BECE403E-EMBEDDED SYSTEM DESIGN

MODULE-5

PROGRAMMING THE PERIPHERALS OF MICROCONTROLLERS

MODULE-5

Programming the Peripherals of Microcontrollers

- Programming GPIO pins
- Timers / Counters, Watchdog Timer
- PWM generation
- ADC, DAC, LED, switches
- Keypad, LCD

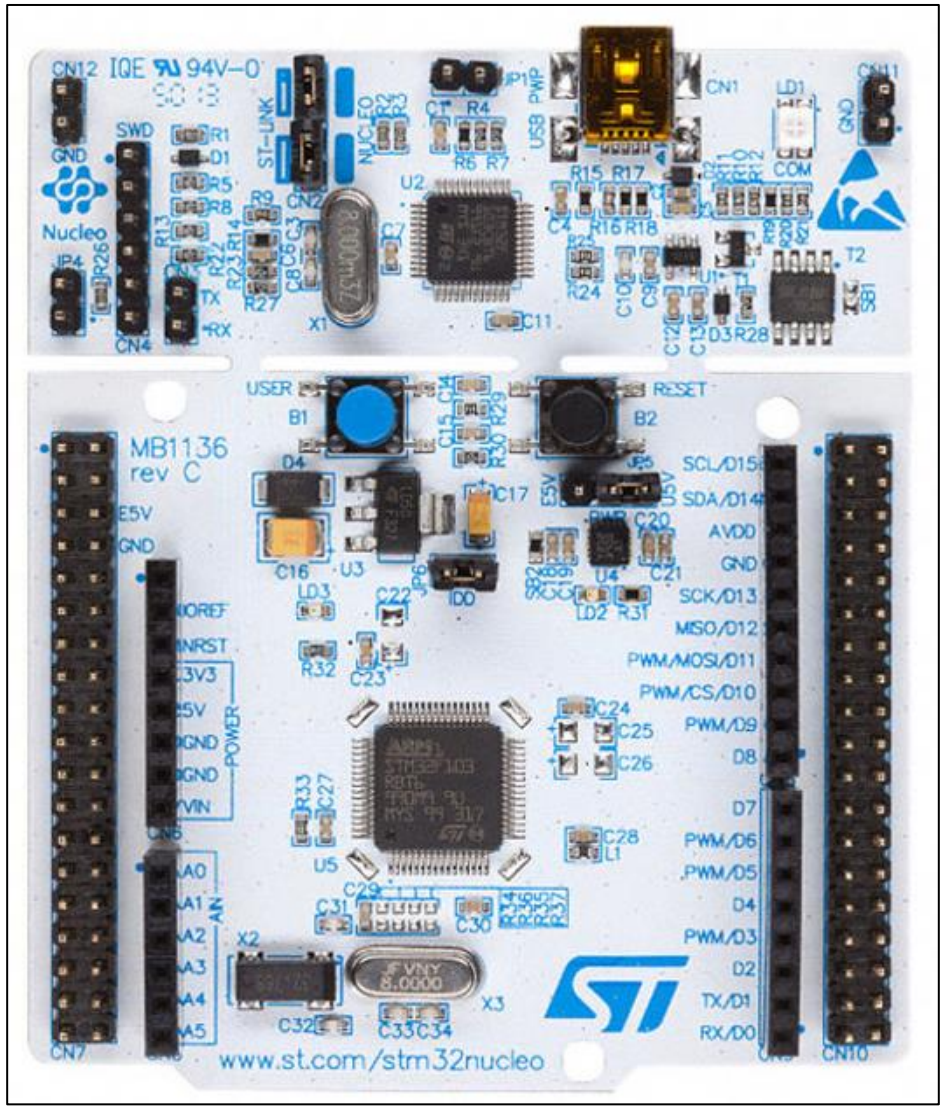
INTRODUCTION TO STM32 NUCLEO-64

INTRODUCTION TO STM32 NUCLEO-64

Nucleo Features

- STM32L152RE ultra-low-power ARM Cortex-M3 based microcontroller
- Two extension connectors: Arduino Uno and ST Morpho
- Embedded ST –LINK/V2-1 Debugger/Programmer
- Flexible board power supply: 3.3 V, 5 V, 7 - 12 V
- User LED (LD2), Two push buttons: USER and RESET
- MBED Enabled
- Interfaces supported on USB: Virtual COM port, Debug port, Mass storage (USB Disk drive) for drag 'n' drop programming

STM32 Nucleo-64 - Board Details



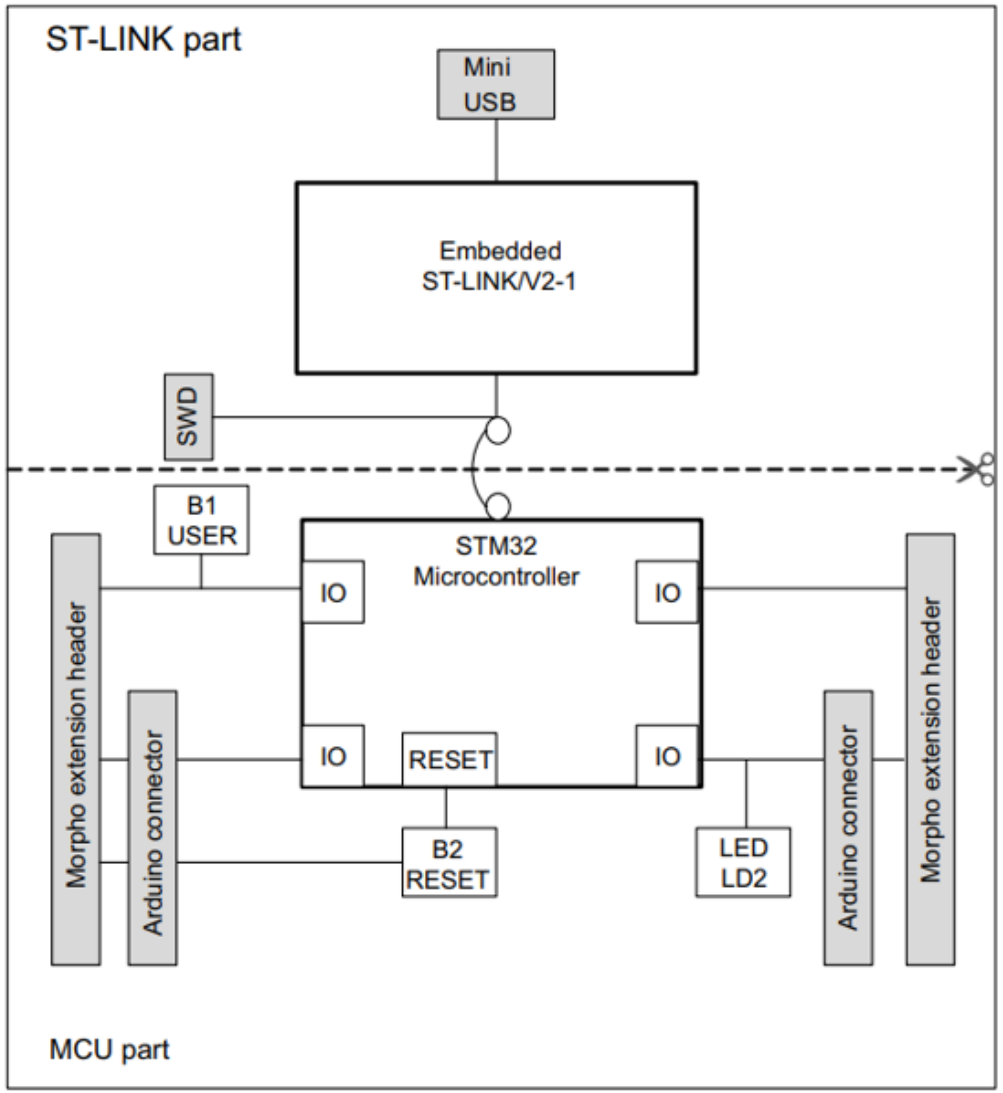
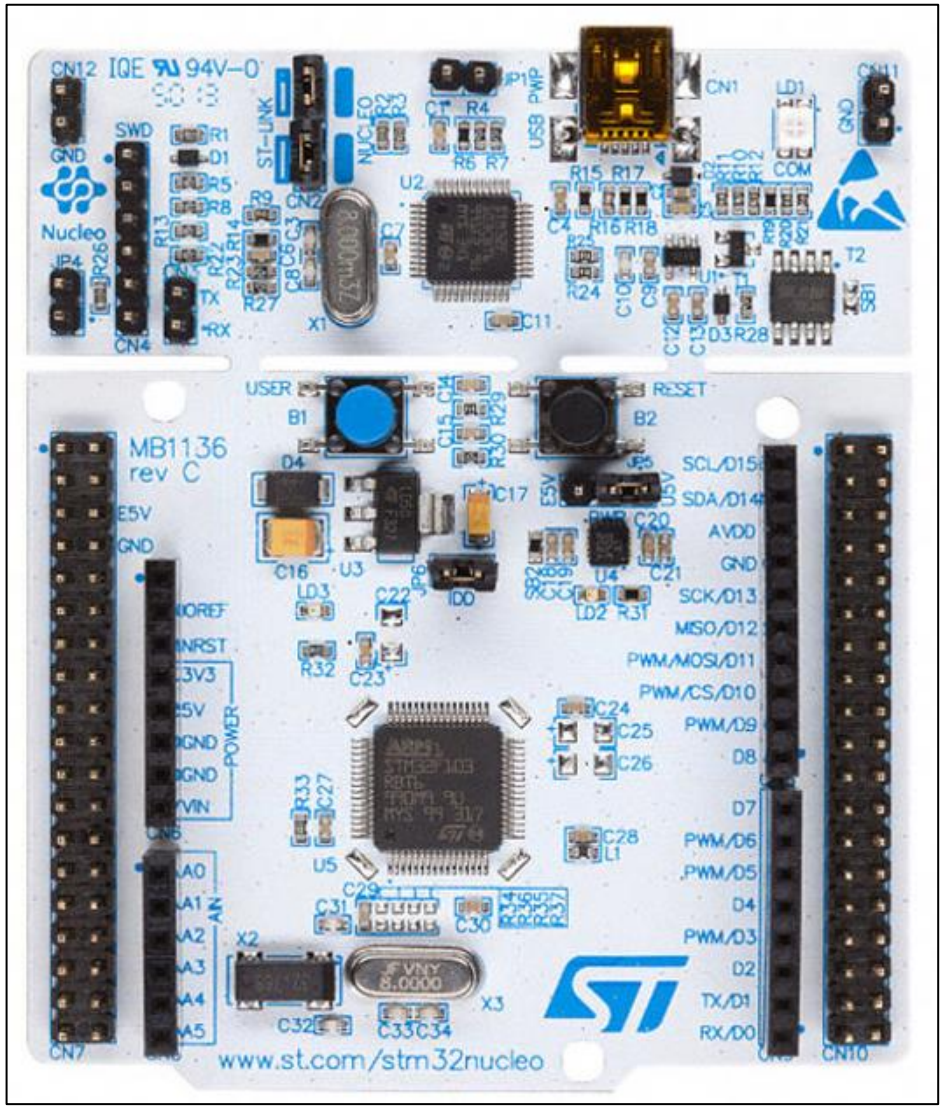


Image credit: Zephyr Ref. URL : https://docs.zephyrproject.org/latest/boards/arm/nucleo_l152re/doc/index.html

STM32 Nucleo-64 - Board Details



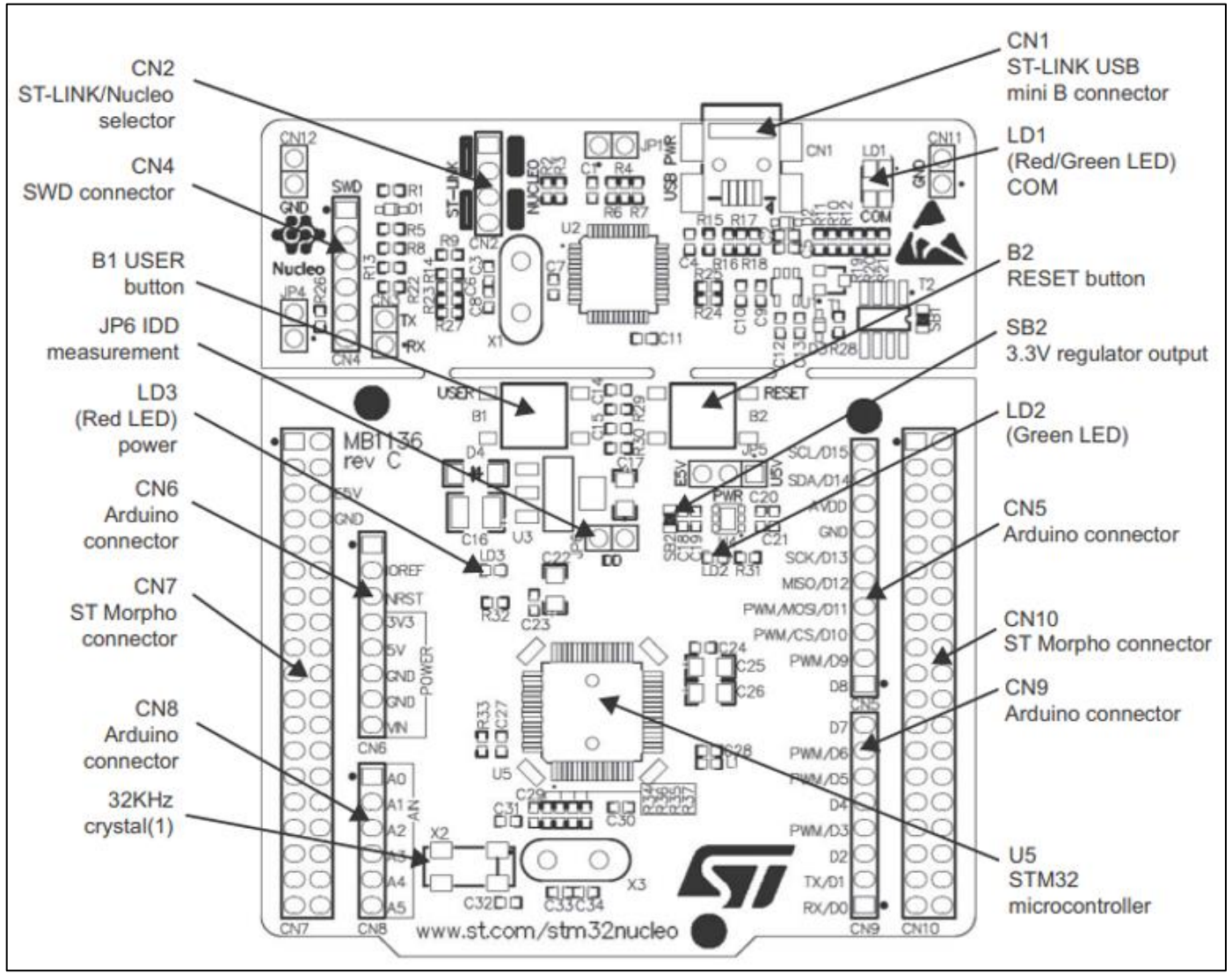


Image credit: Zephyr Ref. URL : https://docs.zephyrproject.org/latest/boards/arm/nucleo_l152re/doc/index.html

STM32 Nucleo-64 - Pin Details (Arduino Headers)

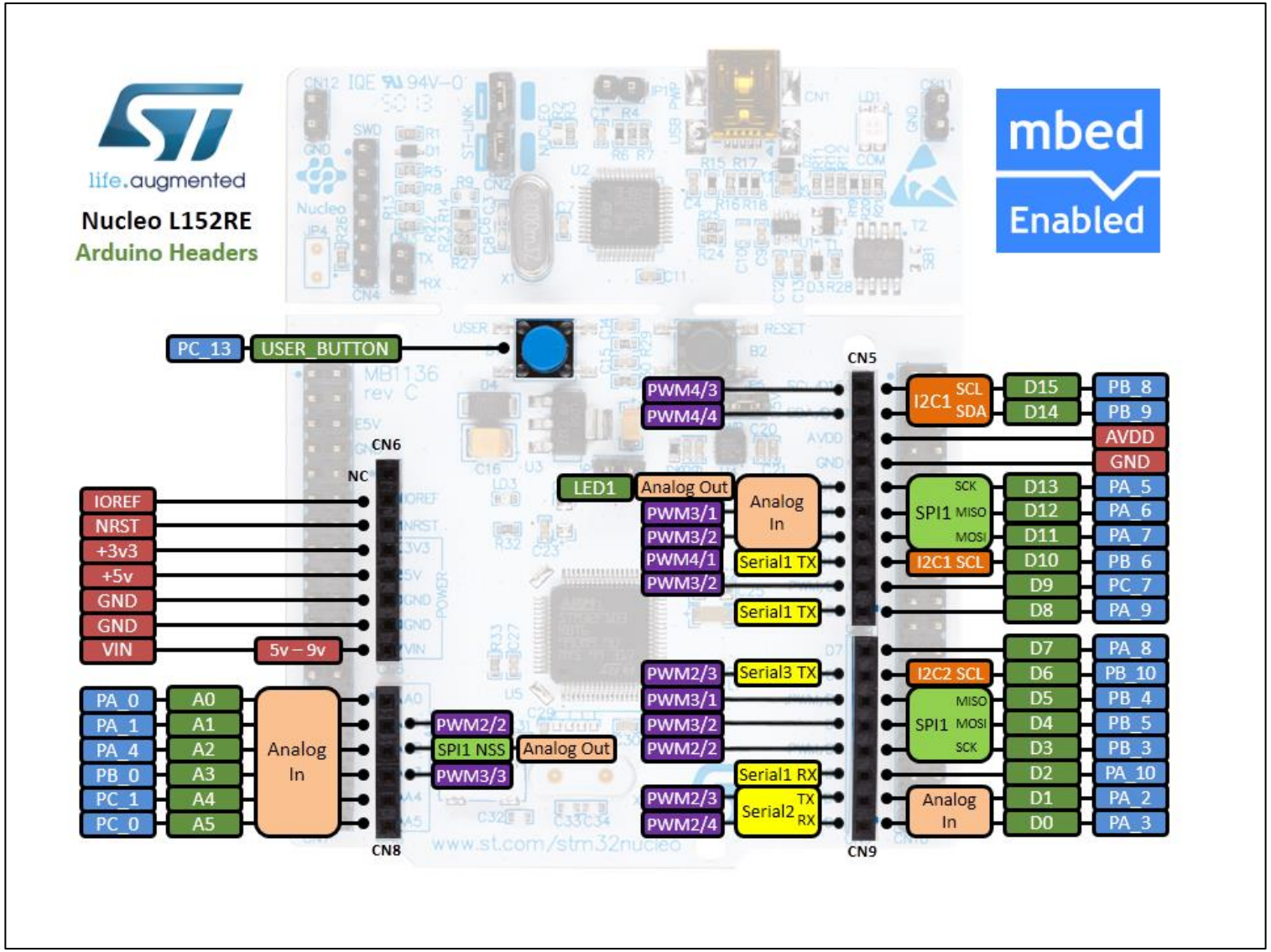
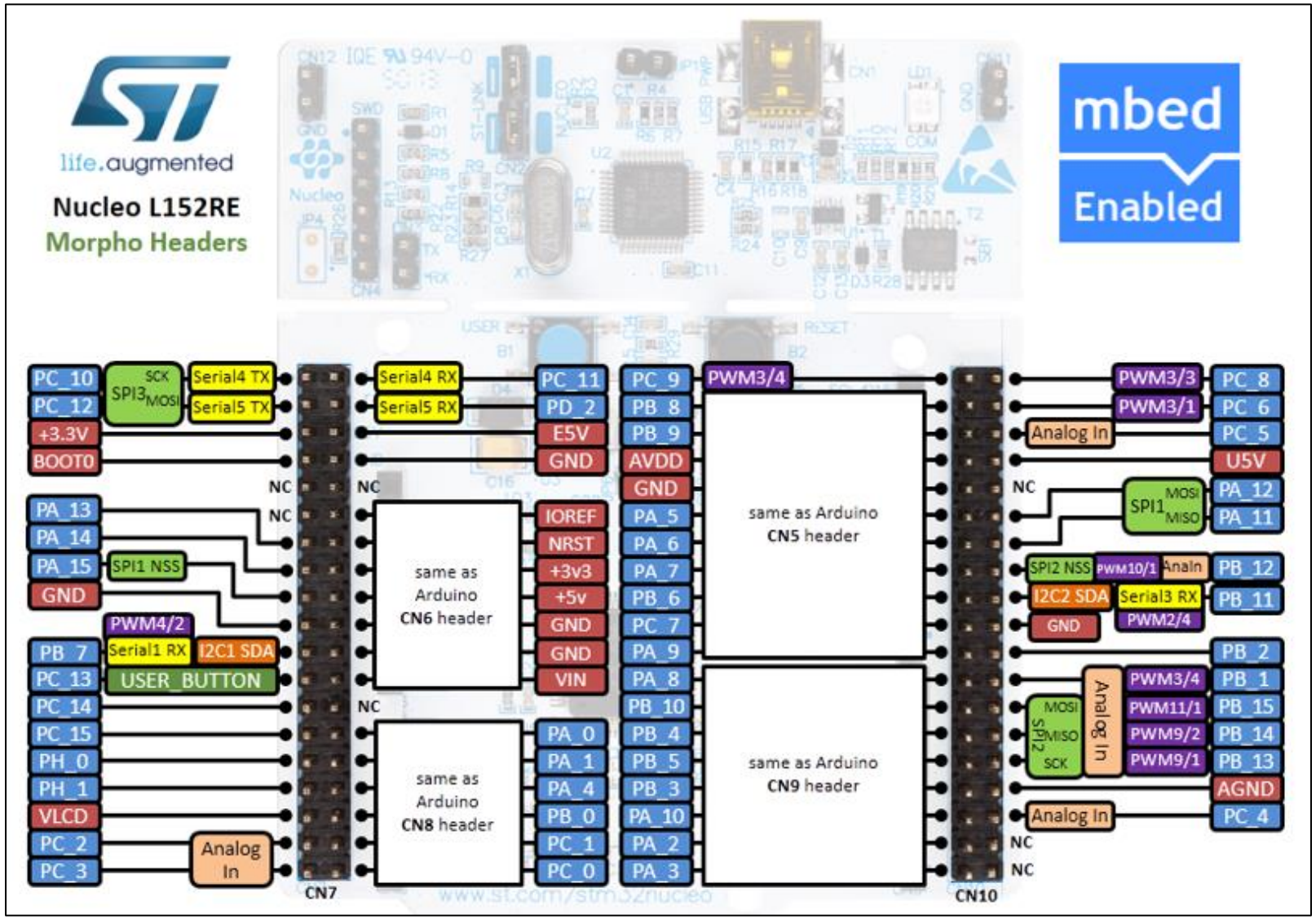


