-world signals?
- How can a computer read an analog signal?
- How can we approximate functions?

Signals in the “Real World”: Voltage

- Real world analog signals are often changes in voltage:
  - Ex: Microphone encodes sound into voltage levels

Audio: Zoomed in
Audio: Zoomed out
Voltage Ranges

- 5.0V: Some circuits (Arduino)
- 3.3V: Many circuits (BeagleBone)
- 1.8V: BeagleBone A2D ref V
- 0V: Ground

These are all DC voltage (Direct Current)
Out of the wall comes AC Voltage (Alternating Current)

Electronics Components (“Parts”)

- Many electronics components run on, manage, and work with voltages.

**Voltage Regulator:** Converts input voltage to stable output voltage.
- May fluctuate a little
- Stable

**Potentiometer:** Turning the knob adjusts the output voltage on \( V_{out} \).
- 1.8V
- 4-8V
- 3.3V

**Light Sensor:** The more light, the lower the voltage on \( V_{out} \).
- Gnd
- Gnd
- 1.8V
- Gnd

Reading a Voltage

- How can we read a signal into the computer?
  - Real world is analog voltages; computer are digital.
  - We need an analog to digital converter (ADC)
    - Also called an A2D (Analog “to” Digital)
- BeagleBone has a 12 bit A2D:
  - It reads a voltage and gives a number between 0 and \( 2^{12} - 1 = 4095 \)
  - It can sample voltages between 0V and 1.8V
    - It is easily damaged by higher voltages!

Quantization & Sampling

- **Quantization:**
  - Since it has 4096 samples over 1.8V
    - Resolution of a single bit is:
      - \( 1.8V / 4096 = 0.00044V = 0.44 \text{ mV} \)
      - This is pretty good for most applications!
- **Sample Rate:**
  - How fast the A2D can read samples
    - Need 44100 Hz (44.1kHz) for CD audio
    - BeagleBone can sample at 1.6MHz (1600kHz)
    - Some applications (reading a POT for volume) may need low sample rates (~10Hz)
BBB A2D Demo for POT

- Enable A2D in Linux (virtual cape):
  
  # echo BB-ADC > /sys/devices/platform/bone_capemgr/slots

- Change to sys file system folder:
  
  # cd /sys/bus/iio/devices/iio:device0

- Read voltage 0 (for POT):
  
  # cat in_voltage0_raw

Approximating Functions:
Piece Wise Linear

Function Approximations

- Real world functions can be hard to approximate.
  - Some approximations are computationally expensive (high-order polynomials, cubic-spline, ..)
  - Piecewise Linear (PWL)
    Approximate a function with a series of lines.

Piece Wise Linear

- Pick good points on the function f(x) to capture its shape
  - can be evenly spaced, or
  - can be specially selected points
- Between adjacent points, draw a straight light.
- The approximation f'(x) is the straight lines.
Computing Piecewise Linear

- Given an input value \( s \), use points on either side.
- Compute \( f'(s) \) by solving the point on the line.

\[
f'(s) = \left( \frac{s-a}{b-a} \right) \cdot (n-m) + m
\]

Understanding Piecewise Linear

\[
f'(s) = \left( \frac{s-a}{b-a} \right) \cdot (n-m) + m
\]

Piecewise Linear Details

- Some extra points:
  - If a reading is < min or > max data point, clip it to min & max.
  - Enter the points into a program as two arrays:

```c
#define PIECEWISE_NUM_POINTS 11
const float PIECEWISE_DoD[] = { .0, .1, ... .8, .9, 1};
const float PIECEWISE_V[] = {12.6, 12.3, ... 11.2, 11.1, 10};
```

- Make sure to use the correct data types for your calculation (possibly floating point).
- Watch for array out of bounds!

Summary

- Many sensor generate analog voltage signals.
  - Be careful that signal is in correct voltage range!
- BBB can sample voltages between 0 and 1.8V
  - 12-bit A2D: digital values between 0 and 4095
- Piecewise Linear approximates functions
  - Given a reading (on the X axis), use the selected points and straight lines to approximate desired value (on the Y axis)