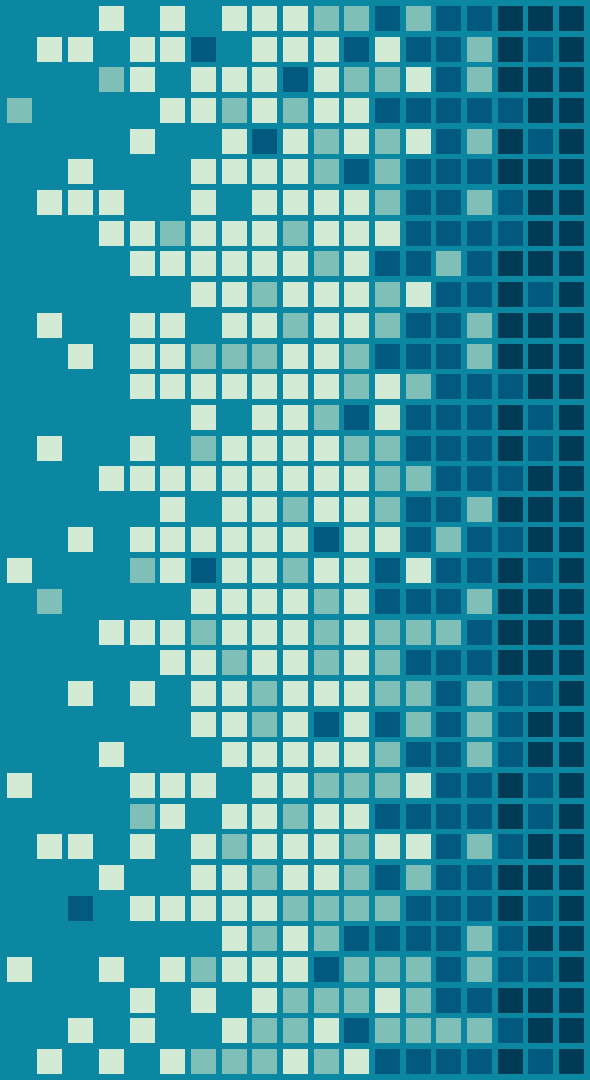
PERIPHERAL INTERFACES

Michael D'Argenio – mjdargen@ncsu.edu
Electrical Engineering – SS 2019 – Duke TIP



Arduino Uno's Peripheral Interfaces

- Digital Pins or General Purpose Input/Output (GPIO)
  - Digital inputs
  - Digital outputs
  - PWM output
  - External hardware interrupts
  - Serial communications (UART, SPI, I^2C)
- Analog Pins
  - Analog inputs

DIGITAL PINS:

INTRO

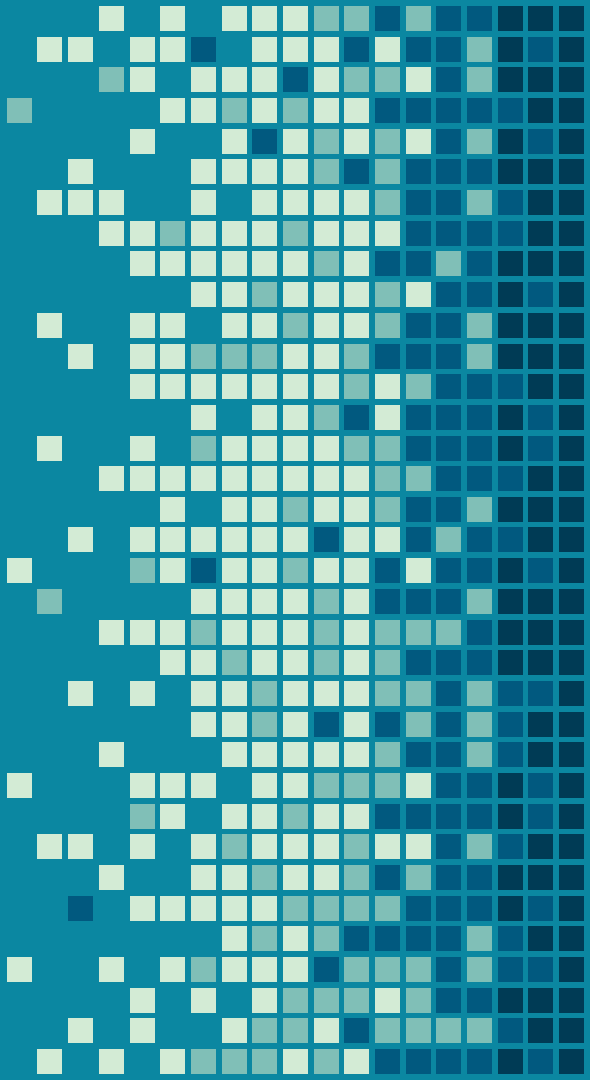
Digital or GPIO Pins

- There are only two states the MCU can drive or read on these pins: HIGH (ON) or LOW (OFF).
  - HIGH range: 3V – 5.5V
  - LOW range: 0.5V -1.5V
  - All other voltages are indecipherable.

- Avoid pins 0 and 1. These are used for communication to computer using USB and can interfere with uploading new programs.

DIGITAL PINS:

Digital Output



Digital Outputs

- 14 digital pins
- Can source (provide positive current) or sink (provide negative current) up to 40 mA of current.
- Enough current to brightly light up an LED (don't forget the series resistor)or run many sensors, but not enough current to run most relays, solenoids, or motors.
- To prevent accidental overcurrent, it is a good idea to connect OUTPUT pins to other devices with 470Ω or 1k series resistor.