Image credit: arm MBED, Ref. URL : https://os.mbed.com/forum/electronics/topic/15346/?page=1#comment-52189

STM32 Nucleo-64 - Pin Details (Morpho Headers)



Pins Legend

- XXX — Power and control pins
- PX_Y — MCU pin without conflict
- XXX — LEDs and Buttons
- XXX — AnalogIn (ADC)
- XXX — Serial pins (USART/UART)
- XXX — SPI pins
- XXX — I2C pins
- XXX — PWMOut pins (TIMER n/c[N])
  n = Timer number c = Channel
  N = Inverted channel

Image credit: arm MBED, Ref. URL : https://os.mbed.com/forum/electronics/topic/15346/?page=1#comment-52189

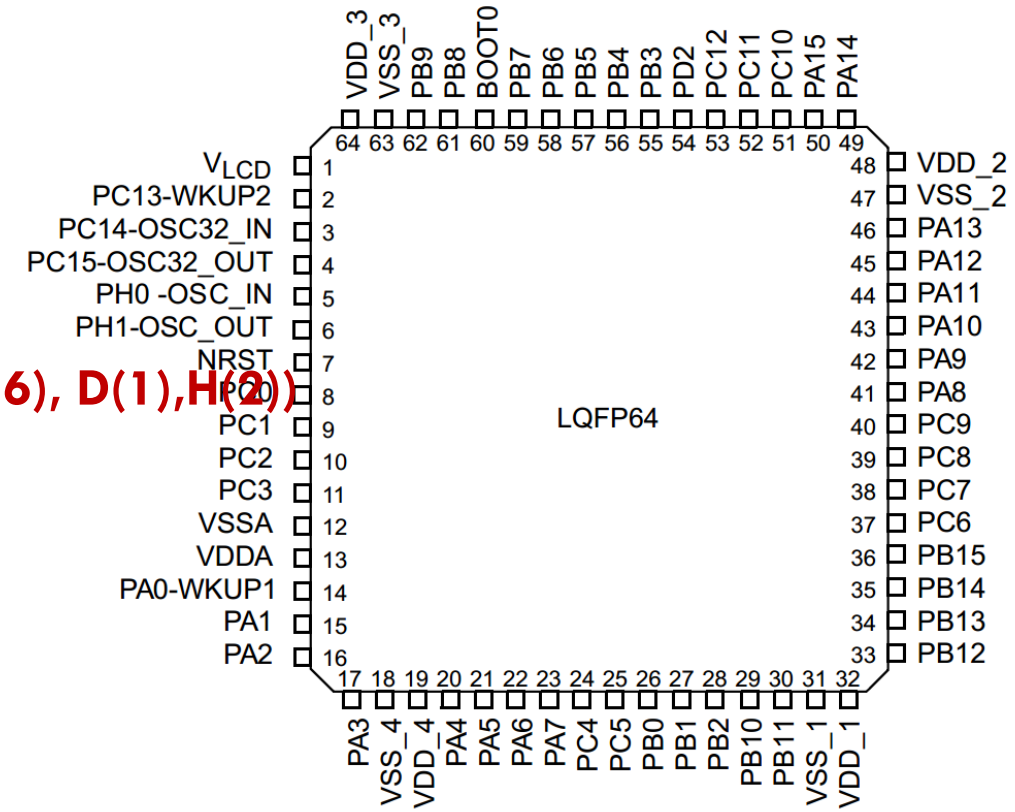
STM32 Nucleo-64 - Pin Details (Morpho Headers)

CN7 odd pins		CN7 even pins		CN10 odd pins		CN10 even pins	
Pin No.	Name	Name	Pin No.	Pin No.	Name	Name	Pin No.
1	PC10	PC11	2	1	PC9	PC8	2
3	PC12	PD2	4	3	PB8	PC6	4
5	VDD	E5V	6	5	PB9	PC5	6
7	BOOT0(1)	GND	8	7	AVDD	U5V(2)	8
9	-	-	10	9	GND	-	10
11	-	IOREF	12	11	PA5	PA12	12
13	PA13(3)	RESET	14	13	PA6	PA11	14
15	PA14(3)	+3V3	16	15	PA7	PB12	16
17	PA15	+5V	18	17	PB6	PB11	18
19	GND	GND	20	19	PC7	GND	20
21	PB7	GND	22	21	PA9	PB2	22
23	PC13	VIN	24	23	PA8	PB1	24
25	PC14	-	26	25	PB10	PB15	26
27	PC15	PA0	28	27	PB4	PB14	28
29	PH0	PA1	30	29	PB5	PB13	30
31	PH1	PA4	32	31	PB3	AGND	32
33	VLCD	PB0	34	33	PA10	PC4	34
35	PC2	PC1 or PB9(4)	36	35	PA2	-	36
37	PC3	PC0 or PB8(4)	38	37	PA3	-	38

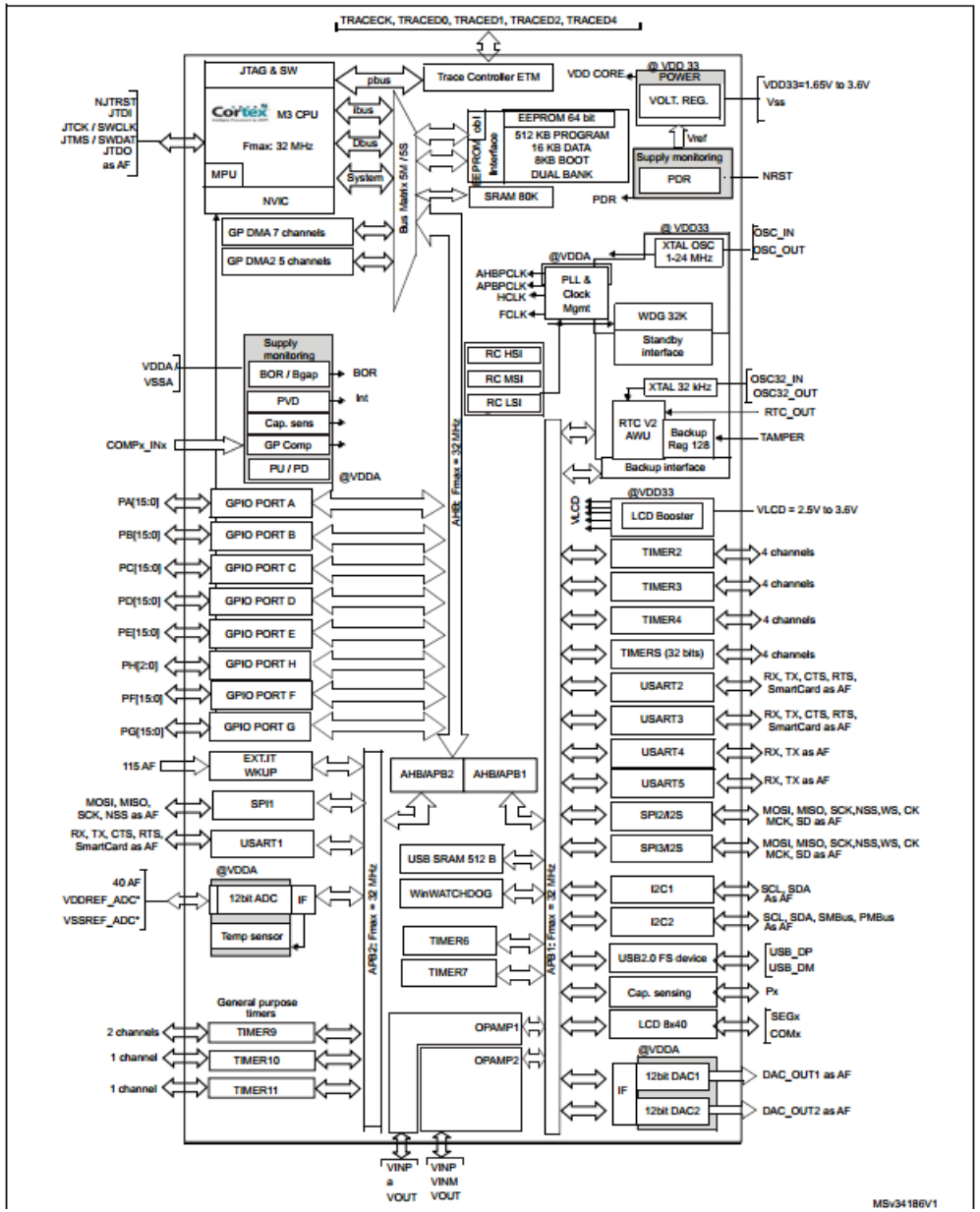
INTRODUCTION TO STM32L152RE

STM32L152RE Microcontroller Features

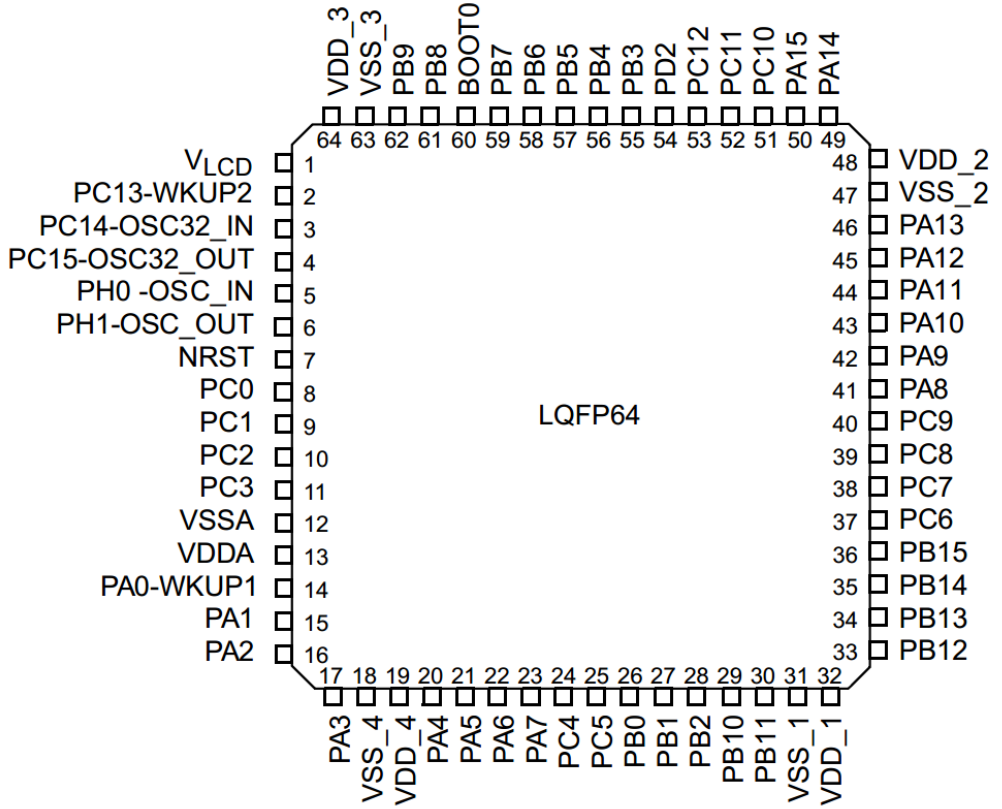
- Package Type : LQFP64 (Low-profile Quad Flat Package)
- Instruction set Architecture : RISC (Reduced Instruction Set Architecture)
- Data bus size : 32-bit
- Address bus size : 32-bit
- Operating Frequency : 32 MHz max
- Operating Voltage : 1.65 V to 3.6 V
- Program memory (flash) : 512 KB
- Data memory (SRAM) : 80 KB
- Data memory (EEPROM) : 16 KB
- Input/output ports : 51**(Port A(16), B(16), C(16), D(1),H(2))**
- Timers : 9
- Watch-dog Timers : 2
- Interrupts : 56
- ADC modules (12-bit) : 21 channels
- DAC modules (12-bit) : 2 channels
- PWM (used with timers) : 18
- Serial communication : SPI(8), USART(5), I2C(2)



STM32L152RE - Architecture



STM32L152RE – Pin diagram



INTRODUCTION TO Keil Studio IDE

- Keil Studio Cloud is the successor to the Mbed Online Compiler, and allows you to develop Mbed 2, Mbed OS 5 and 6 projects on supported Mbed-enabled boards.
- Free to use, browser-based IDE for the evaluation and development of embedded, IoT, and Machine Learning software for Cortex-M devices.
- You can compile projects using Arm Compiler 6, run the projects directly on supported development boards, and debug from supported browsers without the need to install any software.
- You can access Keil Studio Cloud using an Arm or Mbed account (use below link) and get started by opening a reference design to evaluate.

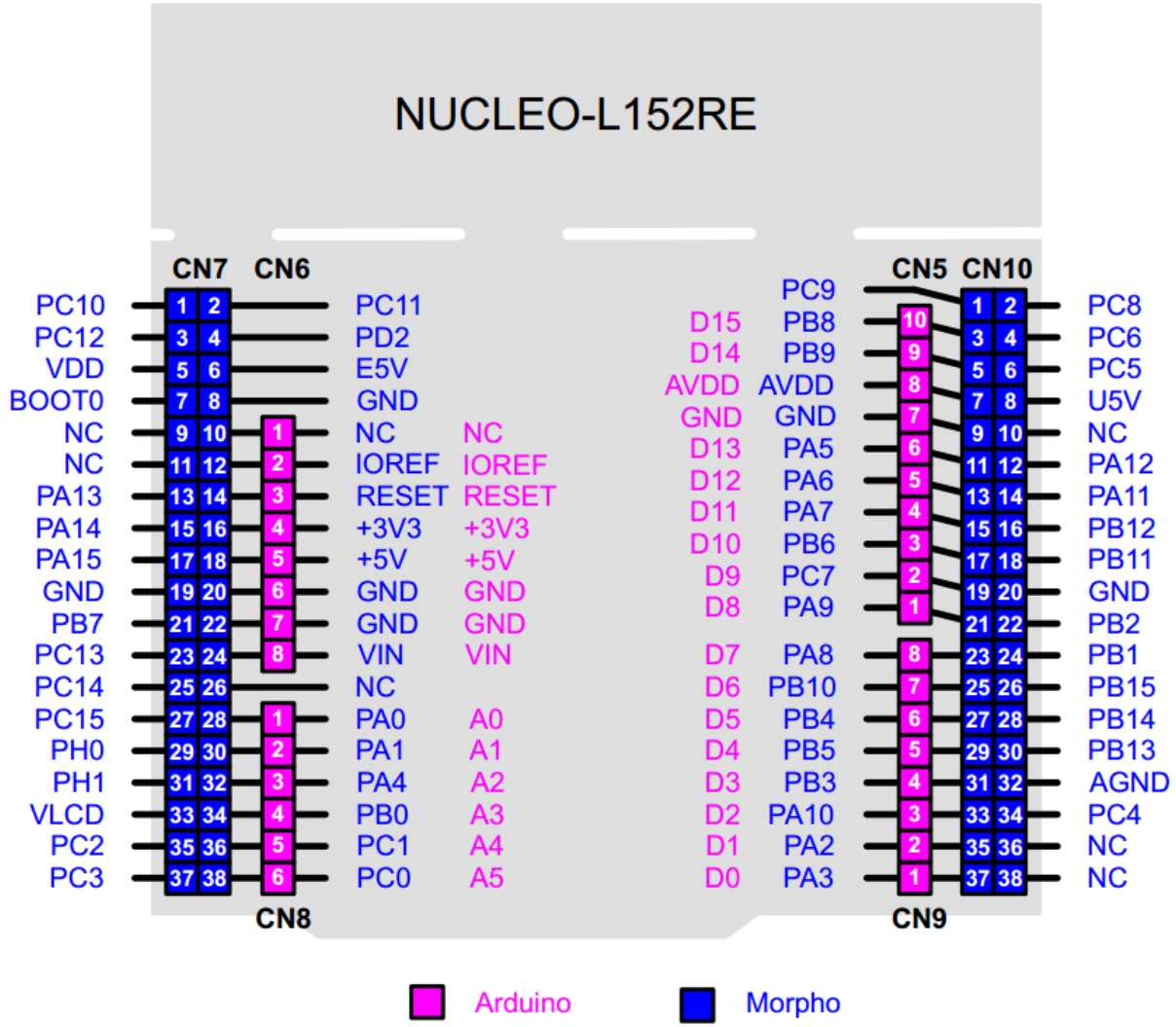
https://studio.keil.arm.com

PROGRAMMING GPIO PINS (LED & SWITCHES)

