Arduino Workshop

(ESP32 Intermediate Level)

ดร. วาธิส ลีลาภัทร

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยขอนแก่น

Agenda

Content

- Serial Read
- Analog mapping
- Pulse-width modulation
- Timer
- Bluetooth communication
- I^2C LCD display
- Arduino IDE plotter

Prerequisite:

1. Basic C/C++ programming language (Arduino IDE)

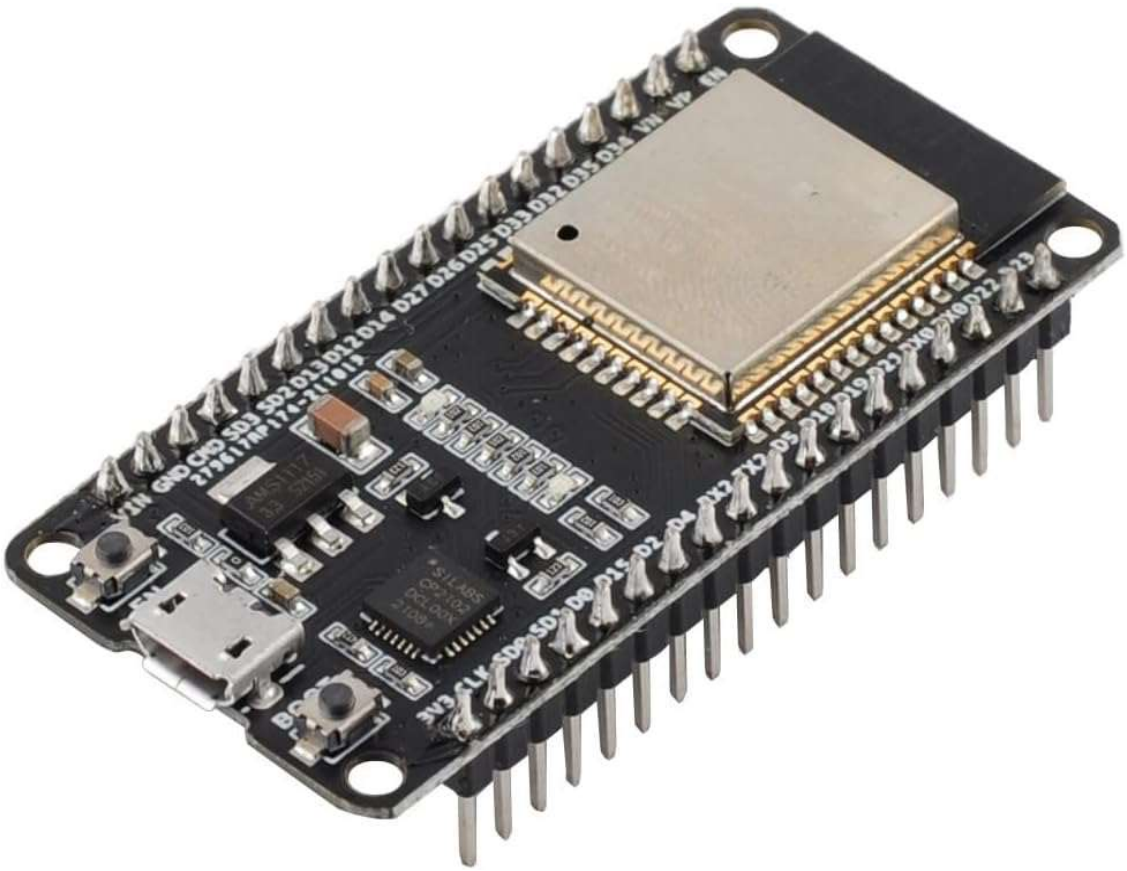
2. Basic ESP32 course

3. Electronics (not required, but will be advantageous)

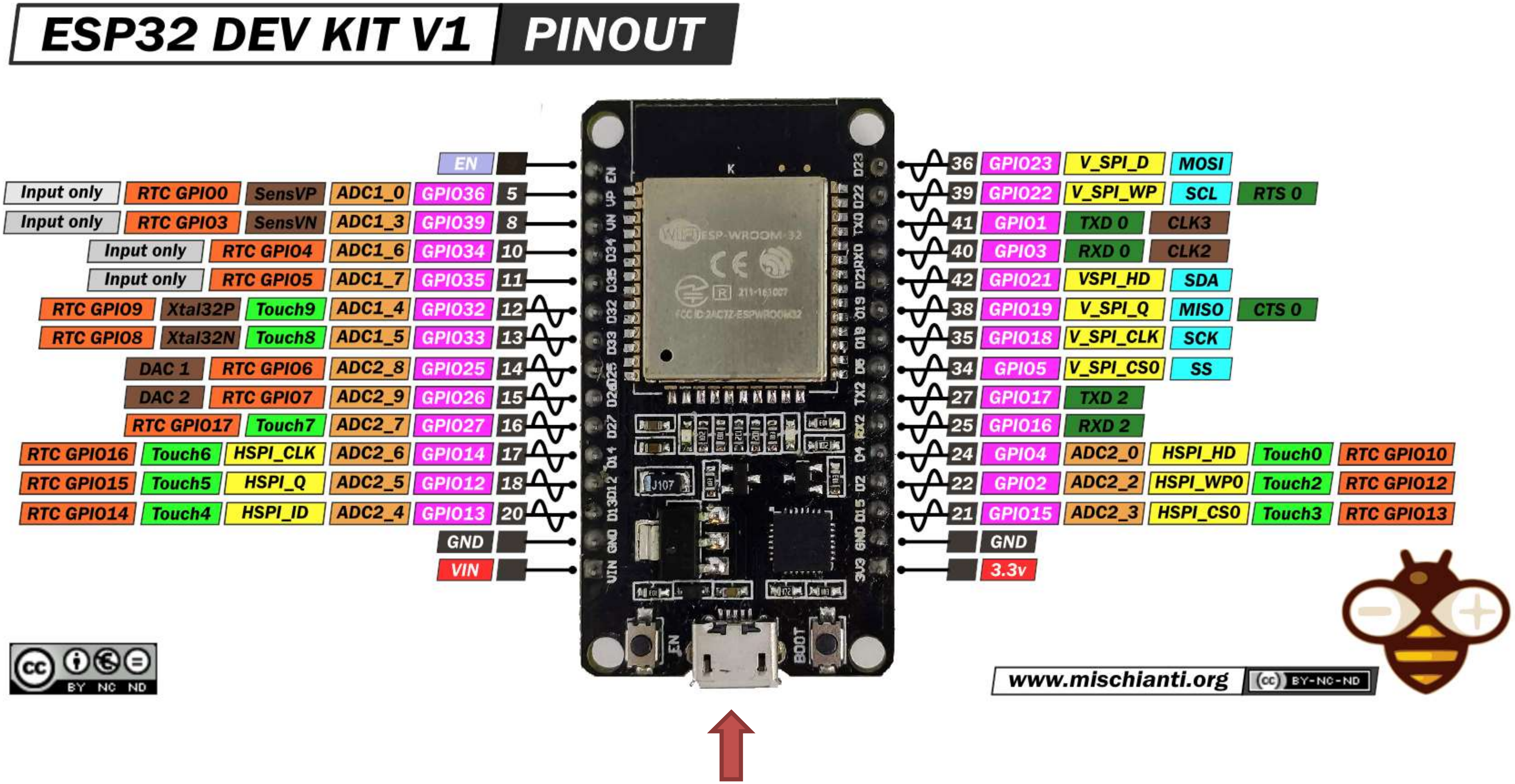
ESP32 DEVKIT

ESP-WROOM-32 DevKit

- utilizes ESP32 chip, 240MHz, RAM 512KB, 4MB flash memory
- WiFi 2.4GHz / Bluetooth



ESP32 DEVKIT Pinout


ESP32 DEV KIT V1 PINOUT
EN
Input only | RTC GPIO0 | SensVP | ADC1_0 | GPIO36 | 5
Input only | RTC GPIO3 | SensVN | ADC1_3 | GPIO39 | 8
Input only | RTC GPIO4 | ADC1_6 | GPIO34 | 10
Input only | RTC GPIO5 | ADC1_7 | GPIO35 | 11
RTC GPIO9 | Xtal32P | Touch9 | ADC1_4 | GPIO32 | 12
RTC GPIO8 | Xtal32N | Touch8 | ADC1_5 | GPIO33 | 13
DAC 1 | RTC GPIO6 | ADC2_8 | GPIO25 | 14
DAC 2 | RTC GPIO7 | ADC2_9 | GPIO26 | 15
RTC GPIO17 | Touch7 | ADC2_7 | GPIO27 | 16
RTC GPIO16 | Touch6 | HSPI_CLK | ADC2_6 | GPIO14 | 17
RTC GPIO15 | Touch5 | HSPI_Q | ADC2_5 | GPIO12 | 18
RTC GPIO14 | Touch4 | HSPI_ID | ADC2_4 | GPIO13 | 20
GND
VIN
36 | GPIO23 | V_SPI_D | MOSI
39 | GPIO22 | V_SPI_WP | SCL | RTS 0
41 | GPIO1 | TXD 0 | CLK3
40 | GPIO3 | RXD 0 | CLK2
42 | GPIO21 | VSPI_HD | SDA
38 | GPIO19 | V_SPI_Q | MISO | CTS 0
35 | GPIO18 | V_SPI_CLK | SCK
34 | GPIO5 | V_SPI_CS0 | SS
27 | GPIO17 | TXD 2
25 | GPIO16 | RXD 2
24 | GPIO4 | ADC2_0 | HSPI_HD | Touch0 | RTC GPIO10
22 | GPIO2 | ADC2_2 | HSPI_WP0 | Touch2 | RTC GPIO12
21 | GPIO15 | ADC2_3 | HSPI_CS0 | Touch3 | RTC GPIO13
GND
3.3v
EN
BOOT
www.mischianti.org
BY-NC-ND

