ENG20009
Engineering Technology Inquiry Project

Unit Convenor : Dr Rifai Chai

Email: rchai@swin.edu.au

Phone: 9214 8119

Office: EN606B

Swinburne University of Technology

Seminar 1 – Unit Introduction

Topics:
1. Unit outline
2. Portfolio - Practical
3. Portfolio – Project
4. Hardware - Board

1. Unit outline

- The aims of this unit and the unit learning outcomes are

  - ✓ This unit of study is a project-based unit in which students work in teams and aims to provide you with the skills to enquire and solve challenges oriented around engineering technologies. You will also be provided with the skills to select and utilise appropriate engineering technology tools to address these challenges.
  - ✓ An Academic 'facilitator' will guide your learning towards achieving these outcomes.

- Learning and teaching hours
  - ✓ Seminar (12 hours):

    One 1-hour on-campus at BA201 per week
  - ✓ Workshop/Laboratory (24 hours):

    One 2-hour on-campus Workshop/Laboratory at AD102
  - ✓ Facilitator meeting

    Group meeting with your facilitator

- Teaching schedule (subject to minor changes)

Week	Week Beginning	Teaching and Learning Activity	Student Task or Assessment
1	Feb 27	**Seminar:** ▪ Introduction to Unit ▪ Getting started – basic programming ▪ General purpose input/output (GPIO) ▪ OH&S **Project Activities:** ▪ Formation of student groups **Laboratory:** ▪ Portfolio Practical: Lab 1 - Getting started and basic programming ▪ Formation of student groups	▪ Formation of student groups. ▪ Lab 1 - Getting started and basic programming (No assessment required)
2	Mar 6	**Seminar:** ▪ Guest Lecture 1 (related to the project): Rudy Gunawan, Application Specialist, Thermo Fisher Scientific ▪ Introduction to the project **Project Activities:** ▪ Introduction to the project ▪ Problem identification and project planning **Laboratory:** ▪ Portfolio Practical: Lab 2 - GPIO	▪ Hardware kit will be lent out ▪ Additional components for the project will be lent out to groups. ▪ Assessment: Portfolio Practical Demonstration – Lab 2.GPIO
3	Mar 13	**Seminar:** ▪ Interfacing ▪ Literature review **Project Activities:** ▪ Literature review **Laboratory:** ▪ Portfolio Practical: Lab 3 – Interfacing	▪ Assessment: Portfolio Practical Demonstration – Lab 2.GPIO ▪ Assessment: Portfolio Practical Demonstration – Lab 3. ADC
4	Mar 20	**Seminar:** ▪ Analog-to-Digital converter (ADC) ▪ Inclusive Behaviours & Strategies **Project Activities:** ▪ Inclusive Behaviours & Strategies ▪ Design and development **Laboratory:** ▪ Portfolio Practical: Lab 4 – Analog-to-Digital converter (ADC)	▪ Assessment: Portfolio Practical Demonstration – Lab 3. ADC ▪ Assessment: Portfolio Practical Demonstration – Lab 4. Interfacing

- Teaching schedule (subject to minor changes) cont..

4	Mar 20	**Seminar:** ▪ Analog-to-Digital converter (ADC) ▪ Inclusive Behaviours & Strategies **Project Activities:** ▪ Inclusive Behaviours & Strategies ▪ Design and development **Laboratory:** ▪ Portfolio Practical: Lab 4 – Analog-to-Digital converter (ADC)	▪ Assessment: Portfolio Practical Demonstration – Lab 3. ADC ▪ Assessment: Portfolio Practical Demonstration – Lab 4. Interfacing
5	Apr 3	**Seminar:** ▪ Memory – RAM, ROM & Data Storage **Project Activities:** ▪ Design and development **Laboratory:** ▪ Portfolio Practical: Lab 5 – Memory	▪ Assessment: Portfolio Practical Demonstration – Lab 4. Interfacing. ▪ Assessment: Portfolio Practical Demonstration – Lab 5. Memory ▪ Assessment: Portfolio Project Brief
6	Mar 25	**Seminar:** ▪ Interrupts **Project Activities:** ▪ Risk management **Laboratory:** ▪ Portfolio Practical: Lab 6 – Interrupts	▪ Assessment: Portfolio Practical Demonstration – Lab 5. Memory ▪ Assessment: Portfolio Practical Demonstration– Lab 6. Interrupts
(Mar 6 – Mar 12) Mid-Semester Break (No classes)			
7	Apr 17	**Seminar:** ▪ Debugging **Project Activities:** ▪ Design and development **Laboratory:** ▪ Portfolio Practical 6 – Advanced Interrupts	▪ Assessment: Portfolio Practical Demonstration– Lab 6. Interrupts. ▪ Submitting Portfolio Practical Demonstration codes. ▪ Assessment: Portfolio Practical Report
8	Apr 24	**Seminar:** ▪ Risk management Krzysztof Stachowicz - Laboratory Manager, Academic Operations Unit (AOU), Swinburne University of Technology **Project Activities and Workshop:** ▪ Design and development	▪ Project design and development

- Teaching schedule (subject to minor changes) cont..

9	May 1	**Seminar:** ▪ Programming style **Project Activities and Workshop:** ▪ Design and development	▪ Project design and development
10	May 8	**Seminar:** ▪ Deploying standalone embedded system **Project Activities and Workshop:** ▪ Prototyping and testing	▪ Prototyping and testing
11	May 15	**Seminar:** ▪ Guest Lecture 3 (related to Arduino): Massimo Sacchi , Corporate Partnerships Manager & Business Developer, Arduino PRO **Project Activities and Workshop:** ▪ Prototyping and testing	▪ Prototyping and testing
12	May 22	**Seminar:** ▪ Detailing project demonstration **Project Activities and Workshop:** ▪ Project demonstration	▪ Assessment: Portfolio Project demonstration ▪ Returning Hardware kit ▪ Project codes due in Week 13 ▪ Assessment: Portfolio Project Report (Due in Week 13)

- Teaching Staff

Name	Role	Phone	Email
Dr Rifai Chai	Unit Convenor, Lecturer and Workshop facilitator	9214 8119	rchai@swin.edu.au
Dr Jason But	Workshop facilitator		jbut@swin.edu.au
Dr Muhammad Hasan	Workshop facilitator		mhasan@swin.edu.au
Dulan Perera	Workshop facilitator		dgperera@swin.edu.au
Khanh Ha Nguyen	Workshop facilitator		khanhhanguyen@swin.edu.au
Matthew Arrowsmith	Workshop facilitator		marrowsmith@swin.edu.au

✓Seminars (SE1) and Workshops (WK1) according to the schedule below:

Activity Name	Activity Type Name	Day	Start	End	Activity Duration	Teacher/facilitator	Location	Teaching Week
ENG20009_1_HS1_HAW_1/SE1/01	Seminars	Mon	11:30am	12:30pm	1:00	Rifai Chai	BA201	1-12
ENG20009_1_HS1_HAW_1/WK1/01	Workshops	Tue	8:30am	10:30am	2:00	Dulan Perera	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/02	Workshops	Tue	12:30pm	2:30pm	2:00	Rifai Chai	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/03	Workshops	Tue	4:30pm	6:30pm	2:00	Muhammad Hasan	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/04	Workshops	Wed	8:30am	10:30am	2:00	Matthew Arrowsmith	AD102	1-7
ENG20009_1_HS1_HAW_1/WK1/04	Workshops	Wed	8:30am	10:30am	2:00	Jason But	AD102	8-12
ENG20009_1_HS1_HAW_1/WK1/05	Workshops	Wed	12:30pm	2:30pm	2:00	Matthew Arrowsmith	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/06	Workshops	Wed	4:30pm	6:30pm	2:00	Muhammad Hasan	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/07	Workshops	Thu	8:30am	10:30am	2:00	Khanh Ha Nguyen	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/09	Workshops	Thu	12:30pm	2:30pm	2:00	Khanh Ha Nguyen	AD102	1-12
ENG20009_1_HS1_HAW_1/WK1/08	Workshops	Thu	4:30pm	6:30pm	2:00	Muhammad Hasan	AD102	1-12

- Assessment task details:

Tasks and Details	Individual or Group	Weighting	Unit Learning Outcomes that this assessment task relates to	Assessment Due Date
Portfolio - Practical (i) Portfolio Practical Demonstration (30%) (ii) Portfolio Practical Report (10%)	Individual	40%	1,4,5	(i) From Week 2 to Week 7 during laboratory session. Each lab has 2 weeks for the demonstration. Demonstrated codes due in Week 7 (ii) Apr 21, 2023 by 23:59pm (End of Week 7); submit in Canvas
Portfolio - Project i) Portfolio Project Brief (10%) ii) Portfolio Project Demonstration (35%) iii) Portfolio Project report (15%)	Group/ Individual Individual Group Individual	60%	2, 3, 4, 5	i) Mar 31, 2023 by 23:59pm (End of Week 5); submit in Canvas ii) Your on-campus workshop class in week 12 & project codes due in week 13 iii) Jun 2, 2023 by 23:59pm (End of Week 13)

2. Portfolio - Practical

(i) Practical Demonstration

- ✓ Provide practical demonstration for the completed task from Lab 2 to Lab 6 in the lab to the Lab Supervisor/Tutor.
- ✓ Each lab has 2 weeks for the demonstration. For example Lab 2 tasks demonstration is from Week 2 to Week 3 for all Lab's 2 tasks (P, PP, C, D and HD). The Lab 6's demonstration will be from Week 6 to Week 7.
- ✓ The demonstrated codes must be submitted in Week 7 as a single compressed ZIP file, which must be named as 'your ID-PC'. For example: 12345678-PC.zip.

(i) Practical Report

- ✓ The portfolio practical report covert the flowchart or pseudo code for each completed tasks.
- ✓ The portfolio can be submitted as a single PDF or Word file, which must be named as 'your student ID-PF'. For example 12345678-PF.pdf. Additional file for the codes can be submitted compressed in ZIP file.

2. Portfolio – Practical – Example

Portfolio Practical – Lab 3

Topic: Interfacing with UART, SPI and I^2C

Resources: Week 1, 2 and 3 seminar and portfolio practical/lab notes.

Demonstration: With tutor in the lab in week 3 or week 4

Report: Week 7, in Canvas

Tool: Using Swinburne Lab board with Arduino Zero

Pass: Create menu using the UART and display in the serial monitor which has selection at the following:

- The first menu should toggle the first buzzer on and off.

Pass Plus: Continue from the Pass question above with more selection below:

- The second menu should increase the speed of the second LED's blinking.
- The third menu should decrease the speed of the second LED's blinking.
- The fourth menu should toggle the brightness of the third LED between high and low.

Credit: Using the RTC and LCD. Create a digital clock by reading the data from RTC and displaying it on LCD.

Distinction: Using the accelerometer of the IMU create a spirit level that displays the current angle away from level on the LCD display and provide a graphic that will assist with levelling the board.

High Distinction:

Using the IMU, move a symbol (**not** a circle or dot) around the LCD screen and output the coordinates to serial terminal. Use the accelerometer values for direction and for speed, the symbol must start in the middle of the board when the program starts no matter the angle of the IMU and wait till the IMU is moved before moving on screen.

3. Project

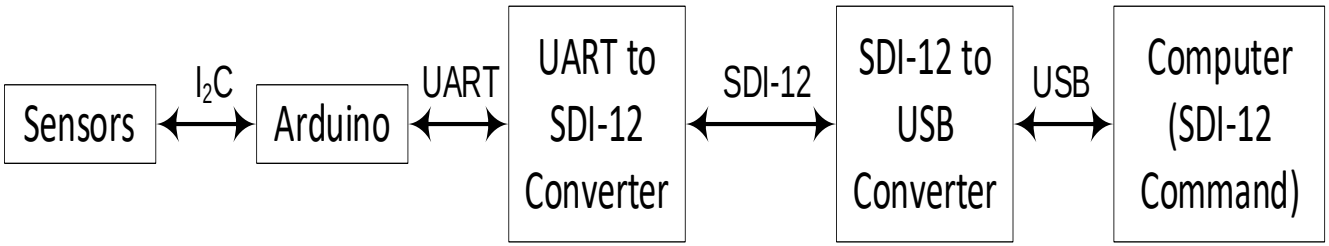
Industry:

ThermoFisher SCIENTIFIC

Thermo Fisher Scientific Australia Pty Ltd

https://www.thermofisher.com/au/en/home/

- For the portfolio project, you will work in a group of 5 students. Your team will be formed on the first week in your laboratory class.
- In the project task, students will work in a group consisting of 5 students to develop an SDI-12 sensor and data logger.
- Each group can choose at least two sensors:
  - BME680 - Temperature, Humidity, Pressure and Gas Sensor
  - BH1750FVI - Digital Light intensity Sensor
  - U-blox NEO-6M GPS Module



3. Project

Pass: Design Requirement – SDI12 sensor

- Program the Arduino to read the relevant data from the sensor
- Use address =0 for the sensor 1 and address =1 for sensor 2 by default.
- Program the Arduino based on the SDI-12 communication protocol so it can understand the SDI-12 commands, at least 5 commands below

Name	Command	Response
Address Query	?!	a<CR><LF>
Change Address	aAb!	b<CR><LF>
Start Measurement		atttn<CR><LF>
Send Data	aD0! . . . aD9!	a<values><CR><LF> or a<values><CRC><CR><LF> a<values><CR><LF> or a<values><CRC><CR><LF> a<values><CR><LF> or a<values><CRC><CR><LF> a<values><CR><LF> or a<values><CRC><CR><LF> a<values><CR><LF> or a<values><CRC><CR><LF>
Continuous Measurements	aR0! ... aR9!	a<values><CR><LF>

A complete SDI-12 protocol/commands can be found at the following link:

https://www.sdi-12.org/current_specification/SDI-12_version-1_4-Jan-30-2021.pdf

3. Project

Credit: Design Requirement – Standalone data logger

- Save the selected sensor data and RTC into the SD CARD
- Create Menu in the system with LCD to select different sensors;
- Display the data LCD with the relevant graphical representation.

Distinction and High Distinction: Design Requirement – Interrupt and Advanced Interrupt

- Applying interrupt service routine to program the SDI-12 sensor above.
- Applying event-driven programming using interrupt and advanced interrupt-driven programming.

3. Project

(i) Project Brief

- Project planning and task division of each member.
- Each student in a group will need to select at one of the applications of the SDI-12 sensor based on their selected major and provide the literature review of using the sensor within the selected application.
- The report uses the IEEE, two column format with the maximum of 4 pages. Template can be obtained using the following link: https://www.ieee.org/conferences/publishing/templates.html
- The report must be submitted as a single PDF, which must be named as 'your group ID-PB'. For example: 12345678-PB.pdf.

(ii) Project Demonstration

- To demonstrate project according to the design requirements.
- One student in each group must submit a copy of project code demonstrated in class.
- The code must be submitted as a single compressed ZIP file, which must be named as 'your group ID-PC'. For example: 12345678-PC.zip.

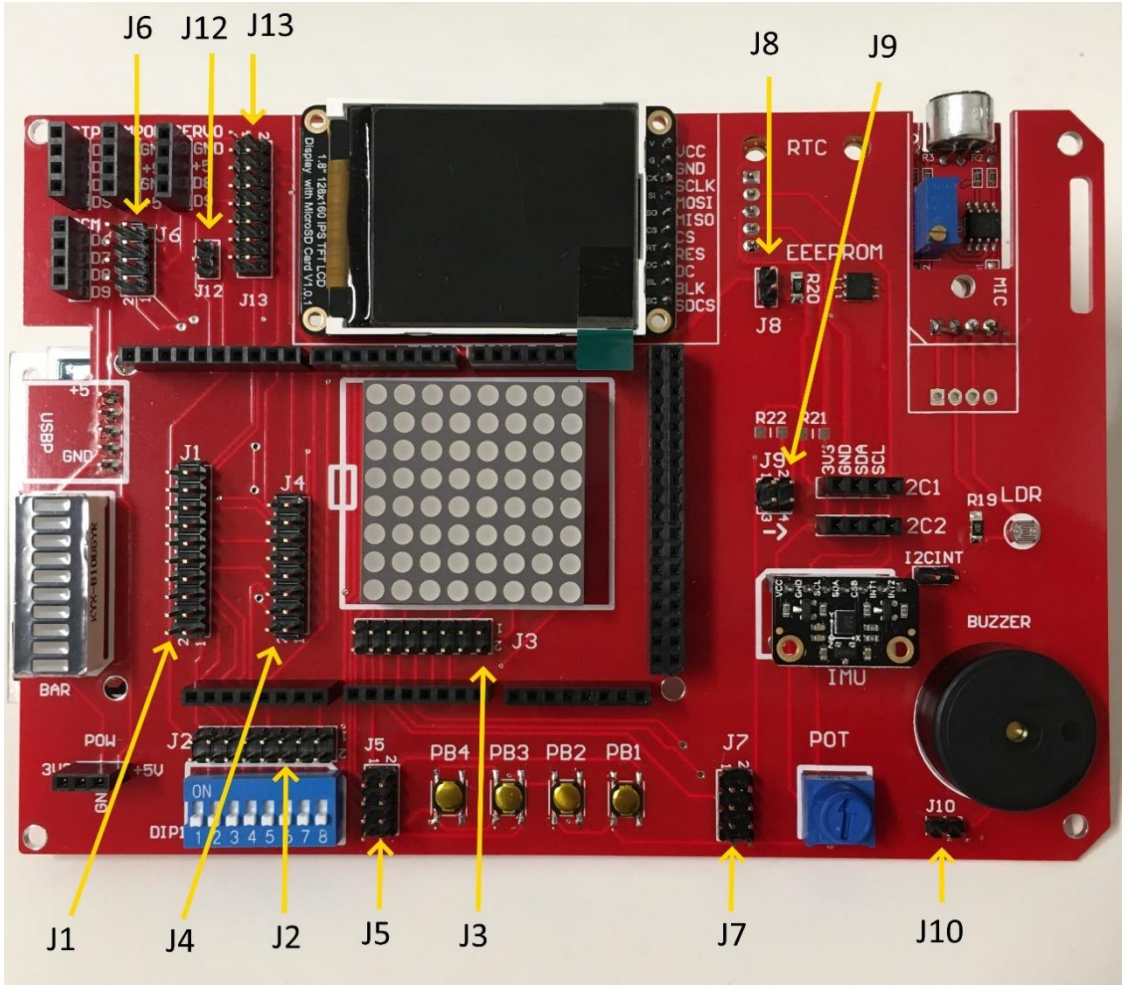
(iii) Project Report

- Flowchart or pseudo code of the project tasks based on the task division.
- Risk Assessment for each application of the sensor.
- Reflection on the relevant knowledge learned in seminar, facilitation, lab and workshop for project work.
- Reflection on the teamwork.
- The report must be submitted as a single PDF, which must be named 'your group ID-PR'. For example: 12345678-PR.pdf.

3. Project

- At the end of the semester, teaching team will select best completed project. A certificate of best project award will be given to the group and will be signed by Chair, Department of Engineering Technologies and relevant Industry.