INTRODUCTION TO GPIO PINS

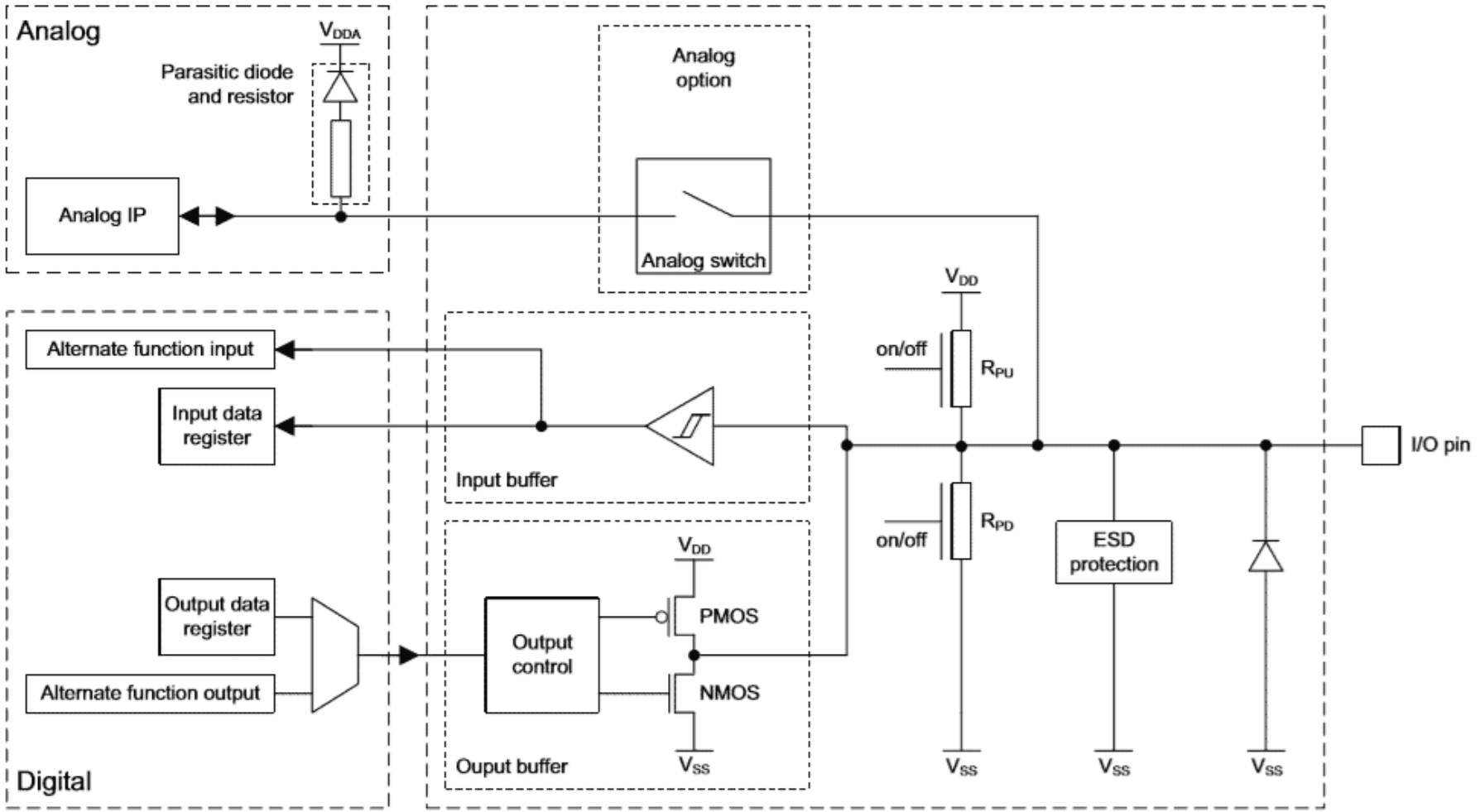
- GPIO pin stands for the General-purpose input/output pin on MCU.
- In STM-32 NUCLEO-L152RE development board, there 51 GPIO pins are available and they are organized as Port A(16), B(16), C(16), D(1),H(2).
- Most of the GPIO pins are shared with digital or analog or alternate functions, and can be individually remapped using dedicated registers.
- Each of the GPIO pins can be configured by software as output, as input or as peripheral alternate function.
- The use of GPIO is driving LEDs, reading digital signals from switch, issuing interrupts, and many.

PROGRAMMING GPIO PINS



GPIO pins of STM32 Nucleo L152RE board

PROGRAMMING GPIO PINS



Internal block of the typical STM32 MCU

PROGRAMMING GPIO PINS

The mbed digital output API summary.

Functions	Usage
`DigitalOut`	Create a DigitalOut object, connected to the specified pin
`write`	Set the output, specified as 0 or 1 (int)
`read`	Return the output setting, represented as 0 or 1 (int)
`mbed-defined operator: =`	A shorthand for write
`mbed-defined operator: int( )`	A shorthand for read

The mbed digital input API summary.

Functions	Usage
`DigitalIn`	Create a DigitalIn object, connected to the specified pin
`read`	Read the input, represented as 0 or 1 (int)
`mode`	Set the input pin mode, with parameter chosen from: PullUp, PullDown, PullNone, OpenDrain
`mbed-defined operator: int()`	A shorthand for read ()

mbed library wait functions.

C/C++ Function	Action
`wait`	Waits for the number of seconds specified (float)
`wait_ms`	Waits for the whole number of milliseconds specified (int)
`wait_us`	Waits for the whole number of microseconds specified (int)

PROGRAMMING GPIO PINS

API required:

- DigitalOut Identifier(PinName) - Used configure GPIO pin as digital output pin
  - Identifier: User defined name to access this digital pin
  - PinName: Name of the pin as specified in the Nucleo board pin details
  - Examples: `DigitalOut myled(LED1);` //LED1 is an onboard LED
    `DigitalOut myA3(PC_0);` //PC_0 is 0th pin of port C

- operator= (int value) - used to set the digital output as 0 or 1. A shorthand for write() API.
  - Example: `myA3=1;` //set the PC_0 digital output pin value as HIGH or 1

- wait(s) – Used to pauses the execution of a program for the amount of time (in seconds)
  - s: Float value indicate the number of seconds to pause.
  - Example: `wait(1.0);` //will pause the program for 1 second.
    `wait(0.5);` //will pause the program for 0.5 second

PROGRAMMING GPIO PINS

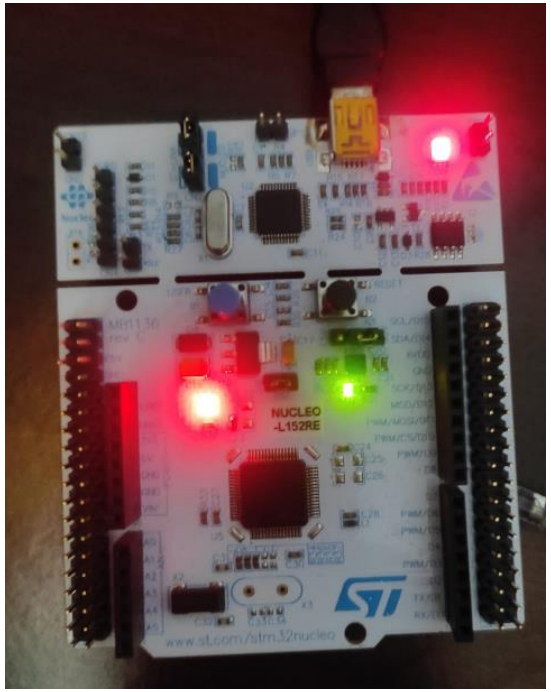
Example-1 - Blinking of LED on Board

Write a program to blink an on-board LED (LD2) of the STM32 Nucleo-64 board with 0.5s delay between ON and OFF state using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
  while(1) {
    myled = 1;
    wait(0.5);
    myled = 0;
    wait(0.5);
  }
}

```

Output:



PROGRAMMING GPIO PINS

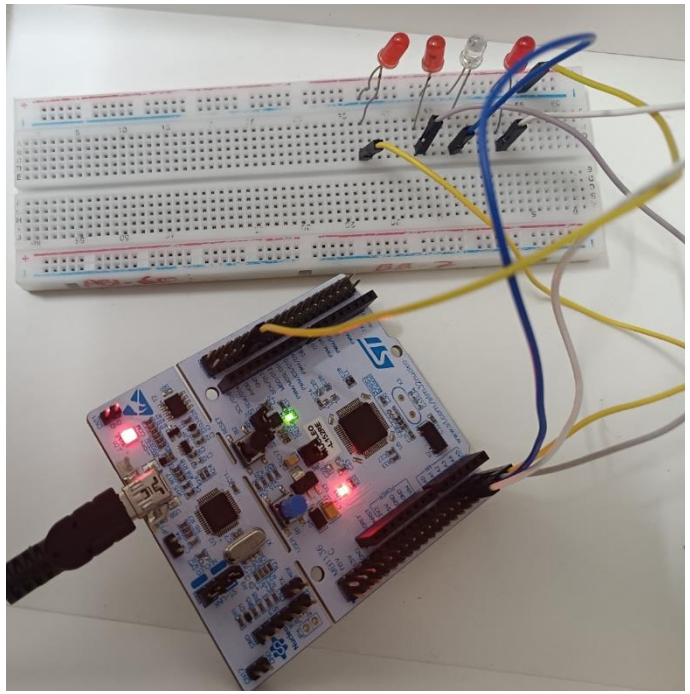
Example-2 - Blinking of 4 LEDs alternatively in pairs

Write a program to blink four LEDs in alternatively in pairs (1010 and 0101 pattern) with 0.5 Sec delay between each. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"
DigitalOut myled1(PC_4);
DigitalOut myled2(PB_13);
DigitalOut myled3(PB_14);
DigitalOut myled4(PB_15);
int main() {
while(1) {
myled1 = 1;
myled2 = 0;
myled3 = 1;
myled4 = 0;
wait(0.2);
myled1 = 0;
myled2 = 1;
myled3 = 0;
myled4 = 1;
wait(0.2);
}
}
```

Connection diagram:



PROGRAMMING GPIO PINS

Example-3 - Display Hexadecimal counting sequence in LEDs

Write a C++ program with mbed APIs to display Hexadecimal Counting Pattern from 0 to 15 by blinking LEDs. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

API required:

- BusOut Identifier(PinNames) - Used to combine a number of DigitalOut pins to write them at once. You can have up to 16 pins in a Bus.
  - Identifier: User defined name to access this digital pin
  - PinNames: Name of the pins as specified in the Nucleo board pin details
  - Examples: `BusOut myleds(a,b,c,d,e,f,g,h);` //a-LSB and h-MSB
    `BusOut hexa(PA_3,PB_13,PAB_14,PB_15);` //PA_3-LSB, PB_15-MSB

PROGRAMMING GPIO PINS

Example-3 - Display Hexadecimal counting sequence in LEDs

Program:

```
#include "mbed.h"
BusOut myled(PC_4,PB_13,PB_14,PB_15);
int i;
int main() {
while(1) {
myled = 0x00;
for(i=0;i<16;i++)
{
myled = myled+1;
wait(1);
}
}
}
```

PROGRAMMING GPIO PINS

Example-4 - Controlling LED Blinking Sequence using Switch

Write a C++ program with mbed APIs to blink one LED at a time serially in a group of 4 LEDs using switch.

- If switch is LOW, blink the LEDs one at a time from left to right (1000, 0100, 0010, 0001 pattern) with 0.5 Sec delay between each.
- If switch is HIGH, blink the LEDs one at a time from right to left (0001, 0010, 0100, 1000 pattern) with 0.5 Sec delay between each.

Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

API required:

- DigitalIn Identifier(PinName) - To read the value of a digital input pin (logic level 0 or 1).
  - Identifier: User defined name to access this digital pin
  - PinName: Name of the pins as specified in the Nucleo board pin details
  - Examples: DigitalIn switch(PA_3); //PA_3 is configured as Digital input pin

PROGRAMMING GPIO PINS

Program:

```
#include "mbed.h"
DigitalIn switch1(PC_10);
BusOut myled(PC_4,PB_13,PB_14,PB_15);
int main() {
while(1) {
if(switch1 == 0)
{
myled = 8;//Binary value 1000
wait(0.5);
myled = 4;//Binary value 0100
wait(0.5);
myled = 2;//Binary value 0010
wait(0.5);
myled = 1;//Binary value 0001
wait(0.5);
}
```

```
else
{
myled = 1;//Binary value 0001
wait(0.5);
myled = 2;//Binary value 0010
wait(0.5);
myled = 4;//Binary value 0100
wait(0.5);
myled = 8;//Binary value 1000
wait(0.5);
}
}
}
```

PROGRAMMING GPIO PINS

Exercise-1 – Controlling LED Blinking Sequence using Switch

Write a C++ program with mbed APIs to design a ***traffic light controller system*** for a four lane junction (North, South, East, West) to coordinate the traffic moves.

- Use 12 LEDs (3 for each direction) for traffic light signal and Switch decide the traffic light mode.
- If Switch is LOW, traffic light operate in late night mode in which only yellow light (LED) in all lanes blink for every 2 seconds continuously.
- If Switch is HIGH then traffic light operates in normal signalling operation in which similar signal sequence are applied in opposite lanes (for ex. east and west) with Green signal for 15 Sec. then Yellow for 2 Sec. and then Red. Meanwhile, north & south lane will be at red.
- Once this sequence is completed, switch to next opposite lanes (i.e north & south) and carry out signalling with Green signal for 10 Sec. then Yellow signal for 2 Sec. and then Red meanwhile, east & west lane will be in red.
- Continuously switch between these two signal sequence.

Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Hint: Use 6 Digital pins for lane signalling LEDs (3 for East & West, 3 for North & South) and one Switch for Mode selection

PROGRAMMING GPIO PINS

Example-5 - Interfacing with IR Sensor

Write a C++ program with mbed APIs to test the IR sensor by switching ON the LED according to the status of the IR sensor. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

IR Sensor:

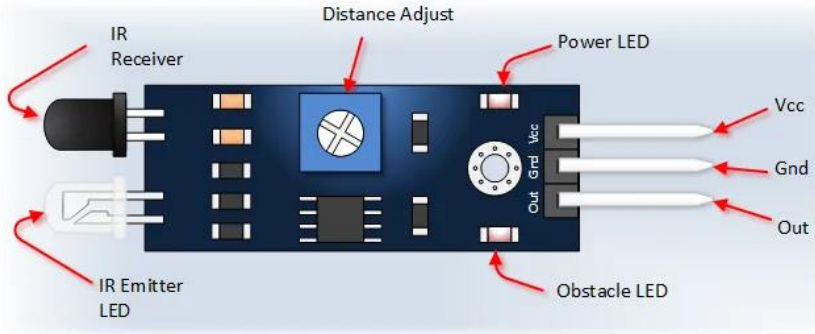
- An infrared sensor (IR sensor) is an optoelectronic device widely used to sense some object of the surroundings. It can measure the heat of an object as well as detects the motion.
- IR sensor typically has an IR LED (transmitter emitting IR radiations) & an IR photodiode (responds to the infrared light), and combining these two gives way to a photo-coupler or optocoupler.
- The resistance of photo-diode and its output voltage is directly proportional to the infrared light.
- IR sensor can be used for proximity detection, item counter, line following robot etc.,

PROGRAMMING GPIO PINS

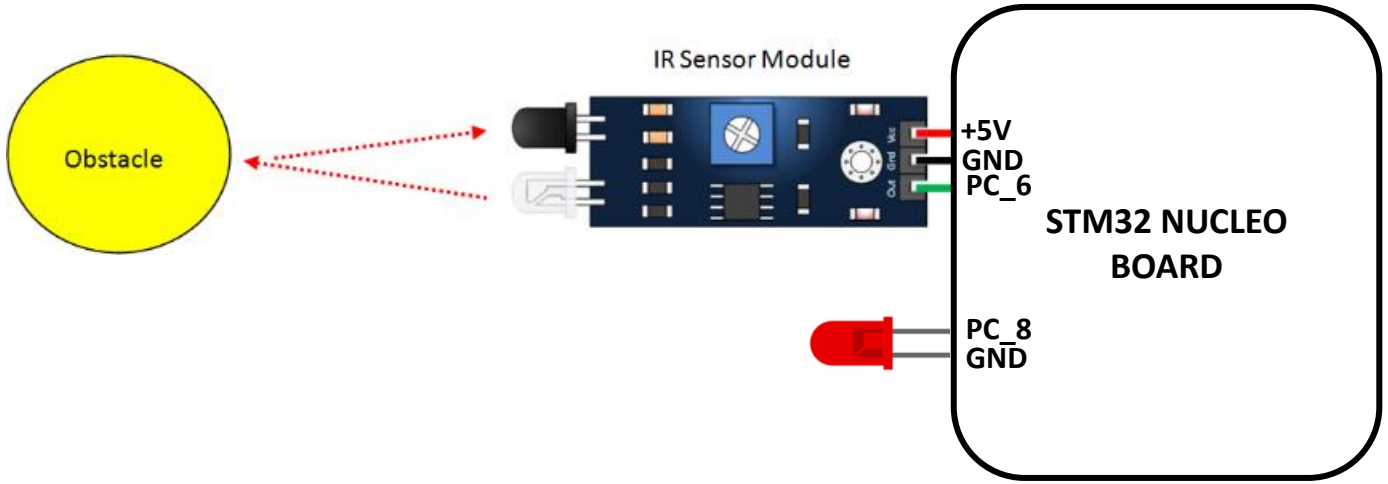
Program:

```
#include "mbed.h"
DigitalOut redled(PC_8);
DigitalIn opto_switch(PC_6);

int main()
{
while(1)
{
if (opto_switch==1)
redled = 1 ;
else
redled= 0;
}
}
```

IR Sensor pin details



Connection Diagram:



PROGRAMMING GPIO PINS

Example-7 - Display 0 to 9 in 7-Segment display

Write a C++ program with mbed APIs to display 0 to 9 on 7-segment display unit. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

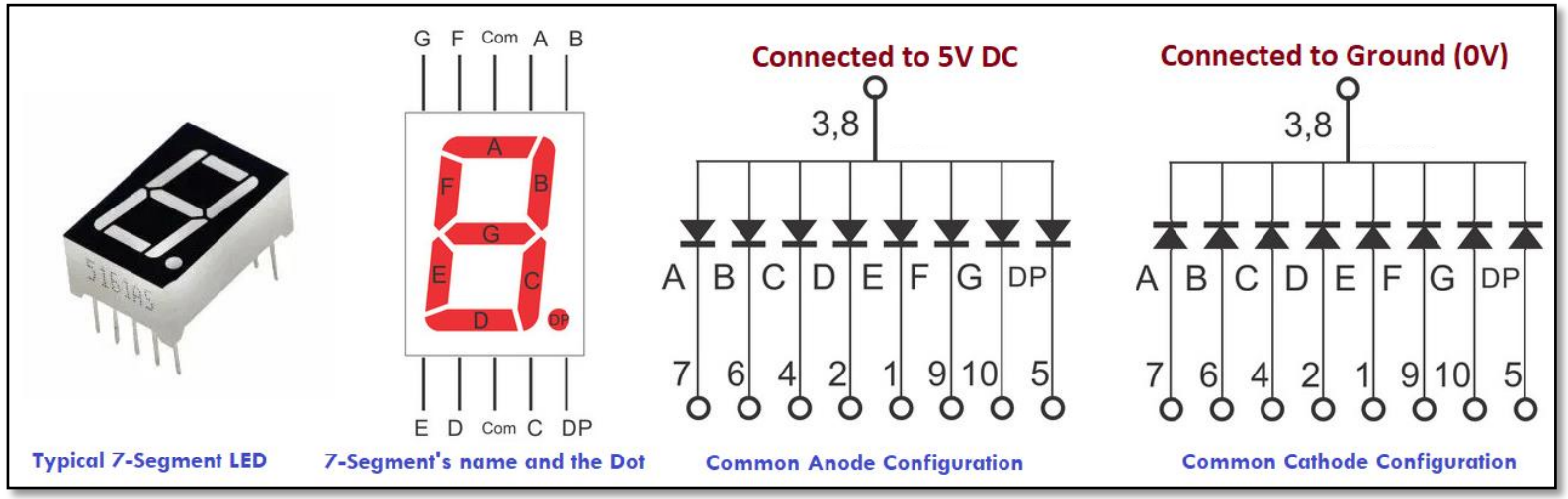
7-Segment display:

- Seven segment displays are the output display devices that provide a way to display information in the form of alphanumeric characters and digits.
- This is most commonly used in many gadgets, and electronic appliances like digital meters, digital clocks, microwave oven and electric stove, etc.
- These displays consist of seven segments (indicated as A-G) of light emitting diodes (LEDs) and that is assembled into a structure like numeral 8. In addition, an extra 8th segment is used to display dot (indicated as H/DP), which is useful while displaying non integer number.

PROGRAMMING GPIO PINS

Example-7 - Display 0 to 9 in 7-Segment display

- A seven segment displays are generally available in ten pin package, 8 pins relate to the 8 LEDs, the remaining two pins at middle are internally shorted.
- These segments come in two outlines they are common cathode and common anode.
  - In common cathode configuration, the all LED's negative terminals are connected to the common ground pin hence HIGH on corresponding positive pin glow particular LED.
  - In a common anode arrangement, the common pin is given to a logic HIGH and the pins of the LED are given LOW to display a number.