Micro USB Port

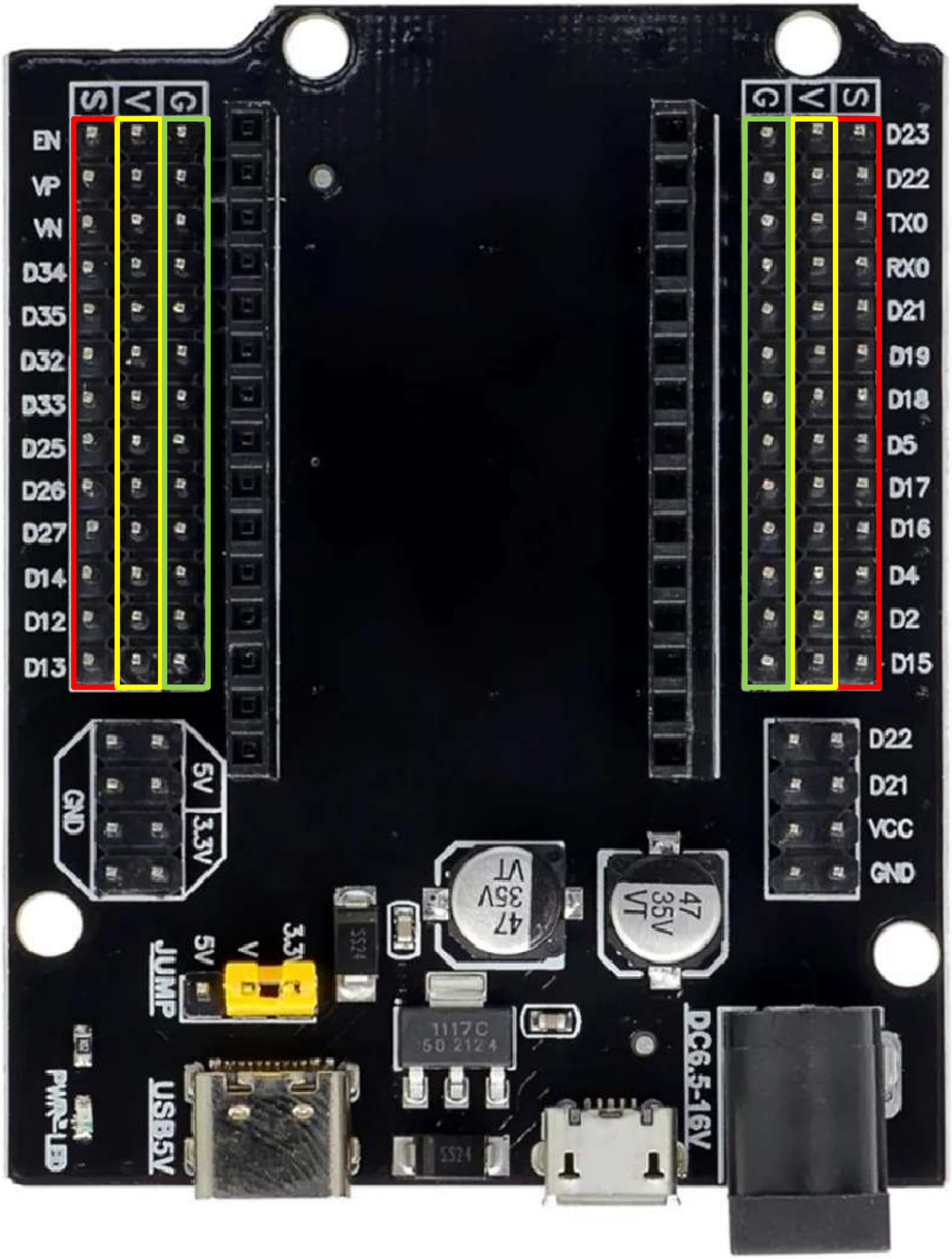
ESP32 Extension Board

- Pins extension
- External power supply

S (Signal) Pins

V (3V Power supply) Pins

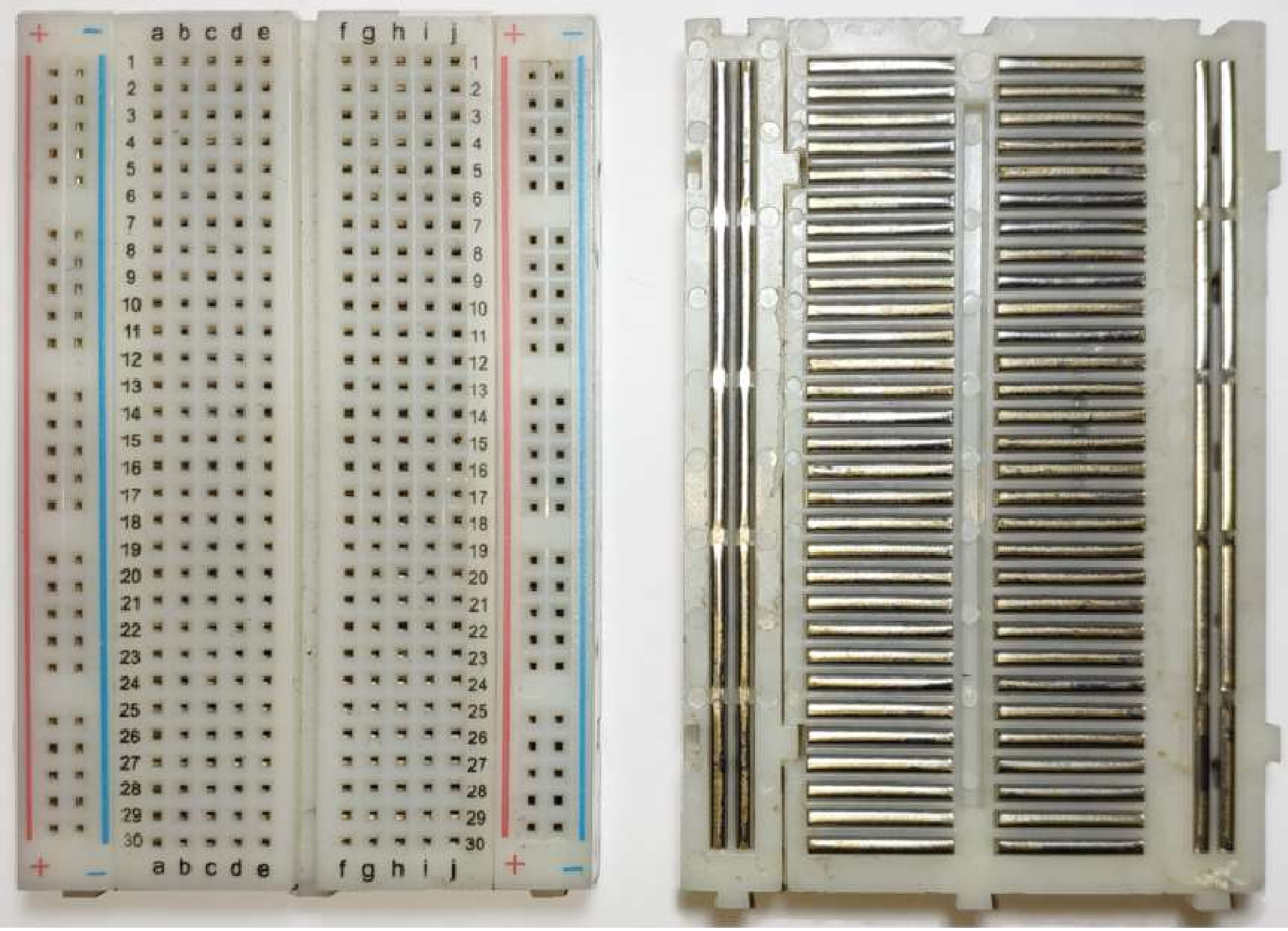
G (Ground) Pins



Circuit Construction

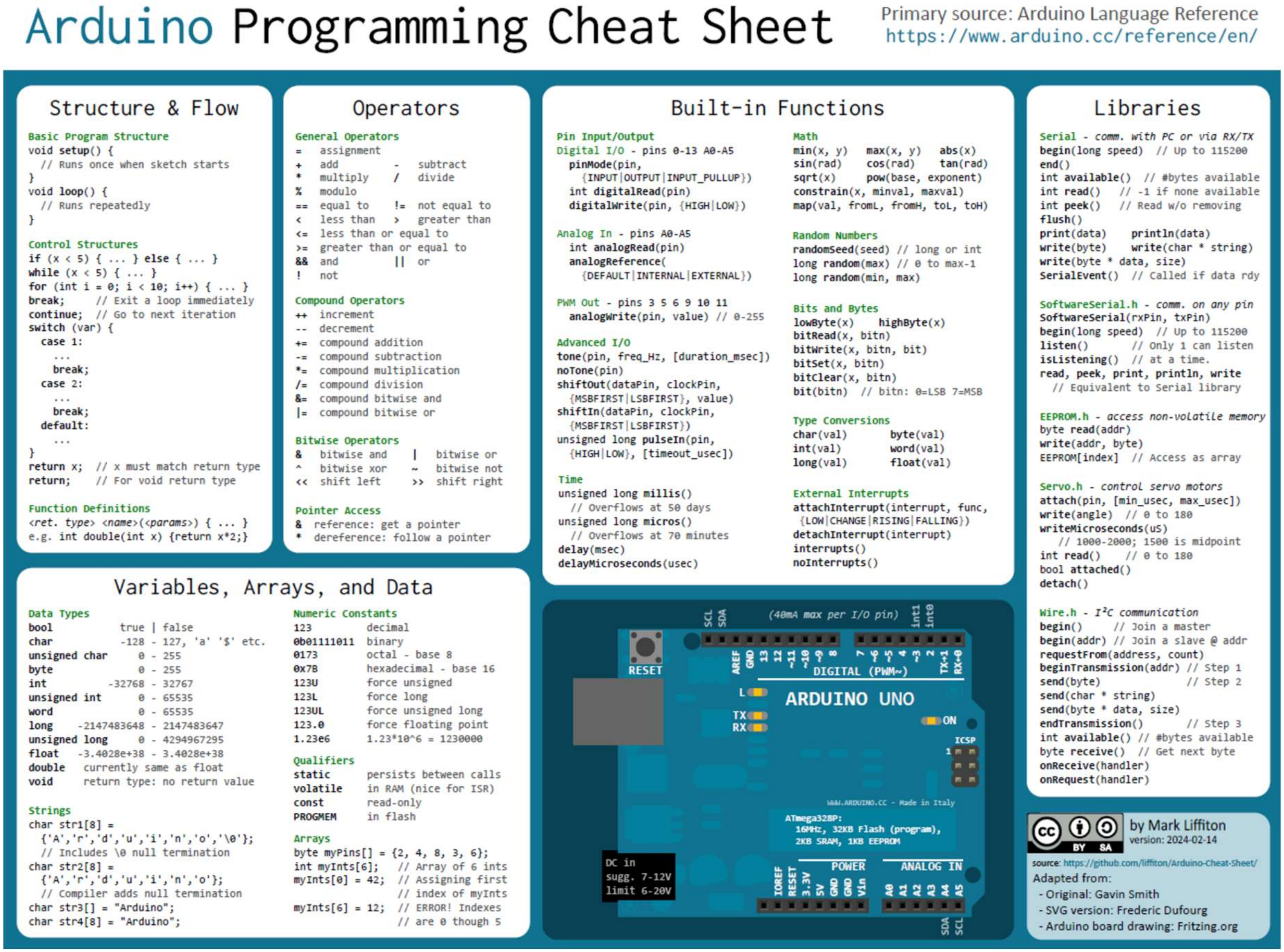
Prototyping board

- Pre-connected conductor stripes
- Horizontal stripes on 2 top rows and 2 bottom rows
- Vertical stripes in the middle



Arduino Reference Sheet

https://kku.world/e664b0



Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
https://www.arduino.cc/reference/en/

Structure & Flow

Basic Program Structure

```
void setup() {
  // Runs once when sketch starts
}
void loop() {
  // Runs repeatedly
}
```

Control Structures

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
for (int i = 0; i < 10; i++) { ... }
break;     // Exit a loop immediately
continue;  // Go to next iteration
switch (var) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
return x;  // x must match return type
return;    // For void return type
```

Function Definitions

```
<ret. type> <name>(<params>) { ... }
e.g. int double(int x) {return x*2;}
```

Operators

General Operators

=	assignment		
+	add	-	subtract
*	multiply	/	divide
%	modulo		
==	equal to	!=	not equal to
<	less than	>	greater than
<=	less than or equal to		
>=	greater than or equal to		
&&	and	||	or
!	not		

Compound Operators

++	increment
--	decrement
+=	compound addition
-=	compound subtraction
*=	compound multiplication
/=	compound division
&=	compound bitwise and
|=	compound bitwise or

Bitwise Operators

&	bitwise and	|	bitwise or
^	bitwise xor	~	bitwise not
<<	shift left	>>	shift right

Pointer Access

&	reference: get a pointer
*	dereference: follow a pointer

Variables, Arrays, and Data

Data Types

bool	true | false
char	-128 - 127, 'a' '$' etc.
unsigned char	0 - 255
byte	0 - 255
int	-32768 - 32767
unsigned int	0 - 65535
word	0 - 65535
long	-2147483648 - 2147483647
unsigned long	0 - 4294967295
float	-3.4028e+38 - 3.4028e+38
double	currently same as float
void	return type: no return value

Strings

```
char str1[8] =
  {'A','r','d','u','i','n','o','\0'};
  // Includes \0 null termination
char str2[8] =
  {'A','r','d','u','i','n','o'};
  // Compiler adds null termination
char str3[] = "Arduino";
char str4[8] = "Arduino";
```

Numeric Constants

123	decimal
0b01111011	binary
0173	octal - base 8
0x7B	hexadecimal - base 16
123U	force unsigned
123L	force long
123UL	force unsigned long
123.0	force floating point
1.23e6	1.23*10^6 = 1230000

Qualifiers

static	persists between calls
volatile	in RAM (nice for ISR)
const	read-only
PROGMEM	in flash

Arrays

```
byte myPins[] = {2, 4, 8, 3, 6};
int myInts[6];    // Array of 6 ints
myInts[0] = 42;   // Assigning first
                  // index of myInts
myInts[6] = 12;   // ERROR! Indexes
                  // are 0 though 5
```

Built-in Functions

Pin Input/Output

Digital I/O - pins 0-13 A0-A5

```
  pinMode(pin,
    {INPUT|OUTPUT|INPUT_PULLUP})
  int digitalRead(pin)
  digitalWrite(pin, {HIGH|LOW})
```

Analog In - pins A0-A5

```
  int analogRead(pin)
  analogReference(
    {DEFAULT|INTERNAL|EXTERNAL})
```

PWM Out - pins 3 5 6 9 10 11

```
  analogWrite(pin, value) // 0-255
```

Advanced I/O

```
tone(pin, freq_Hz, [duration_msec])
noTone(pin)
shiftOut(dataPin, clockPin,
  {MSBFIRST|LSBFIRST}, value)
shiftIn(dataPin, clockPin,
  {MSBFIRST|LSBFIRST})
unsigned long pulseIn(pin,
  {HIGH|LOW}, [timeout_usec])
```

Time

```
unsigned long millis()
  // Overflows at 50 days
unsigned long micros()
  // Overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)
```

Math

```
min(x, y)   max(x, y)   abs(x)
sin(rad)    cos(rad)    tan(rad)
sqrt(x)     pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
```

Random Numbers

```
randomSeed(seed) // long or int
long random(max) // 0 to max-1
long random(min, max)
```

Bits and Bytes

```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Type Conversions

```
char(val)       byte(val)
int(val)        word(val)
long(val)       float(val)
```

External Interrupts

```
attachInterrupt(interrupt, func,
 {LOW|CHANGE|RISING|FALLING})
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```

Libraries

Serial - *comm. with PC or via RX/TX*

```
begin(long speed)  // Up to 115200
end()
int available()  // #bytes available
int read()   // -1 if none available
int peek()   // Read w/o removing
flush()
print(data)     println(data)
write(byte)     write(char * string)
write(byte * data, size)
SerialEvent()  // Called if data rdy
```

SoftwareSerial.h - *comm. on any pin*

```
SoftwareSerial(rxPin, txPin)
begin(long speed)  // Up to 115200
listen()        // Only 1 can listen
isListening()  // at a time.
read, peek, print, println, write
  // Equivalent to Serial library
```

EEPROM.h - *access non-volatile memory*

```
byte read(addr)
write(addr, byte)
EEPROM[index]  // Access as array
```

Servo.h - *control servo motors*

```
attach(pin, [min_usec, max_usec])
write(angle)  // 0 to 180
writeMicroseconds(uS)
   // 1000-2000; 1500 is midpoint
int read()    // 0 to 180
bool attached()
detach()
```

Wire.h - *I^2C communication*

