ARDUINO BASICS

Prepared by: Achilles C. Macunlay

Updated: 2021-06-09

Part 1: Introduction

- **What is a Microcontroller?**

A microcontroller is a compact integrated circuit used to govern specific operation/s in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.

A microcontroller is different from a microprocessor. Microprocessors are commonly used for intensive processing such as personal computers, laptops, mobile devices, and video games while microcontrollers are usually used when the given task is fixed and predefined such as commercial appliances.

- **What is Arduino?**

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards can read inputs such as light on a sensor, a finger on a button, or a Twitter message, and turn it into an output like activating a motor, turning on an LED, or publishing something online. You can tell the board what to do by sending a set of instructions to the microcontroller on the board.

- **Kinds of Arduino Boards**

The following boards are the commonly used ones for projects. Only the arrangements of the pins change mostly in each board. Although the smaller the board gets, the fewer features it will have. This does not mean that bigger boards are much better than smaller ones. When choosing a board for your project, you should also consider other factors such as price, compatibility, power efficiency, and portability. You are highly encouraged to explore further which Arduino board suits your specific needs:

1. Arduino Uno

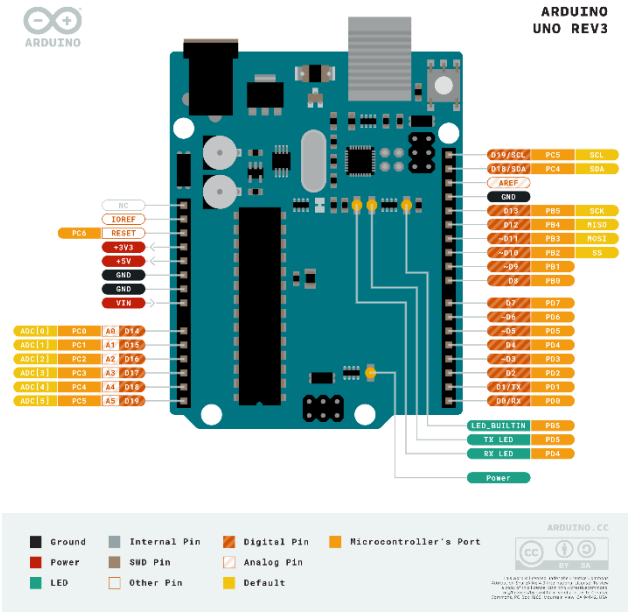


Fig. 1: Arduino Uno Pin Configuration

2. Arduino Micro

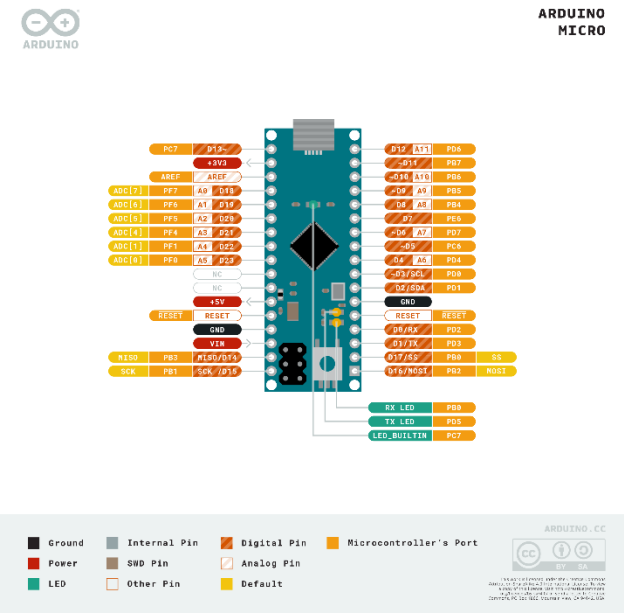


Fig. 2: Arduino Micro Pin Configuration

3. Arduino Nano

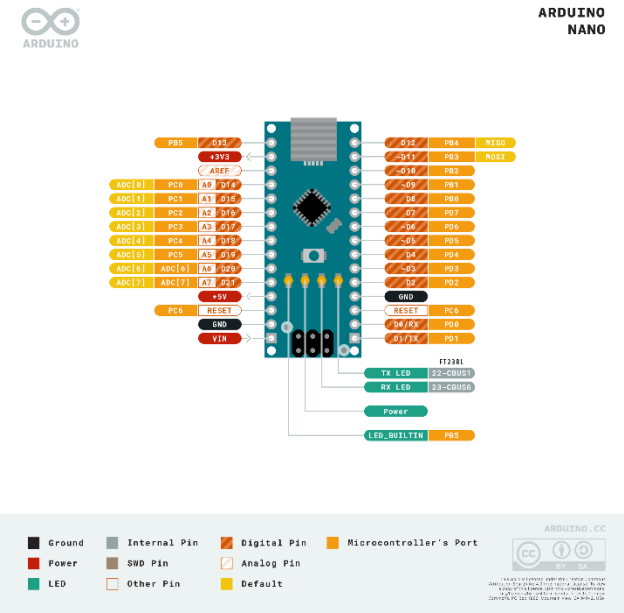


Fig. 3: Arduino Nano Pin Configuration

4. Arduino Mega

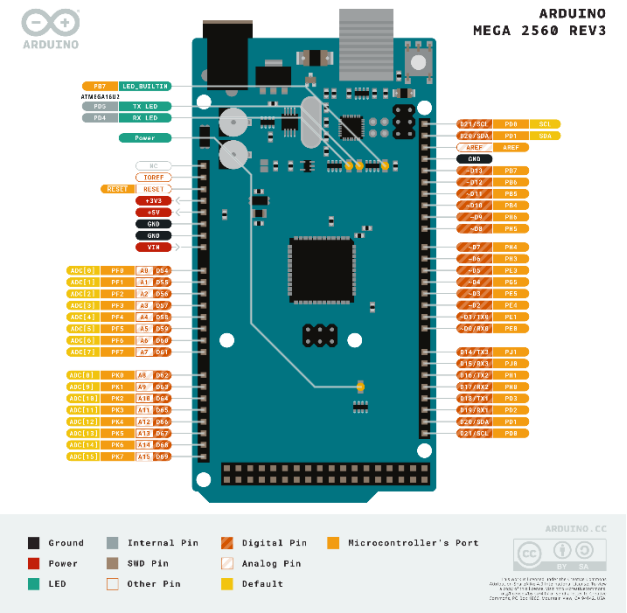


Fig. 4: Arduino Mega Pin Configuration

Part 2: The Arduino Uno Board

- **Specifications:**

Table 1: Arduino Uno Technical Specifications

Microcontroller	ATmega328P
Operating Voltage	5 V
Input Voltage (limit)	6 - 20 V
Digital I/O Pins	14 (6 supports PWM)
PWM Digital I/O Pins	6 (D3, D5, D6, D9, D10, D11)
Analog Input Pins	6 (A0, A1, A2, A3, A4, A5)
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P; 0.5 KB for bootloader)
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	Pin 13 (D13)
Length	68.6 mm
Width	53.4 mm
Weight	25 g

