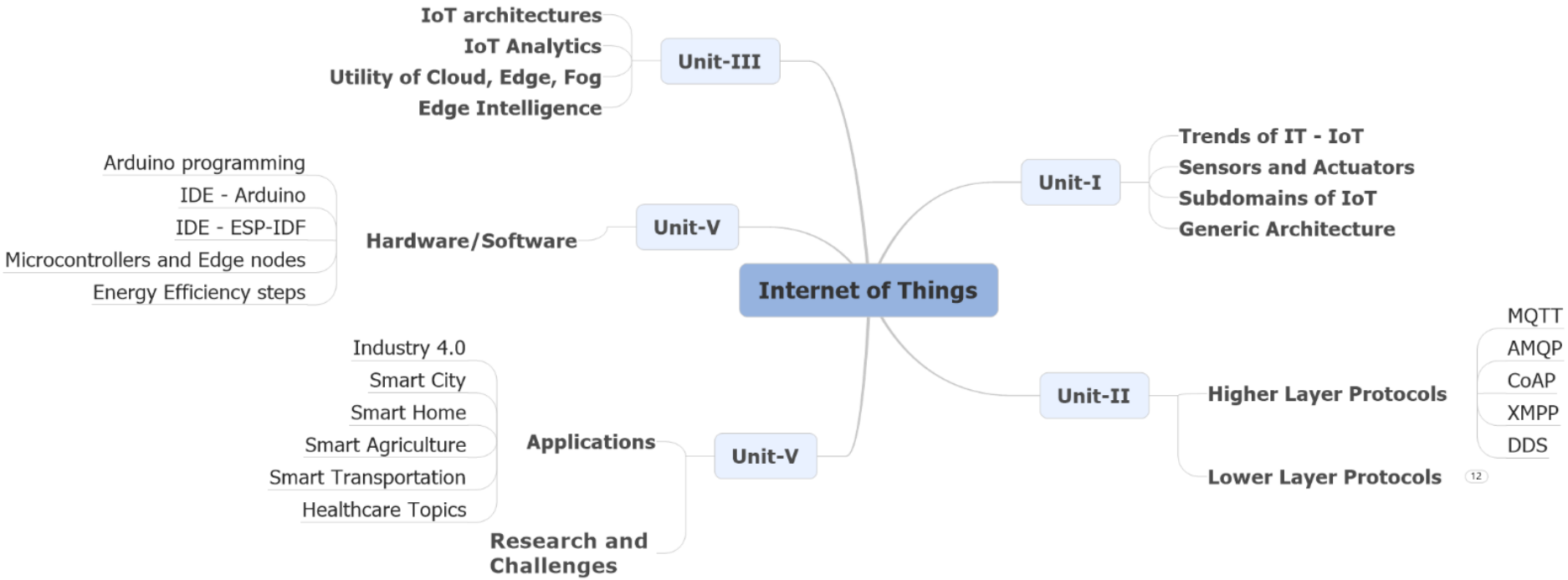
Selected Topics in Computer Architecture, Computer Networks, and Distributed Systems
(Internet of Things)
(IN3450)

–

Dr. Shajulin Benedict
shajulin@iiitkottayam.ac.in
shajulinbenedict@mytum.de

Prof. Dr. Michael Gerndt
gerndt@in.tum.de

Syllabus


Internet of Things
Unit-I
Trends of IT - IoT
Sensors and Actuators
Subdomains of IoT
Generic Architecture
Unit-II
Higher Layer Protocols
MQTT
AMQP
CoAP
XMPP
DDS
Lower Layer Protocols
12
Unit-III
IoT architectures
IoT Analytics
Utility of Cloud, Edge, Fog
Edge Intelligence
Unit-V
Hardware/Software
Arduino programming
IDE - Arduino
IDE - ESP-IDF
Microcontrollers and Edge nodes
Energy Efficiency steps
Unit-V
Applications
Industry 4.0
Smart City
Smart Home
Smart Agriculture
Smart Transportation
Healthcare Topics
Research and Challenges

- PROGRAMMING IOT DEVICES
  - A step to edge analytics

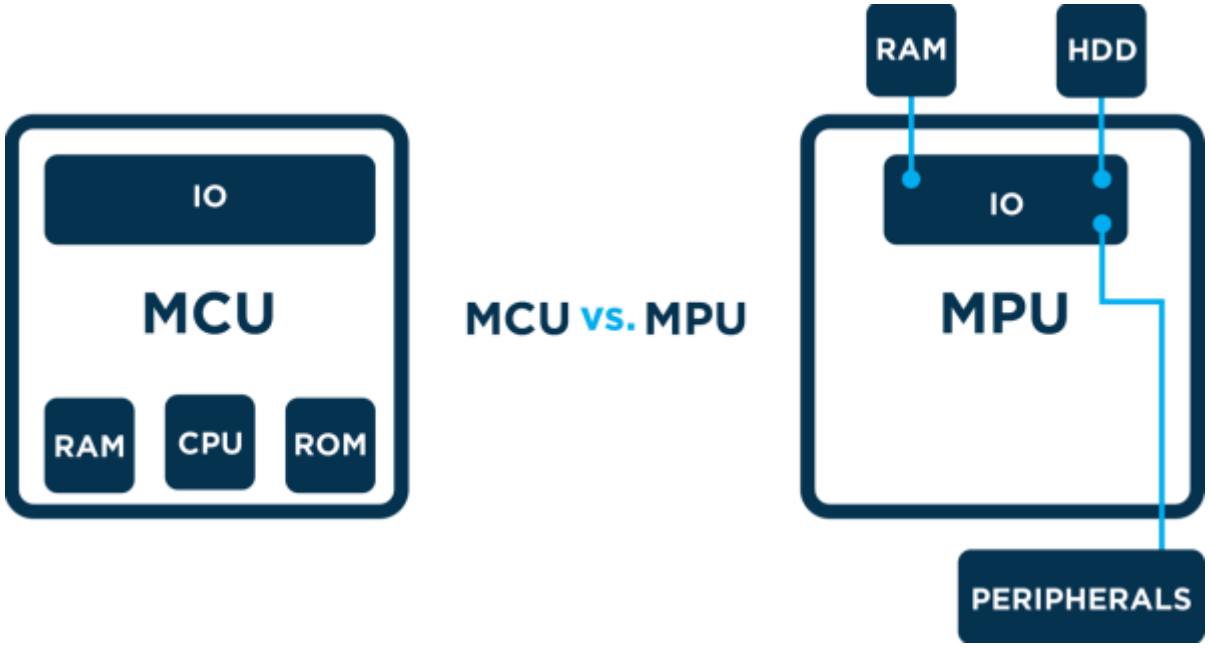
Microcontrollers

- Microcontrollers are smaller integrated IC-driven computers.
- It contains CPUs, memory, i/o peripherals on the chip.
- It is also termed as System on Chip (SoC).

- About software:
  - It includes a minimal version of OS – e.g., freeRTOS.

Microcontroller vs. Microprocessors

- Microprocessors do not have integrated I/O or memory units. They have only CPUs.

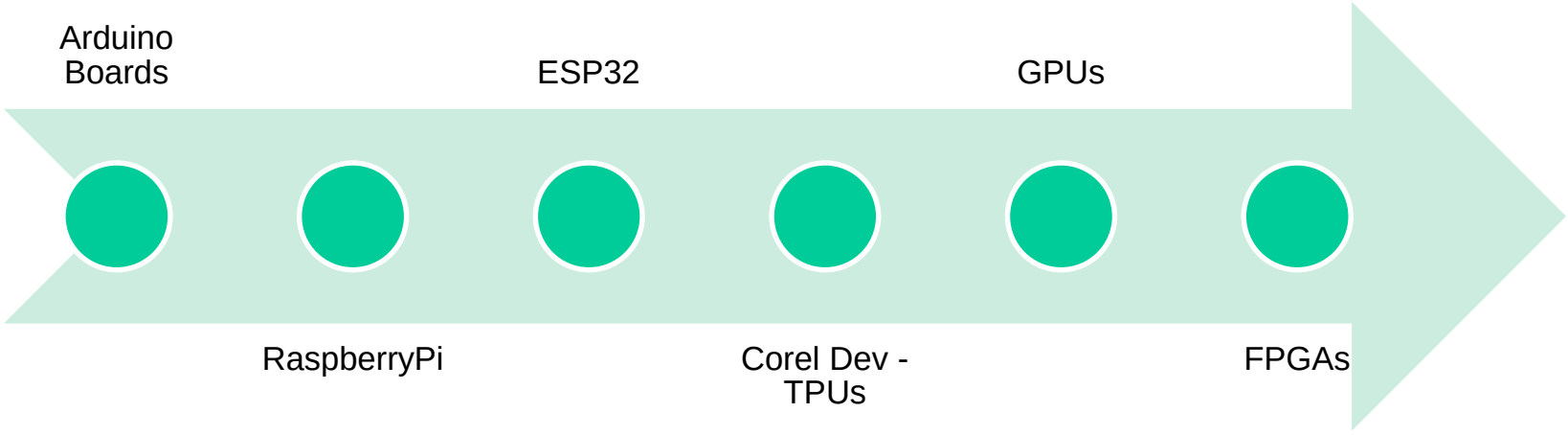


www.particle.io

Microcontroller vs. Microprocessors

Microprocessor	Microcontroller
Applied in computer system	Applied in embedded system
It includes only CPUs in an IC	CPUs, memory, IO are embedded in a chip
Many power supplies	Single power supply (low supply)

Edge Devices – For Analytics

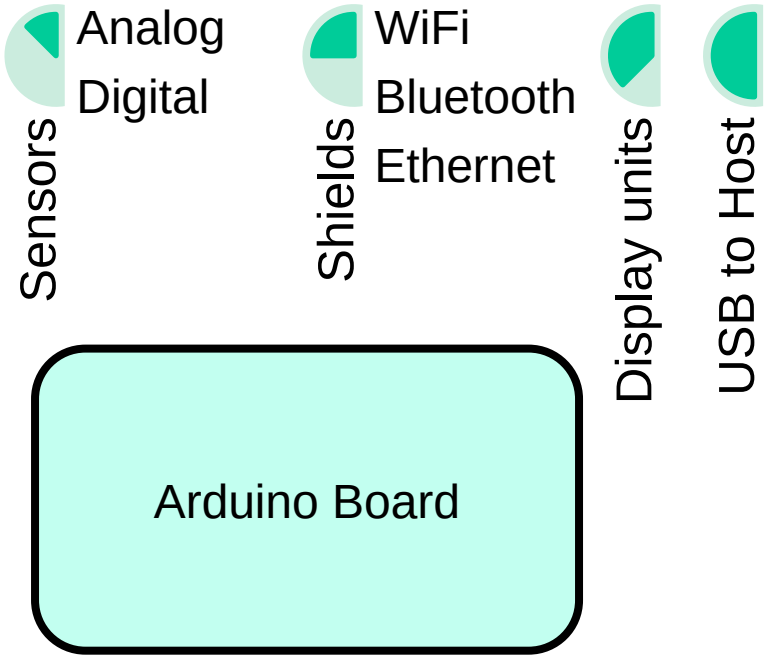


Arduino Board -- Microcontroller

- Arduino is popular because of the following:
  - Cross-platform – Linux, Windows, MacOS, ..
  - Cheap (approx. 20 euros)
  - Open-source support
  - Extensible-software/hardware
  - Low power consumption
- The main purpose of Arduino is to control things (devices)
  - i.e., by interfacing sensors and actuators with the control units.

What sensors or things are connected?

- Boards are connected to several devices…



Arduino boards

- Nano Family
  - These boards have tiny footprints with specific features.
  - i.e., exclusively for specific sensors, actuators, or so forth.
- MKR Family
  - It is a series of boards (each specifying unique features).
  - Boards are separately available for WiFi, LoRa, Bluetooth, and so forth.
- Classic Family
  - A classic ones (most commonly utilized in Arduino projects).
  - E.g., Arduino UNO

Arduino UNO – Popular Board

- Voltage – 5 V
- Speed – 16 MHz
- Digital and Analog pins
- UART – Universal Asynchronous Receiver/Transmitter
- Interface - USB

Arduino UNO board – An anatomy

1. USB socket
2. DC power socket
3. Reset switch
4. Builtin LED
5. Digital IO pins – 14
6. ATmega328
7. Analog inputs
8. Power connectors

Digital pins

USB Connector

Power Connector

Classic Arduino boards

- Similar to UNO, there is Arduino Leanardo (same size and pins)
- There also exists a few Arduino boards that are huge in size (with more number of pins and connections)
  - Arduino Due
- There also exists a few TINY Arduino boards with lesser number of pins than UNO
  - Eg. LilyPad -- mostly, utilized for textiles.
  - EtherTen (or, Ether10)
  - Leostick

- Note: Some boards are not utilized anymore…!