```
begin()      // Join a master
begin(addr) // Join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(byte)                // Step 2
send(char * string)
send(byte * data, size)
endTransmission()         // Step 3
int available() // #bytes available
byte receive()  // Get next byte
onReceive(handler)
onRequest(handler)
```

by Mark Liffiton
version: 2024-02-14

source: https://github.com/liffiton/Arduino-Cheat-Sheet/

Adapted from:
- Original: Gavin Smith
- SVG version: Frederic Dufourg
- Arduino board drawing: Fritzing.org

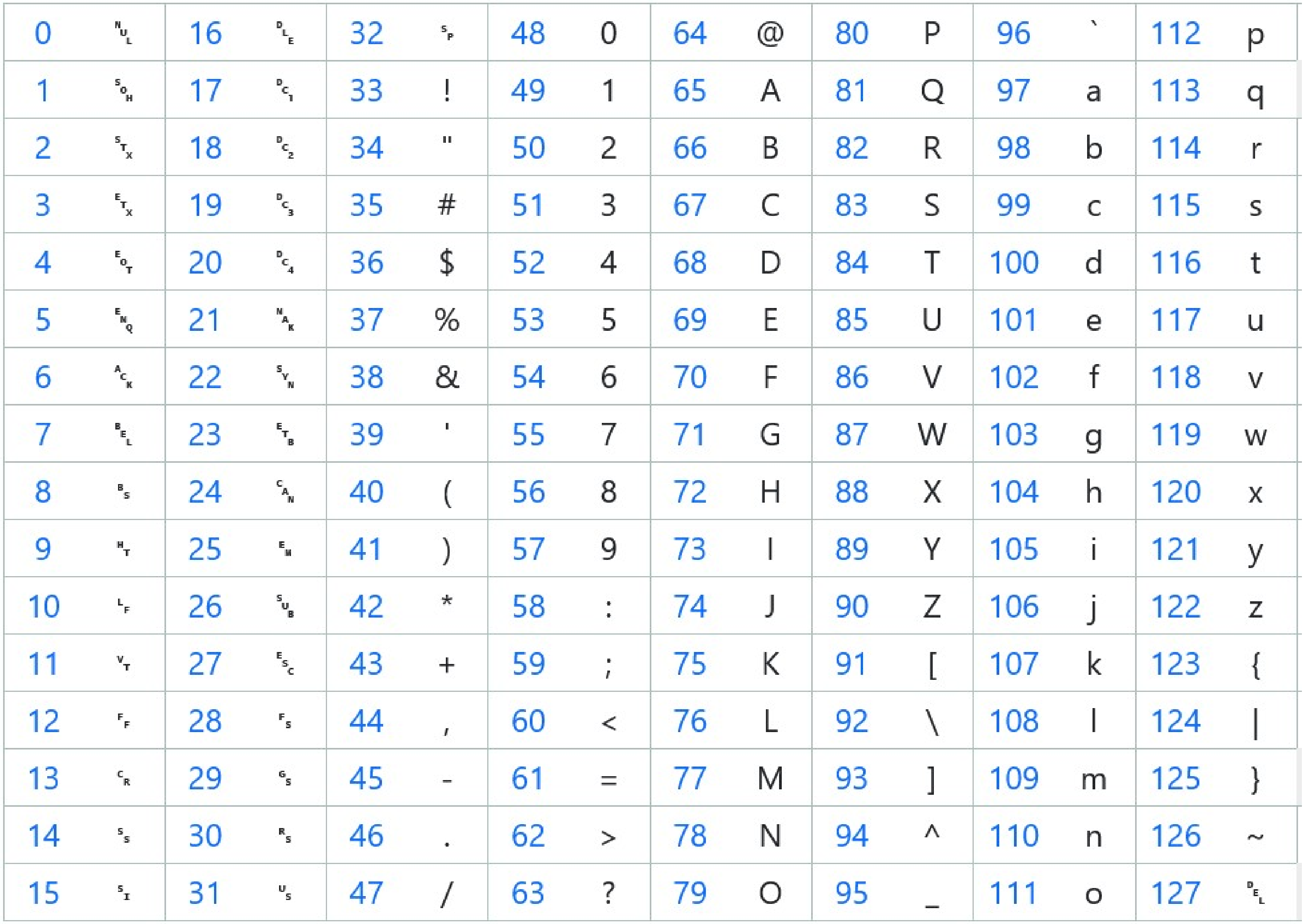
ESP32 Serial Communication

- ESP32 can use PC screen and keyboard
- **Serial.print()** is used for sending message to PC screen
- Open sketch: **serial_read_basic.ino**

- **Serial.available() → use to check if there is a data sent from PC**
- **Serial.read() is used for receiving 1 character from PC keyboard**
- **This function returns ASCII code of a character**
- For example: 'A' = 65, 'B' = 66, '$' = 36, 'a' = 97, 'b' = 98
- Use (char) inside **Serial.print()** to convert ASCII code to character

ESP32 Serial Communication

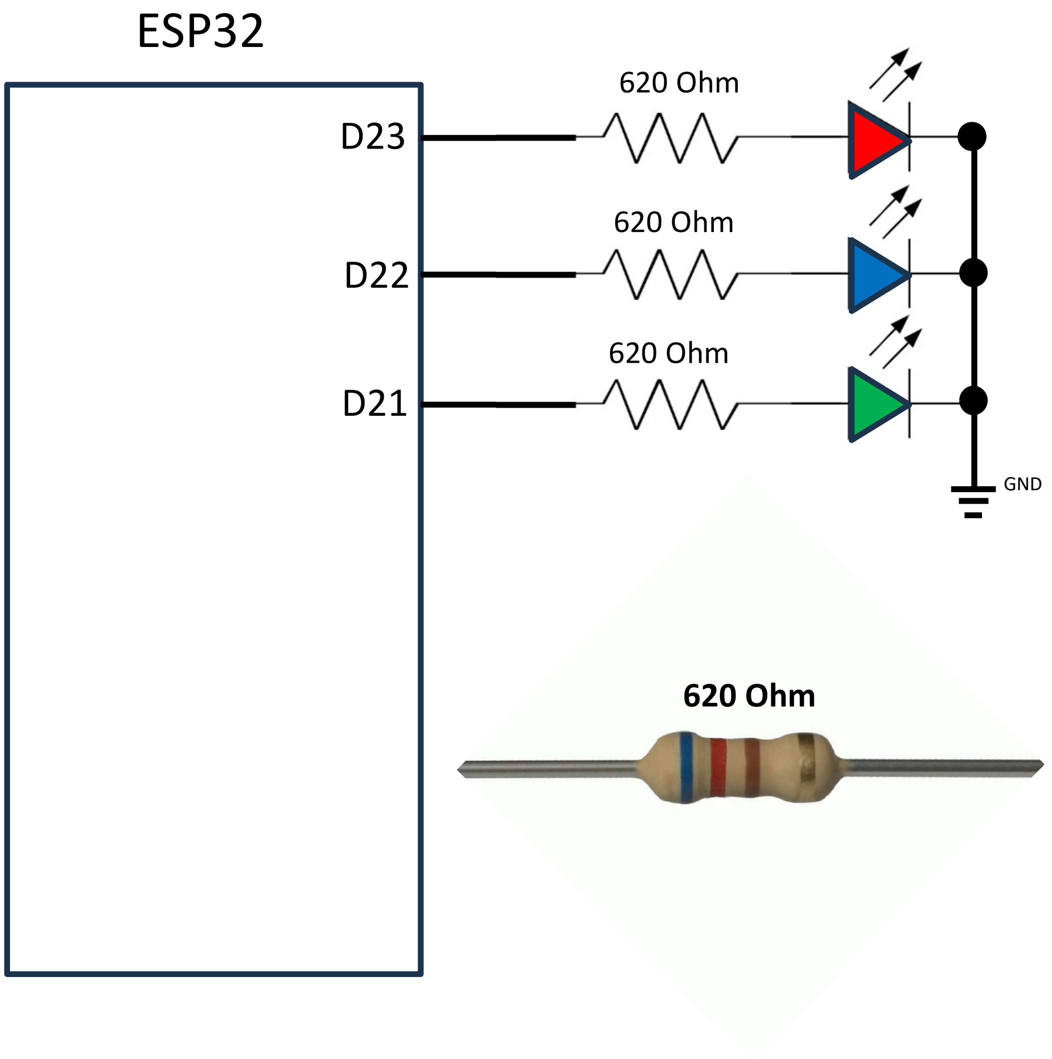
ASCII code table

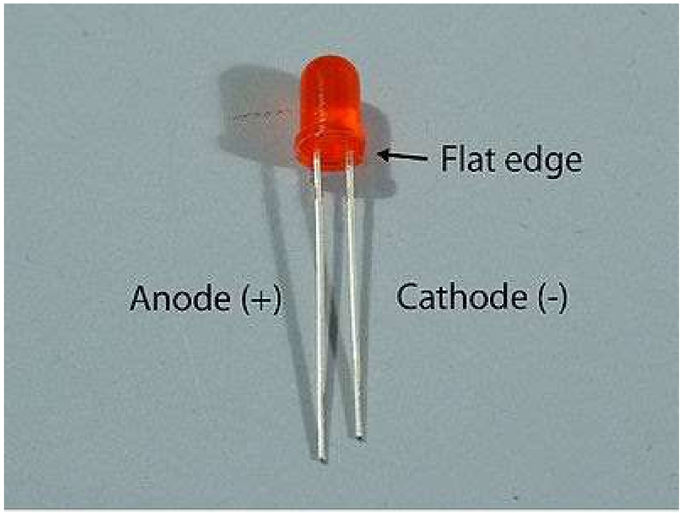


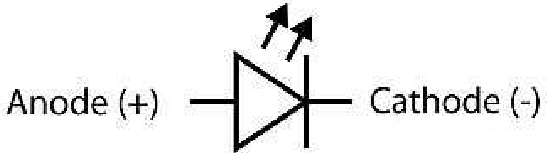
0	NUL	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

ESP32 Serial Communication

- Application: Control LEDs from PC keyboard
- Open sketch: **serial_read_LEDs.ino**

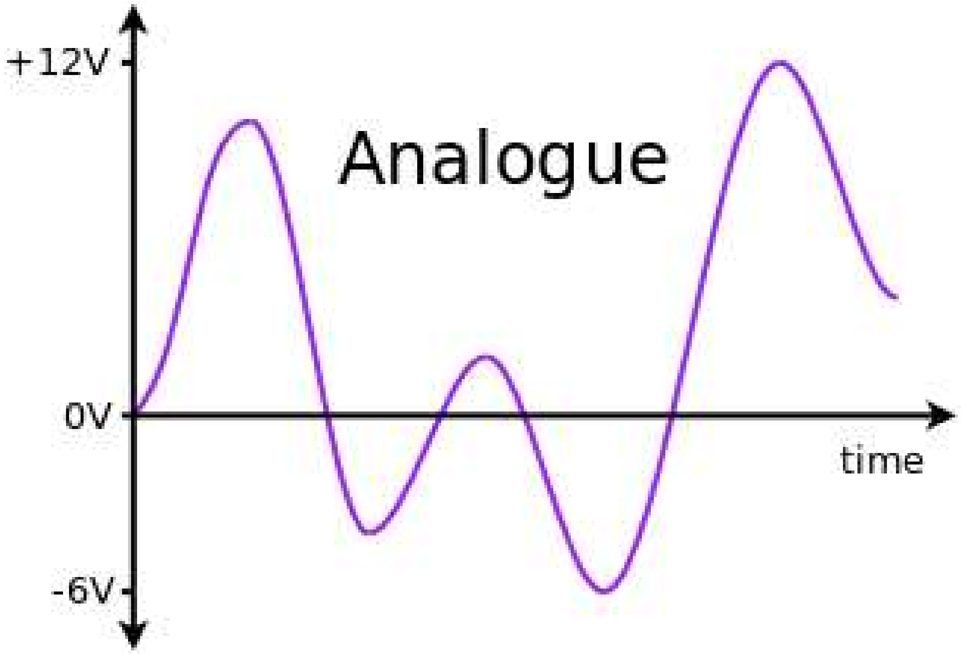


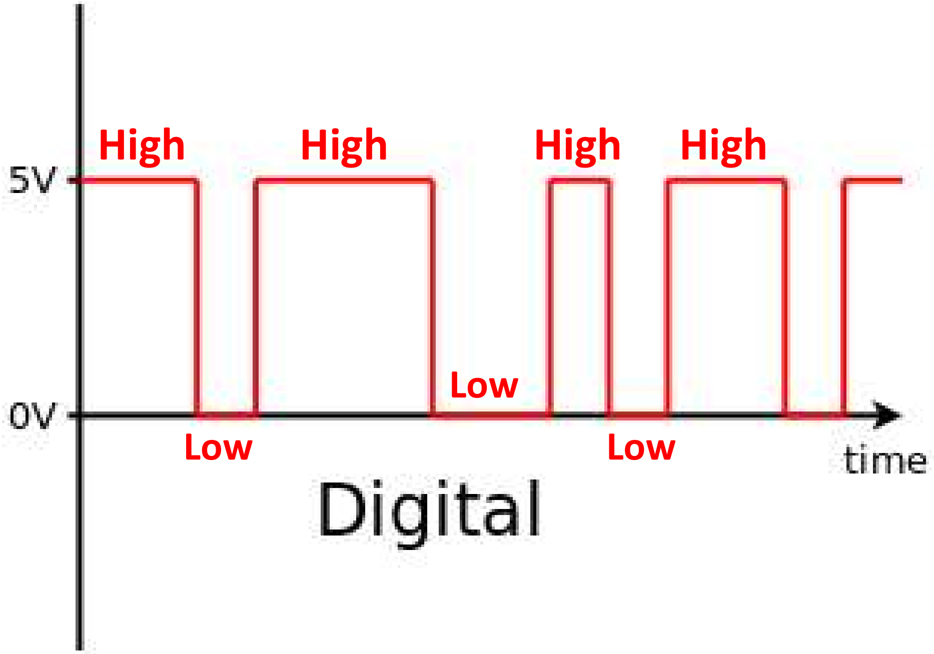




Analog Input

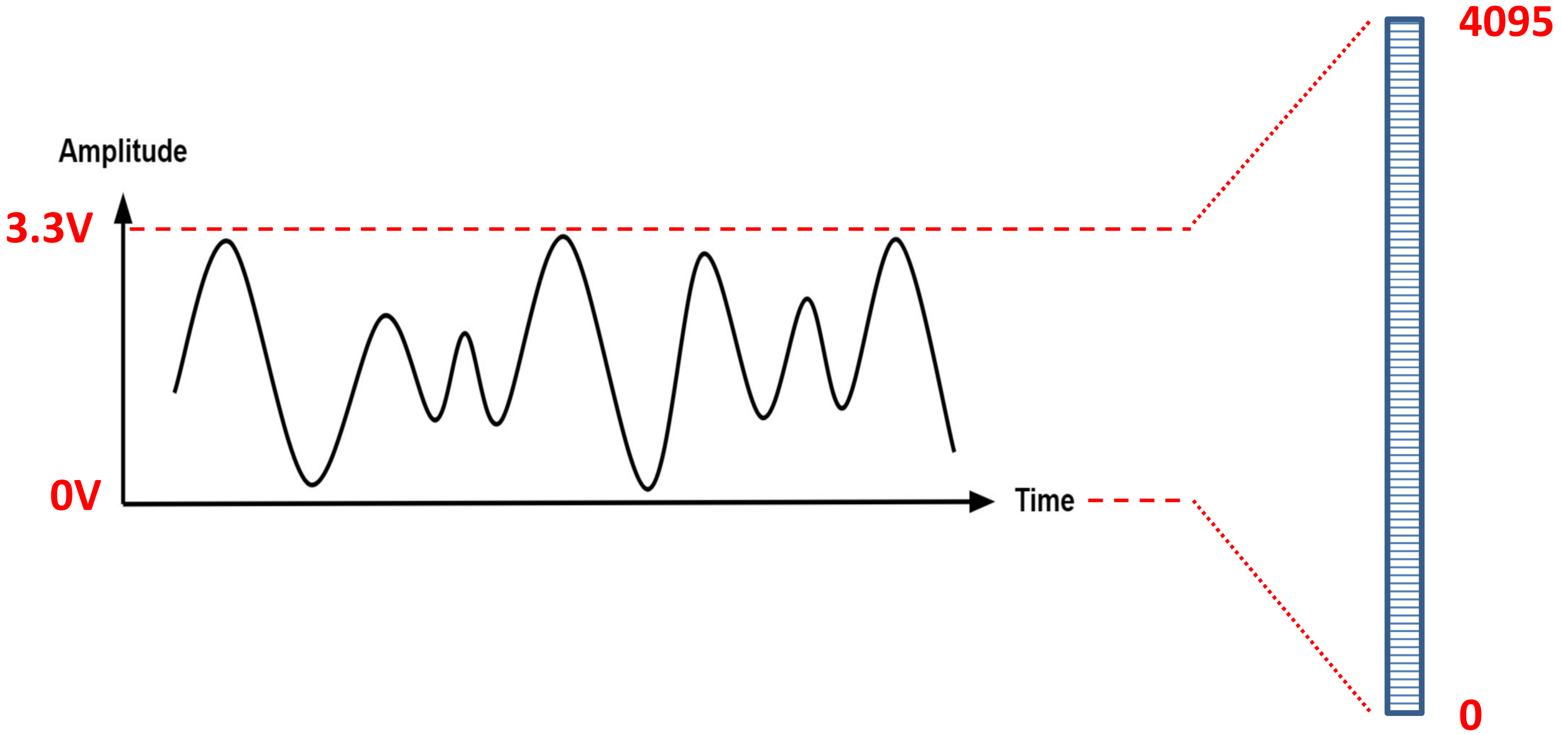
- Analog signal is continuous and can be any value
- Digital signal is limited to two possible value 0 and 1 (low and high)
- Use ***AnalogRead( )*** to get analog value





Analog Input (revisited)

- Use ***AnalogRead( )*** to get analog value
- Analog signal is continuous and can be any value
- Digital signal is limited to two possible value 0 and 1 (low and high)



- Max analog voltage is 3.3V and converted to 4095
- Formula: *Analog value = 4095*(X/3.3)*
- <u>Ex</u> if analog input = 2.5V, *analogRead()* will return value of 4095*(2.5/3.3) = 3102

Analog Input

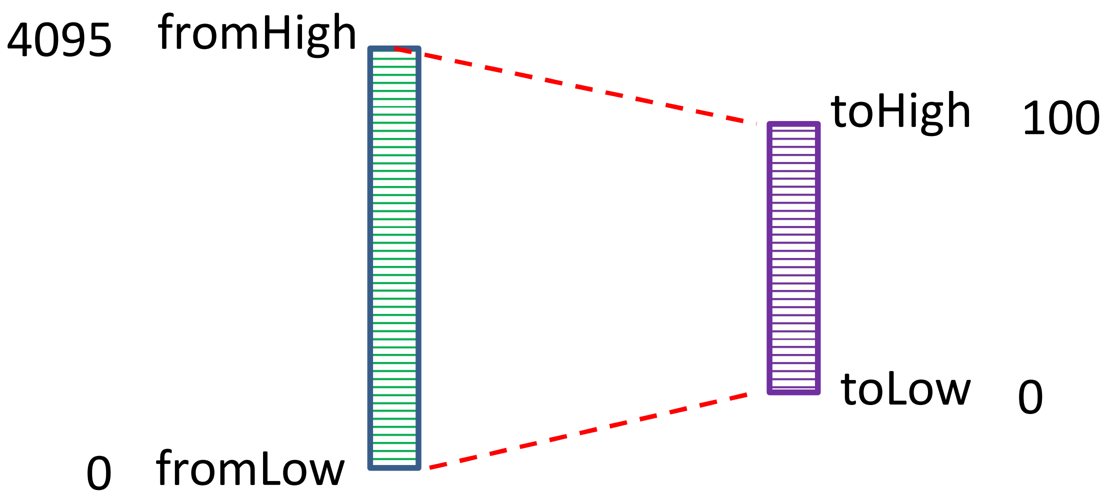
- ***AnalogRead( )*** always gives value in range 0-4095
- Values in this range can be mapped to different ranges
- For example, 0-100 (for percentage calculation)
- Use map() to convert analog read value:

Syntax:

NewValue = **map**(Value, fromLow, fromHigh, toLow, toHigh);

Example

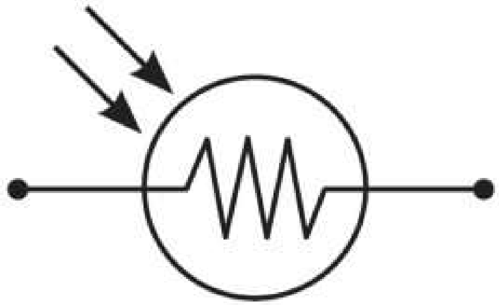
NewRange = map(val, 0, 4095, 0, 100);

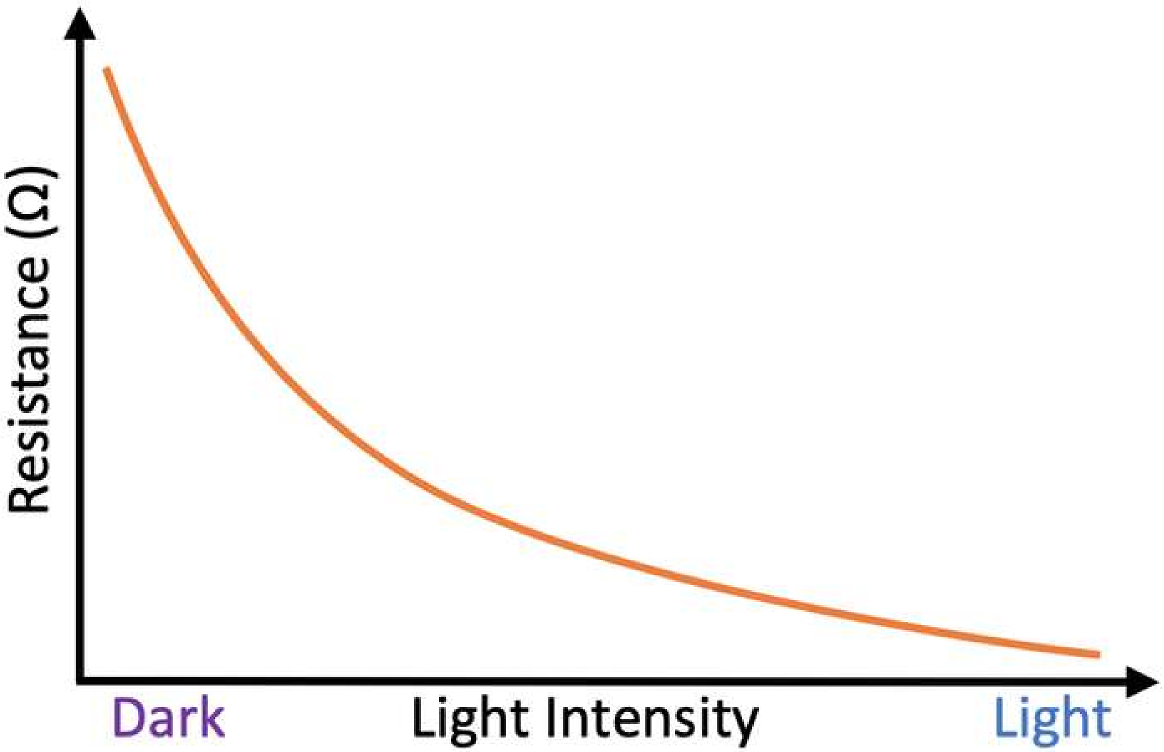


Analog Sensor

Light Dependent Resistor (LDR)

- LDR is a type of resistor whose resistance varies with light intensity
- Dark → high resistance, bright → low resistance



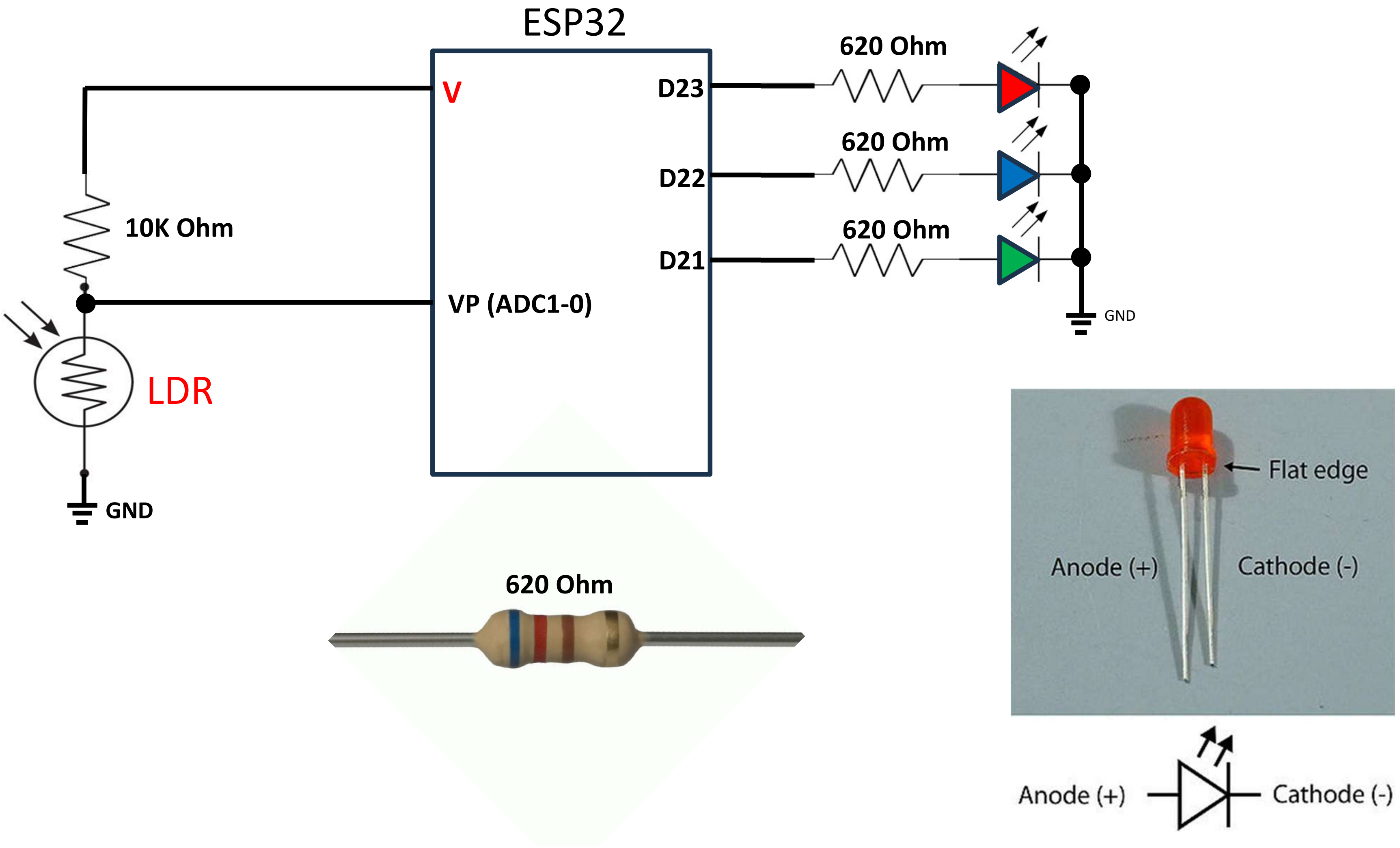


Analog Sensor

Light Dependent Resistor (LDR)

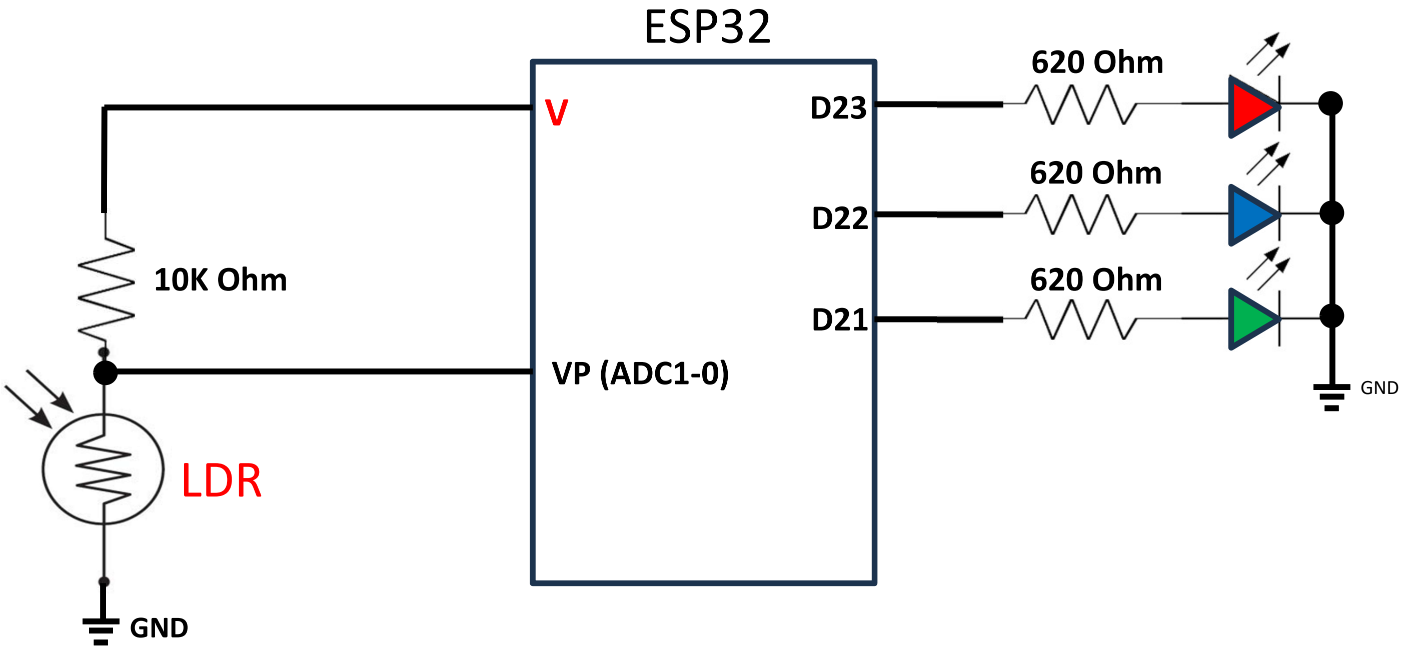
- LDR is a type of resistor whose resistance varies with light intensity
- Dark → high resistance, bright → low resistance

Open sketch: **LDR_Test.ino**



Analog Sensor

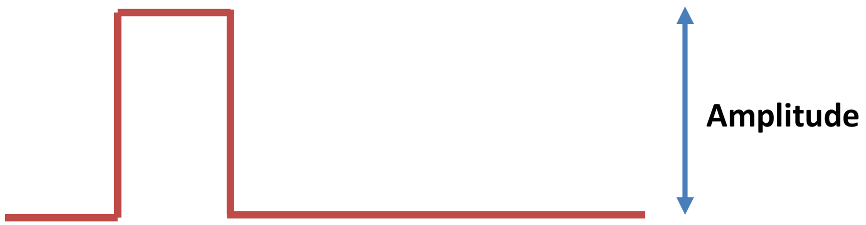
- Examine raw analog value obtained from ***analogRead()***
- What is the minimum and maximum value?
- **Exercise**
  - Modify sketch to show light intensity in percentage
  - 0% = darkest and 100% = brightest
  - Further modification: turn on LEDs according to darkness
  - Turn 3 LEDs on → 30% brightness, 2 LEDs → 60%, 1 LEDs → 90%
  - Turn off all LEDs when brightness is greater than 90%



Pulse Width Modulation (PWM)

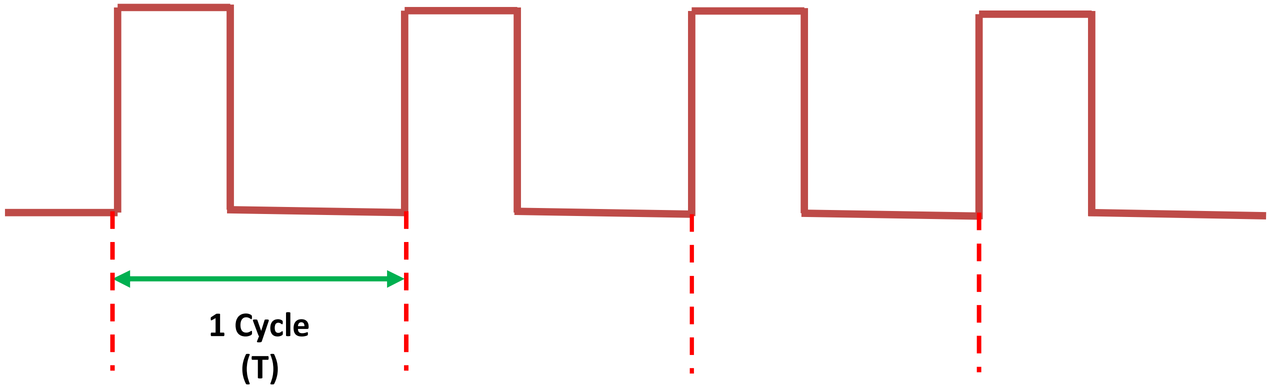
- Pulse is a rapid, transient change in the amplitude of a signal

Pulse

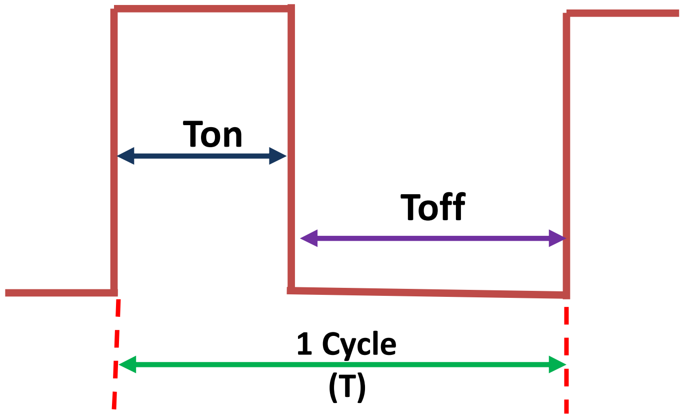


Amplitude

Pulse train



1 Cycle
(T)



Ton

Toff

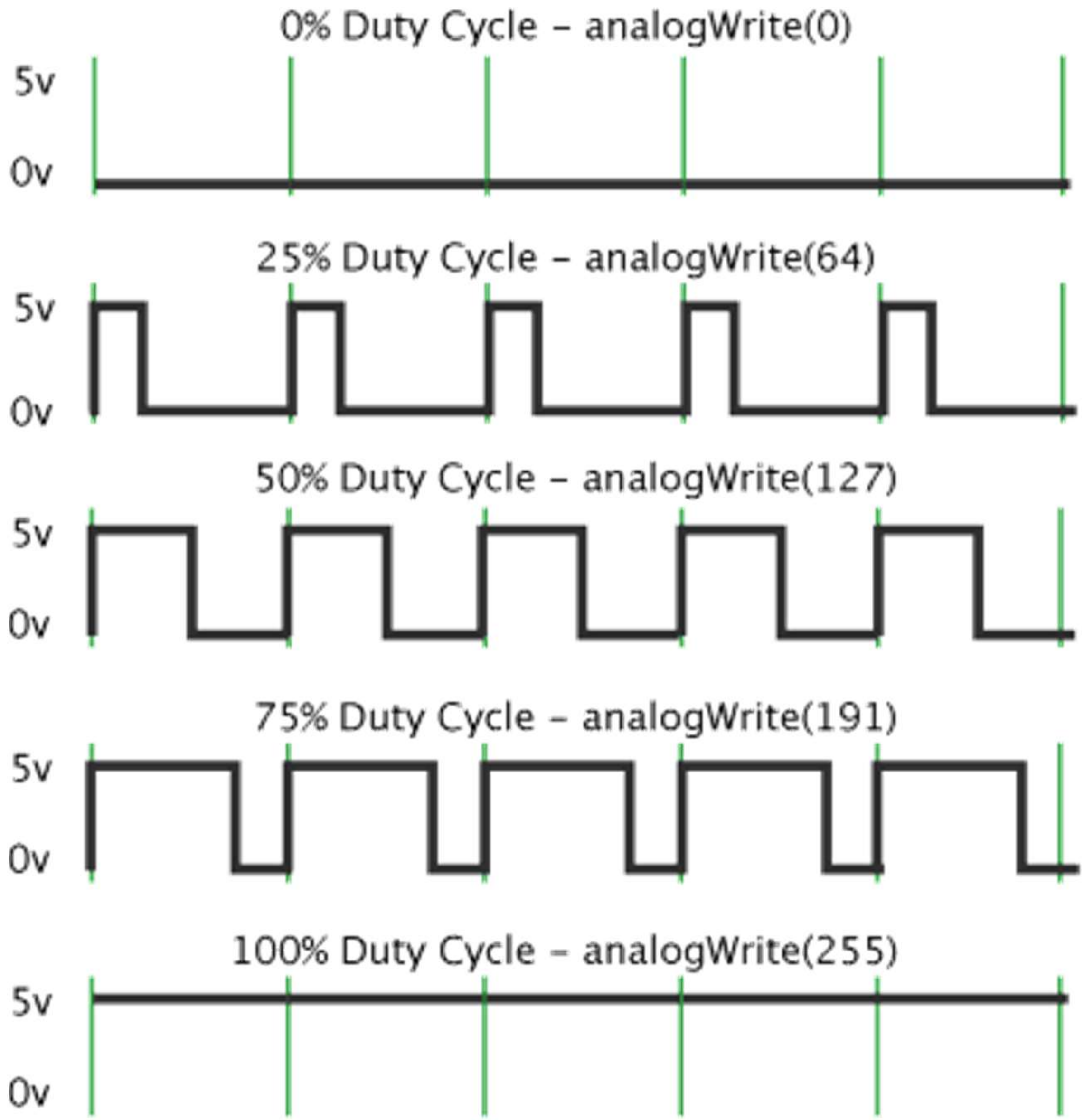
1 Cycle
(T)

(Period) T = Ton + Toff

Duty cycle (%) = (Ton / T)*100%

Pulse Width Modulation (PWM)

- PWM is used as a technique to vary power delivered to load

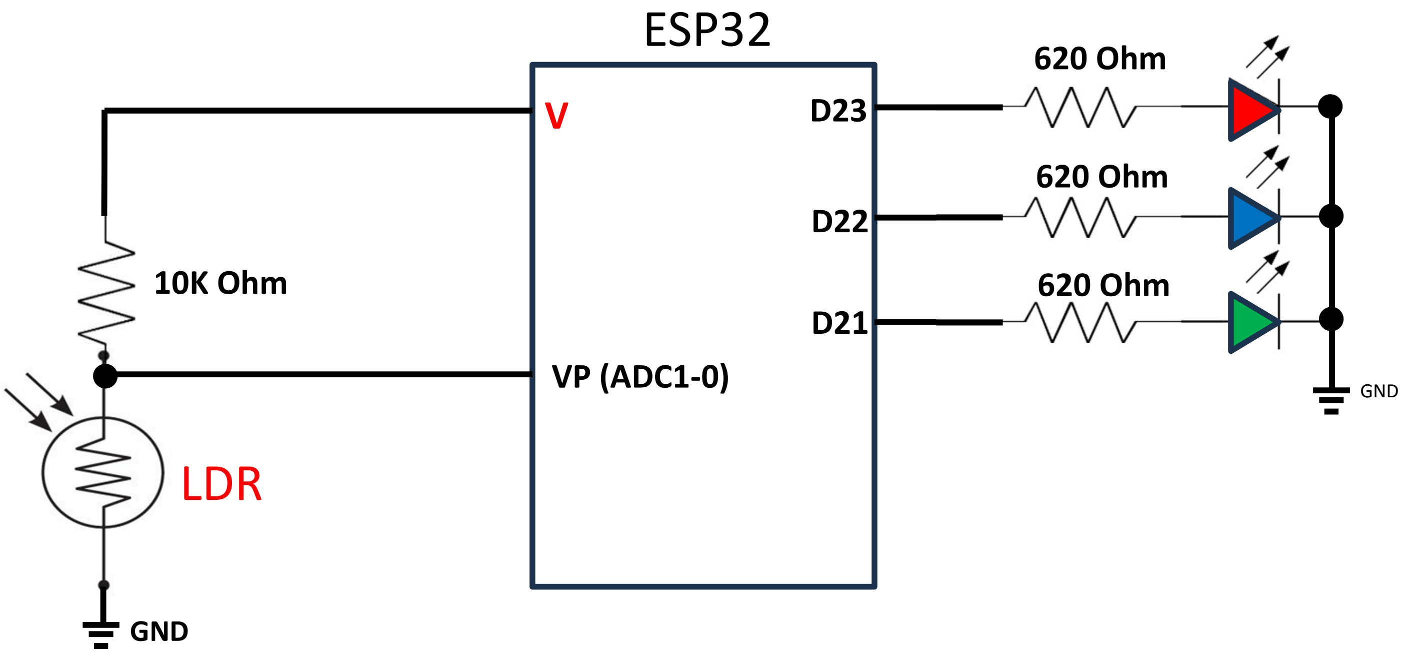


Load is OFF

Load is fully ON

Pulse Width Modulation (PWM)

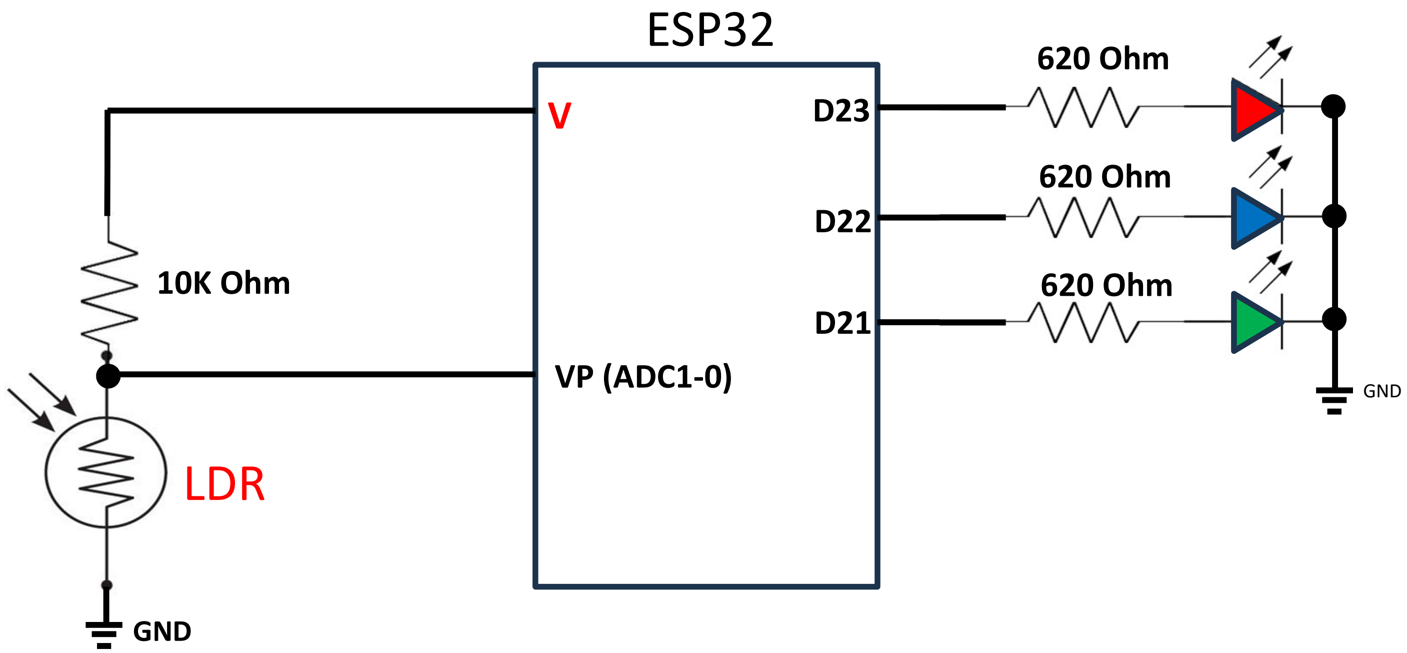
- Use **analogWrite()** function to control PWM
- Syntax: **analogWrite(*pin*, *duty_cycle*)**
- Open sketch: **PWM_LED.ino**



Pulse Width Modulation (PWM)

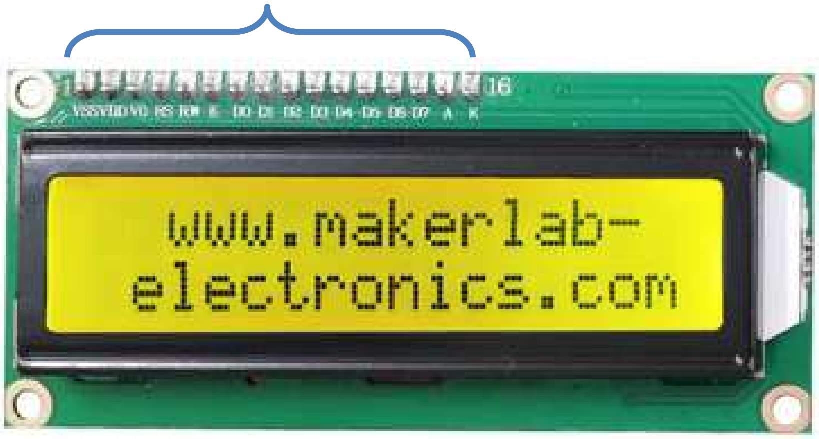
- A common application is to control brightness of LEDs
- **Exercise**

  Write a program that dims LEDs when the LDR is in high brightness and brighten LEDs when the LDR is in dark



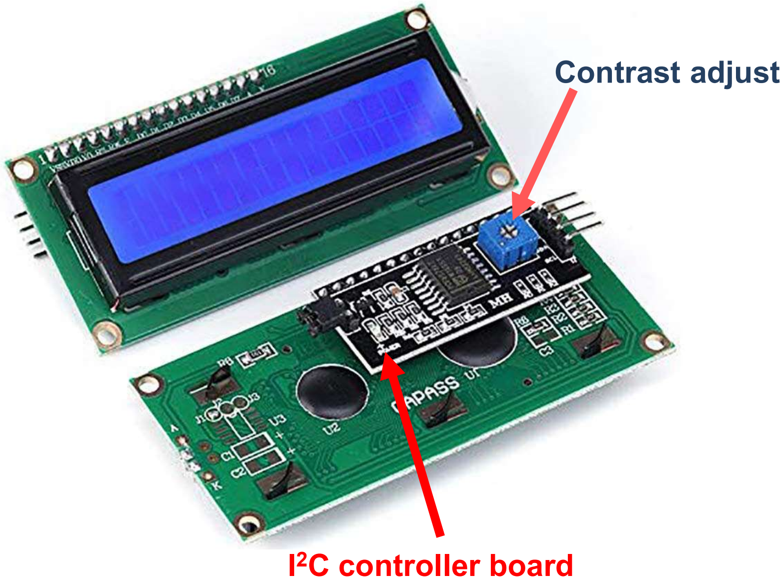
Alphanumeric LCD

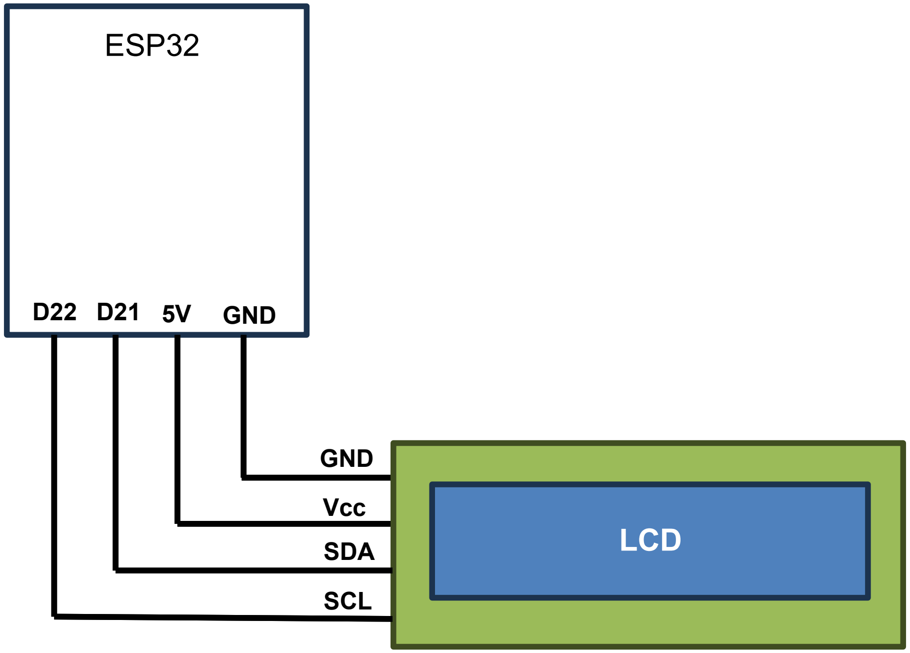
- Liquid Crystal Display (LCD) is an electronic component used for displaying Texts, numbers and symbols
- Available in several models measured by number of lines and characters
- For example: 1x16, 2x16, 2x20, 4x20, etc.
- Backlight is optional