4. Hardware - Board



Part	Jumper
LED BARGRAPH	J1
DIP SWITCHES	J2
LED MATRIX Column	J3
LED MATRIX Row	J4
PUSH BUTTONS	J5
DC MOTOR	J6
POT	J7
LDR	J7
MIC	J7
EEPROM	J8
IMU	J9
RTC	J9
BUZZER	J11
SD Card	J12
LCD	J13

Hardware - Board

- Arduino Zero for Portfolio - Practical

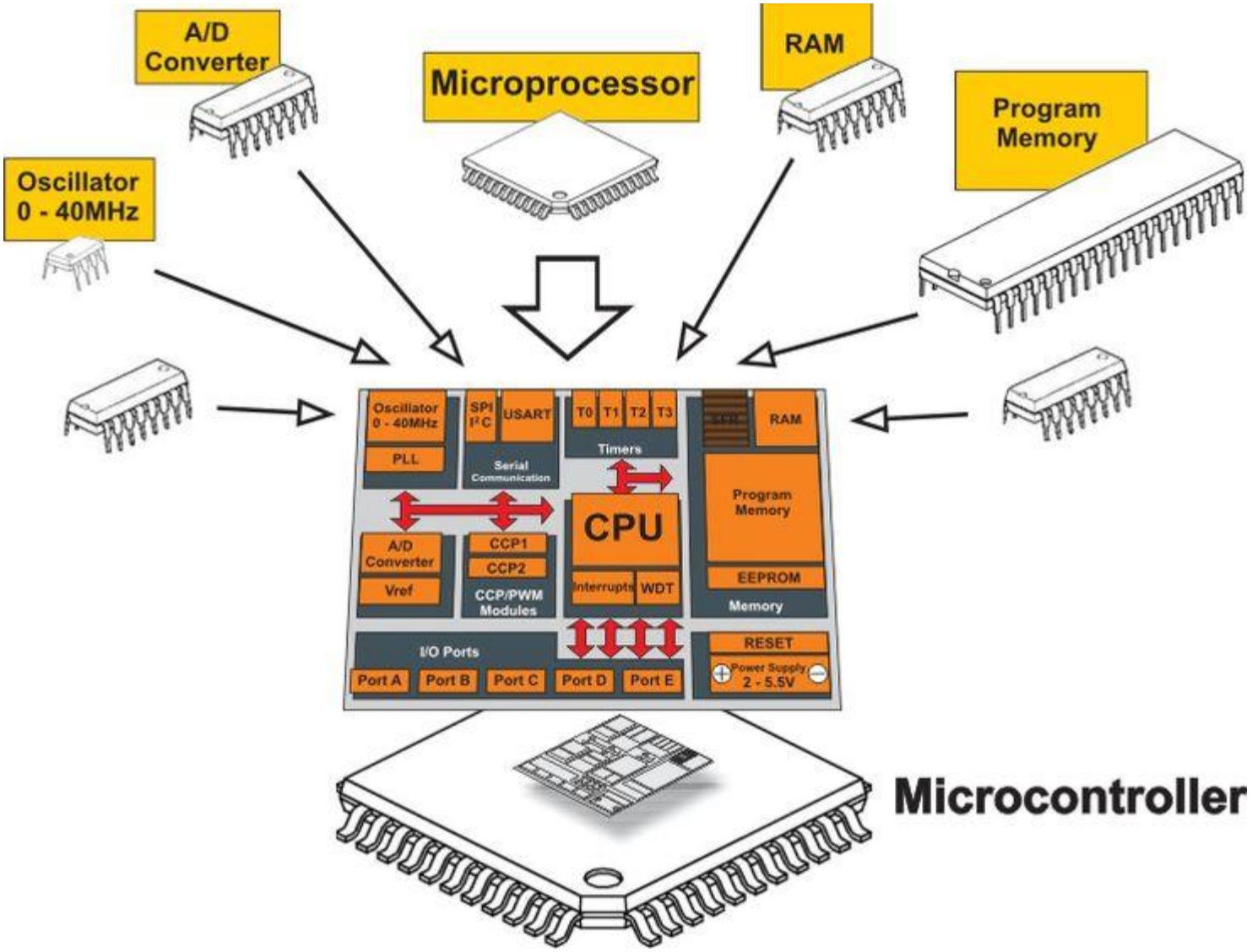
- Arduino Due for Portfolio - Project

Seminar 1
Arduino Introduction – programming basic

Topics:
1. Microprocessor vs microcontroller
2. Arduino

1. Microprocessor vs Microcontroller

- Small computer integrated into a single chip
- It contains Processing core, Flash Memory for program, I/O peripherals, RAM, Peripherals such as clocks,timers,PWM etc
- Microcontrollers are an example of embedded systems. Typically "embedded" inside some device that they control
- Microprocessors are used for general purpose applications, while microcontrollers are self sufficient and are used for specific tasks.
- A microcontroller is often small and low cost.


A/D Converter
Microprocessor
RAM
Program Memory
Oscillator 0 - 40MHz
Oscillator 0 - 40MHz
PLL
SPI I²C
USART
Serial Communication
T0
T1
T2
T3
Timers
CPU
RAM
Program Memory
EEPROM Memory
A/D Converter
Vref
CCP1
CCP2
CCP/PWM Modules
Interrupts
WDT
I/O Ports
Port A
Port B
Port C
Port D
Port E
RESET
Power Supply 2 - 5.5V
Microcontroller

2. What is Arduino ?

- Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.
- It is a small microcontroller board with a USB plug.
- Based on a simple i/o board and a development environment that implements the Processing/writing language.
- Arduino can be used to develop stand-alone interactive objects or can be connected to software on your computer.
- It is programmed in Arduino Programming language(APL) similar to C/C++.
- Way more easy to program compared to other microcontroller packages.
- The Arduino is a microcontroller development platform.

Arduino Boards

ARDUINO UNO

8 bit | AVR | < 20 mA | Usb

5V | Standard (~20) | No battery

ARDUINO NANO

8 bit | AVR | < 20 mA

5V | Standard (~20) | No battery

ARDUINO ZERO

32 bit | ARM | Usb

Standard (~20)

ARDUINO MICRO

8 bit | AVR | < 20 mA | Microusb

5V | Standard (~20) | No battery

ARDUINO LEONARDO

8 bit | AVR | 50 mA | Microusb

5V | Standard (~20) | No battery

ARDUINO MEGA 2560

8 bit | AVR | 50 mA | Usb

5V | Large (~70) | No battery

ARDUINO DUE

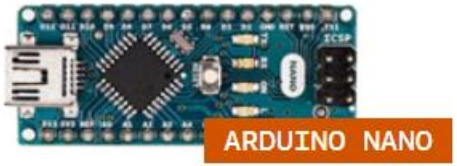
32 bit | ARM | Usb

Large (~70)

Arduino Uno

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Arduino Nano

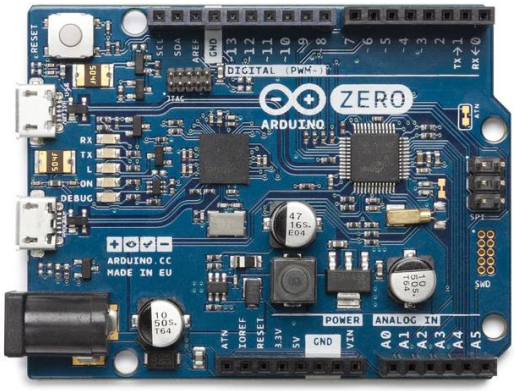


Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog IN Pins	8
EEPROM	1 KB
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12 V
Digital I/O Pins	22 (6 of which are PWM)
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm
Weight	7 g
Product Code	A000005

Arduino Mega

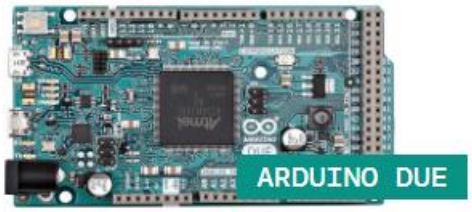
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Arduino Zero



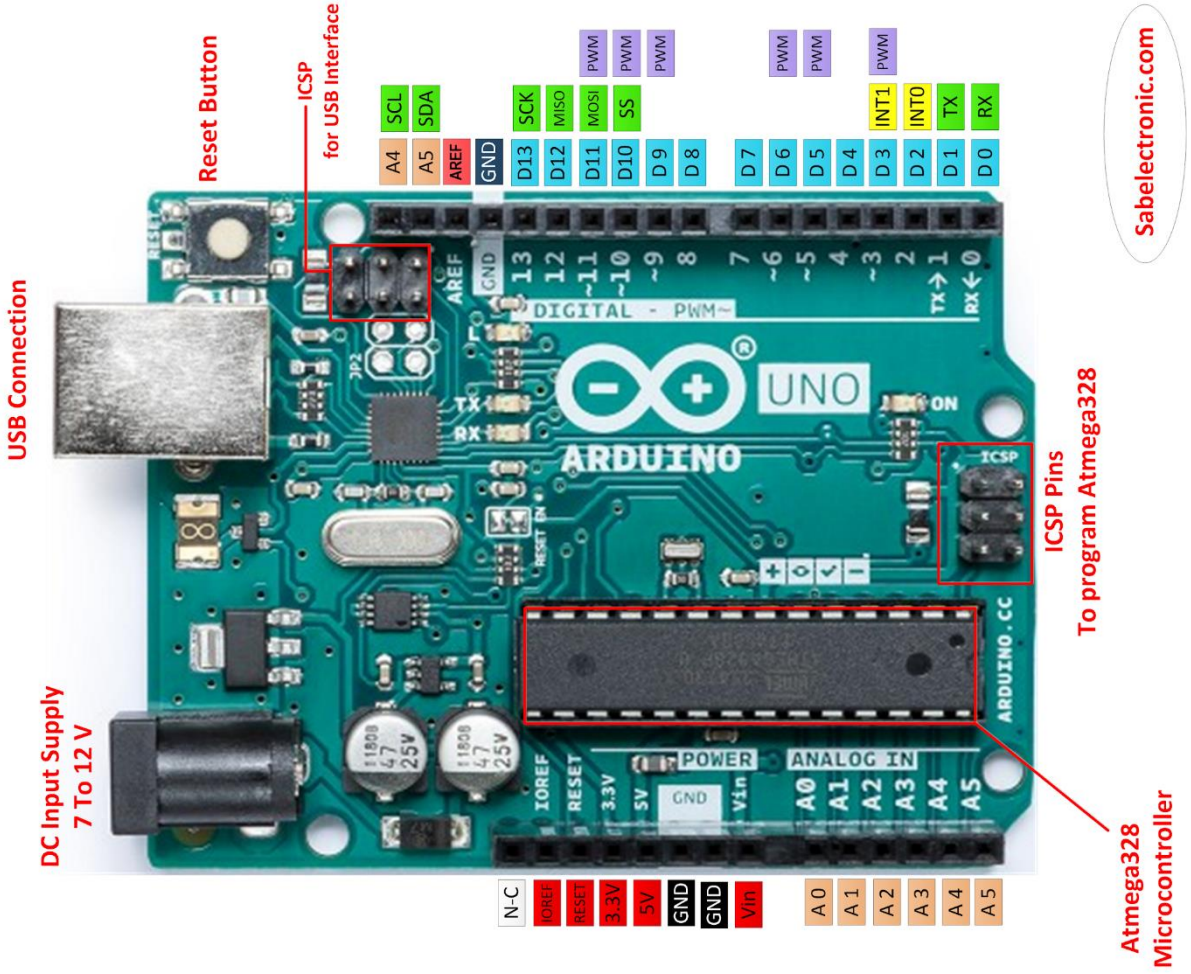
Microcontroller	ATSAMD21G18, 32-Bit ARM® Cortex® M0+
Operating Voltage	3.3V
Digital I/O Pins	20
PWM Pins	3, 4, 5, 6, 8, 9, 10, 11, 12, 13
UART	2 (Native and Programming)
Analog Input Pins	6, 12-bit ADC channels
Analog Output Pins	1, 10-bit DAC
External Interrupts	All pins except pin 4
DC Current per I/O Pin	7 mA
Flash Memory	256 KB
SRAM	32 KB
EEPROM	None. See documentation
LED_BUILTIN	13
Clock Speed	48 MHz
Length	68 mm
Width	53 mm
Weight	12 gr.

