Espressif 8266 and 32 platforms

A short overview

- Short description of what they are
- History
- Programming-environments
- Indepth on both ESP8266 and ESP32
- Questions, comments, tomatoes

History

- Suddenly available as ‘arduino wifi module’
- Investigations reveal powerful platform
- Documentation gets translated, arduino gets ported
- ESP8266 appears everywhere (sonoff, wifi-lamps/ledstrips/etc)
- ESP32 addresses many shortcomings and improves flexibility for users of the platform greatly.

ESP-01

- 8 pin ‘module’, non FCC compliant
- AT-command-set ‘wifi’ chip
- Undocumented but obviously ‘capable of more than an atmel avr8 chip.
- Soon after the world notices esp-01, more models pop up.

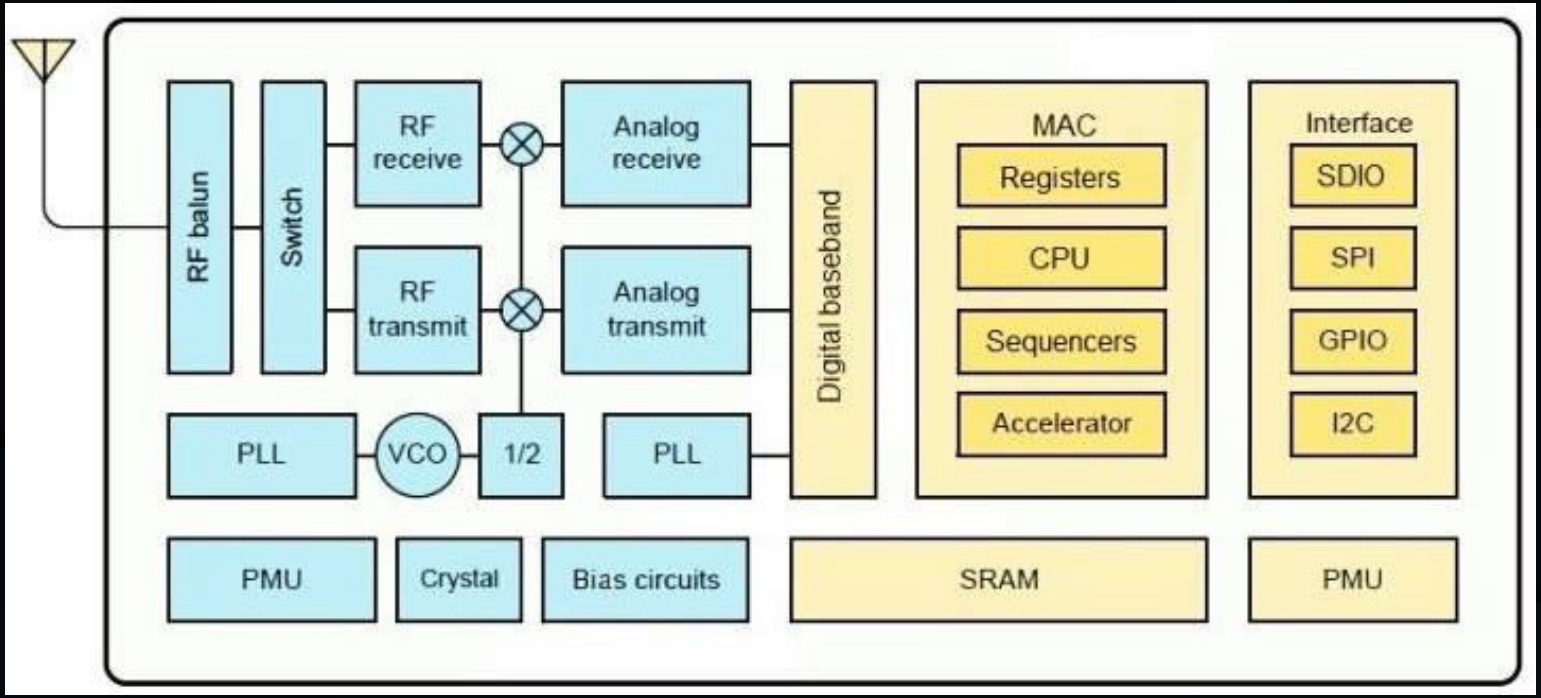
ESP8266


ESP-01
ESP-02
ESP-03
ESP-04
ESP-05
ESP-06
ESP-07
ESP-08
ESP-09
ESP-10
ESP-11
ESP-12
ESP-12E
ESP-12F
ESP-13
ESP-14
ESP-01S
ESP-07S
ESP-08S
ESP-12S

