Analog Output

CSE 1302

Today's Outline

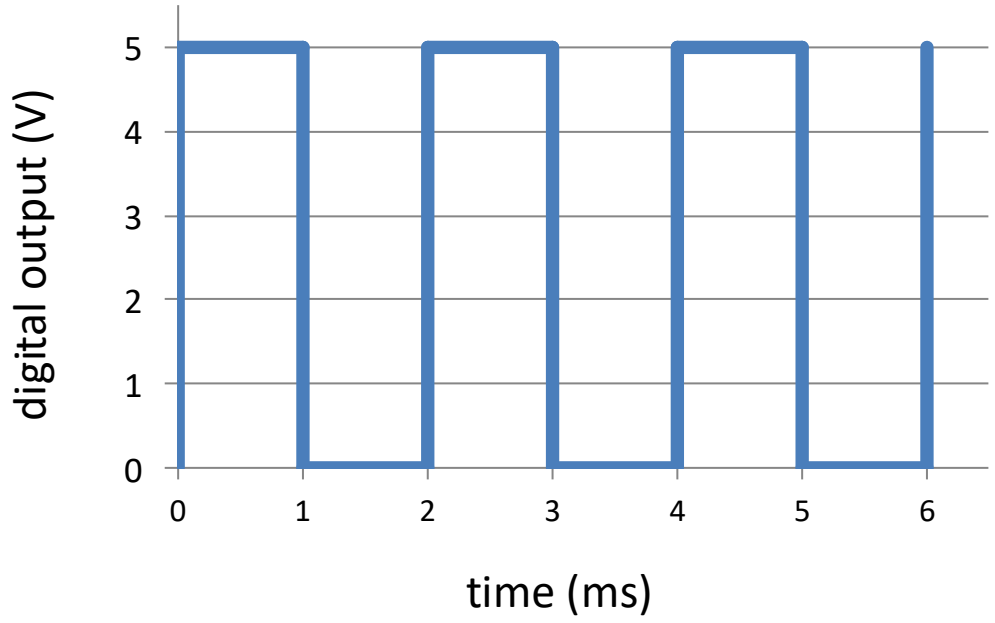
- Analog Output
- FSM for Receiving Protocol Messages
- Demos

Pulse Width Modulation (PWM)

- Analog output, built using a digital output
- Technique is to exploit the fact that many physical devices are slow, and respond to average of a fast-moving signal
  - E.g., What does our eye do with 30 frames/sec?
  - Our brain smooths out the motion so it looks continuous to us
- Send digital signal up and down quickly, and the “analog output” is the average value

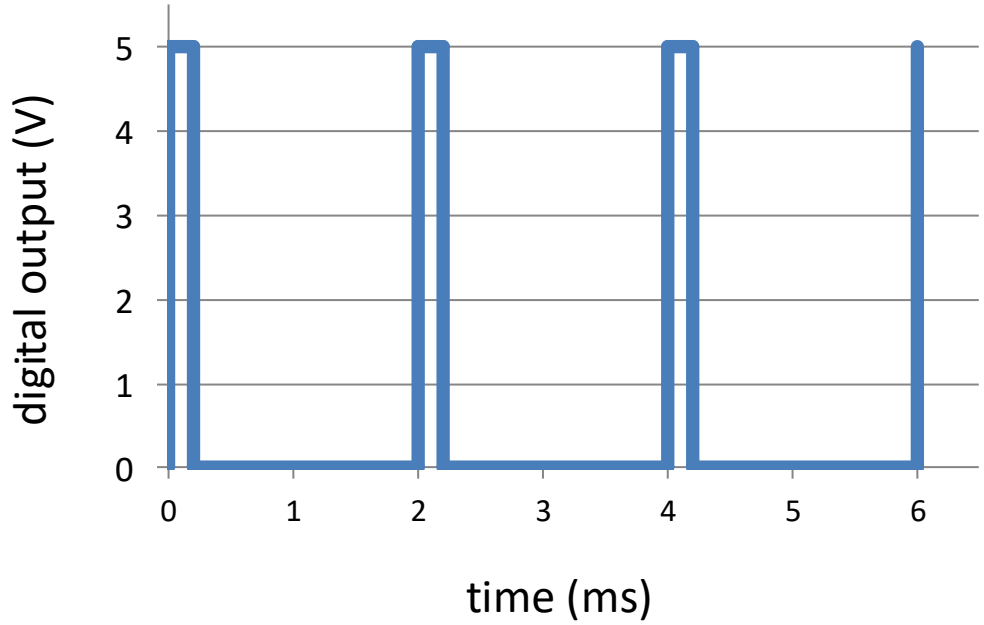
50% Analog Output

- 500 Hz period (2 ms)
- Repeating signal, ½ time 5 V and ½ time 0 V
- Average is 2.5 V