Digital Outputs API

- `pinMode(pin, mode)`
  - pin: the pin number
  - mode: OUTPUT
- `digitalWrite(pin, value)`
  - pin: the pin number
  - Value: HIGH or LOW
- More info: https://www.arduino.cc/en/Tutorial/DigitalPins

Digital Output Example

Example Code

The code makes the digital pin 13 an `OUTPUT` and toggles it by alternating between `HIGH` and `LOW` at one second pace.

```
void setup() {
  pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // sets the digital pin 13 on
  delay(1000);            // waits for a second
  digitalWrite(13, LOW);  // sets the digital pin 13 off
  delay(1000);            // waits for a second
}
```

Activity: LED Digital Output

- Goal: Flash an external LED every 2 seconds.
- Remember series resistor!
  - Recall: why do we need this again?

DIGITAL PINS:

Digital Input

Digital Inputs

- 14 digital pins
- Can only withstand up to 5.5V.
- Can only differentiate between HIGH and LOW
- Useful for reading capacitive touch sensors, photo diodes, switch states, and other devices that provide a logical state information (HIGH or LOW).

Digital Inputs API

- `pinMode(pin, mode)`
  - pin: the pin number
  - mode: INPUT
- `digitalRead(pin)`
  - pin: the pin number
  - Returns HIGH or LOW (int value)
- More info: https://www.arduino.cc/en/Tutorial/DigitalPins

Digital Input Example

Example Code

Sets pin 13 to the same value as pin 7, declared as an input.

```
int ledPin = 13;  // LED connected to digital pin 13
int inPin = 7;    // pushbutton connected to digital pin 7
int val = 0;      // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the digital pin 13 as output
  pinMode(inPin, INPUT);    // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's value
}
```

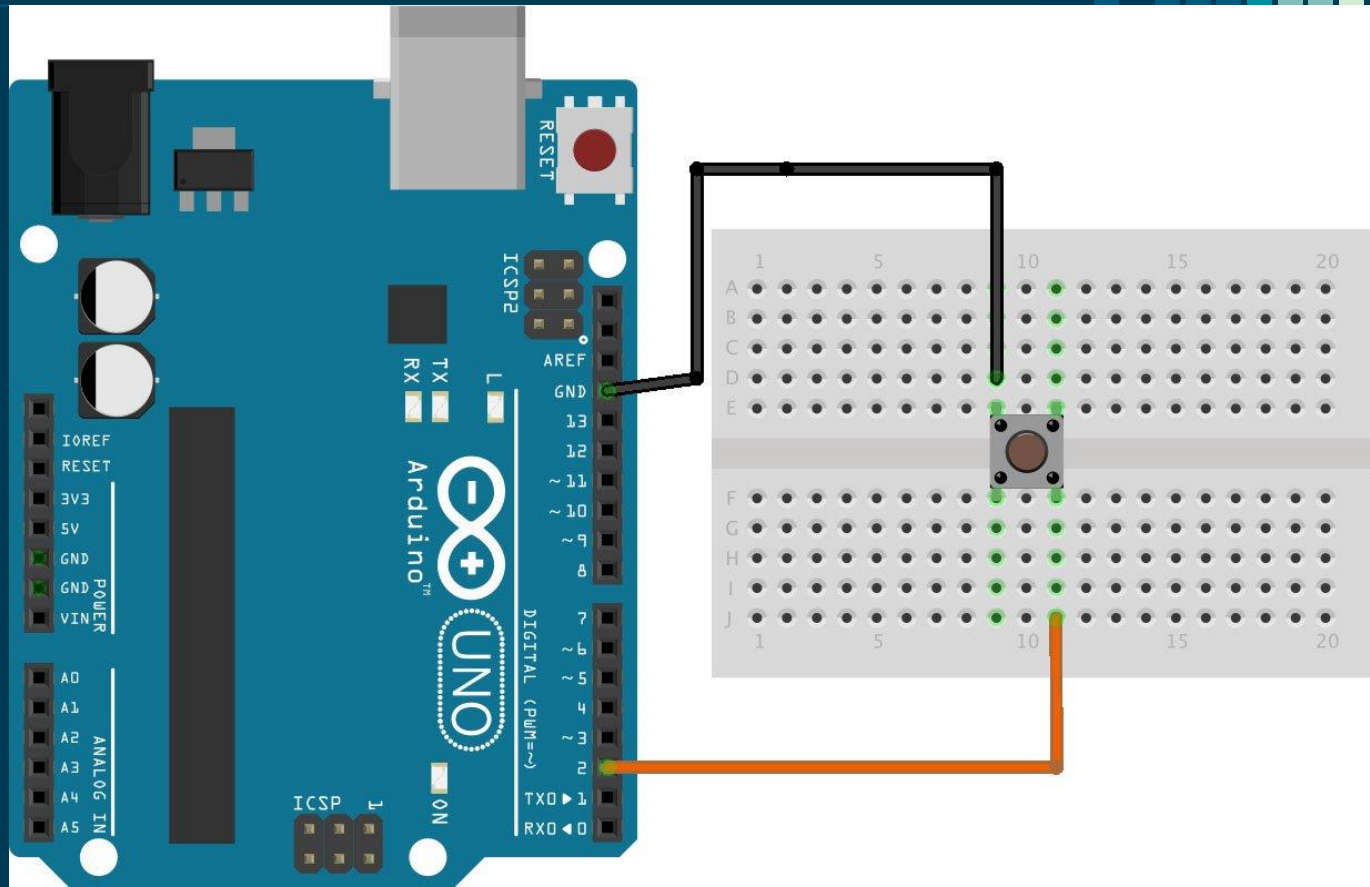
* If pin 7 is high, turn on internal LED connected to pin 13.