- Typical LCD module requires 16 data lines for communication



Alphanumeric LCD

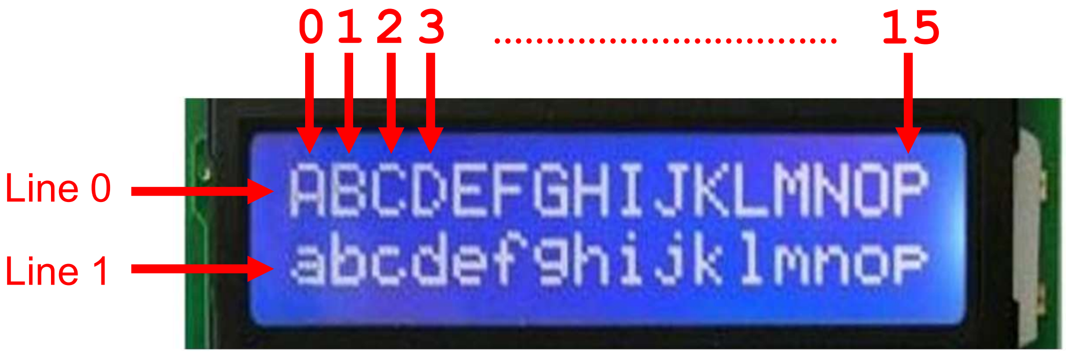
- Inter-Integrated Circuit (IIC or I^{2}C) is used to simplified communication between LCD module and ESP32
- I^{2}C uses only 2 I/O lines for communication





Alphanumeric LCD

- To use I^2C LCD module, install "**LiquidCrystal I2C**" library from provided .zip file
- Run test code: **I2C_LCD_Test.ino**
- This library provides the following LCD display functions:
  - *** lcd object must be created first ***
  - lcd.init() -- initialize LCD module
  - lcd.backlight() -- turn on backlight
  - lcd.noBacklight() -- turn off backlight
  - lcd.print(" ") -- print text message at current cursor position
  - lcd.setCursor(x, y) -- move cursor to column x, line y



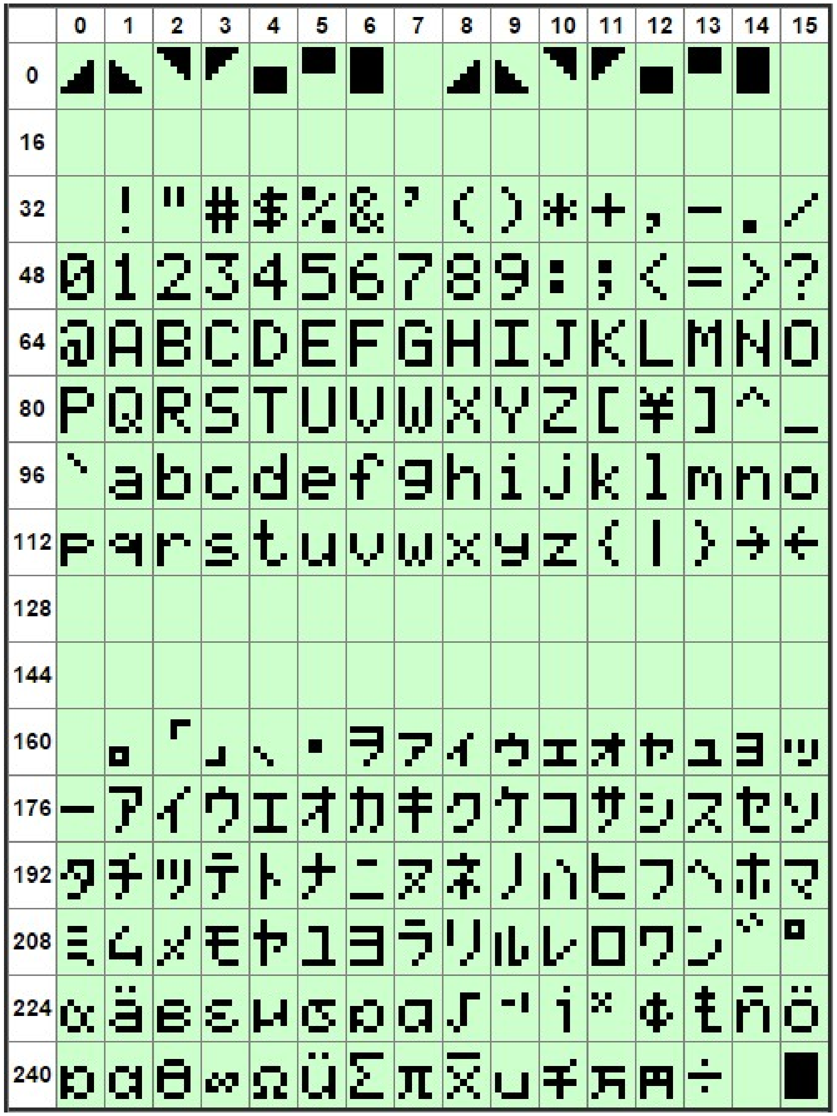
Alphanumeric LCD

- Floating point numbers can be printed with specified number of fraction
- Run test code: **I2C_LCD_Numbers.ino**
- LCD display contain preloaded special characters:

  - To print these characters,
    use *(char) x* inside lcd.print()

  - For example:

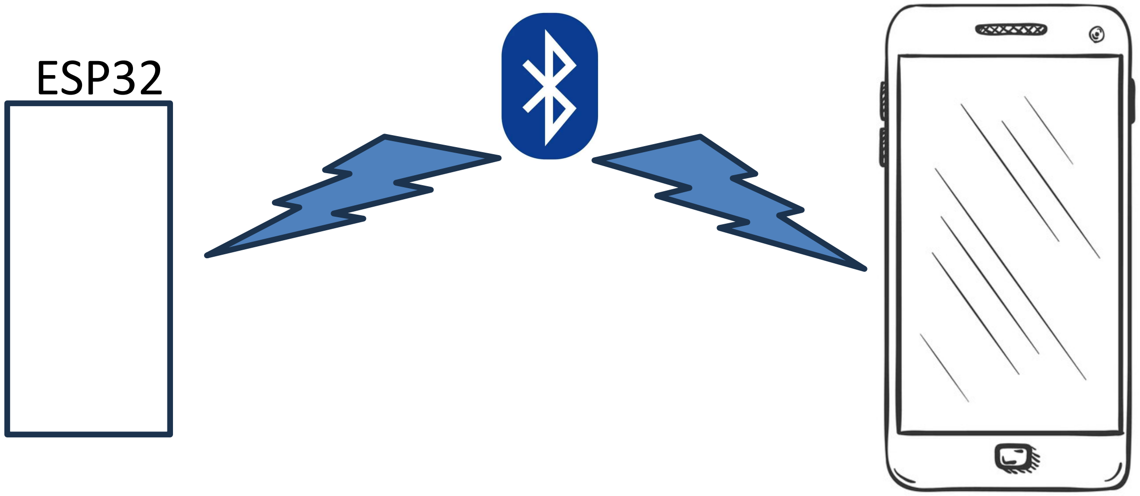
    lcd.print((char) 247);   // print π



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	◢	◣	◥	◤	▄	▀	█		◢	◣	◥	◤	▄	▀	█	
16																
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	¥	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	→	←
128																
144																
160		。	「	」	、	・	ヲ	ァ	ィ	ゥ	ェ	ォ	ャ	ュ	ョ	ッ
176	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
192	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
208	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	゛	゜
224	α	ä	β	ε	μ	σ	ρ	g	√	$^{-1}$	j	^{x}	¢	£	ñ	ö
240	p	q	θ	∞	Ω	ü	Σ	π	x̄	y	千	万	円	÷		█

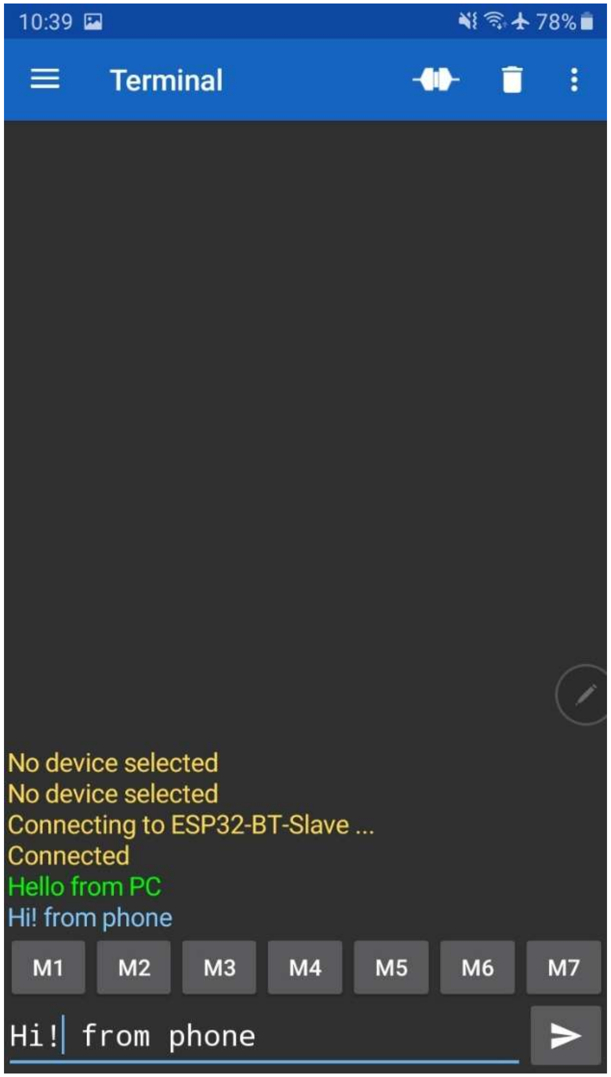
Bluetooth Communication

- ESP32 has built-in Bluetooth classic module for sending/receiving data
- To use ESP32 Bluetooth:
  - Add header file: `#include "BluetoothSerial.h"`
  - Create object: BluetoothSerial SerialBT;
  - Initialize: SerialBT.begin(device_name);
  - To receive data, use .read() method
  - To send data, use .write() method
- Need a paring Bluetooth device



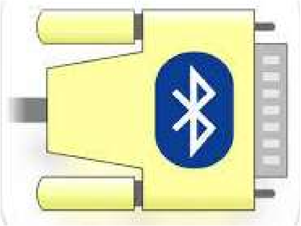
Bluetooth Communication

- On your mobile phone, install **Serial Bluetooth Terminal** App



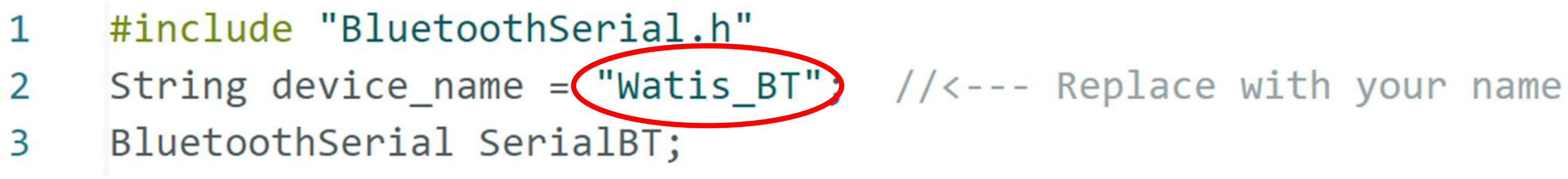
```
No device selected
No device selected
Connecting to ESP32-BT-Slave ...
Connected
Hello from PC
Hi! from phone
```