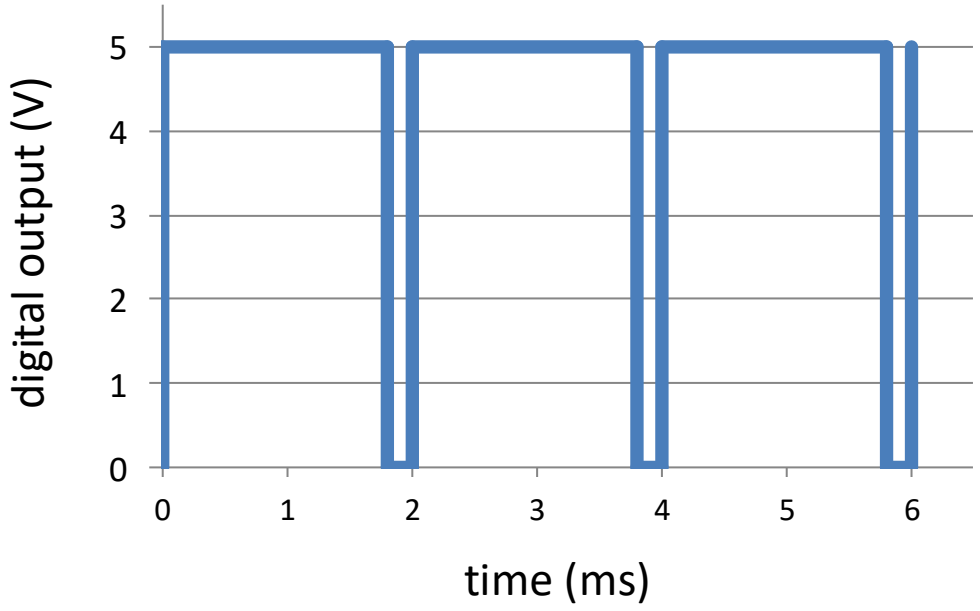
10% Analog Output

- Same 500 Hz period (2 ms)
- In this case, 10% time 5 V and 90% time 0 V
- Average is 0.5V



90% Analog Output

- Same 500 Hz period (2 ms)
- In this case, 90% time 5 V and 10% time 0 V
- Average is 4.5V



analogWrite()

- analogWrite(pin, value)
  - pin is one that supports PWM outputs, not all do!
  - value is 8-bit analog value (range is 0 to 255)

- Useful for slow-moving physical devices, e.g.,
  - LEDs (actually, it is our eyes that are slow)
  - 5 V motors (hard to start/stop at 500 Hz)
- Can be used for other devices if averaging is done by circuitry between Arduino and device

Use Cases

- LEDs
  - Analog value 255 → full bright
  - Analog value 0 → off
  - Analog value 127 → half intensity
- DC motors
  - Analog value 0 → stopped
  - Analog value 127 → half speed
  - Analog value 255 → full speed
- Servo motors
  - Analog value tells holding position
  - Range 0 to 180 degrees

Quiz Time

- Go to Canvas and answer the single question on Quiz 7A

True or False:

Any Arduino pin that supports being used as a digital output can also be used as an analog output by calling analogWrite() with the corresponding pin number.

Communications Receiver

- First check to see if byte has arrived
  - Arduino
    - Serial.available() returns integer count of available bytes
  - Java
    - s.available() returns Boolean if a byte is available
- Next read one (and only one) byte
  - Serial.read() on Arduino (returns an int)
    - Return value is -1 if nothing received, byte value otherwise
  - s.readByte() on Java (returns a byte)
- Check available() prior to *each* read!

Not Like This!

```
if (Serial.available() > 0)
        b1 = Serial.read();
if (Serial.available() > 0)
        b2 = Serial.read();
// code that assumes b1 and b2 are good
```

What could go wrong here?

What if the two bytes are sent based on a human pressing a button, and the human takes a while between the two button presses?