Digital Input Activity 1: Pushbutton

- Goal: Turn on LED when pushbutton is pressed.
- Does it behave as you would expect?

Exp: Flash LED when we press button

What's happening??

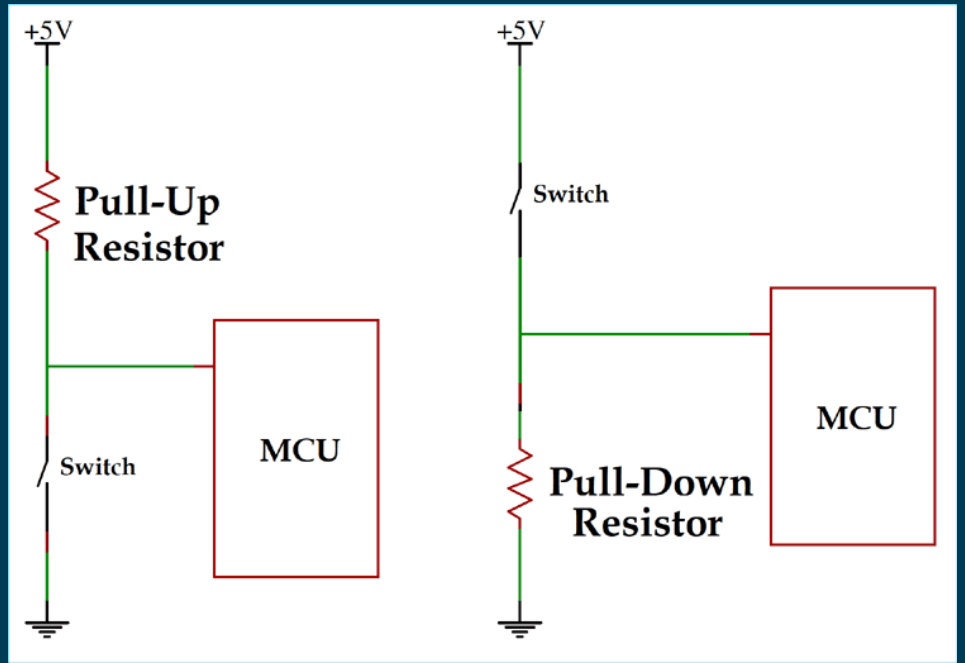


Unknown Switch State

- When we press the pushbutton, the switch connects digital input 2 to ground.
- So in software, we do the following:
  - When we read digital input 2 as LOW, turn on the LED.
  - When we read digital input 2 as HIGH, turn off the LED.
- However, when the button is not pressed, what happens?
- Nothing is connected!!!
- If a pin is left floating (i.e. not connected), it will give us random results (high and low).
  - Causes faulty behavior.
  - We need a solution.

Solution: Pull-Up/Pull-Down Resistors

- Active Low – Pull-up
- Active High – Pull-down



Digital Input Activity 2: External Pull-up

- Goal: Turn on LED when pushbutton is pressed using a pull-up resistor on the breadboard.

- Does it behave as you would expect?

Digital Input API (Another Mode)

- `pinMode(pin, mode)`
  - pin: the pin number
  - mode: INPUT_PULLUP
  - Exp: pinMode(2, INPUT_PULLUP)
- More info: https://www.arduino.cc/en/Tutorial/DigitalPins

Digital Input Activity 3: Internal Pull-up

- Goal: Turn on LED when pushbutton is pressed using the internal pull-up resistor in the MCU.

- Does it behave as you would expect?

Debouncing

- Pushbuttons often generate spurious open/close transitions when pressed, due to mechanical and physical issues: these transitions may be read as multiple presses in a very short time fooling the program.
- This example demonstrates how to debounce an input, which means checking twice in a short period of time to make sure the pushbutton is definitely pressed.
- Without debouncing, pressing the button once may cause unpredictable results.
- https://www.arduino.cc/en/Tutorial/Debounce

Digital Input Activity 4: Debouncing

- Goal: Set up debouncing in software to make sure you are not getting false button presses or registering one button press as multiple.

- How could we possible do this better?
- How could we more accurately and more timely detect a button press?

Digital Input Activity 5: Interrupt

- Goal: Turn on LED when pushbutton is pressed by using interrupts instead of polling.
- Remember: interrupts can only be triggered by digital pins 2 & 3.
- What should trigger the interrupt? What mode should you configure?
- What code should you put in the ISR?

Digital Input Activity 6: RGB LED

- Goal: Now use an RGB LED and add another pushbutton switch. When SW1 is on, turn on green LED. When SW2 is on, turn on blue LED. When both switches are off, turn on RED LED. Use interrupts.

- Can you draw a truth table for the 2 switches?

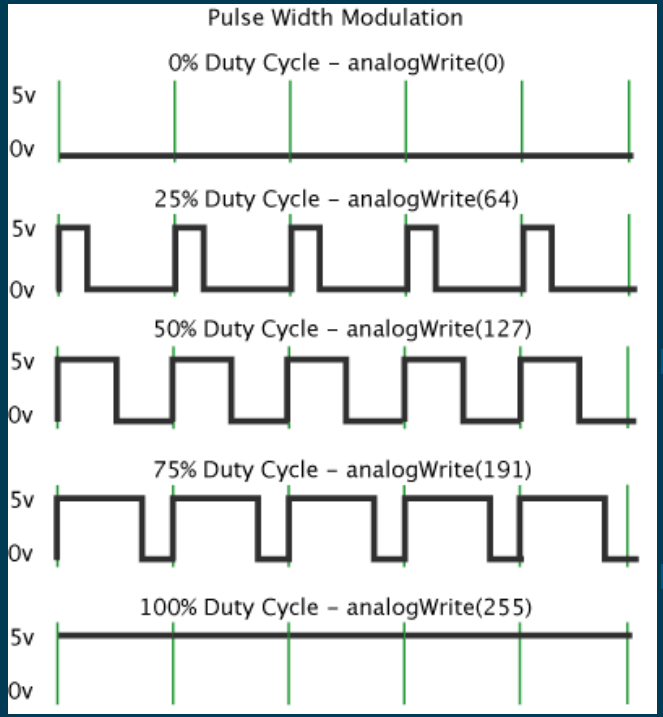
- What happens when both buttons are pressed?