- **Pin Configuration:**

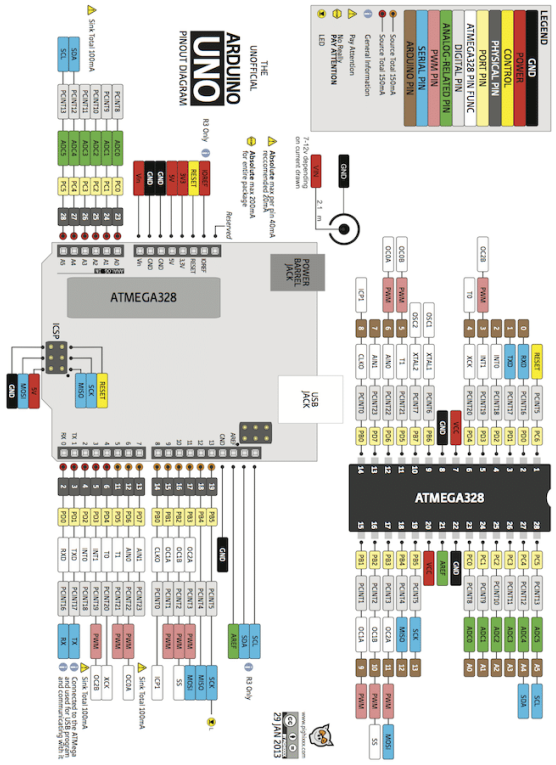


Fig. 5: Arduino Uno Detailed Pin Configuration

- **Ways to Power the Board:**

1. *Using USB Cable*

The USB port of the Arduino Uno can be connected to a desktop or laptop. *If the computer recognizes the device, the current supplied to the board is 500 mA at 5 V. Otherwise, 100m A is supplied at 5 V.* For the demo in Part 4, we will be using this method.

2. *Using an AC to DC Adapter Plugged into the Barrel Connector*

The barrel connector can be supplied with an input of 7 – 12 V. This is regulated to 5 V by the onboard voltage regulator, and the board is powered on.

3. *Using 5 V Input Pin*

The Arduino Uno board can also be powered using the 5 V and GND pins, given that the input voltage is steady and regulated 5 V. *The 5 V pin does not pass through the board's voltage regulator and all the safety measures present on the Arduino Uno*, so if the input exceeds 5 V (5.5 V being the maximum upper limit), the board can be damaged. It is generally advised to avoid powering up the Arduino Uno using this method.

4. *Using Batteries Greater Than 5 V*

Connect a 9 V battery with the positive terminal connected to the Vin pin and the negative terminal connected to the GND pin. *The Vin port allows an input between 7 and 12 V but using a 9 V battery is recommended*. Depending on your application you can input 12 V too but make sure the current values stay around 500 mA.

Part 3: Programming Basics

- **Common Data Types**

1. String (`String`)

The string type is used to store a sequence of characters (text). This is not a built-in type, but it behaves like one in its most basic usage. String values must be surrounded by double quotes (“`This is a string.`”).

2. Boolean (`bool`)

A `bool` holds one of two values, `true` or `false`. (Each `bool` variable occupies one byte (8 bits) of memory.)

3. Byte (`byte`)

A `byte` stores an 8-bit unsigned number, from 0 to 255.

4. Integer (`int`)

The integer data type uses 4 bytes (32 bits) of memory and stores whole numbers that range from - 2 147 483 647 to + 2 147 483 647 if signed and 0 to 4 294 967 294 if unsigned.

- **Control Statements**

`1. if … else`

In an `if` statement, if the specified condition/s are met, then the code inside the `if` block is executed. Otherwise, it will "skip" the `if` block.

```
bool p1 = trainIsAppoaching();
bool p2 = vehicleWantsToCross();
if (p1 && p2) {
        blockTheIntersection();
}
doOtherTasks();
```

In an `if … else` statement, if the specified conditions in the `if` statement are not met, the code in the `else` block is executed.

```
bool p1 = trainIsAppoaching();
bool p2 = vehicleWantsToCross();
if (p1 && p2) {
        blockTheIntersection();
} else {
        openTheIntersection();
}
```

`2. else if`

`else if` statements are used to specify a new condition to test, if the first condition is `false`.

```
bool p1 = trainIsAppoaching();
bool p2 = trainIsCrossing();
bool p3 = trainHasPassedThrough();
if (p1) {
        blockTheIntersection();
} else if (p2) {
        blockTheIntersection();
} else if (p3) {
        openTheIntersection();
} else {
        checkForApproachingTrain();
}
```

`3. switch … case`

Like `if` statements, `switch case` controls the flow of code by allowing programmers to specify different code that should be executed in various conditions. A `switch` statement compares the value of a variable to the values specified in case statements.