Arduino Programming

- Arduino programs are often developed using Arduino IDE.
- The Arduino IDE is an open source software that is used to program the Arduino controller board.
- Based on the variations of C and C++ programming languages.
- Arduino IDE is available in their websites.

Arduino Setup

- Power the board by connecting it to PC via. USB
- Launch the Arduino IDE
- Set the board type and the port for the board.
- The program coded in Arduino IDE is mentioned as a Sketch.
- Sketch consists of
  - Setup (similar to main() in c)
    - The program starts here.
    - I/o variables, pin modes are represented here.
  - Loop
    - Iterates the specified task in the program
  - The example program prints Hello Arduino several times (iteratively).

Arduino IDE

- Choose the appropriate board from Board manager.
- Choose the appropriate PORT for your connection.
- Know where is the SERIAL MONITOR
  - Important to review the program
- Arduino Buttons

  - Verify – checks the code for compilation errors
  - Upload – uploads the final code to the controller board
  - New – creates the new blank sketch;
  - Similarly, open and close…

Arduino datatypes

Data Types	Size in Bytes
boolean	1
char	1
unsigned char, byte, uint8_t	1
int, short	2
unsigned int, word, uint16_t	2
long	4
unsigned long, uint32_t	4
float, double	4

Similar to C/C++

Arduino function libraries

- Several contributory function libraries exist.
- For eg. pinMode()
  - This function sets the functionality of the pin

```
void setup() {
  pinMode(13, OUTPUT);    // sets the digital pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH); // sets the digital pin 13 on
  delay(1000);            // waits for a second
  digitalWrite(13, LOW);  // sets the digital pin 13 off
  delay(1000);            // waits for a second
}
```

  - The code makes the digital pin 13 an OUTPUT and toggles it by alternating between HIGH and LOW at one second pace.

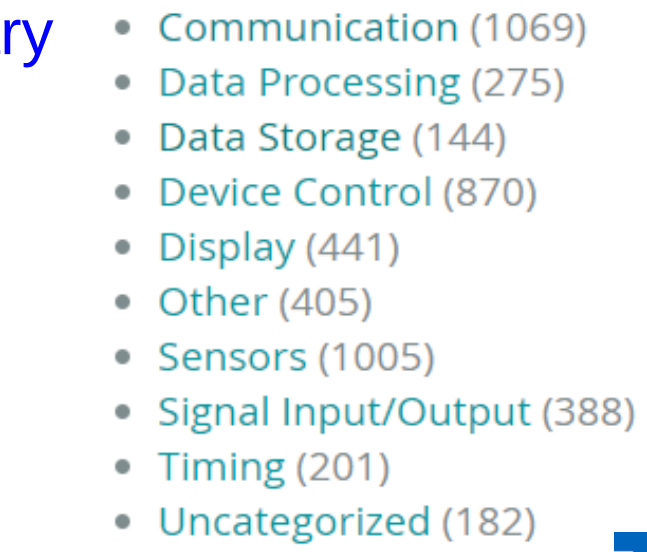
Other useful functions

- digitalWrite() – writes high or low to the digital pin
- analogRead() – reads from the analog pin
- Character-related functions
  - isupper)()
  - Isspace()
  - Isalpha()
  - Isdigit()
  - Islower()
- Delay() function is most commonly used in the sketches.

- Eg programs → LED blink example (most common)

Some useful libraries

- EEPROM – for storing data in EEPROM memory
- Ethernet – for network programming
- Firmata – the serial communication standard for Arduino to computer.
- SD – for reading and writing SD flash memory cards
- Wire – for I2C communication and peripherals
- FFT – Frequency analysis library
- Wifi – for WiFi network access



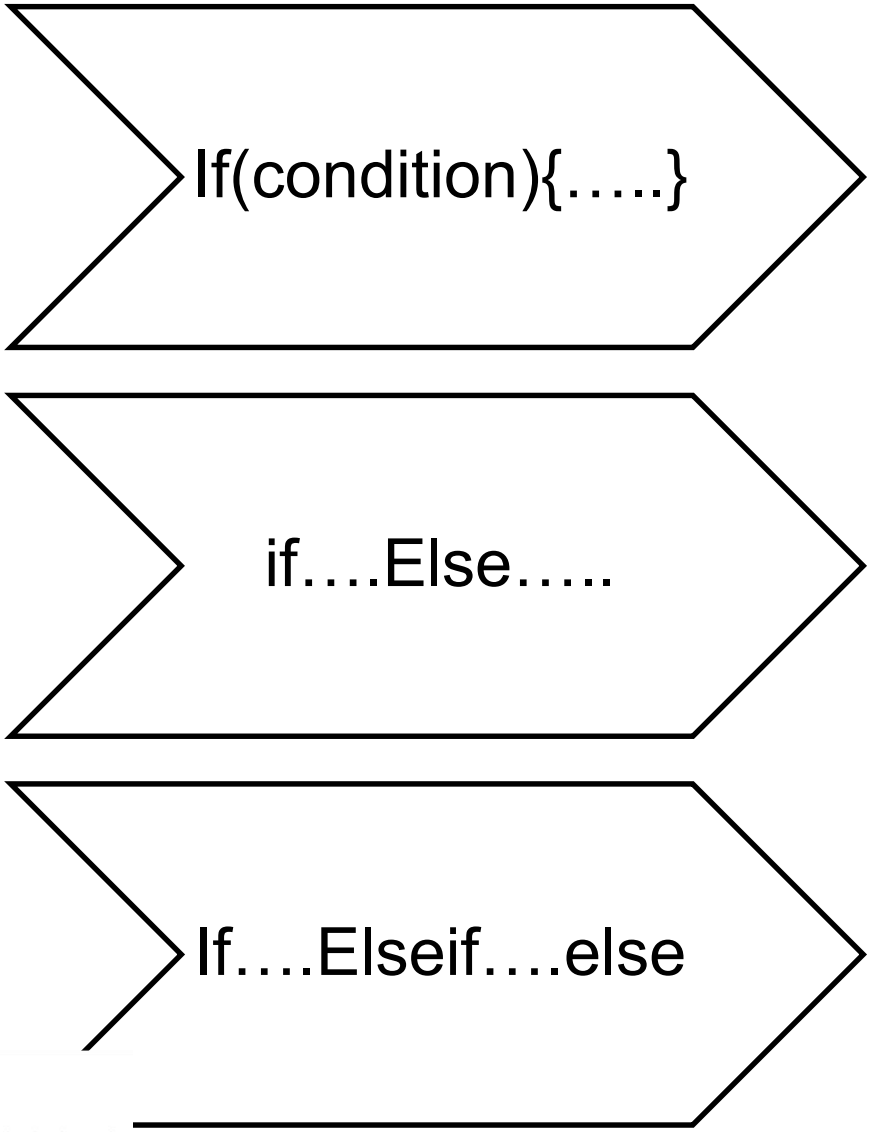
Arduino Programming

- Basic operators

Arithmetic Operators	+, -, *, /
Boolean Operators	||, !, &&
Comparison Operators	==, !=, <, >, <=, >=
Bitwise Operators	|, &, ^, <<, >>

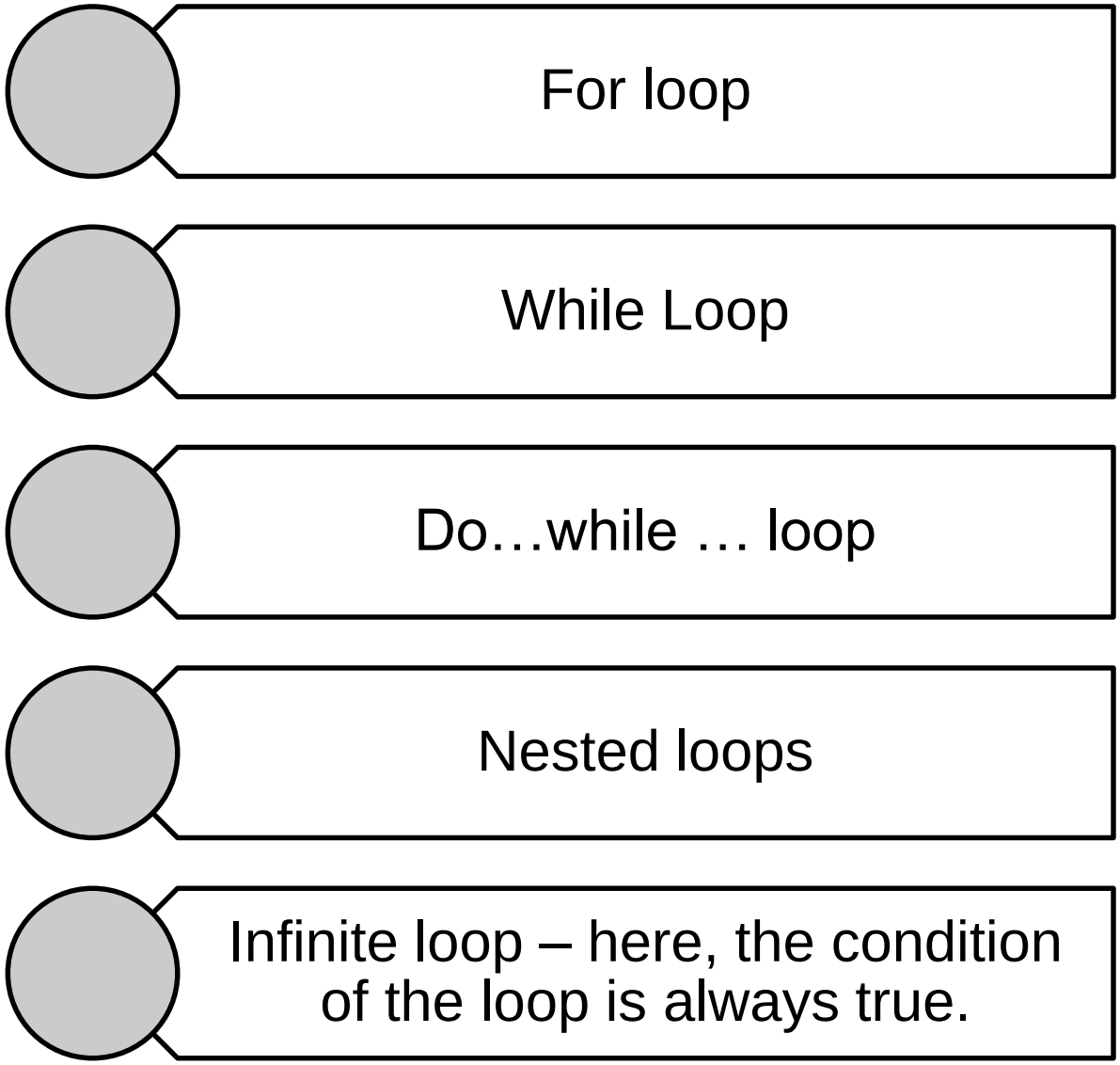
Operators are almost similar to C/C++

Control Statements




```
Switch(choice){
Case 1: ….
Case 2: ….
….
Case n: ….
Default: ….
}
```

Loops


For loop
While Loop
Do…while … loop
Nested loops
Infinite loop – here, the condition of the loop is always true.

String and arrays

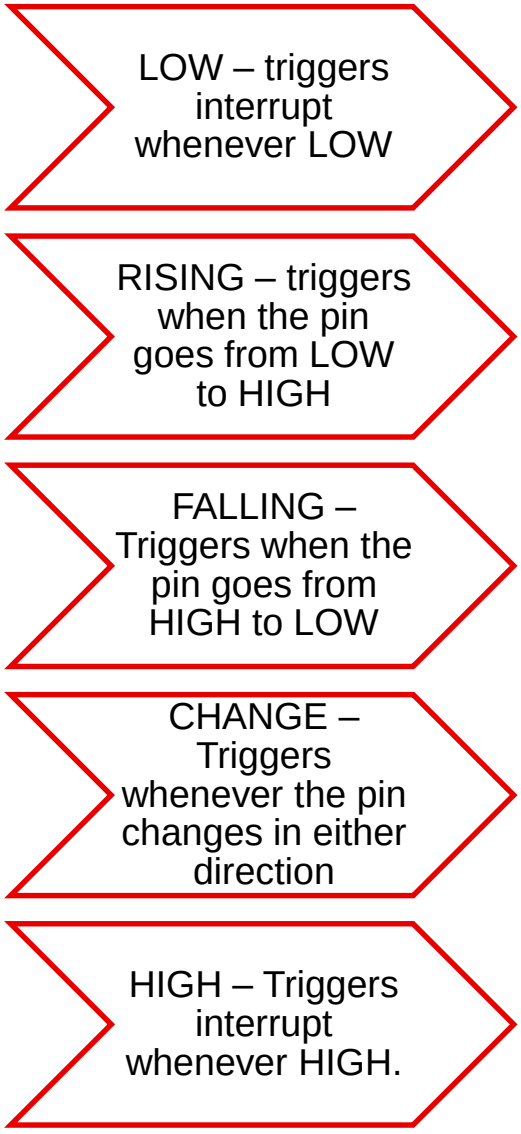
- Declaration using array
  - char str[4];
  - char str[] = “abcd”;
- Declaration using string
  - string str = “abcd”
- Functions of string
  - str.ToUpperCase() – changes all characters to upper case
  - str.length() – returns the length of the string
- Some math functions
  - exp(double val);
  - log(double val);
  - square(double val);

Interrupts

- Interrupts are external signals that allow microcontrollers to see if anything has happened.
- An interrupt blocks the current running process to process that external signal.
- Hardware interrupts
  - Hardware level interrupts
  - Eg. See figure.. A button is pressed!
  - Here, we continually check inputPin
  - if external signal occur,
    - there is a change.
- Interrupt pins
  - Arduino UNO – 0 is named as D2
  - Ardunio UNO – 1 is named as D3 (NOTE: only two interrupts are there).

```
void loop
{
if(digitalRead(inputPin) == LOW){
//do something
}
}
```

Interrupt Modes



How fast is Arduino?

