Timing and Analog Input

CSE 132

Simple Timing

- Use Thread.sleep() in Java
  - Argument is integer number of milliseconds before the method returns

```
for (int i=0; i < endTime; i++) {
    Thread.sleep(1000);
    System.output.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?

```
while (true)
    wait for 1 second
    do some work
    output results
end while
```

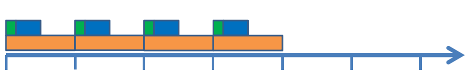
Better Timing

- 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

```
while (true)
    if (millis() > loopEndTime) then
            loopEndTime += deltaTime
            do some work
    end if
end while
```

Impact of Delta Timing

```
while (true)
    if (millis() > loopEndTime) then
            loopEndTime += deltaTime
            do some work
            output results
    end if
end while
```

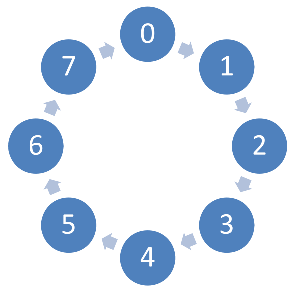


Finite State Machine (FSM)

- Useful concept for today's studio software
- Used extensively in hardware and software systems design and analysis
- Explicitly enumerate (i.e., list) all of the "states" that our design can have, and articulate:
  - What happens (e.g., is output) in each state
  - What state is next under what conditions
- "States" represent what our design wishes to remember

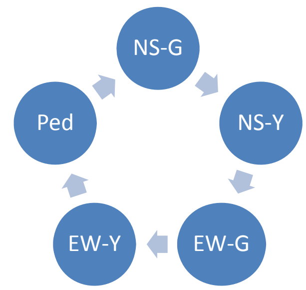
FSM Diagram

- 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



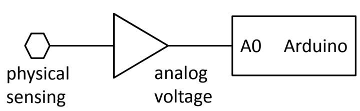
Stoplight Controller

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

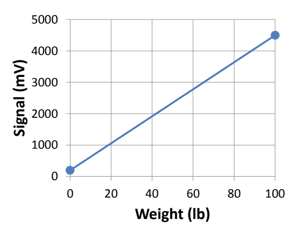


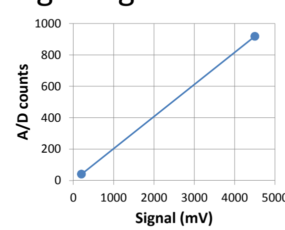
Analog to Digital Conversion

- 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 V_{REF}



Understanding Ranges





$$signal = m \times weight + b$$

$$signal = 43\frac{mV}{lb} \times weight + 200mV$$

$$counts = 8.6\frac{cnt}{lb} \times weight + 40$$

$$counts = m \times signal + b$$

$$counts = 0.2\frac{cnt}{mV} \times signal + 0$$

$$weight = 0.116\frac{lb}{cnt} \times counts - 4.65$$

Noisy Analog Signals

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

$$avg = \frac{1}{N}\sum_{1}^{N} A/D\ input_i$$

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):

val = $a_{n-1} \cdot r^{n-1} + a_{n-2} \cdot r^{n-2} + \ldots + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + a_{-m} \cdot r^{-m}$

- 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 = $w \cdot (-1) + x \cdot (1/2) + y \cdot (1/4) + z \cdot (1/8)$
  - range is now -1 to +7/8, with resolution 1/8

Floating point representation

What about the reals? Use scientific notation.

In base 10: $x \cdot 10^y$ $0.32 \times 10^{-3} = 0.00032$

In base 2: $x \cdot 2^y$ called floating point

x: mantissa, y: exponent

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)

31	30 … 23	22 … 0
s	exponent (e)	fraction (f)
1	8 bits	23 bits

value = $(-1)^s \times 2^{e-127} \times \underline{1}.f$

↑ hidden "1"

range = $\pm\ 2 \times 10^{\pm 38}$

31	30 … 23	22 … 0
s	exponent (e)	fraction (f)

- s = 0, e = 0, f = 0 ⇒ value = zero
- e = 255, f = 0 ⇒ value = $(-1)^s \times$ infinity
- e = 255, f ≠ 0 ⇒ value = "not a number" triggers exception
- e = 0, f ≠ 0 ⇒ denormalized
  value = $(-1)^s \times 2^{-126} \times \underline{0}.f$
  ↑ hidden "0"
- Note use of sign-magnitude for entire number, and excess notation (excess 127) for exponent

Double precision (64 bits)

```
 63 62          52  51                    0
| s | exponent (e) | fraction (f)         |
  1     11 bits          52 bits
```

value = $(-1)^s \times 2^{e-1023} \times \underline{1}.f$
↑ hidden "1"

range = $\pm\ 2 \times 10^{\pm 308}$
$e = 0,\ f \neq 0 \Rightarrow$ denormalized
value = $(-1)^s \times 2^{-1022} \times \underline{0}.f$

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