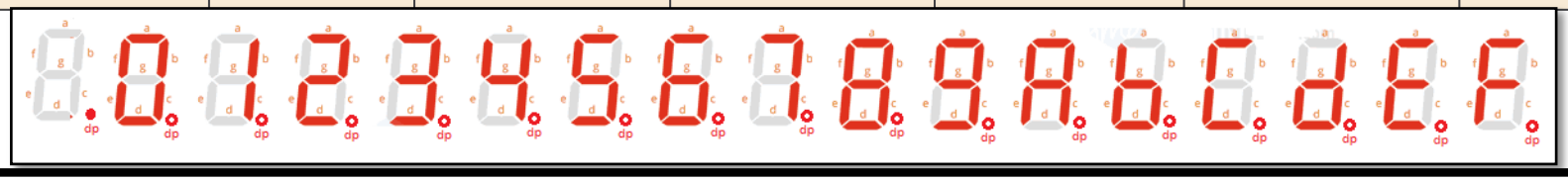


Typical 7-Segment LED | 7-Segment's name and the Dot | Common Anode Configuration | Common Cathode Configuration

PROGRAMMING GPIO PINS

Example-7 - Display 0 to 9 in 7-Segment display

Hexa Value	Decimal Digit	Individual Segments Illuminated for **Common Cathode configuration**							
		dp (PC_7)	g (PC_6)	f (PC_5)	e (PC_4)	d (PC_3)	c (PC_2)	b (PC_1)	a (PC_0)
0x3F	0	0	0	1	1	1	1	1	1
0x06	1	0	0	0	0	0	1	1	0
0x5B	2	0	1	0	1	1	0	1	1
0x4F	3	0	1	0	0	1	1	1	1
0x66	4	0	1	1	0	0	1	1	0
0x6D	5	0	1	1	0	1	1	0	1
0x7D	6	0	1	1	1	1	1	0	1
0x07	7	0	0	0	0	0	1	1	1
0x7F	8	0	1	1	1	1	1	1	1
0x6F	9	0	1	1	0	1	1	1	1

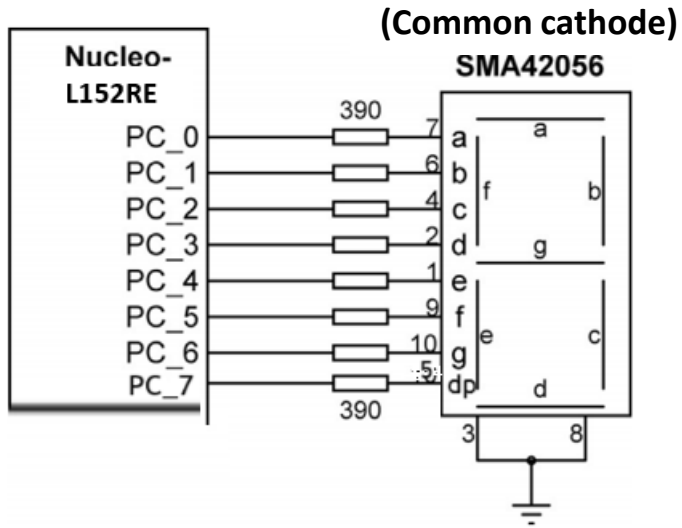


PROGRAMMING GPIO PINS

Program:

```
#include "mbed.h"
BusOut display(PC_0,PC_1,PC_2,PC_3,PC_4,PC_5,PC_6,PC_7);
// segments a, b, c, d, e, f , g, h, dp
int main()
{
while(1)
{
for(int i=0; i<=9; i++)
{
switch (i)
{
case 0: display=0x3F; break;
case 1: display=0x06; break;
case 2: display=0x5B; break;
case 3: display=0x4F; break;
case 4: display=0x66; break;
case 5: display=0x6D; break;
case 6: display=0x7D; break;
case 7: display=0x0F; break;
case 8: display=0x7F; break;
case 9: display=0x6F; break;
}
wait (0.2);
}
}
}
```

Connection Diagram:



PROGRAMMING GPIO PINS

Exercise-2 – Letter counter

Use the slotted opto-sensor (IR sensor), push-button switch and one seven-segment LED display to create a simple letter counter. Increment the number on the display by one every time a letter passes the sensor. Clear the display when the push button is pressed. Use an 7-Segment LED to display count from 0 to 9.

PWM GENERATION

PWM GENERATION

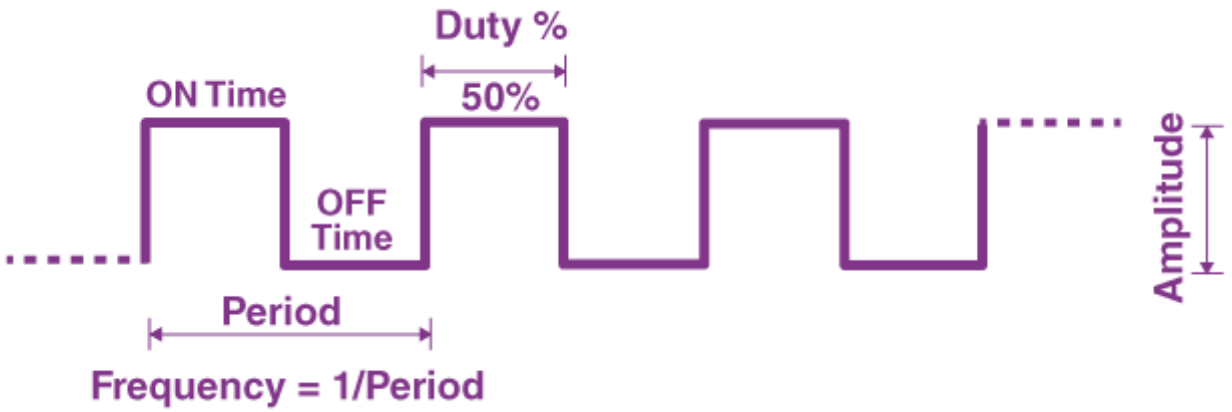
- Pulse width modulation(PWM) is a commonly used control technique that generates analog signals from digital devices such as microcontrollers.
- By rapidly switching between ON and OFF state, the digital signal is modified to imitate an analog signal.
- PWM signal is used in applications such as controlling motors, lights, actuators, etc.,
- In STM-32 NUCLEO-L152RE development board, PWM signal is generated as one of the function of timers and there are 18 PWM supports pins are available.
- The output voltage of the PWM signal will be the percentage of the duty cycle. For example,
  - For a 100% duty cycle, if the operating voltage is 5 V then the output will also be 5 V.
  - If the duty cycle is 50%, then the output voltage will be 2.5 V.

PWM GENERATION

- PWM signal stays "ON" for a given time and stays "OFF" for a certain time, the percentage of time for which the signal remains "ON" is known as the duty cycle.

- If the signal is always "ON," then the signal must have a 100 % duty cycle. The formula to calculate the duty cycle is given as follows:

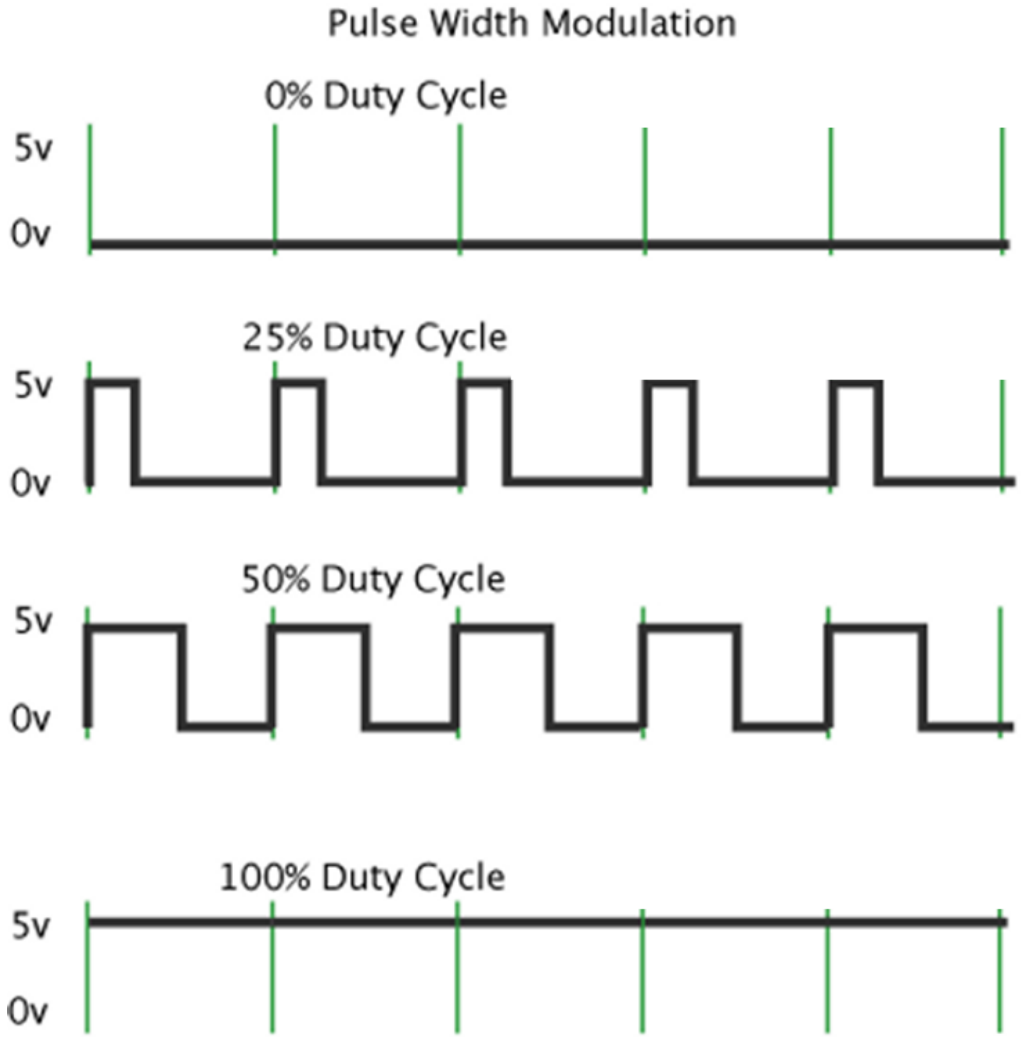
Duty Cycle= Ton / (Ton +Toff) * 100 (percentage)

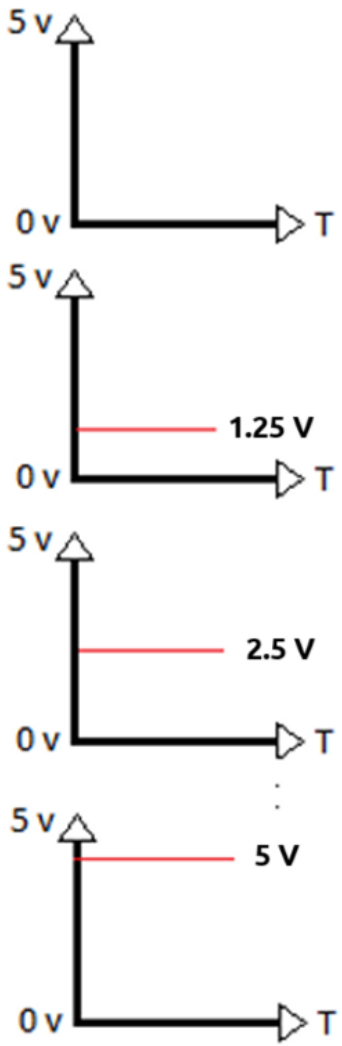


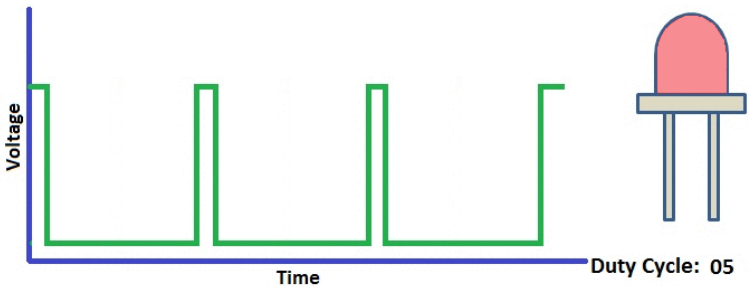
Period = Ton+Toff

Period = 1/Frequency

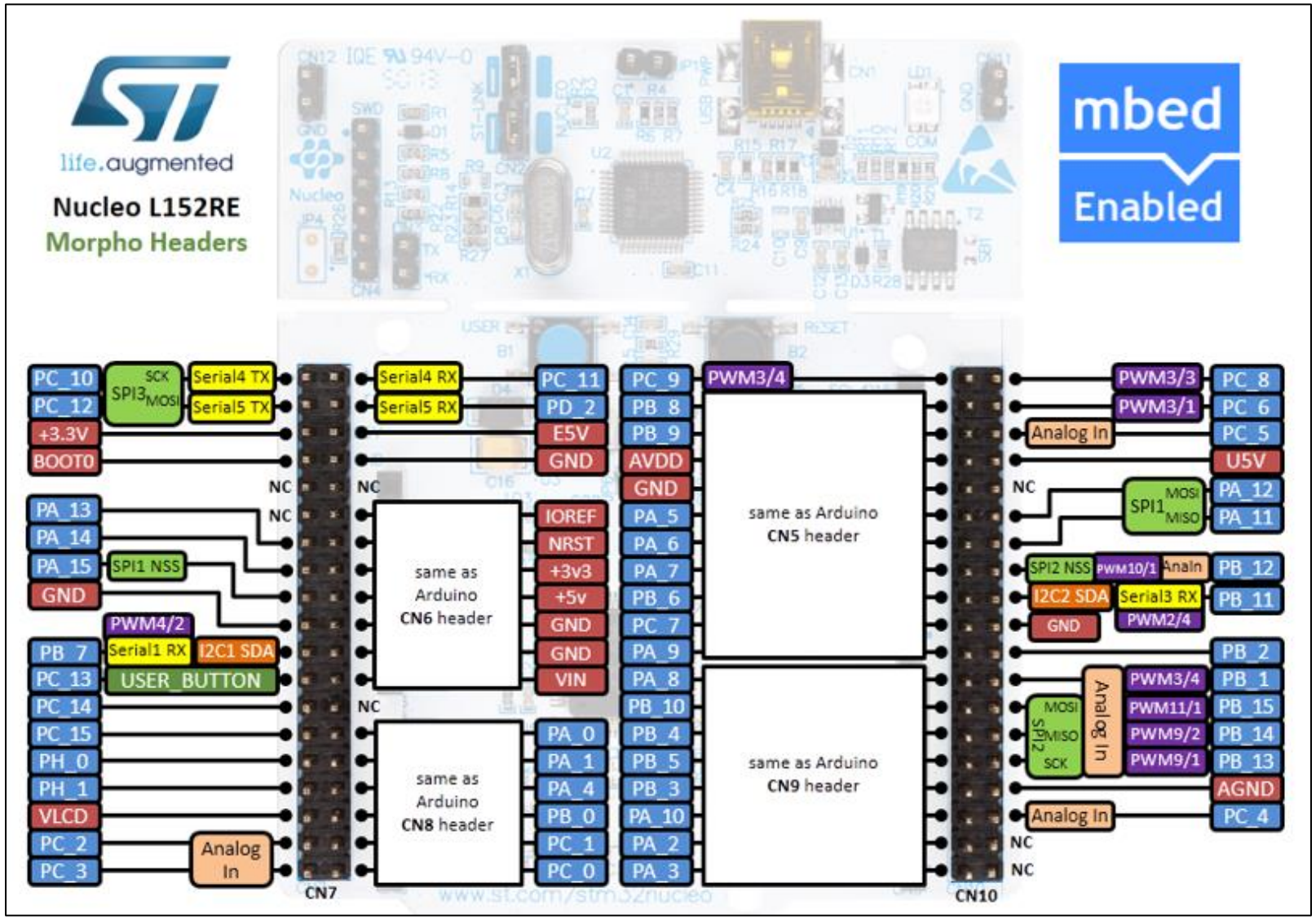
PWM GENERATION







STM32 Nucleo-64 - Pin Details (Morpho Headers)



Pins Legend

- XXX — Power and control pins
- PX_Y — MCU pin without conflict
- XXX — LEDs and Buttons
- XXX — AnalogIn (ADC)
- XXX — Serial pins (USART/UART)
- XXX — SPI pins
- XXX — I2C pins
- XXX — PWMOut pins (TIMER n/c[N])
  n = Timer number c = Channel
  N = Inverted channel

Image credit: arm MBED, Ref. URL : https://os.mbed.com/forum/electronics/topic/15346/?page=1#comment-52189

PWM GENERATION

API required:

Functions	Usage
PwmOut	Create a PwmOut object connected to the specified pin
write	Set the output duty cycle, specified as a normalized float (0.0—1.0)
read	Return the current output duty cycle setting, measured as a normalized float (0.0—1.0)
period	Set the PWM period, specified in seconds (float), keeping the duty cycle the same.
period_ms	Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same.
period_us	Set the PWM period, specified in microseconds (int), keeping the duty cycle the same.
pulsewidth	Set the PWM pulse width, specified in seconds (float), keeping the period the same.
pulsewidth_ms	Set the PWM pulse width, specified in milliseconds (int), keeping the period the same.
pulsewidth_us	Set the PWM pulse width, specified microseconds (int), keeping the period the same.
mbed-defined operator: =	An operator shorthand for write()

https://os.mbed.com/docs/mbed-os/v6.16/apis/pwmout.html

PWM GENERATION

Example-1 Controlling LED brightness with 75% duty cycle

Write a C++ program with mbed APIs to control the brightness of the LED using a PWM signal with duty cycle 75%. Assume PWM period as 2 Seconds. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"
PwmOut led(PC_8);

int main()
{
    // specify period first
    led.period(2.0f);      // 4 second period
    led.write(0.750f);      // 75% duty cycle, relative to period
   // led = 0.75f;          // shorthand for led.write()
   //led.pulsewidth(1.5);   // alternative to led.write, set duty cycle time in seconds
    while (1);
}
```

PWM GENERATION

Example-2 - Increasing/decreasing LED brightness using variable duty cycle

Write a C++ program with mbed APIs to gradually increase and decrease LED brightness using PWM signal with variable duty cycle. Assume the LED is connected to PC_8 PWM pin, time period of the PWM signal as 10ms, increment/decrement duty cycle value is 10% and delay between each increment/decrement duty cycle is 0.5s. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

PWM GENERATION

Program:

```
#include "mbed.h"
PwmOut led(PC_8);
int main ()
{
led.period_ms(10);
led= 0.0f;
while (true)
{
for (float val = 0.0f; val < 1.0f; val += 0.1f )
{
led = val;
wait(0.5f);
}
for (float val = 1.0f; val > 0.0f; val -= 0.1f)
{
led = val;
wait(0.5f);
}
}
}
```

PWM GENERATION

Example-3 - Generate a music note SA-RE-GA-MA-PA-THA-NEE using PWM

Write a C++ code with mbed APIs to generate a music note SA-RE-GA-MA-PA-THA-NEE on the buzzer using PWM signal. Frequency for each music tone is listed below.

Notes	Frequency in Hz
Sa	240
Re	270
Ga	300
Ma	337.5
Pa	360
Dha	400
Ni	450

Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

PWM GENERATION

Program:

```
#include "mbed.h"
PwmOut speaker(PC_8);
int main()
{
    speaker.period(1.0/240.0); // 240hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
    speaker.period(1.0/270.0); // 270hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
    speaker.period(1.0/300.0); // 300hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
    speaker.period(1.0/337.0); // 337hz period
    speaker =0.5; //50% duty cycle - max volume
    speaker.period(1.0/360.0); // 360hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
   speaker.period(1.0/400.0); // 400hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
   speaker.period(1.0/450.0); // 450hz period
    speaker =0.5; //50% duty cycle - max volume
    wait(0.5);
    speaker =0.0;
}
```

PWM GENERATION

Exercise-1 - Warning signals system for Automotive

Write a C++ program with mbed APIs to design a ***warning signals system for Automotive*** with the following logic.

- Use 4 switch and one buzzer to generate four warning signals such as indicator signal, horn sound, seat belt warning and reversing signal.
- Assume each of these warning signals must have 50% duty cycle and 0.5s delay between their ON and OFF state.
- Activate the respective warning signal as per the following switch status,
  - If Switch-1 is HIGH, generate a indicator signal (2 Hz)
  - If Switch-2 is HIGH, generate a Horn signal (400 Hz)
  - If Switch-3 is HIGH, generate a Seat belt warning signal (612 Hz)
  - If Switch-4 is HIGH, generate a Reversing signal (1000 Hz)
  - If all switches are LOW, the buzzer should be in OFF state

Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Hint: Use 4 Digital pins for switches to select the warning signal(Busin) & one buzzer to generate the different warning sound(PWM).

PWM GENERATION

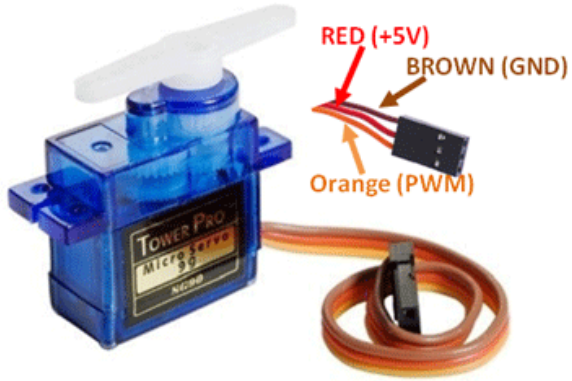
Introduction to Servo Motor

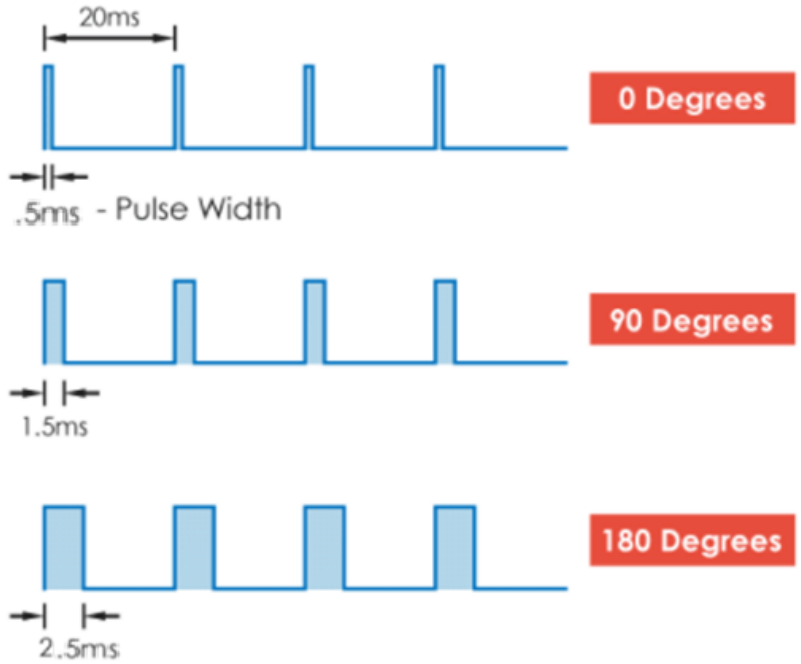
- A Servo Motor is a type of actuator that provides high precision control of linear or angular position.
- A simple servo motor consists of a small DC motor, a potentiometer for providing position feedback, a gear system for increased torque and a control system.
- Servo Motors are used in applications where very high precision motion is required like assembly robots, security cameras, solar tracking system etc.
- Typically, simple servo motors consists of three wires, colour coded as Red(supply), Brown(ground) and Orange (control signal).
- A servo motor is controlled by sending a series of pulses through the signal line. The frequency of the control signal should be 50Hz or a pulse of 20ms.

PWM GENERATION

Introduction to Servo Motor

- The control signal is usually a PWM signal, unlike PWM signal used to control the speed of DC Motor, it is used to determine the position of the servo motor and these type of servos can usually rotate 180 degrees
- Generally pulses with 1ms duration correspond to 0° degrees position, 1.5ms duration to 90° and 2.5ms to 180° (may vary with different models) .





PWM GENERATION

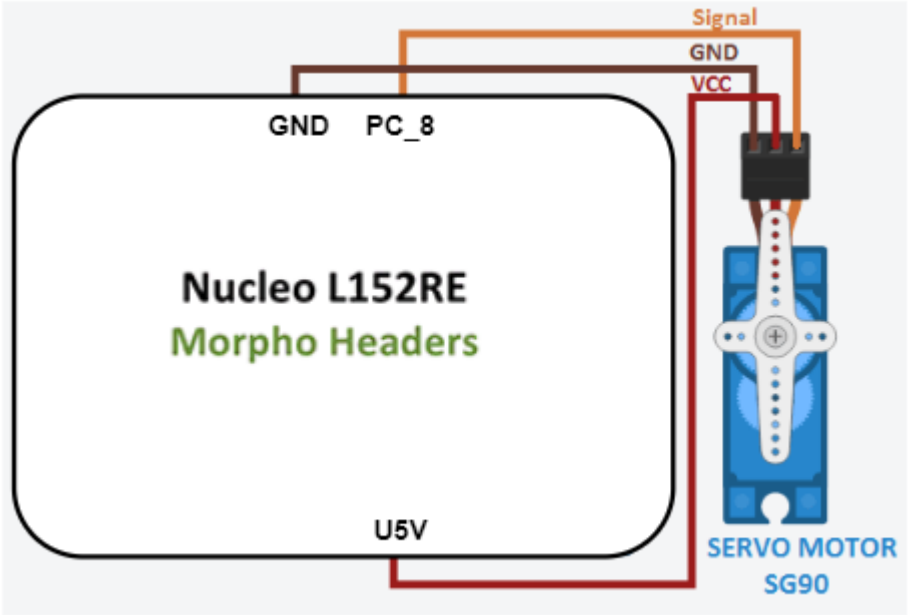
Example-4 - Controlling the servo motor at one degree at a time

Write a program to control the servo motor by rotating slowly from 0 degrees to 180 degrees, 45 degree at a time. When the motor has to be rotated 180 degrees, it will return to the initial position. Implement and verify this logic on STM32 board.

Program

```
#include "mbed.h"
PwmOut PWM1(PC_8);
int main() {
while(1){
PWM1.period_ms(20);
PWM1.pulsewidth_us(500);
wait(1);
PWM1.pulsewidth_us(1000); // 45 degree
wait(1);
PWM1.pulsewidth_us(1500); // 90 degree
wait(1);
PWM1.pulsewidth_us(2000); // 135 degree
wait(1);
PWM1.pulsewidth_us(2500);  // 180 degree
wait(1);
}
}
```

Connection diagram



ANALOG TO DIGITAL CONVERTER (ADC)

ANALOG TO DIGITAL CONVERTER (ADC)

Introduction to USB Virtual Serial Port

- The mbed Microcontroller can communicate with a host PC through a "USB Virtual Serial Port" over the same USB cable that is used for programming.

- This enables you to:
  - Print out messages to a host PC terminal
  - Read input from the host PC keyboard
  - Communicate with applications and programming languages running on the host PC that can communicate with a serial port, e.g. python, java and so on.

- The Serial Interface can be used on supported pins and USBTX/USBRX

- Note that USBTX/USBRX are not pins; they represent the pins that route to the interface USB Serial port so you can communicate with a host PC.

ANALOG TO DIGITAL CONVERTER (ADC)

Example-1 - Controlling on-board LED using USB Virtual serial port

Write a C++ code with mbed APIs to receive a character (H) from PC via host terminal application (Tera term) & switch ON the LED. For all other character LED1 must be in OFF state. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

API required:

- Serial Identifier(USBTX, USBRX) - Used to access "USB Virtual Serial Port"
  - Identifier: User defined name to access this USB Virtual Serial Port
  - Examples: `Serial pc(USBTX, USBRX);` //Accessing USB Virtual Serial Port using the identifier name pc
- putc - Used to send a character from Nucleo board to host PC
- getc - Used to read a character from host PC to Nucleo board

ANALOG TO DIGITAL CONVERTER (ADC)

Program:

```
#include "mbed.h"
Serial pc(USBTX,USBRX); // tx, rx
DigitalOut myled(PC_8);
int main()
    {
    printf("Press a character:");
    while(1)
        {
        int y=pc.getc();
        printf("%c\n",y);
        if(y=='H')
        {
            myled=1;
            wait(0.2);
        }
        else
        {
            myled=0;
        }
        }
    }
```

ANALOG TO DIGITAL CONVERTER (ADC)

Example-2 - Increasing/Decreasing Brightness Using Serial Port

Write a C++ program with mbed APIs to gradually increase and decrease LED brightness by receiving a character ("i" for increase and "d" for decrease) from PC via host terminal application (Tera term) using PWM signal with variable duty cycle. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"
Serial pc(USBTX,USBRX);
PwmOut led(PC_8);
float brightness=0.0;
int main() {
while(1)
{
//pc.printf("Initialising LED....!!");
pc.printf("Enter the command to Increase/Decrease the brightness(i/d):\r\n");
char c = pc.getc();
if((c=='i'))
{
pc.printf("Increasing the brightness by 10%: \r\n");
brightness+=0.1;
led = brightness;
}
if((c=='d')&&(brightness>0.0))
{
pc.printf("Decreasing the brightness by 10%: \r\n");
brightness-=0.1;
led = brightness;
}
}
}
```

- The default period is 0.020s, and the default pulse width is 0.

ANALOG TO DIGITAL CONVERTER (ADC)

Introduction to ADC

- An Analog to Digital Converter(ADC) is a electronic circuit that converts analog signal into digital form which could be understood by a digital device
- The list of possible analog input signals is endless such as s light, temperature, speed, pressure etc.
- ADC effectively measures the analog input voltage and gives a binary output number proportional to its size.
- ADC types: Successive Approximation, Dual slope, delta-sigma, flash type etc.,
- ADC has maximum and minimum permissible input values.

ANALOG TO DIGITAL CONVERTER (ADC)

Introduction to ADC

- Not all the pins on a microcontroller have the ability to do analog to digital conversions.
- On the STM32 Nucleo L152RE board, these pins have an 'Analog In' indicate these pins can read analog voltage.
- There are more than 20 analog channels available supporting Analog In function
- The ADC on the Nucleo is a 12-bit ADC meaning it has the ability to detect 4096 (2^12) discrete analog levels.
- An input of 0 volts would result in a decimal 0 and an input of 5 volts would give the maximum of 4096.

ANALOG TO DIGITAL CONVERTER (ADC)

Introduction to ADC

- The resolution for an ADC is the smallest distinguishable change in analog input that causes the digital output to change.

$$Resolution = \frac{V_r}{2^n}$$

- Nucleo ADC is 12-bit. This leads to a resolution of 5/4096, or 1.2 mV.
- ADC conversion

$$D = \frac{V_i}{V_r} \times 2^n$$

$$D = \frac{2.12V}{5} \times 4096 = 1737$$

Where, D the digital output value, V_i is the input voltage, V_r the reference voltage, n no. of bits in the output

- The output binary number D is an integer, for an n-bit number can take any value from 0 to $(2^n - 1)$.

ANALOG TO DIGITAL CONVERTER (ADC)

API required:

- AnalogIn Identifier(PinName) – Used to access ADC of the specified pin name.
  - Identifier: User defined name to access the specified ADC pin
  - PinName: Name of the pins as specified in the Nucleo board for ADC pin

- read() - Read the input voltage, represented as a float in the range 0 to 1. For example, if you have a 3.3V system and the applied voltage is 1.65V, then AnalogIn() reads 0.5 as the value.

- read_voltage () - Read the input voltage in volts

- get_reference_voltage () - Gets this AnalogIn instance's reference voltage.

- operator float () - An operator shorthand for read()

ANALOG TO DIGITAL CONVERTER (ADC)

Example-3 - Increasing/Decreasing Brightness Using Potentiometer

Write a C++ program with mbed APIs to gradually increase and decrease LED brightness by changing the position of the Potentiometer via ADC. Also print it's voltage value on serial monitor. Use Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"
Serial pc(USBTX,USBRX); //tx,rx for serial communication
AnalogIn ain(PC_3); //API for analog input(4 pins can be utilised)
PwmOut PWM1(PC_8);
int main(){
while(1){
float value = ain*5;
pc.printf("The reading of pot:%f",value);
wait(1);
PWM1.period(0.01);//Set PWM period to 10ms
PWM1 = ain;
}
}
```

ANALOG TO DIGITAL CONVERTER (ADC)

Exercise-1 - Battery level indicator

Write a C++ program with mbed APIs to design a battery level indicator system using potentiometer and LEDs. The system must display the different level of the voltage with the help of 5 LEDs as per following conditions.

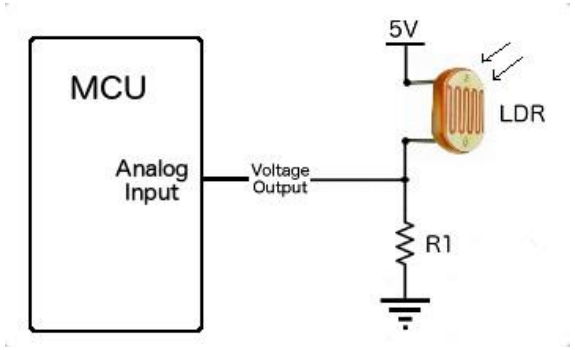
- If the voltage is between 0 to 1V glow LED1 and display “0 to 1V” in serial monitor
- If the voltage is between 1 to 2V glow LED1 and LED2 “1 to 2V” in serial monitor
- If the voltage is between 2 to 3V glow LED1 to LED3 “2 to 3V” in serial monitor
- If the voltage is between 3 to 4V glow LED1 to LED4 “3 to 4V” in serial monitor
- If the voltage is between 4 to 5V glow LED1 to LED5 “4 to 5V” in serial monitor

Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

ANALOG TO DIGITAL CONVERTER (ADC)

Introduction to LDR

- Light Dependent Resistor (LDR) or Photoresistor, is a passive electronic component, basically a resistor which has a resistance that varies depending of the light intensity.
- The resistance is very high in darkness but when there is light that falls on the LDR, the resistance is falling down to low value.
- LDRs are very useful in many electronic circuits, especially in alarms, switching devices, clocks, street lights and more.



ANALOG TO DIGITAL CONVERTER (ADC)

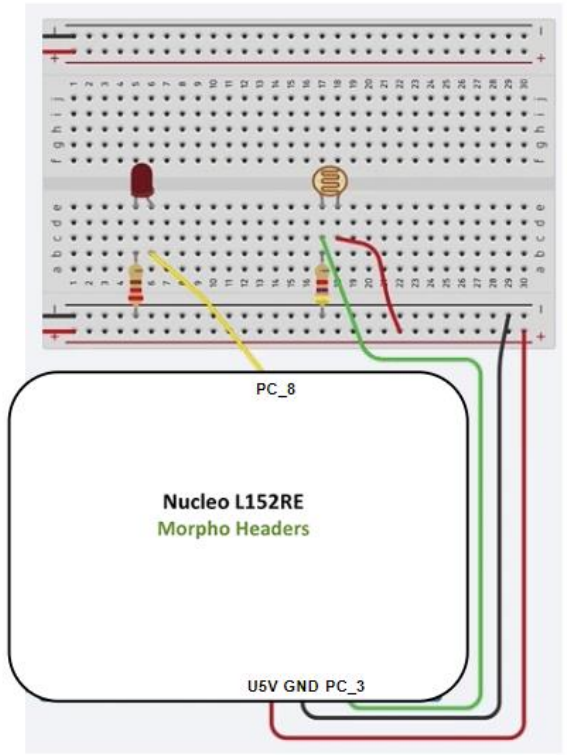
Example-4 - Auto intensity street light controller

Write a program to design auto intensity street light controller. This system helps the street light to get switched on automatically as per surrounding brightness. For example, sometimes when the weather become hazy its quite difficult to see anything then at that point this auto intensity street light gets switched on based on present lighting condition. Implement and verify this logic on STM32 board..

Program

```
#include "mbed.h"
PwmOut PWM1(PC_8);
AnalogIn Ain(PC_3);
int main()
{
while (1)
{
PWM1.period(0.010); // set PWM period to 10 ms
PWM1=1-Ain;   //Analog in value becomes PWM duty
wait(0.1);
}
}
```

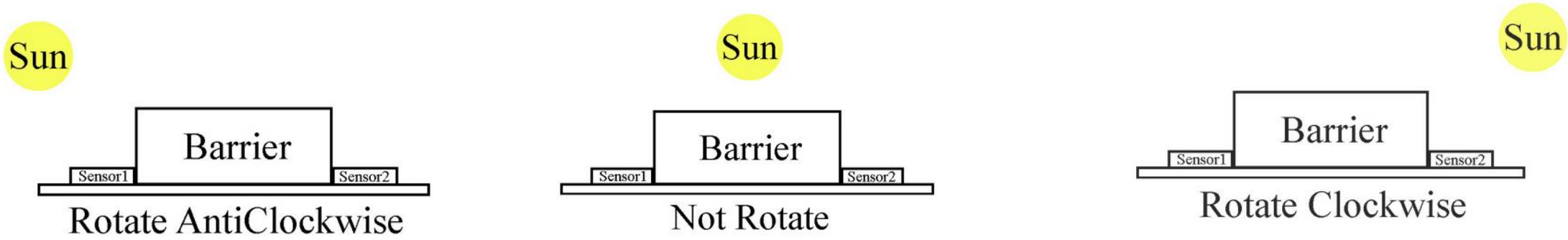
Connection diagram



ANALOG TO DIGITAL CONVERTER (ADC)

Exercise-2 - Solar tracking system

Write a program to design a solar tracking system for harvesting solar energy efficiently by the solar panels. This system is constructed by fitting two LDRs, angled away from each other by around 90, to a servo. Continuously read the light value sensed by the two LDRs and rotates the servo so that each is receiving equal light. As a sun-tracking system will be located to track the sun from sunrise to sunset, i.e. not more than 180. Also, display both LDR values and present Servo motor position on serial window. Implement and verify this logic on STM32 board.



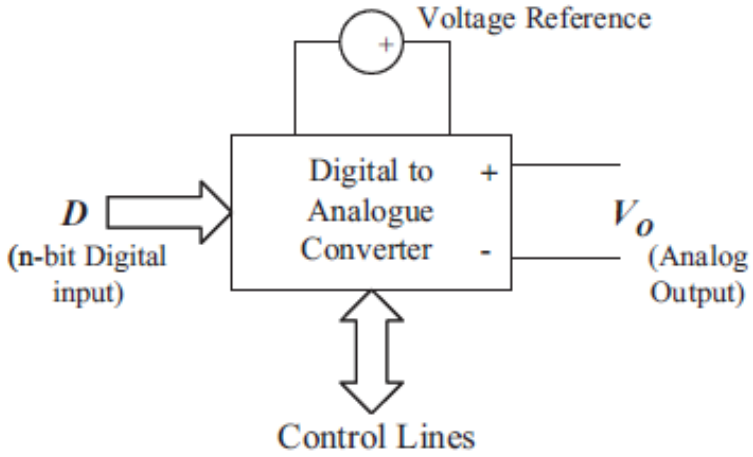
DIGITAL TO ANALOG CONVERTER (DAC)

DIGITAL TO ANALOG CONVERTER (DAC)

Introduction to DAC

- In STM32 Nucleo L152RE, the two 12-bit buffered DAC channels (PA_4 & PA_5) can be used to convert digital signals into analog voltage signal outputs.

- DAC has a digital input, represented by D, and an analog output, represented by Vo.



$$V_o = \frac{D}{2^n} V_r$$

- Here, Vr is the value of the voltage reference, D is the value of the binary input word, n is the number of bits in that word, and Vo is the output voltage.

DIGITAL TO ANALOG CONVERTER (DAC)

Introduction to DAC

Application programming interface summary for mbed analog output.

Functions	Usage
`AnalogOut`	Create an AnalogOut object connected to the specified pin
`write`	Set the output voltage, specified as a percentage (float)
`write_u16`	Set the output voltage, represented as an unsigned short in the range [0x0, 0xFFFF]
`read`	Return the current output voltage setting, measured as a percentage (float)
`mbed-defined operator: =`	An operator shorthand for write()

DIGITAL TO ANALOG CONVERTER (DAC)

Example-1 – Generate analog voltage and turn on On-board LED

Write a C++ program with mbed APIs to gradually increase analog output voltage on PA_4 and SWITCH ON on-board LED when output voltage is >0.5VCC. Also print it's voltage value on serial monitor. Use Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program:

```
#include "mbed.h"
AnalogOut aout(PA_4);
DigitalOut dout(LED1);
int main(void)
{
while (1)
 {
for (float i = 0.0f; i < 1.0f; i += 0.1f)
{
aout = i;
 printf("aout = %1.2f volts\n", aout.read() * 5);
dout = (aout > 0.5f) ? 1 : 0; // turn on the led if the voltage is greater than 0.5f * VCC
wait(1.0f);
}
}
}
```

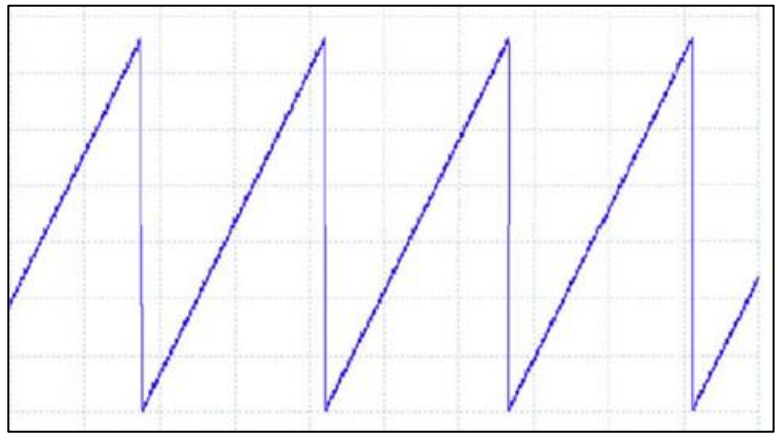
Note: By default "aout" takes a floating point number between 0.0 and 1.0 and outputs this to PA_4.

DIGITAL TO ANALOG CONVERTER (DAC)

Example-2 – Generate Saw tooth waveform on DAC output and View on oscilloscope

Program:

```
#include "mbed.h"
AnalogOut Aout(PA_4);
float i;
int main() {
while(1){
for (i=0;i<1;i=i+0.001)
{ // i is incremented in steps of 0.001
Aout=i;
wait(0.001); // wait 1 millisecond
}
}
}
```



Exercise-1 – Generate Triangular waveform (one that counts down as well as up) on DAC output and View on oscilloscope

LIQUID CRYSTAL DISPLAY (LCD)

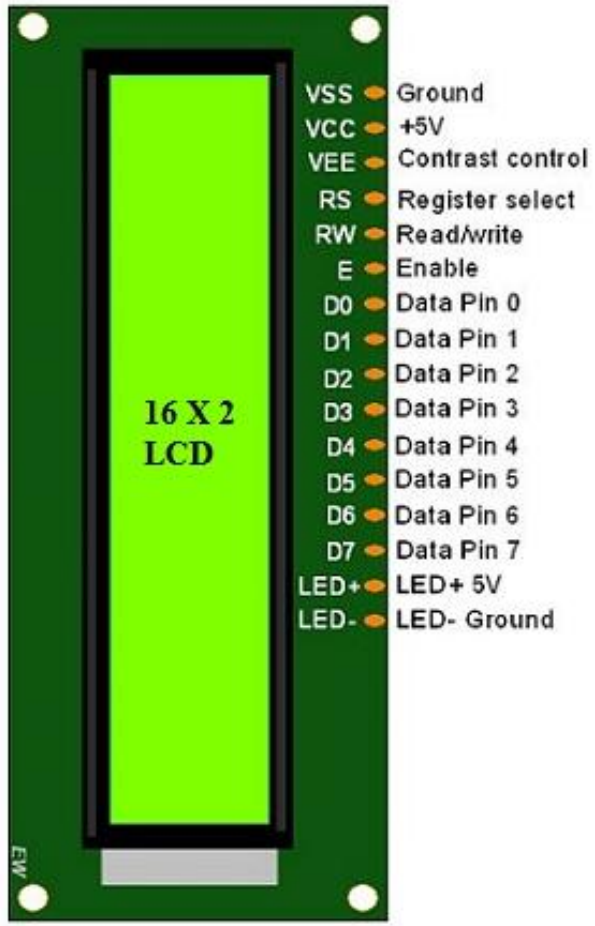
INTRODUCTION TO LCD

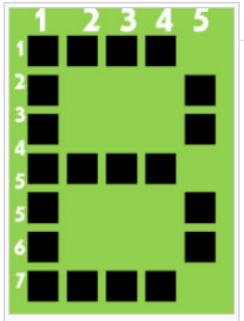
- LCD stands for Liquid Crystal Display and It is one kind of electronic display used in wide range of applications like CD/DVD players, Digital Watch etc.,
- The main benefits of LCD are inexpensive; simply programmable, and there are no limitations for displaying custom characters, simple animations, etc.
- The liquid crystals are made up of a part solid, part liquid substance that can be "twisted" by applying electrical voltage to them.
- In the normal state, the crystals inside the liquid are twisted, and light can pass through, once the crystals are subjected to an electrical current, they untwist, blocking the light and makes the portion of the screen black.

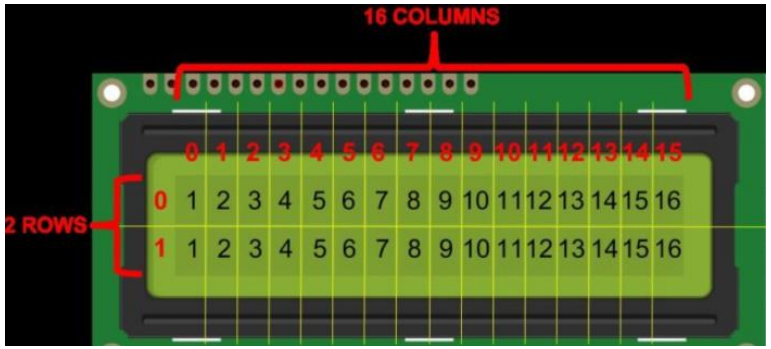
INTRODUCTION TO LCD

- Among different sizes of LCDs (16x1, 16x2, 16x4, 20x4), 16x2 LCD display is very basic module and is very commonly used in various devices and circuits.
- A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix.
- The 16x2 intelligent alphanumeric dot matrix display is capable of displaying 224 different characters and symbols.
- This LCD has two registers, namely, Command and Data.
  - Command register stores various commands given to the display.
  - Data register stores data to be displayed.

INTRODUCTION TO LCD







- Vss- Ground pin
- Vcc- Supply (+5V)
- VEE- Contrast pin (0.4 to 0.9V)
- RS- Register select (1- data register, 0 – command register)
- R/W - Read/ Write mode (1-Read mode, 0- Write mode)
- E- Enable the LCD module (high to low at this pin enable LCD)
- DB0-DB7- Data pins (command and data are send to display)
- LED+ & LED- - Anode and Cathode of backlight

TextLCD Library

- This library allows an Nucleo board to control LiquidCrystal displays (LCDs) based on the Hitachi HD44780 chipset in 4-bit mode
- In Nucleo board TextLCD library is used as #include "TextLCD.h"
- The mbed TextLCD library performs the laborious LCD setup routine for us also tells the LCD object which pins are used for which functions.
- To use the TextLCD library - create a named TextLCD object.
  - Syntax: TextLCD lcd(rs, enable, d4, d5, d6, d7);
  - Example: TextLCD lcd(PC_0,PC_1,PB_0,PA_4, PA_1, PA_0);
- lcd.printf("Text to print")- the character to write to the display

For more details Refer: https://os.mbed.com/users/simon/code/TextLCD/

TextLCD Library

Functions	Description
TextLCD (PinName rs, PinName e, PinName d4, PinName d5, PinName d6, PinName d7)	Create a TextLCD interface.
putc (int c)	Write a character to the LCD.
printf (const char *format,...)	Write a formated string to the LCD.
locate (int column, int row)	Locate to a screen column and row.
cls ()	Clear the screen and locate to 0,0.

For more details Refer: https://os.mbed.com/users/simon/code/TextLCD/docs/tip/classTextLCD.html

Procedure to add TextLCD Library

Method-1 for online compiler:

- Download LCD library from https://os.mbed.com/users/simon/code/TextLCD/ as zip file and extract files
- Add "TextLCD.cpp" and "Text.h" files into project created in Keil studio by right clicking the project folder and select "upload files…" option
- Now Check whether "TextLCD.cpp" and "TextLCD.h" files included in your project folder or not

Method-2 for offline compiler:

- In Keil studio, right clicking the project folder and select "Add Mbed Library…" option and provide the link https://os.mbed.com/users/simon/code/TextLCD/ in URL field then click next
- In the net pop window select "default" under "Release, branch, or tag" field and click finish
- Now Check whether "TextLCD.cpp" and "TextLCD.h" files included in your project folder or not

LIQUID CRYSTAL DISPLAY (LCD)

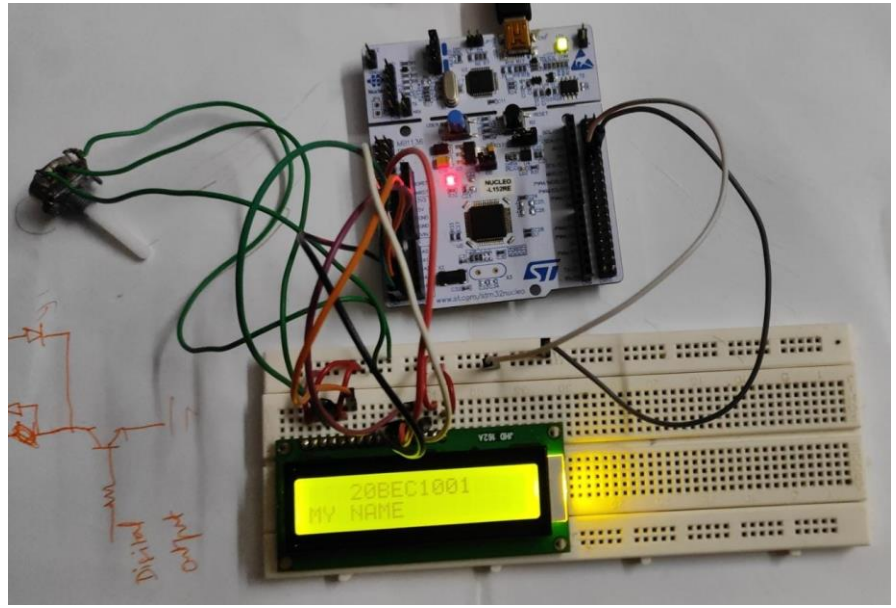
Example-1 - Display Name and Register number on LCD

Write a program for 16x2 LCD to display your register number (ex: “20BEC1001”) at row-0 3rd position and your name (ex:“ABCDEFGH”) at beginning of the row-1. Assume LCD operates in 4-bit with EN and RS active state. Design and verify this logic on Nucleo 152RE board using online Keil Studio platform.

Program

```
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(PC_0,PC_1,PB_0,PA_4, PA_1, PA_0); // rs, e, d4-d7
int main()
{
lcd.locate(3,0);
lcd.printf("20BEC1001\n");
lcd.locate(0,1);
lcd.printf("MY NAME\n");
}
```

Output



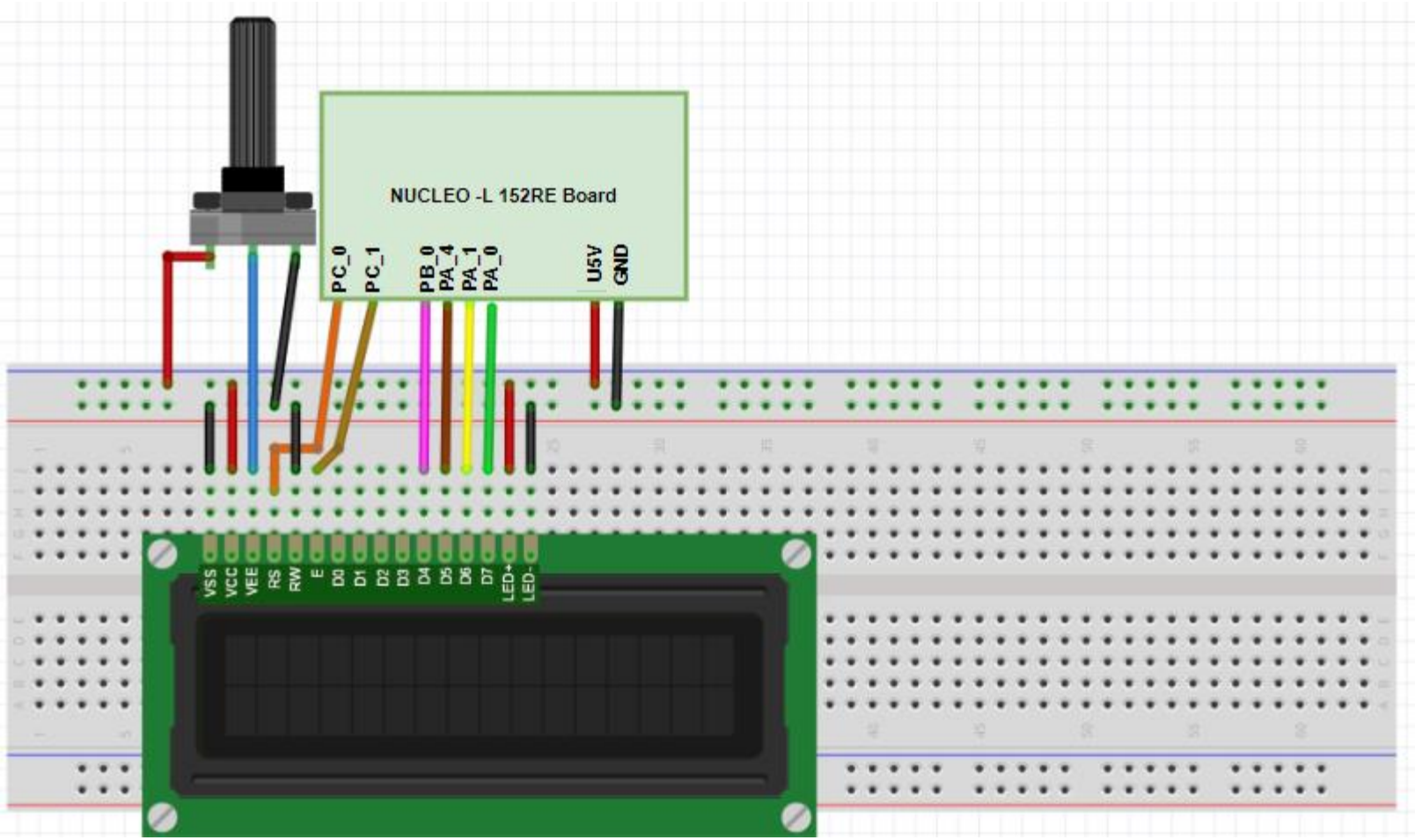
LIQUID CRYSTAL DISPLAY (LCD)

Example-1 - Display Name and Register number on LCD

Pin connections

LCD Pin No.	LCD pin Name	Nucleo pins
1	VSS	GND
2	VCC	U5V
3	VEE	Pot middle pin
4	RS	PC_0
5	RW	GND
6	E	PC_1
7	D0	not connected
8	D1	not connected
9	D2	not connected
10	D3	not connected
11	D4	PB_0
12	D5	PA_4
13	D6	PA_1
14	D7	PA_0
15	LED+	U5V
16	LED-	GND

Connection diagram



LIQUID CRYSTAL DISPLAY (LCD)

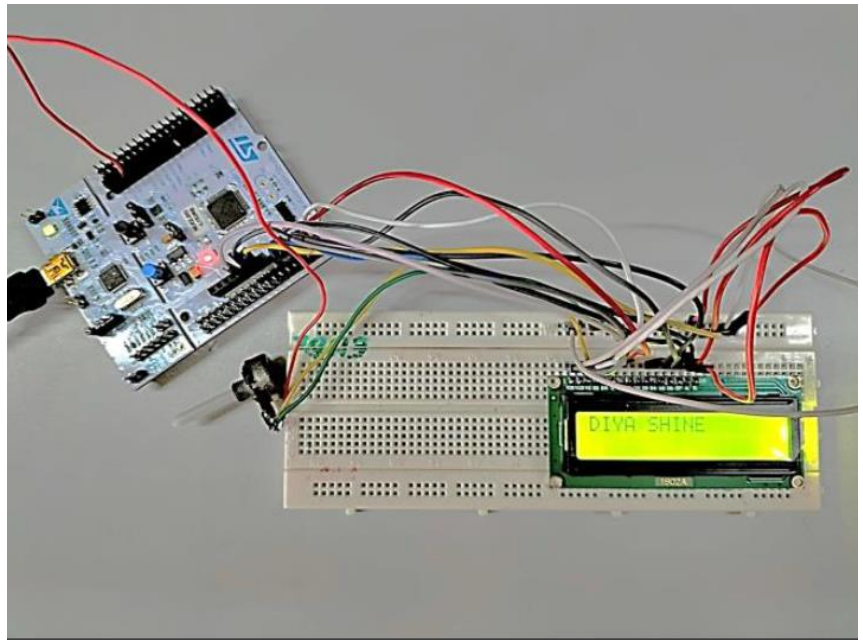
Example-2 Read the input from Serial Window and display it in LCD

Write a program to accepts serial input character via Teraterm software from a host computer and displays it on the LCD. Assume LCD operates in 4-bit with EN and RS active state. Design and verify this logic on Nucleo-152RE board using online Keil Studio platform.

Program