- As we know, ArduinoUNO is clocked at 16MHz.
- A test executed on laptop and arduino for executing 2^9 instructions in the for loop has revealed that the ArduinoUNO is around 400 times slower than the laptops.
- Comparing the same benchmark on different boards:
  - UNO : 28 seconds
  - Leonardo : 29 seconds
  - MiniPro : 28 seconds
  - Mega2560 : 28 seconds
  - Due : 2 seconds

Proper utilization of instructions/statements

- People tends to use float rather than long variables.
  - NoTE: float and long int assigns 4 bytes.
- But, the impact is hefty.
- Float is slower than long or double variables.
- Why? It depends on architectures. Normally, better pipelines and acceleration hardware exists for integer operations than float operations.
- Lookup vs. calculate:
  - Lookup is faster than calculate options.
  - For instance, x=(int) sin(angle)*127)+127
  - If we know earlier, it is better to place the values in the lookup table (array) and utilize them..

Low power arduino

- Arduino UNO draws about 40mA
- Ie. It can run on a small 9V battery(150 mAh) for about 4 hours.
- Power consumption for the other boards
  - UNO (5V USB) : 47mA
  - Uno (9V power supply) : 48mA
  - Leonardo (5V USB) : 42mA
  - Due (5V USB) : 160mA
  - MiniPro (5V USB) : 22mA

Power Efficiency – Procedures

- Reducing the clock speed
- How? There exists a Prescalar library to do so.

- At setup() function, add
  - setClockPrescalar(CLOCK_PRESCALAR_256);
  - This sets to 62.5 kHz

  - CLOCK_PRESCALAR_1 → 16 MHz
  - CLOCK_PRESCALAR_2 → 8 MHz
  - CLOCK_PRESCALAR_4 → 4 MHz
  - CLOCK_PRESCALAR_8 → 2 MHz
  - And, so on

Power Efficiency - procedures

- Turning things off
  - Turns off features that you are not using to save a small amount of current.
  - Ie. Turn on/off Analog-to-Digital converter, UART and so forth.

- How?
  - Utilize avr-libc library
  - Add #include <avr/power.h>
  - power_adc_disable → disables analog to digital converter.
    - You could use them if you know that analog inputs are not utilized at all in the code.
- https://github.com/vancegroup-mirrors/avr-libc/tree/master/avr-libc

Power Efficiency - procedures

- Setting up arduino to sleep mode as and when possible.

- How?
  - Utilize a library called Narcoleptic.
  - For eg. In the blink example, we normally use delay() function

  - If the delay function is too long (ie., delay(10000)), then apply the Narcoleptic delay as follows:
  - Narcoleptic.delay(10000);

- **INSTRUCTIONS**

- Exams will be held in the same hall…
- OFFLINE
- No repetition

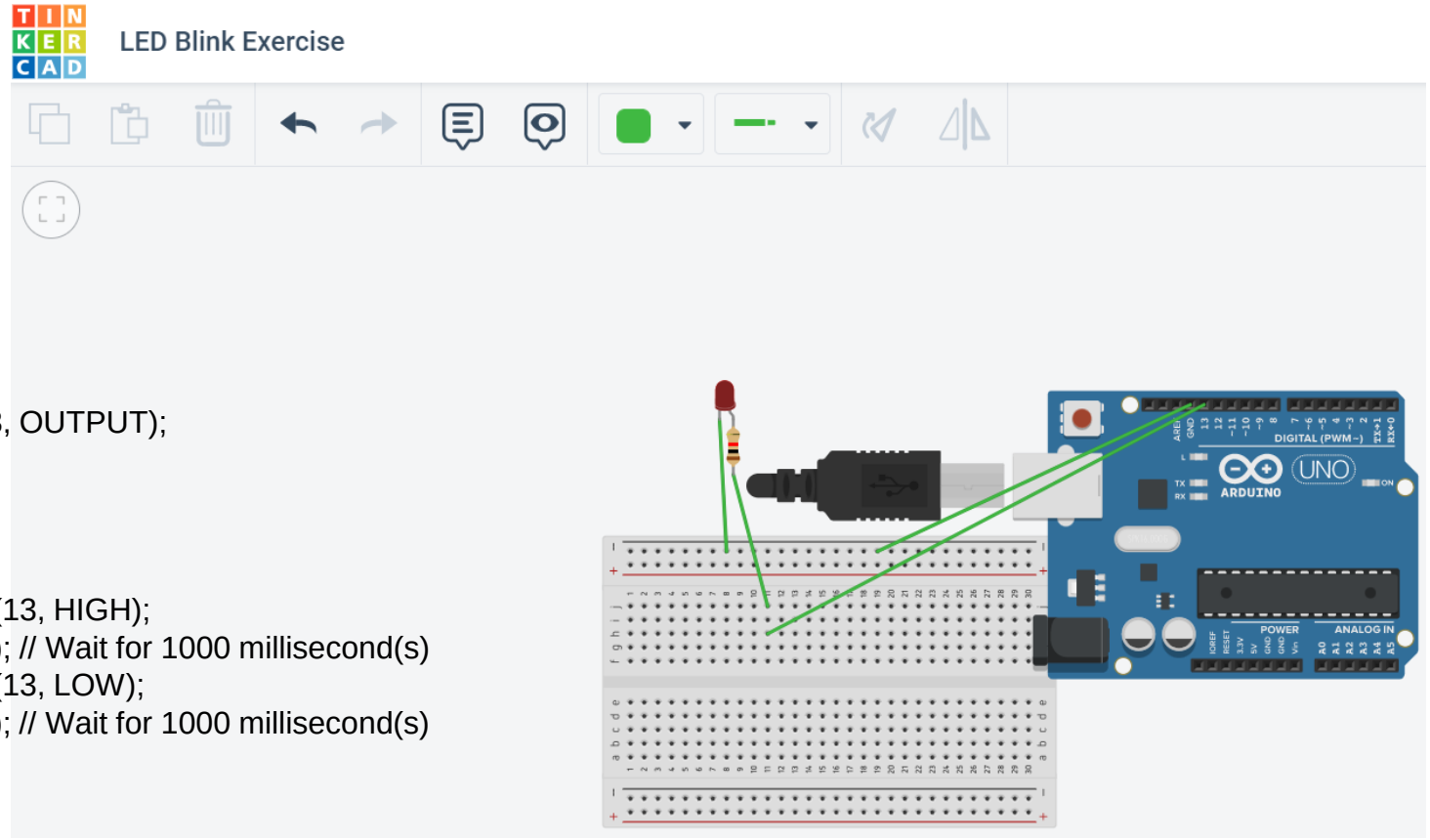
- Learn from the ppts (and, related topics)

- Part A (Multiple choice – 10 questions – 1 mark each)
- Part B (Descriptive – 8 questions – 5 marks each)

- Time: Atleast 1.5 hours will be given; Max: 2 hours.
- Names?

Tinkercad Circuits

- Examples…

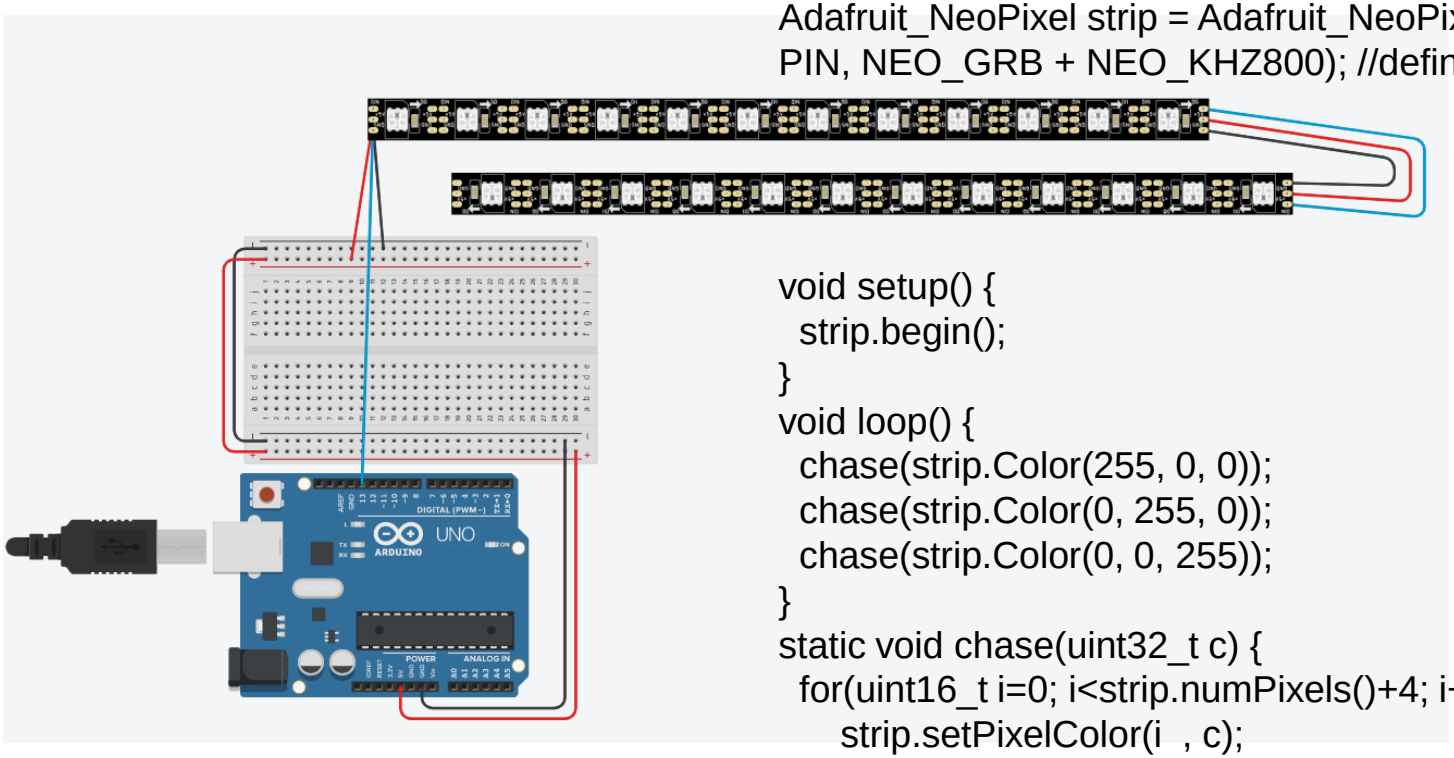



```
// Sketch
//
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

Tinkercad Circuits

- Examples

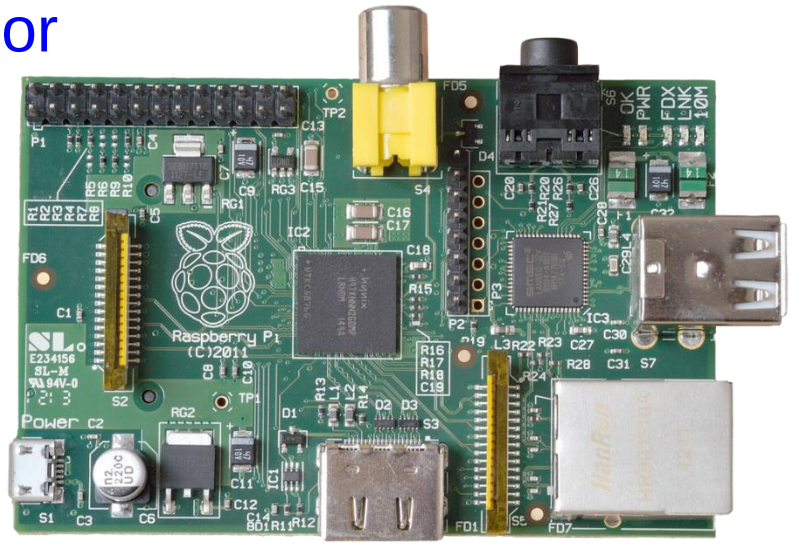



