

Al-Mustaqbal University
College Of Engineering & Technology
Department of Computer Engineering Techniques
(Stage: 3)

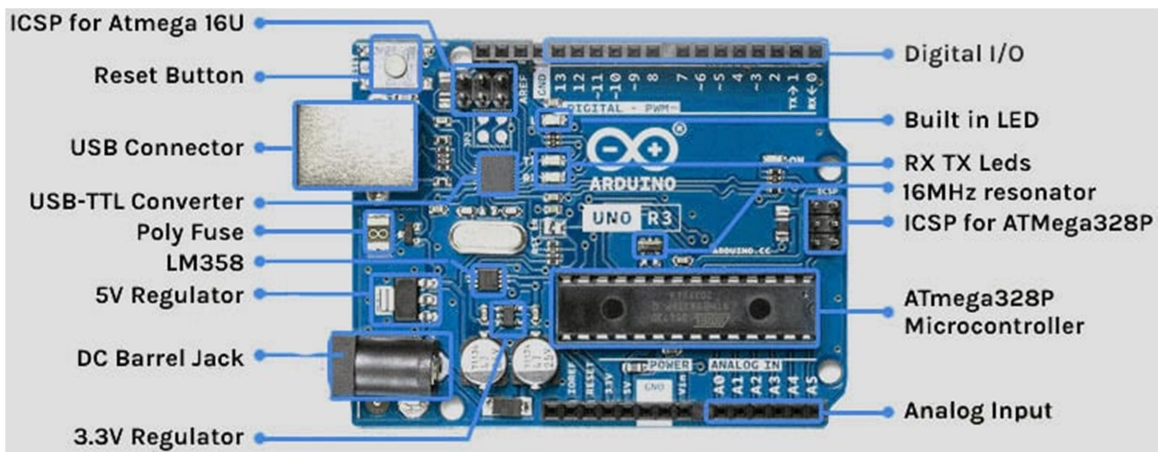
Digital Control

Lecture 1

Introduction to Arduino

Dr.: Fanar Ali Joda

Arduino



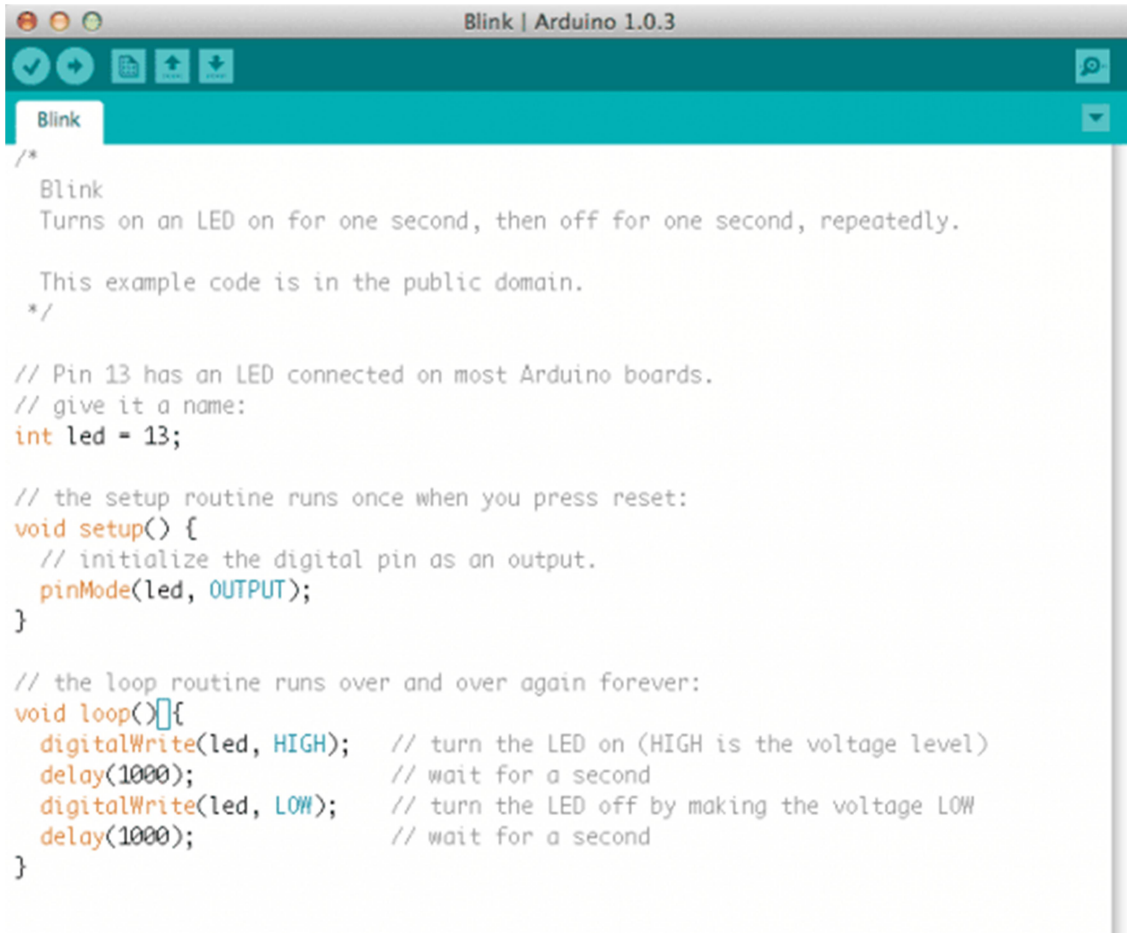
- Analog Reference pin
- Digital Ground
- Digital Pins 2-13 – the digital pins on a board can be used for general-purpose input and output via the `pinMode()`, `digitalRead()`, and `digitalWrite()` commands. Each pin has an internal pull-up resistor which can be turned on and off using `digitalWrite()` (w/ a value of HIGH or LOW, respectively)