When a `case` statement is found whose value matches that of the variable, the code in that case statement is run.

The `break` keyword exits the `switch` statement and is typically used at the end of each case. Without a `break` statement, the `switch` statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

```
byte situation = currentSituation();
switch (situation) {
case noApproachingTrain:
        openTheIntersection();
        break;
case trainIsApproaching:
        blockTheIntersection();
        break;
case trainIsCrossing:
        blockTheIntersection();
        break;
case trainHasPassed:
        openTheIntersection();
        break;
default:
        checkForApproachingTrain();
        break;
}
```

- Common Loop Statements

1. `for` loop

`for` statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. `for` statement is useful for any repetitive operation and is often used in combination with arrays to operate on collections of data/pins.

```
byte end = 255;
byte i = 0;
for (i; i <= end; i ++) {
        sendToRegister(i);
        delay(100);
}
```

2. `while` loop

A `while` loop will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

```
byte end = 255;
byte i = 0;
while (i <= end) {
        sendToRegister(i);
        delay(100);
        i ++;
}
```

Part 4: Demo (Shift Register)

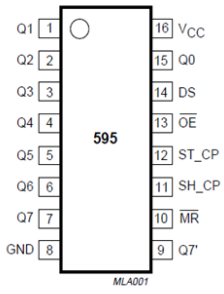
- **What is a Shift Register?**

A shift register allows you to expand the number of I/O pins you can use from your Arduino (or any microcontroller for that matter) and the 74HC595 shift register (nicknamed '595'), a popular shift register.

- **What is a Shift Register's Purpose?**

Shift registers are often used for the purpose of saving pins on the microcontroller, as every microcontroller has a limited number of I/O pins (GPIO). A shift register can even be connected to another shift register to further expand the number of pins you can use. This procedure is called *daisy chaining*. When doing this, *keep in mind the limitations of the Arduino Uno board especially its current limit* (referred to as **DC Current per I/O Pin** in Table 1). A good practice when doing daisy chains is to *draw the power to be used from a power supply rather than from the Arduino Uno board's 5 V pin*.

- **How Does a Shift Register Work?**



PINNING

PIN	SYMBOL	DESCRIPTION
1	Q1	parallel data output
2	Q2	parallel data output
3	Q3	parallel data output
4	Q4	parallel data output
5	Q5	parallel data output
6	Q6	parallel data output
7	Q7	parallel data output
8	GND	ground (0 V)
9	Q7'	serial data output
10	$\overline{MR}$	master reset (active LOW)
11	SH_CP	shift register clock input
12	ST_CP	storage register clock input
13	$\overline{OE}$	output enable (active LOW)
14	DS	serial data input
15	Q0	parallel data output
16	V_{CC}	positive supply voltage

Fig. 6: 74HC595 Pin Configuration

From this point on, the pin labels that will be used will be based on Figs. 1 and 6 for uniformity's sake.

Inputs:

ground (Pin 8, GND) – This must be connected to the same ground the Arduino board is connected to.

serial data output (Pin 9, Q7') – If you want to daisy chain another 595, this pin must be connected to the other 595. In this demo, this pin will not be used.

master reset (Pin 10; MR) – This allows us to clear the current value stored by the 595. If you want to reset the 595, set it to LOW. Otherwise, keep its value HIGH.

clock (Pin 11; SH_CP) – The clock of the Arduino board will be fed to this pin. By doing so, the 595 will be in sync with the Arduino board.

latch (Pin 12; ST_CP) – This acts as a "lock" for the shift register. When turned off (LOW), the Arduino board can change the current value stored in the 595. When turned on (HIGH), the 595 will "save" the new value and display it in its output pins.

output enabled (Pin 13; OE) – When set to HIGH, pins Q0 – Q7 will not show the stored value.

data (Pin 14; DS) – This is where data will be fed to the 595. In this demo, this pin will receive bytes with values ranging from 0 to 255 from the Arduino board.

voltage supply (Pin 16, VCC) – The power to be used by the 595 will be fed through this pin. The voltage must be regulated 5 V.