```
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(PC_0,PC_1,PB_0,PA_4,PA_1,PA_0);
Serial pc(USBTX,USBRX);
int main()
{
lcd.cls();
while(1) {
char in = pc.getc();
lcd.printf("%c",in);
}
}
```

Output



LIQUID CRYSTAL DISPLAY (LCD)

Example-3 - Display a POT value on the LCD

Write a program to display the potentiometer value with 4 decimal places on the LCD display on row-1 5th position. Assume LCD operates in 4 bit with EN and RS active state. Design and verify this logic on Nucleo 152RE board using online Keil Studio Platform.

Program

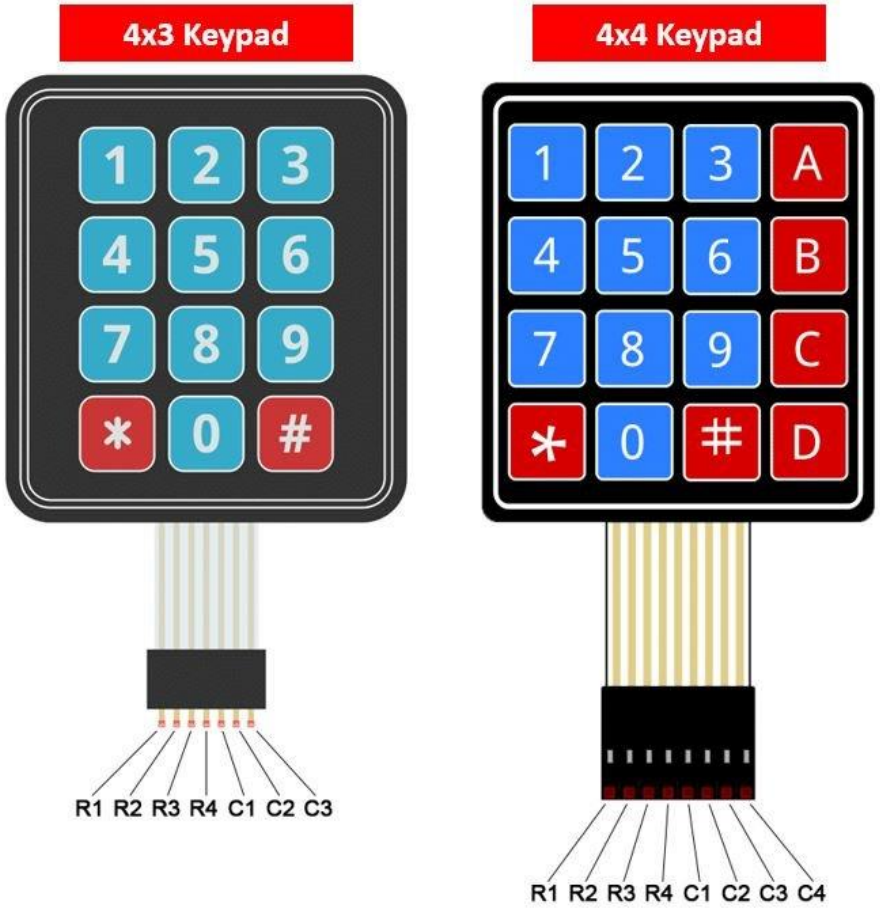
```
#include "mbed.h"
#include "TextLCD.h"
AnalogIn ain(PC_5);
TextLCD lcd(PC_0,PC_1,PB_0,PA_4,PA_1,PA_0); // rs,e,d4-d7
int main()
{
lcd.cls();
while(1) {
float value = ain*5;
lcd.locate(5,0);
lcd.printf("v = %0.3f",value);
wait(2);
lcd.cls();
wait(1);
}
}
```

KEYPAD

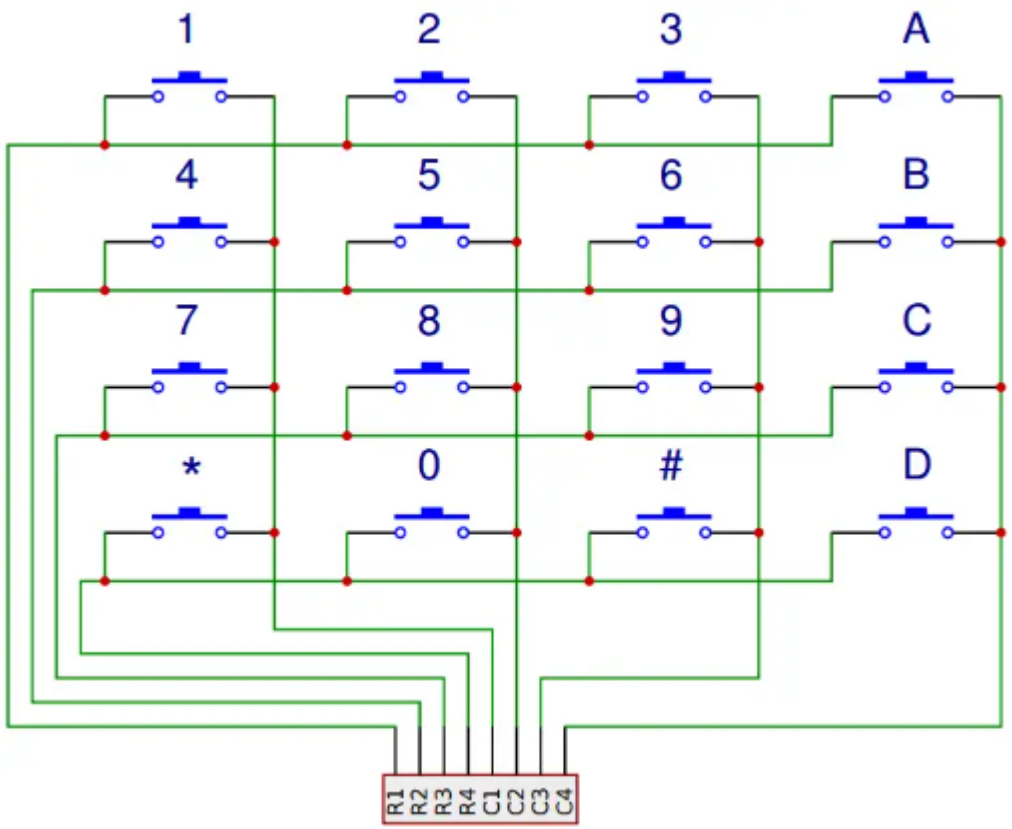
KEYPAD

- Keypads are input devices that are being widely used in many embedded system applications including ATM machines, door locks, and vending machines etc.,.
- Keypads are used to take input from the user as numbers or characters for further processing such as password, menu selection and navigating among different options.
- One of the most commonly used keypads is the matrix keypad with 4×4 or 4×3 buttons. The matrix keypad consists of pushbutton that are connected to the row and column lines.
- There is one pin for each column and one pin for each row. So the 4×4 keypad has 4+4=8 pins, while the 4×3 keypad has 4+3=7 pins.
- The column pins (Col 1–Col4) are connected to the microcontroller as the inputs pins and the rows pins (Row 1–Row 4) are connected to the output pins of the microcontroller.

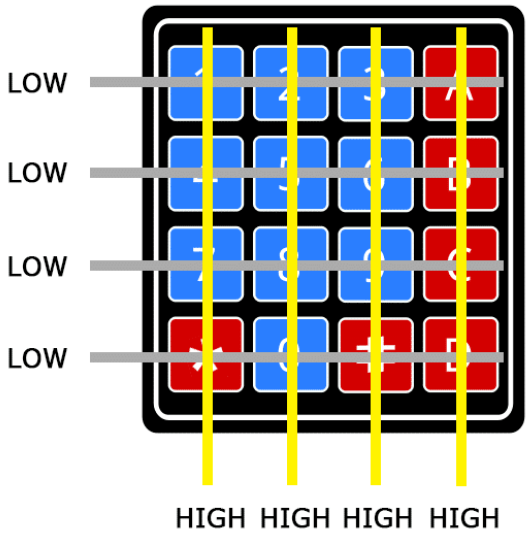
KEYPAD



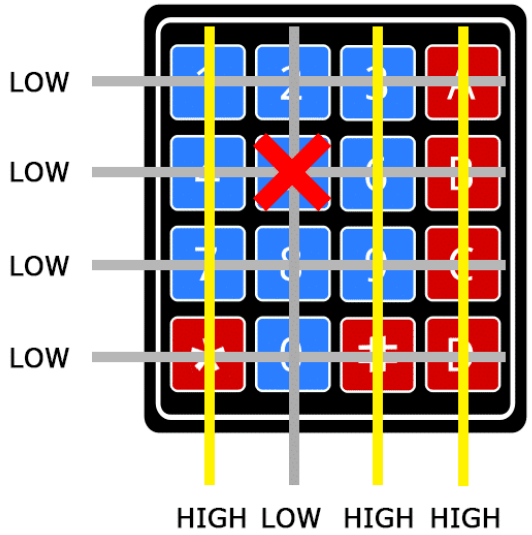
Internal structure of 4x4 keypad



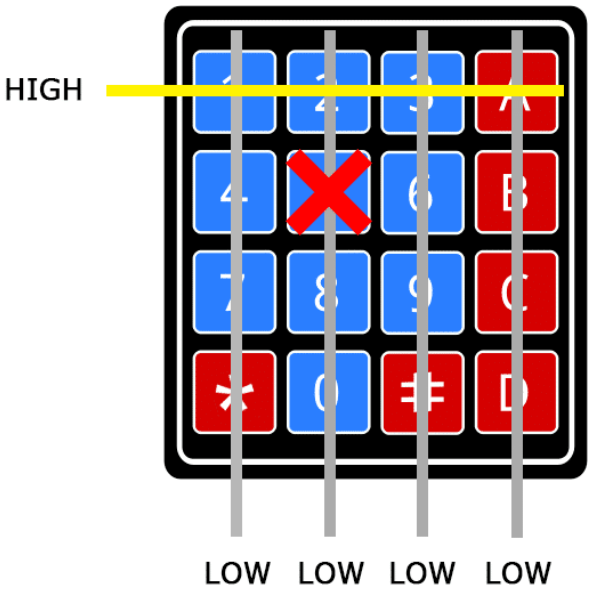
KEYPAD



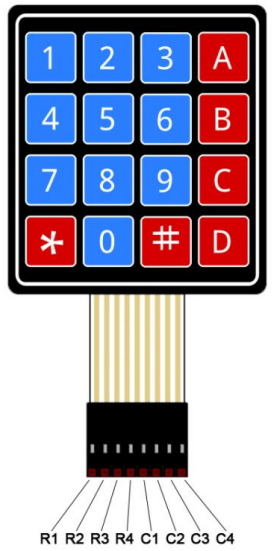
STEP-1

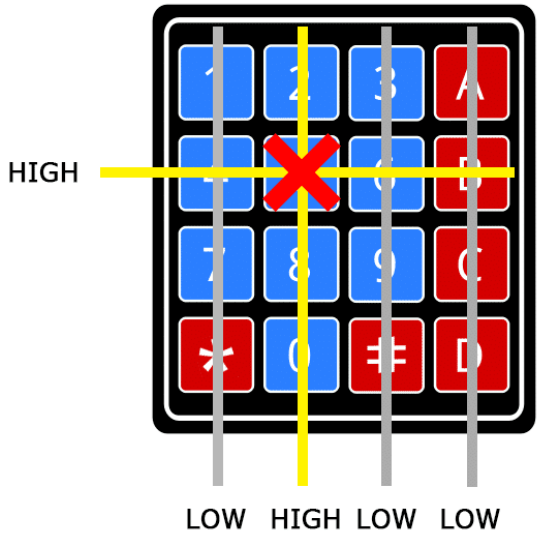


STEP-2



STEP-3





STEP-4

KEYPAD

- The status of each switch is determined by Scanning the rows or columns. If low logic is given to all the Rows and high logic is given to each Column.
- Keypad Scanning
  - Find Column number
    - When a switch/key is pressed, the corresponding row and column will get short, allowing current to flow between them. When the key '4' is pressed, for instance, column 1 and row 2 are connected.
    - The output of the corresponding column goes to go low.
    - Since we have made all the rows zero so this gives the column number of the pressed key.
  - Find Row number
    - After the detection of the column number, the controller set's all the rows to high.
    - Each row is one by one set to zero by the microcontroller and the earlier detected column is checked and obviously, it becomes zero.
    - The row due to which the column gets zero is the row number of the pressed key.

KEYPAD

- #include “keypad.h” is a mbed library for using 4x4 matrix style keypads with the STM32 Nucleo and supports multiple keypresses.
- Default keypad pin map is given below but pin map can be modified by editing the pin map details given in keypad.h header file.

```
const char keys[16] = {'1','2','3','A',
                       '4','5','6','B',
                       '7','8','9','C',
                       '*','0','#','D'
                      };
```

Functions	Description
Keypad identifier (PinName col0, PinName col1, PinName col2, PinName col3, PinName row0, PinName row1, PinName row2, PinName row3)	Create a 4x4 (col, row) or 4x4 keypad interface
chargetKey ()	Returns the letter of the pressed key
boolgetKeyPressed ()	Detects if any key was pressed.
void enablePullUp ()	Enables internal PullUp resistors on the columns pins.

For more details Refer: https://os.mbed.com/users/rlanghbv/code/KeypadLib/shortlog/

KEYPAD

Example-1 Read the input from keypad and display it in LCD & Serial monitor

Write a mbed C++ program to accepts input from a 4x4 keypad and displays it on the serial monitor and LCD. Assume LCD operates in 4-bit with EN and RS active state. Design and verify this logic on Nucleo 152RE board using online Keil Studio platform.

Procedure to add "Keypad" library in online compiler:

- Visit this link https://os.mbed.com/users/rlanghbv/code/KeypadLib/shortlog/ and click on "Import into Keil studio" option and copy the link (http://os.mbed.com/users/rlanghbv/code/KeypadLib/)
- In Keil studio, right clicking the project folder and select "Add Mbed Library…" option and paste the copied link in the URL field then click next
- In the net pop window select "default" under "Release, branch, or tag" field and click finish
- Now Check whether "keypad.cpp" and "keypad.h" files included in your project folder or not

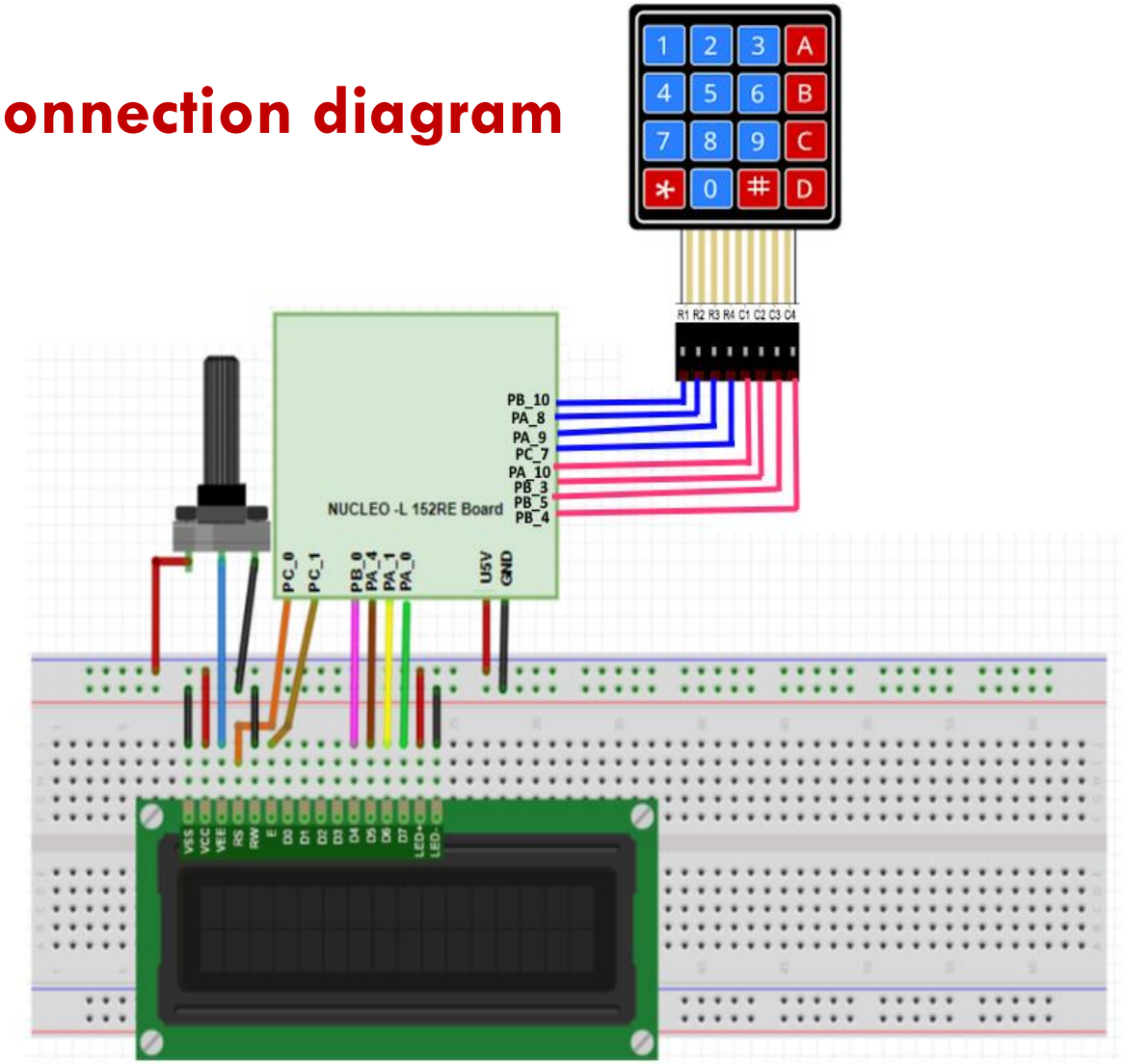
KEYPAD

Program

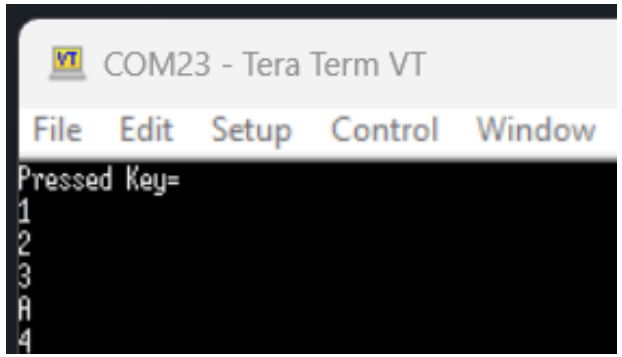
```
#include "mbed.h"
#include "keypad.h"
#include "TextLCD.h"

Serial pc(USBTX, USBRX);
TextLCD lcd(PC_0,PC_1,PB_0,PA_4,PA_1,PA_0);
Keypad keypad(PA_10,PB_3,PB_5,PB_4,PB_10,PA_8,PA_9,PC_7);
                //  c1   c2     c3   c4  r1   r2   r3   r4
char key;
int main() {
    keypad.enablePullUp(); //Enables internal PullUp resistors
                           //on the coloums pins to avoid key floating
    lcd.printf("Pressed Key=");
    pc.printf("Pressed Key=\n\r");
    while (1)
    {
     while (keypad.getKey() == '\0');
         key = keypad.getKey();
         lcd.locate (12,0);
         lcd.printf("%c",key);
         pc.printf("%c\r\n",key);
         wait(1);
    }
}
```

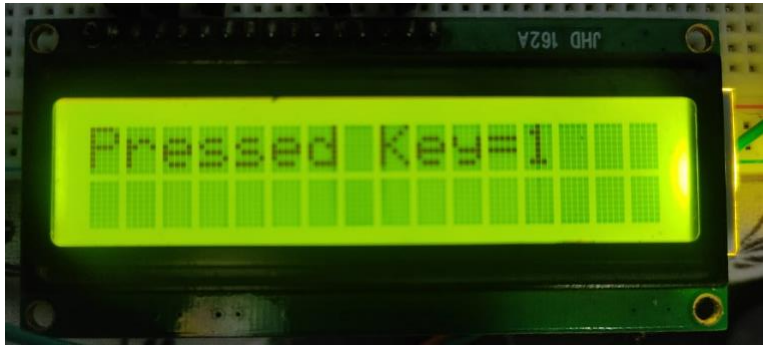
KEYPAD

Connection diagram



Output





KEYPAD

Example-2 - Password based door locking system

Write a mbed C++ program to design a password based door locking system in which the system accepts 4-digit password (last 4-digit of your reg. no) via keypad. Use the LCD display to message “Enter Password” on the first line and display * symbol on the second line of the LCD to represent every digit of the password entered. Check whether entered password matches with actual password, if matched activate the servo to open the door also display message “Correct Password” on LCD first line and “Door opening” message on the LCD second line. If password not matched, activate the buzzer also display message “Incorrect Password” on LCD first line and “Door can’t open” message on the LCD second line. Assume LCD operates in 4-bit with EN and RS active state. Design and verify this logic on Nucleo 152RE board using online Keil Studio platform.

KEYPAD

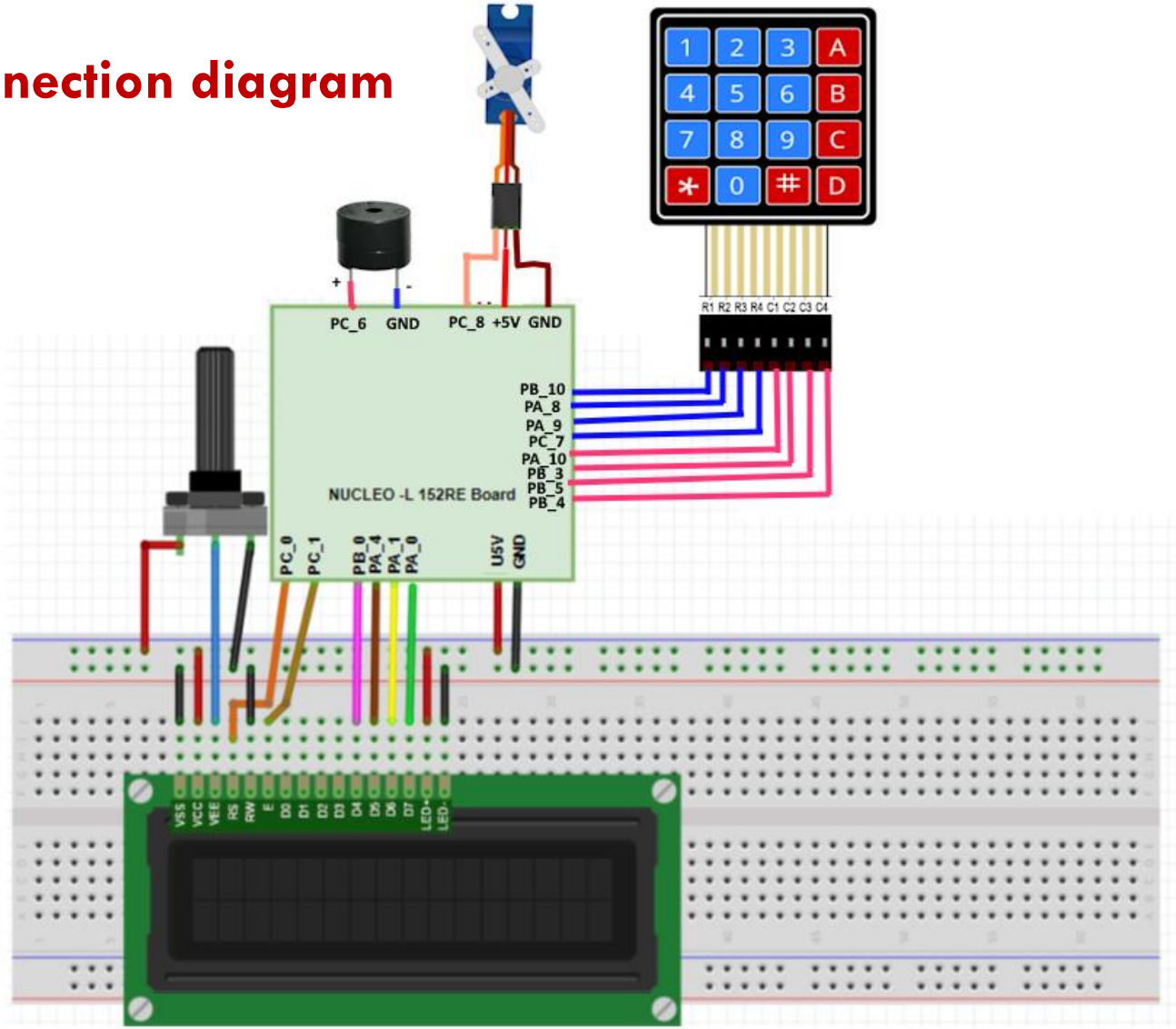
Program

