

# Physics 120B: Lecture 2

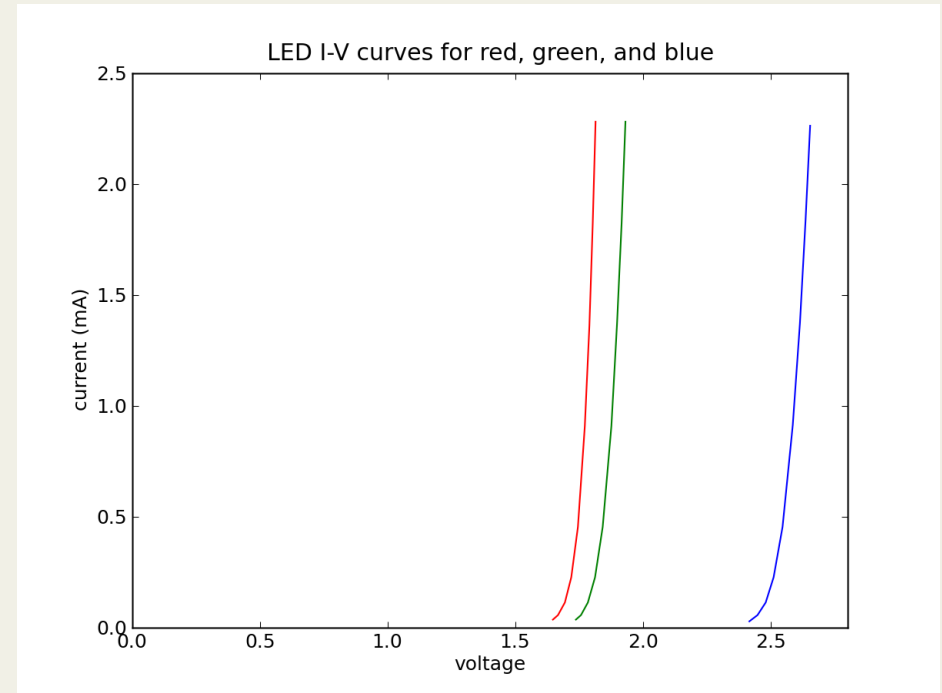
Topics and Techniques for Week 1 Lab

# Week 1 Lab has 4 Exercises

- Blinking an LED in a Morse Code pattern
- Modulating LED brightness via PWM
- Using a switch to toggle LED and set brightness
- Analog input, reading a photocell
  - and possibly doing something about it
- Note that the last two constitute miniature versions of the final project
  - sense something in the real world; make some decisions accordingly; manipulate something in the real world in response
- These tasks largely follow from the *Getting Started* book

# LED hookup

- The output of Arduino digital I/O pins will be either 0 or 5 volts
- An LED has a diode-like I-V curve
- Can't just put 5 V across
  - it'll blow, unless current is limited
- Put resistor in series, so ~2.5 V drop across each
  - 250  $\Omega$  would mean 10 mA
  - 10 mA is pretty bright



# Blink Function (Subroutine)

- For complex blink patterns, it pays to consolidate blink operation into a function

```
void blink(int ontime, int offtime)
{
    // turns on LED (externally defined) for ontime ms
    // then off for offtime ms before returning
    digitalWrite(LED, HIGH);
    delay(ontime);
    digitalWrite(LED, LOW);
    delay(offtime);
}
```

- Now call with, e.g., `blink(600,300)`
- Note function definition expects two integer arguments
- `LED` is assumed to be global variable (defined outside of loop)

# Blink Constructs

- For something like Morse Code, could imagine building functions on functions, like

```
void dot()  
{ blink(200,200); }
```

```
void dash()  
{ blink(600,200); }
```

```
void letterspace()  
{ delay(400); }
```

```
void wordspace()  
{ delay(800); }
```

