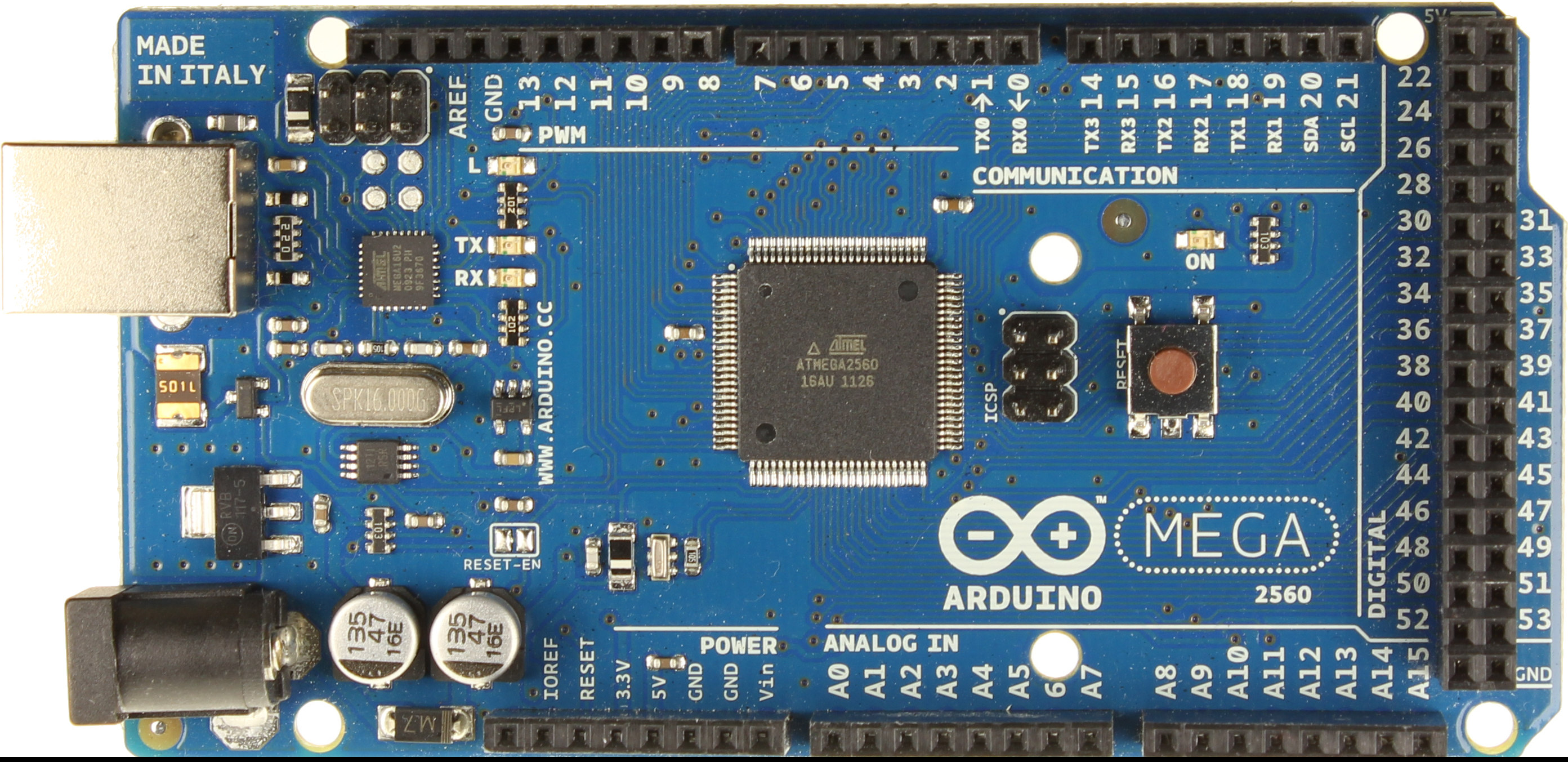
Tutorial for Arduino

**Hardware part of Arduino**

The Arduino board mentioned in this tutorial is Arduino Mega2560, which is a microcontroller board based on the ATmega2560. Users can program, input analog/digital signal and output analog/digital signal. As we can see clearly from the board below, the lower part of the board power pins. The Arduino board can be powered either with an external power supply or via the USB connection, which is located at the left of the picture below. If using external power supply, we should use the Vin pin in the power category; if using USB connection, the board should be connected to computer with a USB wire.

At the lower part of the Arduino board, we can also see fifteen analog-in pins, which can send analog signals into the microcontroller if connected. Analog signal is continuous signal varying with time, while digital signal is used to represent a sequence of discrete values. The PWM 0-13 pins can be used to both write digital signal out and read digital signal. The PWN pin13 is a little special because it is connected to a built-in LED. When the pin is HIGH, the LED is on; when the pin is LOW value, the LED in off.



Programming

One of the most widely used functions of Arduino is programming. After connecting the board to computer with a USB wire and downloading the Arduino software (http://arduino.cc/en/Main/Software), we are ready to program. The programming language of Arduino is based on C language, so most of the C language works for programming.

To start things up, two basic functions is required: *void setup()* and *void loop()*. The *setup()* function is used to initialize variables, pin modes, etc. To initialize, the function *pinMode is* used. The *pinMode* function takes two parameters: pin number and INPUT/OUTPUT.