```
#include <Adafruit_NeoPixel.h>
 #define PIN 13
#define N_LEDS 24
Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS,
PIN, NEO_GRB + NEO_KHZ800); //defines bitstream

void setup() {
  strip.begin();
}
void loop() {
  chase(strip.Color(255, 0, 0));
  chase(strip.Color(0, 255, 0));
  chase(strip.Color(0, 0, 255));
}
static void chase(uint32_t c) {
  for(uint16_t i=0; i<strip.numPixels()+4; i++) {
      strip.setPixelColor(i  , c);
      strip.setPixelColor(i-4, 0);
      strip.show();
      delay(25);
  }
}
```

Raspberry Pi

- Developed in the UK by Raspbian pi foundation.
- University of Cambridge's Computer Laboratory
  - Designed for education
- A credit card sized PC.
- It is a 64-bit ARM-based processor
- Plugs into a TV or monitor
- Inexpensive – Rs.3500each
- Capability:
  - Programming
  - Electronic Projects
  - Create webservers or edge devices.
  - Play HD Videos



Introduction of Raspberry Pi

- Single board computers.
- Hardware platform – Most commonly utilized
  - Raspberry Pi Zero ($5)
  - Raspberry Pi
  - Raspberry Pi 2
  - Raspberry Pi 3 (with Wifi + Bluetooth)
  - Raspberry Pi 4 (1.5 GHz 64-bit quad core ARM Cortex-A72 processor (with WIFI, Bluetooth, Gigabit Ethernet).
- Software platform
  - Officially supported OS
    - – Noobs or Raspbian
  - 3rd party OS
    - – such as Ubuntu mate, Snappy Ubuntu core, Pinet, Windows 10 core, Risc OS
  - https://www.raspberrypi.org/downloads/

Introduction to Raspberry pi

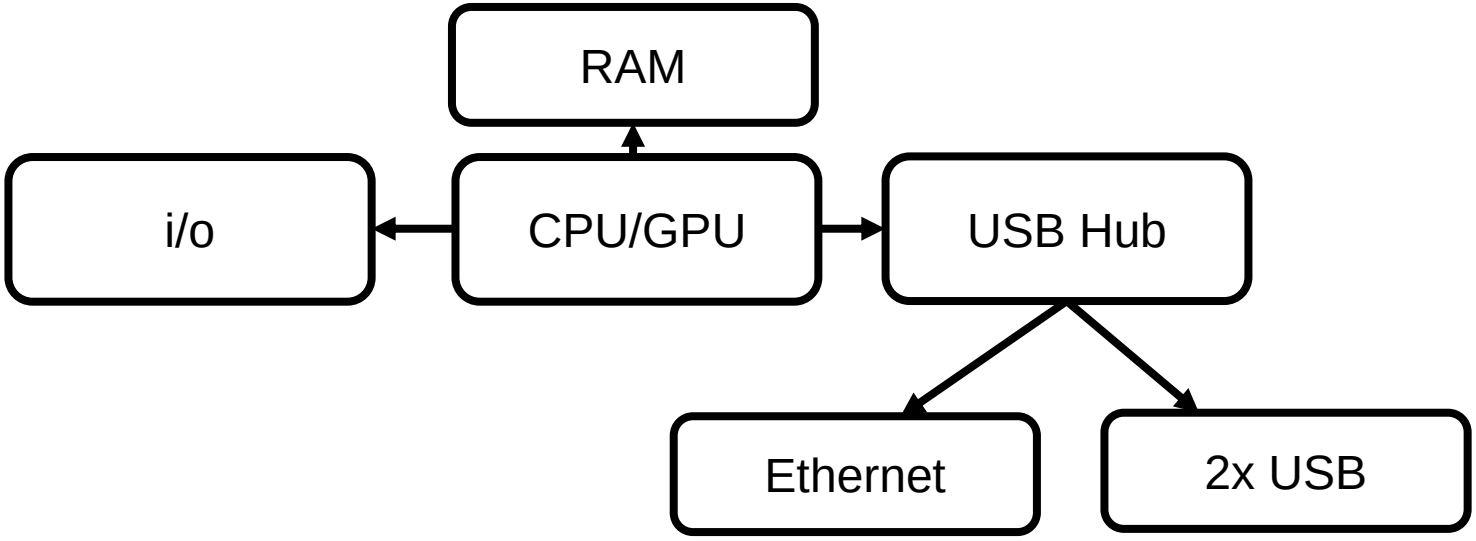
- Now, there is Raspberry pi 400
- Features of Raspberry pi 400
  - It is a complete small computer (built on a keyboard).
  - 64bit processor.
  - 4GB RAM
  - 40-Pin GPIO support.
  - WIFI/Bluetooth support.
  - Edge processing is fine here.

Raspberry pi PICO

- It is based on microcontroller chips.
- It has 264KB RAM and 2MB flash memory.
- In June 2022, Raspberry Pi Pico W was released.
  - It has inbuilt Wi-Fi and Bluetooth facilities.

Raspberry Pi Pico-Board, RP2040, 32 Bit, ARM Cortex-M0+

Basic Architecture of Raspberry Pi boards



USB – Universal Serial Bus

Programming Languages

- The Raspberry Pi Foundation recommends Python
- Any programming language, which will compile for ARMv6, can be utilized here.
- Programming languages supported in the Raspbian OS of Raspberry Pi:
  - C/C++
  - Java
  - Scratch
  - Nodejs
  - Golang
  - Ruby
  - Python 3, 2

Raspberry Pi

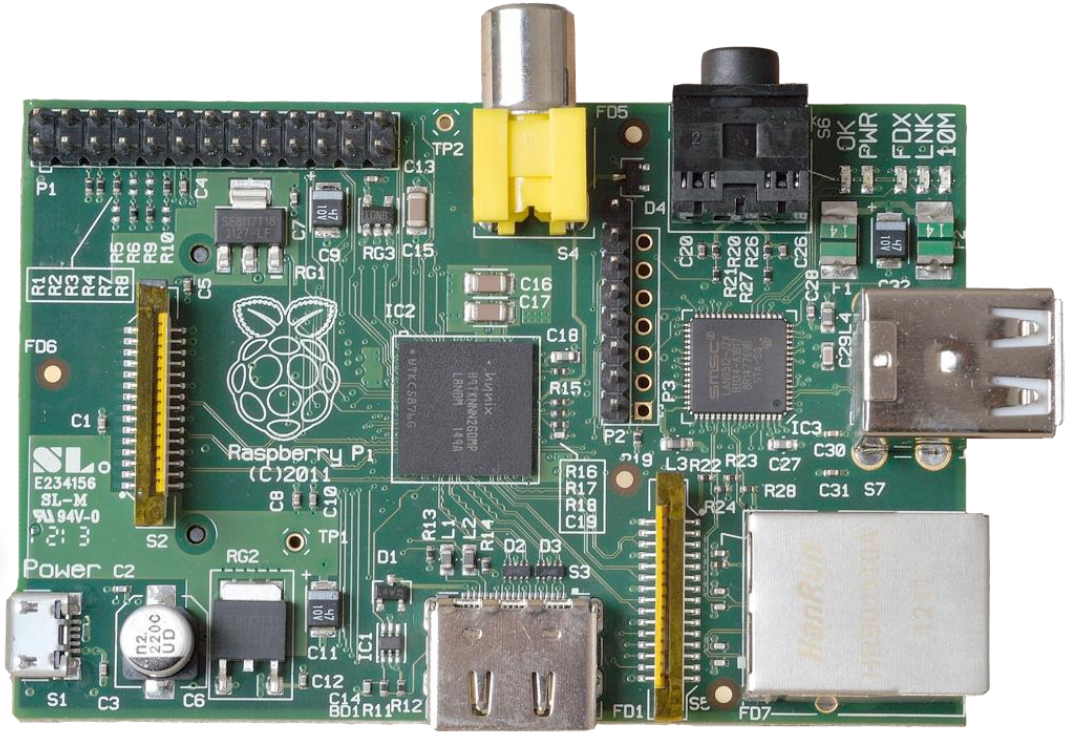
- Utilized in primary education
  - Schools use them for teaching algorithms and programming languages.
  - No need of laptops for them.

  - https://www.youtube.com/watch?v=uXUjwk2-qx4

Anatomy of Raspberry Pi

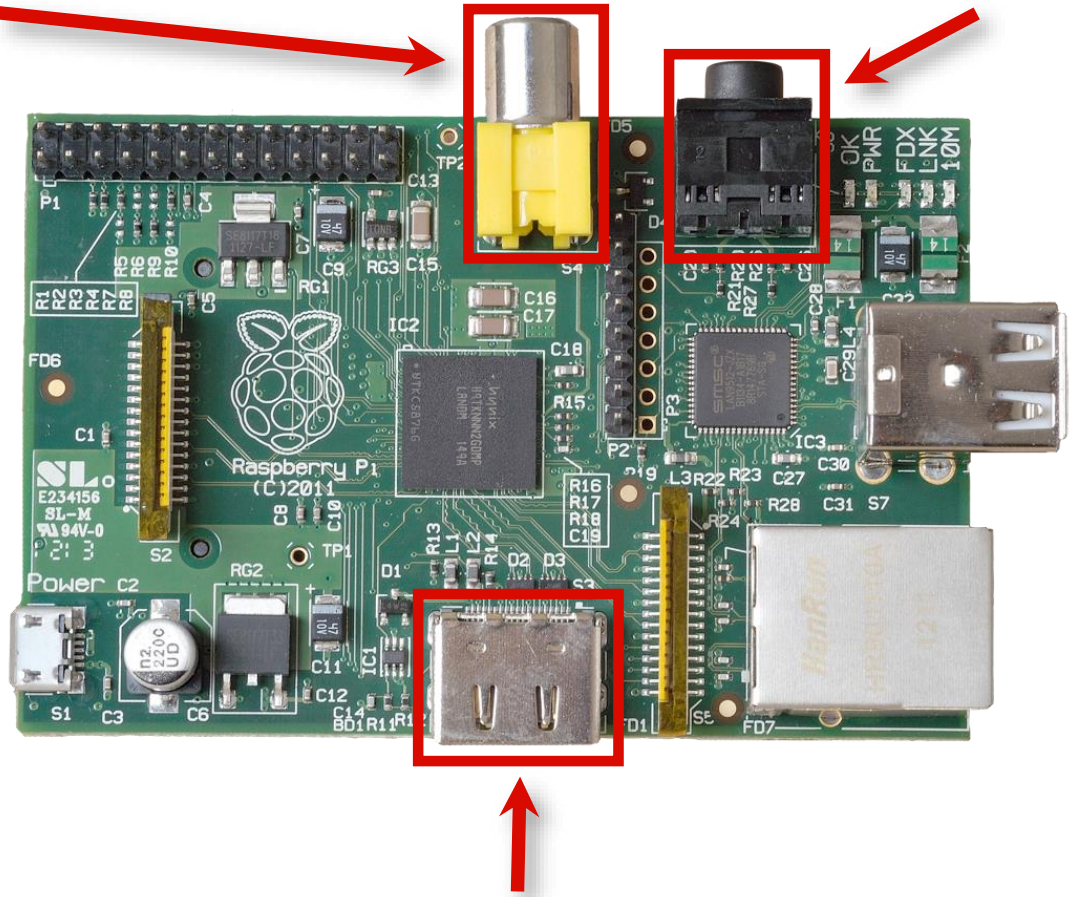
- Power supply – 2000mA

5v micro USB connector



3.5mm Audio Standard headphone socket

Audio/ Video (works with most older TVs)

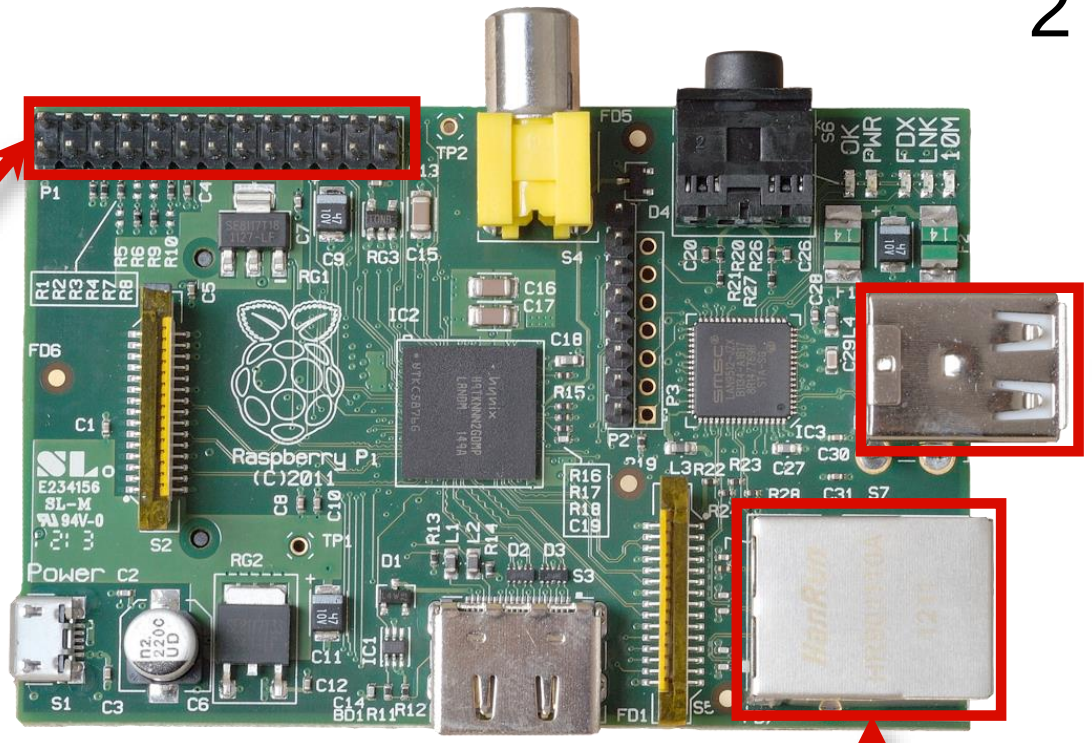


HDMI Audio & Video
(works with modern TVs and DVI monitors)

Connectivity

GPIO (General Purpose Input & Output)

It acts as both Digital and analog inputs



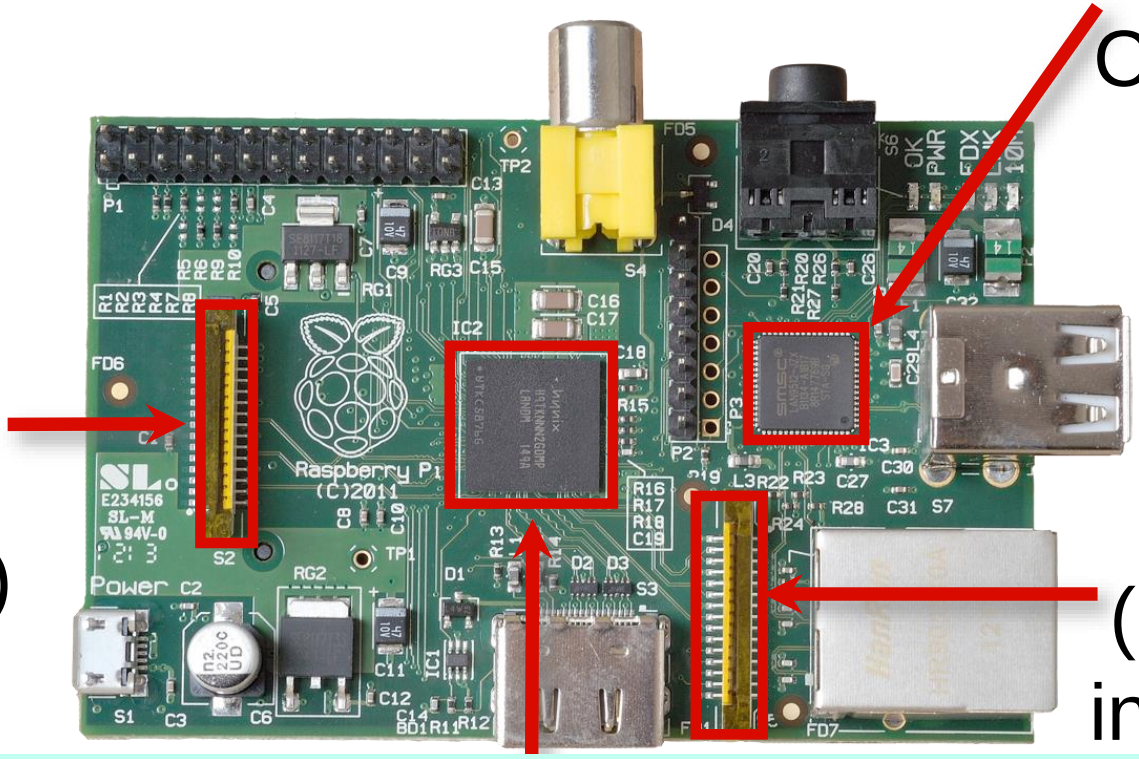
2 x USB 2.0 ports

10/100Mb Ethernet

Internals

LAN Controller

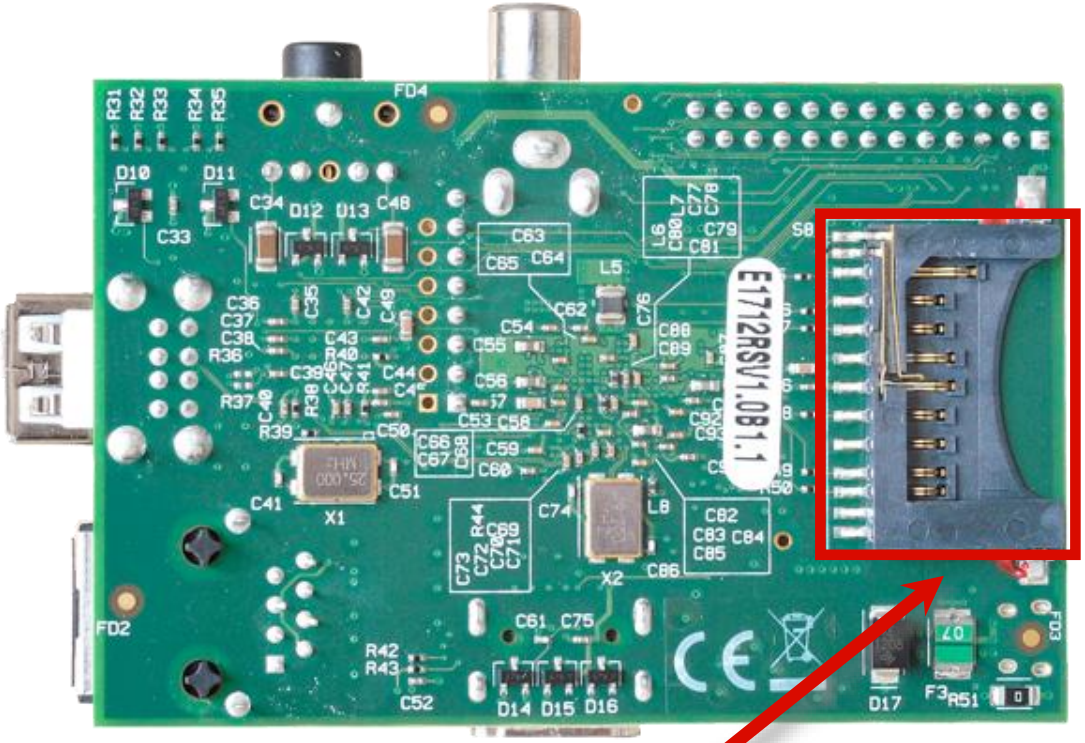
DSI (Display Serial interface)



CSI (camera interface)

SOC (System On a Chip)
Broadcom BCM2835 700Mhz – Low power ARM processor

Storage



SD Card Slot

(supports SD cards up to 32GB)

Install Raspbian OS on Raspberry Pi

- Step 1: Download raspbian OS
- Step 2: format your SD card
- Step 3: Create image using Win32DiskImager on SD card. (Also, Etcher tool is good)
- Step 4: installation is over.
- Step 5: Load SD card on raspberry pi system.

- For more details:
- https://www.youtube.com/watch?v=B5wkXu6tmb4

Basic setup for Raspberry Pi

Following components are required for setting up raspberry pi system

- HDMI cable
- Keyboard
- Monitor
- Mouse
- 5 volt power adapter
- LAN cable (connected to switch or network)
- 16, 32, or 64 GB SD card

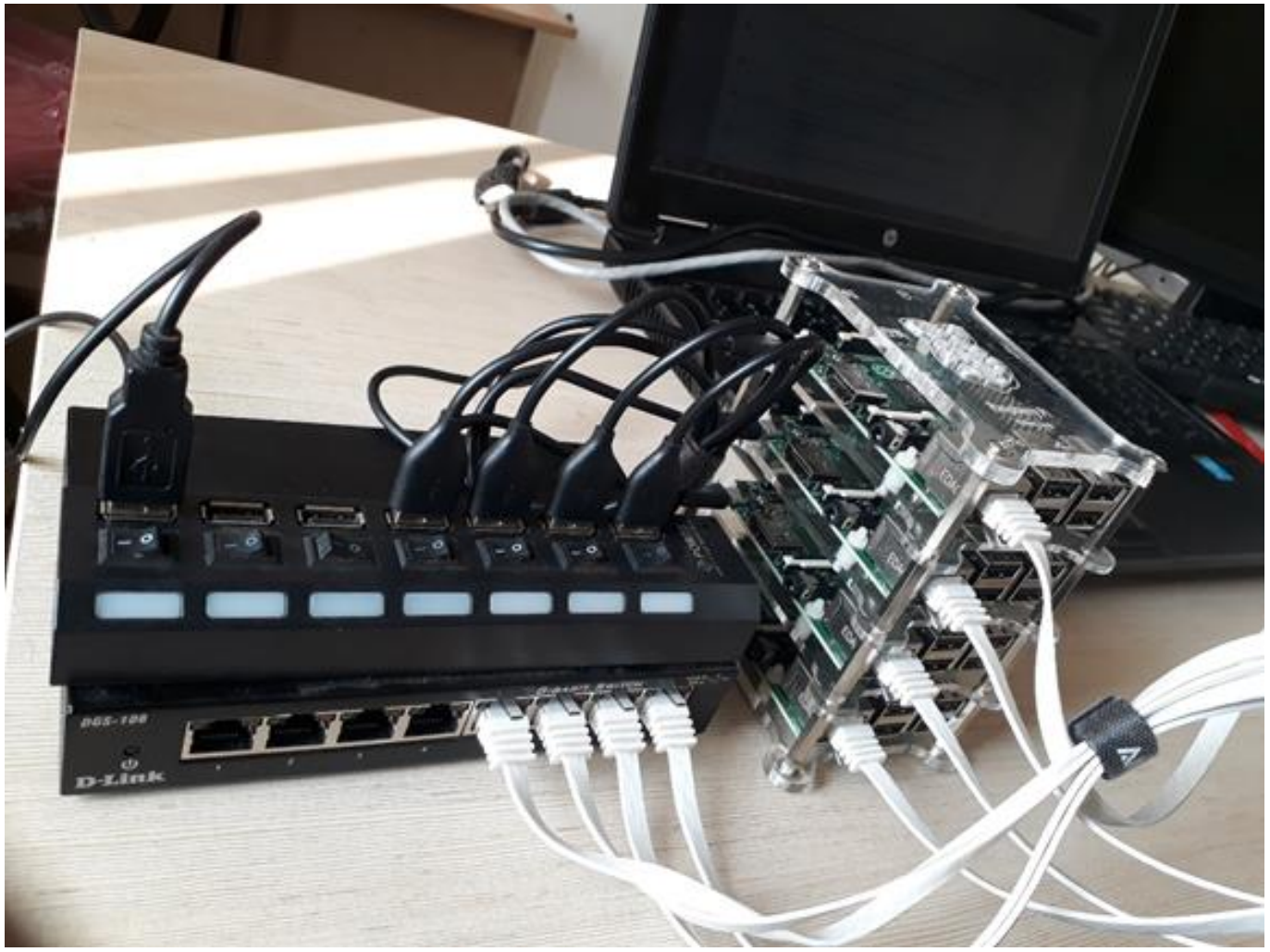
Starting Raspberry pi

Similar to Ubuntu-based system – GUI based.

Setup internet or ethernet

- Connect to wifi (starting from Raspberry pi 3)
- Or, edit /etc/network/interfaces
- Then,
  - Sudo apt-get upgrade
  - Sudo apt-get dist-upgrade (based on updated sources.list)

Raspberrypi Clusters