DIGITAL PINS:

PWM Outputs

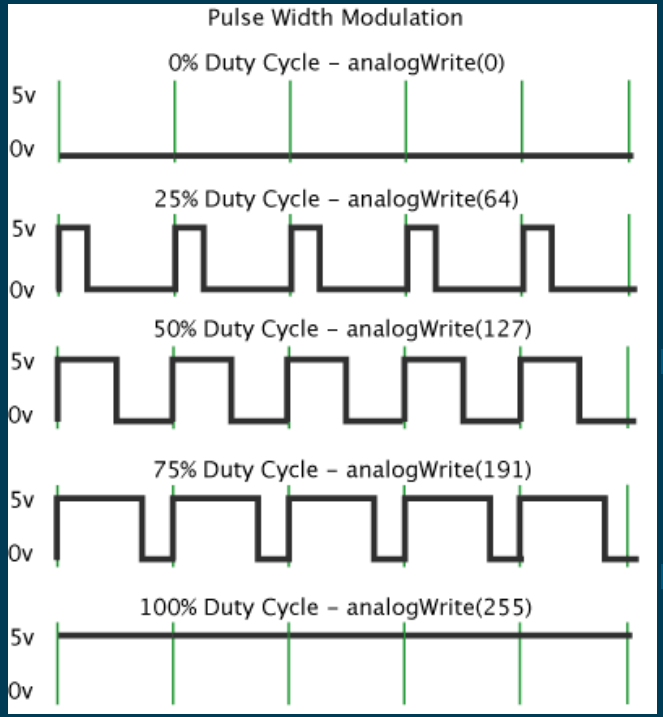
What is PWM?

- PWM: Pulse Width Modulation
- A technique for getting analog results with digital means.
- Digital control is used to create a square wave, a signal switched between on and off.
- Duty Cycle: percentage of time "on" over the period of the signal.



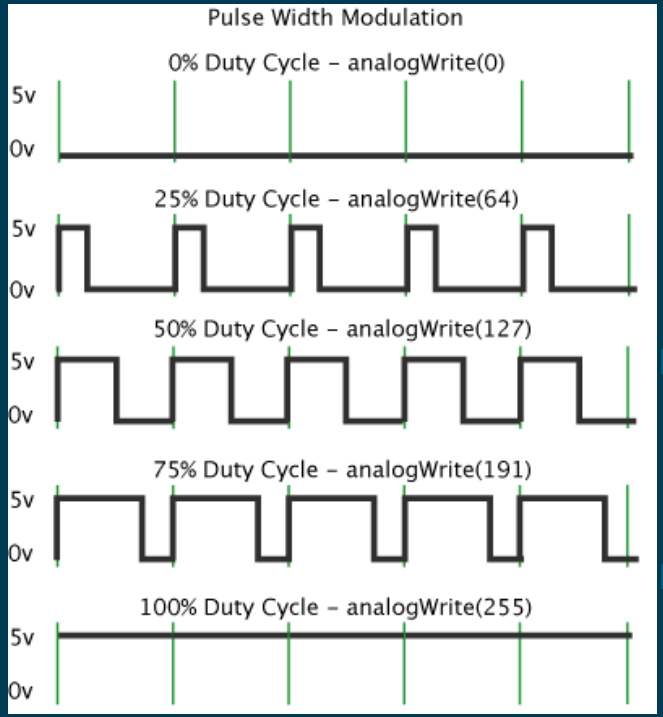
PWM

- Frequency F (Hz): occurrences per second.
  - $F = \frac{1}{T}$
- Period T (s): the duration of one occurrence.
  - $T = \frac{1}{F}$
  - $T = t_{on} + t_{off}$
- On Time t_{on} (s): on duration
- Off Time t_{off} (s): off duration
- Duty Cycle: percentage on
  - $Duty\ Cycle = \frac{t_{on}}{T} * 100\ \%$



Arduino PWM

- PWM available on pins 3, 5, 6, 9, 10, & 11.
- Arduino PWM frequency
  - Pins 3, 9, 10, 11: 500 Hz.
  - Pins 5 ,6: 1000 Hz.
- Provide 8-bit PWM output with the analogWrite() function.
- 8-bit resolution: 2^8 = 256 steps
- analogWrite() function takes integer argument between 0-255.



When do we use PWM?

- To control brightness of LED.
  - It is switching the LED off and on, but it is doing it too fast for the eye to see. So it just appears dimmer.
- To control motors.
  - Works in much the same way as LED.
  - By having a smaller duty cycle, we are supplying less power to the motor. So it runs slower.

