Timing and Analog Input

CSE 132

Simple Timing

- Use `Thread.sleep()` in Java
  - Argument is integer number of milliseconds before the method returns
  ```java
  for (int i=0; i < endTime; i++) {
      Thread.sleep(1000);
      System.out.println(i + " seconds have elapsed");
  }
  ```
- Use `delay()` on Arduino
  - Same approach as in Java

Effects of Simple Timing

- What are possible issues with this code?
  ```java
  while (true)
      wait for 1 second
      do some work
      output results
  end while
  ```

Better Timing

- Use a free-running timer
  - `unsigned long millis()`
  - Returns # of milliseconds since reset
  - Rolls over to zero after about 50 days
- Now we can use delta time techniques
  ```java
  while (true)
      if (millis() > loopEndTime)
          loopEndTime += deltaTime
          do some work
      end if
  end while
  ```

Impact of Delta Timing

```java
while (true)
    if (millis() > loopEndTime) then
        loopEndTime += deltaTime
        do some work
        output results
    end if
end while
```
2/6/2016

FSM Diagram

• A 3-bit counter cycles from 0 to 7, and then roles over back to 0
• Consider each count value to be a “state”
• In each state, output is simply value of count
• In each state, next state is value + 1

Stoplight Controller

• NS-G: North/South Green
• NS-Y: North/South Yellow
• EW-G: East/West Green
• EW-Y: East/West Yellow
• Ped: Pedestrian Walk

Analog to Digital Conversion

• Convert physical property to voltage signal
• A/D converter on Arduino converts voltage signal to digital representation
  – 10-bit A/D converter has range 0 to 2^10 – 1 (0 to 1023) for voltage range 0 to VREF

Understanding Ranges

\[
signal = m \times \text{weight} + b \\
\text{signal} = 43 \frac{\text{mV}}{10^4} \times \text{weight} + 200 \text{mV}
\]

\[
\text{counts} = 8 \times \frac{\text{counts}}{10^4} \times \text{weight} + 40
\]

\[
\text{weight} = 0.116 \times \frac{\text{counts}}{10^4} \times \text{signal} - 4.65
\]

Noisy Analog Signals

• Noise is ever present in analog signals
• For stable signal, quick fix is to average several readings

\[
\text{avg} = \frac{1}{N} \sum_{i=1}^{N} \text{A/D input}_i
\]

What about fractions?

• Positional number systems work on both sides of the decimal point (radix point).

• If radix is r (n integer digits, m fractional digits):
\[
\text{val} = a_{n-1} \cdot r^{n-1} + a_{n-2} \cdot r^{n-2} + \ldots + a_0 \cdot r^0 + a_1 \cdot r^1 + a_2 \cdot r^2 + \ldots + a_{m-1} \cdot r^m
\]

• e.g., \(wx.yz_{16} = w \cdot 16 + x + y \cdot 16^{-1} + z \cdot 16^{-2}\)
  or \(wx.yz_2 = w \cdot 2 + x + y \cdot 2^{-1} + z \cdot 2^{-2}\)
Two kinds of numbers

- **Integers** – radix point is assumed to be at the far right end of the digits:
  - E.g. 01001110.

- **Fixed point** – radix point is at a given, fixed location:
  - E.g. 0100.1110
  - 0.1001110 is a common representation on digital signal processors

Q notation

- Qn.m means a number with n+m bits (digits), n integer and m fractional. Sign bit is often in addition to this.
  - E.g., Q3.4 for 0100.1100, with value 4.75
  - Qm means a number with m+1 bits, m are fractional
  - E.g., Q3 notation would have 4 bits and the following values
    - wxyz = w.xyz = w ∙ (1/2) + x ∙ (1/4) + y ∙ (1/8) + z ∙ (1/16)
    - range is now -1 to +7/16, with resolution 1/16

Floating point representation

What about the reals? Use scientific notation.

In base 10: x ∙ 10^y = 0.32 ∙ 10^-3 = 0.00032

In base 2: x ∙ 2^y called floating point
  - ^| exponent
  - ^| mantissa

IEEE Floating Point

- Limited range of x and y (fixed # of bits) means we cannot represent every real number exactly

- IEEE std. 754 describes a standard form for floating point number representations
  - Single precision is 32 bits in size
  - Double precision is 64 bits in size

Single precision (32 bits)

<table>
<thead>
<tr>
<th>s</th>
<th>exponent (e)</th>
<th>fraction (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

value = (-1)^s ∙ 2^e-127 ∙ 1.f
  - hidden “1”

range = ± 2 ∙ 10^±38

- s = 0, e = 0, f = 0 ⇒ value = zero
- e = 255, f = 0 ⇒ value = (-1)^s ∙ infinity
- e = 255, f ≠ 0 ⇒ value = “not a number” triggers exception
- e = 0, f ≠ 0 ⇒ denormalized
  - value = (-1)^s ∙ 2^-126 ∙ 0.f
  - hidden “0”

- Note use of sign-magnitude for entire number, and excess notation (excess 127) for exponent
Double precision (64 bits)

<table>
<thead>
<tr>
<th>63 62</th>
<th>52 51</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>exponent (e)</td>
<td>fraction (f)</td>
</tr>
</tbody>
</table>

11 bits 52 bits

value = \((-1)^{s} \times 2^{e-1023} \times 1.f\)  
Up hidden “1”

range = \(\pm 2 \times 10^{\pm 308}\)

e = 0, f \neq 0 \Rightarrow \text{denormalized}

value = \((-1)^{s} \times 2^{-1022} \times 0.f\)

Studio Today

• Come to Urbauer labs
• Form groups of 2 to 4
• Do the exercises
  – Red, Green, and Yellow LEDs available in lab
  – OK to use RGB LED for pedestrian signal
  – Explore finite-state machines and delta timing
• Get signed out by a TA
• You are welcome to continue work on assignment 2, once you finish studio exercises