```
Hi! from phone
```



Bluetooth Communication

- Open sketch: **BT_Test.ino**
- Give your ESP32 Bluetooth a name (Line 2 in the sketch)

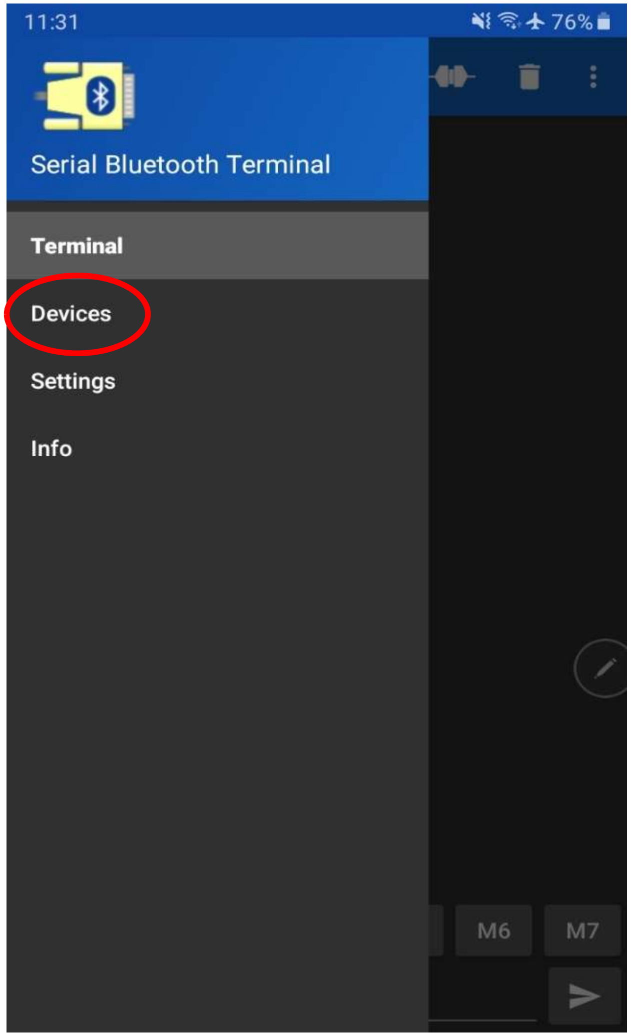