PWM API

- `analogWrite(pin, value)`
  - pin: the pin number.
  - value: the duty cycle scaled between 0 (always off) and 255 (always on).
- Multiply duty cycle by 255 and round to closest int to get the argument to function.
  - 50% * 255 = 127
  - 25% * 255 = 64
- More info: https://www.arduino.cc/en/Tutorial/PWM

PWM Example Code

Example Code

Sets the output to the LED proportional to the value read from the potentiometer.

```
int ledPin = 9;       // LED connected to digital pin 9
int analogPin = 3;    // potentiometer connected to analog pin 3
int val = 0;          // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the pin as output
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023, analogWrite values from 0 to 255
}
```

PWM Activity: Fading an LED

- Turn on external LED for 1 second. Fade to dark over the course of 2 seconds. Off for 2 seconds.
- 5 second period, then repeat.

- How dim can you make it?
- What is the smallest duty cycle that is still visible?

PWM Activity: Fan Control

- Goal: Use a PWM signal to drive a fan at 3 different speeds: high, medium, and low.

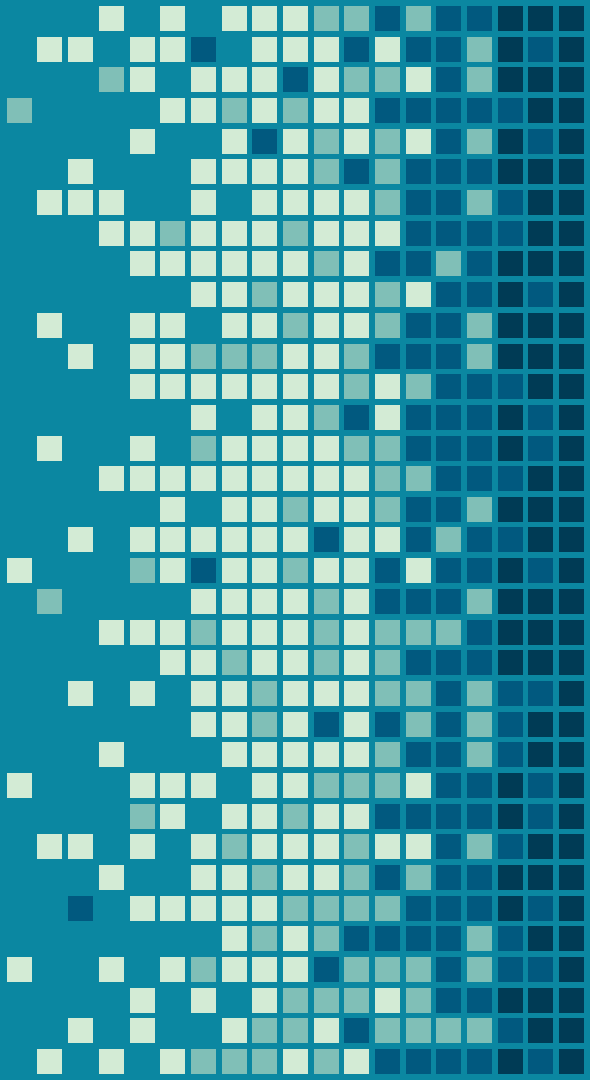
- Remember, there is a current output limit on digital pins! Can’t drive motor directly!

PWM Activity: Play a Song!

- Goal: Send out a square wave to a piezo buzzer (passive) at varying frequencies to play a song!

- https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/
- https://www.arduino.cc/en/Tutorial/ToneMelody

ANALOG PINS



Analog Inputs

- Arduino Uno has an onboard 6 channel analog-to-digital converter (ADC).
- The ADC has 10 bit resolution, returning integers from 0 to 1023 ($2^{10} = 1024$).
- Analog pins can be used as GPIO if more digital pins are needed.

ADC Noise

- The ATmega datasheet also cautions against switching analog pins in close temporal proximity to making A/D readings (analogRead) on other analog pins. This can cause electrical noise and introduce jitter in the analog system.
- It may be desirable, after manipulating analog pins (in digital mode), to add a short delay before using analogRead() to read other analog pins.

ADC Sampling

- In some applications, there may be noise on the analog voltage you are reading.
- You may need to take several readings and average them (or throw out any outliers) to get an accurate measurement.
- For more info: https://www.arduino.cc/en/Tutorial/Smoothing

ADC API

- Note: no setup required unless pin was previously configured as output.
  - Then set as input: pinMode(pin, INPUT)
- `analogRead(pin)`
  - pin: the name of the analog input pin (A0 to A5)
  - Returns int between 0-1023. (10 bit resolution)
  - Must scale to real voltage (use reference voltage)
  - ADC result: 512 --> $512 \times \frac{5\ V}{1024} = 2.5\ V$

Floating Point/Decimal Numbers

- Floating point numbers can be hard to handle.
- You must convert floating point numbers back to char before using serial communications.
- Floating point calculations take a lot longer to compute.
- Keep ADC values as ints as long as possible.
  - Often means you don't scale it until the end.

ADC Example

Example Code

The code reads the voltage on analogPin and displays it.

```
int analogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3
                    // outside leads to ground and +5V
int val = 0;  // variable to store the value read

void setup() {
  Serial.begin(9600);           //  setup serial
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  Serial.println(val);          // debug value
}
```

Activity: Night Light

- Goal: Detect the ambient brightness with a photoresistor. Turn on the night light (external LED) if it gets too dark.

Activity: Adjustable Fan

- Goal: Using a potentiometer, adjust the speed of the fan being driven by a motor.

- Remember: can’t drive motor directly with Arduino.