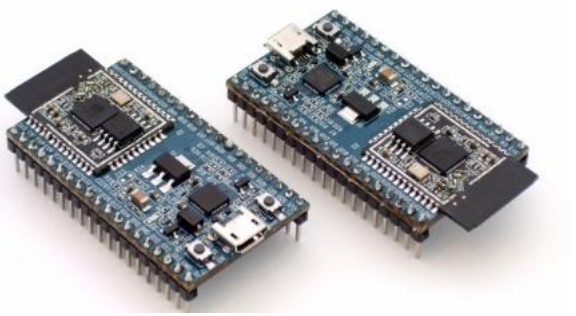


Applications of Raspberry pi

- School education
- Edge analytics
- Webserver
- Programming support
- Machine learning

ESP 32

- ESP 32 is a SoC processor developed by Espressif.
- ESP-WROOM-32 is a popular microcontroller for IoT applications.
- It is a 32-bit microcontroller.
- 2 Xtensa 32-bit CPU cores (Dual core)
- It follows the Harvard architecture
  - i.e., a separate storage and signal pathways for instructions and data.
  - And, the instructons and data have physically separated memory space units.
- ESP32 has a series of microcontrollers – NodeMCU, ESP32-Devkit, ESP32-WROOM, ESP32-WROOM-E.



ESP 32 - WROOM 32 E

- Clock frequency upto 240MHz.
- It has operating voltage of 3.3V and 80mA avg.current.
- It has integrated WiFi and Bluetooth.

- 4/8/16 MB Flash available.
- 26 GPIOs.
- 2.4 GHz WiFi (IEEE 802.11b/g/n)
- On board PCB antenna.

ESP-32 Peripherals

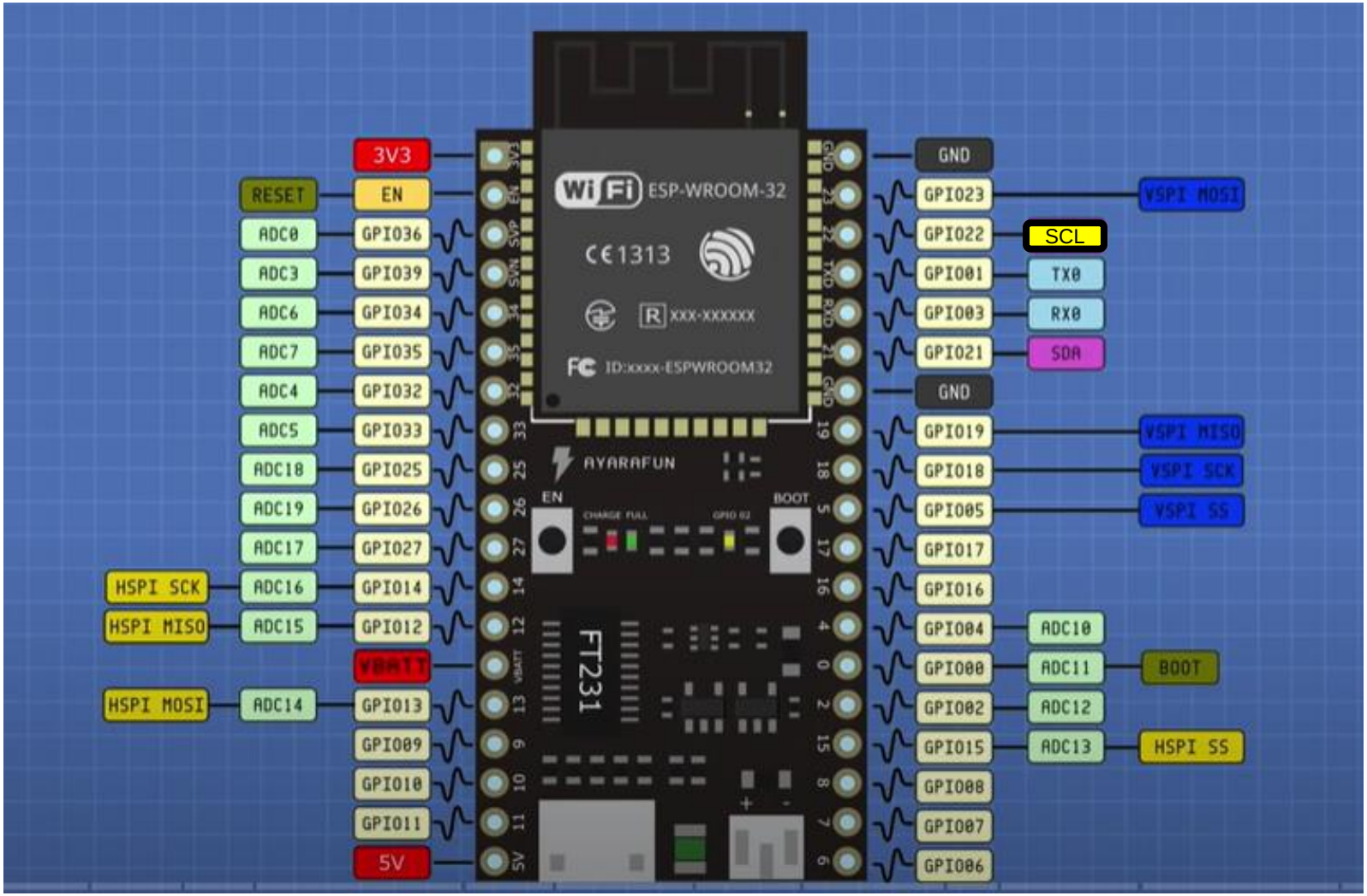
- Peripherals
  - A few serial peripheral interface (SPI) bus channels.
  - UARTs Universal Asynchronous Receiver Transmitter for serial communications.
  - SD Card | Pulse counter
  - ADC/DAC - Two 8-bit digital to analog converters; eighteen 12-bit analog to digital converters;
  - Capacitive touch sensor – 10 pins (can observe variations)
  - I2C – Inter Integrated Circuit (serial communication protocol – synchronous)

    ● I2C is half duplex communication and SPI is full duplex communication.
    ● I2C supports multi master and multi slave and SPI supports single master.
    ● I2C is a two wire protocol and SPI is a four wire protocol.
    ● I2C is slower than SPI.
    ● I2C has extra overhead start and stop bits and SPI does not have any start and stop bits.
    ● I2C has an acknowledgment bit after every byte of transfer.
    (https://prodigytechno.com/)

ESP 32 – WROOM 32 E

- The ESP32 has a dedicated UART interface with pins denoted as TX and RX.
- The TX pin of ESP32 is utilized to transmit data (outbound from ESP32)
- The RX pin of ESP32 is utilized to receive data (inbound into the ESP32).
- The default serial baud rate is 115200.

ESP 32 – Pin Diagram -- ESP-WROOM-32



ESP32-based Drones

https://www.youtube.com/watch?v=StKMekVmM1Q

ESP32-based projects

- Data logging examples
- OTA programming
- Temperature controls
- GPS trackers – vehicles, bags, people
- Neopixel LED examples

Corel Dev

- It is a development board to quickly prototype ML products (coral.ai). It costs around 130 dollars.
- It is a single board computer with inbuilt WIFi.
- It has CPU (QuadCortex), GPU (Integrated), TPUs (ASICs).
- It runs a debian-based Mendel OS.
- It is meant to accelerate AI calculations or ML algorithms. For e.g., quick inferencing of ML applications -- ML accelerators.

Corel Dev

- TPU serves as coprocessors on the board to perform over 4 trillion operations per second.
- TPUs are from Google.
- Additionally, it has GPIO, (3.3V, total 40 pins), video, and audio supports.
- Software: PyCoral API, Libcoral API (for C++), LibedgeTPU API (for C++).

Corel Dev

- One important feature of Corel Dev is System on Module.
- SOM offers the realization of system components as modules.
  - i.e., you could plug in some components, such as, CPU, memory, flash, I/O, and so forth, based on the needs/requirements
- This feature is useful for scalability purposes.
- This feature is useful for custom engineering of edge analytics or learning processes.

GPUs

- GPUs are unique processors that enable SIMD computations. (They are coprocessors).
- They are mostly utilized to perform some expensive floating point operations in parallel with multiple input data.
- E.g., GeForce GTX 1070, GeForce RTX 2060

GPUs

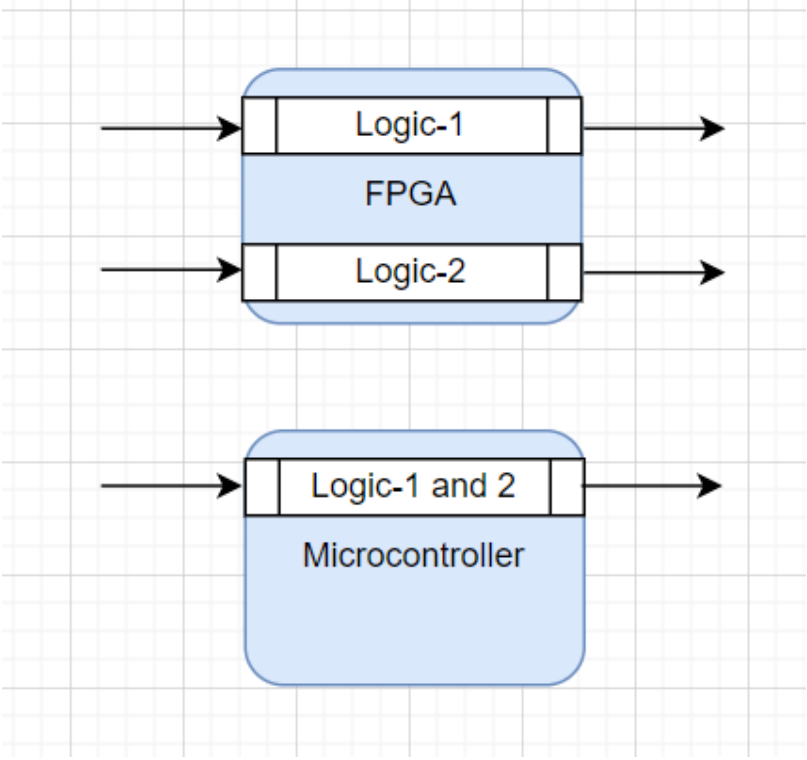
- GPUs are termed as parallel processors or co-processors.
- They are perfect processors to perform some mathematical calculations or matrix evaluations.
- 3 key advantages of GPUs for ML
  - Processors are cheaper than CPUs.
  - Big data processing is possible using GPUs.
  - Clustering GPUs enable multi-layered learning easier (which is crucial for deep learning-based algorithms).

FPGAs

- Field Programmable Gate Arrays.
- These are ICs with some programmable hardware in them.
- --i.e., they are not hard etched; they have provision to modify the hardware circuits in a programmatic fashion.