A Byte Might *Not* Be Available

```
if (Serial.available() > 0)
        b1 = Serial.read();
else
        wait until Serial.available() is > 0!!!!
```

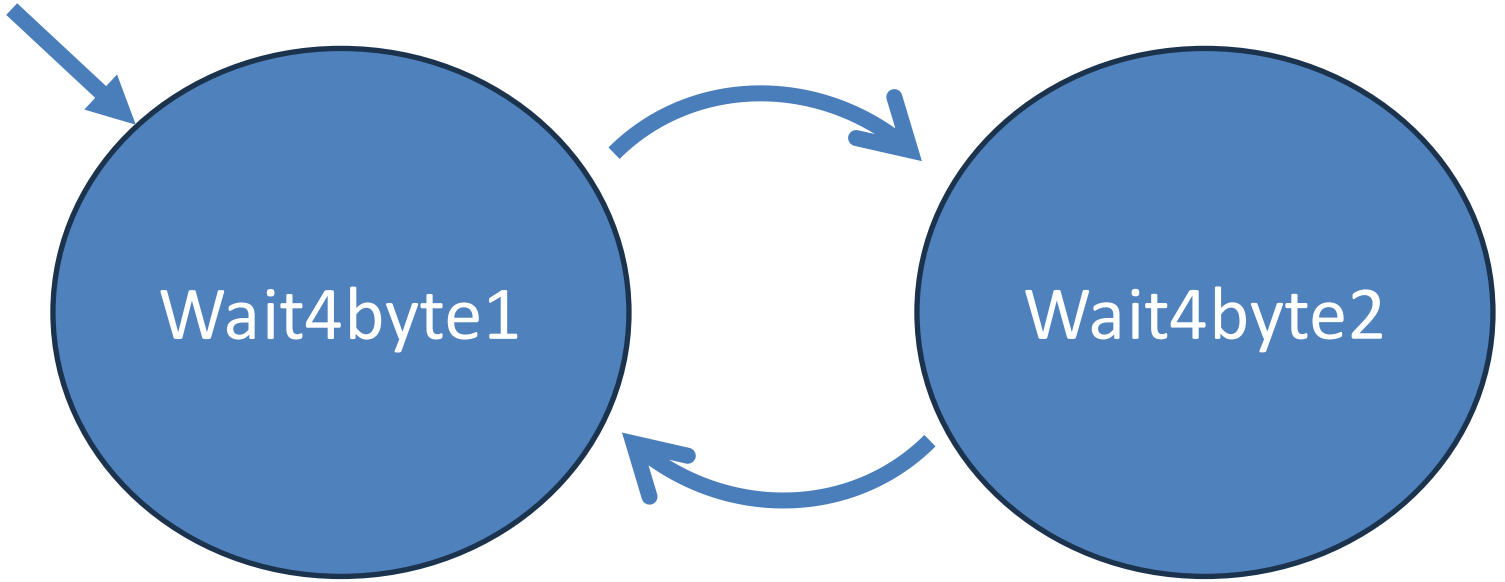
The above looks like it will block (*note*: it will!)

We need to transition to a non-blocking approach

FSMs to the rescue!

FSM to Receive 2 Byte Integer

- Initial state: Wait4byte1



- State transition: receipt of a byte
  - available() followed by read()
  - Save incoming byte on transition

FSM to Receive 2 Byte Integer

```
while (true)
   if (Serial.available() > 0)
         inputByte = Serial.read();
         switch (state) {
         case Wait4byte1: b1 = inputByte;
                          nextState = Wait4byte2;
         case Wait4byte2: b2 = inputByte;
                          nextState = Wait4byte1;
                          inputValue = (b1 << 8) + b2;
         }
   }
```

Receive 2 Byte Integer

- Use FSM to save b1 (first byte) and b2 (second)

- Compose int from 2 bytes:

```
int value = (b1 << 8) + b2;
```

- Watch out for sign extension in Java

```
int value = ((0xff & b1) << 8) + (0xff & b2);
```

