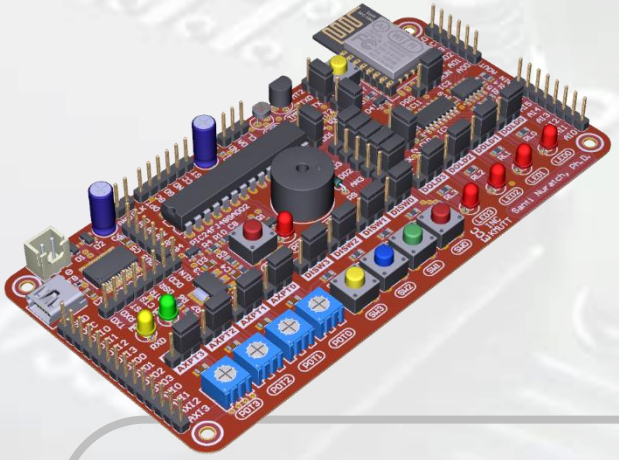
Day #1, Section #2
Embedded C Programming
System Setup
&
RTOS+APIs Programming



Santi Nuratch., Ph.D.

Embedded Computing and Control Lab. @ INC-KMUTT

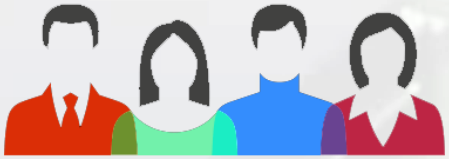
santi.inc.kmutt@gmail.com, santi.nur@kmutt.ac.th

Department of Control System and Instrumentation Engineering,
King Mongkut's University of Technology Thonburi, **KMUTT**

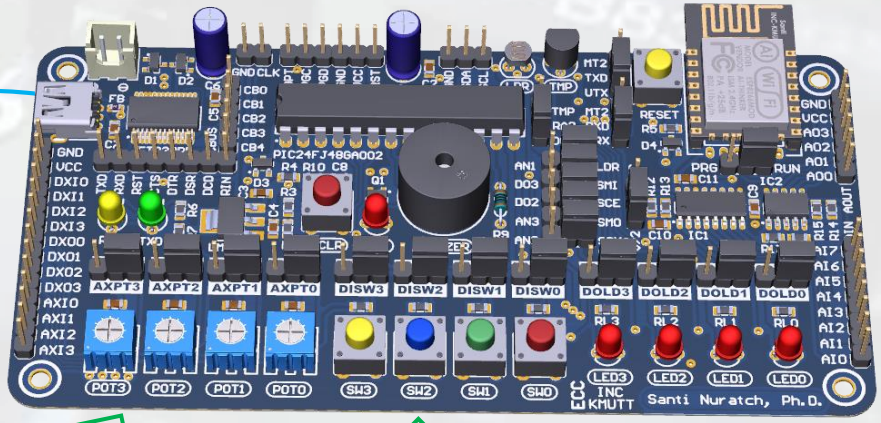
Embedded System and IoT

Internet
Network

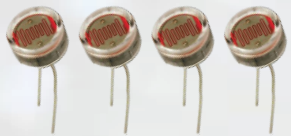
Router



Clients (Users)



Development Computer
(Programmer)



Analog Inputs

Digital Inputs

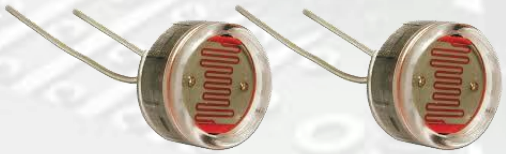
Digital Outputs

What is Embedded Systems

Embedded System

UART
Serial Port

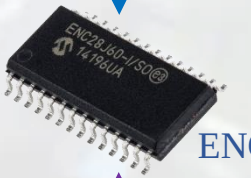
LDR



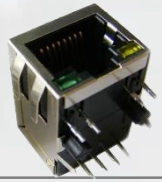
Push Button Switch

Limit Switch

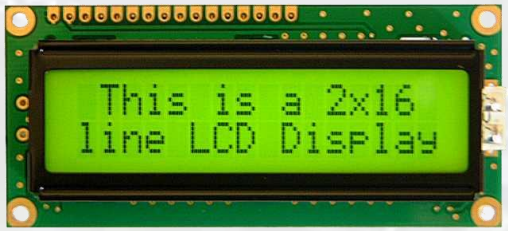
SPI



Ethernet

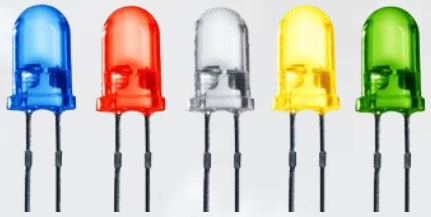


LCD



Servo Motor

LED



Sensors

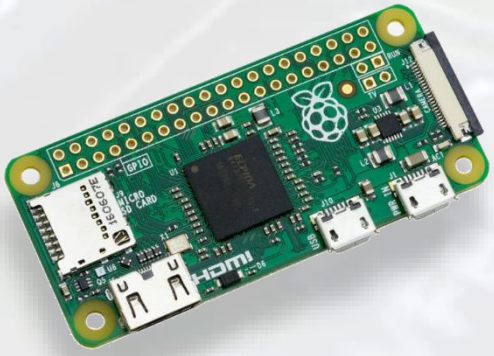
Actuators

Smart Devices vs. Smart Systems

Single-Board Computer and Microcontroller for IoT

Raspberry Pi

ESP32

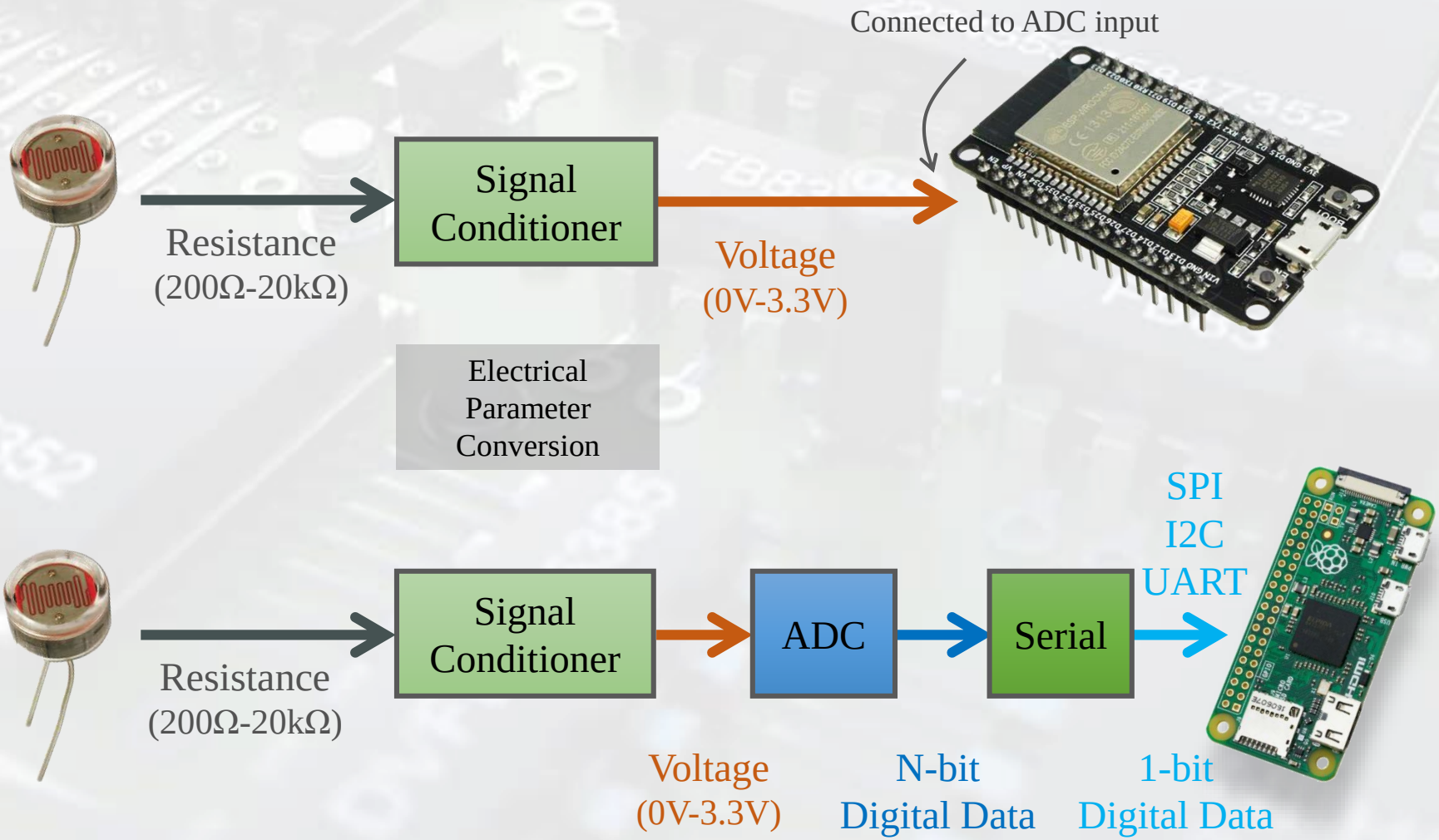


Raspberry Pi-Zero

ESP8266

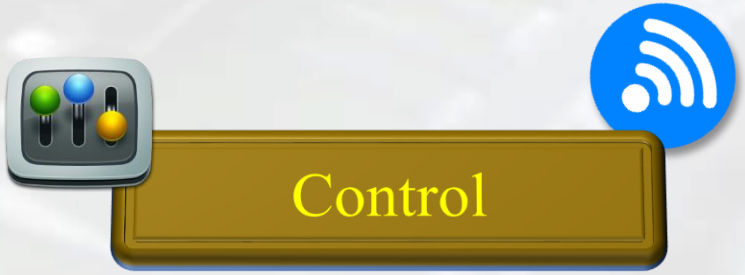
Smart Devices vs. Smart Systems

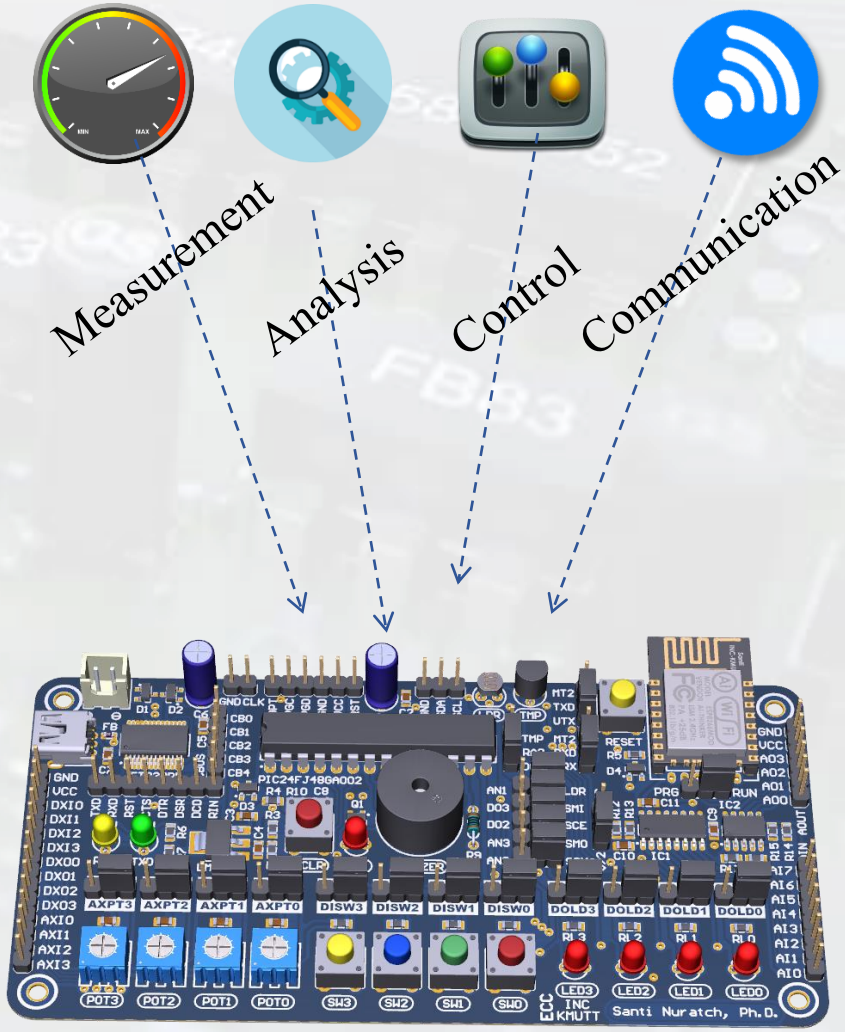
Sensor and Signal Conditioner/Converter



IoT Devices Key Features

Smart Embedded Devices





Smart Embedded Device

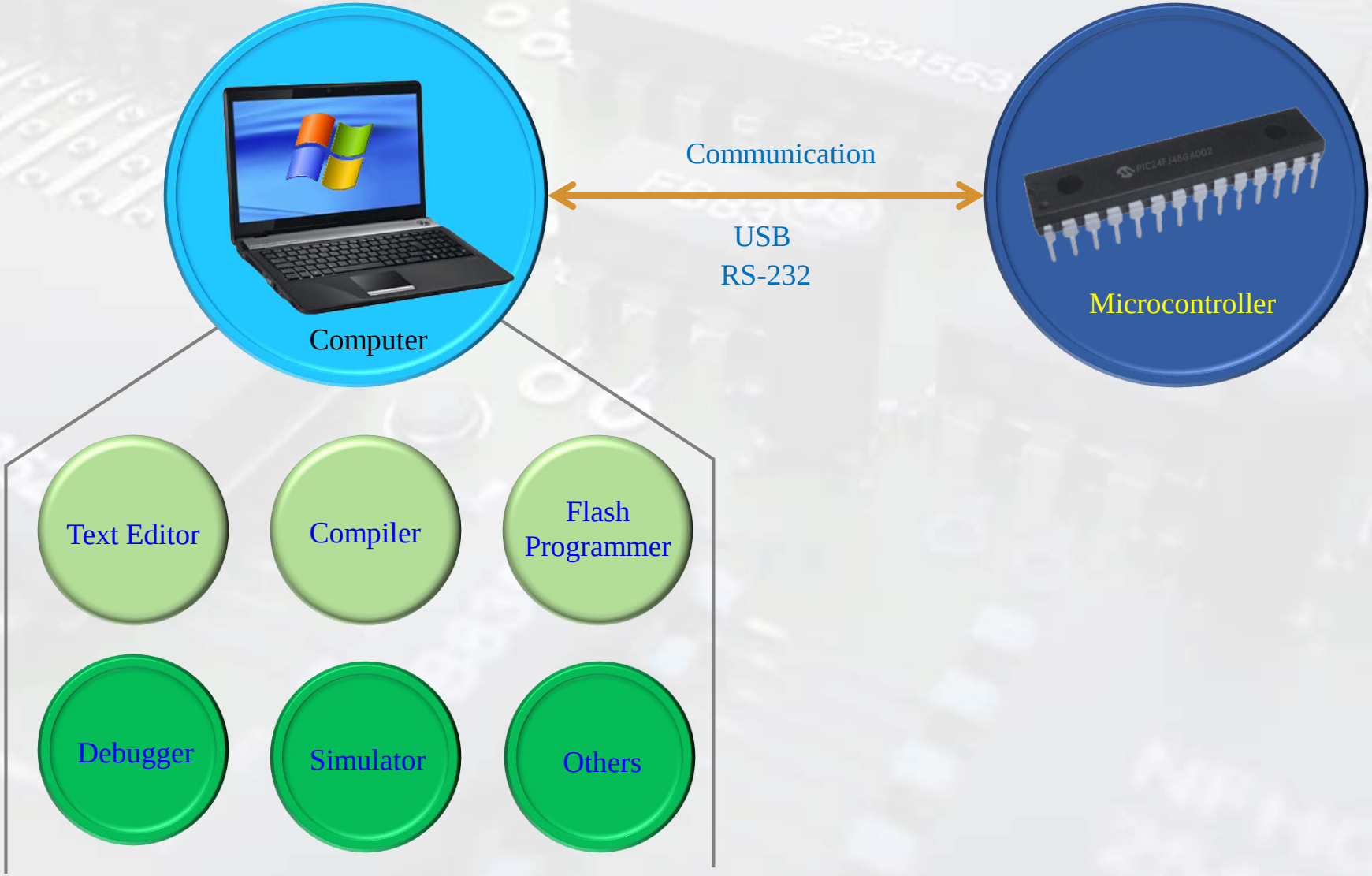
Smart Devices vs. Smart Systems

Smart Device (Electronics, Embedded Systems and Computer Technologies)