Arduino Due

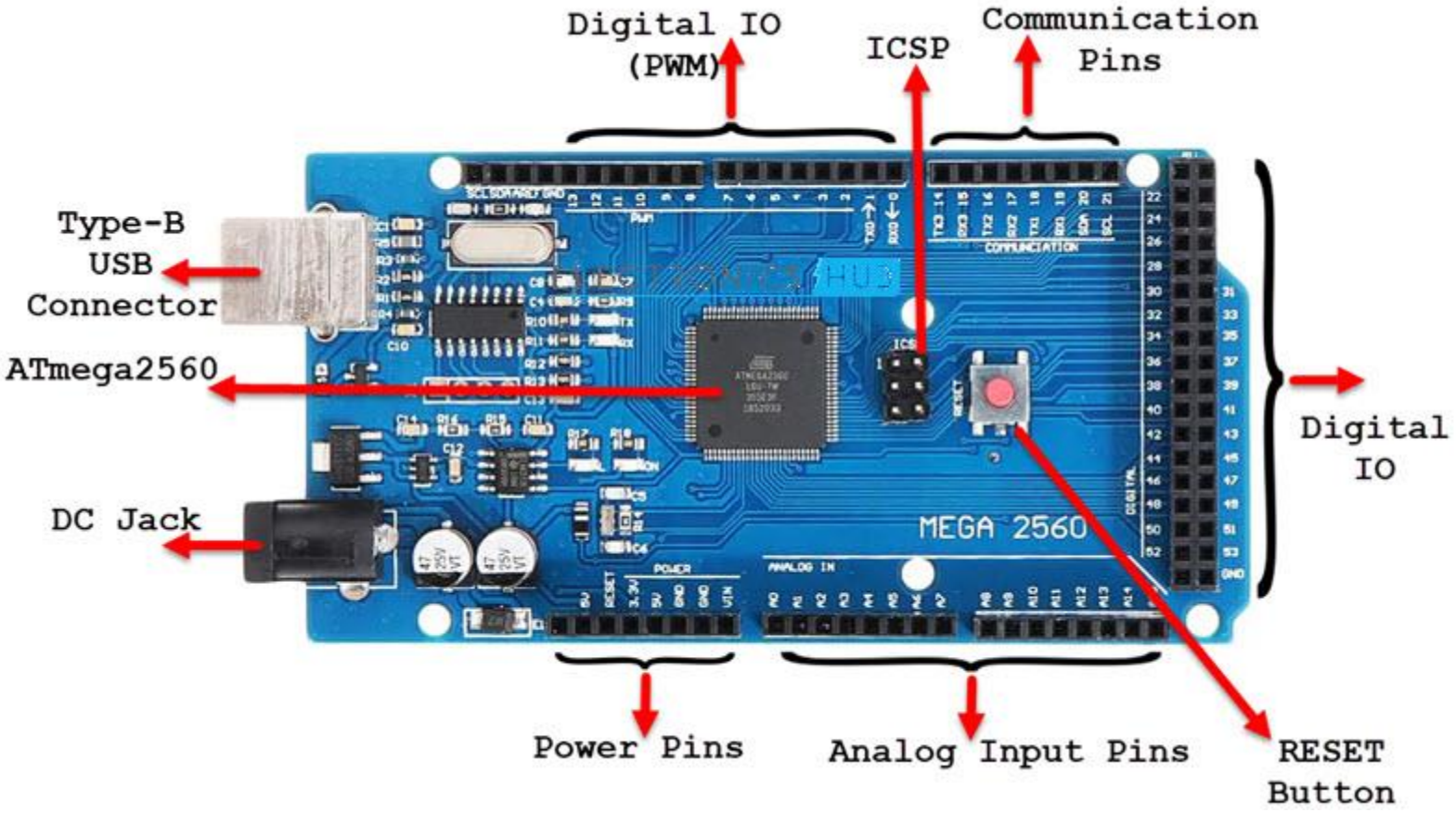


Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

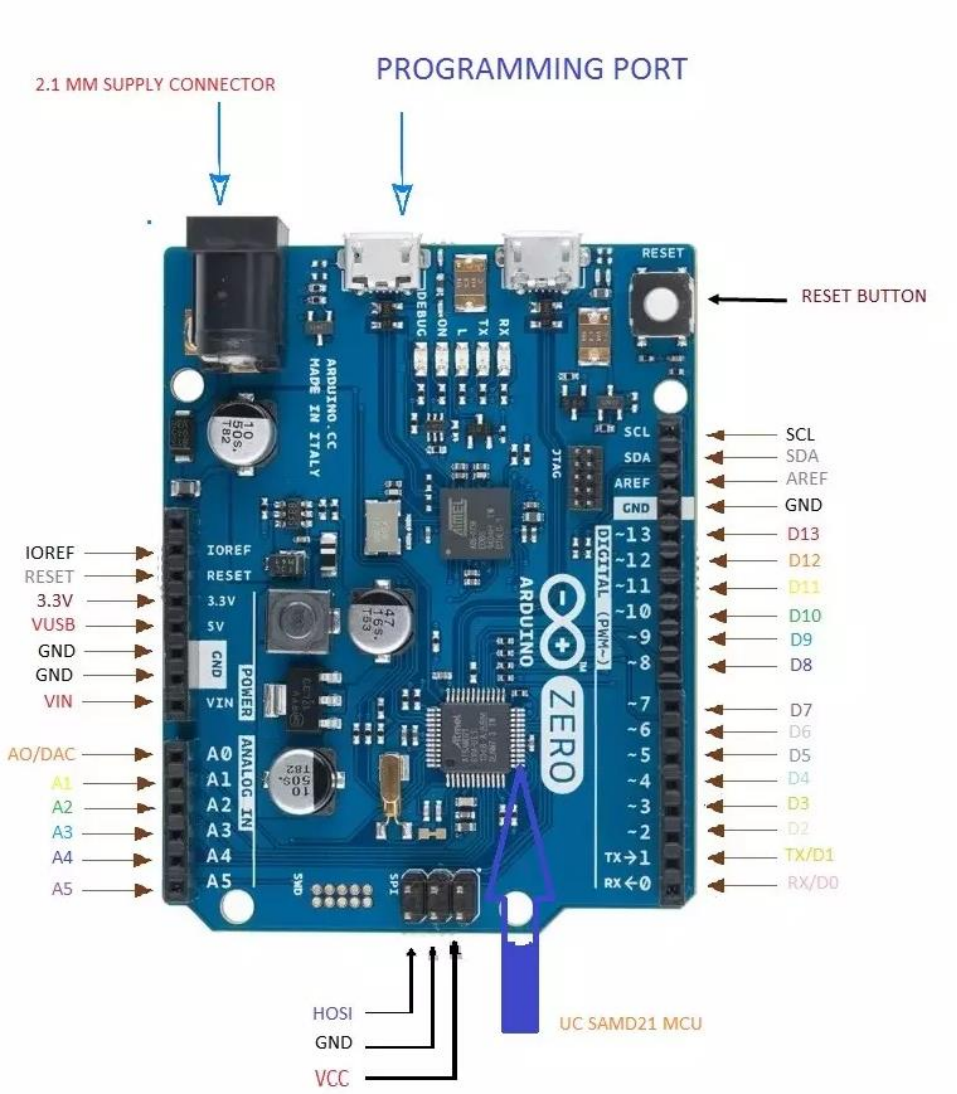
Pinout - Arduino Uno



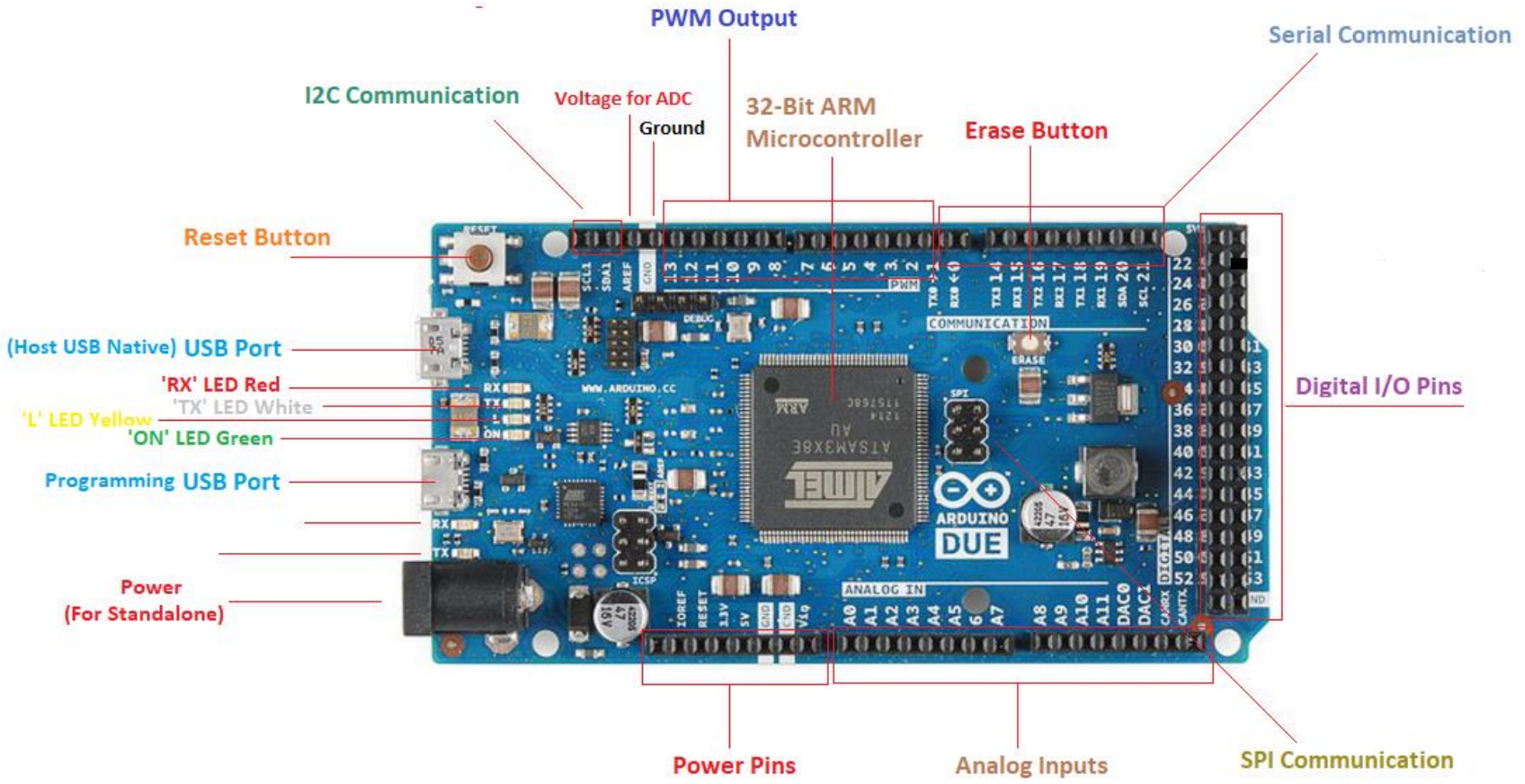
Arduino Mega



Arduino Zero



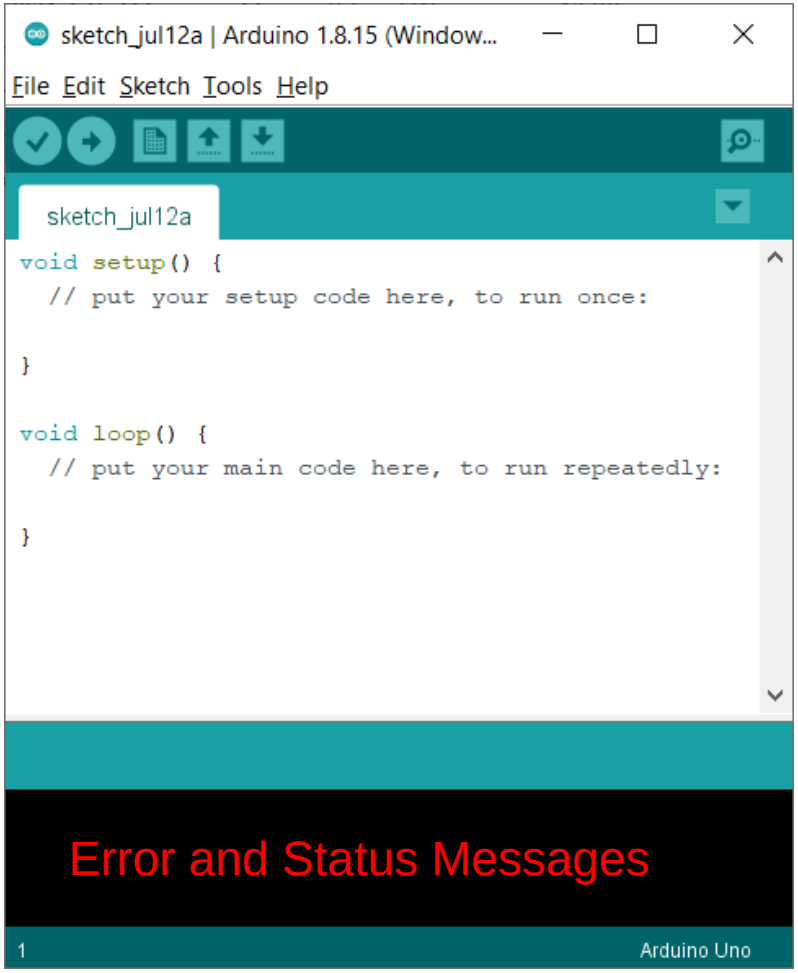
Arduino Due


PWM Output
Serial Communication
I2C Communication
Voltage for ADC
Ground
32-Bit ARM Microcontroller
Erase Button
Reset Button
(Host USB Native) USB Port
'RX' LED Red
'TX' LED White
'L' LED Yellow
'ON' LED Green
Programming USB Port
Power (For Standalone)
Digital I/O Pins
Power Pins
Analog Inputs
SPI Communication
Arduino Due
www.TheEngineeringProjects.com

Programming & Communication

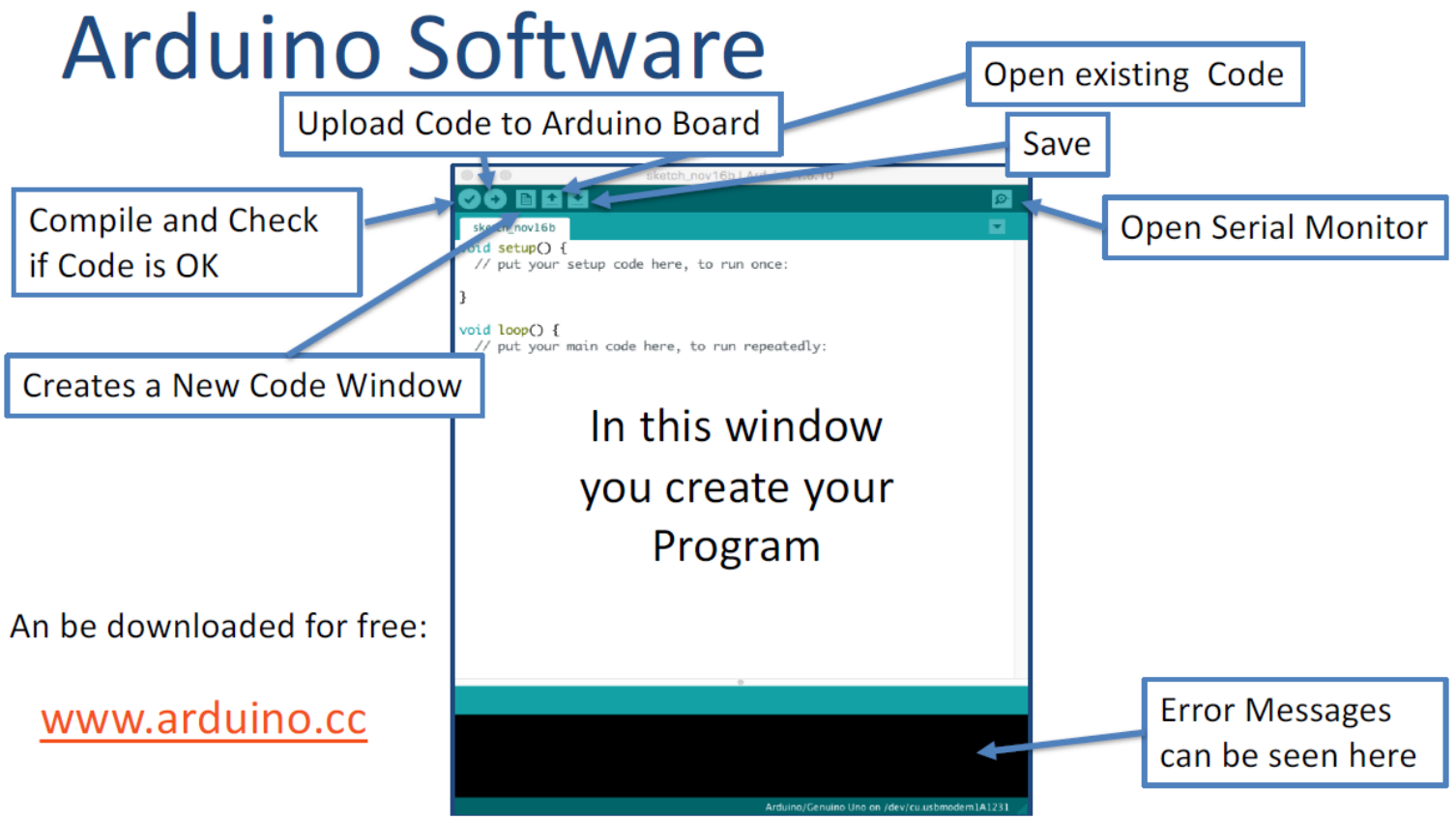
- Almost all Arduino Boards can be programmed by Arduino Software, IDE for making programs known as sketches.
- One more interesting point here is that no external burner like boot loader is required to burn our code on board. When it comes to uploading these sketches, procedure is quite different from other Arduino boards because we have to erase flash memory before re-programming our board.
- Uploading on SAM3X is managed by ROM and it runs only when all data is erased and flash memory is empty.
- We can make use of any USB port from native or programming port for sketch uploading.

Arduino IDE



- Verify - Checks your code for errors compiling it.
- Upload - Compiles your code and uploads it to the configured board.
- New - Creates a new sketch.
- Open - Presents a menu of all the sketches in your sketchbook.
- Save - Saves your sketch.
- Serial Monitor - Opens the serial monitor.

Arduino IDE


Arduino Software
Upload Code to Arduino Board
Open existing Code
Save
Compile and Check if Code is OK
Open Serial Monitor
Creates a New Code Window
In this window you create your Program
An be downloaded for free:
www.arduino.cc
Error Messages can be seen here

Arduino Sketches

- A sketch is the name that Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.
- There are two special functions that are a part of every Arduino sketch: setup() and loop().
- setup : It is called only when the Arduino is powered on or reset. It is used to initialize variables and pin modes
- loop : The loop functions runs continuously till the device is powered off. The main logic of the code goes here. Similar to while (1) for micro-controller programming.

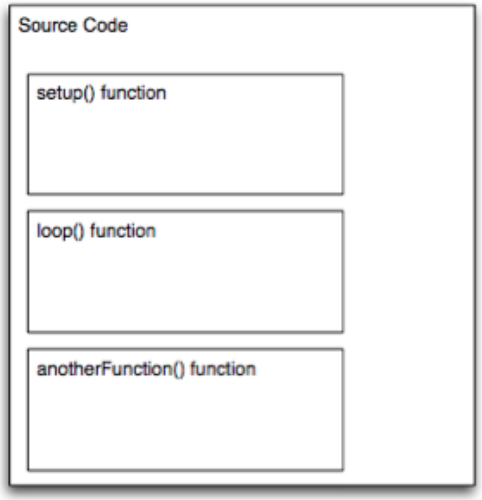
Arduino Sketches

- Where is Main Function ?

In fact, Arduino has main, but it's hiden. You can find it here: \Arduino 1.0.5\hardware\arduino\cores\arduino\main.cpp, as following:

```
#include <Arduino.h>

int main(void)
{
    init();
#if defined(USBCON)
    USBDevice.attach();
#endif
    setup();
    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
    return 0;
}
```

Source code




```
// this variable is declared at the highest level, making it visible
everywhere
int globalString;

void setup(){
  // … some code
}
void loop(){
  int a; // a is visible inside the loop function only
  anotherFunction(); // calling the global function anotherFunction

// … some other code
}

void anotherFunction() {
  // … yet another code
  int veryLocalVar; // veryLocalVar is visible only in anotherFunction
function
}
```

Data Types

- **boolean** (8 bit) - simple logical true/false
- **byte** (8 bit) - unsigned number from 0-255
- **char** (8 bit) - signed number from -128 to 127. The compiler will attempt to interpret this data type as a character in some circumstances, which may yield unexpected results
- **unsigned char** (8 bit) - same as 'byte'; if this is what you're after, you should use 'byte' instead, for reasons of clarity
- **word** (16 bit) - unsigned number from 0-65535
- **unsigned int** (16 bit)- the same as 'word'. Use 'word' instead for clarity and brevity
- **int** (16 bit) - signed number from -32768 to 32767. This is most commonly what you see used for general purpose variables in Arduino example code provided with the IDE
- **unsigned long** (32 bit) - unsigned number from 0-4,294,967,295. The most common usage of this is to store the result of the `millis()` function, which returns the number of milliseconds the current code has been running
- **long** (32 bit) - signed number from -2,147,483,648 to 2,147,483,647
- **float** (32 bit) - signed number from -3.4028235E38 to 3.4028235E38. Floating point on the Arduino is not native; the compiler has to jump through hoops to make it work. If you can avoid it, you should. We'll touch on this later.

Statement and Operators

- Statement represents a command, it ends with ;

```
int x;
x=10;
```

- Operators are symbols that used to indicate a specific function:
  - Math operators: [+,-,* ,/,%,^]
  - Logic operators: [==, !=, &&, ||]
  - Comparison operators: [==, >, <, !=, <=, >=]
- Syntax:
- ; Semicolon, {} curly braces, //single line comment, /*Multi-line comments */

Statement and Operators

- Precedence operators:

Precedencies groups	Operators	Names
2	++	Suffix increment
	--	Suffix decrement
	()	Function call
	[]	Array element access
3	++	Prefix increment
	--	Prefix decrement
5	*	Multiplication
	/	Division
	%	Modulo
6	+	Addition
	-	Subtraction
8	<	Less than
	<=	Less than or equal to
	>	Greater than
	>=	Greater than or equal to
9	==	Equal to
	!=	Not equal to
13	&&	Logical AND
14	||	Logical OR
15	?:	Ternary conditional
16	=	Assignment
	+=	Assignment by sum
	-=	Assignment by difference
	*=	Assignment by product
	/=	Assignment by quotient
	%=	Assignment by remainder

Statement and Operators

- Relational operators:

Operator	Interpretation
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

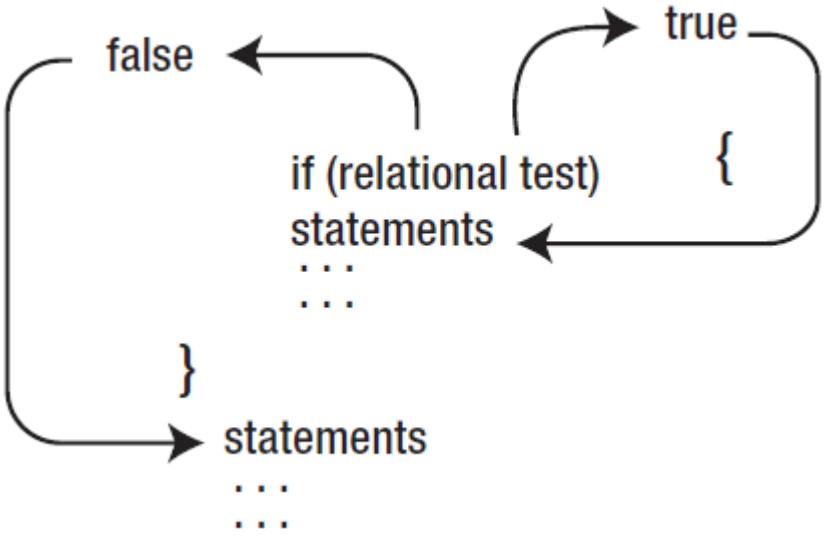
The result of all relational operations is either logic `true` (non-zero) or logic `false` (zero). For example:

```
5 > 4   // Logic true
5 < 4   // logic false
5 == 4  // logic false
5 != 4  // logic true
```

Control Statements

- If Conditioning:

```
if(condition)
{
     statements-1 ;
     …
     Statement-N;
}
else if(condition-2 )
{
     Statements ;
}
Else {statements ;}
```



Control Statements

- Examples of If Conditioning:

```
void loop() {
  if (counter % 2 == 1) {
        digitalWrite(led1, LOW);     // turn the LED off by making the voltage LOW
        digitalWrite(led2, HIGH);    // turn the LED on (HIGH is the voltage level)
  } else {
        digitalWrite(led1, HIGH);    // turn the LED on (HIGH is the voltage level)
        digitalWrite(led2, LOW);     // turn the LED off by making the voltage LOW
  }
  delay(1000);                // wait for a second
  counter = counter + 1;
}
```

Control Statements

- Examples Cascading if Statements:

```
if (myDay == 1) {
        doSundayStuff();
} else {
        if (myDay == 2) {
                doMondayStuff();
        } else {
                if (myDay == 3) {
                        doTuesdayStuff();
                } else {
                        // you get the idea...
                }
        }
}
```

Control Statements

- Switch Case:

```
switch (var) {
case 1 :
     //do something when var equals 1
     break;
case 2 :
     //do something when var equals 2
     break;
default:
// if nothing else matches , do the default
// default is optional
}
```

Control Statements

- Switch Case:

```
float midiCCMessage;
switch (midiCCMessage) {
  case 7:
  changeVolume();
  break;
  case 10:
  changePan();
  break;
  default:
  LedOff();
}
```

This code is equivalent to:

```
float midiCCMessage;
if (midiCCMessage == 7) changeVolume();
else if (midiCCMessage == 10) changePan ();
else LedOff();
```

Loop Statements

- Do… while:

```
do{
        Statements;
}while(condition)
```

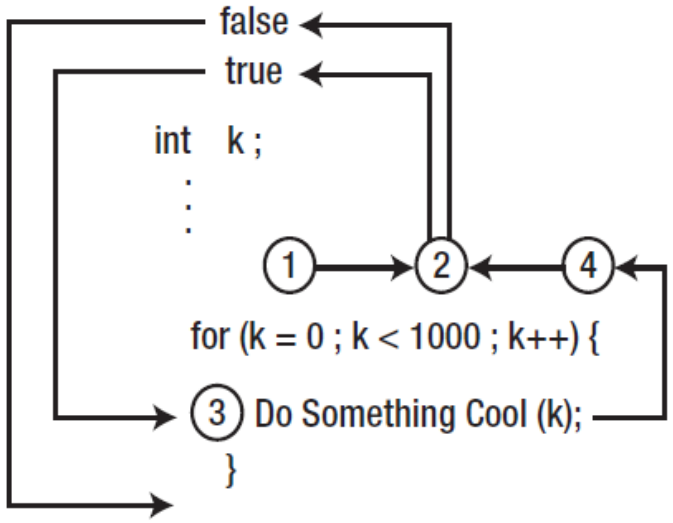
- While:

The statements are run at least once.

```
while(condition){
        statements;
}
```

- For Statement

```
for (int i=0; i <= va l; i++){
        statements;
}
```



Breaking the loop

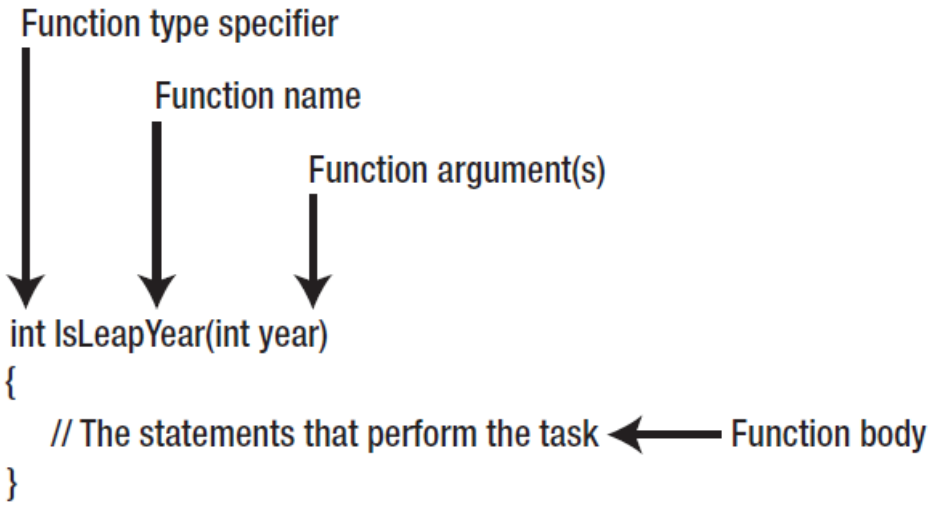
- The **break** and **continue** statements are often used within loops structures. Simply stated, a break statement sends program control to the statement that immediately follows the closing brace of the loop body.
- The continue statement immediately sends program control to the test conditions of the loop (i.e., expression2) for this pass through the loop.