- E.g., rendering videos from cameras of IoT-enabled applications.

Compared to Microcontrollers -- FPGAs

- FPGAs can have different logics performed in one chip.
- For e.g., blink exercise could perform independently on multi input/output pins.
-



- PROGRAMMING ESP

ESP 32 – Integrated Development Env.

Programming using ESP-IDF

- ESP-IDF follows simple C-based programming.
- We need to compile applications on laptops or machines and flash it to the device.
- By using ESP-IDF framework
  - Setup the build environment (install.sh and export.sh)
  - Create a project – E.g., idf.py create-project --path <folder> <project-name>
  - Configure the project (It comes along with a lengthy python code – idf.py
    - E.g., idf.py set-target; idf.py menuconfig; and, so forth.
  - Compile and build the project
    - E.g., idf.py build

Programming using ESP-IDF -- BUILD Process

- The build process undergoes 4 phases:
  - Phase – I -- Initialization
    - Here, a few settings are configured.
    - For e.g., versions, setting targets, and so forth.
  - Phase – II -- Enumeration
    - This process builds a list of components to be processed.
    - For e.g., registering components.
    - Components are termed as the modular pieces of standalone code that are compiled using static libraries.

Programming with ESP-IDF – BUILD process

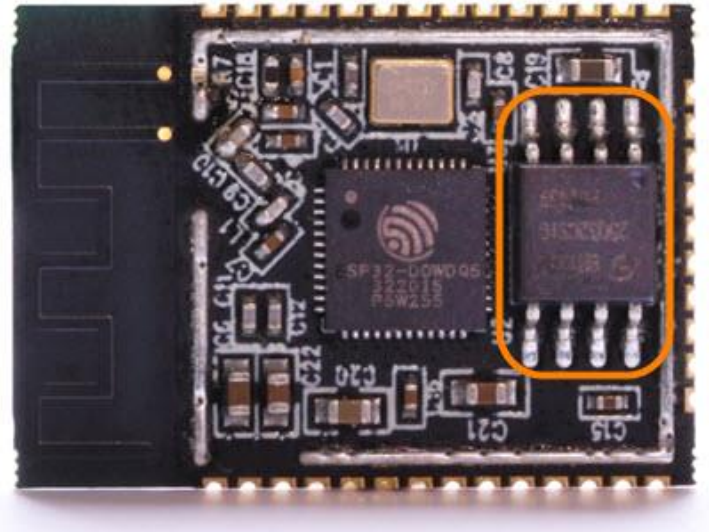
- Phase – III -- Processing
  - In this phase, the components are included with apt header files;
  - The components are placed in specific sub directories;
  - The components select required dependencies for linking them with binaries.
- Phase – IV -- Finalization
  - Here, the executables (binaries) are created and linked with libraries.
  - Also, a project metadata file is generated named as *project_description.json*

Programming using ESP-IDF – Flashing Process

- Flash the project
  - E.g., idf.py –p PORT flash
  - For windows – COM3/5 …
  - For ubuntu - /dev/ttyUSB0
- Flashing a project means
  - Erasing entire flash;
  - Flash the new app;
  - Create bootloader and partition table;

Programming using ESP-IDF – Flashing Process

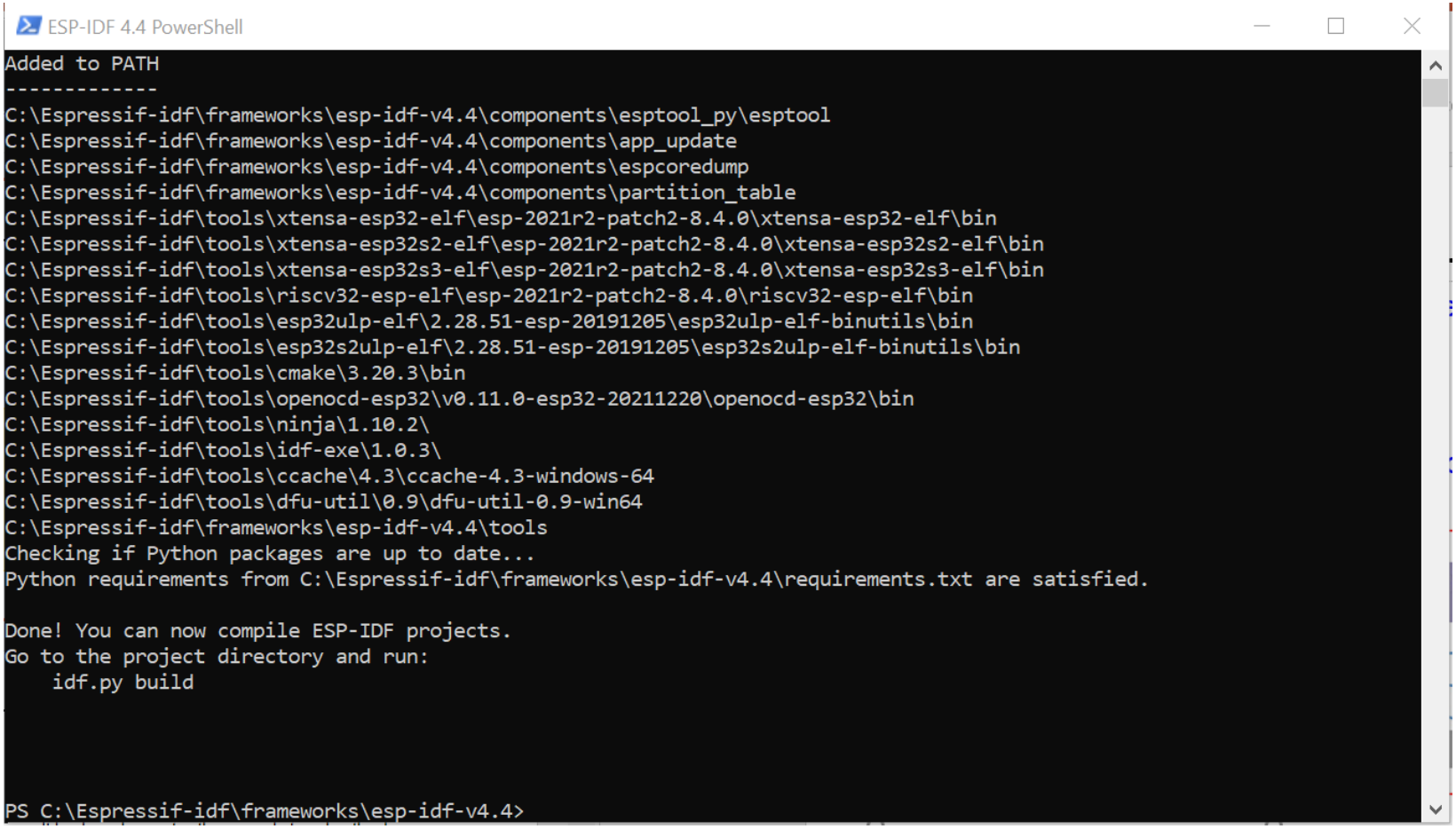
- Flash
  - Data, Config files, and programs are stored in the external flash.
  - The location of these data, config, and programs is defined/stored in the memory of ESP32 (ie., partition table).
  - Typically, 4MB flash is located outside the chip of ESP32.
  - This flash is connected to the chip using SPI bus.



Programming with ESP-IDF – Commands for Monitoring, Compiling, and Erasing flashes

- Monitor the project
  - E.g., idf.py flash monitor
- Compiling and flashing only app
  - E.g., idf.py app build
  - E.g., idf.py app-flash –p PORT
- Erasing flash
  - E.g., idf.py erase-flash –p PORT

Snapshots


ESP-IDF 4.4 PowerShell
Added to PATH

C:\Espressif-idf\frameworks\esp-idf-v4.4\components\esptool_py\esptool
C:\Espressif-idf\frameworks\esp-idf-v4.4\components\app_update
C:\Espressif-idf\frameworks\esp-idf-v4.4\components\espcoredump
C:\Espressif-idf\frameworks\esp-idf-v4.4\components\partition_table
C:\Espressif-idf\tools\xtensa-esp32-elf\esp-2021r2-patch2-8.4.0\xtensa-esp32-elf\bin
C:\Espressif-idf\tools\xtensa-esp32s2-elf\esp-2021r2-patch2-8.4.0\xtensa-esp32s2-elf\bin
C:\Espressif-idf\tools\xtensa-esp32s3-elf\esp-2021r2-patch2-8.4.0\xtensa-esp32s3-elf\bin
C:\Espressif-idf\tools\riscv32-esp-elf\esp-2021r2-patch2-8.4.0\riscv32-esp-elf\bin
C:\Espressif-idf\tools\esp32ulp-elf\2.28.51-esp-20191205\esp32ulp-elf-binutils\bin
C:\Espressif-idf\tools\esp32s2ulp-elf\2.28.51-esp-20191205\esp32s2ulp-elf-binutils\bin
C:\Espressif-idf\tools\cmake\3.20.3\bin
C:\Espressif-idf\tools\openocd-esp32\v0.11.0-esp32-20211220\openocd-esp32\bin
C:\Espressif-idf\tools\ninja\1.10.2\
C:\Espressif-idf\tools\idf-exe\1.0.3\
C:\Espressif-idf\tools\ccache\4.3\ccache-4.3-windows-64
C:\Espressif-idf\tools\dfu-util\0.9\dfu-util-0.9-win64
C:\Espressif-idf\frameworks\esp-idf-v4.4\tools
Checking if Python packages are up to date...
Python requirements from C:\Espressif-idf\frameworks\esp-idf-v4.4\requirements.txt are satisfied.
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:
idf.py build
PS C:\Espressif-idf\frameworks\esp-idf-v4.4>

Snapshots