Smart System is composed of many devices

Software Development

Software Development



Required Software Tools

https://www.google.com/chrome/

https://code.visualstudio.com/

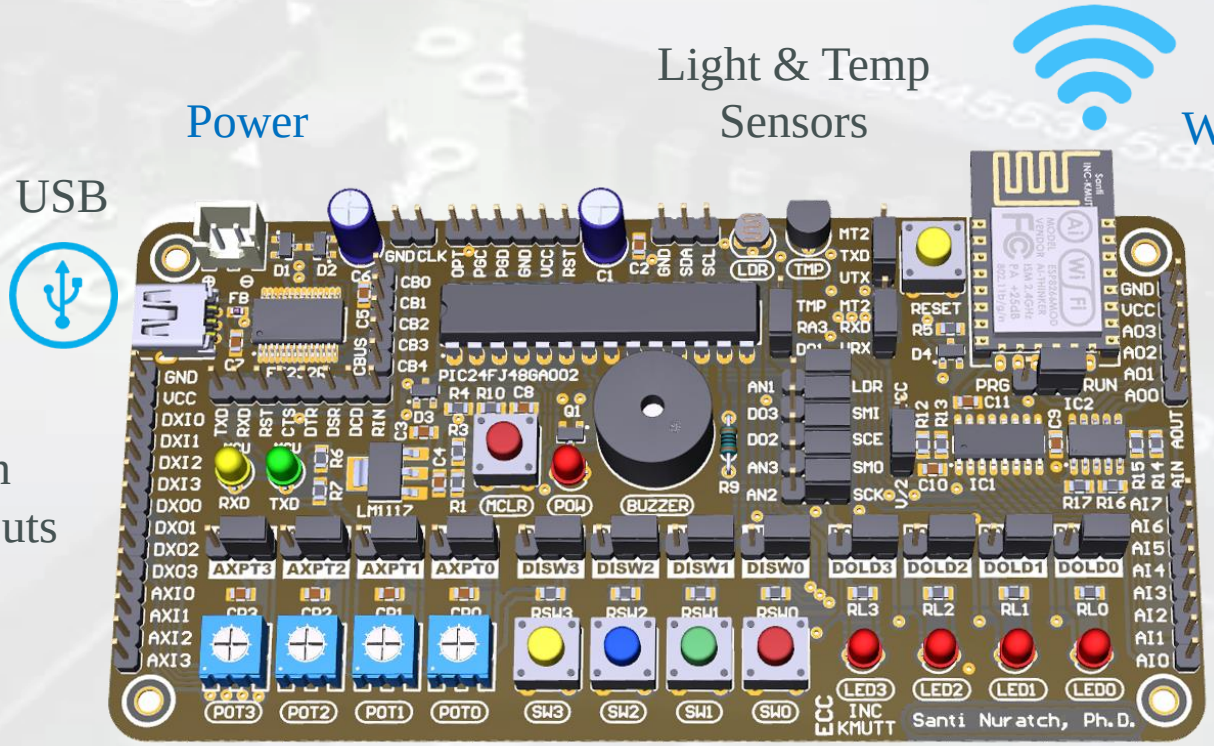
http://www.microchip.com/mplab/compilers

ecc-pic24-cli (given in class)

https://nodejs.org/en/download/

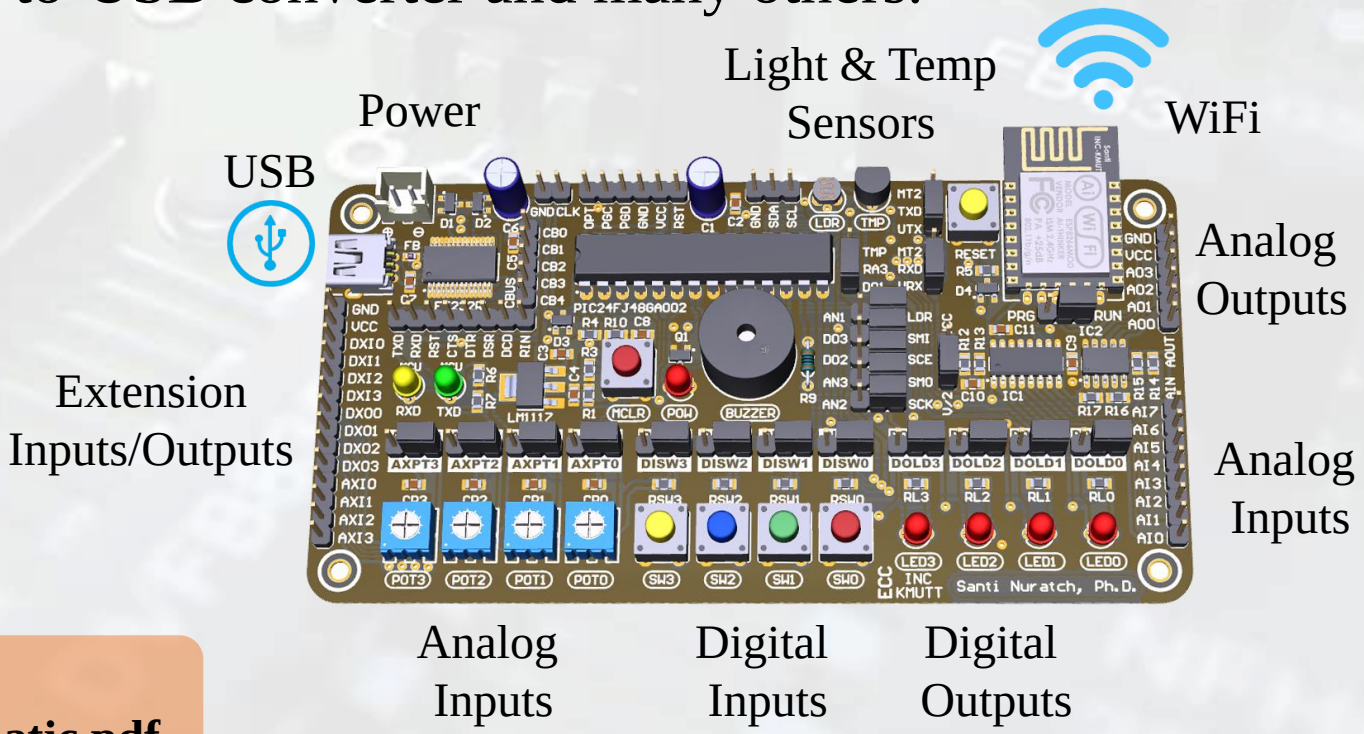
https://www.mongodb.com/download-center

Required Experimental Board



The ECC-PIC24 MCU Board

The ECC-PIC24 experimental board is design for embedded learners. The board contains many onboard input/output devices/modules, e.g.; light-emitting diodes (LEDs), push button switches (PBS), light-dependent resistor (LDR), Buzzer, UART-to-USB converter and many others.



More details:
EccPic24_Schematic.pdf

MCU + RTOS + Drivers

ECC-OS

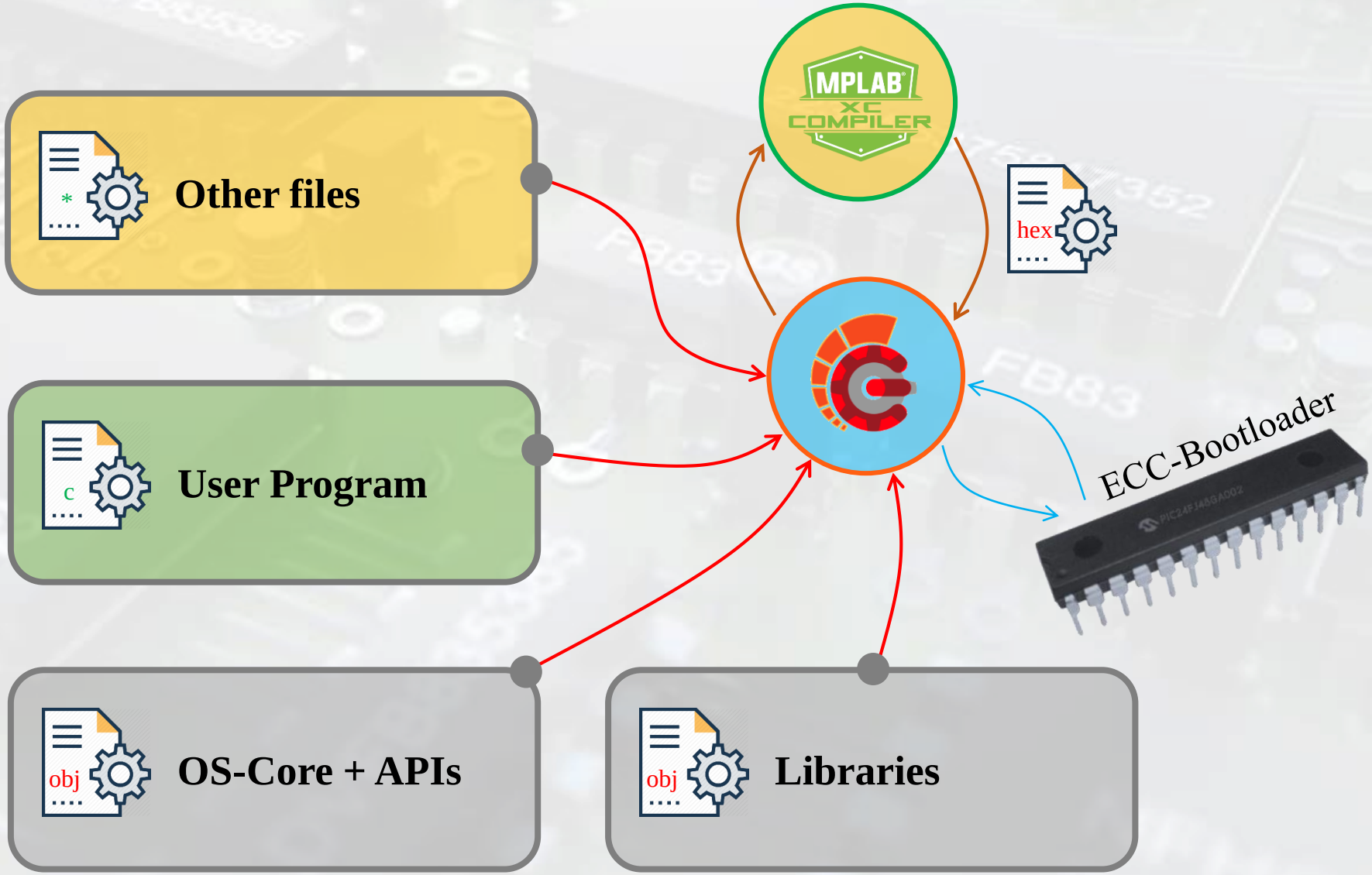
Event-Driven Real-time Multitasking Operation System (OS-Core)

Optimized Drivers and Middleware with (APIs)

ECC-Bootloader

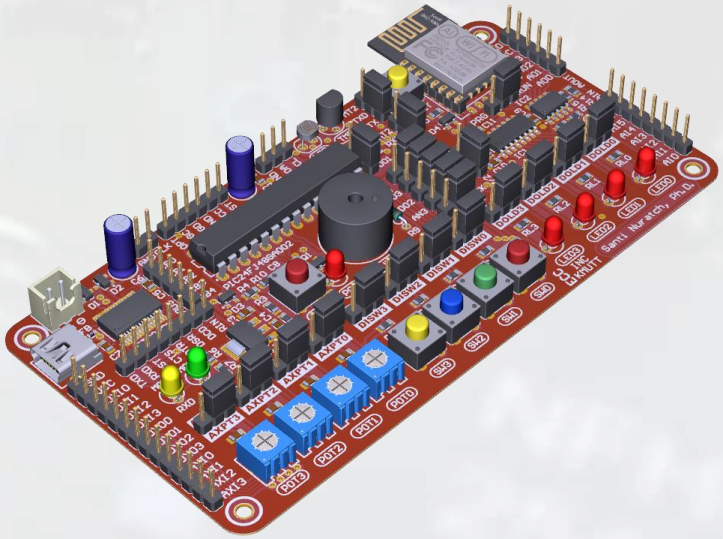
PIC24FJ48GA002

MCU + RTOS + Drivers



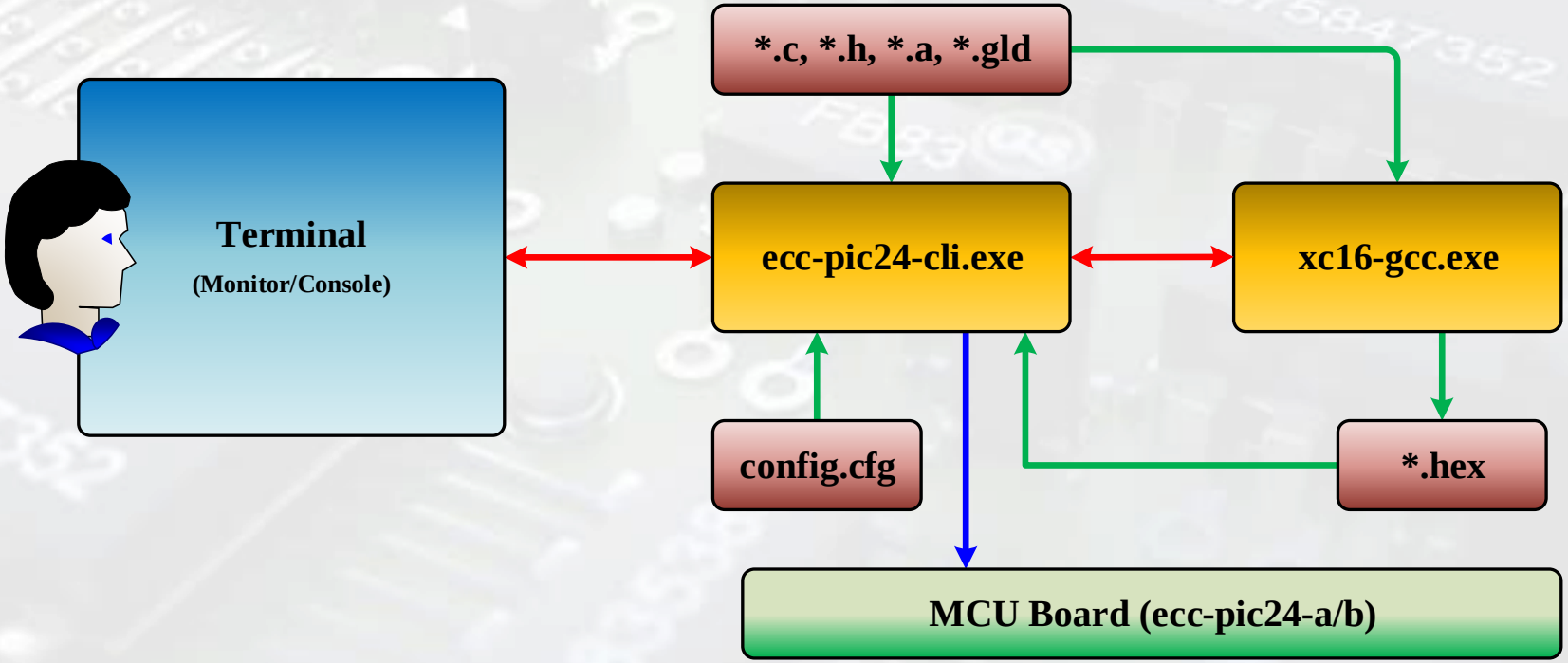
The ECC-OS (The Real-time Multitasking Operating System)

The ECC-OS is a small footprint operating system for small microcontrollers. It is deigned and developed using event-driven and callback-function techniques. It fully supports many real-time multitasking applications. Internally, the ECC-OS provides many utility functions, e.g.; serial communication, basic function of input/output ports (Digital and Analog) and time management. Also, it has built-in functions to working with the ECC-PIC24 experiment board



What is the ecc-pic24-cli?

The **ecc-pic24-cli** is a command-line interface application used for linking to **xc16-gcc** compiler and microcontroller board



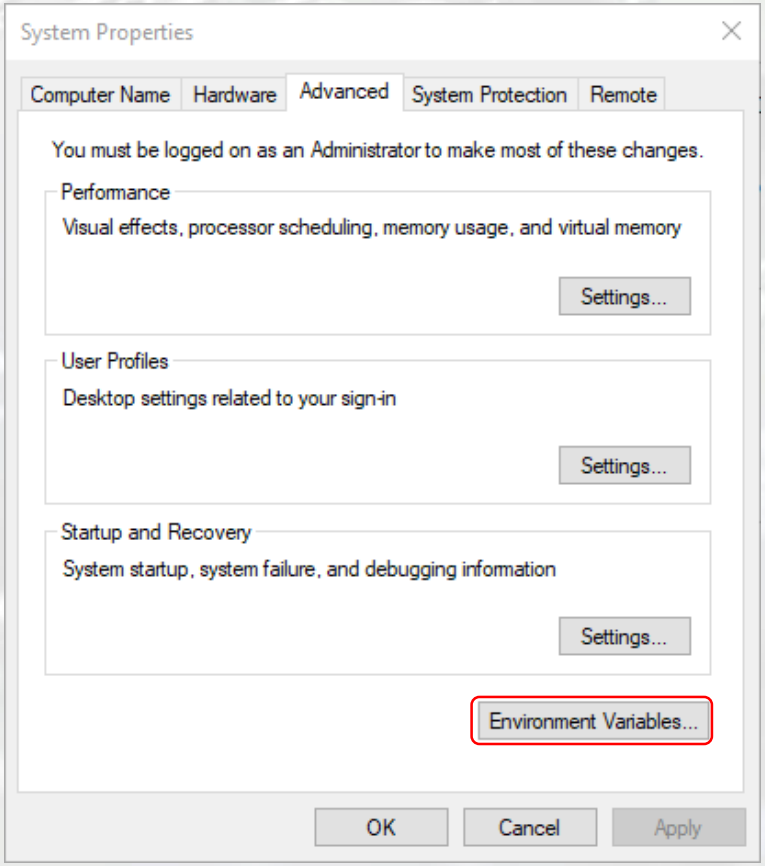
Mainly, the ecc-pic24-cli receives user's command(s), reads a configuration file (config.cfg) and prepares special commands for the xc16-gcc (compiler). It also reads and processes HEX file (*.hex) before sending to flash memory of target microcontroller.