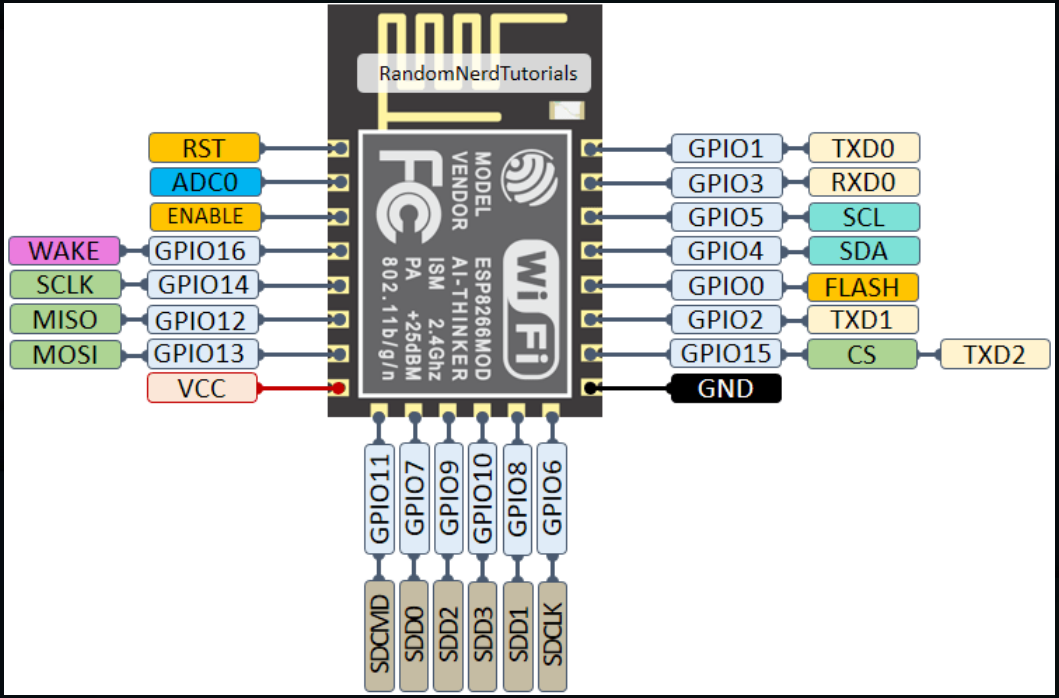
ESP8266

- Single core , up to 160Mhz, 32-bit RISC
- Can use external (flash) mem
- WiFi 802.11n radio integrated
- Sleep-mode abilities
- A host of built-in ‘peripherals’
- Available as ‘chip’, but mostly sold as ‘module’ (with or without FCC certification)
- ESP-12E/F : 16mm x 24mm