Imagine a LED. We want its intensity to grow from 0 to 100 percent then to go back to 0, every time. But we also want to use a nice distance sensor that resets this loop each time the distance between a user and the sensor is greater than a value.

```
for ( intensity = 0 ; intensity < 100 ; intensity++ ){
  ledIntensity (intensity);
  if (distance > maxDistance) { // if the user is far
    intensity = 0;    // switch off the LED
    break;      // exits the loop
  }
}
```

Functions

- A function is a body of code designed to solve a particular task.
- A function library is simply a collection of functions that share a common area of interest (e.g., the Math and Time functions in Arduino C).



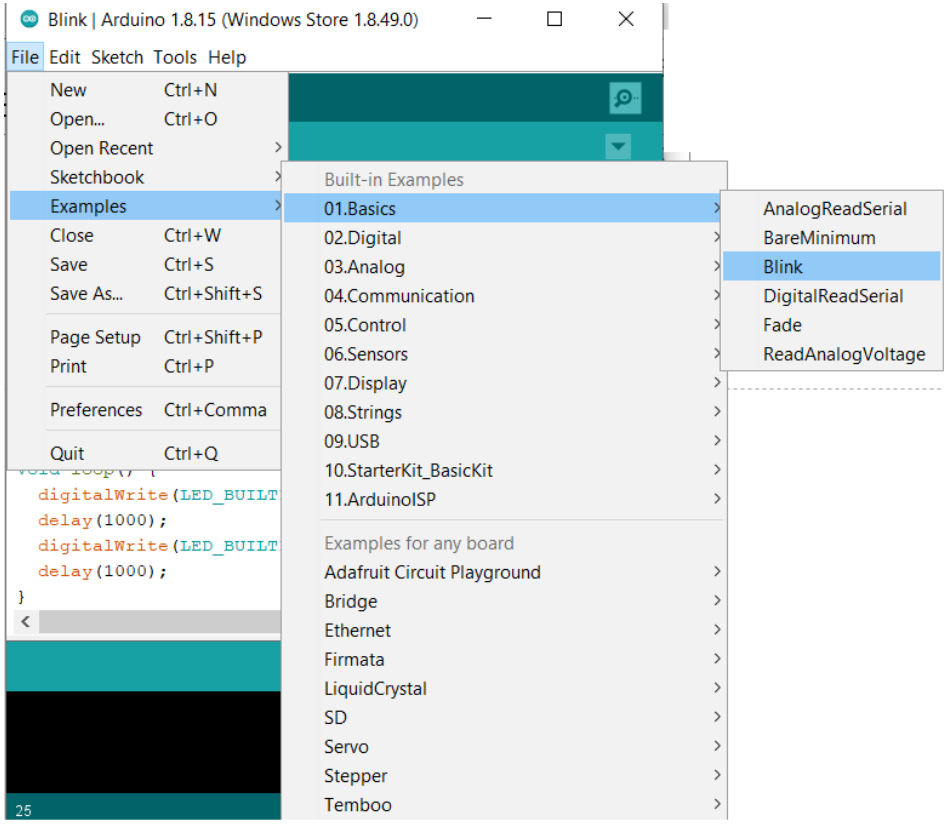

```
int VolumeOfCube(int width, int length, int height)
{
      int volume;
      volume = width * length * height;
      return volume;
}
```

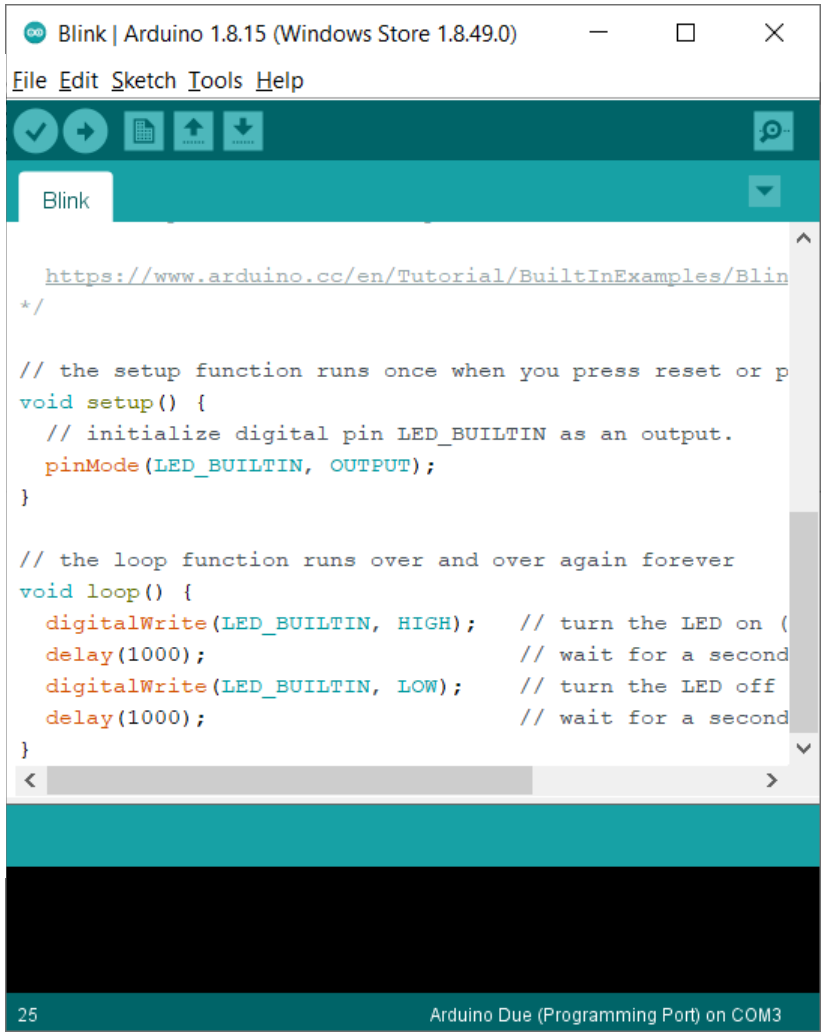
Seminar 1 - GPIO

Topics:
1. Hardware testing
2. GPIO – Outputs
3. GPIO – PWM
4. LED Bar
5. LED Matrix
6. Button
7. Button with no resistor
8. Button with Debouncing

1. Hardware – Testing

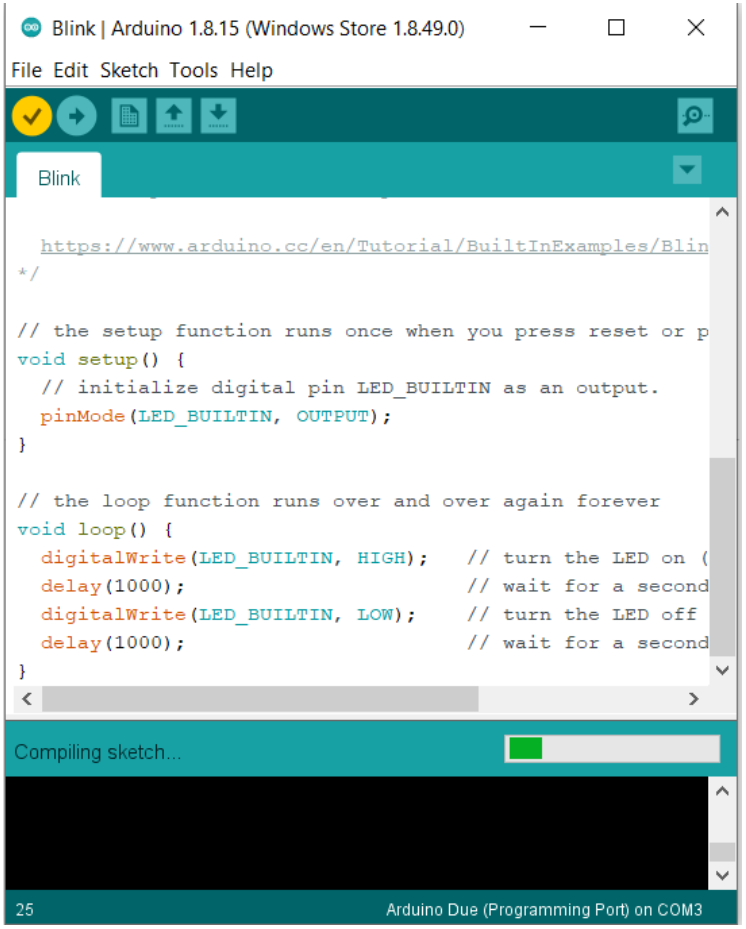
- The initial testing can be done by test to program the internal LED
- Example 'Blink' program can be used.