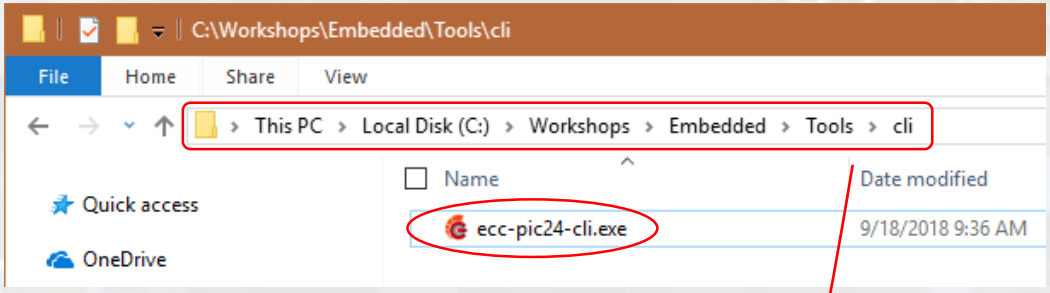
ecc-pic24-cli
Setting up

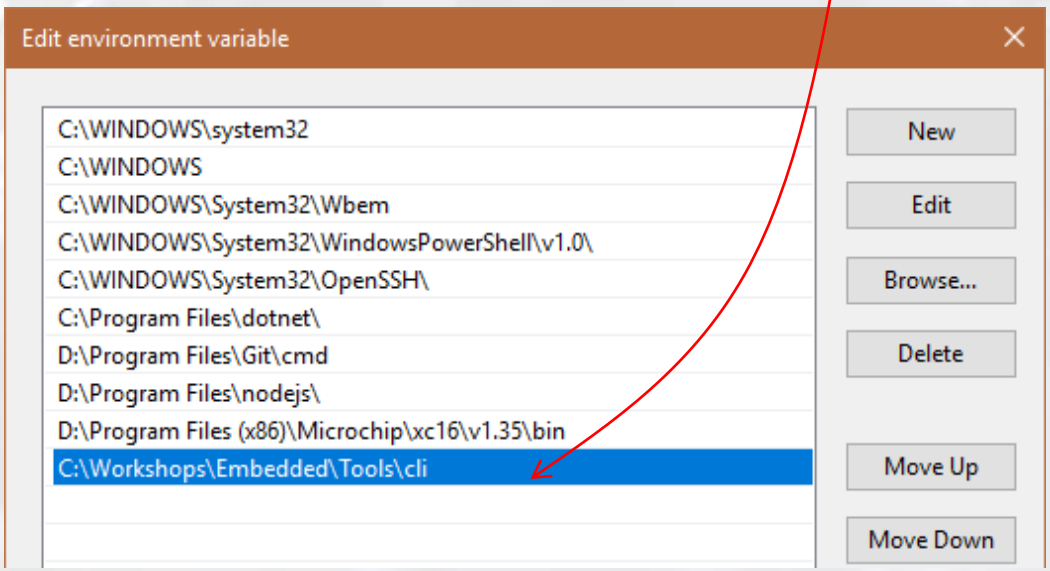
ccc-pic24-cli Setup

01 : Copy the **Workshops** folder and paste in **C:/** (or another e.g. D:/)

02 : Add the **C:\Workshops\Embedded\Tools\cli** into System **Environment Variables**

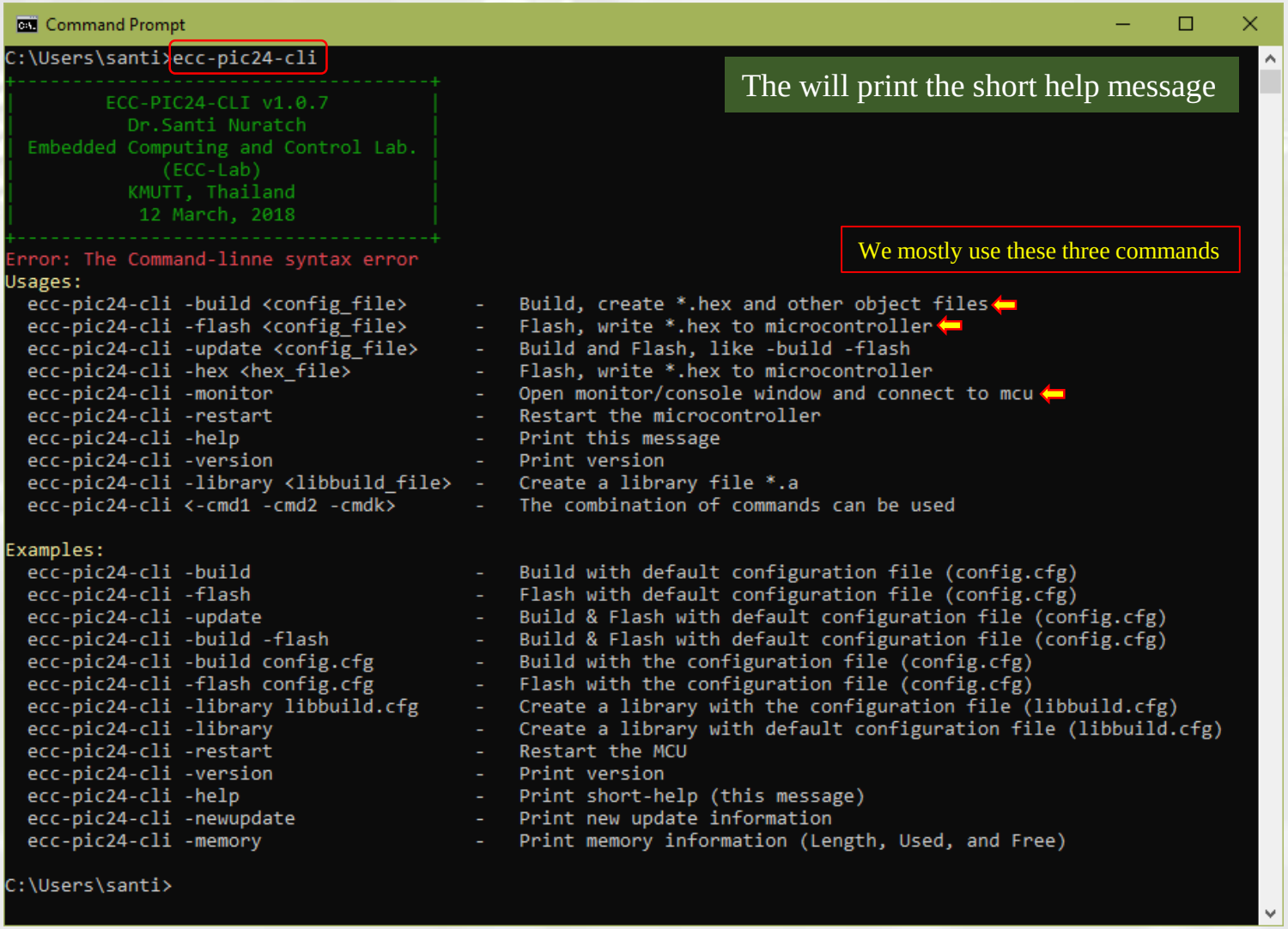






ccc-pic24-cli Setup

03 : Run a Terminal/Console and give it the command ecc-pic24-cli



ecc-pic24-cli commands

To work with the **ecc-pic24-cli**, or others command-line interface applications, a terminal/console is required, e.g.; the Command Prompt (CMD) or other console emulators for Windows. The **cmder** is recommended. Let's try the first command, **ecc-pic24-cli -help**

Cmder

```
D:\MCU_Projects\EccOS\edf\usr.X
λ ecc-pic24-cli -h
+-------------------------------------+
|        ECC-PIC24-CLI v1.0.2         |
|          Dr.Santi Nuratch           |
| Embedded Computing and Control Lab. |
|             (ECC Lab)               |
|          KMUTT, Thailand            |
+-------------------------------------+
Error: The Command-linne syntax error
Usages:
  ecc-pic24-cli -build <config_file>      -   Build, create *.hex and other object files
  ecc-pic24-cli -flash <config_file>      -   Flash, write *.hex to microcontroller
  ecc-pic24-cli -update <config_file>     -   Build and Flash, like -build -flash
  ecc-pic24-cli -hex <hex_file>           -   Flash, write *.hex to microcontroller
  ecc-pic24-cli -monitor                  -   Open monitor/console window and connect to mcu
  ecc-pic24-cli -restart                  -   Restart the microcontroller
  ecc-pic24-cli -help                     -   Print this message
  ecc-pic24-cli -version                  -   Print version
  ecc-pic24-cli -library <libbuild_file>  -   Create a library file *.a
  ecc-pic24-cli <-cmd1 -cmd2 -cmdk>       -   The combination of commands can be used

Examples:
  ecc-pic24-cli -build                    -   Build with default configuration file (config.cfg)
  ecc-pic24-cli -flash                    -   Flash with default configuration file (config.cfg)
  ecc-pic24-cli -update                   -   Build & Flash with default configuration file (config.cfg)
```

Syntax:
ecc-pic24-cli -COMMAND PARAMETER

Nothing to remember, but more practices are required (not only this, but also all in engineering fields)

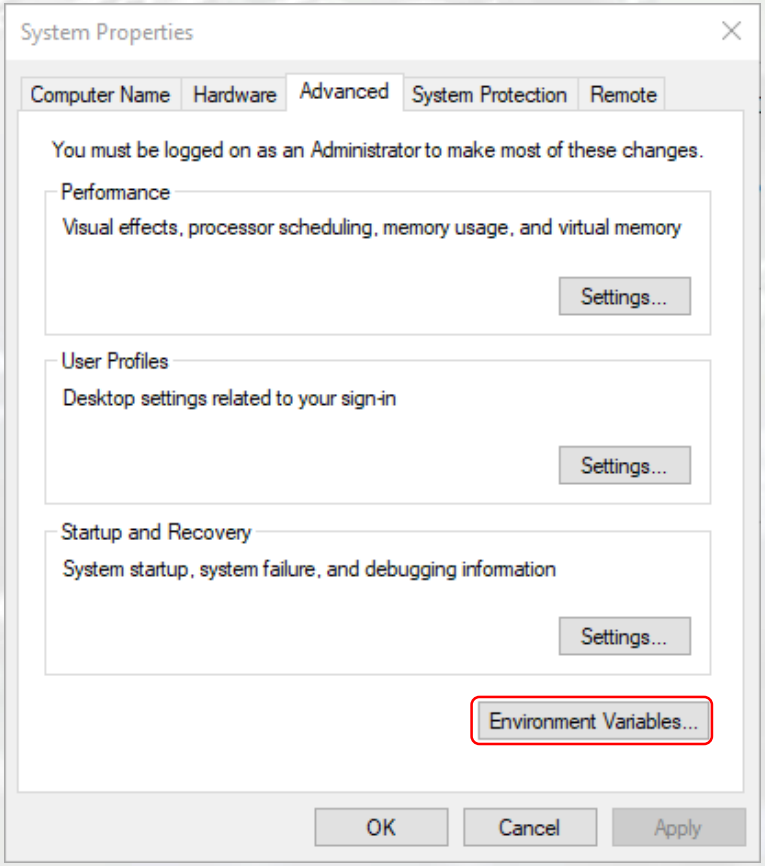
Setup the xc16-gcc

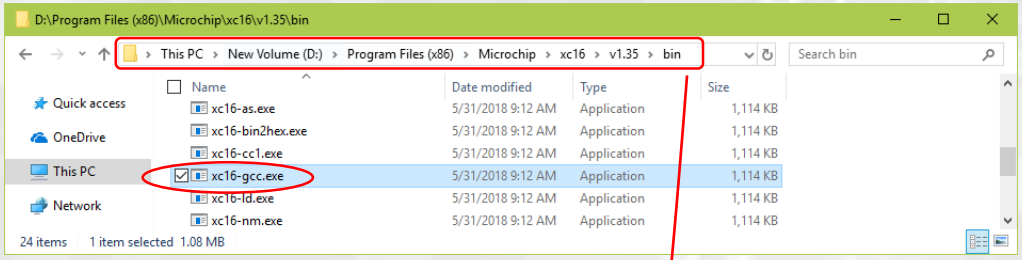
xc16-gcc
Setting up

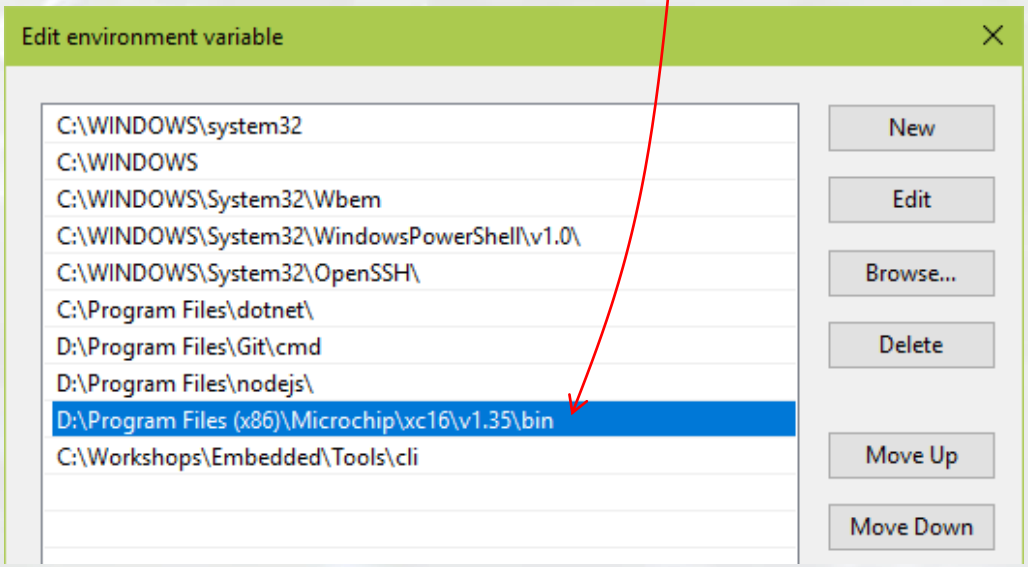
xc16-gcc Setup

01 : Install the MPLAB XC16 C Compiler v1.35 (http://www.microchip.com/mplab/compilers)

02 : Add the D:\Program Files (x86)\Microchip\xc16\v1.35\bin into System Environment Variables

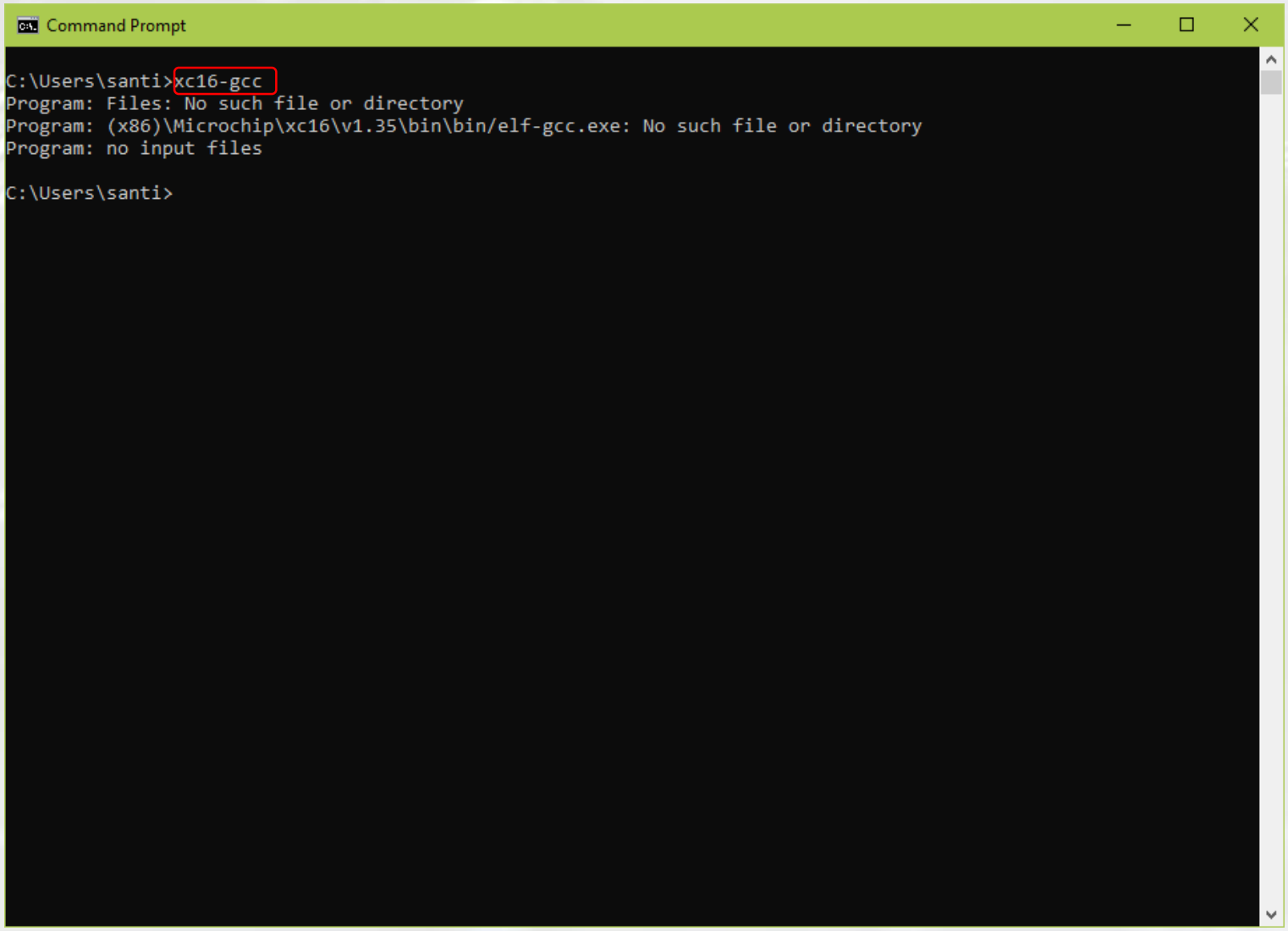






xc16-gcc Setup

03 : Run a Terminal/Console and give it the command **xc16-gcc**



Setup the config.cfg

Example of the ecc-pic24-cli configuration file

To make the **ecc-pic24-cli** works correctly, a configuration file (config.cfg) is needed to be written carefully. The config.cfg contains all information of a project, e.g.; source files and directories, communication properties, compiler's path and others.