```
1    #include "BluetoothSerial.h"
2    String device_name = "Watis_BT";   //<--- Replace with your name
3    BluetoothSerial SerialBT;
```

- Upload the sketch to ESP32

Bluetooth Communication

- Open Serial Bluetooth Terminal App
- Go to menu ☰
- Choose Devices
- Under Bluetooth Classic tab, look for your ESP32 Bluetooth device



11:31 76%

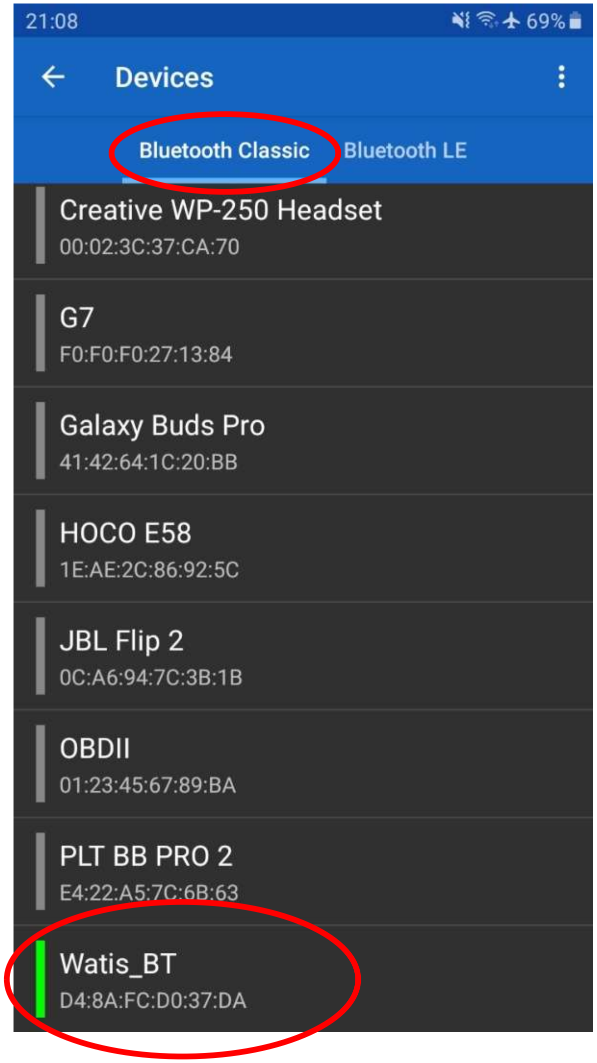
Serial Bluetooth Terminal

Terminal

Devices

Settings

Info



21:08 69%

Devices

Bluetooth Classic | Bluetooth LE

Creative WP-250 Headset
00:02:3C:37:CA:70

G7
F0:F0:F0:27:13:84

Galaxy Buds Pro
41:42:64:1C:20:BB

HOCO E58
1E:AE:2C:86:92:5C

JBL Flip 2
0C:A6:94:7C:3B:1B

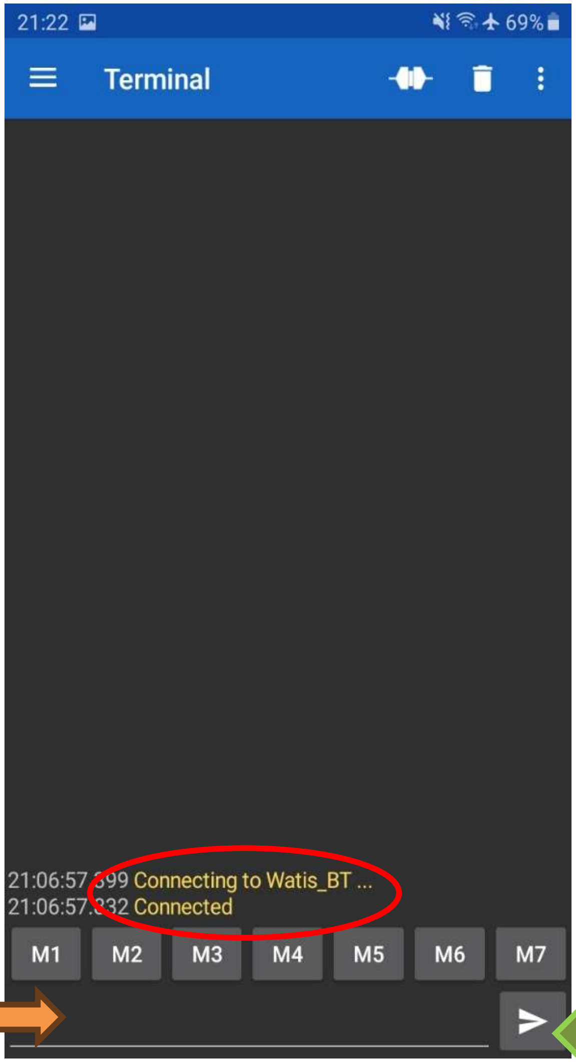
OBDII
01:23:45:67:89:BA

PLT BB PRO 2
E4:22:A5:7C:6B:63

Watis_BT
D4:8A:FC:D0:37:DA

Bluetooth Communication

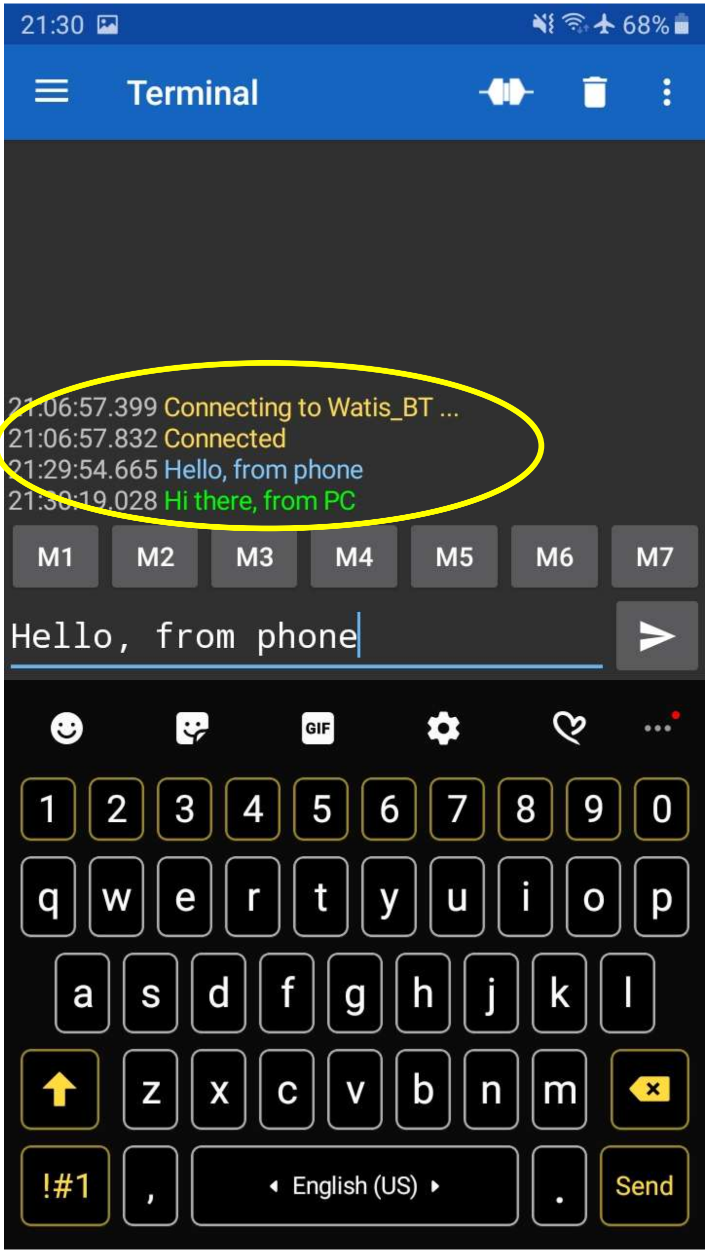
- After selecting your ESP32 Bluetooth device, go to Terminal screen
- Connection is established, you can send/receive message