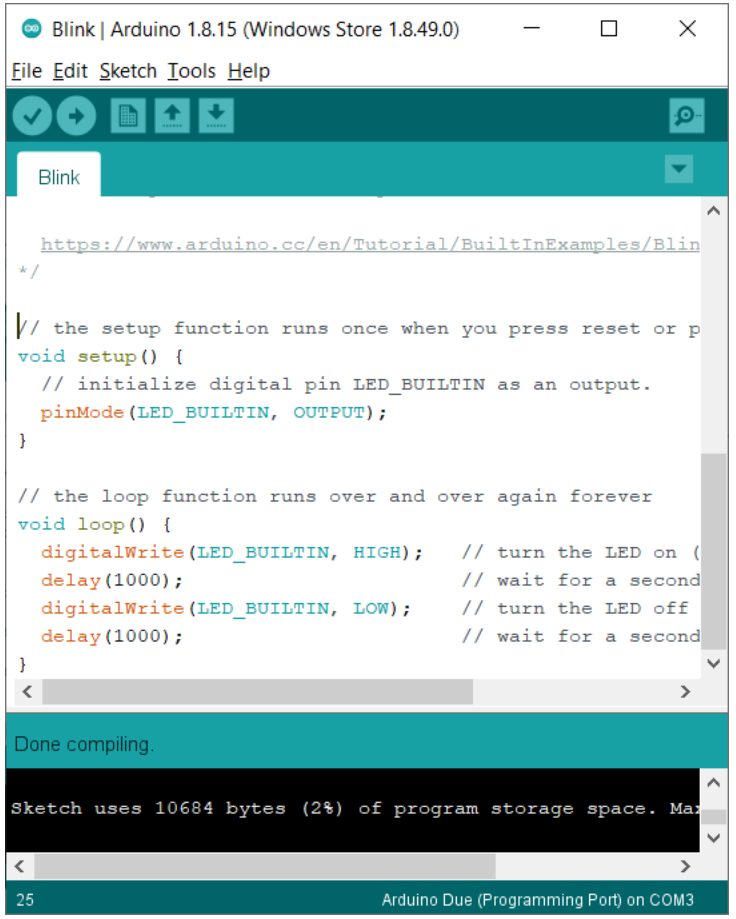




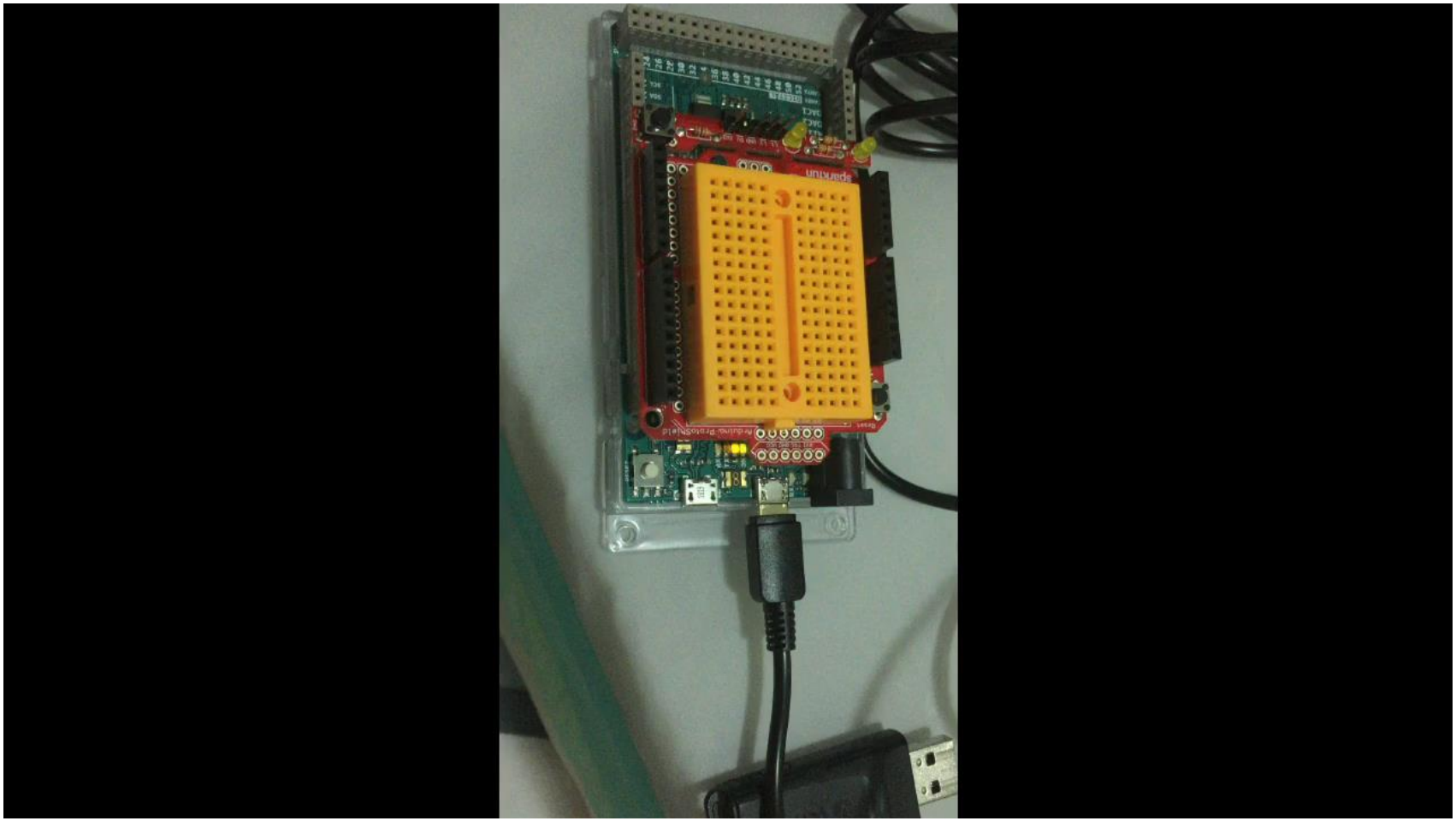
Hardware – Testing

- Compiling the code

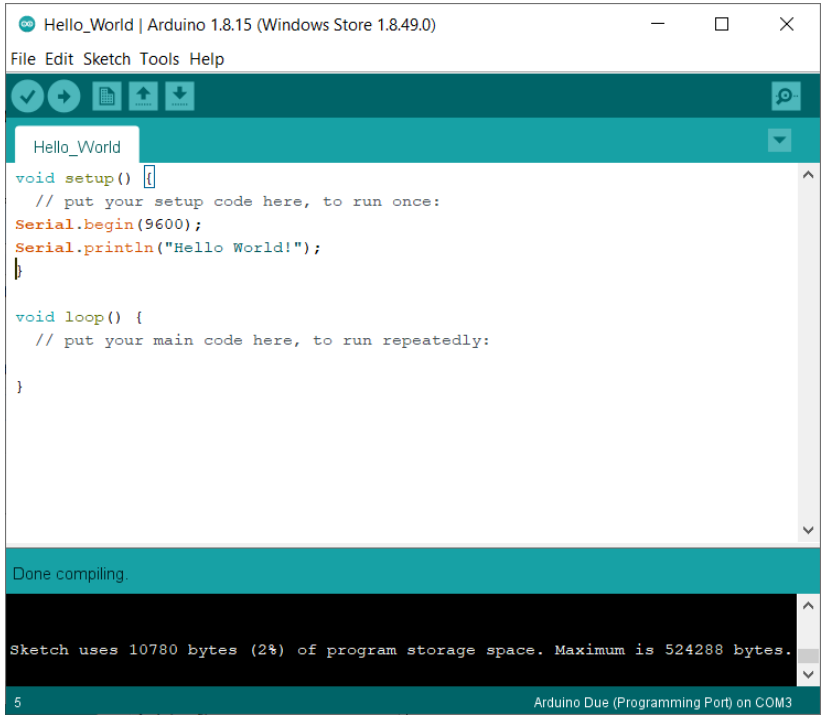


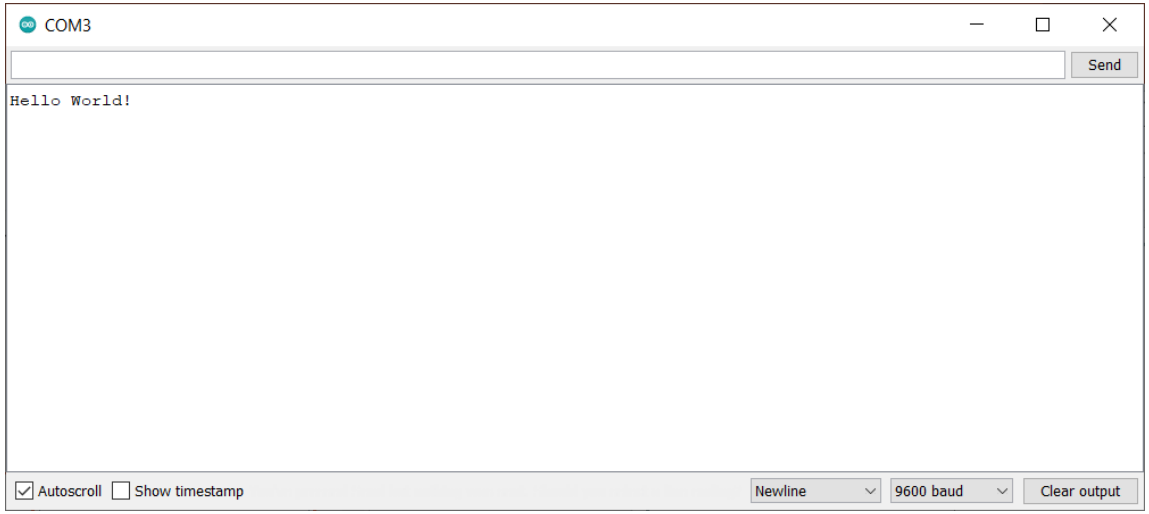


Hardware – Arduino Due – Testing

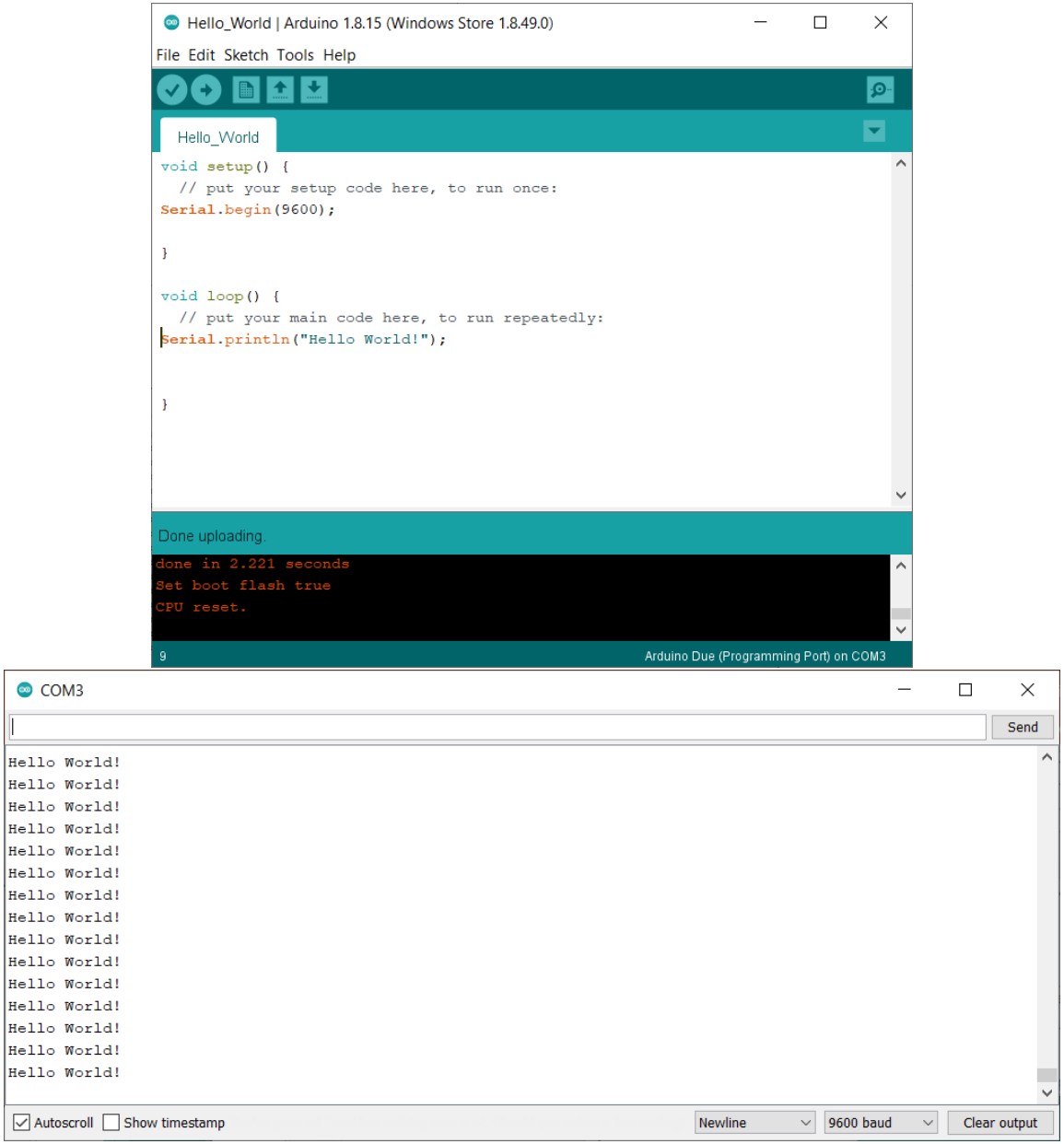


Hardware – Arduino Due – Hello World



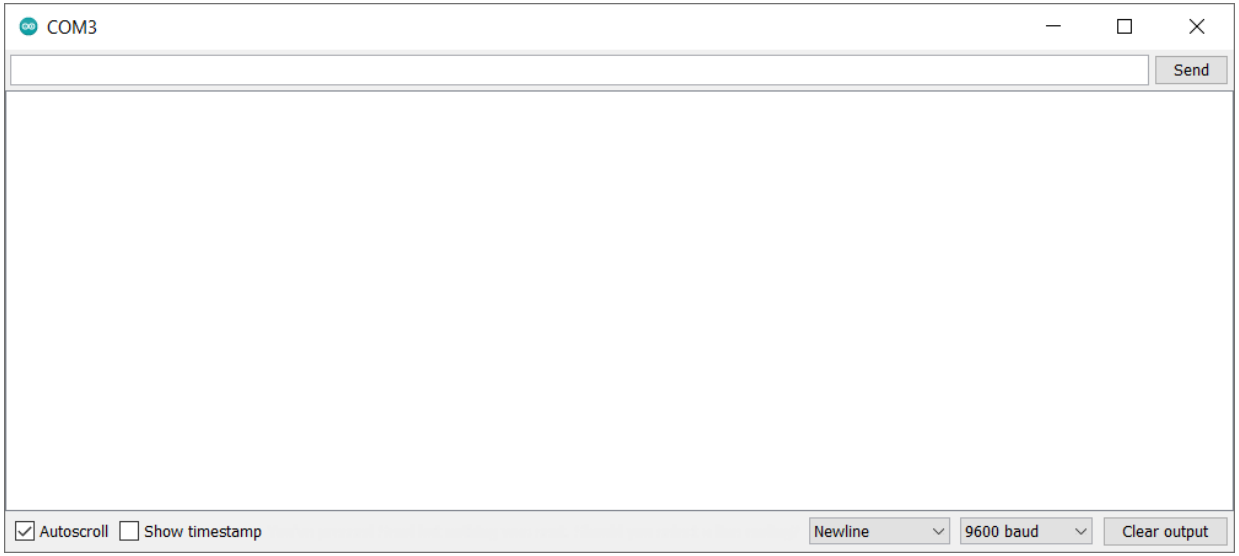


Hardware – Arduino Due – Hello World



Displaying Data from the Arduino in the Serial Monitor

- To open the Serial Monitor, start the IDE and click the Serial Monitor icon button on the tool bar, shown in below.
- Serial Monitor displays an input field at the top, consisting of a single row and a Send button, and an output window below it, where data from the Arduino is displayed. When the Autoscroll box is checked, the most recent output is displayed, and once the screen is full, older data rolls off the screen as newer output is received. If you uncheck Autoscroll, you can manually examine the data using a vertical scroll bar.



Displaying Data from the Arduino in the Serial Monitor

- **Starting the Serial Monitor**: Before we can use the Serial Monitor, we need to activate it by adding this function to our sketch in void setup().
- The value 9600 is the speed at which the data will travel between the computer and the Arduino, also known as baud. This value must match the speed setting at the bottom right of the Serial Monitor.

```
Serial.begin(9600);
```

- **Sending Text to the Serial Monitor**: To send text to the Serial Monitor to be displayed in the output window, you can use Serial.print. This sends the text between the quotation marks to the Serial Monitor's output window.

```
Serial.print("Arduino for Everyone!");
```

Displaying Data from the Arduino in the Serial Monitor

- You can also use Serial.println to display text and then force any following text to start on the next line:

```
Serial.println("Arduino for Everyone!");
```

- **Displaying the Contents of Variables**: You can also display the contents of variables on the Serial Monitor. For example, this would display the contents of the variable results.

```
Serial.println(results);
```

- If the variable is a float, the display will default to two decimal places. You can specify the number of decimal places used as a number between 0 and 6 by entering a second parameter after the variable name. For example, to display the float variable results to four decimal places.

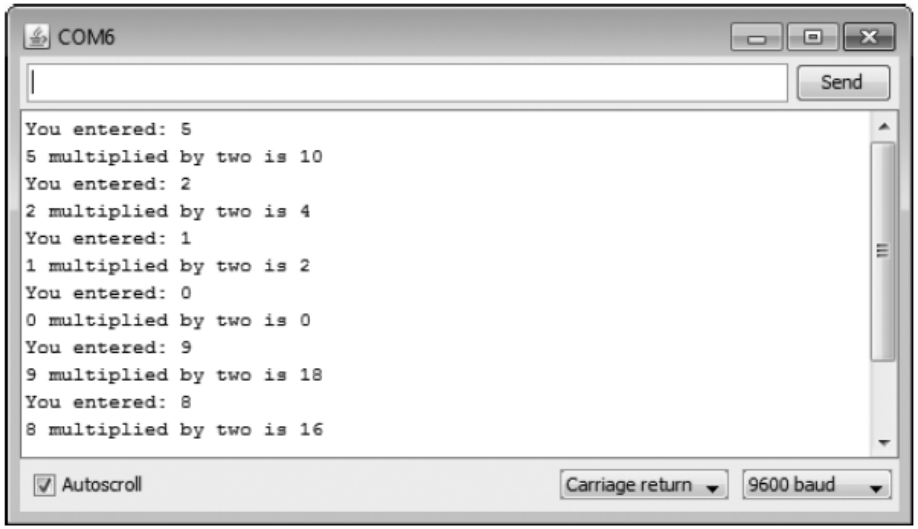
```
Serial.println(results);
```

Sending Data from the Serial Monitor to the Arduino

- To send data from the Serial Monitor to the Arduino, we need the Arduino to listen to the serial buffer.
- Example: Multiplying a Number by Two. To demonstrate the process of sending and receiving data via the Serial Monitor, let's dissect the following sketch. This sketch accepts a single digit from the user, multiplies it by 2, and then displays the result in the Serial Monitor's output window.
- The Serial.available() test in the first while statement at u returns 0 if nothing is entered yet into the Serial Monitor by the user. In other words, it tells the Arduino, "Do nothing until the user enters something." The next while statement at v detects the number in the serial buffer and converts the text code that represents the data entered into an actual integer number. Afterward, the Arduino displays the number from the serial buffer and the multiplication results.
- The Serial.flush() function at the start of the sketch clears the serial buffer just in case any unexpected data is in it, readying it to receive the next available data.

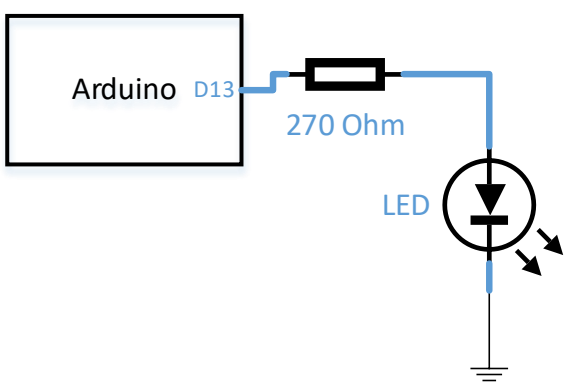
Sending Data from the Serial Monitor to the Arduino

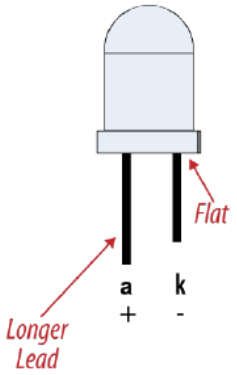
```
int number;

void setup()
{
  Serial.begin(9600);
}
void loop()
{
  number = 0;     // zero the incoming number ready for a new read
  Serial.flush(); // clear any "junk" out of the serial buffer before waiting
❶ while (Serial.available() == 0)
  {
    // do nothing until something enters the serial buffer
  }
❷ while (Serial.available() > 0)
  {
    number = Serial.read() - '0';
// read the number in the serial buffer,
// remove the ASCII text offset for zero: '0'
  }
  // Show me the number!
  Serial.print("You entered: ");
  Serial.println(number);
  Serial.print(number);
  Serial.print(" multiplied by two is ");
  number = number * 2;
  Serial.println(number);
}
```



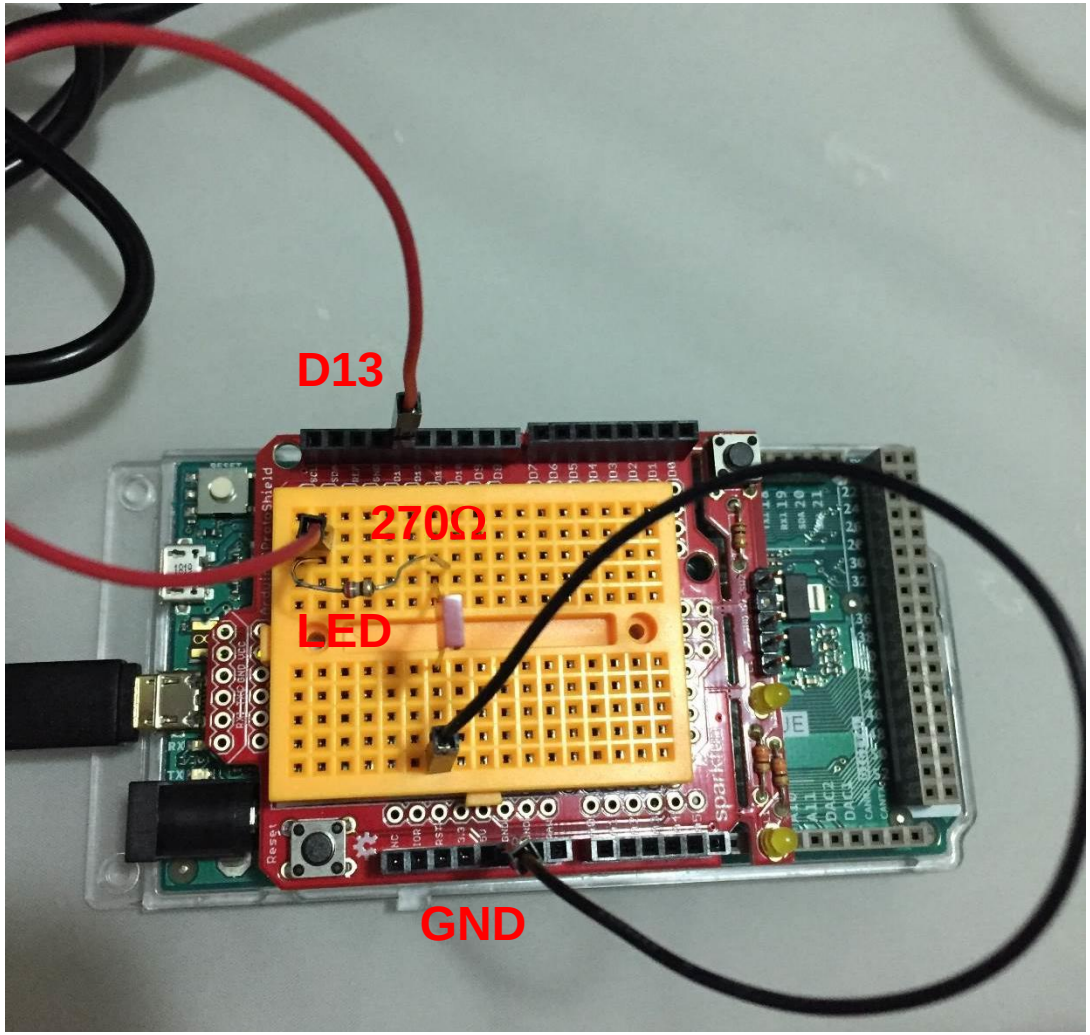
2. GPIO Output – external LED

- we need the following component:
  1. An Arduino board connected to the computer via USB
  2. A breadboard and jumper wires
  3. A regular LED
  4. A resistor between 220–1,000 ohm
- Mount the resistor on the breadboard. Connect one end of the resistor to a digital pin on the Arduino board using a jumper wire.
- Mount the LED on the breadboard. Connect the anode (+) pin of the LED to the available pin on the resistor. We can determine the anode on the LED in two ways. Usually, the longer pin is the anode. Another way is to look for the flat edge on the outer casing of the LED. The pin next to the flat edge is the cathode (-).
- Connect the LED cathode (-) to the Arduino GND using jumper wires.

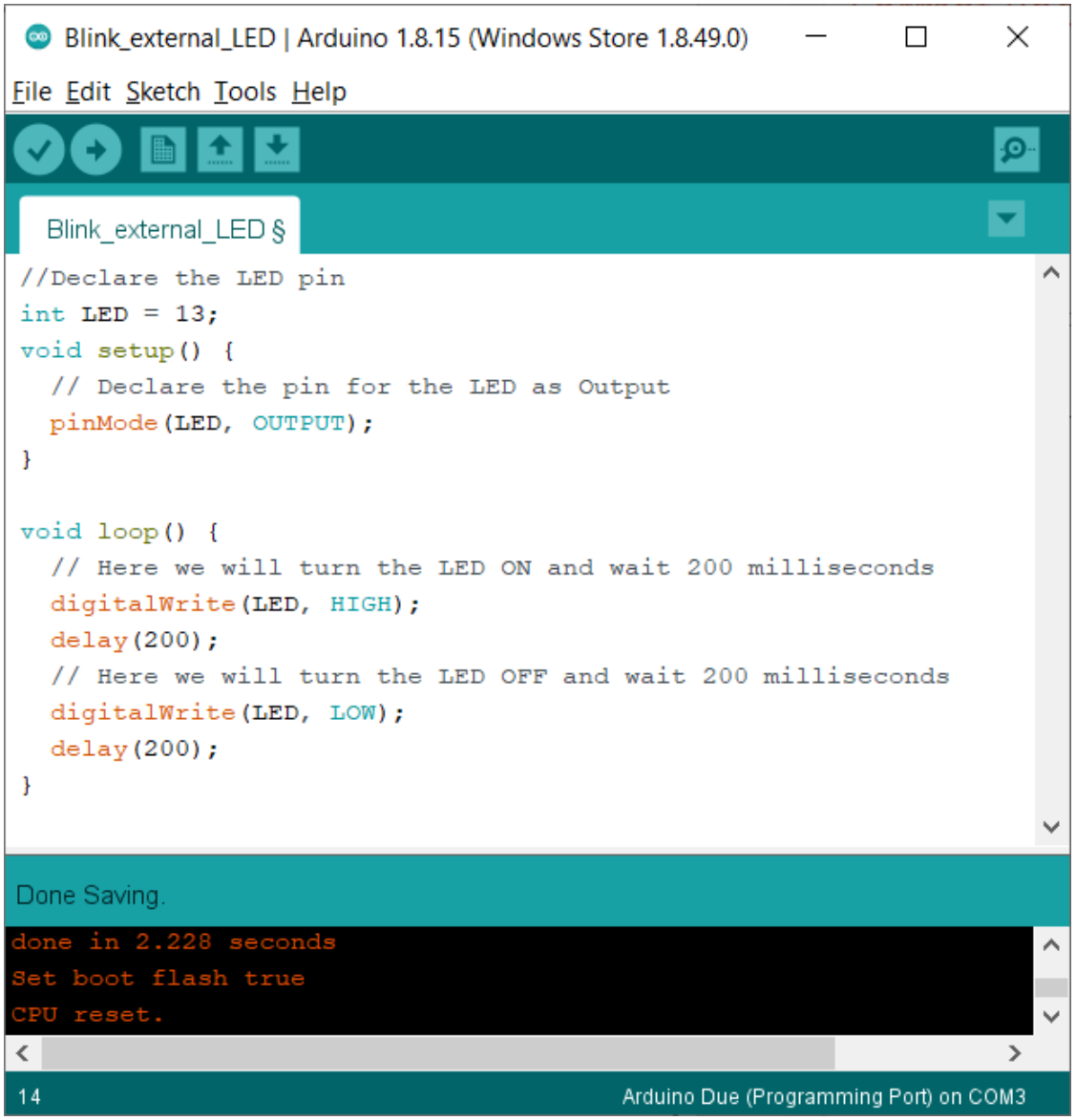




GPIO Output – external LED



GPIO Output – external LED



GPIO Output – Connecting an external LED

- How it works: when the second digital pin is set to HIGH, the Arduino provides 3.3 or 5 V of electricity, which travels through the resistor to the LED and GND.
- When enough voltage and current is present, the LED will light up. The resistor limits the amount of current passing through the LED.
- Without it, it is possible that the LED (or worse, the Arduino pin) will burn.
- Try to avoid using LEDs without resistors; this can easily destroy the LED or even your Arduino.