ESP8266 blocks



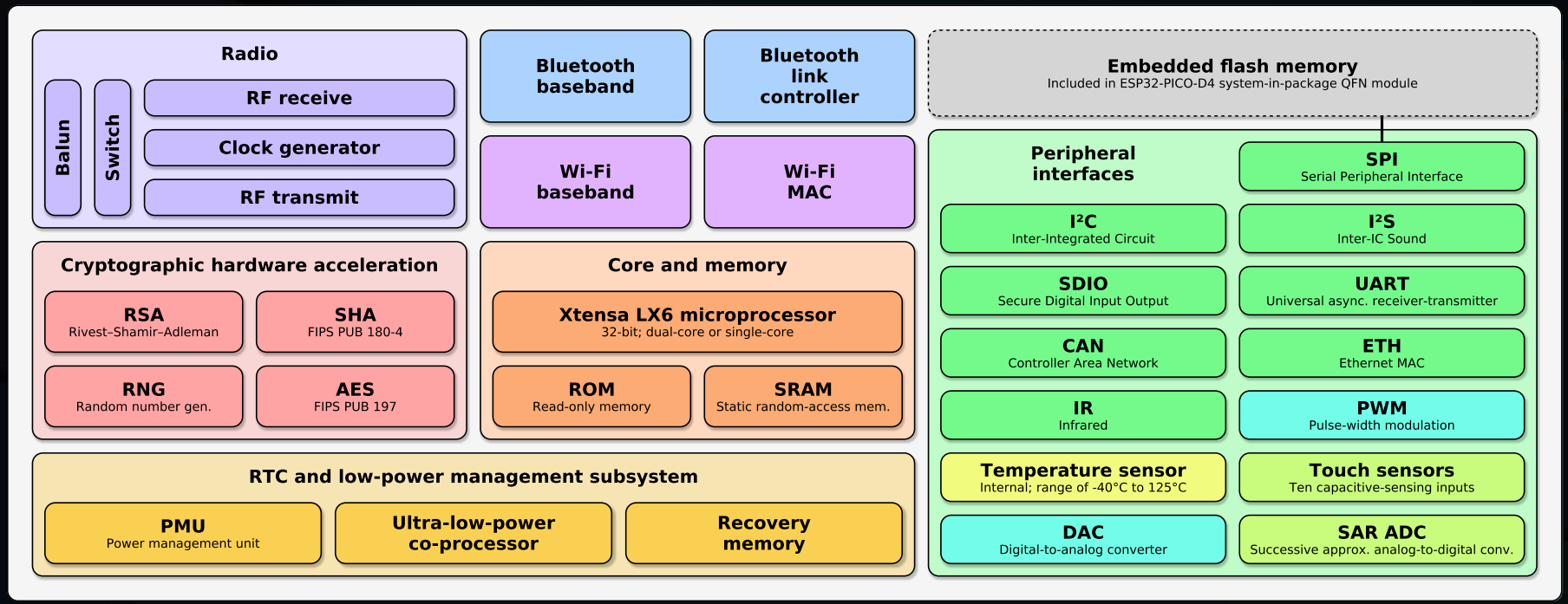
ESP8266, module



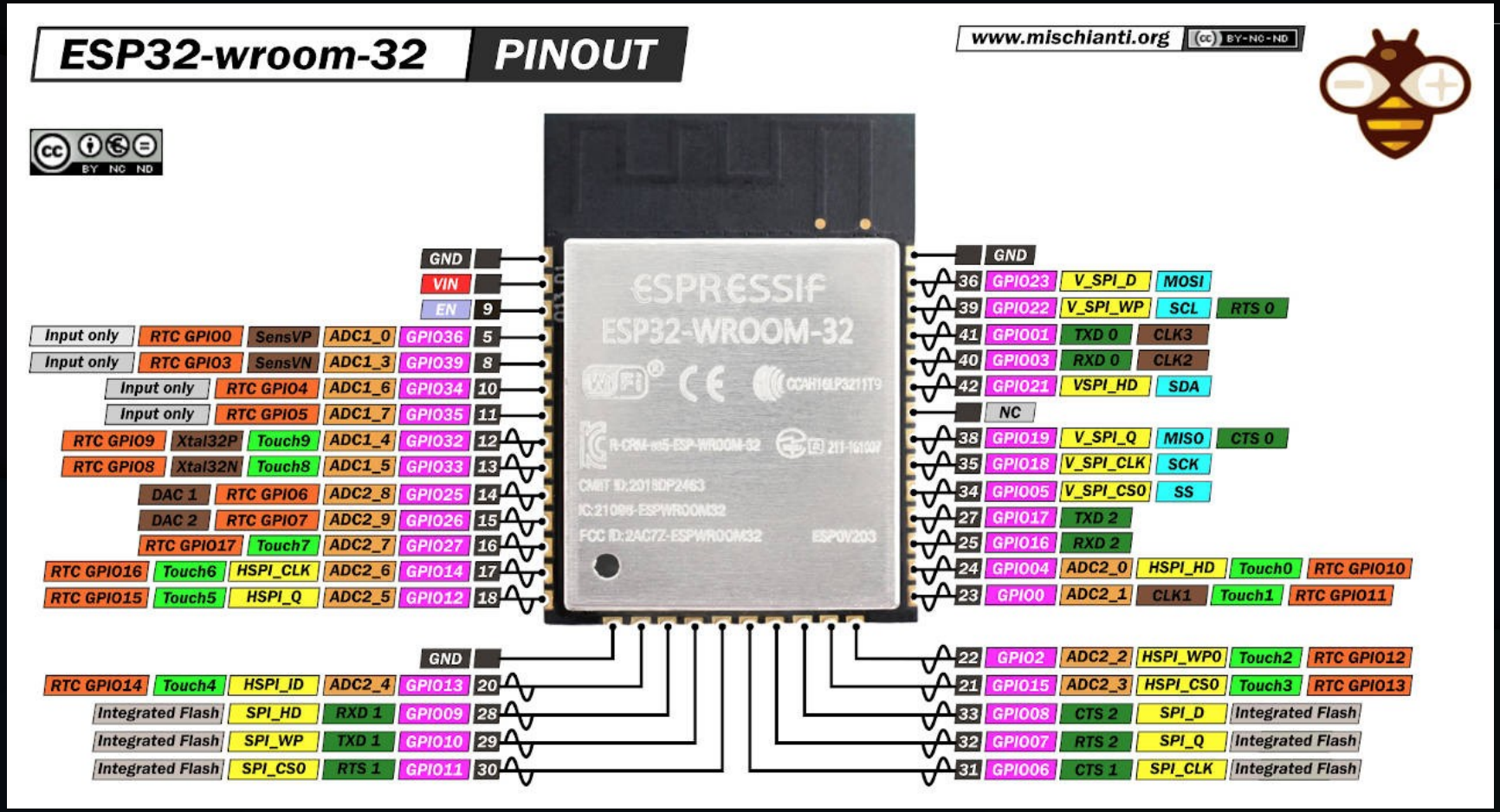
ESP32

- Dual Core, up to 240Mhz, 32-bit risc CPU
- WiFi included, bluetooth added
- ‘ulp’ core alive while it (deep)sleep
- More peripheral choice
- Less application constraints due to dual-CPU
- Greater pin-flexibility due to GPIO-Matrix/IO-Mux
- 18mm x 25mm (2mm x 1mm larger than ESP8266)

ESP32, blocks



ESP32, module



ESP’s side-by-side 1/2

	ESP8266	ESP32
		
MCU	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6 with 600 DMIPS
802.11 b/g/n Wi-Fi	HT20	HT40
Bluetooth	X	Bluetooth 4.2 and BLE
Typical Frequency	80 MHz	160 MHz
SRAM	X	✓
Flash	X	✓

ESP’s side-by-side 2/2

GPIO	17	34
Hardware /Software PWM	None / 8 channels	None / 16 channels
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	X	✓
Ethernet MAC Interface	X	✓
Touch Sensor	X	✓
Temperature Sensor	X	✓(old versions)
Hall effect sensor	X	✓
Working Temperature	-40ºC to 125ºC	-40ºC to 125ºC
Price	$ (3$ - $6)	$$ ($6 - $12)

ESP’s, how to use as Arduino

- Requires an ‘arduino-core’ package
- https://github.com/esp8266/Arduino
- https://github.com/espressif/arduino-esp32
- Needs proper ‘strapping’ of boot-pins
- Requires (usb-to)serial connection
- USB=5v, ESP=3.3v; board needs voltage-regulator
- Components/boards need to work with 3.3V logic

ESP8266 Arduino Core install

- Arduino 1.65 (1.66 has issues)
- Preferences → Additional board Manager URL:
- http://arduino.esp8266.com/stable/package_esp8266com_index.json http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Board-manager → esp8266 → install
- PlatformIO: included/automatic

ESP32 Arduino Core Install

- Latest Arduino IDE
- Preferences → Additional Board Manager URLs:
- https://dl.espressif.com/dl/package_esp32_index.json
- http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Board-manager → ESP32 → Install
- PlatformIO: included/automatic

ESP’s not mentioned (GOOD) things

- GPIO-Matrix/IO-MUX, zomg
- Interrupt-sources, zomg
- Former slide has a lie: ESP32 has HW-PWM (16 channels)
- Sleep-modes (!) on ESP32 can go down to 10µA consumption with RTC and TOUCH working(!)

ESP’s not mentioned (BAD) things

- Some pins are more special then others (bootstrap pins need certain state at boot)
- 3.3Volt power!
- Wifi + other functions can collide esp. on esp8266 (yield())
- Many boards available, no ‘default’ footprint.