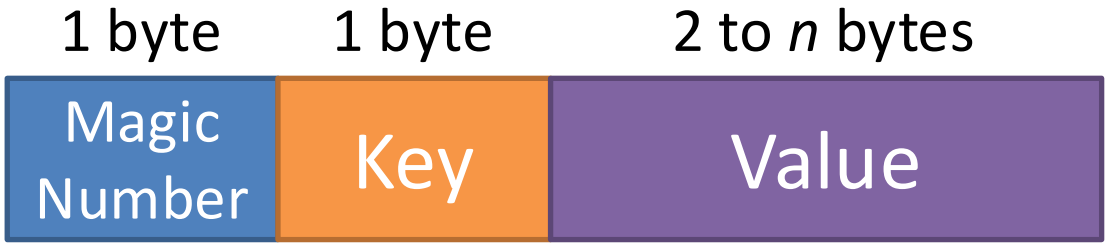
  – Because bytes get promoted to int before math operations (silly Java rule)

Communications and Delta Time

```
while (true)
        now = millis()
        if (byte available) then
                read byte and save (i.e., FSM)
        endif
        if (delta time has expired)
                do time-based stuff
        endif
endwhile
```

Our Protocol

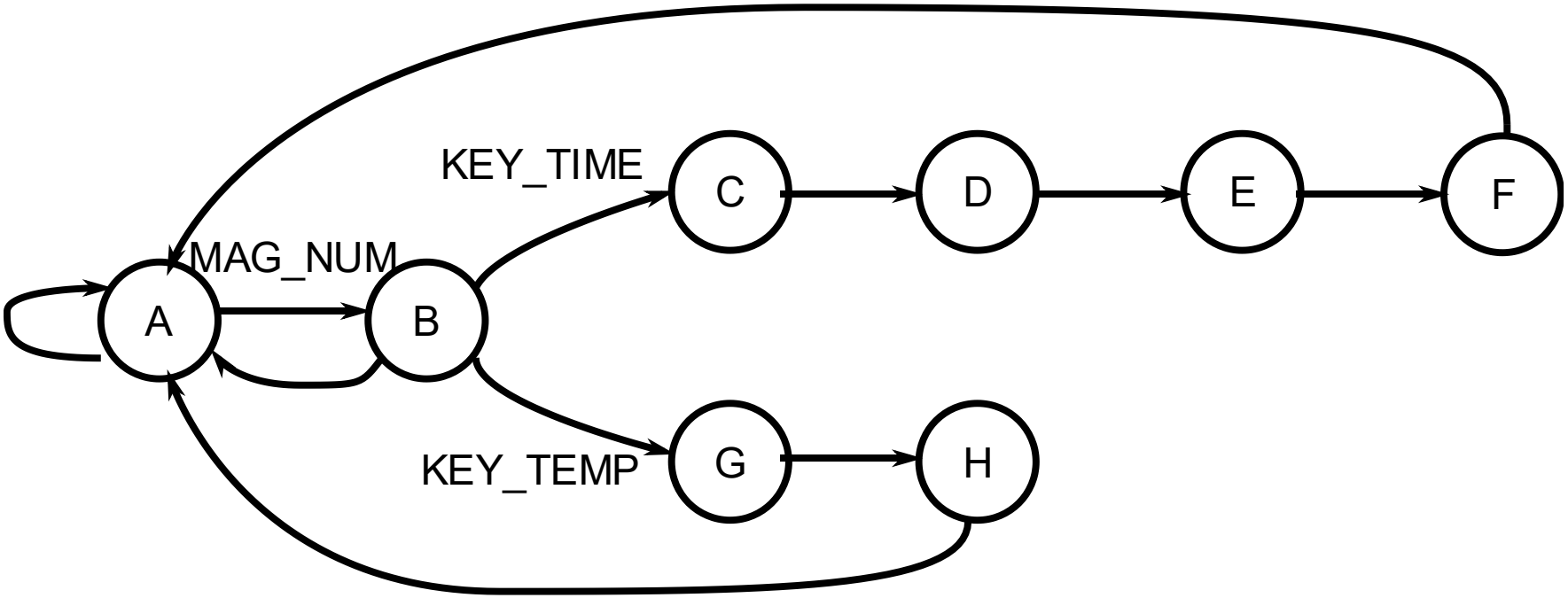
Message format:



Examples:

- MAG_NUM is magic number
- KEY_TIME is key for 4-byte time value
- KEY_TEMP is key for 2-byte temperature

FSM to Receive Our Protocol



States:

- A is wait for magic number
- B is wait for key
- C to F wait for bytes of time value
- G to H wait for bytes of temperature value