GPIO Output – external LED

- Code breakdown:
  - The code simply turns the LED on, waits, and then turns it off again.
  - We will use a blocking approach by using the delay() function.
  - Here we declare the LED pin on digital pin 13 for Arduino Due;
  - In the setup() function we set the LED pin as an output:

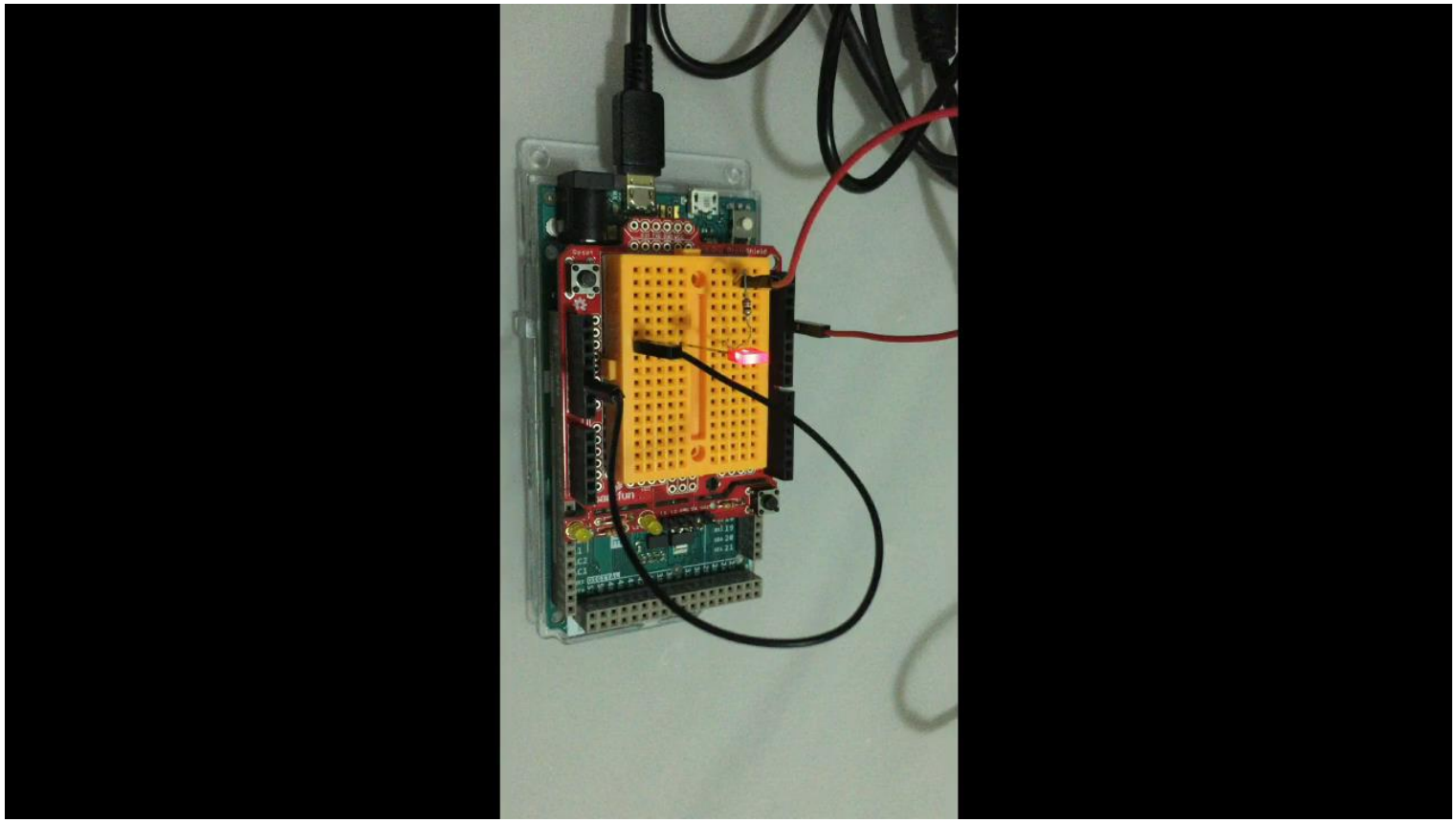
```
//Declare the LED pin
int LED = 13;
void setup() {
  // Declare the pin for the LED as Output
  pinMode(LED, OUTPUT);
}
```

GPIO Output – external LED

- In the loop() function, we continuously turn the LED on, wait 200 milliseconds, and then we turn it off.
- After turning it off we need to wait another 200 milliseconds, otherwise it will instantaneously turn on again and we will only see a permanently on LED.

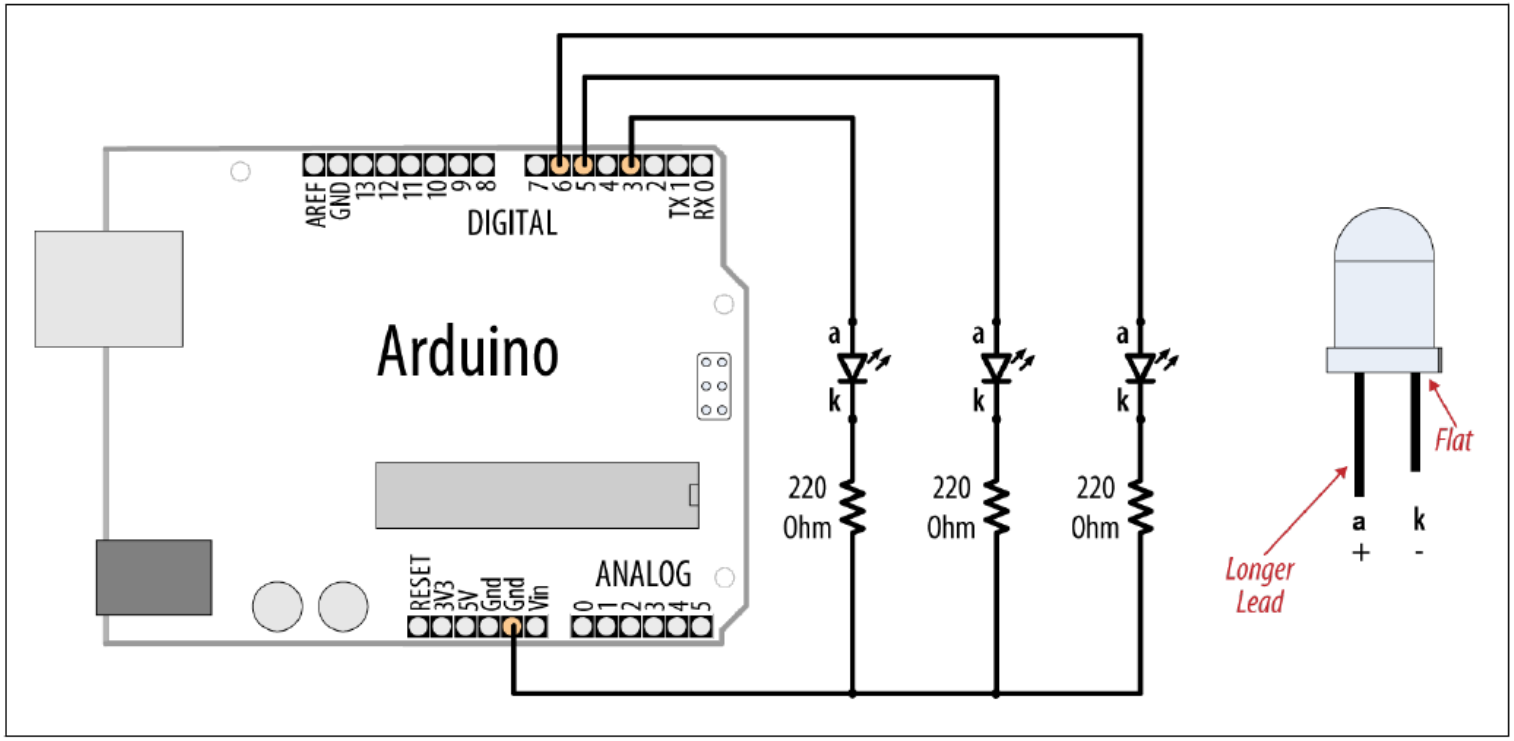
```
void loop() {
  // Here we will turn the LED ON and wait 200 milliseconds
  digitalWrite(LED, HIGH);
  delay(200);
  // Here we will turn the LED OFF and wait 200 milliseconds
  digitalWrite(LED, LOW);
  delay(200);
}
```

GPIO Output – external LED



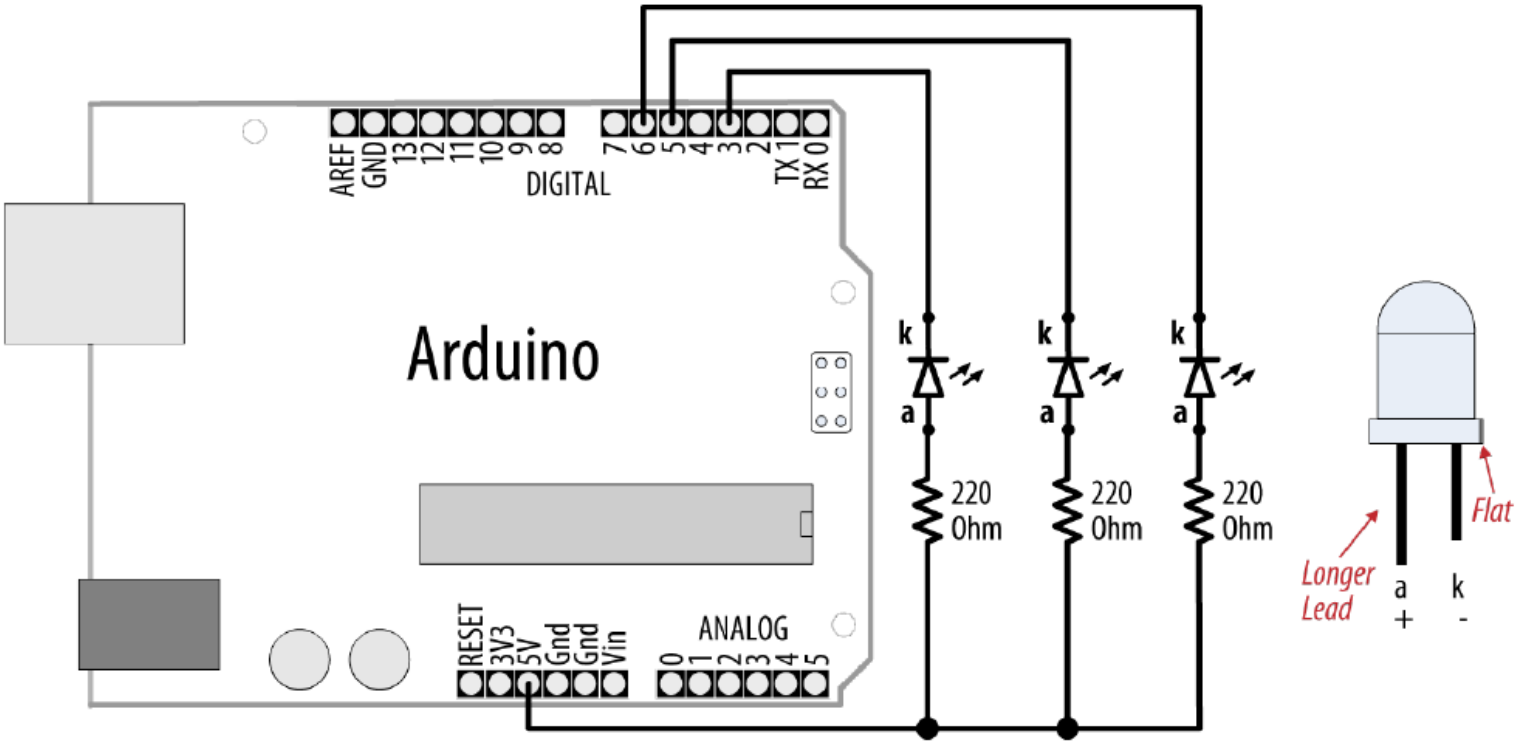
GPIO Output – Multiple LEDs

- Connecting external LEDs with the Anode connected to pins of embedded system

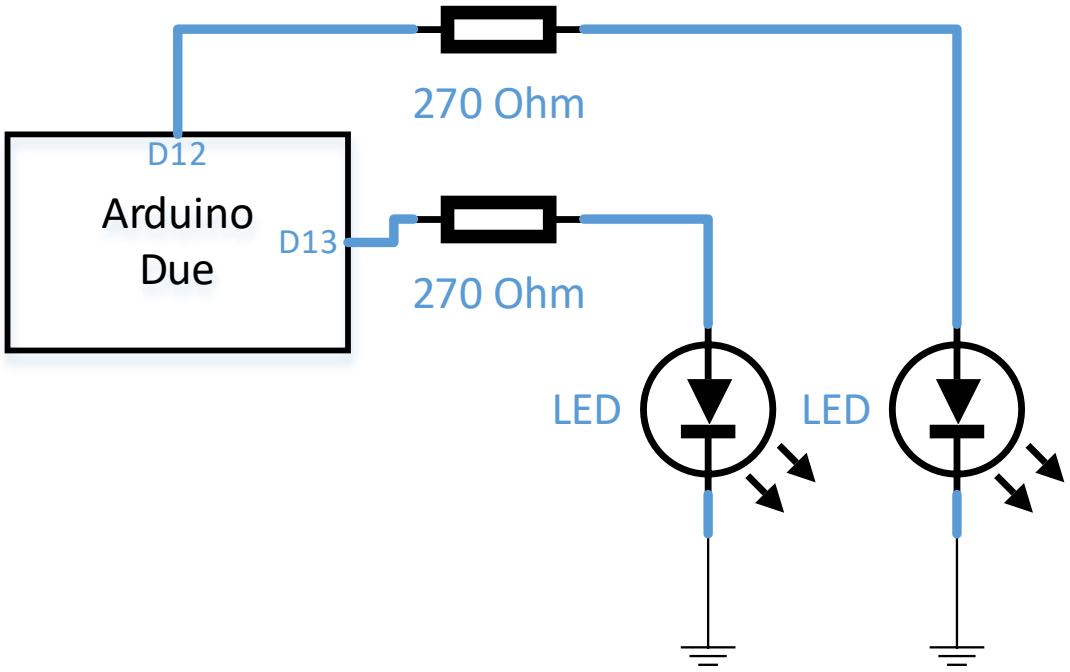


GPIO Output – Multiple LEDs

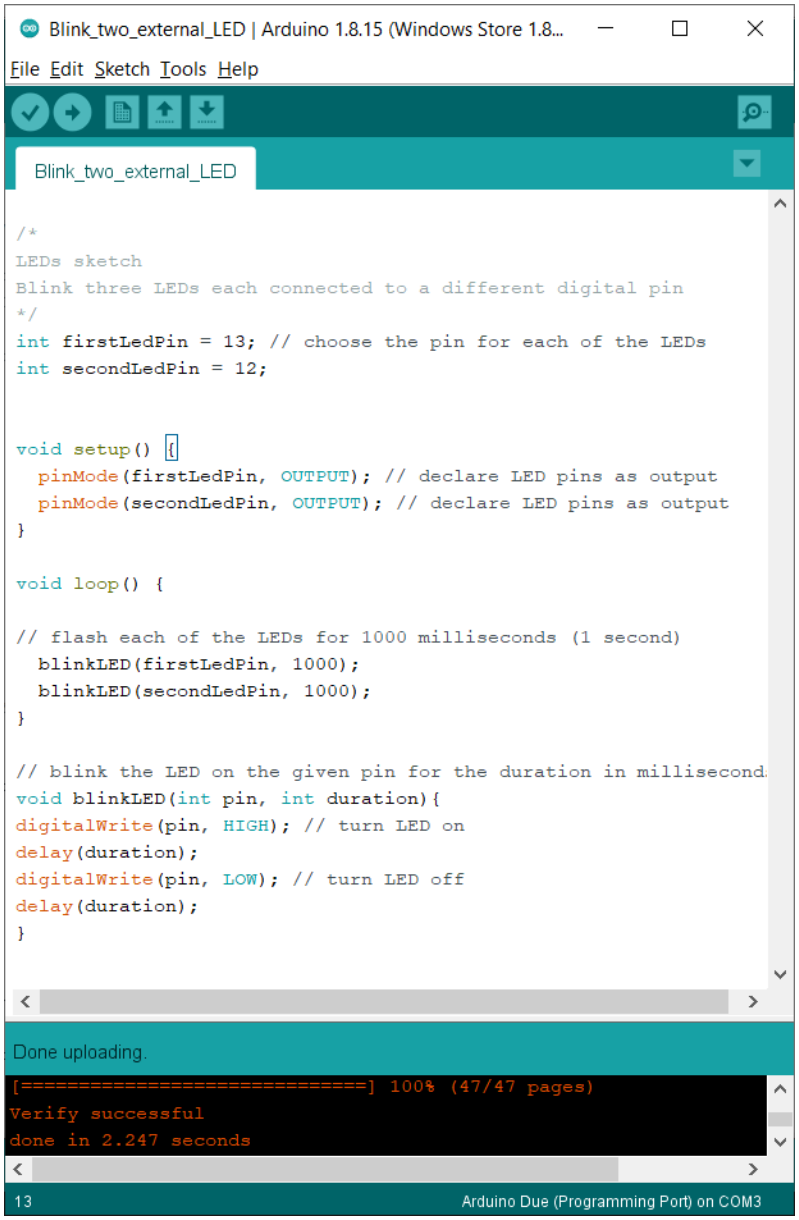
- Connecting external LEDs with the Chatode connected to pins of embedded system



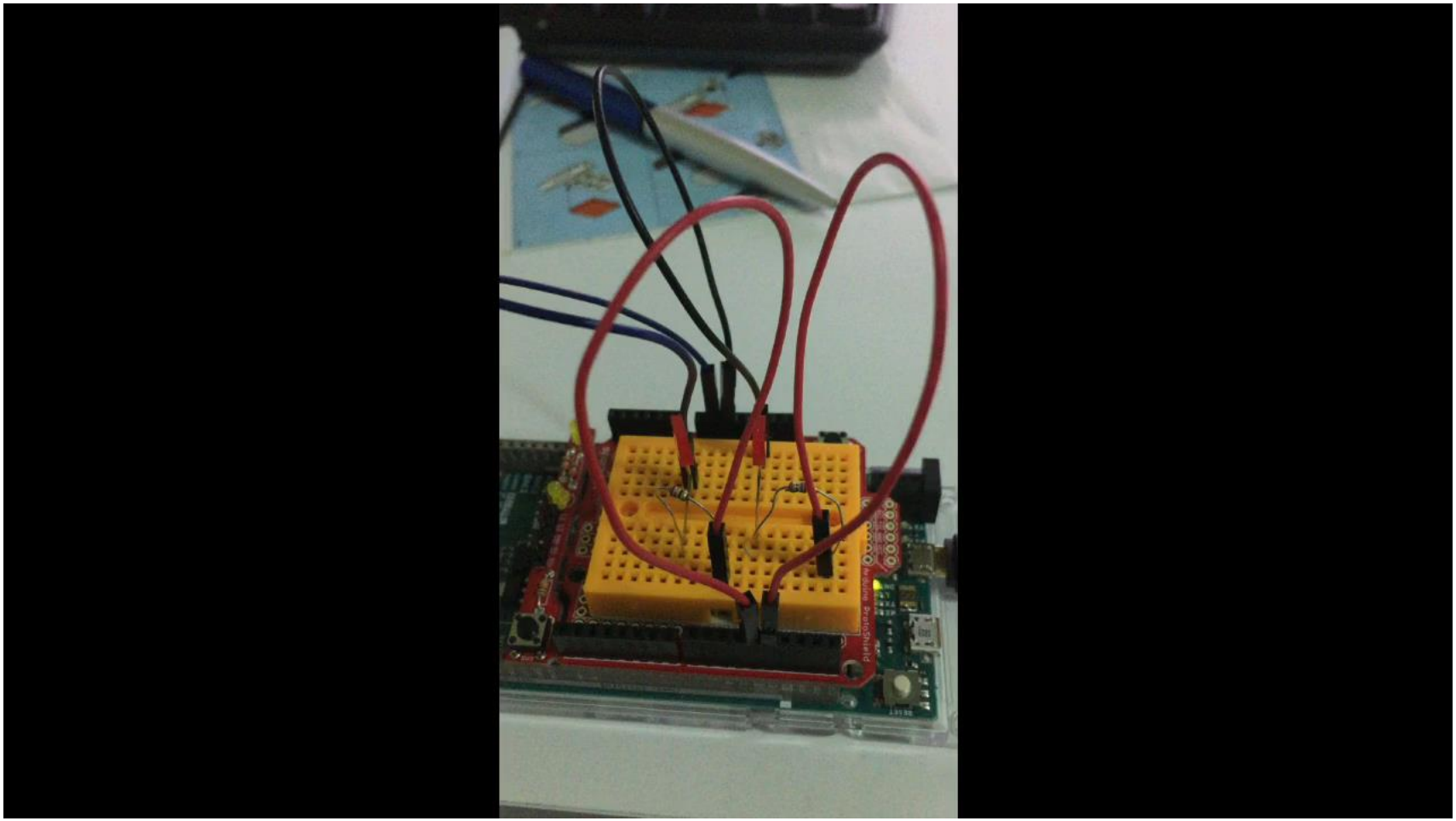
GPIO Output – Multiple LEDs



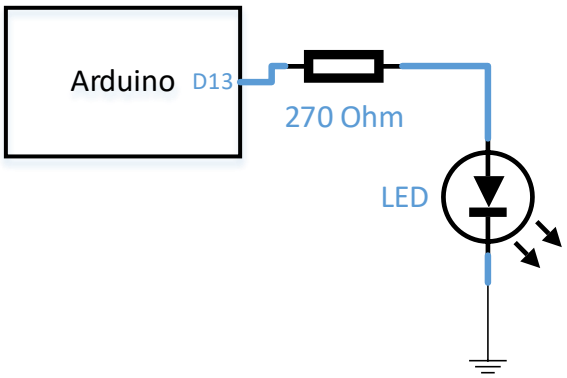
GPIO Output – Multiple LEDs



GPIO Output – Multiple LEDs



3. GPIO Output – LED with PWM



LED_PWM | Arduino 1.8.15 (Windows Store 1.8....