```
PS C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink> idf.py build
Executing action: all (aliases: build)
Running ninja in directory c:\espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink\build
Executing "ninja all"...
[1/4] cmd.exe /C "cd /D C:\Espressif-idf\frameworks\esp-id...s/esp-idf-v4.4/examples/get-started/blink/build/blink.bin"
blink.bin binary size 0x29f90 bytes. Smallest app partition is 0x100000 bytes. 0xd6070 bytes (84%) free.
[2/4] Performing build step for 'bootloader'
[1/1] cmd.exe /C "cd /D C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink\build\bootloader\esp-idf\esp
tool_py && C:\Espressif-idf\python_env\idf4.4_py3.8_env\Scripts\python.exe C:/Espressif-idf/frameworks/esp-idf-v4.4/comp
onents/partition_table/check_sizes.py --offset 0x8000 bootloader 0x1000 C:/Espressif-idf/frameworks/esp-idf-v4.4/example
s/get-started/blink/build/bootloader/bootloader.bin"
Bootloader binary size 0x62c0 bytes. 0xd40 bytes (12%) free.

Project build complete. To flash, run this command:
C:\Espressif-idf\python_env\idf4.4_py3.8_env\Scripts\python.exe ..\..\..\components\esptool_py\esptool\esptool.py -p (PO
RT) -b 460800 --before default_reset --after hard_reset --chip esp32  write_flash --flash_mode dio --flash_size detect -
-flash_freq 40m 0x1000 build\bootloader\bootloader.bin 0x8000 build\partition_table\partition-table.bin 0x10000 build\bl
ink.bin
or run 'idf.py -p (PORT) flash'
PS C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink>
```

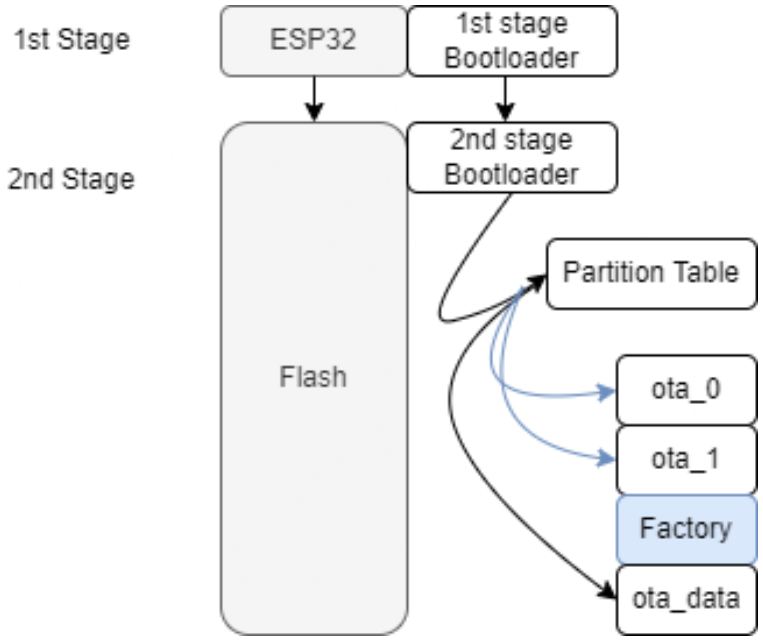
Snapshots