Ex1 now we want to initialize pin3 as our digital output, the codes go:

void setup() {

pinMode(3, OUTPUT);

}

The loop() function, as its name is, will run the command in it repeatedly.

Ex2 If we want the built-in LED connected to pin13 to blink repeatedly, all we need to do is to write a function in loop() and LED will keep blinking. The next step is how to make the LED blink. Theoretically, the LED is on when its voltage is HIGH; the LED is off when its voltage is LOW. Now all we have to do is to set its voltage HIGH or LOW. To accomplish this, we can use the digitalWrite function. This function will output digital signal to whatever pins we assign. It takes two parameters: the pin we are about to write digital signal to and the content we are about to write.

Codes:

void setup(){

pinMode(13, OUTPUT); //set pin13 as our output pin

}

void loop(){

digitalWrite(13, HIGH);

digitalWrite(13, LOW);

}

After finishing the codes, we want to see how it works. We can hit the verify button on the screen, and after that, hit upload button. Once upload is done, we can observe the behavior.

(picture here)

So now the LED is blinking consecutively. However, to make the blinking more obvious, we can add a delay function after each of the digitalWrite function, which will delay the time of on and off.

void loop(){

digitalWrite(13, HIGH);

delay(1000);

digitalWrite(13, LOW);

delay(1000);

}

After learning how to use digitalWrite function, we will move on to the digitalRead function. In contrast to digitalWrite, digitalRead function reads digital signal.

Ex3 If we connect a clock signal to one of the digital pin, i.e. pin7, we can use the digitalRead function to read the clock signal into the microcontroller. digitalRead function takes only one parameter: the pin number we wish to read signal from and return the value it reads.

Codes:

int datainfo; //initialize a variable to hold digital signal

void setup(){

pinMode(7, INPUT); //as usual, set pin7 as our input pin. Make sure signal is already connected to the pin we assign

}

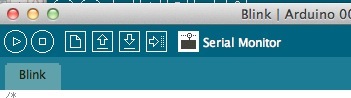
void loop(){

datainfo=digitalRead(7); // read signal from pin 7

}

One thing worth to mention is that due to the loop function, datainfo will update to the new signal value every time loop() is executed.

Of course, after we are able to read signal, we wish to see the signal somewhere. Serial monitor is very helpful here. Serial monitor is a tool in Arduino software used to monitor the digital/analog signal we input. We can print the signal out with printIn function.



To make serial monitor work, we should first initialize it in setup() function using a line of code:

Serial.begin(9600); //9600 is how fast we want the monitor to read our signal

In the loop() function, we also have to insert a line of code to ask serial monitor to print our input signal:

Serial.println(datainfo);

Completely, we can now write our codes:

int datainfo;

void setup(){

Serial.begin(9600);

pinMode(7, INPUT);

}

void loop(){

datainfo=digitalRead(7);

Serial.println(datainfo);

}

Similarly, to read analog signal (such as from potentiometer), we just need to modify the codes above a little.

Codes:

int datainfo;

void setup(){

Serial.begin(9600);

pinMode(A0, INPUT); //different with the digital pins, the analog pins are A0 – A15

}

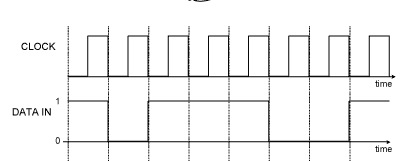
void loop(){

datainfo=analogRead(7); //we want to change digitalRead into analogRead here

Serial.println(datainfo);

}

Next step we will look at two signal inputs instead of one input. Suppose we have a positive-triggered clock input and a data input. We expect that Arduino will read data input only when clock pulses from 0 to 1.



According to our expectation and the timing graph above, we want our Arduino to read data：10111001.

The problem left for us now is how to tell Arduino to read data in when clock pulses, but not to read data in at any other time? Now we will forget about data input for a while, and do state change detection. The principle is to print “clock impulse” to serial monitor when clock pulses from 0 to 1.

Code:

int clockpulse=0; //initialize variable clockpulse to be 0

int laststate=0; //laststate is to detect when clock impulse changes

void setup(){

Serial.begin(9600);

pinMode(7, INPUT); //set pin7 as our digital signal-in pin

}

void loop(){

clockpulse=digitalRead(7);

if(clockpulse != laststate) //if the clock state changes, see if the current state is HIGH

{

if(clockpulse==HIGH) //if the current state is HIGH, that means clock pulses from 0 to 1

{

println(“state changes!”);

}

}

laststate=clockpulse; //store the current state

}

With this state change detection code, we can easily add our data input to the code with minor changes so that we can see our data printed out on the serial monitor.

Code:

int clockpulse=0; //initialize variable clockpulse to be 0

int laststate=0; //laststate is to detect when clock impulse changes

int datainfo;

void setup(){

Serial.begin(9600);

pinMode(7, INPUT); //set pin7 as our digital signal-in pin

pinMode(8, INPUT);

}

void loop(){

clockpulse=digitalRead(7);

if(clockpulse != laststate) //if the clock state changes, see if the current state is HIGH

{

if(clockpulse==HIGH) //if the current state is HIGH, that means clock pulses from 0 to 1

{

datainfo=digitalRead(8);

Serial.println(datainfo);

}

}

laststate=clockpulse; //store the current state

}