File Edit Sketch Tools Help

LED_PWM

```
// Declare the LED pin with PWM
int LED = 13;

void setup() {
// Declare the pin for the LED as Output
  pinMode(LED, OUTPUT);
}

void loop() {
// Here we will fade the LED from 0 to maximum, 255
  for (int i = 0; i < 256; i++){
    analogWrite(LED, i);
    delay(5);
  }
// Fade the LED from maximum to 0
  for (int i = 255; i >= 0; i--){
    analogWrite(LED, i);
    delay(5);
  }
}
```

Done uploading.

```
done in 5.053 seconds
Set boot flash true
CPU reset.
```

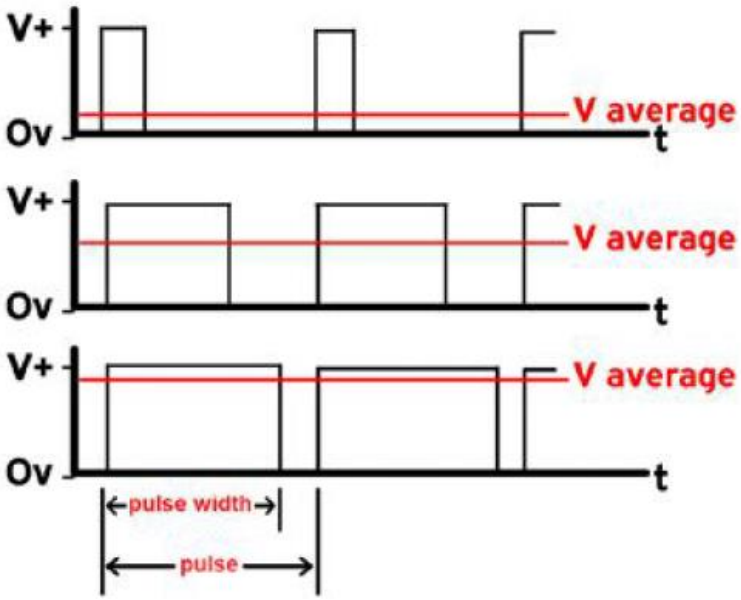
20 Arduino Due (Programming Port) on COM3

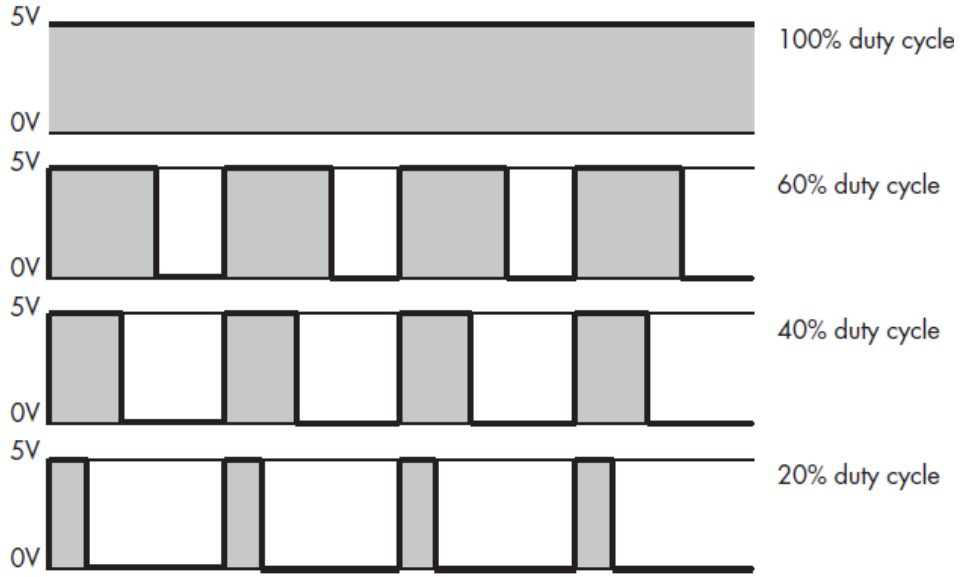
GPIO Output – LED with PWM

- Rather than just turning LEDs on and off rapidly using digitalWrite(), we can define the level of brightness of an LED by adjusting the amount of time between each LED's on and off states using pulse-width modulation (PWM).
- The brightness we perceive is determined by the amount of time the digital output pin is on versus the amount of time it is off—that is, every time the LED is lit or unlit. Because our eyes can't see flickers faster than 50 cycles per second, the LED appears to have a constant brightness.
- The greater the duty cycle (the longer the pin is on compared to off in each cycle), the greater the perceived brightness of the LED connected to the digital output pin.
- This all works with Pulse Width Modulation (PWM), which works by switching between LOW and HIGH very fast.
- If we turn a digital pin on and off a thousand times per second, we will obtain, on average, a voltage that is half of the HIGH voltage.
- If the ratio between HIGH and LOW is 2:3, the obtained voltage will be two-thirds of the HIGH voltage and so on.

GPIO Output – LED with PWM

- The following diagrams better explains how PWM works and various PWM duty cycles:



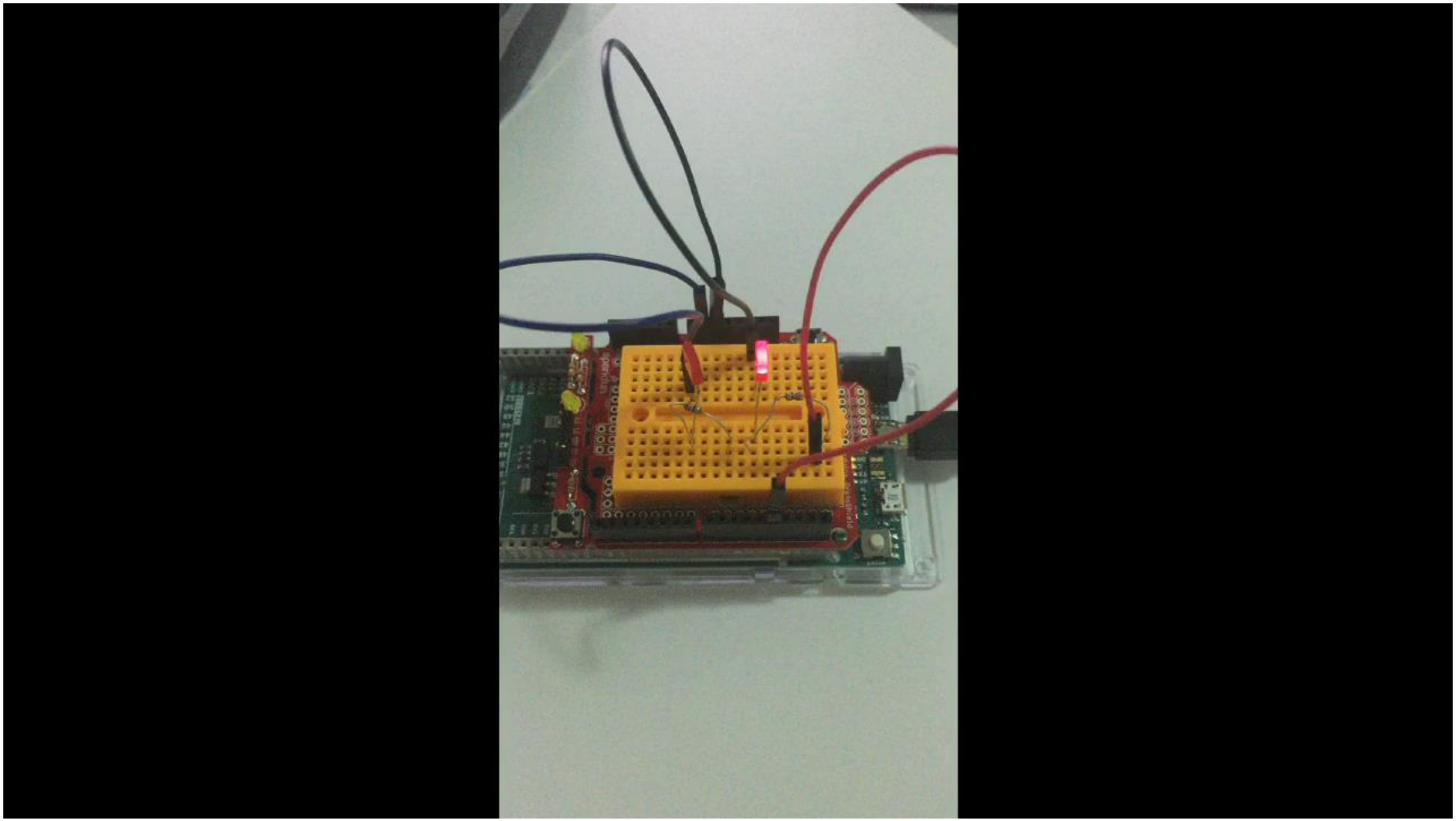


GPIO Output – LED with PWM

- In the loop() function, we use the important PWM function analogWrite().
- This function provides an analog signal on the digital PWM pin.
- The values for the voltage can be between 0–255, 0 for 0 volts and 255 for 5 V or 3.3 V, depending on the Arduino board used.
- Here, we fade in the LED slowly using a for function and then we fade it out:

```
void loop(){
  // Here we will fade the LED from 0 to maximum, 255
  for (int i = 0; i < 256; i++){
    analogWrite(LED, i);
    delay(5);
  }
  // Fade the LED from maximum to 0
  for (int i = 255; i >= 0; i--){
    analogWrite(LED, i);
    delay(5);
  }
}
```

GPIO Output – LED with PWM

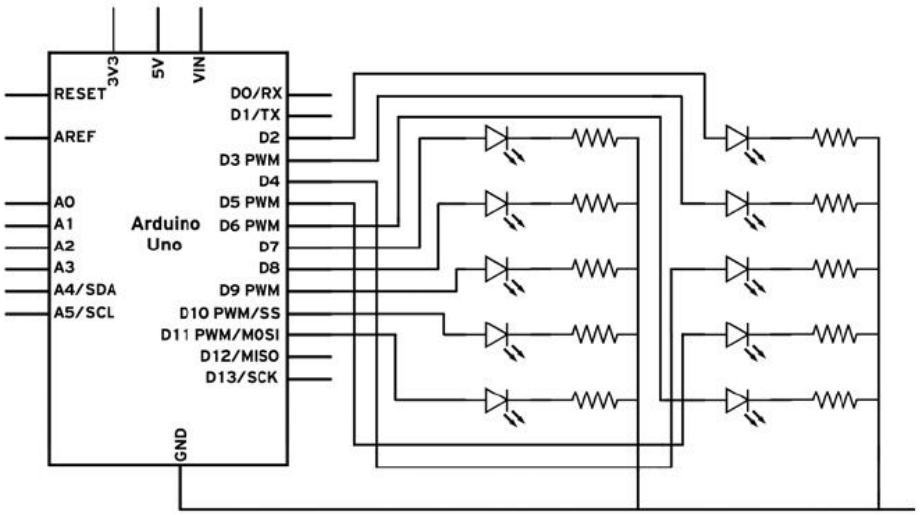


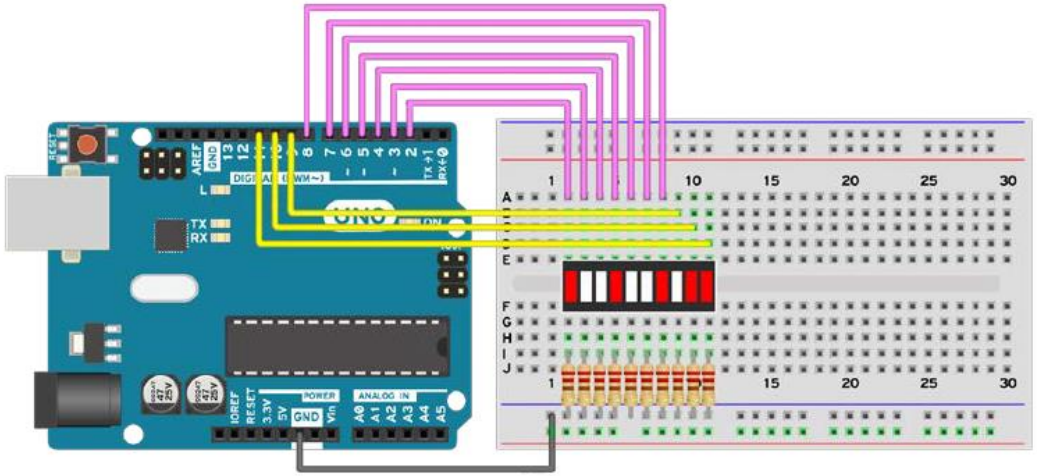
4. GPIO Output – LED bar graph

- An LED bar graph is just a bunch of LEDs put together in a fancy case, but there are many uses for it.
- We can display the date from a sensor, show a critical condition, or make a funny light show with it, etc.
- Following are the steps to connect a 10-segment bar graph to the Arduino:
  - Mount the LED bar graph onto the breadboard.
  - If the bar graph is a common anode (+) configuration, connect the common anode (+) pin to the 5V/3.3V port on the Arduino. If the bar graph is a common cathode (-), connect the pin to the GND port on the Arduino.
  - Connect each individual segment pin to one individual Arduino digital pin, using a resistor. To make things simple, connect all the segment pins to successive digital pins on the Arduino.

GPIO Output – LED bar graph

- Example connection of a common anode (+) 10-segment LED bar graph:





GPIO Output – LED bar graph

- The following code will make the LED bar graph full and then empty (designed for for a common anode (+) configuration)

```
// Declare the first and last Pin of the LED Bar
int pin1 = 2;
int pin10 = 11;

void setup() {
  // Declare the pins as Outputs
  for (int i = pin1; i <= pin10; i++){
    pinMode(i, OUTPUT);
  }
}

// A simple function to set the value of the LED Bar
void setBarValue(int value){
  // First we turn everything off
  for (int i = pin1; i <= pin10; i++){
    digitalWrite(i, HIGH);
  }

  // Write the value we want
  for (int i = pin1; i <= pin1 + value; i++){
    digitalWrite(i, LOW);
  }

  // In case we have value 0
  if (value == 0){
  digitalWrite(pin1, HIGH);
  }
}

void loop(){
  // Play with a few displays

  // Ping-Pong
  for (int i = 0; i <= 10; i++){
    setBarValue(i);
    delay(100);
  }
 for (int i = 10; i >= 0; i--){
    setBarValue(i);
    delay(100);
  }
}
```

GPIO Output – LED bar graph

- Codes breakdown: This code loads and unloads the LED bar graph just like a progress bar. Here, we declare the first and the last pins used in the LED bar. There is no point in declaring all of them as we know they are consecutive in this implementation:

```
int pin1 = 2;
int pin10 = 11;
```

- In the setup() function, we set each LED pin as an output. This simple trick, used here, helps to set all the pins between pin1 and pin10 as outputs:

```
void setup() {
  // Declare the pins as Outputs
  for (int i = pin1; i <= pin10; i++){
    pinMode(i, OUTPUT);
  }
}
```

GPIO Output – LED bar graph

- In the custom setBarValue() function, we make the bar show a certain progress level. As the maximum is 10 and the minimum is 0, if we write 5, half the LEDs on the bar will be on while the other half are off:

```
// A simple function to set the value of the LED Bar
void setBarValue(int value){
  // First we turn everything off
  for (int i = pin1; i <= pin10; i++){
    digitalWrite(i, HIGH);
  }

  // Write the value we want
  for (int i = pin1; i <= pin1 + value; i++){
    digitalWrite(i, LOW);
  }

  // In case we have value 0
  if (value == 0) digitalWrite(pin1, HIGH);
}
```

GPIO Output – LED bar graph

- Finally, in the loop() function, we use our custom function to load the bar and then unload it. In the following code, we use a for loop to increase the bar value to the maximum and then we decrease it back to 0:

```
void loop(){
  for (int i = 0; i <= 10; i++){
    setBarValue(i);
    delay(100);
  }
 for (int i = 10; i >= 0; i--){
    setBarValue(i);
    delay(100);
  }
}
```

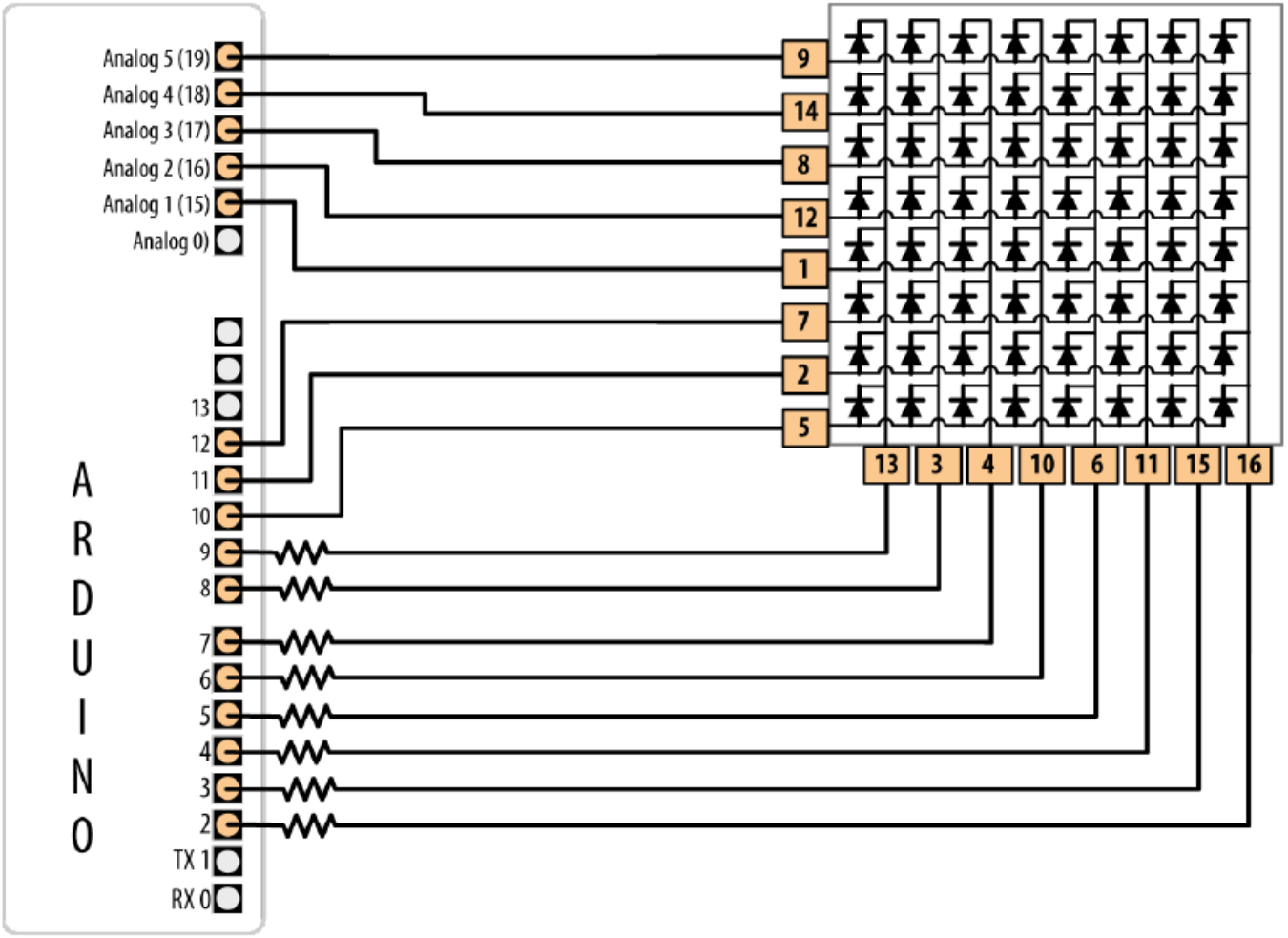
- Variations of LED bar graph: Common anode (+) and common cathode (-)
- Each LED bar is either a common anode (+) or a common cathode (-).
- If it's a common anode (+), we connect the anode to 5V/3.3V, each other pin to a resistor, and the resistors to individual digital pins on the Arduino.
- For the common cathode (-), connect the cathode to the GND and each pin, using a resistor, to individual Arduino digital pins.