```
PS C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink> idf.py -p COM5 flash
Executing action: flash
Running ninja in directory c:\espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink\build
Executing "ninja flash"...
[1/5] cmd.exe /C "cd /D C:\Espressif-idf\frameworks\esp-id...s/esp-idf-v4.4/examples/get-started/blink/build/blink.bin"
blink.bin binary size 0x29f90 bytes. Smallest app partition is 0x100000 bytes. 0xd6070 bytes (84%) free.
[2/5] Performing build step for 'bootloader'
[1/1] cmd.exe /C "cd /D C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink\build\bootloader\esp-idf\esp
tool_py && C:\Espressif-idf\python_env\idf4.4_py3.8_env\Scripts\python.exe C:/Espressif-idf/frameworks/esp-idf-v4.4/comp
onents/partition_table/check_sizes.py --offset 0x8000 bootloader 0x1000 C:/Espressif-idf/frameworks/esp-idf-v4.4/example
s/get-started/blink/build/bootloader/bootloader.bin"
Bootloader binary size 0x62c0 bytes. 0xd40 bytes (12%) free.
[2/3] cmd.exe /C "cd /D C:\Espressif-idf\frameworks\esp-id.../esp-idf-v4.4/components/esptool_py/run_serial_tool.cmake"
esptool.py esp32 -p COM5 -b 460800 --before=default_reset --after=hard_reset write_flash --flash_mode dio --flash_freq 4
0m --flash_size 2MB 0x1000 bootloader/bootloader.bin 0x10000 blink.bin 0x8000 partition_table/partition-table.bin
esptool.py v3.2-dev
Serial port COM5
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x00039fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 25280 bytes to 15801...
Writing at 0x00001000... (100 %)
Wrote 25280 bytes (15801 compressed) at 0x00001000 in 0.7 seconds (effective 273.3 kbit/s)...
Hash of data verified.
Compressed 171920 bytes to 90258...
Writing at 0x00010000... (16 %)
Writing at 0x0001b30a... (33 %)
Writing at 0x00020ad2... (50 %)
Writing at 0x000264ca... (66 %)
Writing at 0x0002eac3... (83 %)
Writing at 0x00036e1a... (100 %)
Wrote 171920 bytes (90258 compressed) at 0x00010000 in 2.5 seconds (effective 546.7 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 537.0 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
Done
PS C:\Espressif-idf\frameworks\esp-idf-v4.4\examples\get-started\blink>
```

Bootloader – ESP32

- ESP32 has two-level bootloaders to execute applications.
  - First-stage bootloader
  - Second-stage bootloader
- Once powered ON, the first-stage bootloader will be executed.



Bootloader – ESP32

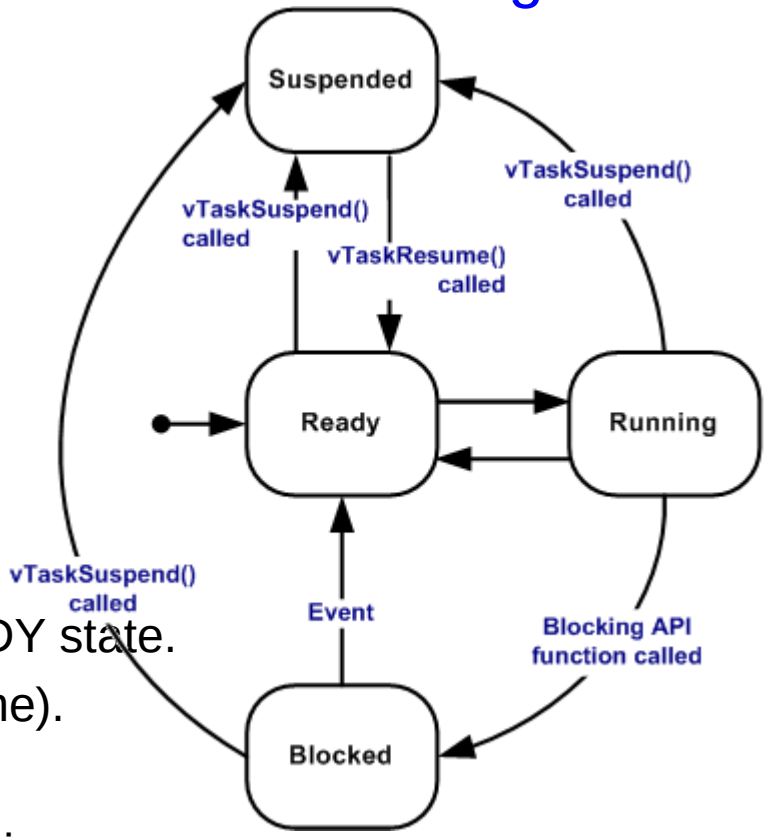
- First-stage Bootloader
  - This bootloader is responsible for pointing out the contents of flash memory.
  - This bootloader is executed at each reset of the chip.
  - It loads and executes the second-stage bootloader.
- Second-stage Bootloader
  - This is located in the flash memory.
  - It reads the partition table at specific physical address location.
  - Next, it searches for the “app”-related partition contents.
  - It selects the suitable app from the partition table based on the information provided by the **otadata** partition.
  - If no application are found, it would select the app from the **factory** partition.
  - Maximum 95 partition table entries are possible.

FreeRTOS

- We all know that tiny devices would not be comfortable with loading OS.
- But, in ESP32 microcontrollers, we could load small OS (such as FreeRTOS).
- Obviously, it is not similar to normal OS.
- But, it has the capability to handle schedulers and multi-tasking (parallelism – an important feature of OS).
- https://www.freertos.org

FreeRTOS – ESP32 states

- ESP32 sets the running tasks to one of the following states due to the FreeRTOS.
  - Running
    - Tasks are being executed.
  - Ready
    - Tasks are ready for execution.
  - Blocked
    - Tasks are blocked during execution.
    - E.g., vTaskDelay()
    - A blocked task could not enter READY state.
    - Not processed (or, no processing time).
  - Suspended
    - Tasks are suspended from its operation.
    - E.g. vTaskSuspend(), vTaskResume()
    - It could continue in the later stage.
    - Hence, the processing time could continue in the future.



FreeRTOS – Task priorities

- Tasks are assigned a priority from 0 to configMAX_PRIORITIES-1
- The settings are defined in the FreeRTOSConfig.h file.
- Task scheduling in FreeRTOS is followed in three approaches:
  - freeRTOS scheduling policy (Single Core) – Default
    - Fixed priority preemptive scheduling policy
    - i.e., equal priority for tasks with round-robin time slicing approach.
  - freeRTOS AMP scheduling policy (Asymmetric Multi-processing)
    - Each core runs its own tasks independently.
  - freeRTOS SMP scheduling policy (Symmetric Multi-processing)
    - Shared memory configuration is required.
    - Here, one instance of FreeRTOS schedules RTOS tasks across cores.

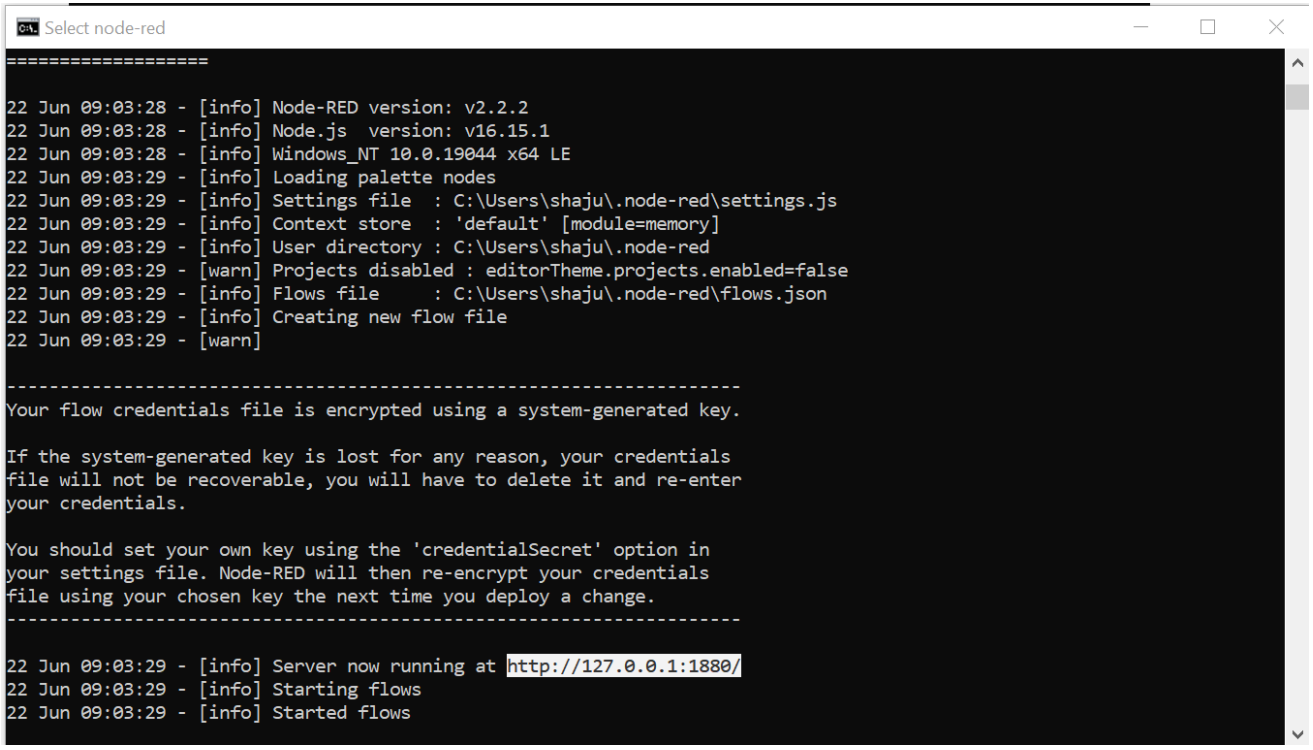
NodeRed Programming

- It is an opensource programming tool.
- It provides a visual representation of programming devices.
- It enables a browser-based flow-editor for programming devices, apps, or services.
- It is built on nodejs.
- It uses predefined blocks of code named nodes.
- Features
  - Lightweight
  - Event-driven
  - Non-blocking
  - The flows are stored using JSON formats.

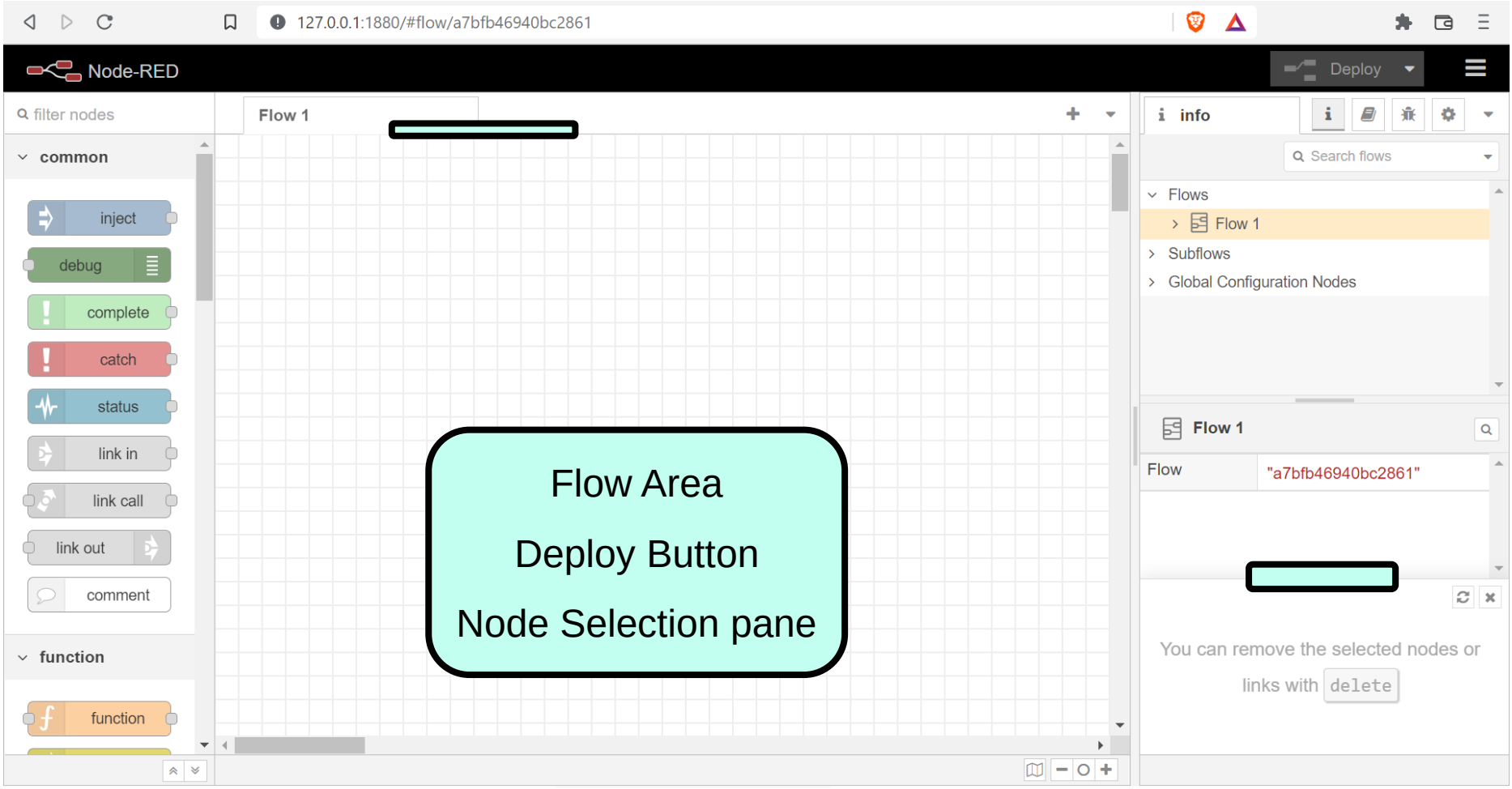
  Runs on local computer, Raspberrypi (or similar), or cloud.

NodeRed Programming

- Installation steps:
  - Install node, npm
  - Install node-red
  - Start the server once it is launched….

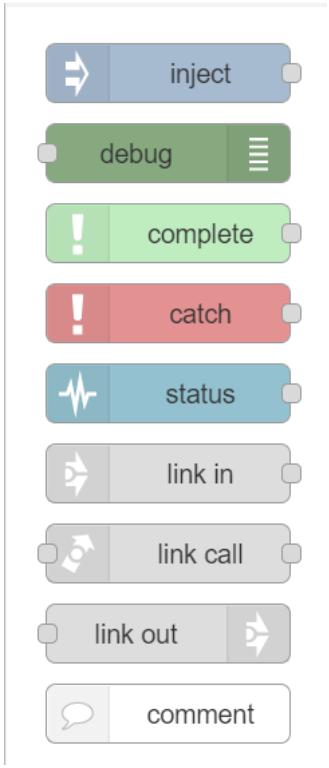


NodeRed workplace



NodeRed Programming -- Nodes

- Common Nodes – 9 Nos.
  - Inject node
  - Debug node
  - Complete node
  - Catch node
  - Status node
  - Link in
  - Link call
  - Link out
  - Comment



Basic input/output nodes

They are utilized

to specify input/output

structures…

NodeRed Programming -- Nodes

- Function Nodes
  - Perform simple functions.
- Network Nodes
  - Perform network-related tasks.
- Sequence Nodes
  - For sequencing operations.
- Parser Nodes
  - To deal with different data format
- Storage Nodes
  - To monitor and handle files.

Hello - Example

- Create two nodes
  - Inject node
  - Debug node
- Modify the payload
  - With messages “Hello IoT”
- Use debug option
  - To view messages