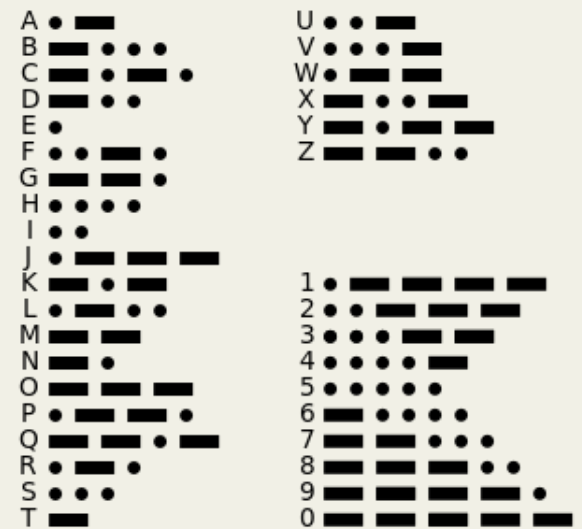
- And then perhaps letter functions:

```
void morse_s()  
{ dot(); dot(); dot(); letterspace(); }
```

```
void morse_o()  
{ dash(); dash(); dash(); letterspace(); }
```

## International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



# Morse, continued

- You could then spell out a word pretty easily like:

```
morse_s();  
morse_o();  
morse_s();  
wordspace();
```

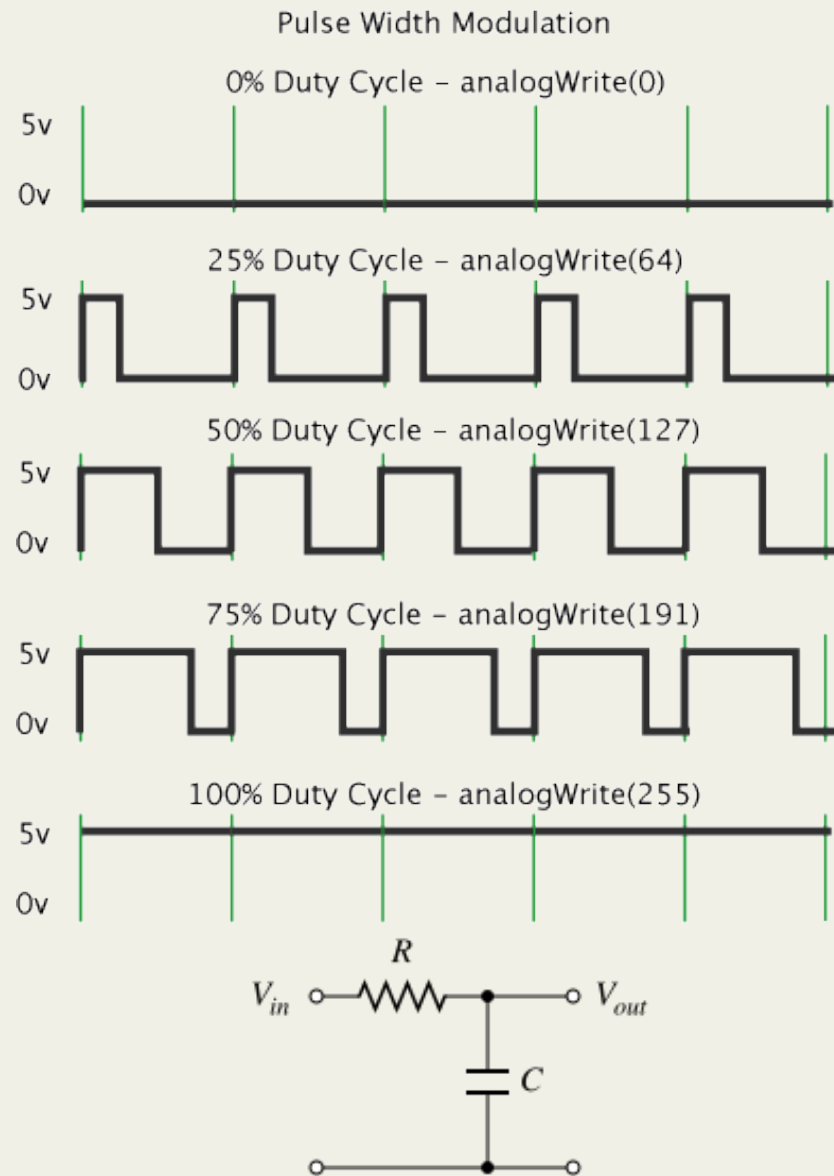
- Once you have a library of all the letters, it would be very simple to blink out anything you wanted