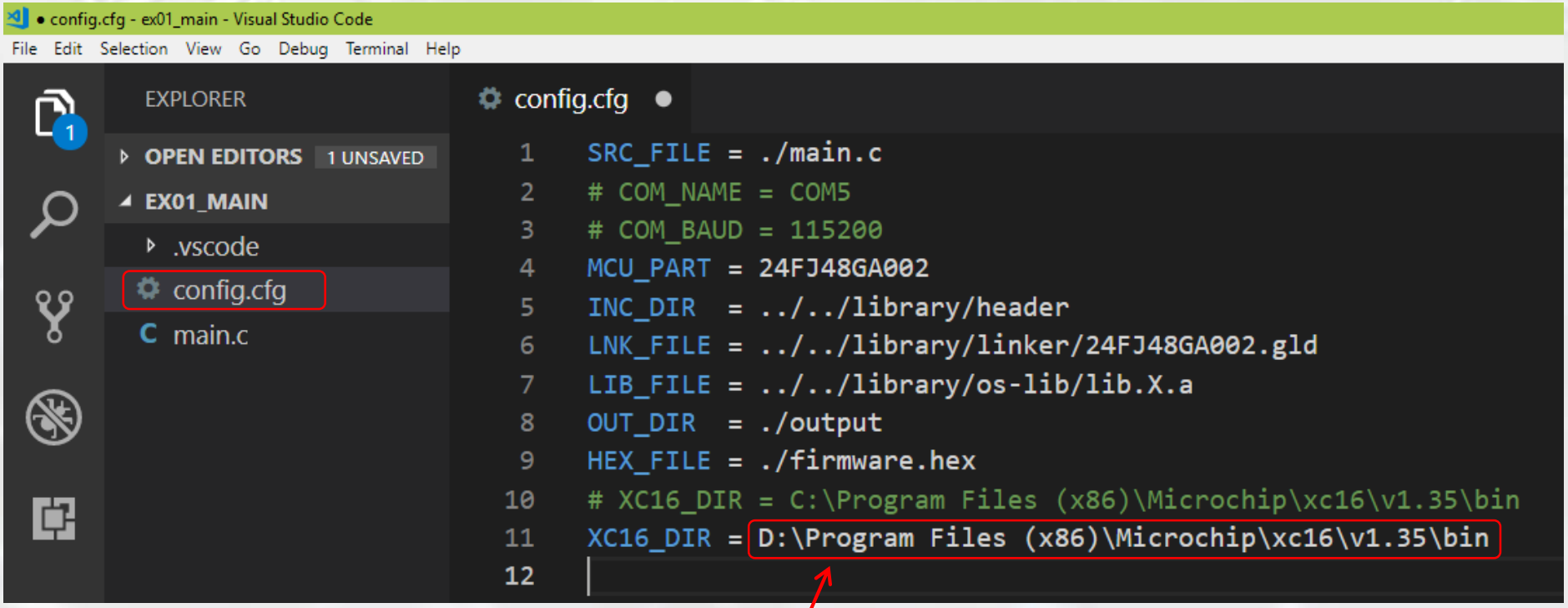
```
SRC_FILE    = ./main.c
# COM_NAME  = COM5
# COM_BAUD  = 115200
MCU_PART    = 24FJ48GA002
INC_DIR     = ../../library/header
LNK_FILE    = ../../library/linker/24FJ48GA002.gld
LIB_FILE    = ../../library/os-lib/lib.X.a
OUT_DIR     = ./output
HEX_FILE    = ./firmware.hex
XC16_DIR = C:\Program Files (x86)\Microchip\xc16\v1.35\bin
# XC16_DIR   = D:\Program Files (x86)\Microchip\xc16\v1.35\bin
```

We have to know exactly their meanings

config.cfg Setup

01 : Run Visual Code Studio and Open Folder
C:\Workshops\Embedded\Examples\ex01_main

02 : Double-click the config.cfg to open and check all lines carefully



This line links the ecc-pic24-cli to the xc16-gcc

config.cfg Setup

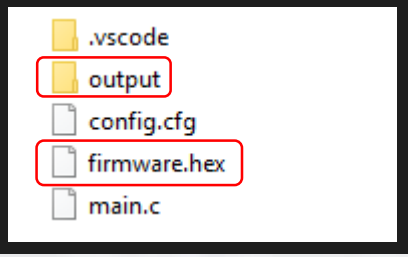
03 : In the Visual Code Studio, click the **Terminal** | **New Terminal** on the menu bar, and check the working directory (the directory can be changed by **cd**

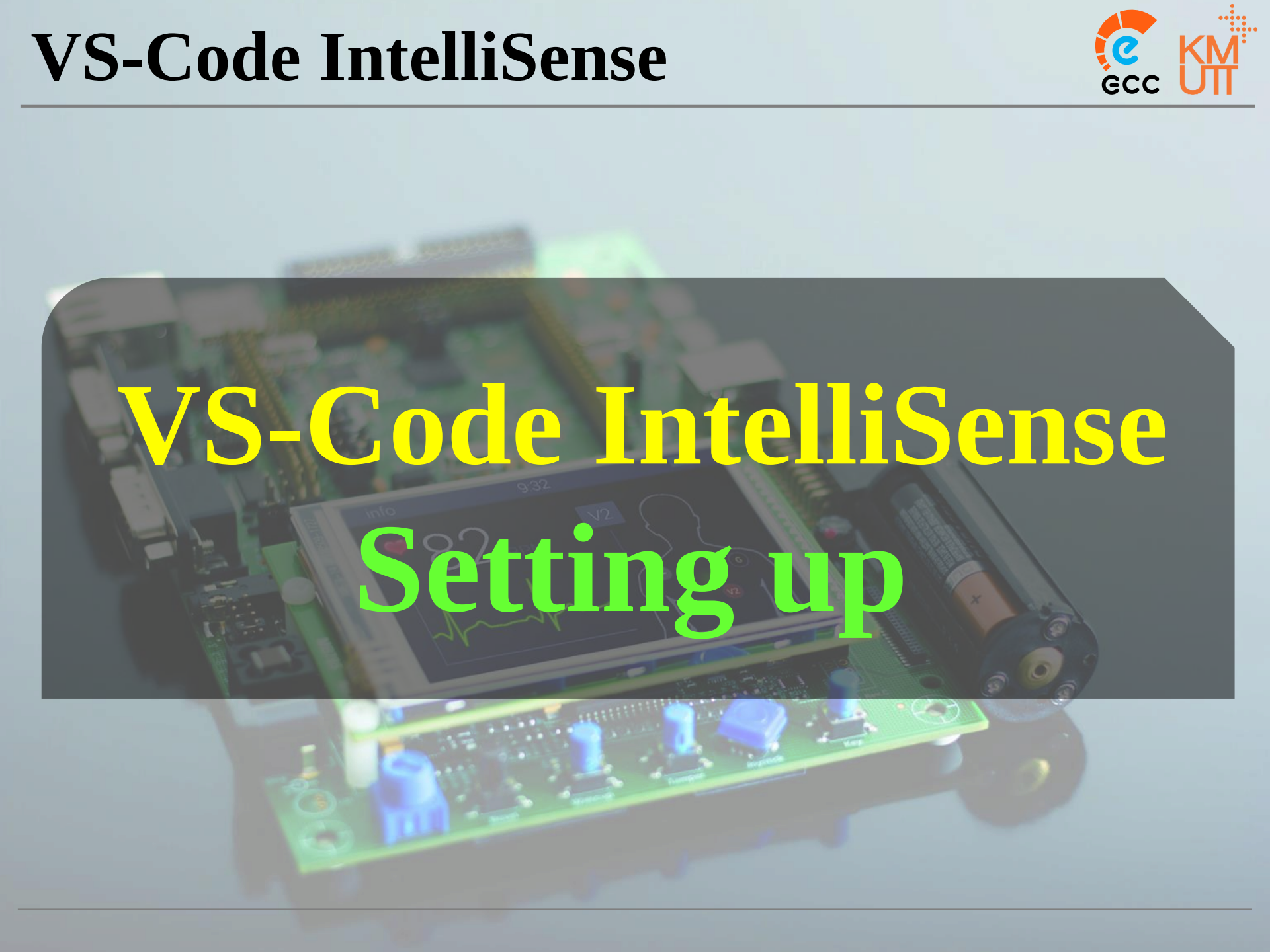
```
PROBLEMS   TERMINAL   ...                                  2: cmd

Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Workshops\Embedded\Examples\ex01_main>
```

04 : Enter a command **ecc-pic24-cli -build**

```
Linking: "24FJ48GA002.gld" --> "elf.elf"
Linking finished [1205 mS]
+--------------------------------------+
|          Creating hex file           |
+--------------------------------------+
Converting: elf.elf --> firmware.hex
The new firmware "firmware.hex" is created [50 mS]
(see in "." directory)

C:\Workshops\Embedded\Examples\ex01_main>
```

ecc-pic24-cli -build compiles and creates firmware.hex and other files, see in the ex01_main





VS-Code IntelliSense

Setting up

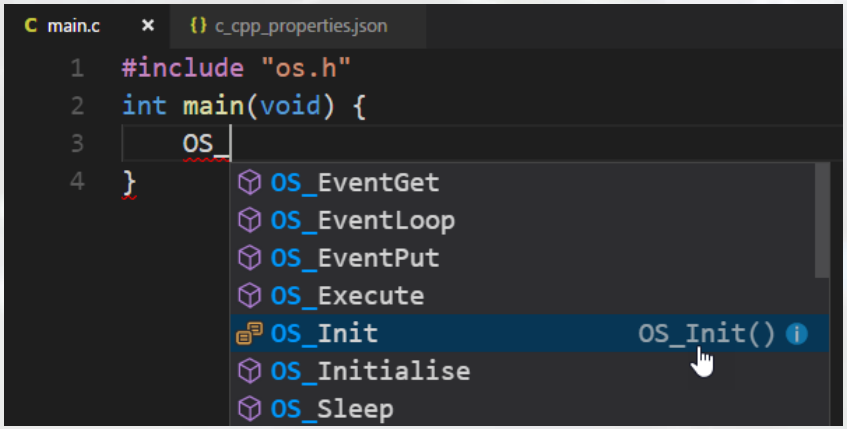
VS-Code Setup

To make the VS-Code shows **intellisense**, the following steps are required:

1) Open the application directory (root or app)

2) Open the **c_cpp_properties.json**, and add the lines shown below into the **includePath** and **intelliSenseMode**

```
"D:/MyDirectory/root/library/header/*",
"C:/Program Files (x86)/Microchip/xc16/v1.35/include",
"C:/Program Files (x86)/Microchip/xc16/v1.35/support/generic/h",
"C:/Program Files (x86)/Microchip/xc16/v1.35/support/PIC24F/h",
```



All are depended on your system.

3) Restart the VS-Code and check the intellisense

The First C-Program (based-on ECC-OS)

Programming with ECC-OS (and others) requires special knowledge of C-Programming, the Embedded C Programming. In this class, we are not going in details on that topic, but you have to do!. The lines of code below are the main function of the C-Program. Inside the main, just two functions of the OS are required. The **`OS_Init()`** must be executed before all functions of the ECC-OS are called. The **`OS_Start()`** must be executed to start the OS.

```
#include “os.h”
int main(void)
{
    // peripherals and variables initialization
    OS_Init();
    // Other initializations
    OS_Start();
}
```

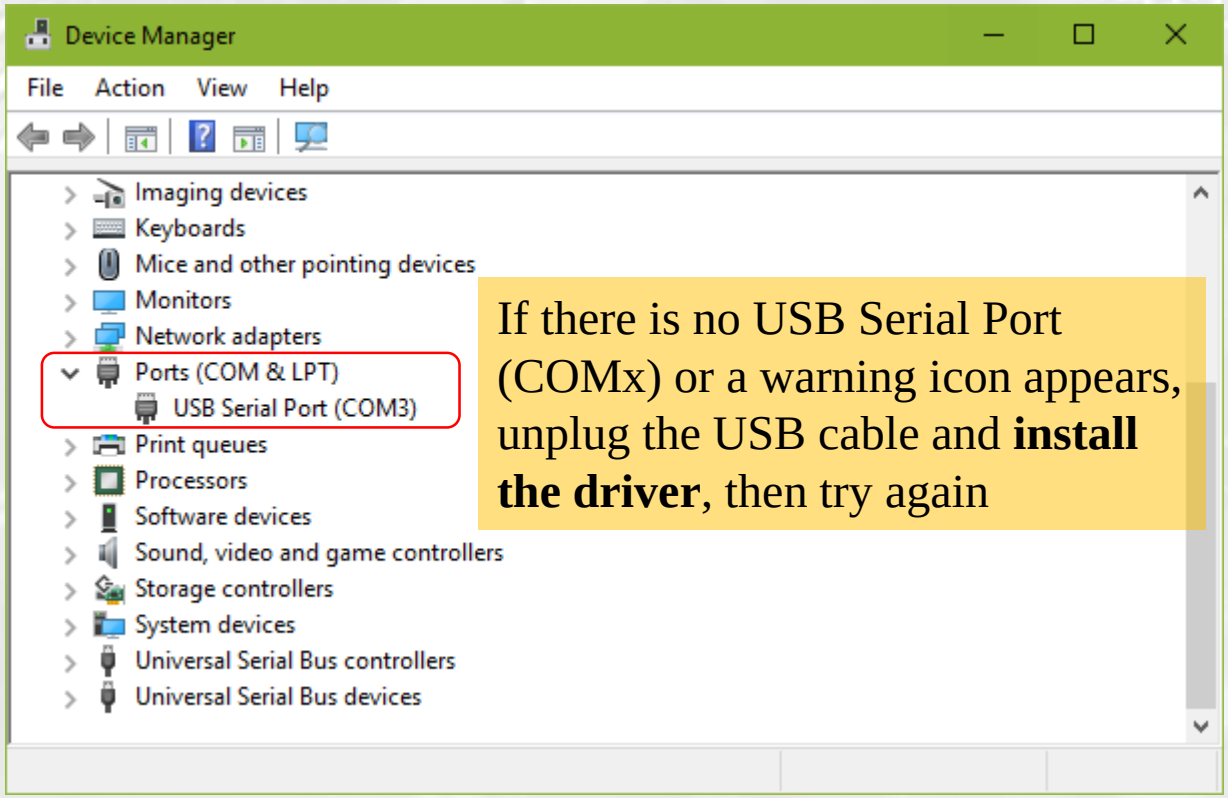
Setup the Microcontroller Board

ECC-PIC24 MCU Board

Setting up

ECC-PIC24 MCU Board Setup

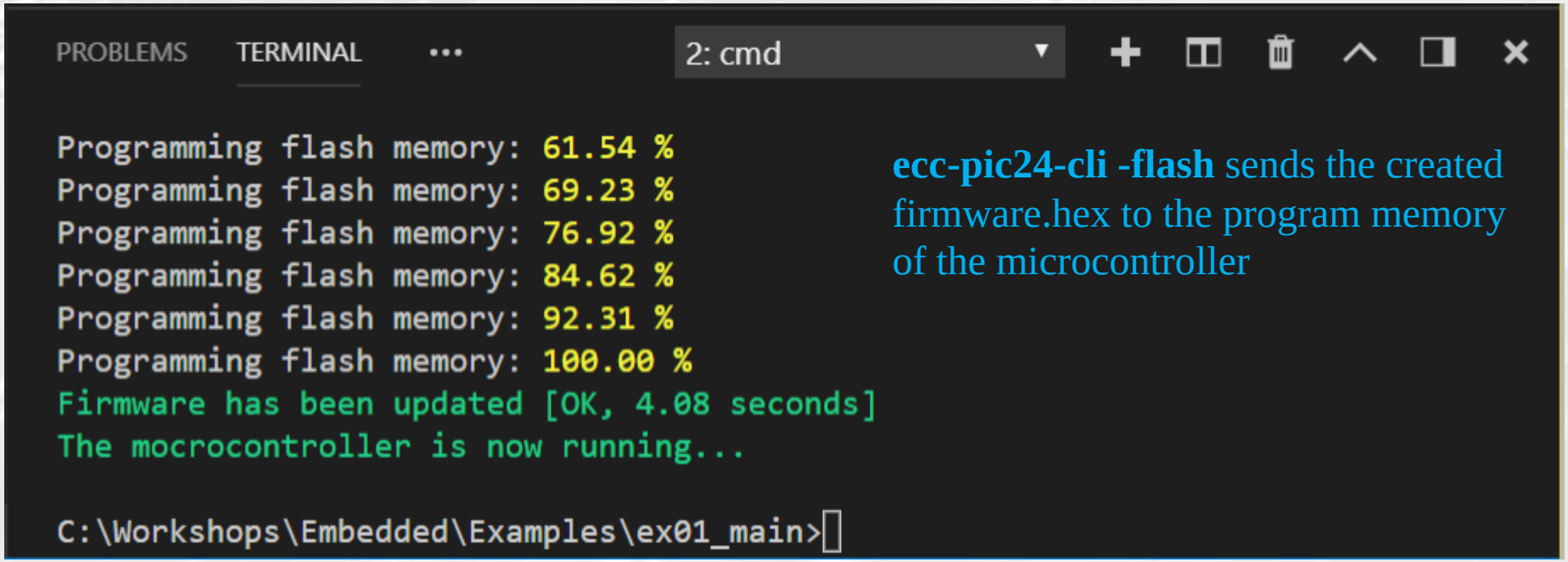
01 : Connect he ECC-PIC24 MCU Board to the computer using the USB cable and check its existing in the Device Manager



The VCP driver (FTDI) can be downloaded from https://www.ftdichip.com/Drivers/VCP.htm

ECC-PIC24 MCU Board Setup

02 : Back to the Terminal of the Visual Code Studio again and give it a command **ecc-pic24-cli -flash**

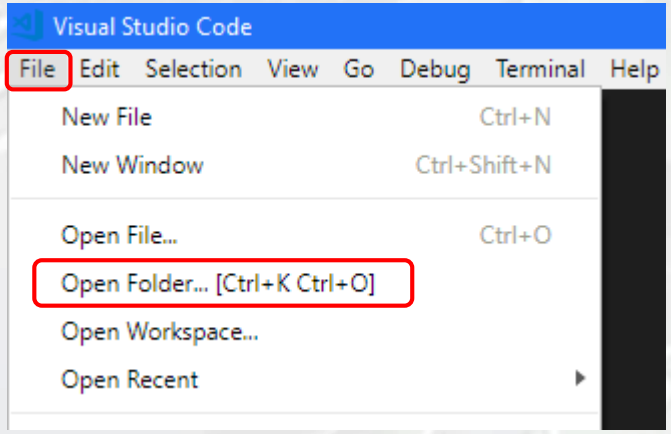


ecc-pic24-cli -flash sends the created firmware.hex to the program memory of the microcontroller