when the pin is configured as an input. The maximum current per pin is 40 mA.

- Digital Pins 0-1/Serial In/Out – TX/RX- These pins cannot be used for digital i/o (`digitalRead` and `digitalWrite`) if you are also using serial communication (e.g. `Serial.begin`).
- Reset Button – S1
- In-circuit Serial Programmer
- Analog In Pins 0-5 – The analog input pins support 10-bit analog-to-digital conversion (ADC) using the `analogRead()` function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.
- Power and Ground Pins
- External Power Supply In (9-12VDC) – X1
- Toggles External Power and USB Power (place jumper on two pins closest to desired supply) – SV1
- USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board)
- Arduino Uno has a low drop-out voltage regulator. It dissipates less power in the form of heat.

Arduino Integrated Development Environment (IDE)

Arduino IDE allows you to write programs and upload them on your board. It is available as an online tool that allows you to save designs on the cloud. However, it is also available as an offline tool. The boards work out of the box with the web editor.

A screenshot of the Arduino IDE interface. The title bar reads "Blink | Arduino 1.0.3". The menu bar includes "Blink" and a dropdown arrow. The code editor contains the following text:

```
/*  
  Blink  
  Turns on an LED on for one second, then off for one second, repeatedly.  
  
  This example code is in the public domain.  
  */  
  
// Pin 13 has an LED connected on most Arduino boards.  
// give it a name:  
int led = 13;  
  
// the setup routine runs once when you press reset:  
void setup() {  
  // initialize the digital pin as an output.  
  pinMode(led, OUTPUT);  
}  
  
// the loop routine runs over and over again forever:  
void loop(){  
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)  
  delay(1000);             // wait for a second  
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW  
  delay(1000);             // wait for a second  
}
```

The online IDE automatically detects the board and the port it is connected to without having to select the ports individually. You can use their Forum to discuss any technical issue with coding or board.

Different Types of Arduino Boards

Every board that Arduino makes has a separate set of features. As per the features and characteristics, we have classified this development board into 3 categories i.e. entry-level boards, enhanced-level boards, and IoT boards



