INTRODUCTION TO THE ARDUINO MICROCONTROLLER

Hands-on Research in Complex Systems
Shanghai Jiao Tong University
June 17 – 29, 2012

Instructor: Thomas E. Murphy (University of Maryland)
Assisted by: Hien Dao (UMD), Caitlin Williams (UMD) and 徐浩 (SJTU)
What is a Microcontroller (µC, MCU)

- Computer on a single integrated chip
  - Processor (CPU)
  - Memory (RAM / ROM / Flash)
  - I/O ports (USB, I2C, SPI, ADC)
- Common microcontroller families:
  - Intel: 4004, 8008, etc.
  - Atmel: AT and AVR
  - Microchip: PIC
  - ARM: (multiple manufacturers)
- Used in:
  - Cellphones,
  - Toys
  - Household appliances
  - Cars
  - Cameras
The ATmega328P Microcontroller
*(used by the Arduino)*

- AVR 8-bit RISC architecture
- Available in DIP package
- Up to 20 MHz clock
- 32kB flash memory
- 1 kB SRAM
- 23 programmable I/O channels
- Six 10-bit ADC inputs
- Three timers/counters
- Six PWM outputs
What is Arduino Not?

• It is not a chip (IC)
• It is not a board (PCB)
• It is not a company or a manufacturer
• It is not a programming language
• It is not a computer architecture

(although it involves all of these things...)

So what is Arduino?

It’s a *movement*, not a microcontroller:

- Founded by Massimo Banzi and David Cuartielles in 2005
- Based on “Wiring Platform”, which dates to 2003
- Open-source hardware platform
- Open source development environment
  - Easy-to learn language and libraries (based on Wiring language)
  - Integrated development environment (based on Processing programming environment)

Available for Windows / Mac / Linux
The Many Flavors of Arduino

- Arduino Uno
- Arduino Leonardo
- Arduino LilyPad
- Arduino Mega
- Arduino Nano
- Arduino Mini
- Arduino Mini Pro
- Arduino BT
Arduino-like Systems

- Cortino (ARM)
- Xduino (ARM)
- LeafLabs Maple (ARM)
- BeagleBoard (Linux)
- Wiring Board (Arduino predecessor)
Arduino Add-ons (Shields)

• TFT Touch Screen
• Data logger
• Motor/Servo shield
• Ethernet shield
• Audio wave shield
• Cellular/GSM shield
• WiFi shield
• Proto-shield
• ...many more
Where to Get an Arduino Board

• Purchase from online vendor (available worldwide)
  – Sparkfun
  – Adafruit
  – DFRobot
• ... or build your own
  – PC board
  – Solderless breadboard

http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard
Getting to know the Arduino: Electrical Inputs and Outputs

- Input voltage: 7-12 V (USB, DC plug, or Vin)
- Max output current per pin: 40 mA
Download and Install

• Download Arduino compiler and development environment from: http://arduino.cc/en/Main/Software
• Current version: 1.0.1
• Available for:
  – Windows
  – MacOX
  – Linux
• No installer needed... just unzip to a convenient location
• Before running Arduino, plug in your board using USB cable (external power is not necessary)
• When USB device is not recognized, navigate to and select the appropriate driver from the installation directory
• Run Arduino
Select your Board
Select Serial Port
Elements of the Arduino IDE

• Text editor
  – syntax and keyword coloring
  – automatic indentation
  – programming shortcuts
• Compiler
• Hardware Interface
  – Uploading programs
  – Communicating with Arduino via USB
Using the Arduino IDE

- Name of sketch
- Compile sketch
- Upload to board
- Program area
- Messages / Errors
- Serial Monitor
- Save
- Open
- New
Arduino Reference is installed locally or available online at http://arduino.cc/
Arduino Sketch Structure

• **void setup()**
  – Will be executed only when the program begins (or reset button is pressed)

• **void loop()**
  – Will be executed repeatedly

```c
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

Text that follows // is a comment (ignored by compiler)

Useful IDE Shortcut: Press Ctrl-/ to comment (or uncomment) a selected portion of your program.
Activity 1: LED Blink

- Load the “Blink” example (File → Examples → Basics → Blink)