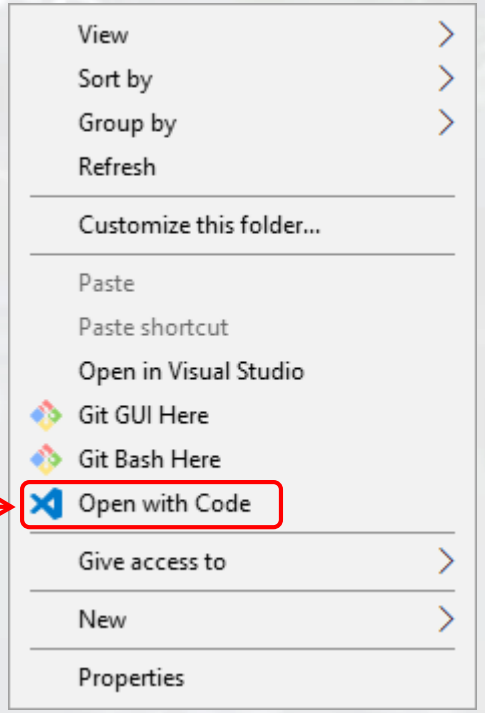

Embedded C-Programming
with
ECC-OS + APIs

Getting Started with Embedded C-Programming

01 : Run the VS-Code and open the first example, C:\Workshops\Embedded\Examples\ex01_main, File | Open Folder…

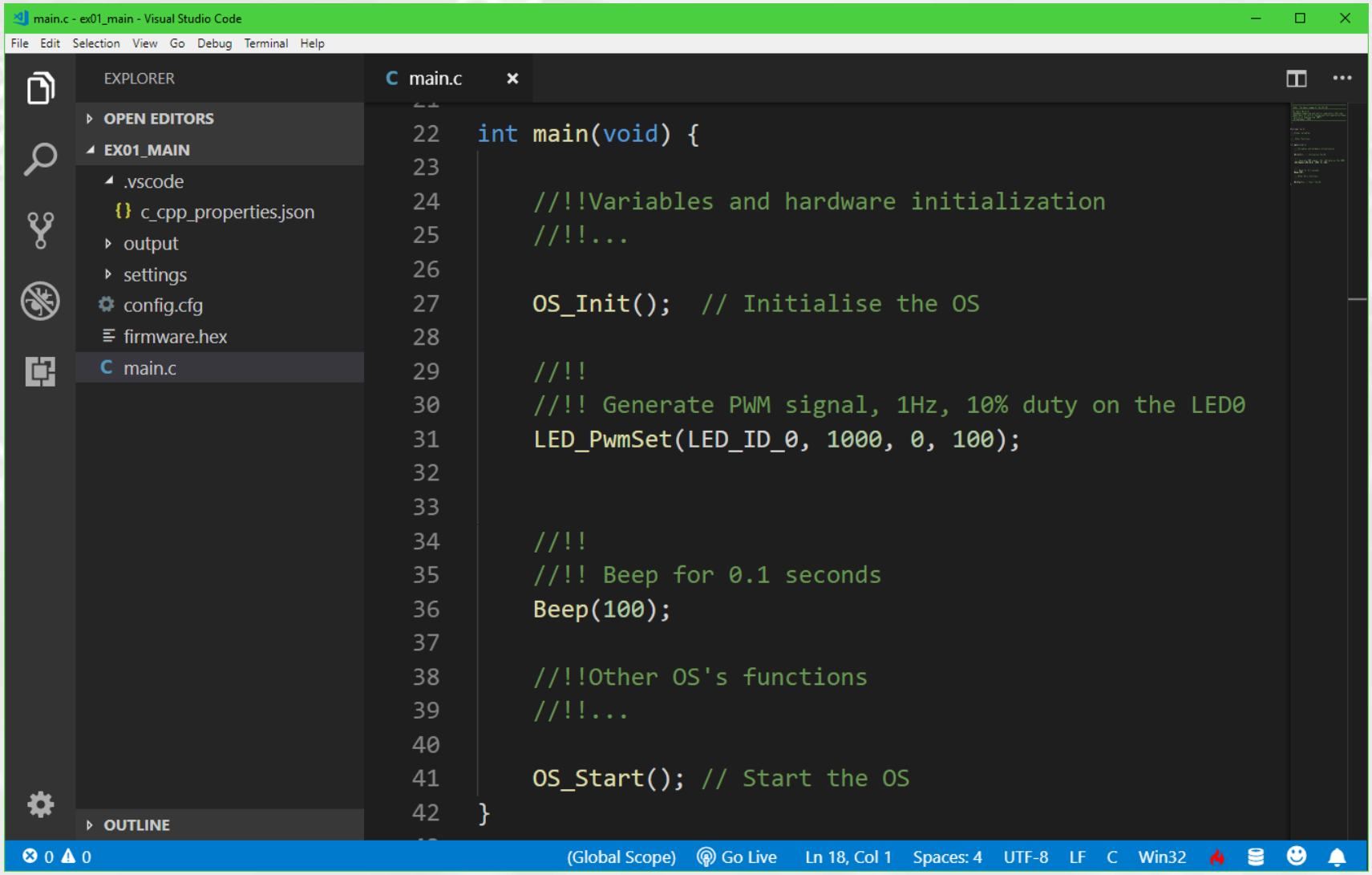


If the VS-Code is configured correctly, go to the target directory, right-click and choose Open with Code

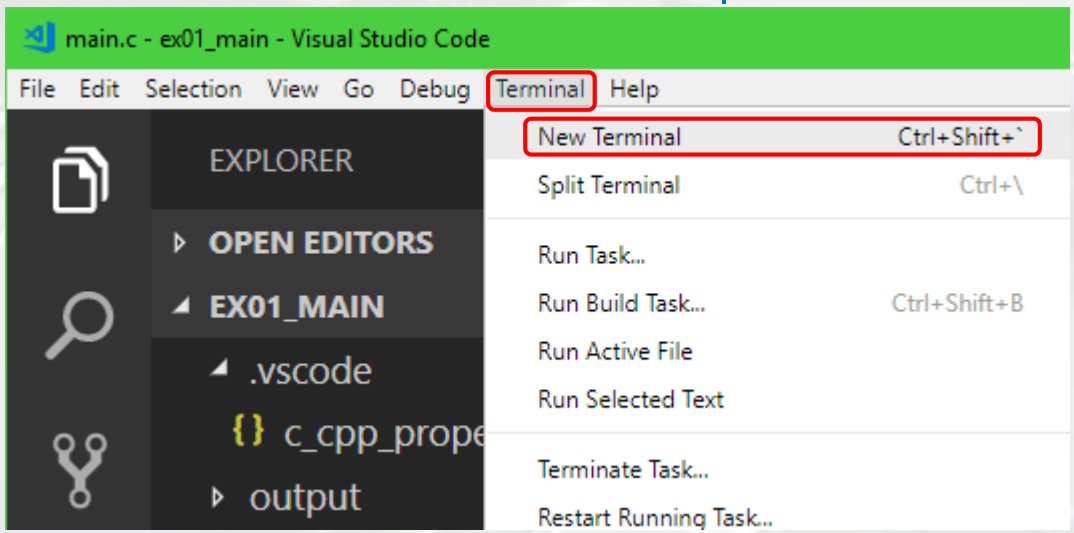


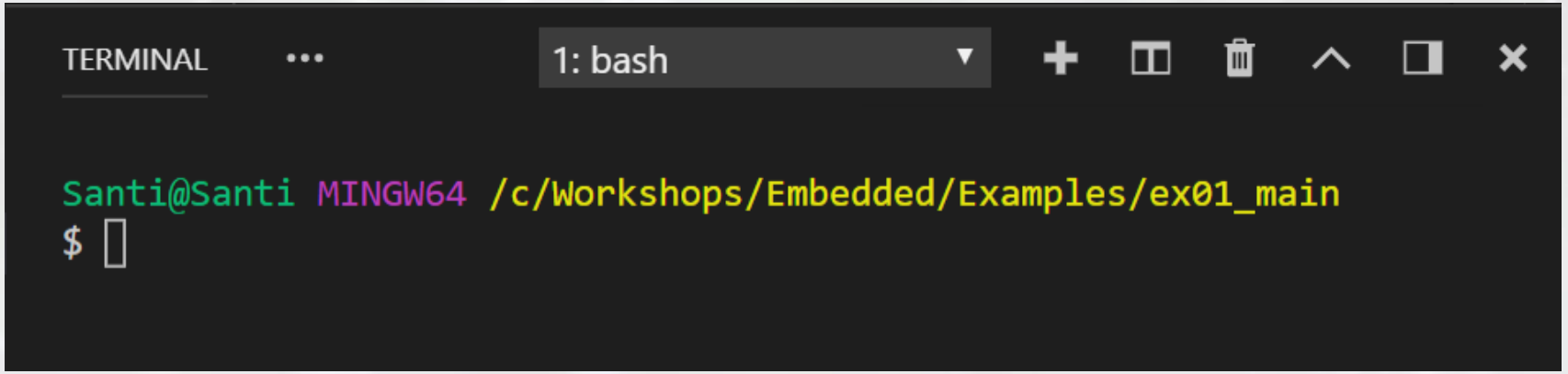
Getting Started with Embedded C-Programming

02 : Double-click on the main.c and see closely how it is written



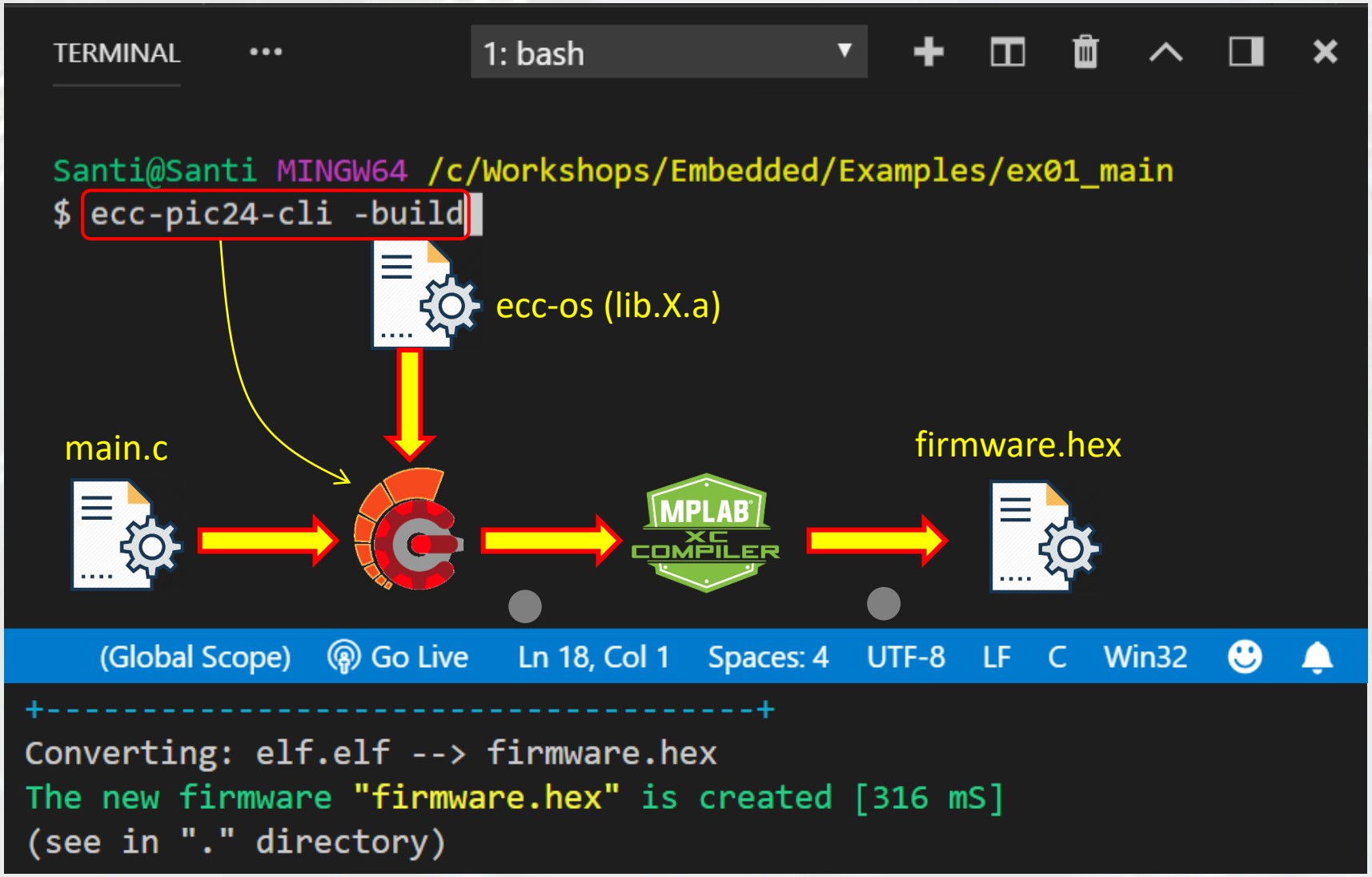
03 : On the menu bar, click **Terminal | New Terminal**





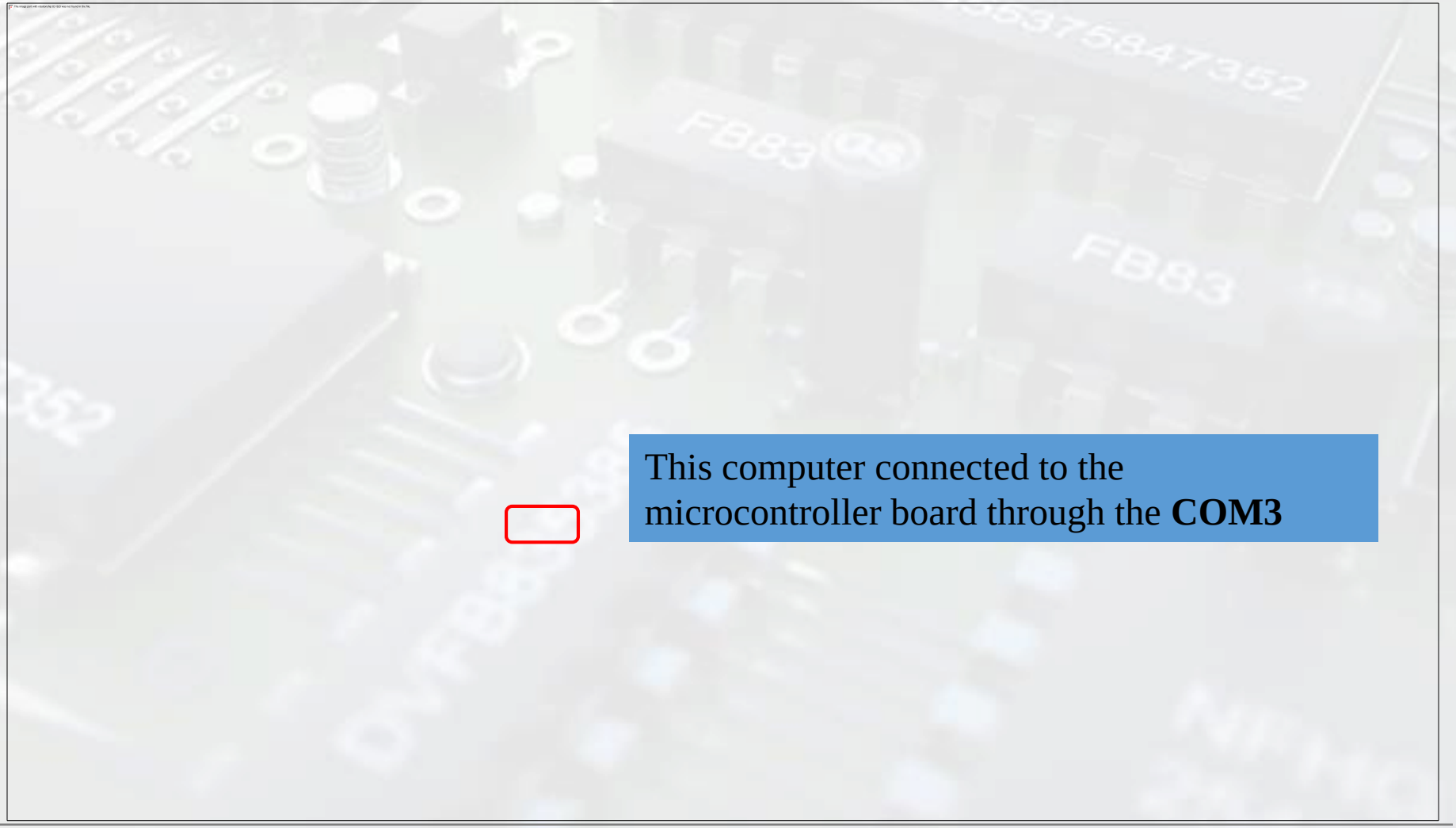
Getting Started with Embedded C-Programming

04 : Use the **ecc-pic24-cli -build** to compile the main.c to *.hex (firmware.hex)



Getting Started with Embedded C-Programming

05 : Connect the MCU board to computer using a USB cable, then check if the driver of USB-Serial is installed and ready to exchange data