Outputs:

Q0 (Pin 15), Q1 (Pin 1), Q2 (Pin 2), Q3 (Pin 3), Q4 (Pin 4), Q5 (Pin 5), Q6 (Pin 6), Q7 (Pin 7)

Each pin from Q0 – Q7 represents a bit of the value stored in the 595. Since there are only 8 pins available to output a value, the 595 can store 256 (2^8) possible values (hence the 0 – 255 range). Table 2 shows the values each pin from Q0 – Q7 represents and examples on how the output pins will represent a value. Keep in mind that the values that each pin represents can be fully inverted (i.e., Q0 can represent either 2^0 or 2^7, Q1 can represent either 2^1 or 2^6 and so on.).

Table 2: 595 Output and Sample Values

	Pins							
	Q0	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
2^n	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
Output	1	2	4	8	16	32	64	128
Sample Value	**Output**							
0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	0
99	1	1	0	0	0	1	1	0
129	1	0	0	0	0	0	0	1
255	1	1	1	1	1	1	1	1

- **Setup**

Table 3 shows how the Arduino board will be connected to the 74HC595 shift register. The pin names used in the table are based on Figs. 1 and 6.

Table 3: Connection of Arduino Pins and the 595

Arduino's Pin…	Connected to 74HC595's Pin…
D4	Pin 14 (DS)
D5	Pin 12 (ST_CP)
D6	Pin 11 (SH_CP)
5V	Pin 10 (MR) Pin 16 (VCC)
GND	Pin 08 (GND) Pin 13 (OE)

After connecting the Arduino board to the 595, connect each output pin of the 595 (Q0 – Q7) to an LED and resistor in series as shown in Fig. 7 where QX is the output pin of the 595. You should have 8 LEDs and 8 resistors each connected to every output pin of the 595.

Hint:

For LEDs, the shorter leg is the negative (-) side and must be connected to GND. Resistors have no positive or negative side. It can be placed in any direction.

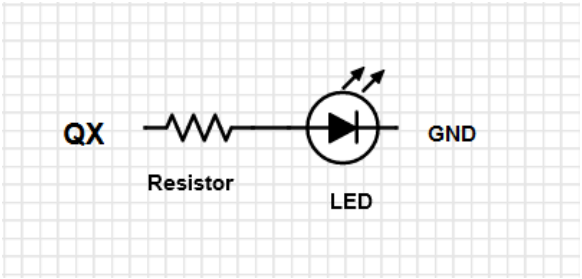


Fig. 7: 74HC595 Output Pins and LED Connection

After doing all the wiring, you should have a circuit like what is shown in Fig. 8.