# Pulse Width Modulation

- A “poor man’s” analog output can be synthesized out of a digital (0–5 V) signal by pulsing at variable *duty cycle*
  - the *time average* voltage can then be anything between 0 and 5 V
- Arduino provides `analogWrite(pin, value)`, valid for 6 of the 14 digital I/O pins on the Uno
  - *value* is a number from 0 to 255 (one byte)
- For controlling LED brightness, the fraction of time in the ON state determines perceived brightness
- For other applications, may want capacitor to average (smooth) out the frenzied pulse sequence

# PWM, Visually

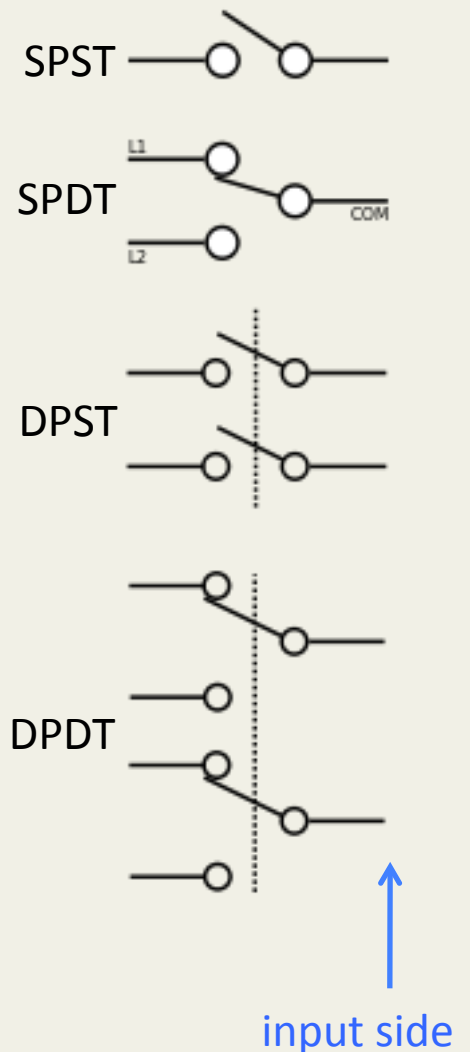
- At right, pulse period denoted by green markers
- Can go from always LOW (0% duty cycle) to always HIGH (100% duty cycle)
  - or anything in between, in 255 steps
- Can change period, if needed
  - though only among limited selection of options