```
21:06:57.399 Connecting to Watis_BT ...
21:06:57.832 Connected
```

Type message here and tab on send button

send button

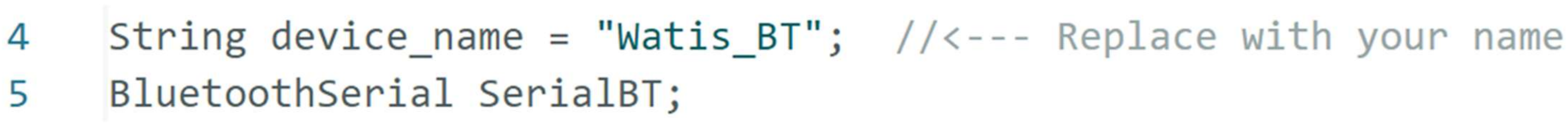


```
21:06:57.399 Connecting to Watis_BT ...
21:06:57.832 Connected
21:29:54.665 Hello, from phone
21:30:19.028 Hi there, from PC
```

Hello, from phone

Bluetooth Communication

- Application: **LED control via Bluetooth**
- Open sketch: **BTLED.ino**
- Give your ESP32 Bluetooth a name (Line 4 in the sketch)



```
4    String device_name = "Watis_BT";  //<--- Replace with your name
5    BluetoothSerial SerialBT;
```

- Upload the sketch to ESP32
- From Serial Bluetooth Terminal App:

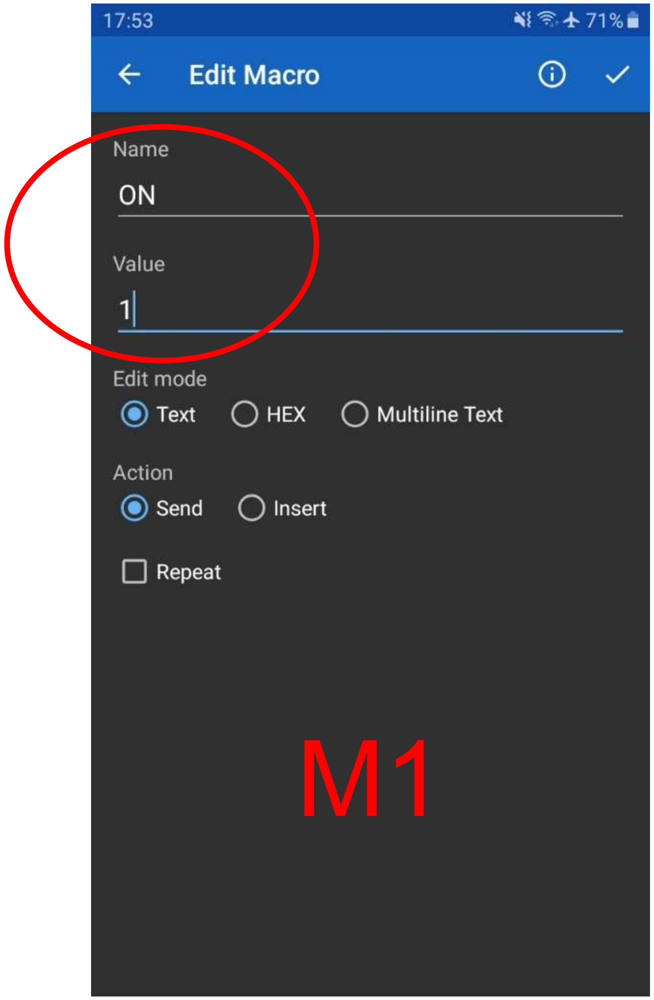
  - send number **1** to turn on the onboard LED

  - send number **0** to turn off the onboard LED

Bluetooth Communication

- We can use available soft buttons on the application to send **0** or **1**
- Long press M1 button to edit its function
- Long press M2 button to edit its function

ON



Edit Macro

Name

ON

Value

1

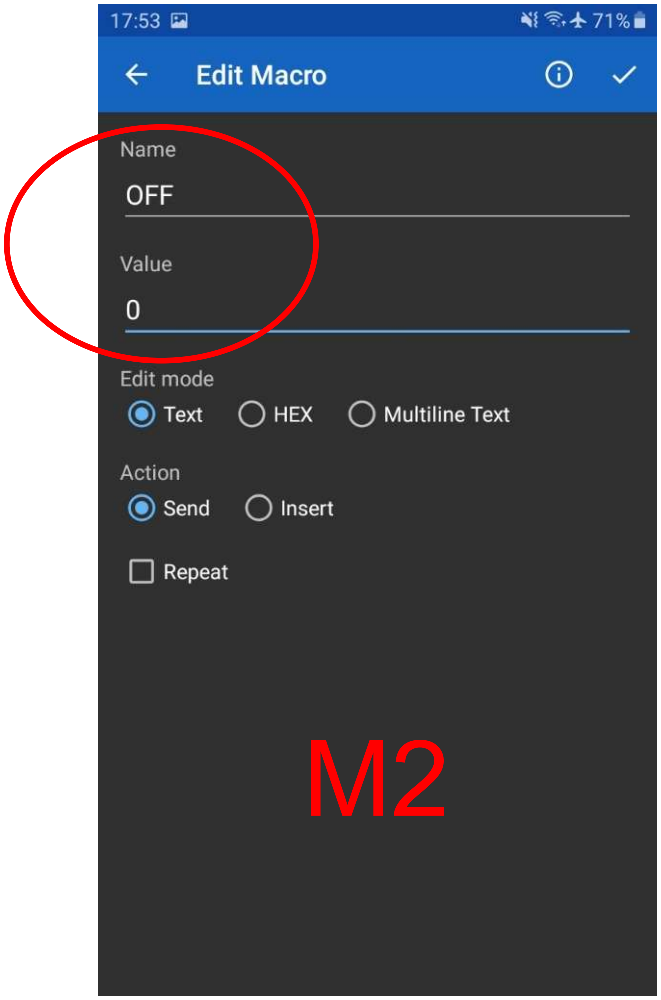
Edit mode

● Text ○ HEX ○ Multiline Text

Action

● Send ○ Insert

□ Repeat



Edit Macro

Name

OFF

Value

0

Edit mode

● Text ○ HEX ○ Multiline Text

Action

● Send ○ Insert

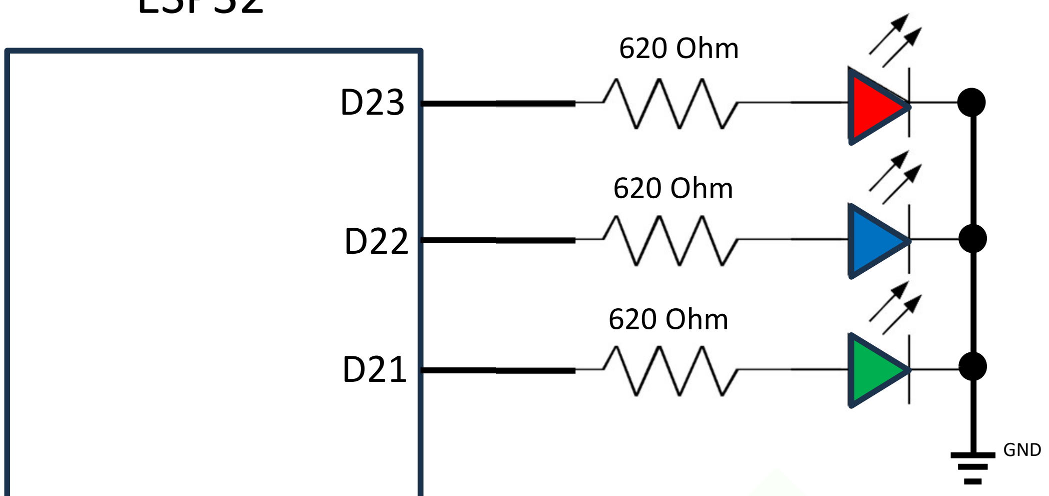
□ Repeat

OFF

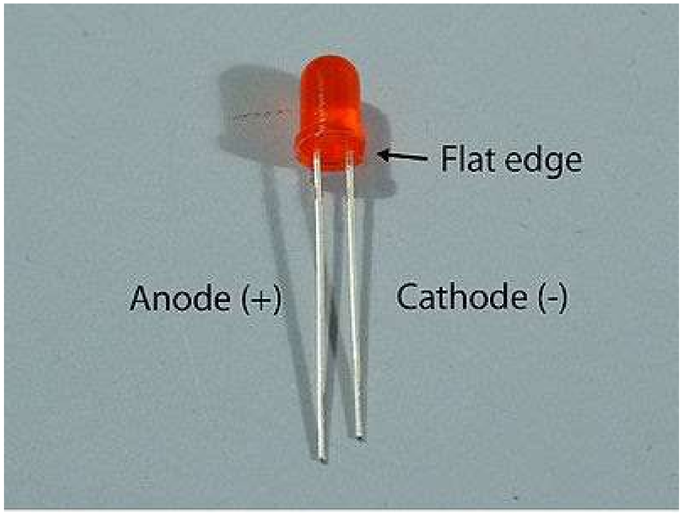
ESP32 Input/Output

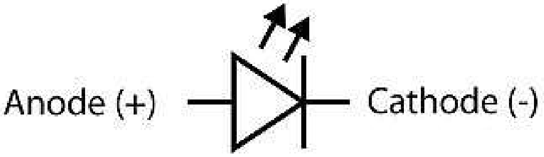
Exercise : Use Bluetooth to control 3 external LEDs

M1	M2	M3	M4	M5	M6	M7
R on	R off	G on	G off	B on	B off	M7



Resistor 620 Ohm



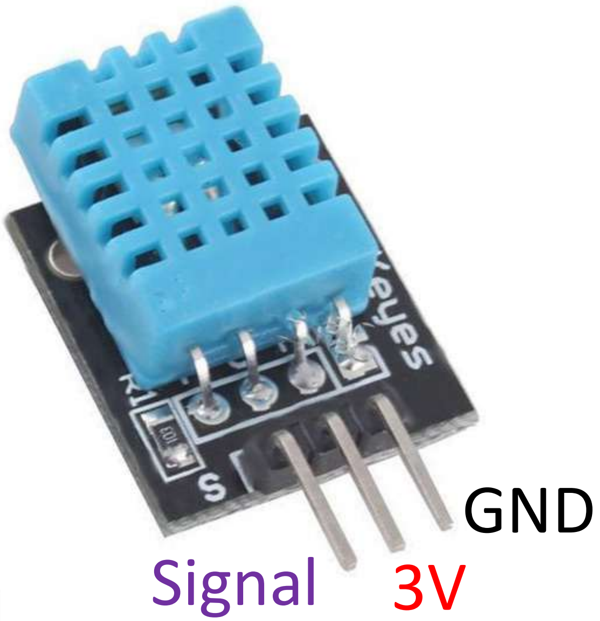


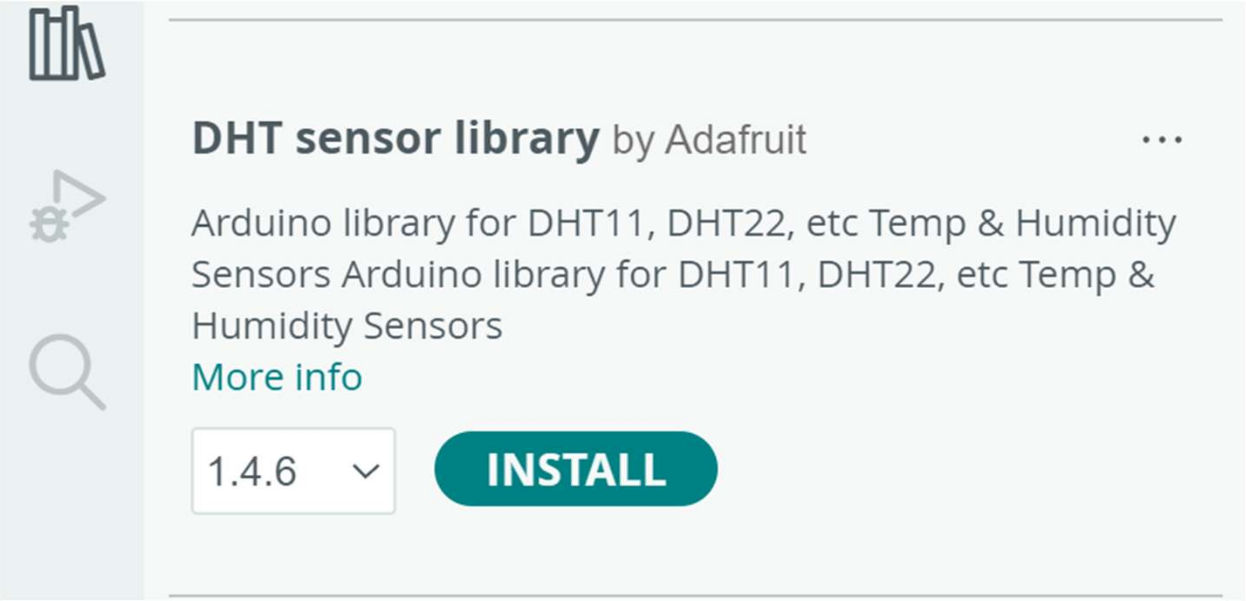
DHT11 Digital Temperature & Humidity Sensor

Specifications:

- Temperature 0-50°C
- Humidity 20-90%

In Arduino IDE, Install DHT sensor library





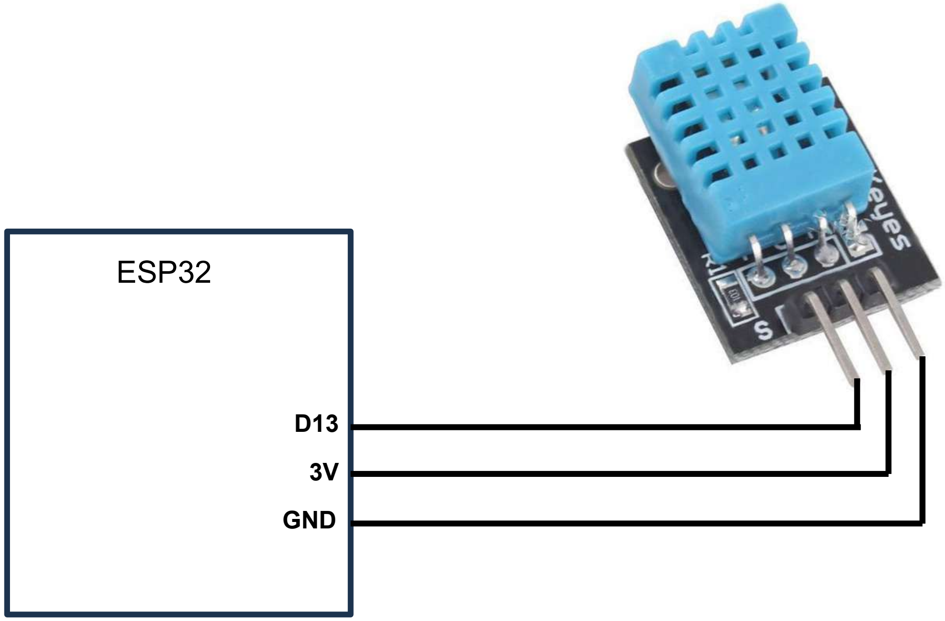
DHT sensor library by Adafruit

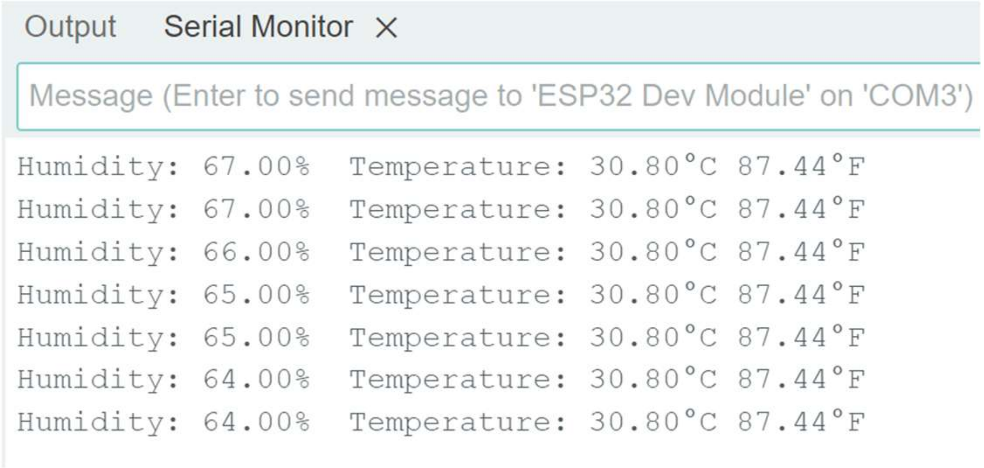
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors

More info

DHT11 Digital Temperature & Humidity Sensor

- Open sketch **dht11_demo.ino**
- Open serial monitor to see current temperature and humidity



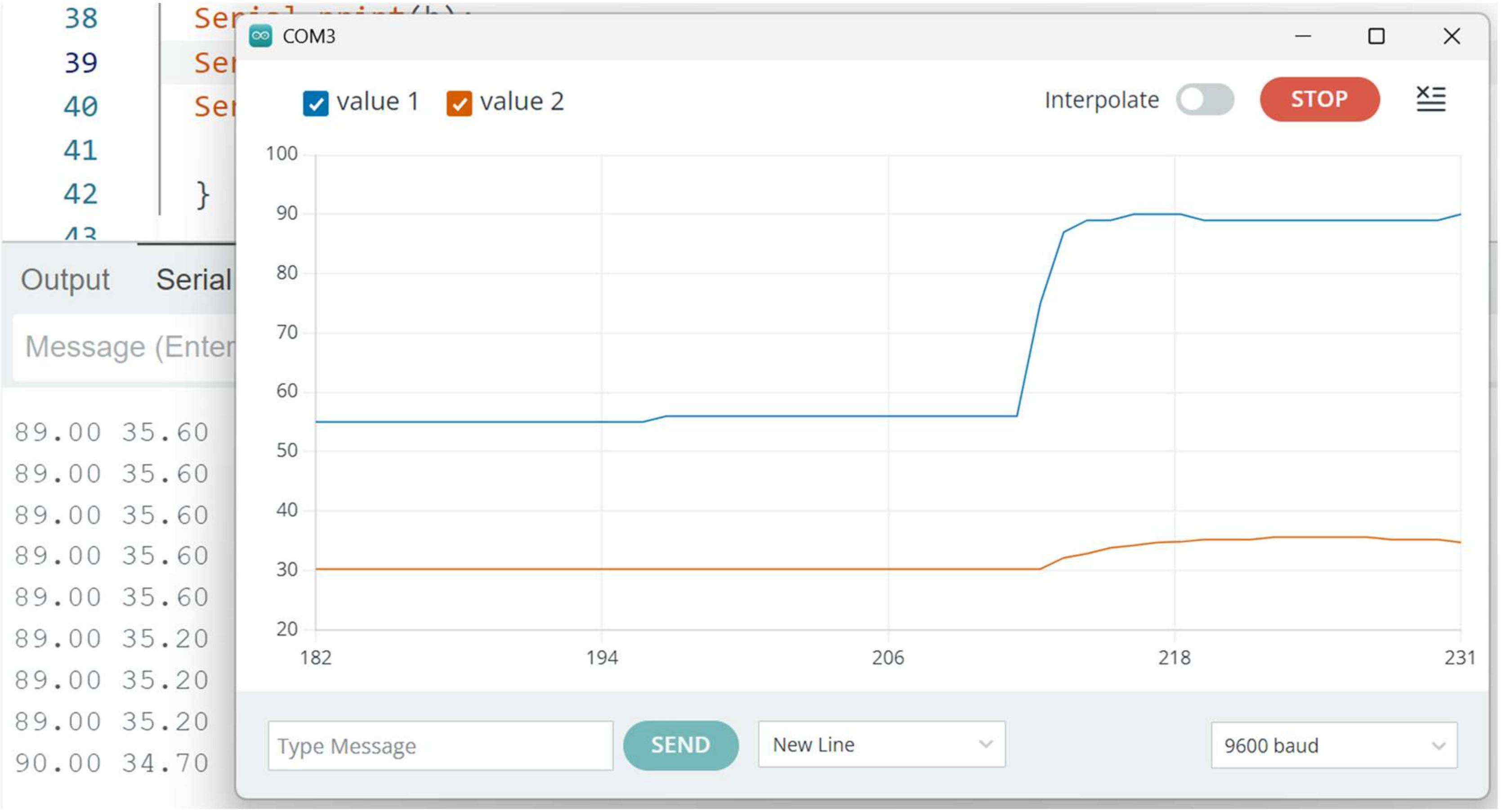


Message (Enter to send message to 'ESP32 Dev Module' on 'COM3')

```
Humidity: 67.00%  Temperature: 30.80°C 87.44°F
Humidity: 67.00%  Temperature: 30.80°C 87.44°F
Humidity: 66.00%  Temperature: 30.80°C 87.44°F
Humidity: 65.00%  Temperature: 30.80°C 87.44°F
Humidity: 65.00%  Temperature: 30.80°C 87.44°F
Humidity: 64.00%  Temperature: 30.80°C 87.44°F
Humidity: 64.00%  Temperature: 30.80°C 87.44°F
```

DHT11 Digital Temperature & Humidity Sensor

- Comment **Serial.print()** statements from lines 30 – 36
- Uncomment **Serial.print()** statements from lines 38 – 40
- Upload the sketch to ESP32
- Open serial plotter to see graphical plot of temperature and humidity

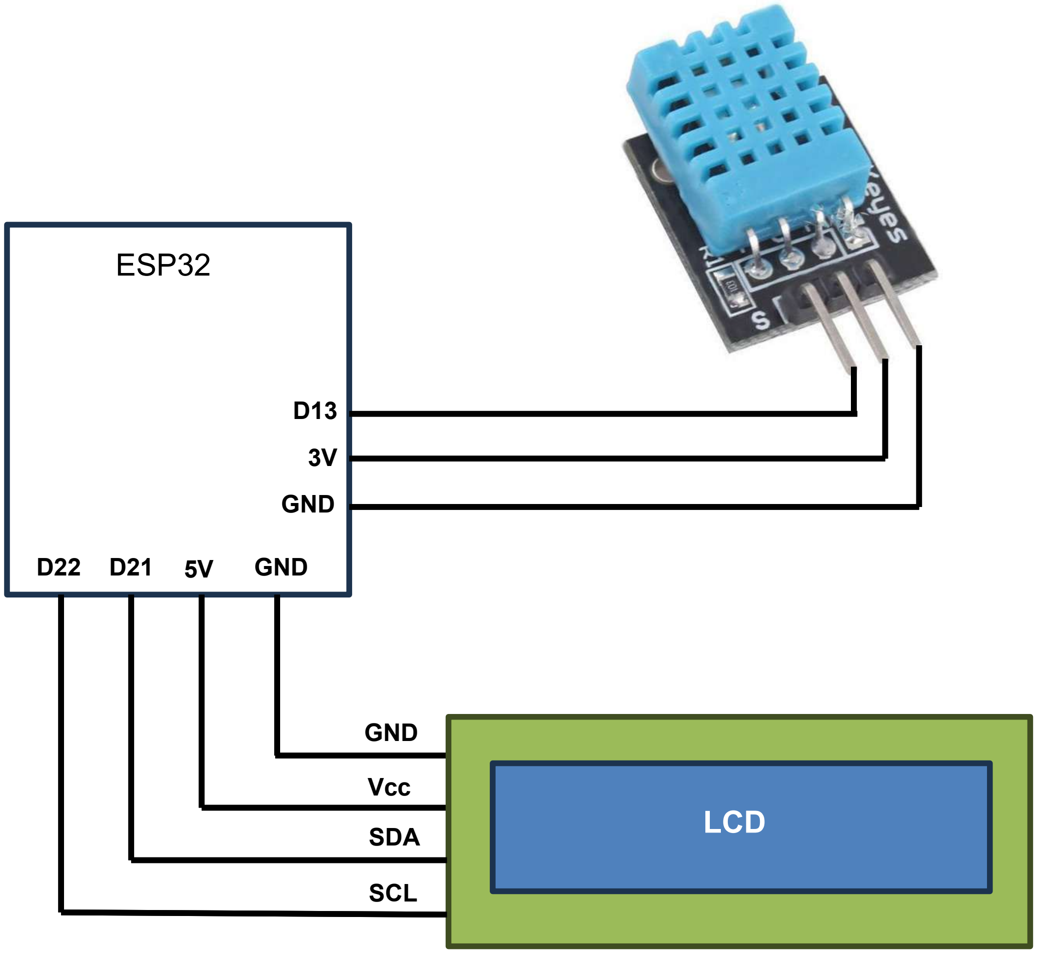


```
38    Serial.print(h);
39    Ser
40    Ser
41
42    }
43
```

```
89.00 35.60
89.00 35.60
89.00 35.60
89.00 35.60
89.00 35.60
89.00 35.20
89.00 35.20
89.00 35.20
90.00 34.70
```

DHT11 Digital Temperature & Humidity Sensor

Exercise: Display temperature and humidity on LCD display


ESP32
D13
3V
GND
D22
D21
5V
GND
GND
Vcc
SDA
SCL
LCD
S