DIGITAL PINS:

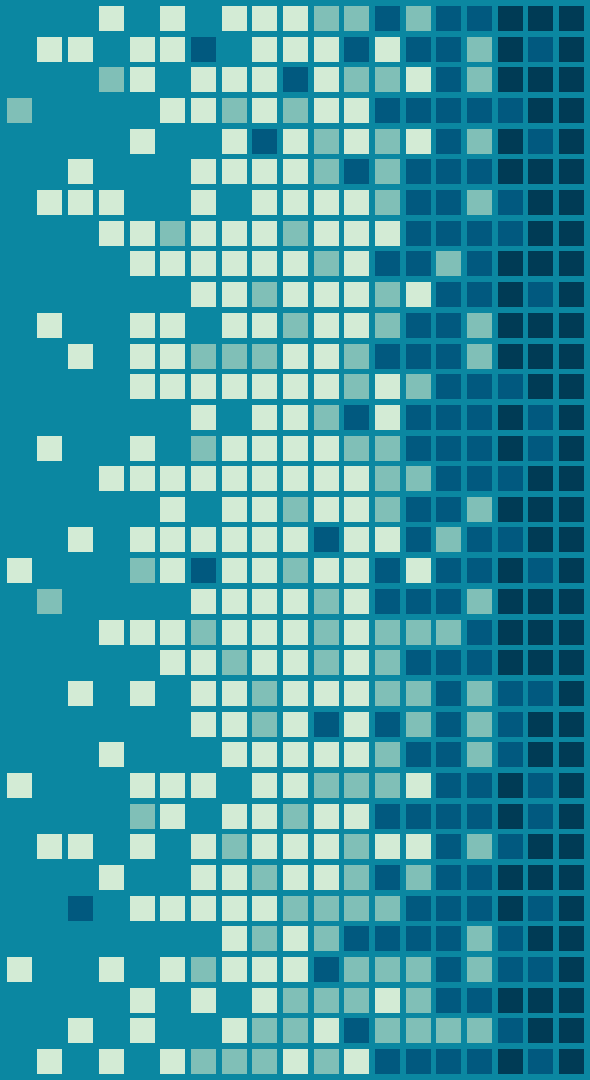
Parallel Communications: LCD

Activity: LCD “Hello, World!”

- Goal: Set up communications with the LCD screen. Display “Hello, World!”

DIGITAL PINS:

Serial Communications



What are serial communications?

- What has serial meant for circuit elements?
  - Series resistors versus parallel resistors

- Parallel communications – using multiple wires to communicate multiple bits of data at a time
- Serial communications – using one wire to communicate a single bit of data at a time.

Why serial communications?

- Parallel
  - Faster: could send more data at once.
  - Data is packed into bytes. Could send 8 bits at once.
- Serial
  - Cost and weight: more wires, larger connectors needed
  - Mechanical reliability: more wires => more connector contacts to fail
  - Timing complexity: some bits may arrive later than others due to variations in capacitance/resistance across conductors
  - Circuit complexity and power: may not want to have 16 different radio transmitters + receivers in the system

Two Type of Serial Communications

- Synchronous
  - Explicit clock signal, separate signal
  - In addition to data line, it also has a clock line.
  - Transmitter and receiver use dedicated clock line to “clock” data
- Asynchronous
  - Implicit clock
  - Only data lines, no clock lines
  - Transmitter and receiver use internal clocks to “clock” data
- “Clock” data – each new clock signal marks a new data bit.

DIGITAL PINS:

Serial Communications: UART

Asynchronous - UART

- UART – Universal Asynchronous Receiver/Transmitter
  - Universal – configurable to fit all asynchronous protocol requirements
  - Asynchronous – no clock line needed to de-serialize bits
  - Receiver/Transmitter – only requires two lines (receiver and transmitter)

- Arduinos support UART

UART Properties

- Transmitter and receiver must agree on several things
  - Order of data bits (LSB or MSB)
  - Number of data bits (7,8,9)
  - What a start bit is (1 or 0)
  - What a stop bit is (1 or 0)
  - Parity (odd, even, none) - optional
  - Baud rate
- Baud rate – number of bits per second.
  - Receiver and transmitter must agree on rate.
  - Common rates: 9600, 19200, 115200, etc.

Asynchronous – Transmitter

- When there is no data to send
  - Keep sending 1 (stop bit)
- When there is a data word to send
  - Send a 0 (start bit) to indicate the start of a word
  - Send each data bit in the word (use a shift register for the *transmit buffer*)
  - Send a 1 (stop bit) to indicate the end of the word (keep sending it until more data to send)

Asynchronous – Receiver

- Waits for a falling edge (beginning of a Start bit)
  - Then wait ½ bit time
  - Do the following for as many data bits in the word
    - Wait 1 bit time
    - Read the data bit and shift it into a *receive buffer* (shift register)
  - After entire word received, wait 1 bit time
  - Read the bit
    - if 1 (Stop bit), then OK
    - if 0, there's a problem!

More UART Resources

- http://www.circuitbasics.com/basics-uart-communication/

UART API – Pins 0 & 1 – USB Comms.

- Only for pins 0 and 1 on the Arduino Uno
- https://www.arduino.cc/reference/en/language/functions/communication/serial/
- If you want to use UART on other pins, must use the SoftwareSerial Library.
- https://www.arduino.cc/en/Reference/SoftwareSerial
- Elegoo Lesson 1

UART API – Not Pins 0 & 1

- Only for pins 0 and 1 on the Arduino Uno
- https://www.arduino.cc/reference/en/language/functions/communication/serial/
- If you want to use UART on other pins, must use the SoftwareSerial Library.
- https://www.arduino.cc/en/Reference/SoftwareSerial

Activity: RGB LED

- Goal: Set up UART serial port to receive messages from your computer. When you send ‘R’, turn on/off red LED. When you send ‘G’, turn on/off green LED. When you send ‘B’, turn on/off blue LED.