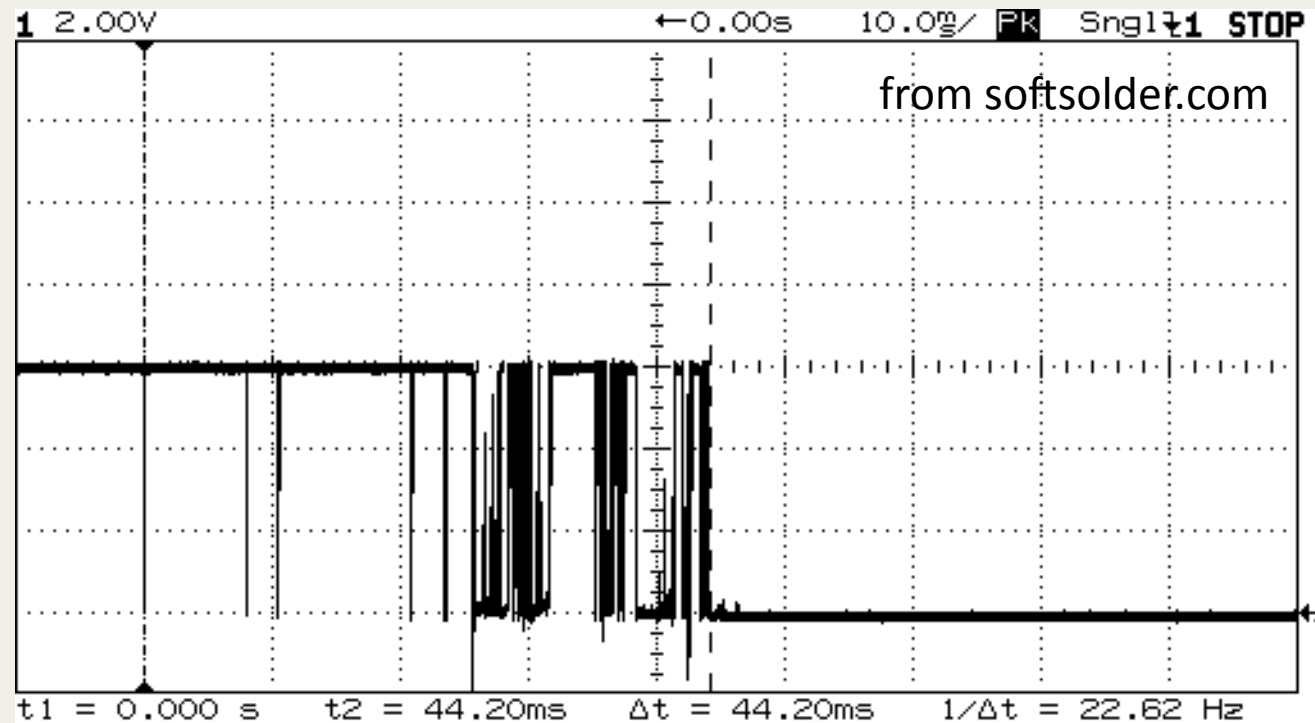


# Switches & Debouncing

- Switches come in a dizzying variety
  - normally open (NO), normally closed (NC)
    - applies to single throw, typically
  - single pole (SP), double pole (DP), etc.
    - how many inputs to the switch
  - single throw (ST), double throw (DT), etc.
    - how many contacts each input may make
    - DT can also come in CO variety: center open
- The Arduino kit button is NO, SPST
  - it is normally open, one input (shared two pins), one output (shared two pins)
- But switches are not as simple as you think
  - transition from open to closed can be erratic, random, fast oscillation, bouncing many times between states before settling



# Typical Bounce



- On the tens of milliseconds timescale, a switch can actually go through any number of transitions
- Each time will look completely different
- Idea is to catch first transition, then hold off until you're sure things have settled out

# Delay Can Save the Day

- A fast microprocessor looking for switch transitions can catch all these bounces, as if you had pressed the button many times in fast succession
  - This is seldom the behavior we want
- Inserting a delay gives the physical switch time to settle out
  - something like 50–100 ms is usually good; faster than you can intentionally press twice (see `dt_pair`)
- Often use hardware solution too, with flip-flops
  - lock in first edge
- Will also be relevant when we get to interrupts

# Thinking Through Complex Logic

- In the dimmer exercise, it's tough to keep track of the states
- Tendency to want to grasp entire scheme at once
- Brains don't often work that way
  - break it down to little pieces you understand
  - ask yourself questions *throughout the process*
    - Do I just need to know the state of the button, or catch change?
    - If catching a change, what am I comparing against?
    - Do I need a variable to keep track of a previous state?
    - If so, when do I store the “old” value?
    - If the button has just been pressed, what should I do?
    - Does the answer depend on the LED state?
    - Then do I need a variable to track this? (and the list goes on!)

# Analog to Digital Conversion (ADC)

- Computers are digital, while the physical world is analog
- Converting voltage (analog value expressed electrically) into a digital number is a fundamental task in computer/world interface
- Internally, the processor is doing a “guess and check” approach from most significant bit (MSB) to LSB
- Arduino Uno has six analog inputs, turning each into a 10-bit number, 0..1023
  - measure 0–5 V range to 0.1%, or 5 mV precision
- This is your key portal into using sensors

# Assignments/Announcements

- First week exercises due Tuesday, 1/15 by 2PM
  - can drop in slot on TA room in back of MHA 3544
  - expect code printout (can be common to group), and some paragraphs *from each group member* as to contribution: how do we know you did something and *learned*?
  - indicate which TA/prof checked off your operating circuit
    - if this has not happened yet, keep your code handy so we can check this off next time we cross paths
    - most convenient form is to print code during lab and have the behavior checked and signed off on that page (which is to be turned in with the complete package).