Peripheral zoo, GPIO 17* vs 34*

- ‘digital’ pin
- Either ‘Input’ or ‘Output’ (pinMode(pin,mode))
- digitalRead/digitalWrite
- ESP8266 requires careful planning, ESP32 allows flexible remapping
- 18 pins on esp32 are ‘rtc_gpio’, work in deep sleep
- * Some pins ‘useless’: require specific state at boot

GPIO code

```
void setup() {
 // initialize digital pin LED_BUILTIN as an output.
 pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Peripheral zoo, PWM, 0vs16

- LEDC pwm for led/sound/power
- MCPWM for motor-control, has ‘input’ logic
- Software PWM costs CPU-time, can be complex
- ESP32 has groups of 8 channels of PWM, connectable to any* GPIO

LEDC code

```
void setup(){
  // configure LED PWM functionalitites
  // channel, basefreq, resolution
  ledcSetup(0, 5000, 8);

  // attach the channel to the GPIO to be controlled
  LedcAttachPin(0, 16);
}
void loop() {
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }
}
```

Peripheral zoo, ADC 1vs18*

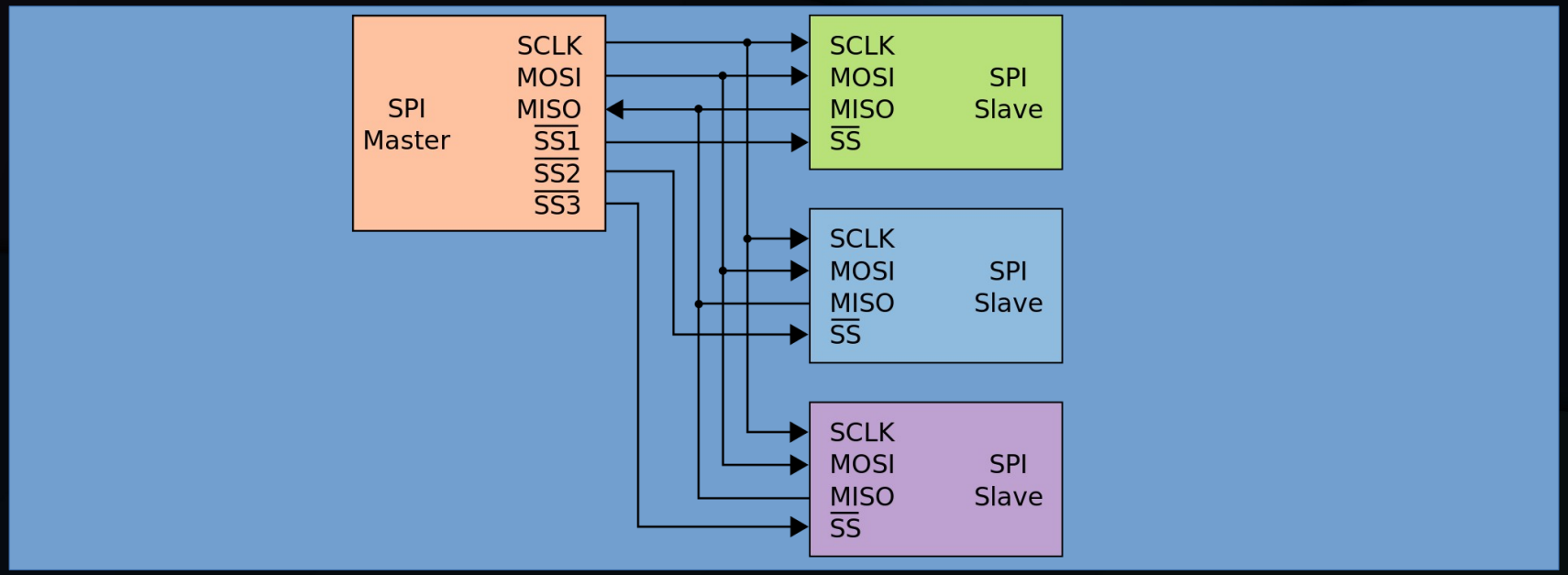
- ESP8266 has 10bit, ESP32 has 12bit
- ESP32 has two ‘groups’. Second group blocks with wifi; like with esp8266 ADC
- Some pins not ‘useful’ due to boot-strap function

ADC code