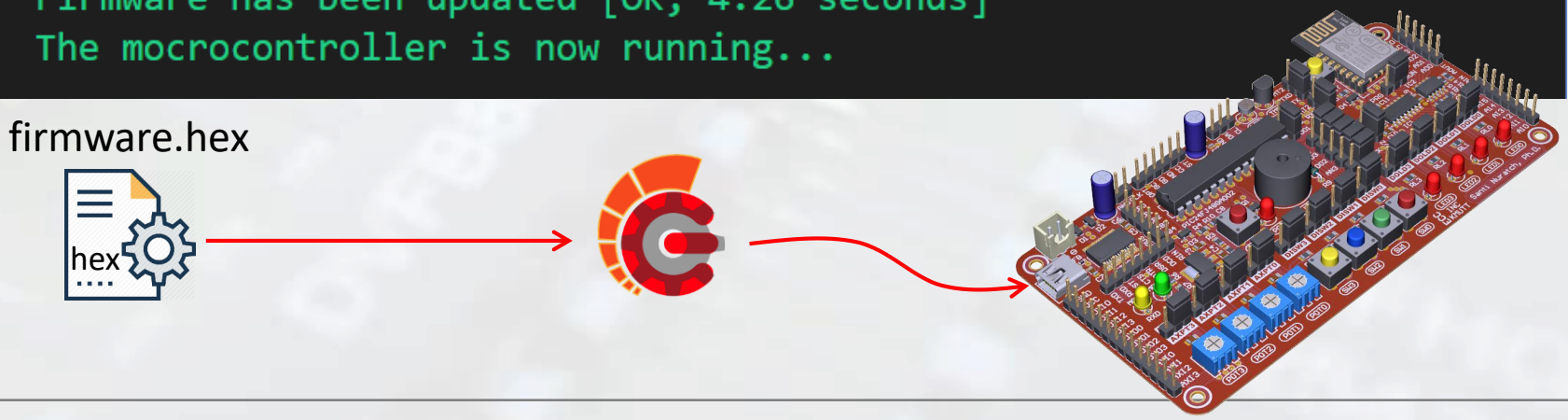


Getting Started with Embedded C-Programming

06 : Back to the terminal window of the VS-Code, and give it the command **ecc-pic24-cli -flash**

```
Santi@Santi MINGW64 /c/Workshops/Embedded/Examples/ex01_main
$ ecc-pic24-cli -flash
+-------------------------------------+
|         ECC-PIC24-CLI v1.0.7        |
|           Dr.Santi Nuratch          |
| Embedded Computing and Control Lab. |
|               (ECC-Lab)             |
```

```
Programming flash memory: 84.62 %
Programming flash memory: 92.31 %
Programming flash memory: 100.00 %
Firmware has been updated [OK, 4.26 seconds]
The mocrocontroller is now running...
```



07 : Open the **main.c** and change this line

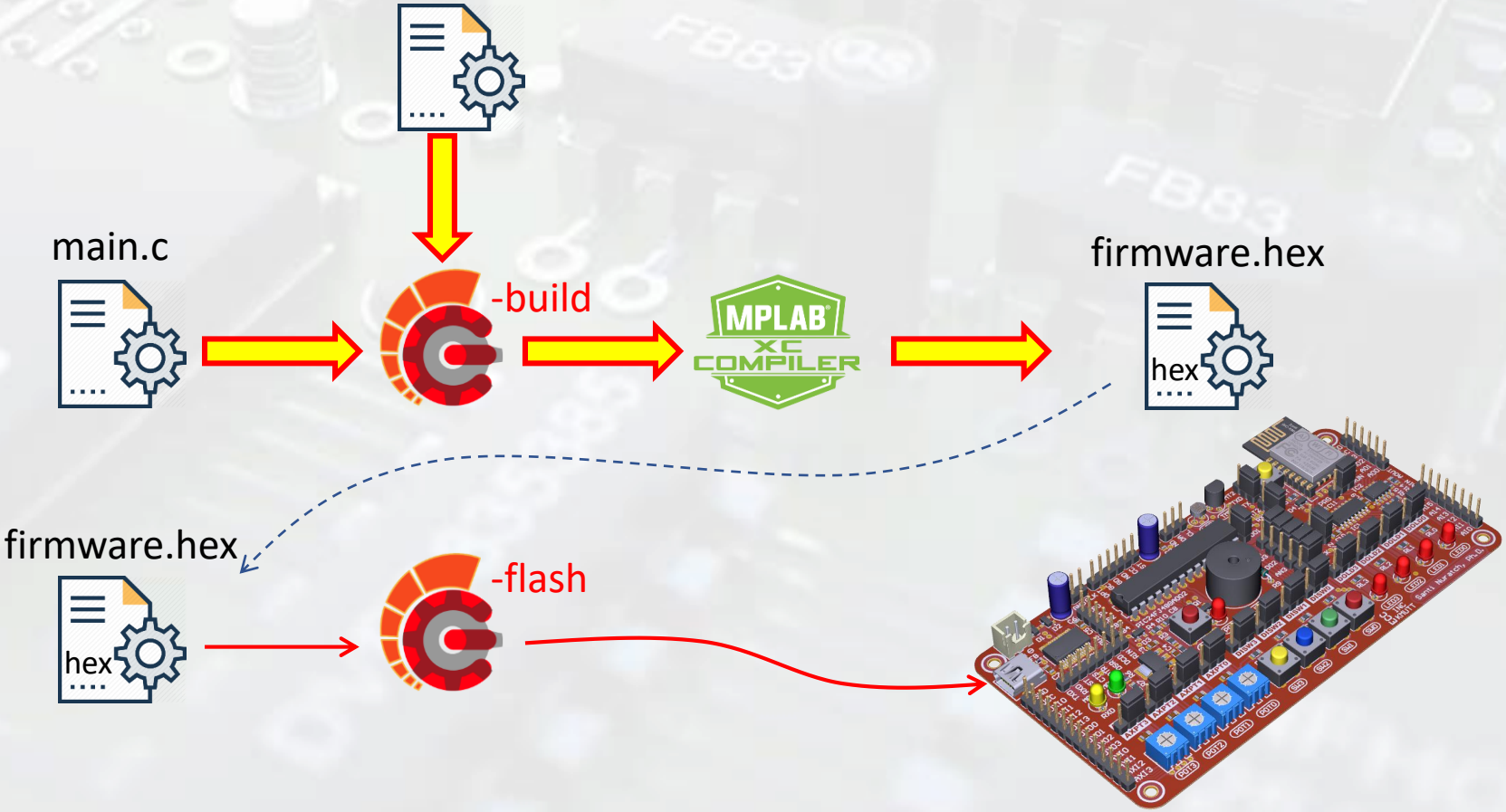
```
LED_PwmSet(LED_ID_0, 1000, 0, 100);
```

to

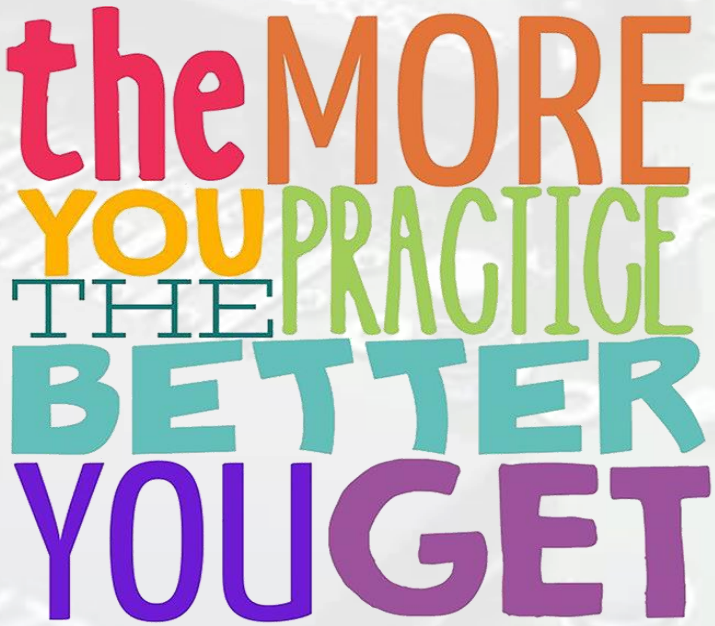
```
LED_PwmSet(LED_ID_3,  500, 0, 100);
```

08 : In the terminal window, enter the command **ecc-pic24-cli -build -flash**

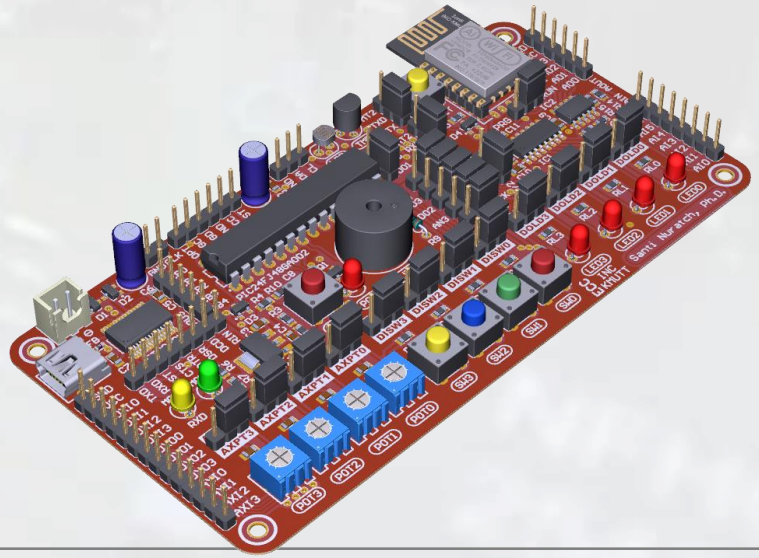
```
Santi@Santi MINGW64 /c/Workshops/Embedded/Examples/ex01_main
$ ecc-pic24-cli -build -flash
```



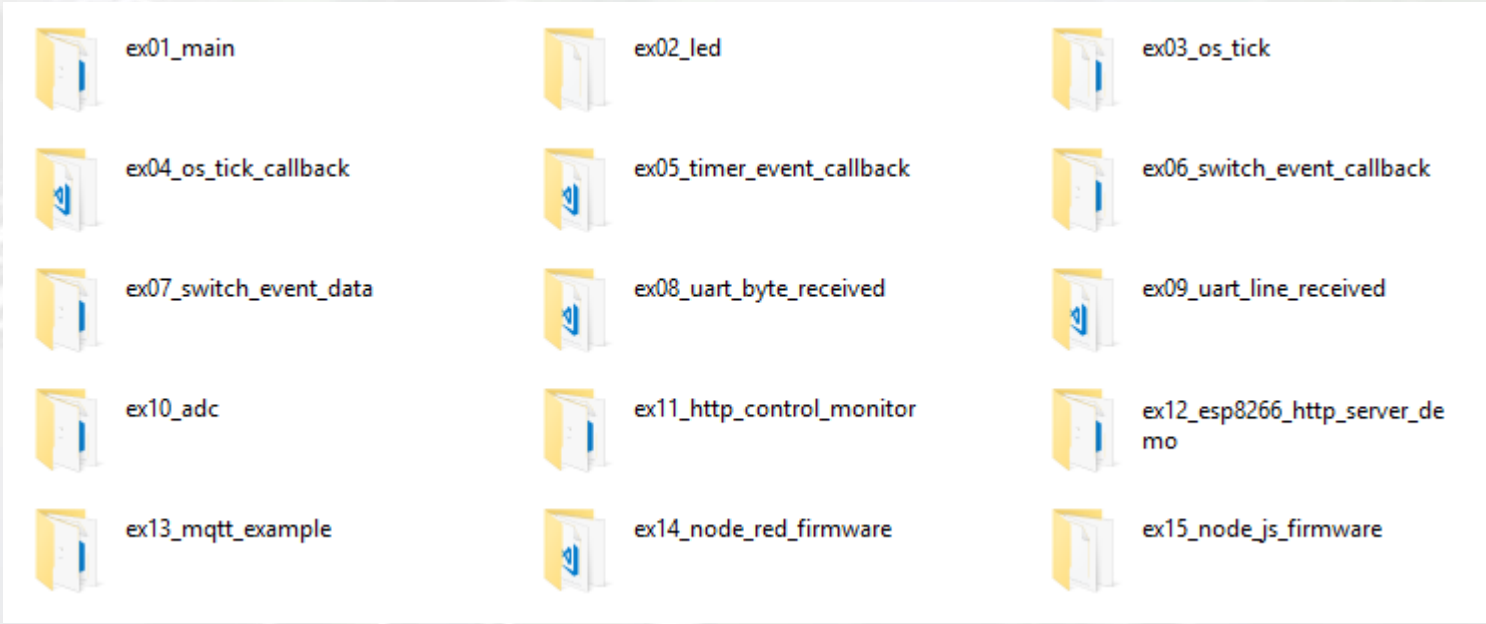
Let's Coding…

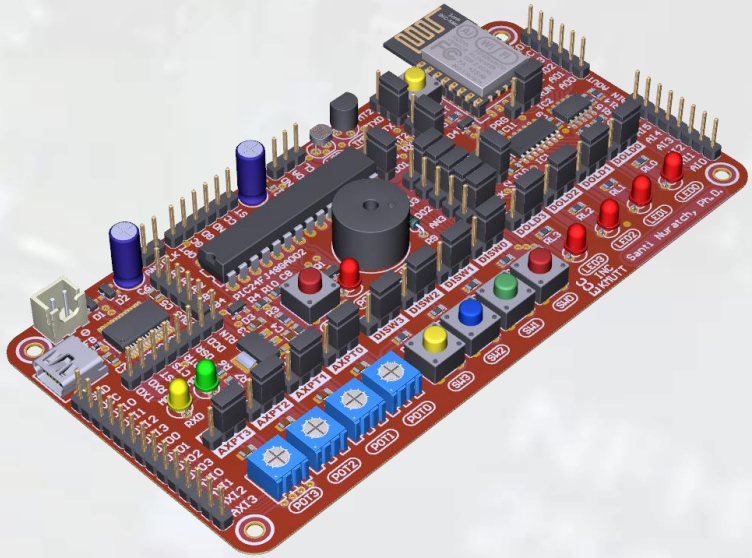


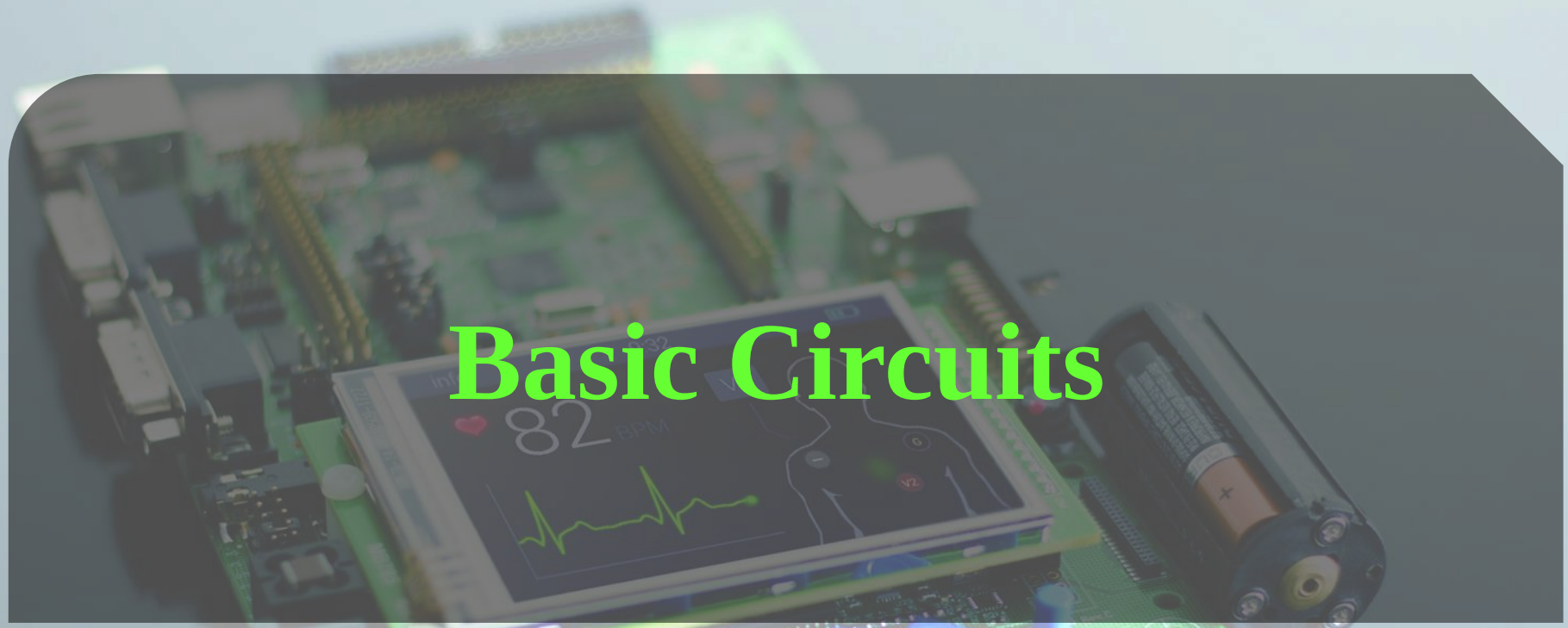

```
543  int main(void)
544  {
545      OS_Init();
546      AT_Init();
547      ESP_Init();
548      WiFi_Init();
549      Internet_Init();
550      OS_TimerCreate("AT_Service", 100, 1, AT_Service);
551      OS_WorkerCreate("WiFi_Init", Worker_ESPInitialise);
552      OS_Uart2SetLineReceivedCallback(ESP_LineReceived);
553      UART1_AsyncWriteString("\r\nMQTT Client...\r\n");
554      Beep(100);
555      OS_Start();
556  }
557
```



Learning by doing is the best way!