Arduino Uno



Arduino LilyPad



Arduino Mega 2560



Arduino Leonardo



Arduino Mega ADK



Arduino Ethernet



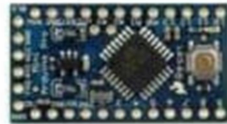
Arduino Pro



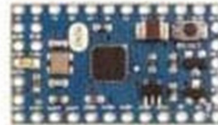
Arduino Nano



Arduino BT



Arduino Pro Mini



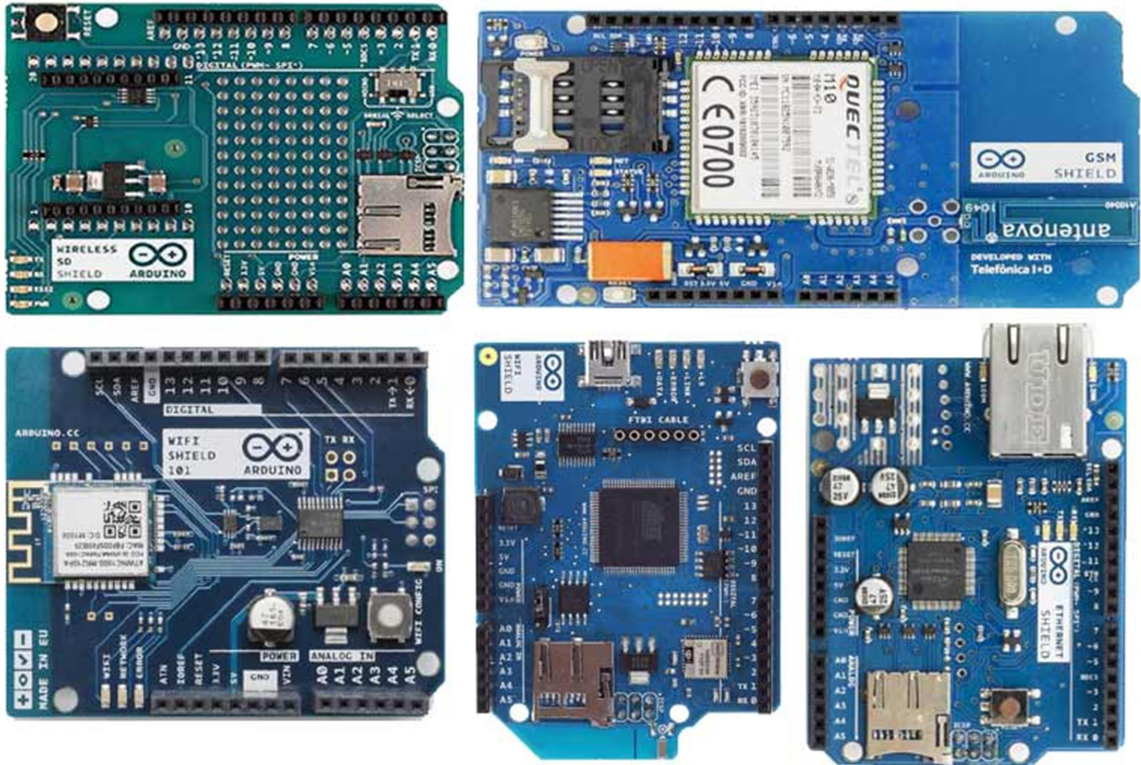
Arduino Mini



Arduino Fio

Commonly Available Arduino Shields

A shield is basically a pre-built circuit board that fits on top of the board and provides additional capabilities like controlling motors, connecting to the internet, providing cellular or other wireless communication, controlling an LCD screen, and much more.



- Ethernet Shield
- GSM Shield 2
- WiFi Shield 101
- WiFi Shield
- Wireless SD Shield

Structure of Arduino program

The basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
void setup()
```

```
{
```

```
    statements;
```

```
}
```

```
void loop()
```

```
{
```

```
    statements;
```

```
}
```

Where `setup()` is the preparation, `loop()` is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

```
setup()
```

The `setup()` function is called once when your program starts. Use it to initialize pin modes, or begin serial. It must be included in a program even if there are no statements to run.

void setup()

{

pinMode(pin, OUTPUT); // sets the 'pin' as output

}

loop()

After calling the `setup()` function, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

void loop()

{

digitalWrite(pin, HIGH); // turns 'pin' on

delay(1000); // pauses for one second

digitalWrite(pin, LOW); // turns 'pin' off

delay(1000); // pauses for one second

}

Functions

A function is a block of code that has a name and a block of statements that are executed when the function is called. The

functions void setup() and void loop() have already been discussed and other built-in functions will be discussed later.

Custom functions can be written to perform repetitive tasks and reduce clutter in a program. Functions are declared by first declaring the function type. This is the type of value to be returned by the function such as 'int' for an integer type function. If no value is to be returned the function type would be void. After type, declare the name given to the function and in parenthesis any parameters being passed to the function.

type functionName(parameters)

```
{  
  
    statements;  
  
}
```

The following integer type function *delayVal()* is used to set a delay value in a program by reading the value of a potentiometer. It first declares a local variable v, sets v to the value of the potentiometer which gives a number between 0-1023, then divides that value by 4 for a final value between 0-255, and finally returns that value back to the main program.

int delayVal()

```
{  
  
    int v;           // create temporary variable 'v'  
  
    v = analogRead(pot); // read potentiometer value  
  
    v /= 4;         // converts 0-1023 to 0-255  
  
    return v;       // return final value  
  
}
```


}

} curly braces

Curly braces (also referred to as just "braces" or "curly brackets") define the beginning and end of function blocks and statement blocks such as the void loop() function and the for and if statements.

type function()

{

statements;

}

An opening curly brace { must always be followed by a closing curly brace }. This is often referred to as the braces being balanced. Unbalanced braces can often lead to cryptic, impenetrable compiler errors that can sometimes be hard to track down in a large program.

The Arduino environment includes a convenient feature to check the balance of curly braces. Just select a brace, or even click the insertion point immediately following a brace, and its logical companion will be highlighted.

; Semicolon

A semicolon must be used to end a statement and separate elements of the program.

A semicolon is also used to separate elements in a for loop.

```
int x = 13; // declares variable 'x' as the integer 13
```

Note: Forgetting to end a line in a semicolon will result in a compiler error. The error text may be obvious, and refer to a missing semicolon, or it may not. If an impenetrable or seemingly illogical compiler error comes up, one of the first things to check is a missing semicolon, near the line where the compiler complained.

/* ... */ block comments

Block comments, or multi-line comments, are areas of text ignored by the program and are used for large text descriptions of code or comments that help others understand parts of the program. They begin with `/*` and end with `*/` and can span multiple lines.

/ this is an enclosed block comment*

don't forget the closing comment -

they have to be balanced!

**/*

Because comments are ignored by the program and take no memory space they should be used generously and can also be used to “comment out” blocks of code for debugging purposes.

Note: While it is possible to enclose single line comments within a block comment, enclosing a second block comment is not allowed.

// line comments

Single line comments begin with // and end with the next line of code. Like block comments, they are ignored by the program and take no memory space.

// this is a single line comment

Single line comments are often used after a valid statement to provide more information about what the statement accomplishes or to provide a future reminder.