```cpp
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);  // set the LED on
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);   // set the LED off
  delay(1000);             // wait for a second
}
```

- Compile, then upload the program
- Congratulations! you are now blinkers!
Now connect your own LED

Anatomy of an LED:

- Epoxy lens/case
- Wire bond
- Reflective cavity
- Semiconductor die
- Anvil Post
  
Symbol: +

Flat

Resistor is needed to limit current

- Resistor and LED may be interchanged (but polarity of LED is important)
- Pin 13 is special: has built-in resistor and LED
- Change program and upload

Notes:
Aside: Using a Solderless Breadboard

Connected together

Connected together

300 mils
Example: Using a Solderless Breadboard
Experimenting

- Change the blink rate
  - how fast can the LED blink (before you can no longer perceive the blinking?)
- How would you make the LED dimmer?
  - (...without changing the resistor?)
Digital Input: Reading Switches and Buttons

- Turn on/off LED based on switch
- Pin 12 reads **LOW** when switch is closed
- Pin 12 reads **HIGH** when switch is open (pull-up)

```cpp
void setup() {
  pinMode(11, OUTPUT); // Use pin 11 for digital out
  pinMode(12, INPUT); // Use pin 12 for digital input
  digitalWrite(12, HIGH); // Enable pull-up resistor
}

void loop() {
  boolean state;
  state = digitalRead(12); // read state of pin 12
  digitalWrite(11, state); // set state of pin 11 (LED)
  delay(100); // wait for a 1/10 second
}
```

Writing HIGH to an input pin: enables an internal pull-up resistor

Without the internal pull-up resistor, unconnected digital inputs could read either high or low
Activity 2: Seven-Segment Display

- Write a program that counts from 0 to 9 and displays the result on a seven-segment LED display.

- Consider writing a function:

  ```cpp
  void writeDigit(int n)
  ```

  that writes a single digit.
Seven-Segment Display Table

<table>
<thead>
<tr>
<th>Digit</th>
<th>ABCDEFG</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x7E</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>1</td>
<td>0x30</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>2</td>
<td>0x6D</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>3</td>
<td>0x79</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
</tr>
<tr>
<td>4</td>
<td>0x33</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>5</td>
<td>0x5B</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>6</td>
<td>0x5F</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>7</td>
<td>0x70</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>8</td>
<td>0x7F</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>9</td>
<td>0x7B</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>on</td>
<td>off</td>
<td>on</td>
<td>on</td>
</tr>
</tbody>
</table>

Useful:

- **bitRead(x,n)**
  - Get the value of the n\textsuperscript{th} bit of an integer x

**Example:**

- `bitRead(0x7E,7);`  // returns 1 (see table above)
Serial Communication - Writing

**Serial.begin(baud)**
Initialize serial port for communication (and sets baud rate)

**Example:**
- `Serial.begin(9600);` // 9600 baud

**Serial.print(val), Serial.print(val,fmt)**
Prints data to the serial port

**Examples:**
- `Serial.print("Hi");` // print a string
- `Serial.print(78);` // works with numbers, too
- `Serial.print(variable);` // works with variables
- `Serial.print(78,BIN);` // will print 1001110

**Serial.println(val)**
Same as `Serial.print()`, but with line-feed

**IMPORTANT:**
USB serial communication is shared with Arduino pins 0 and 1 (RX/TX)

**Note:** `Serial.end()` command is usually unnecessary, unless you need to use pins 0 & 1

Format can be: BIN, HEX, OCT, or an integer specifying the number of digits to display
Activity 3: Hello World!

- Write an Arduino program that prints the message “Hello world” to the serial port
- ...whenever you press a switch/button
- Use the Serial Monitor to see the output (Ctrl-Shift-M)
- Try increasing baud rate

Serial Monitor:

Make sure this agrees with your program, i.e., `Serial.begin(9600);`
Serial Communication - Reading

- `Serial.available()`
  Returns the number of bytes available to be read, if any

Example:
```java
if (Serial.available() > 0) {
    data = Serial.read();
}
```

To read data from serial port:
- `letter = Serial.read()`
- `letters = Serial.readBytesUntil(character, buffer, length)`
- `number = Serial.parseInt()`
- `number = Serial.parseFloat()`
Activity 4 – User Controlled Blinker

• When available (``Serial.available``), read an integer from the serial port (``Serial.parseInt``), and use the result to change the blink rate of the LED (pin 13)

Useful:
• `constrain(x,a,b)`
  Constrains the variable x to be from a to b

Examples:
– `constrain(5,1,10);` // returns 5
– `constrain(50,1,10);` // returns 10
– `constrain(0,1,10);` // returns 1
Analog Input and Sensors

- Six analog inputs: A0, A1, A2, A3, A4, A5
- AREF = Reference voltage (default = +5 V)
- 10 bit resolution:
  - returns an integer from 0 to 1023
  - result is proportional to the pin voltage
- All voltages are measured relative to GND

Note: If you need additional digital I/O, the analog pins can be re-assigned for digital use:

pinMode(A0, OUTPUT);
Reading Analog Values

• value = `analogRead(pin)`
  Reads the analog measurement on pin
  Returns integer between 0 and 1023

• `analogReference(type)`
  type can be:
  – **DEFAULT** - the default analog reference of 5 volts (on 5V Arduino boards)
  – **INTERNAL** – Built-in reference voltage (1.1 V)
  – **EXTERNAL** – AREF input pin

**Note:** Do NOT use `pinMode(A0, INPUT)` unless you want to use A0 for **DIGITAL** input.
Aside: Potentiometers
(variable resistors, rheostats)
Activity 5 – Volume Knob

- Connect the potentiometer from 5V to GND
- Use `analogRead(A0)` to measure the voltage on the center pin
- Set the LED blink rate depending on the reading
Activity 6 – Arduino Thermometer

- Build a circuit and write a sketch to read and report the temperature at 1 second intervals.
Data Logging Ideas

- **millis()**
  Returns the number of milliseconds elapsed since program started (or reset)

Time functions
- **setTime(hr,min,sec,day,month,yr)**
- **hour(), minute(), day(), month(), year()**

Real-time Clock (RTC):
- Use an external, battery-powered chip (e.g., DS1307) to provide clock
Activity 7 – Arduino Nightlight

- CdS Photoresistor: resistance depends on ambient light level

- Build a circuit and write a sketch that turns on an LED whenever it gets dark
  **Hint:** connect the photoresistor in a voltage divider
Analog Output?

- Most microcontrollers have only digital outputs
- **Pulse-width Modulation**: Analog variables can be represented by the duty-cycle (or pulse-width) of a digital signal

http://arduino.cc/en/Tutorial/PWM
PulseWidth Modulation (PWM)

- analogWrite(pin,val)