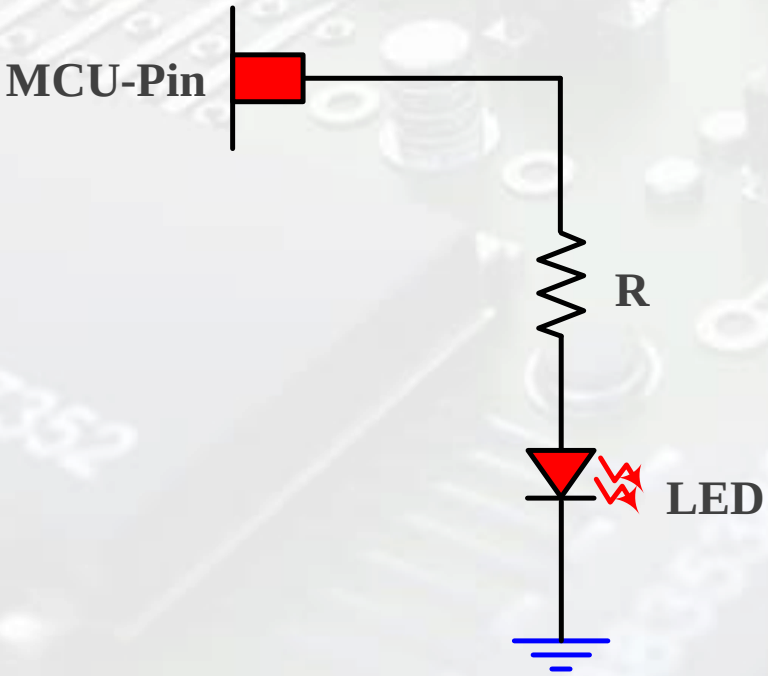




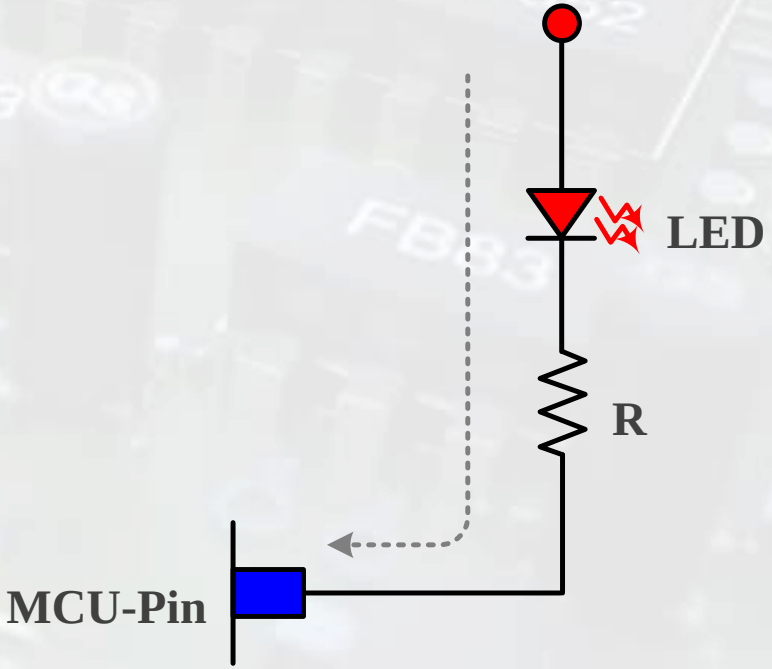

Basic Circuits

Basic Circuits

LED Circuits



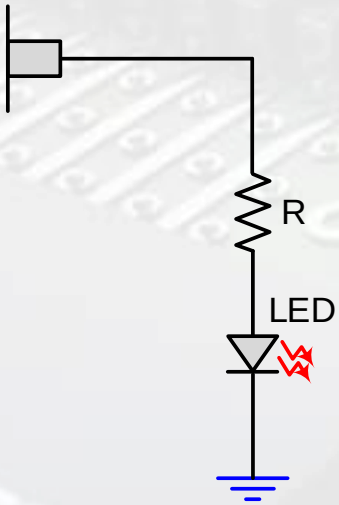
Active-High

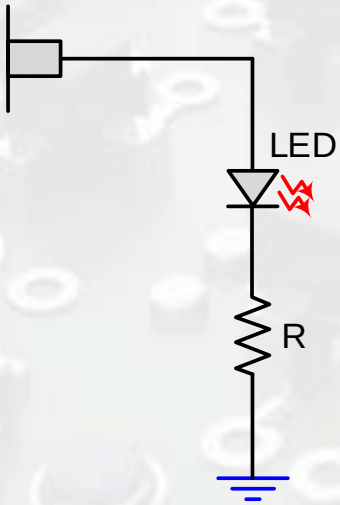


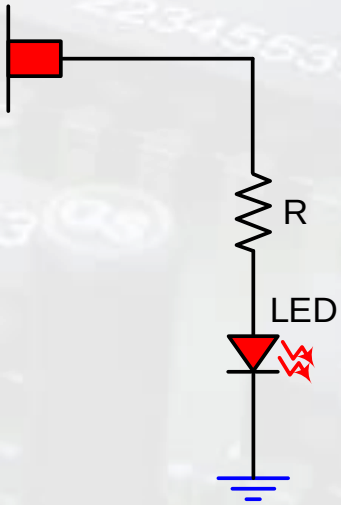
Active-Low

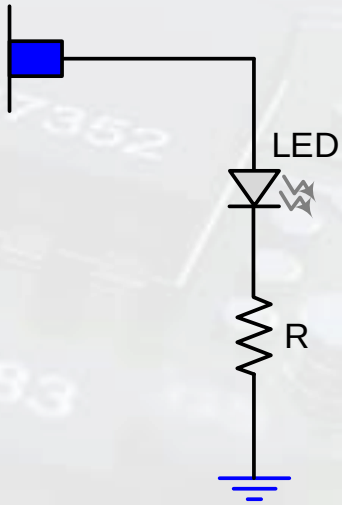
Basic Circuits

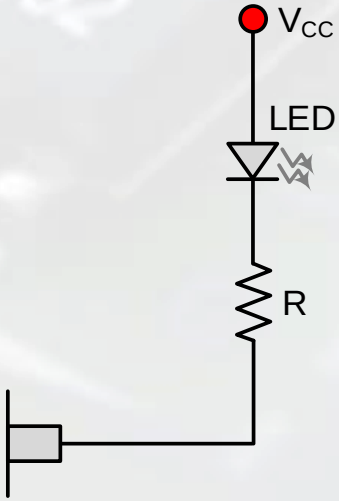
LED Circuits

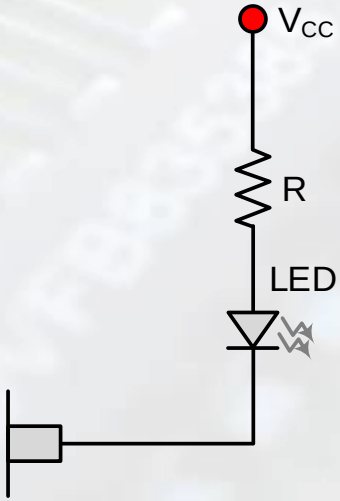


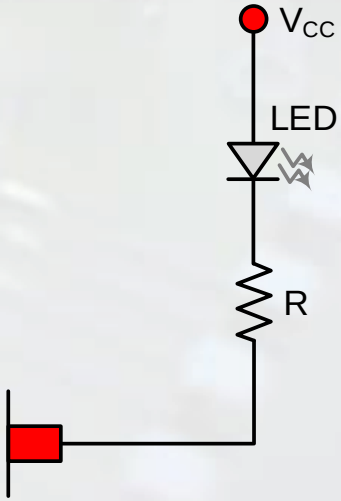


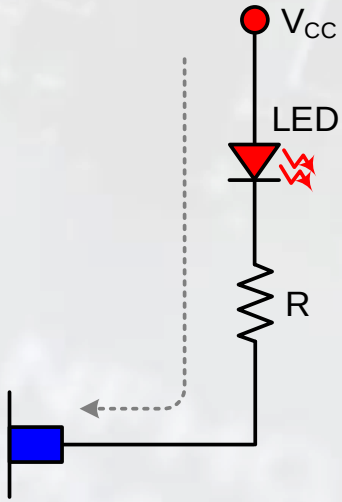






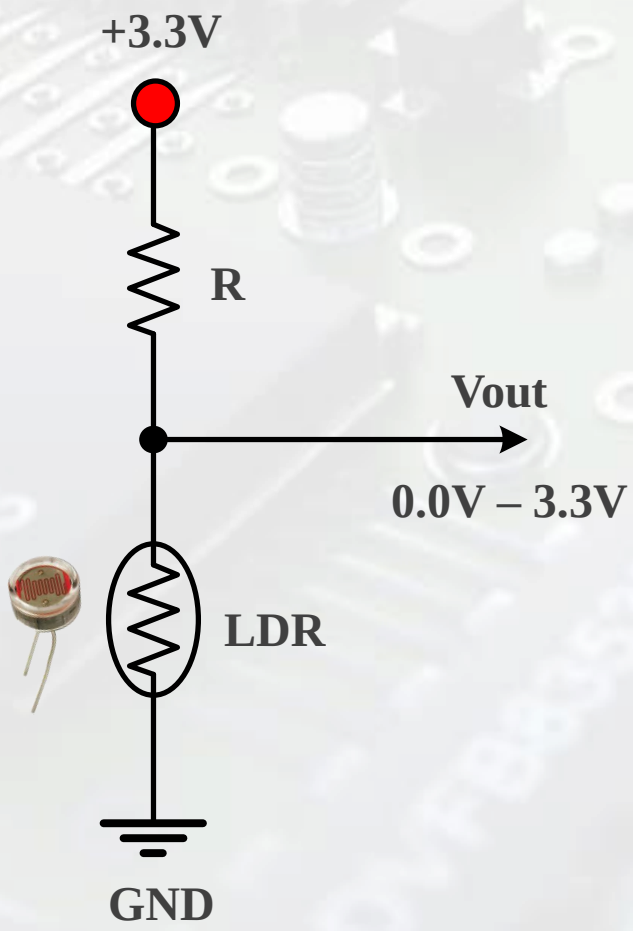


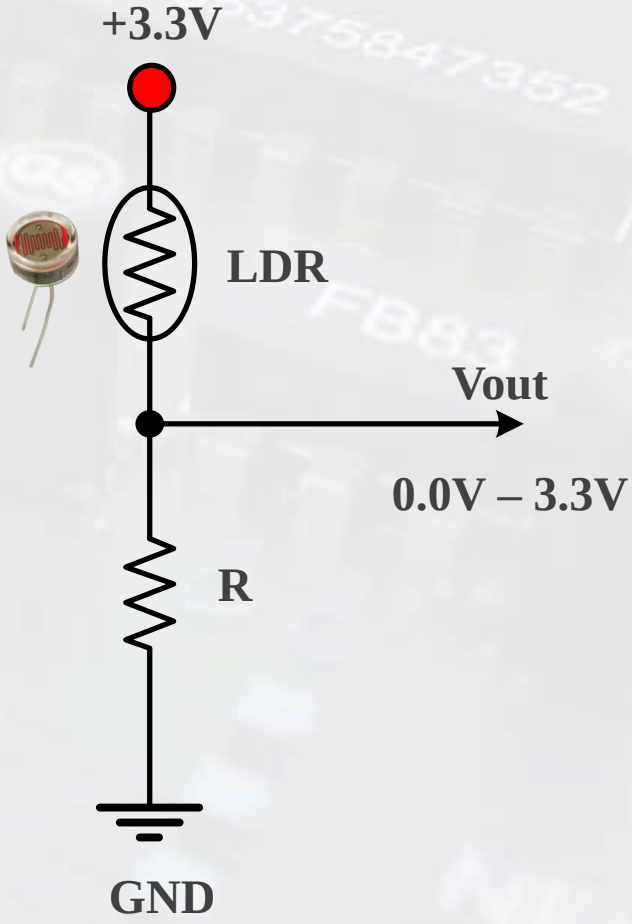




Basic Circuits

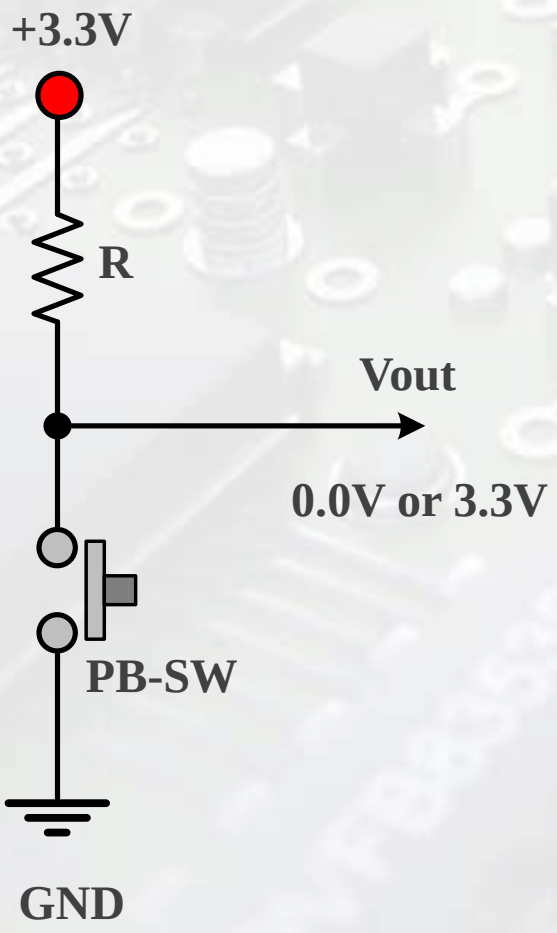
LDR (and other Variable Resistors) Circuits