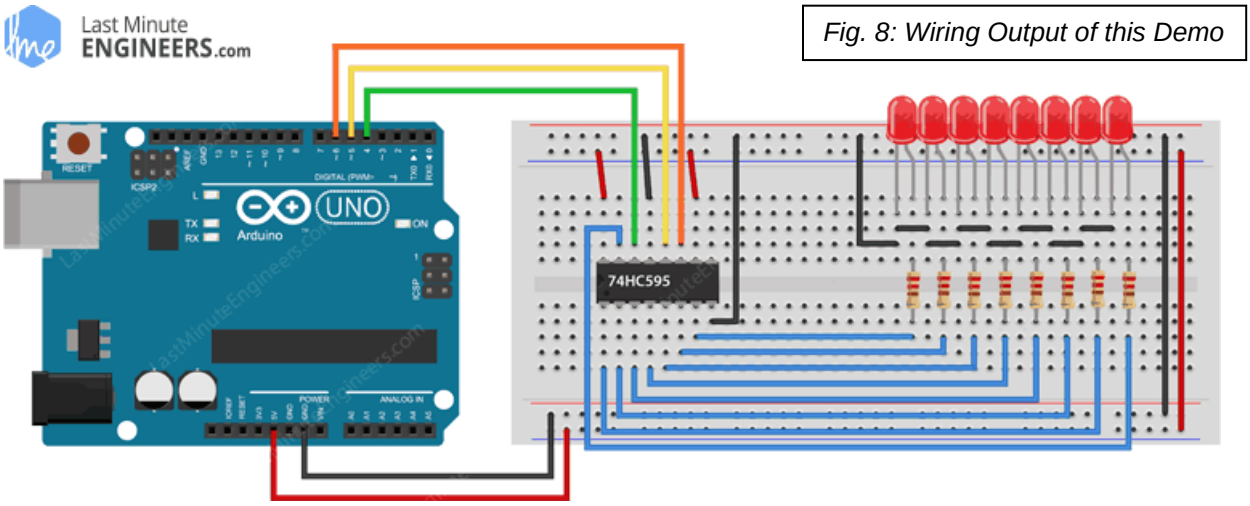


Fig. 8: Wiring Output of this Demo

- **The Code**

The following is the working code for this demo. For further explanation of the code's inner workings, refer to these URLs:

https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/

https://www.oomlout.com/oom.php/products/ardx/circ-05-1.html

```
int latch = 5; // latch (Pin 12) of the 595 connected to D5
int clock = 6; // clock (Pin 11) of the 595 connected to D6
int data = 4;  //  data (Pin 14) of the 595 connected to D4

void setup() {
  // each pin is set up to send out bits
  pinMode(latch, OUTPUT);
  pinMode(data, OUTPUT);
  pinMode(clock, OUTPUT);
}

void loop() {
  delay(1000);
  for (int i = 0; i < 256; i++) {
    updateRegister(i);
  }
}

void updateRegister (int value) {
   digitalWrite(latch, LOW); // unlocks the 595
   shiftOut(data, clock, MSBFIRST, value); // feeds the new value
   digitalWrite(latch, HIGH); // locks the 595
}
```

- **Code Explained**

The following explains what is happening in the code in order and its flow:

1. The `void setup() {…}` function will be executed once the Arduino board is turned on or when the reset button in the board is pressed.
2. After the Arduino board finishes executing the code in the `void setup() {…}` function, it will directly execute the code in the `void loop() {…}` function.
3. In the `void loop() {…}` function, the code will pause for 1000 ms before executing the `for` loop.
4. In the for loop, it will cyclically enter values contained in the variable `i` from 0 to 255 to the `updateRegister()` function.
5. In the `updateRegister()` function, the latch is unlocked, the new value is fed, and the latched gets locked again.
6. The code will then execute step 3.

REFERENCES

- https://www.allaboutcircuits.com/technical-articles/what-is-a-microcontroller-introduction-component-characteristics-component/
- https://components101.com/articles/difference-between-microprocessor-and-microcontroller
- https://www.arduino.cc/en/guide/introduction
- https://store.arduino.cc/usa/arduino-uno-rev3
- https://store.arduino.cc/usa/arduino-micro
- https://store.arduino.cc/usa/arduino-nano
- https://store.arduino.cc/usa/mega-2560-r3
- https://www.researchgate.net/figure/Figure-A1-Arduino-Uno-Pinout_fig57_275651047
- https://technobyte.org/2016/07/power-up-the-arduino-uno/
- https://www.w3schools.com/cpp/cpp_data_types_string.asp
- https://www.arduino.cc/reference/en/language/variables/data-types/byte/
- https://www.arduino.cc/reference/en/language/variables/data-types/bool/
- https://www.w3schools.com/cpp/cpp_conditions.asp
- https://www.arduino.cc/reference/en/language/structure/control-structure/switchcase/
- https://www.arduino.cc/reference/en/language/structure/control-structure/for/
- https://www.arduino.cc/reference/en/language/structure/control-structure/while/
- https://lastminuteengineers.com/74hc595-shift-register-arduino-tutorial/
- https://www.oomlout.com/oom.php/products/ardx/
- https://www.electronics-lab.com/project/3-wire-serial-lcd-using-a-shift-register/