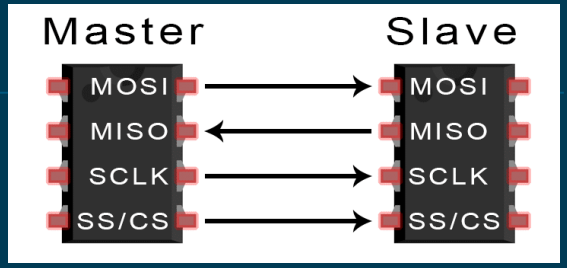
Activity: Display Voltage Values

- Goal: Connect potentiometer between 5V and GND. Use ADC to read voltage value. Set up serial communications with computer and ADC values to your computer to display them on a graph.
- Use Serial Plotter in Arduino IDE.

DIGITAL PINS:

Serial Communications: SPI

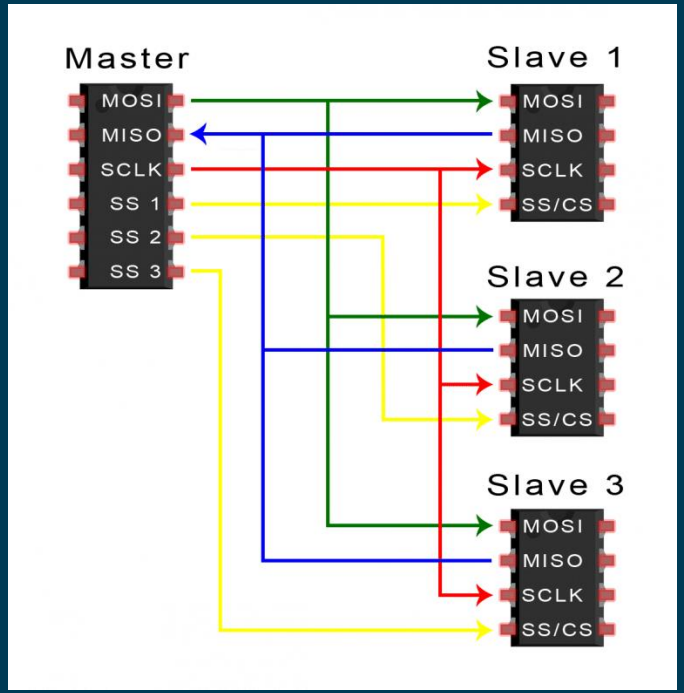
Synchronous – SPI



- SPI – Serial Peripheral Interface
- 4 lines required
  - MOSI (Master Output/Slave Input) – Line for the master to send data to the slave.
  - MISO (Master Input/Slave Output) – Line for the slave to send data to the master.
  - SCLK (Clock) – Line for the clock signal.
  - SS/CS (Slave Select/Chip Select) – Line for the master to select which slave to send data to.

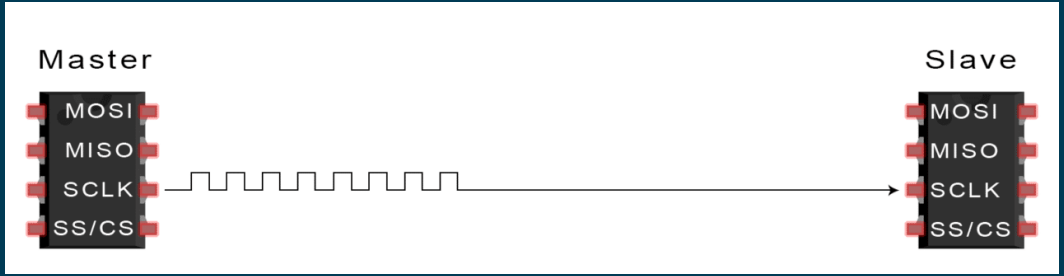
SPI: CS for Multiple Slave Devices

- Can theoretically have infinite # of slave devices
- CS (Chip Select) – 1 CS line for each slave device.
- Signals which slave should listen by driving line low.
- Stays low if the Master expects a response from the slave.

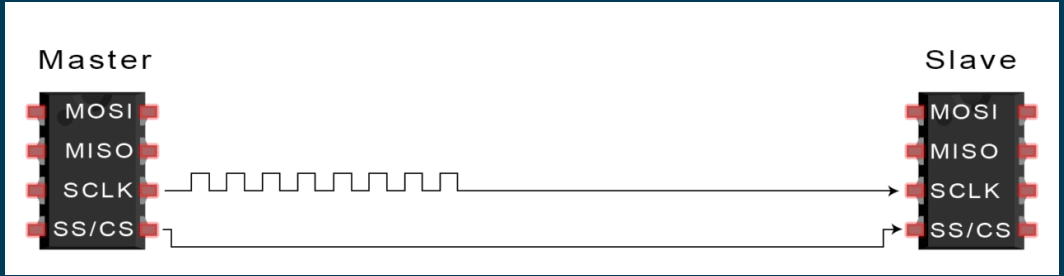


SPI Data Transmission

- 1. The master outputs the clock signal:

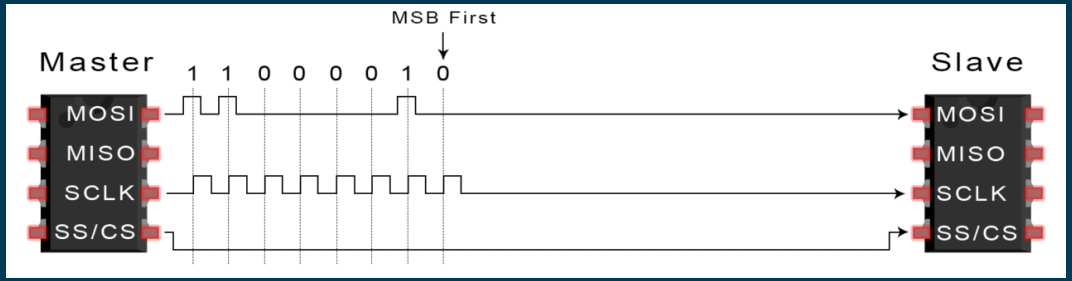


- 2. The master switches the CS line low, activating the slave:

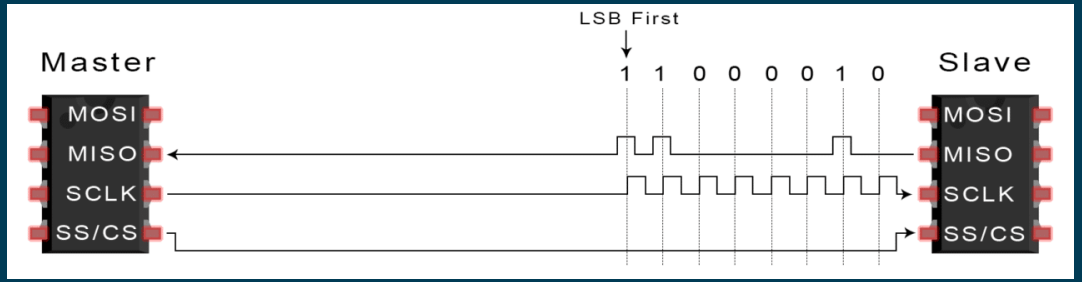


SPI Data Transmission

- 3. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads bits as they are received:



- 4. If a response is needed, the slave returns data one bit at a time to master along MISO line. The master reads bits as they are received:



SPI Resources

- More info: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/
- SPI Arduino Library: https://www.arduino.cc/en/Reference/SPI

Activity: SPI comms w/ LED Matrix

- Goal: Successfully communicate with the MAX7219 chip using the SPI protocol to control the 8x8 LED Matrix.

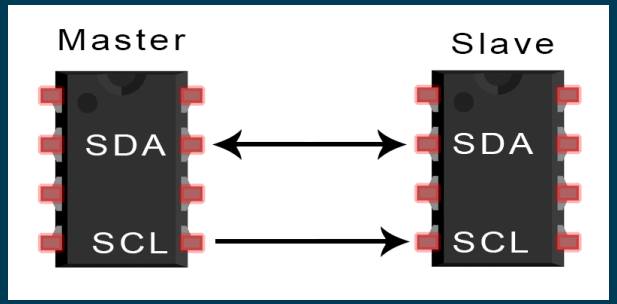
- We will use this for another activity.

DIGITAL PINS:

Serial Communications: I²C

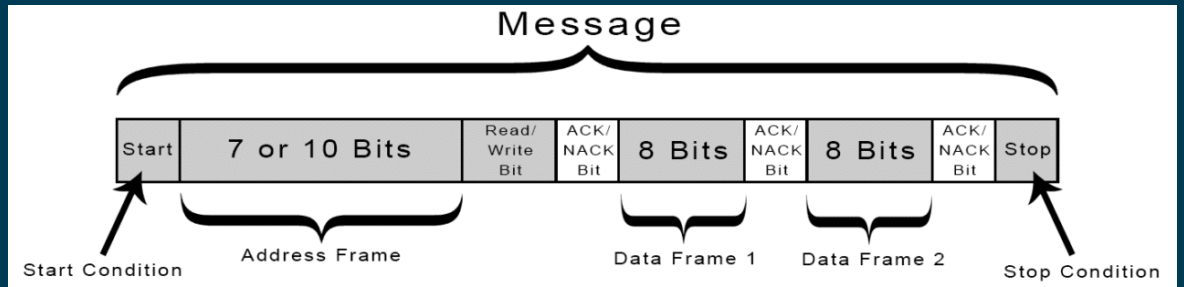
Synchronous – I²C or TWI

- I²C or I2C – Inter-Integrated Circuit
- Uses only 2 wires to communicate
  - SDA (Serial Data) – The line for the master and slave to send and receive data.
  - SCL (Serial Clock) – The line that carries the clock signal.
- Can have multiple masters and multiple slaves.
- Packages data into messages with frames of data.



I²C Message

- Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low.
- Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.
- Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
- Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
- ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.



I²C Resources

- More info: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/
- Arduino Library info: https://www.arduino.cc/en/Reference/Wire

Activity: IMU - Magic 8-Ball

- Goal: Use the IMU and the LCD screen to create a new version of a magic 8-ball.
- The IMU will determine the position and rotation of the device.
- Turn over the Arduino to ask it a question. Then when you flip it back, it should display a message.

ADDITIONAL ACTIVITIES

Activity: 8x8 LED Matrix Animation

- Goal: Create an animation on the 8x8 LED Matrix.
- Must have at least 5 different frames.

Activity Alarm System

- Use the ultrasonic sensor or the PIR sensor as a proximity/motion detector.
- When it detects a presence, trigger an alarm.
- The alarm can be anything you want!
- Be as creative and obnoxious as you can!

Activity: Scrolling LCD

- Can you get your LCD to automatically scroll for text that is longer than 16 characters?

Activity: LCD Menu

- Can you use a potentiometer as a scroll wheel for your LCD menu and a pushbutton to select menu options?
- Add apps to your Arduino!
  - Magic 8-ball
  - Rock-paper-scissors
  - More!

Activity: LED Cube

- Goal: Construct an LED cube. You should be able to individually drive each LED in the cube with your Arduino to produce some cool animations and effects.
- Could construct 3x3x3 or 4x4x4.