5. GPIO Output – LED Matrix

- If there is a matrix of LEDs and want to minimize the number of Arduino pins needed to turn LEDs on and off, one of the solutions is by *controlling an LED Matrix using multiplexing*.
- The following example uses an LED matrix of 64 LEDs, with anodes connected in rows and cathodes in columns.
- LED matrix displays do not have a standard pinout, so you must check the data sheet for your display. Wire the rows of anodes and columns of cathodes as shown in the following figure but use the LED pin numbers shown in your data sheet.
- The resistor's value must be chosen to ensure that the maximum current through a pin does not exceed 40 mA. Because the current for up to eight LEDs can flow through each column pin, the maximum current for each LED must be one-eighth of 40 mA, or 5 mA.
- Each column of the matrix is connected through the series resistor to a digital pin. When the column pin goes low and a row pin goes high, the corresponding LED will light. For all LEDs where the column pin is high or its row pin is low, no current will flow through the LED and it will not light.

GPIO Output – LED Matrix

- Example of the schematic/diagram circuit for LED Matrix:



GPIO Output – LED Matrix

- Example of coding for LED Matrix:

```
/*
  matrixMpx sketch

 Sequence LEDs starting from first column and row until all LEDS are lit
 Multiplexing is used to control 64 LEDs with 16 pins
 */

const int columnPins[] = { 2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]   = { 10,11,12,15,16,17,18,19};

int pixel       = 0;        // 0 to 63 LEDs in the matrix
int columnLevel = 0;        // pixel value converted into LED column
int rowLevel    = 0;        // pixel value converted into LED row

void setup() {
  for (int i = 0; i < 8; i++)
  {
    pinMode(columnPins[i], OUTPUT);  // make all the LED pins outputs
    pinMode(rowPins[i], OUTPUT);
  }
}
```

GPIO Output – LED Matrix

- Example of coding for LED Matrix:

```
void loop() {
  pixel = pixel + 1;
  if(pixel > 63)
     pixel = 0;

  columnLevel = pixel / 8;                    // map to the number of columns
  rowLevel = pixel % 8;                       // get the fractional value
  for (int column = 0; column < 8; column++)
  {
    digitalWrite(columnPins[column], LOW);    // connect this column to Ground
    for(int row = 0; row < 8; row++)
    {
      if (columnLevel > column)
      {
        digitalWrite(rowPins[row], HIGH);  // connect all LEDs in row to +5 volts
      }
      else if (columnLevel == column && rowLevel >= row)
      {
          digitalWrite(rowPins[row], HIGH);
      }
      else
      {
        digitalWrite(columnPins[column], LOW); // turn off all LEDs in this row
      }
      delayMicroseconds(300);     // delay gives frame time of 20ms for 64 LEDs
      digitalWrite(rowPins[row], LOW);         // turn off LED
    }

    // disconnect this column from Ground
    digitalWrite(columnPins[column], HIGH);
  }
}
```

GPIO Output – LED Matrix

- The for loop scans through each row and column and turns on sequential LEDs until all LEDs are lit.
- The loop starts with the first column and row and increments the row counter until all LEDs in that row are lit; it then moves to the next column, and so on, lighting another LED with each pass through the loop until all the LEDs are lit.
- You can control the number of lit LEDs in proportion to the value from a sensor by making the following changes to the sketch/Arduino program. Comment out or remove these three lines from the beginning of the loop:

```
pixel = pixel + 1;
if(pixel > 63)
   pixel = 0;
```

- Replace them with the following lines that read the value of a sensor on pin 0 and map this to a number of pixels ranging from 0 to 63:

```
int sensorValue = analogRead(0);              // read the analog in value
pixel =   map(sensorValue, 0, 1023, 0, 63);   // map sensor value to pixel (LED)
```

- The number of LEDs lit will be proportional to the value of the sensor.

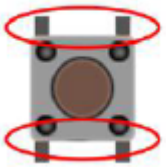
GPIO Input – Button

- Buttons are the basis of human interaction with the Arduino. We press a button, and something happens. They are simple components, as they only have two states: opened or closed. When a button is closed, current can pass though it. When it's opened, no current can pass. Some buttons are closed when we push them, some when they are released.
- Momentary buttons are active as long as they are pressed, while maintained buttons keep the state we let them in. Keyboards have momentary buttons while the typical light switch is a maintained button.
- Momentary buttons can either be opened or closed. This reflects the connection state when not pressed. A closed momentary switch will conduct current while not pressed and interrupt the current when pressed. An opened button will do the opposite. Lastly, there are poles and throws. The following figure explains the main two types:

6. GPIO Input – Button

- Single Pole Single Throw (SPST) switches have a closed state in which they conduct current, and an opened state in which they do not conduct current. Most momentary buttons are SPST.

- Single Pole Double Throw (SPDT) switches route the current from the common pin to one of the two outputs. They are typically maintained; one example of this is the common light switch.
- The common button we'll be using in this chapter is a push button. It's a small momentary opened switch. It typically comes in a 4-pin case.
- The pins inside the two red ellipses are shorted together. When we press the button, all four pins are connected.

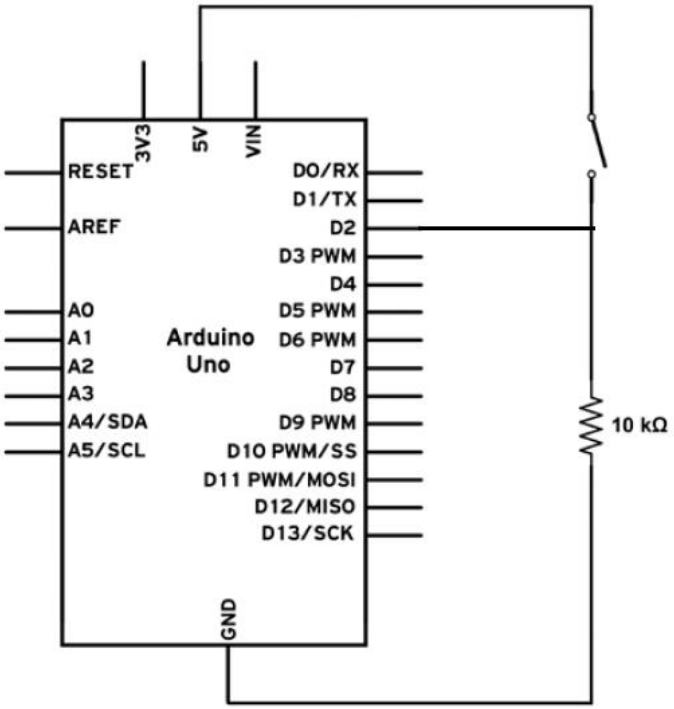


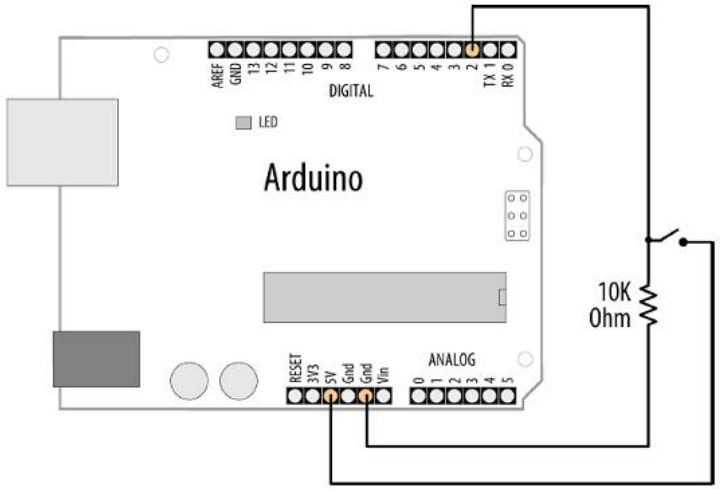
GPIO Input – Button

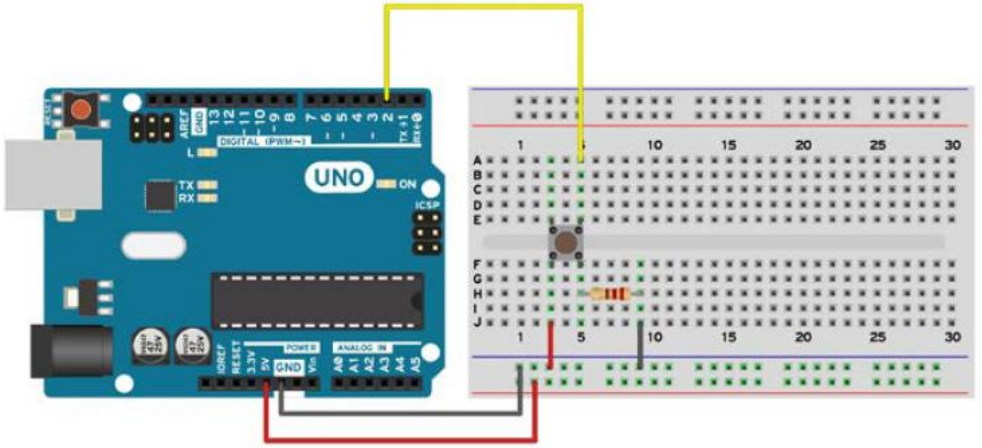
- The following is example of connecting a button:
  - Connect the Arduino GND and 5V/3.3V to separate long strips on the breadboard.
  - Mount the push button on the breadboard and connect one terminal to the 5V/3.3V long strip and the other to a digital pin on the Arduino—in this example, pin 2.
  - Mount the resistor between the chosen digital pin and the GND strip. This iscalled a pull-down setup. More on this later.

GPIO Input – Button

- Schematic for connecting button:







GPIO Input – Button

- The following code will read if the button has been pressed and will control the built-in LED.
- If the button is connected to a different pin, simply change the buttonPin value to the value of the pin that has been used.
- The purpose of the button is to drive the digital pin to which it's connected to either HIGH or LOW. In theory, this should be very simple: just connect one end of the button to the pin and the other to 5V. When not pressed, the voltage will be LOW; otherwise it will be 5V, HIGH.
- However, there is a problem. When the button is not pressed, the input will not be LOW but instead a different state called floating. In this state, the pin can be either LOW or HIGH depending on interference with other components, pins, and even atmospheric conditions.
- That's where the resistor comes in. It is called a pull-down resistor as it pulls the voltage down to GND when the button is not pressed. This is a very safe method when the resistor value is high enough. Any value over 1K will work just fine, but 10K ohm is recommended.

GPIO Input – Button

- Codes:

```
// Declare the pins for the Button and the LED
int buttonPin = 2;
int LED = 13;

void setup() {
  // Define pin #2 as input
  pinMode(buttonPin, INPUT);
  // Define pin #13 as output, for the LED
  pinMode(LED, OUTPUT);
}

void loop() {
  // Read the value of the input. It can either be 1 or 0.
  int buttonValue = digitalRead(buttonPin);

  if (buttonValue == HIGH) {
    // If button pushed, turn LED on
    digitalWrite(LED,HIGH);
  } else {
    // Otherwise, turn the LED off
    digitalWrite(LED, LOW);
  }
}
```

GPIO Input – Button

- The codes breakdown: the code takes the value from the button. If the button is pressed, it will start the built-in LED. Otherwise, it will turn it off.
- Here, we declare the pin to which the button is connected as pin 2, and the built-in LED on pin 13:

```
int buttonPin = 2;
int LED = 13;
```

- In the setup() function, we set the button pin as a digital input and the LED pin as an output:

```
void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(LED, OUTPUT);
}
```

- The important part comes in the loop() function. The first step is to declare a variable that will equal the value of the button state. This is obtained using the digitalRead() function:

```
int buttonValue = digitalRead(buttonPin);
```

GPIO Input – Button

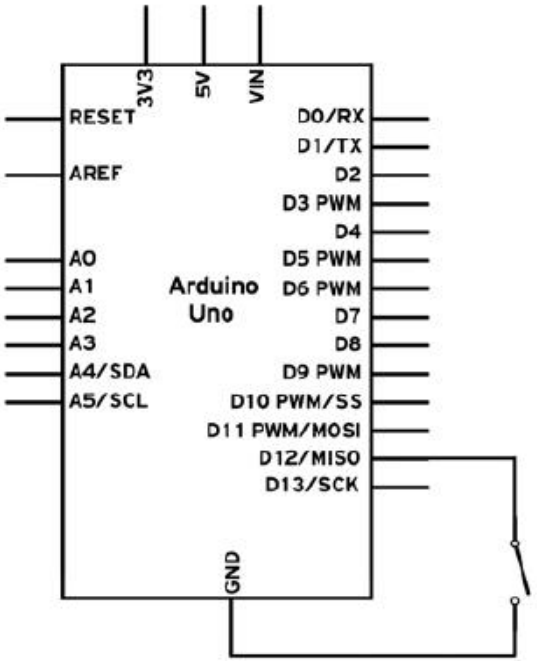
- Lastly, depending on the button state, we initiate another action. In this case, we just light up the LED or turn it off:

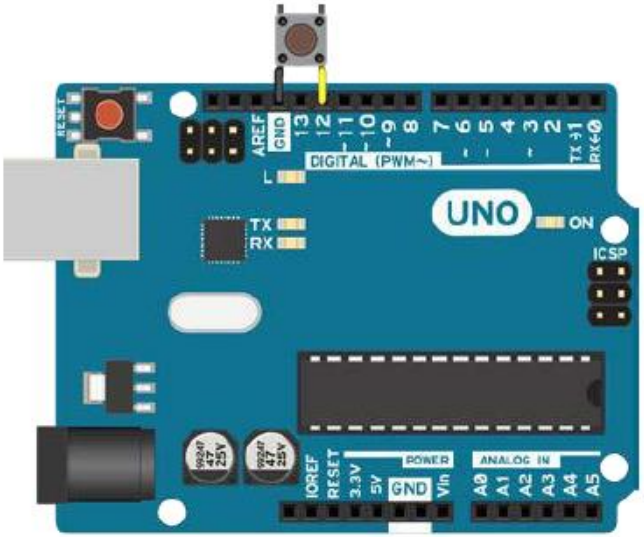
```
if (buttonValue == HIGH){
        digitalWrite(LED,HIGH);
  } else {
    // Otherwise, turn the LED off
    digitalWrite(LED, LOW);
  }
```

- In a pull-up configuration, the resistor will pull up the voltage to 5V when the button is not pressed. To implement it, connect one terminal of the button to the digital pin and the other one to GND. Now, connect the resistor between the digital pin and 5V. This configuration will return inverted values. When pressed, the button will give LOW, not HIGH, as it will draw the pin down to GND, 0 V. However, it brings no advantages over the pull-down configuration.

7. GPIO Input – Button with no resistor

- The resistor is mandatory for proper operation of a button, and everybody will insist on using it. However, there is a little secret embedded in each Arduino pin. Each pin already has a pull-up resistor that we can enable with just one small change in our code.
- The connection by Connect the Arduino GND to a terminal on the button and connect the chosen digital pin to the other terminal.





GPIO Input – Button with no resistor

- Codes: If the button is connected to a different pin, change the buttonPin value to the value of the pin that has been used.
- When we press the button, the value of the Arduino pin should be either LOW or HIGH. In this configuration, when we press the button, the pin is connected directly to GND, resulting in LOW.
- However, when it is not pressed, the pin will have no value; it will be in a floating state. To avoid this, an internal pull-up resistor is connected between each pin and 5V. When we activate the resistor, it will keep the pin at HIGH until we press the button, thus connecting the pin to GND.

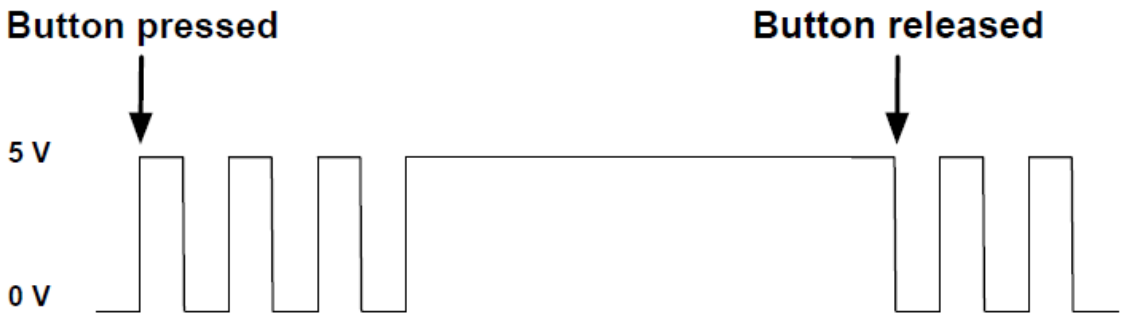
GPIO Input – Button with no resistor

- Codes:

```
// Declare the pins for the Button and the LED
int buttonPin = 12;
int LED = 13;

void setup() {
  // Define pin #12 as input and activate the internal pull-up
resistor
  pinMode(buttonPin, INPUT_PULLUP);
  // Define pin #13 as output, for the LED
  pinMode(LED, OUTPUT);
}

void loop(){
  // Read the value of the input. It can either be 1 or 0
  int buttonValue = digitalRead(buttonPin);

  if (buttonValue == LOW){
    // If button pushed, turn LED on
    digitalWrite(LED,HIGH);
  } else {
    // Otherwise, turn the LED off
    digitalWrite(LED, LOW);
  }
}
```

8. GPIO Input – Button with Debouncing

- You will still find some cases when the button doesn't behave fully as expected. Problems mainly occur in the moment you release the button.
- These problems occur because the mechanical buttons bounce for a few milliseconds when you press them. In the following figure, you can see a typical signal produced by a mechanical button. Right after you have pressed the button, it doesn't emit a clear signal.
- To overcome this effect, you have to debounce the button. It's usually sufficient to wait a short period of time until the button's signal stabilizes. Debouncing ensures that the input pin reacts only once to a push of the button:



8. GPIO Input – Button with Debouncing

- This final version of our LED switch differs from the previous one in only a single line: to debounce the button, we wait for 50 milliseconds in line 21 before we enter the main loop again.

BinaryDice/DebounceButton/DebounceButton.ino

```
Line 1  const unsigned int BUTTON_PIN = 7;
        const unsigned int LED_PIN    = 13;
        void setup() {
          pinMode(LED_PIN, OUTPUT);
5         pinMode(BUTTON_PIN, INPUT);
        }

        int old_button_state = LOW;
        int led_state = LOW;
10
        void loop() {
          const int CURRENT_BUTTON_STATE = digitalRead(BUTTON_PIN)
          if (CURRENT_BUTTON_STATE != old_button_state &&          ;
              CURRENT_BUTTON_STATE == HIGH)
15        {
            if (led_state == LOW)
              led_state = HIGH;
             else
              led_state = LOW;
20          digitalWrite(LED_PIN, led_state);
            delay(50);
          }
          old_button_state = CURRENT_BUTTON_STATE;
        }
```

Commonwealth of Australia
Copyright Act 1968

Notice for paragraph 135ZXA (a) of the *Copyright Act 1968*

Warning

This material has been reproduced and communicated to you by or on behalf of Swinburne University of Technology under Part VB of the *Copyright Act 1968* (the ***Act***).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.