  set the PWM fraction:
  - val = 0: always off
  - val = 255: always on
- Remember to designate pin for digital output:
  pinMode(pin,OUTPUT);
  (usually in setup)
- Default PWM frequency:
  - 16 MHz / $2^{15}$ = 488.28125 Hz

Note: the PWM frequency and resolution can be changed by re-configuring the timers
Activity 8 – PWM LED Dimmer

• Use PWM to control the brightness of an LED
  – connect LED to pin 3, 5, 6, 9, 10 or 11
  – remember to use 220 Ω current-limiting resistor
• Set the brightness from the serial port, or potentiometer
• Watch the output on an oscilloscope

Useful:
• $\text{newValue} = \text{map}(\text{oldValue}, a, b, c, d)$
  Converts/maps a number in the range $(a:b)$ to a new number in the range $(c:d)$

$Example:$
  – $\text{newValue} = \text{map}(\text{oldValue}, 0, 1023, 0, 255);$
Activity 8 – PWM LED Dimmer (cont’d)

• Change your program to sinusoidally modulate the intensity of the LED, at a 1 Hz rate
  – *Hint*: use the `millis()`, `sin()`, and `analogWrite()` functions
Servomotors

- **Standard servo:**
  - PWM duty cycle controls direction:
  - 0% duty cycle $\rightarrow$ 0 degrees
  - 100% duty cycle $\rightarrow$ 180 degrees

- **Continuous-rotation servo:**
  - duty cycle sets speed and/or direction

[Diagram showing standard servo pinout and connection to microcontroller]

[Table showing standard servo pin descriptions and voltage specifications]

http://www.parallax.com/
Activity 9 – Servomotor Control

• Build a program that turns a servomotor from 0 to 180 degrees, based on potentiometer reading
• Report setting to the serial monitor
Solid State Switching - MOSFETs

- Logic-level MOSFET (requires only 5 V)
- Acts like a voltage-controlled switch
- Works with PWM!
Activity 10 – PWM Speed Control

• Build a circuit to control the speed of a motor using a PWM-controlled MOSFET
• Enter the speed (PWM setting) from the serial port (`Serial.parseInt`)
Controlling Relays and Solenoids

- Electromechanically-actuated switch
- Provides electrical isolation
- Typically few ms response time

Note: Arduino cannot supply enough current to drive relay coil
Relay Driver Circuit

- NPN transistor: acts like a current-controlled switch
- MOSFET will also work
- Diode prevents back-EMF (associated with inductive loads)
- Coil voltage supply and Arduino share common GND
Activity 11: Bidirectional Motor Driver

• Build a circuit (and write an Arduino sketch) that will use a DPDT relay to change the direction of a DC motor:

(Note: this is called an H-bridge circuit. It can also be made with transistors)
Communication: I\textsuperscript{2}C, SPI

• I\textsuperscript{2}C (Inter-Integrated Circuit)
  – Developed by Phillips
  – Speed = 100 kHz, 400 kHz, and 3.4 MHz (not supported by Arduino)
  – Two bi-directional lines: SDA, SCL
  – Multiple slaves can share same bus

• SPI (Serial Peripheral Interface Bus)
  – Speed = 1-100 MHz (clock/device limited)
  – Four-wire bus: SCLK, MOSI, MISO, SS
  – Multiple slaves can share same bus (but each needs a dedicated SS, slave select)
Connecting Multiple Devices (I²C and SPI)

Master (µC) with three I²C slaves:

Master with three SPI slaves:

http://en.wikipedia.org/
SPI and I²C on the Arduino

**SPI pins:**
- SCK = serial clock
- MISO = master in, slave out
- MOSI = master out slave in
- SS = slave select

**I²C pins:**
- SDA = data line
- SCL = clock line
# Basic Arduino I²C Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wire.begin()</td>
<td>Join the I²C bus as master (usually invoked in setup)</td>
</tr>
<tr>
<td>Wire.beginTransmission(address)</td>
<td>Begin communicating to a slave device</td>
</tr>
<tr>
<td>Wire.write(byte)</td>
<td>Write one byte to I²C bus (after request)</td>
</tr>
<tr>
<td>Wire.endTransmission(address)</td>
<td>End transmission to slave device</td>
</tr>
</tbody>
</table>

**Note:** you must include the Wire library:  
#include `<Wire.h>`

**Note:** pinMode() not needed for I²C on pins A4 and A5
Example: MCP4725 12-bit DAC

MCP4725 write command (taken from data sheet)

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 0</td>
<td>C2</td>
<td>D11</td>
<td>D3</td>
</tr>
<tr>
<td>A2 A1 A0 0</td>
<td>C1</td>
<td>D10</td>
<td>D2</td>
</tr>
<tr>
<td>7-bit I²C address (1100000)</td>
<td>C0</td>
<td>D9</td>
<td>D1</td>
</tr>
<tr>
<td>x x</td>
<td>PDI</td>
<td>D8</td>
<td>D0</td>
</tr>
<tr>
<td>command (010)</td>
<td>x x</td>
<td>D7</td>
<td>x</td>
</tr>
<tr>
<td>power down mode (00)</td>
<td>x</td>
<td>D6</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D5</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D4</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D3</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D2</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D1</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>D0</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>data bits (MSB → LSB)</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Note: binary numbers are preceded by B:

B1100000 = 96

Arduino program segment:

```cpp
Wire.beginTransmission(B1100000); // Byte 1 (Initiate communication)
Wire.write(B01000000); // Byte 2 (command and power down mode)
Wire.write(data >> 4); // Byte 3 (send bits D11..D4)
Wire.write((data & B00001111) << 4); // Byte 4 (send bits D3..D0)
Wire.endTransmission();
```

Remember: you must include the Wire library at the top:

```cpp
#include <Wire.h>
```
and you must also use `Wire.begin()` in setup
# Additional I²C Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wire.begin()</td>
<td>Join the I²C bus as master (usually invoked in <code>setup</code>)</td>
</tr>
<tr>
<td>Wire.begin(address)</td>
<td>Join the I²C bus as slave, with address specified (usually invoked in <code>setup</code>)</td>
</tr>
<tr>
<td>Wire.beginTransmission(address)</td>
<td>Begin communicating to a slave device</td>
</tr>
<tr>
<td>Wire.write(byte)</td>
<td>Write one byte to I²C bus (after request)</td>
</tr>
<tr>
<td>Wire.write(bytes,length)</td>
<td>Write length bytes to I²C bus</td>
</tr>
<tr>
<td>Wire.endTransmission(address)</td>
<td>End transmission to slave device</td>
</tr>
<tr>
<td>Wire.requestFrom(address, quantity)</td>
<td>Request bytes (quantity) from slave</td>
</tr>
<tr>
<td>Wire.requestFrom(address, quantity, stop)</td>
<td>Request bytes (quantity) from slave with stop flag</td>
</tr>
<tr>
<td>Wire.available()</td>
<td>The number of bytes available for reading</td>
</tr>
<tr>
<td>Wire.read()</td>
<td>Reads a byte that was transmitted from a slave. (Preceded by <code>Wire.requestFrom</code>)</td>
</tr>
</tbody>
</table>

**Note:** you must include the Wire library: 
```
#include <Wire.h>
```

**Note:** `pinMode()` not needed for I²C on pins A4 and A5
Activity 12: Sawtooth Wave

- Program the MCP4725 DAC to produce a sawtooth (ramp) wave:
  - What is the frequency of the sawtooth wave?
  - Can you make $f = 100$ Hz?

Note: the I²C bus requires pull-up resistors on SCL and SDA (provided on the board)

MCP4725 breakout board:

http://www.sparkfun.com/
# Basic Arduino SPI Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI.begin()</td>
<td>Initializes the SPI bus, setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low and SS high.</td>
</tr>
<tr>
<td>byteIn = SPI.transfer(byteOut)</td>
<td>Transfer one byte (both send and receive) returns the received byte</td>
</tr>
</tbody>
</table>

**Note:** you must include the SPI library: 
```
#include <SPI.h>
```

**Note:** pinMode() not needed. It is automatically configured in SPI.begin()
## Additional Arduino SPI Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPI.begin()</td>
<td>Initializes the SPI bus, setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low and SS high.</td>
</tr>
<tr>
<td>SPI.end()</td>
<td>Disables the SPI bus (leaving pin modes unchanged) – in case you need to use pins 10-13 again.</td>
</tr>
<tr>
<td>SPI.setBitOrder(order)</td>
<td>Set bit order for SPI ( \text{order} = {\text{LSBFIRST, MSBFIRST}} )</td>
</tr>
<tr>
<td>SPI.setClockDivider(divider)</td>
<td>Set the SPI clock divider ( \text{divider} = {2, 4, 8, 16, 32, 64, 128} ). SPI clock speed = 16 MHz/divider</td>
</tr>
<tr>
<td>SPI.setDataMode(mode)</td>
<td>Set the SPI data mode ( \text{mode} = {\text{SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3}} )</td>
</tr>
<tr>
<td>SPI.transfer(byte)</td>
<td>Transfer one byte (both send and receive) returns the received byte</td>
</tr>
</tbody>
</table>

**Note:** you must include the SPI library: #include `<SPI.h>`  
**Note:** pinMode() not needed
Example: AD5206 Digital Potentiometer

**Features:**
- six independent, 3-wiper potentiometers
- 8-bit precision (256 possible levels)
- Available in 10kΩ, 50kΩ and 100kΩ
- Programmed through SPI interface
AD5206 Write Sequence

Arduino program segment:

```c
SPI.begin(); // initialize SPI (in setup)
...
digitalWrite(SS, LOW); // hold SS pin low to select chip
SPI.transfer(potnumber); // determine which pot (0..5)
SPI.transfer(wipervalue); // transfer 8-bit wiper setting
digitalWrite(SS, HIGH); // de-select the chip
```

Notes:
- same as MOSI (master out slave in)
- same as SS (slave select)
Activity 13: Programmable Voltage Divider

- Use the AD5206 to build a programmable voltage divider
- Allow the user to set the resistance from the serial port
- Measure resistance with an Ohm meter, or using `analogRead()`
AD5206: Summary of Pins and Commands

- **SS (10)**: hold SS pin low to select chip
- **MOSI (11)**: determine which pot (0..5)
- **MISO (12)**: transfer 8-bit wiper setting
- **SCK (13)**: de-select the chip

Remember: `SPI.begin()` needed in `setup()` and `#include <SPI.h>`

```c
digitalWrite(SS, LOW);  // hold SS pin low to select chip
SPI.transfer(potnumber); // determine which pot (0..5)
SPI.transfer(wipervalue); // transfer 8-bit wiper setting
digitalWrite(SS, HIGH);  // de-select the chip
```