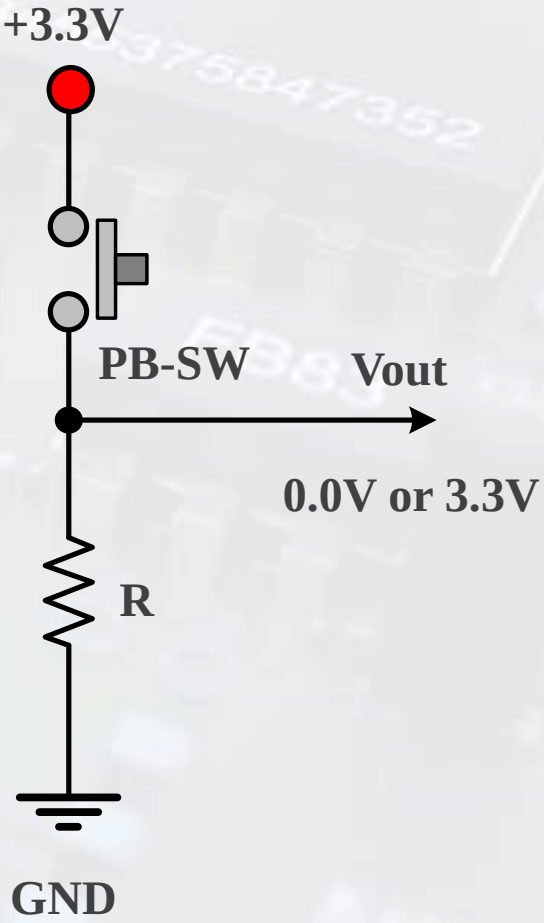


Basic Circuits

Switch Circuits



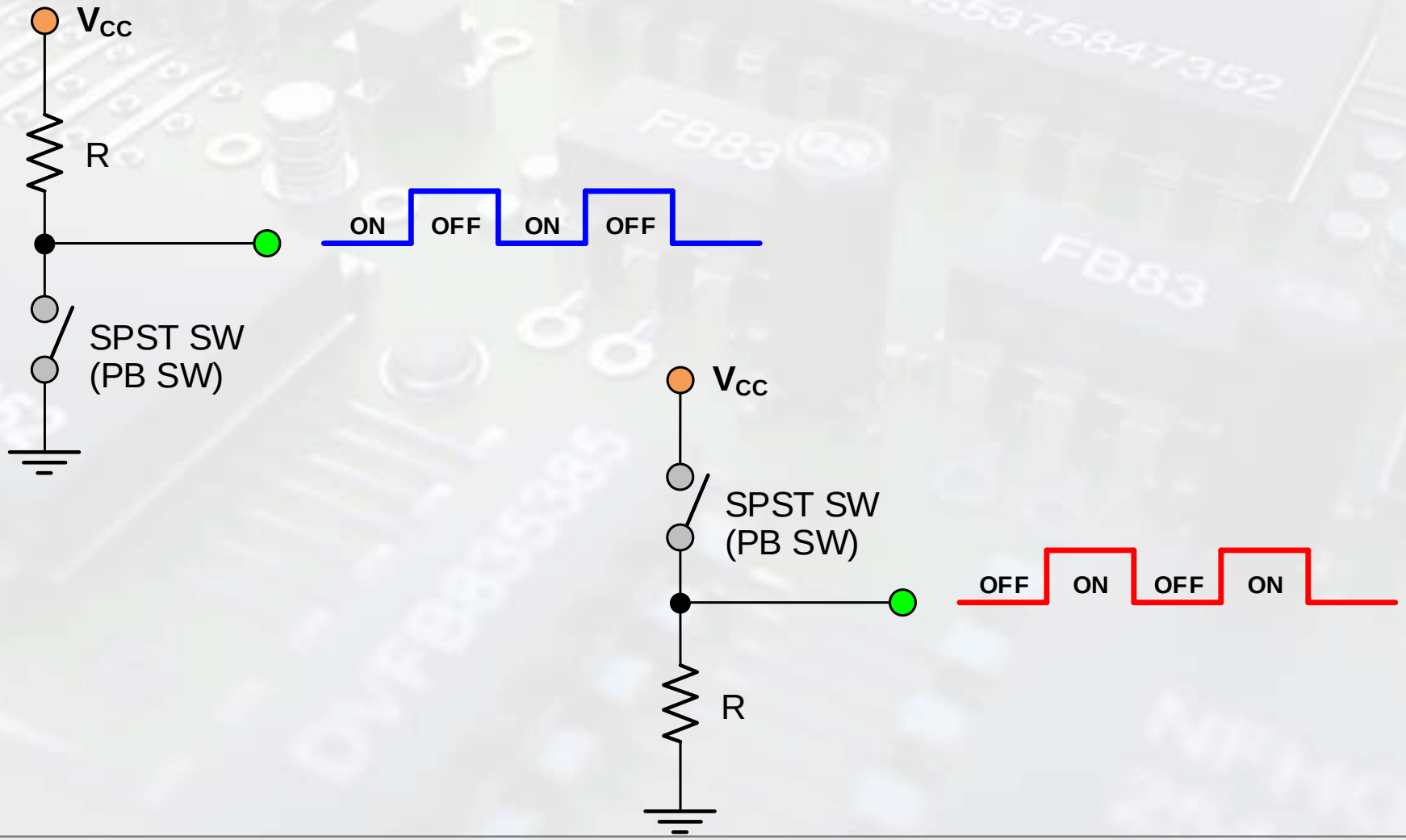
Active-Low



Active-High

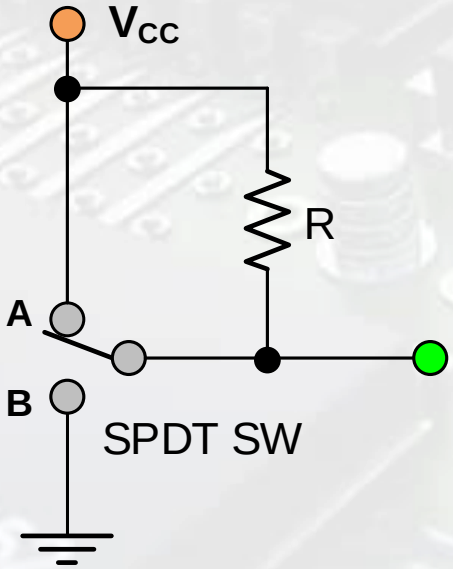
Basic Circuits

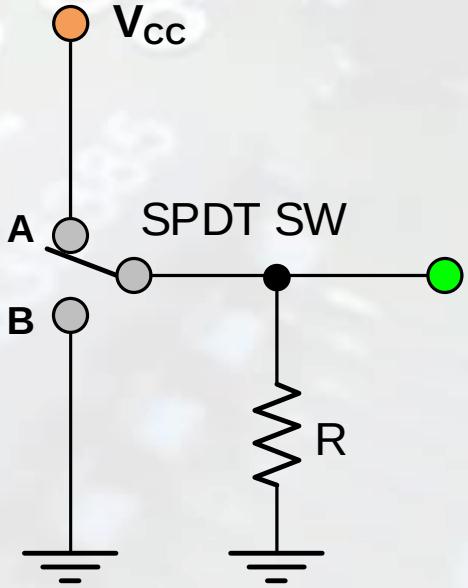
Single Pole Single Throw Switch



Basic Circuits

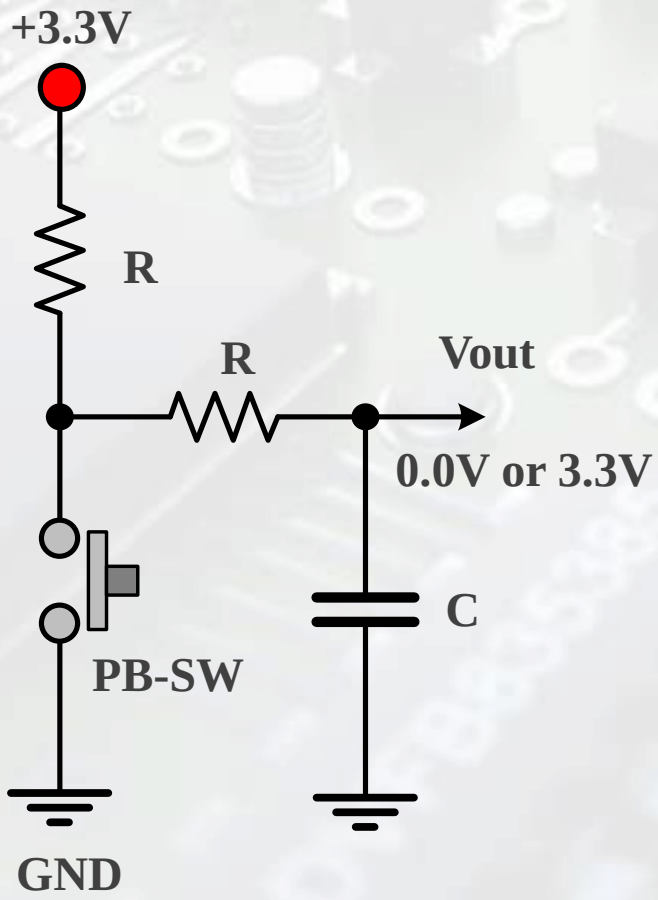
Single Pole Double Throw Switch

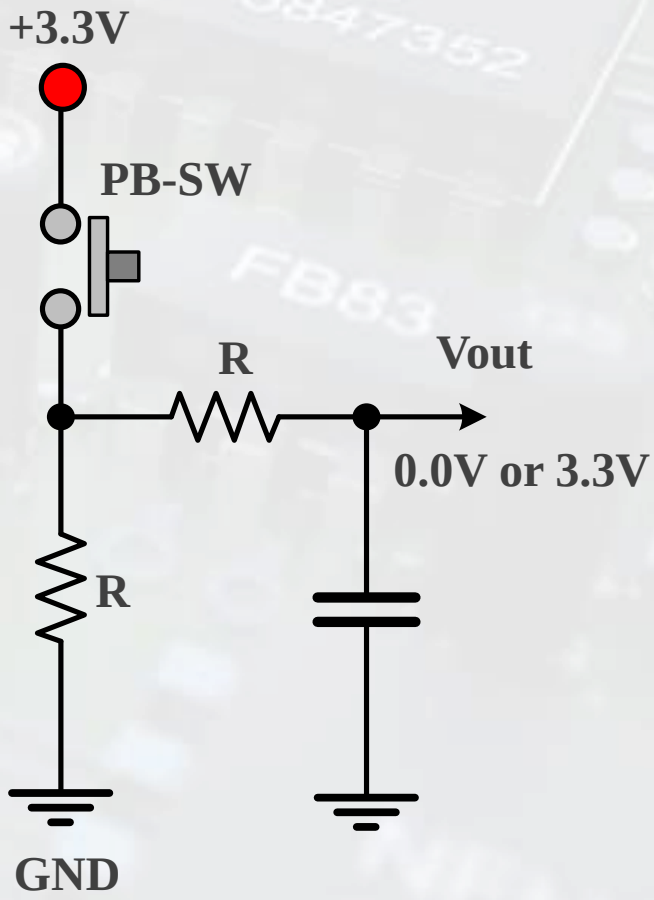




Basic Circuits

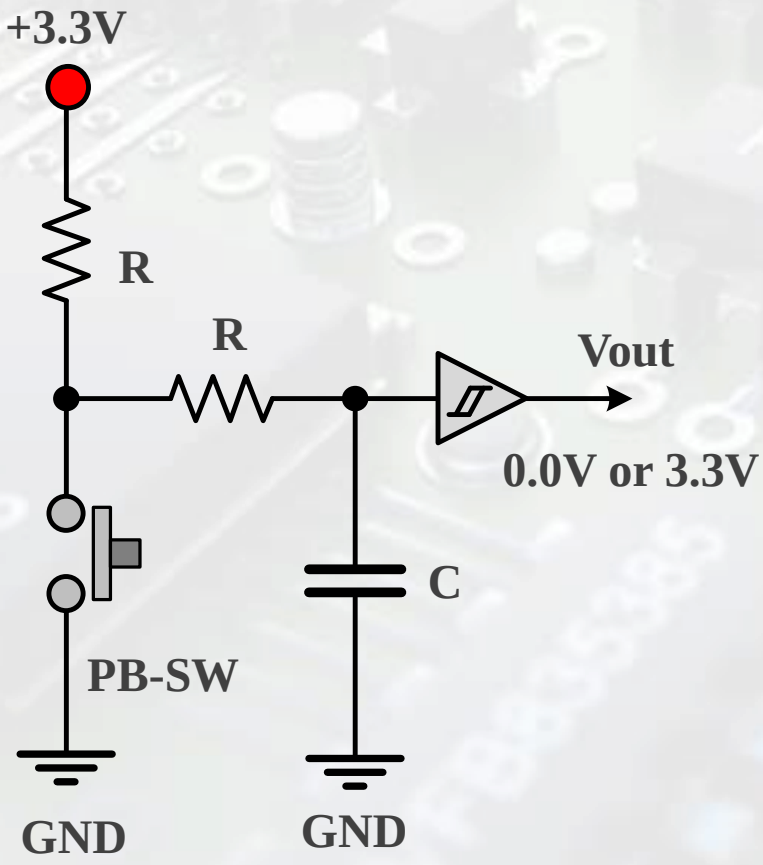
Switch and Capacitor Denouncing

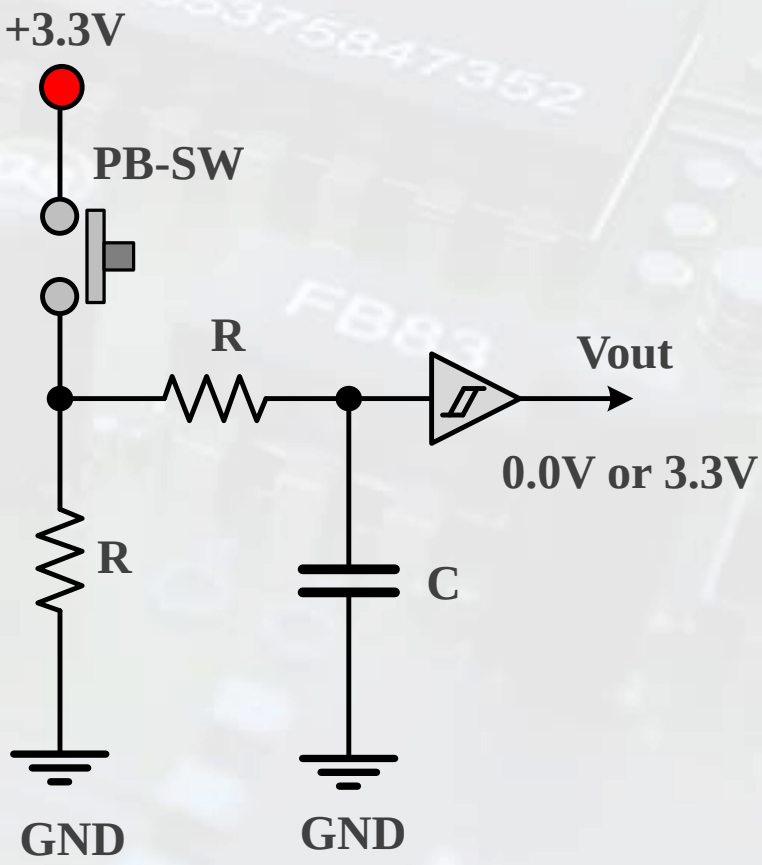




Basic Circuits

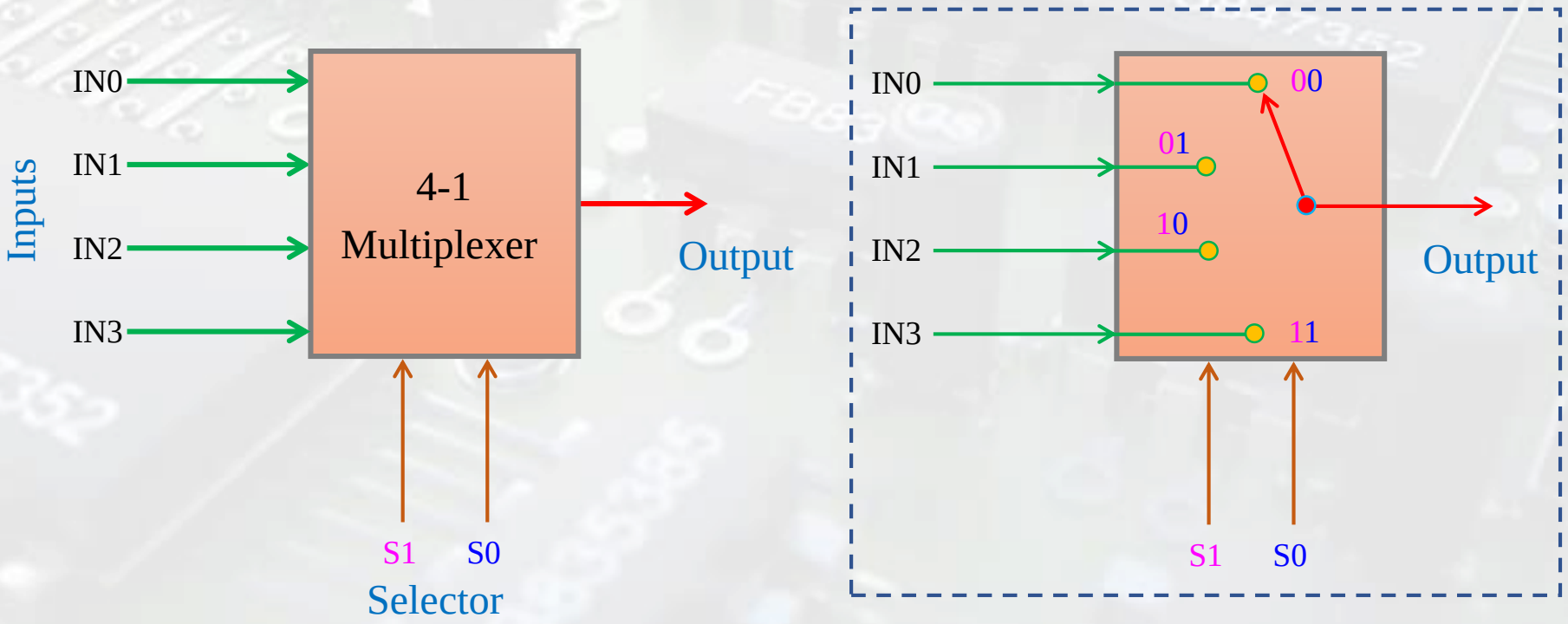
Switch and Schmitt-Trigger Denouncing





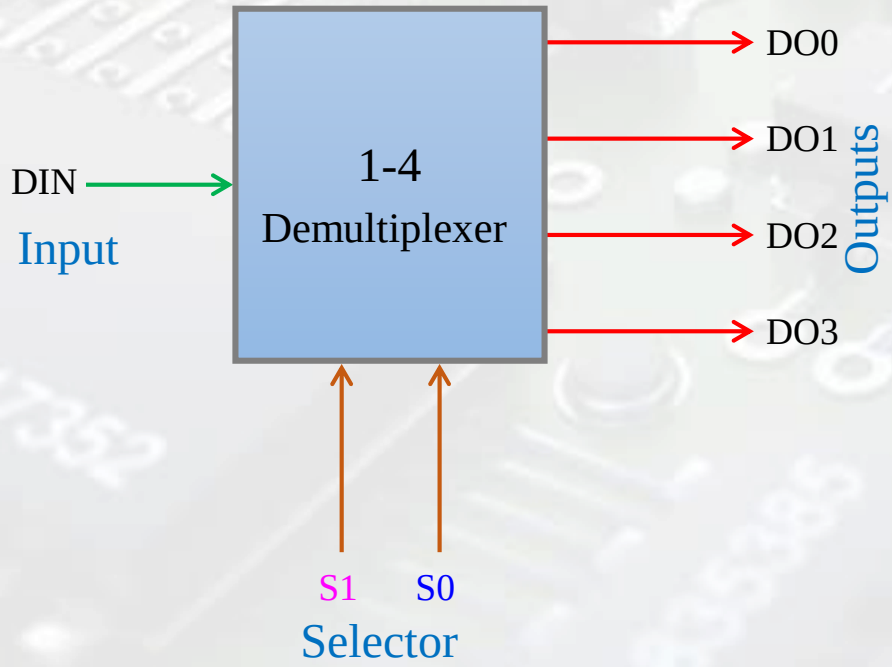
Basic Circuits

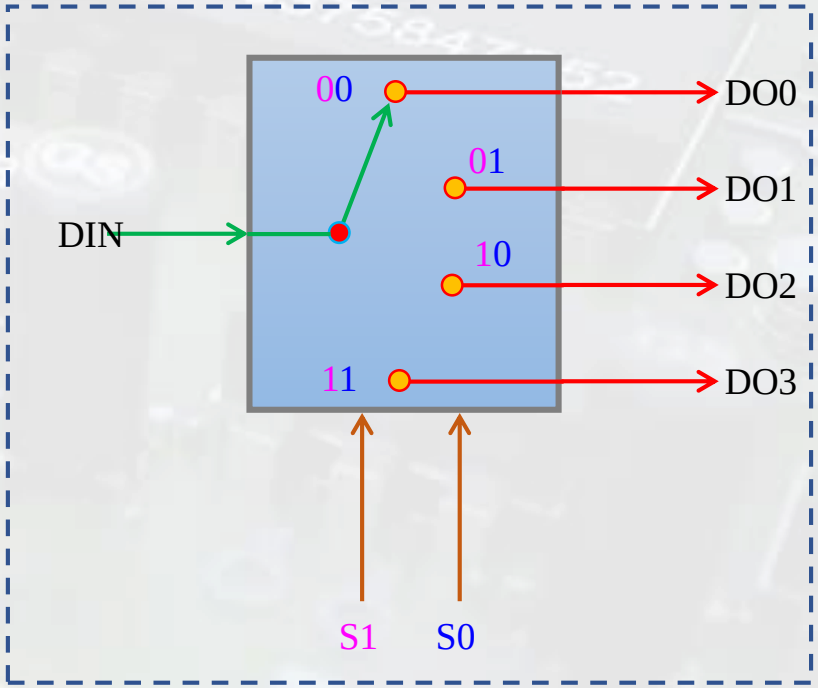
Digital and Analog Multiplexers



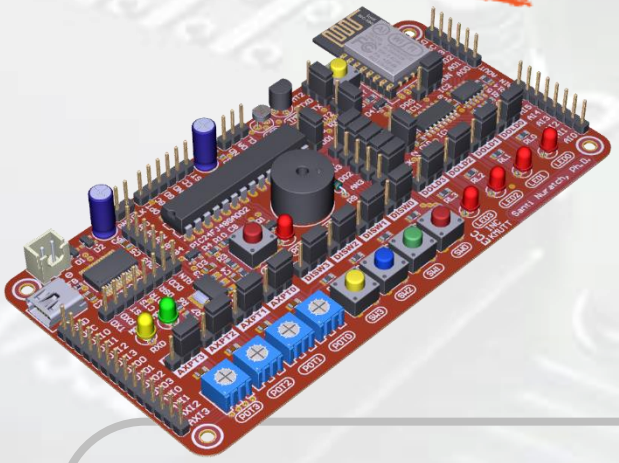
Basic Circuits

Digital and Analog Multiplexers





THANK YOU!



Santi Nuratch., Ph.D.

Embedded Computing and Control Lab. @ INC-KMUTT

santi.inc.kmutt@gmail.com, santi.nur@kmutt.ac.th

Department of Control System and Instrumentation Engineering,
King Mongkut's University of Technology Thonburi, **KMUTT**