```
#include "mbed.h"
#include "keypad.h"
#include "TextLCD.h"

TextLCD lcd(PC_0,PC_1,PB_0,PA_4,PA_1,PA_0);
Keypad keypad(PA_10, PB_3, PB_5,PB_4, PB_10,PA_8,PA_9,PC_7);
DigitalOut BUZZER(PC_6);
PwmOut SERVO(PC_8);
const char ACTUAL_PW[4] = {'1','2','3','4'};
char PW_ENTERED[4];
int count;

int main() {
    keypad.enablePullUp();
    while (1)
    {
        lcd.cls();
        lcd.locate (0,0);
        lcd.printf(" Enter Password ");
        SERVO.period_ms(20); //Set PWM period to 20ms
        SERVO.pulsewidth_us(500);
        BUZZER=0;
```

```
for(count=0; count <4; count++)
{
 while (keypad.getKey() == '\0');
 PW_ENTERED[count] = keypad.getKey();
 {
    lcd.locate (count,1);
    lcd.printf("*");
    wait(1);
 }
}
    if ((ACTUAL_PW[0] == PW_ENTERED[0]) && (ACTUAL_PW[1] == PW_ENTERED[1]) &&
        (ACTUAL_PW[2] == PW_ENTERED[2]) &&  (ACTUAL_PW[3] == PW_ENTERED[3]) )
 {
    lcd.cls();
    lcd.locate (0,0);
    lcd.printf("CORRECT PASSWORD");
    lcd.locate (0,1);
    lcd.printf("DOOR OPENING");
    SERVO.period_ms(20);
    SERVO.pulsewidth_us(1500);
    wait(3);
}
            else
            {
                lcd.cls();
                lcd.locate (0,0);
                lcd.printf("INCORRECT PW");
                lcd.locate (0,1);
                lcd.printf("DOOR CAN'T OPEN");
                BUZZER=1;
                wait(3);
            }
    }
}
```

KEYPAD

Connection diagram



Output

KEYPAD

Exercise-1: Home security system

Write a mbed C++program to design home security system that consists of three main modules (1) Intruder detection (2) Activate fence light during night time (2) password based door lock system.

1) The intruder detection system consists of PIR sensor interfaced with processing unit to detect and alert under human presence condition.

2) Automatically activate the fence lights during night time using LDR.

3) Password based door lock system uses numeric keypad to accept the password (last 4-digit of your reg. no) from user and LCD to display the message whether permission is granted or not. Upon receiving correct password signal, enable motor to open the door.

In case of password mismatch or intruder detection condition activate the buzzer. Assume LCD operates in 4-bit with EN and RS active state. Design and verify this logic on Nucleo 152RE board using online Keil Studio platform.

TIMERS/COUNTER & WATCHDOG TIMERS

TIMER/COUNTER & WATCHDOG TIMERS

- All modern microcontrollers are embedded with timer-counter modules and generally they are used for generating time bases, counting pulses (input capture mode) generating pulse width modulation (PWM) signals, triggering external devices (output capture mode) etc.,
- The ultra-low-power STM32L152RE devices include seven general-purpose timers, two basic timers.
- General-purpose timers (TIM2, TIM3, TIM4, TIM5, TIM9, TIM10 and TIM11) have all the features of a typical timer-counter module such as PWM generation, input capture, time-base generation and output compare
- Basic timers (TIM6 and TIM7) are mainly used for DAC trigger generation. These timers don't have I/O channels for input capture/PWM generation and so they are strictly used for time-base generation purposes.

TIMER/COUNTER & WATCHDOG TIMERS

Timer feature comparison

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM2, TIM3, TIM4	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
TIM5	32-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
TIM9	16-bit	Up, down, up/down	Any integer between 1 and 65536	No	2	No
TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No
TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

TIMER/COUNTER & WATCHDOG TIMERS

- Watch-Dog Timer (WDT) used to reset the system when a malfunction occurs.
  - In normal operation the WDT counts up(or down) while your program periodically resets it (this is called "kicking the dog“)
  - If your program crashed and did not kick WDT in time, it will overflow and generate a hardware reset signal to restarts the system and recovers it from the crash.

- STM32L152RE has two watchdog timers: Independent Watchdog (IWDG) and System Window Watchdog (WWDG).

- Window watchdog (WWDG)
  - Suited for applications which require the watchdog to react within an accurate time window.
  - The window watchdog is based on a 7-bit down counter that can be set as free-running.
  - It detects not only that a counter is overflowing, but also that its timing is earlier than expected.
  - since WWDG runs on the microcontroller’s clock, so it can be stopped.

TIMER/COUNTER & WATCHDOG TIMERS

- Independent watchdog (IWDG) is a counter that runs and resets itself when it overflows.
  - IWDG used to detect and resolve malfunctions resulting from software failures.
  - It triggers a restart sequence when it is not refreshed within the expected time window.
  - IWDG is based on a 12-bit down counter, can be used either as a watchdog to reset the device when a problem occurs or as a free-running timer for application timeout management.
  - The reason it is called an independent type is because the independent 37 kHz internal RC clock that runs the timer is separate from the main clock of the microcontroller.
  - Even if the microcontroller's clock stops due to some fault, it can still be reset if the IWDG is running.

TIMER/COUNTER & WATCHDOG TIMERS

- The mbed timer is based on the 32-bit counters, and can time up to a maximum of $(2^{31}-1)$ µs, i.e., something over 30 min.

- Based on the theory above, one might expect the timer to count up to $(2^{32}-1)$. However, 1 bit is reserved in the API object as a sign bit, so only 31 bits are available for counting.

API Required – TIMER

Function	Usage
`start`	Start the timer
`stop`	Stop the timer
`reset`	Reset the timer to 0
`read`	Get the time passed in seconds
`read_ms`	Get the time passed in milliseconds
`read_us`	Get the time passed in microseconds

Example-1 : Calculate time taken to print message

Measures the time taken to write a message to the screen, and displays that message. Compile and run the program, with Tera Term.

```
#include "mbed.h"
Timer t;                                    // define Timer with name “t”
Serial pc(USBTX, USBRX);

int main() {
  t.start();                                //start the timer
  pc.printf("Hello World!\n");
  t.stop();                                 //stop the timer
  pc.printf("The time taken was %f seconds\n", t.read());  //print to pc
}
```

Example-2 : Calculate time taken to print message

The program creates two timers, named timer_fast and timer_slow. The main program starts these running, and tests when each exceeds a certain number. When the time value is exceeded, a function is called, which flips the associated led.

```
#include "mbed.h"
Timer timer_fast;              // define Timer with name "timer_fast"
Timer timer_slow;              // define Timer with name "timer_slow"
DigitalOut ledA (PC_3);
DigitalOut ledB (PC_4);

void task_fast(void);          //function prototypes
void task_slow(void);
```

TIMER/COUNTER & WATCHDOG TIMERS

```
int main() {
  timer_fast.start();    //start the Timers
  timer_slow.start();
  while (1){
    if (timer_fast.read()>0.2){ //test Timer value
     task_fast();               //call the task if trigger time is reached
     timer_fast.reset();            //and reset the Timer
    }
    if (timer_slow.read()>1){     //test Timer value
     task_slow();
     timer_slow.reset();
    }
  }
}
        void task_fast(void){          //"Fast" Task
          ledA = !ledA;
        }

        void task_slow(void){          //"Slow" Task
          ledB = !ledB;
        }
```

TIMER/COUNTER & WATCHDOG TIMERS

Introduction to ultrasonic sensor

- An ultrasonic sensor is an electronic device that measures the distance of a target object by emitting ultrasonic sound waves, and converts the reflected sound into an electrical signal.

- Ultrasonic sensors have two main components:
  - Transmitter (which emits the sound using piezoelectric crystals)
  - Receiver (which encounters the sound after it has travelled to and from the target).

- In order to calculate the distance between the sensor and the object, the sensor measures the time it takes between the emission of the sound by the transmitter to its contact with the receiver.

- The formula for this calculation is $D = \frac{1}{2} T \times C$ (where D is the distance, T is the time, and C is the speed of sound ~ 343 meters/second).

TIMER/COUNTER & WATCHDOG TIMERS

Introduction to ultrasonic sensor

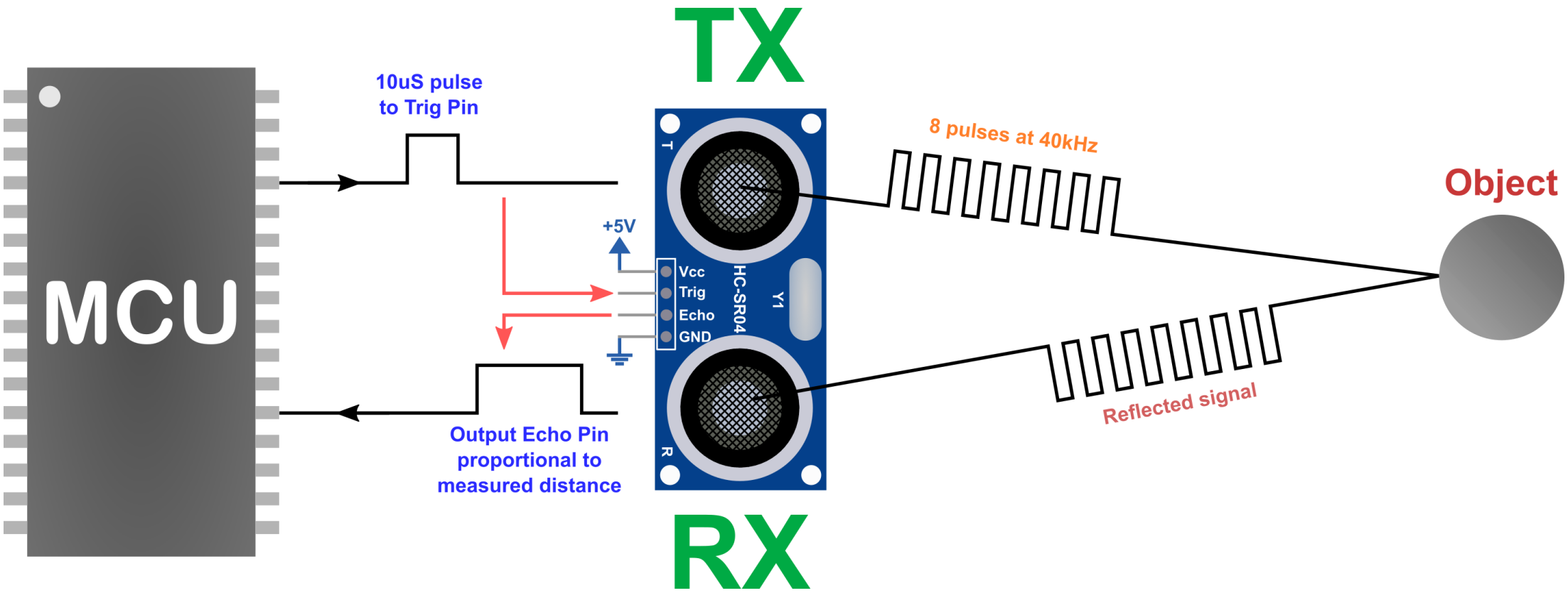
- Ultrasonic sensors are used primarily as proximity sensors.
- They can be found in automobile self-parking technology, anti-collision safety systems, robotic obstacle detection systems also used as level sensors to detect, monitor, and regulate liquid levels in closed containers.
- The HC-SR04 ultrasonic sensor offers excellent non-contact range detection with high accuracy and stable readings from 2 cm to 400 cm.
- It comes complete with ultrasonic transmitter and receiver module.
- The HC-SR04 Ultrasonic Module has 4 pins, Ground (Gnd), VCC (5V) , Trig and Echo. The trig and echo pins to any Digital I/O pin on the Nucleo Board.

TIMER/COUNTER & WATCHDOG TIMERS

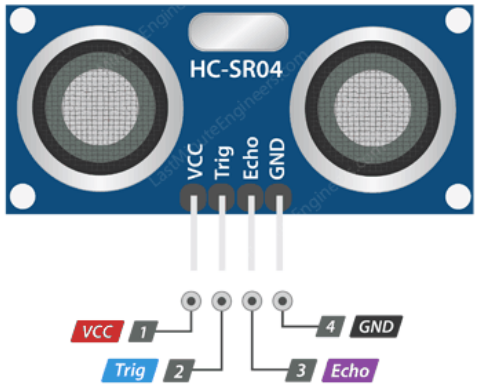
Introduction to ultrasonic sensor

- Steps involved in measuring distance using HC-SR04 Ultrasonic module are,
  - Transmit trigger pulse of at least 10 us to the HC-SR04 Trig Pin.
  - Then the HC-SR04 automatically sends Eight 40 kHz sound wave and generate the rising edge at Echo pin.
  - When the rising edge capture occurs at Echo pin, start the Timer and wait for falling edge on Echo pin.
  - As soon as the falling edge is captured at the Echo pin, read the count of the Timer. This time count is the time required by the sensor to detect an object and return back from an object (to and from).

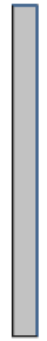
Introduction to ultrasonic sensor



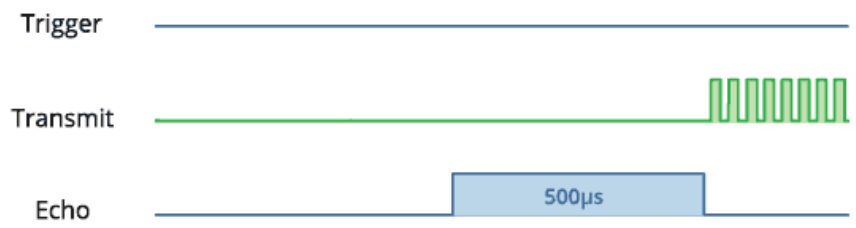
TIMER/COUNTER & WATCHDOG TIMERS



Distance = (Speed x Time)/ 2
Distance = (0.034 cm/µs x 500 µs) / 2
Distance = 8.5 cm



When no obstacles then timeout after 38ms



When obstacle detected pulse width of 50 µS to 25 mS received

TIMER/COUNTER & WATCHDOG TIMERS

Example-3 - Read the distance value from HC-SR04 and print it on serial monitor

Write a program to read distance value from HC-SR04 ultrasonic sensor module in cm and print it on the serial monitor. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

Program

```
#include "mbed.h"
Serial PC(USBTX,USBRX);
DigitalOut trigger(PC_8);
DigitalIn  echo(PC_6);
int distance = 0;
Timer sonar;

int main()
{
  //Loop to read Sonar distance values, scale, and print
    while(1) {
// trigger sonar to send a ping
        trigger = 1;
        sonar.reset();
        wait_us(10.0);
        trigger = 0;
//wait for echo high
        while (echo==0);
//echo high, so start timer
        sonar.start();
//wait for echo low
        while (echo==1);
//stop timer and read value
        sonar.stop();
        distance = (sonar.read_us())/58.0;
        PC.printf(" Distance is %d cm \n\r",distance);
//wait so that any echo(s) return before sending another ping
        wait(0.2);
    }
}
```

Example-4 - Reverse parking sensor module

Write a program to design a reverse parking sensor module. This module consist of HC-SR04 ultrasonic sensor, LCD and buzzer interfaced with Nucleo. The ultrasonic sensor continuously measure the distance (in cm) between the car and obstacle, then display it on the first row of the LCD. Whenever the measured distance is lesser than 30cm generate warning signal to driver using buzzer also display a message "Obstacle !!!" on the second row of the LCD display. Implement and verify this logic on the STM32 Nucleo-64 board using Keil Studio Cloud IDE.

TIMER/COUNTER & WATCHDOG TIMERS

Program

```
#include "mbed.h"
#include "TextLCD.h"
TextLCD lcd(PC_0,PC_1,PB_0,PA_4,PA_1,PA_0); // rs,e,d4-d7
DigitalOut trigger(PC_8);
DigitalIn  echo(PC_6);
DigitalOut buzzer(PC_10);
int distance = 0;
Timer sonar;

int main()
{
  //Loop to read Sonar distance values, scale, and print
  lcd.cls();
    while(1) {
// trigger sonar to send a ping
        trigger = 1;
        sonar.reset();
        wait_us(10.0);
        trigger = 0;
        while (echo==0);
//echo high, so start timer
        sonar.start();
//wait for echo low
        while (echo==1);
//stop timer and read value
        sonar.stop();
        distance = (sonar.read_us())/58.0;
        lcd.locate(5,0);
        lcd.printf("Distance is %d cm \n\r",distance);
//wait so that any echo(s) return before sending another ping
        wait(0.2);
        if (distance>15)
        {
            buzzer=1;
            lcd.locate(2,1);
            lcd.printf("Obstacle !!!",distance);
        }
        else
        {buzzer=0;
        }
    }
}
```

TIMER/COUNTER & WATCHDOG TIMERS

Exercise-1 - Smart Parking system

Write a program to design smart parking system using HC-SR04 ultrasonic sensor, servo motor, buzzer, LCD and STM32 Nucleo-64 board.

- The ultrasonic sensor module place near the gate entrance continuously check for the incoming vehicles. The LCD display "Smart Parking" on the first row and "Avail. slot: XY" in second row of the display.
- When a vehicle comes closer to the ultrasonic sensor detection area and parking slot is available then the system open a gate barrier to 90° (close after 10 seconds) to allow the vehicle enter into the parking slot and decrement parking slot by 1.
- If no parking slot available then display a message "No Parking slot" on LCD (2nd line) and switch on the buzzer (for 5 Seconds). Have a similar system on the exit and increment the free slot by 1 for every vehicle leaves the parking slot.

Simulate and verify this logic on Nucleo using Tinkercad circuits simulator.

Note: XY is number of available slot and initially assume total available parking slot is 5.

THANK YOU