```
/*********
  Rui Santos
  Complete project details at https://randomnerdtutorials.com
*********/

const int analogInPin = A0;  // ESP8266 Analog Pin ADC0 = A0

int sensorValue = 0;  // value read from the pot

void setup() {
  // initialize serial communication at 115200
  Serial.begin(115200);
}

void loop() {
  // read the analog in value
  sensorValue = analogRead(analogInPin);

  // print the readings in the Serial Monitor
  Serial.print("sensor = ");
  Serial.print(sensorValue);

  delay(1000);
}
```

Peripheral zoo, SPI, 2 vs 3 (4?)

- SPI = Serial Peripheral Interface



Peripheral zoo, SPI

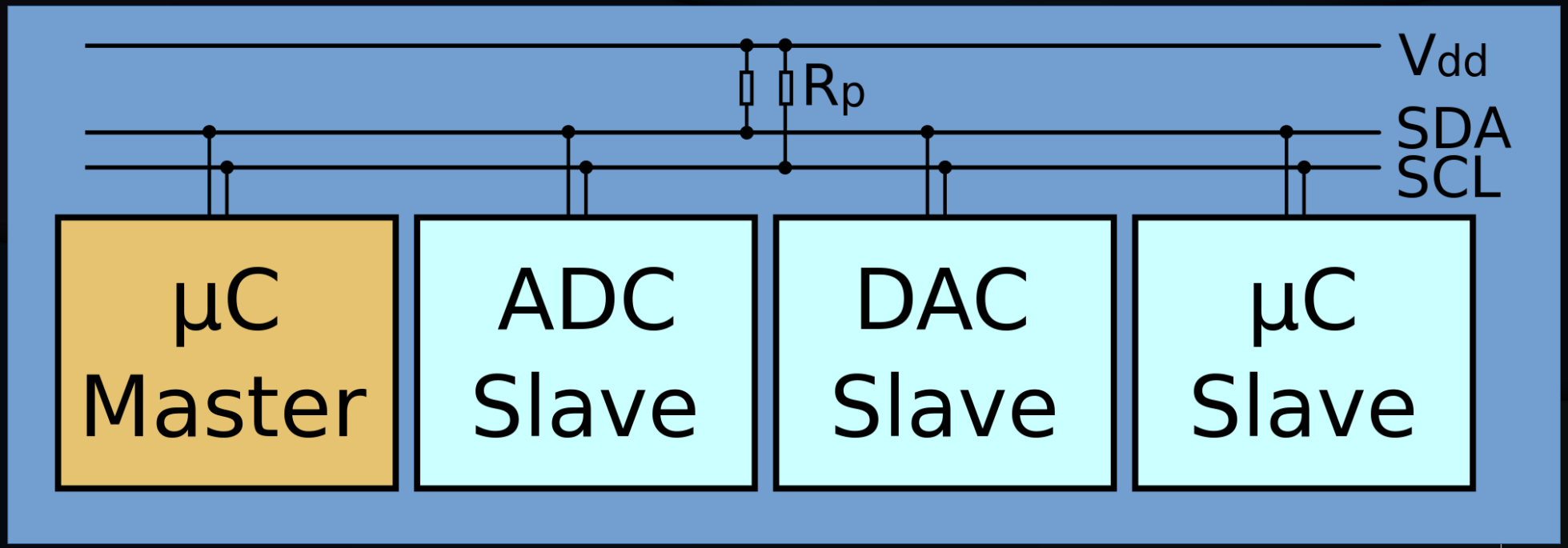
- Dot-matrix LCD’s
- SD/MMC cards (require pullup-resistors)
- Flash-chips
- LoRa radio-chipsets (RFM95, SX127x)
- High-speed serial IO (shift-registers) IO-expanding, led-driving (direct-mode)

SPI code

```
#include "SPI.h"
char buff[]="Hello World\n";

void setup() {
 SPI.begin();
}

void loop() {
 for(int i=0; i<sizeof buff; i++)
 {
  SPI.transfer(buff[i]);
 }
 delay(1000);
}
```

Peripheral Zoo, I2C, 1 vs 2

- I2C = Inter-Integrated-Circuit (I-Squared-C)



Peripheral zoo, I2C

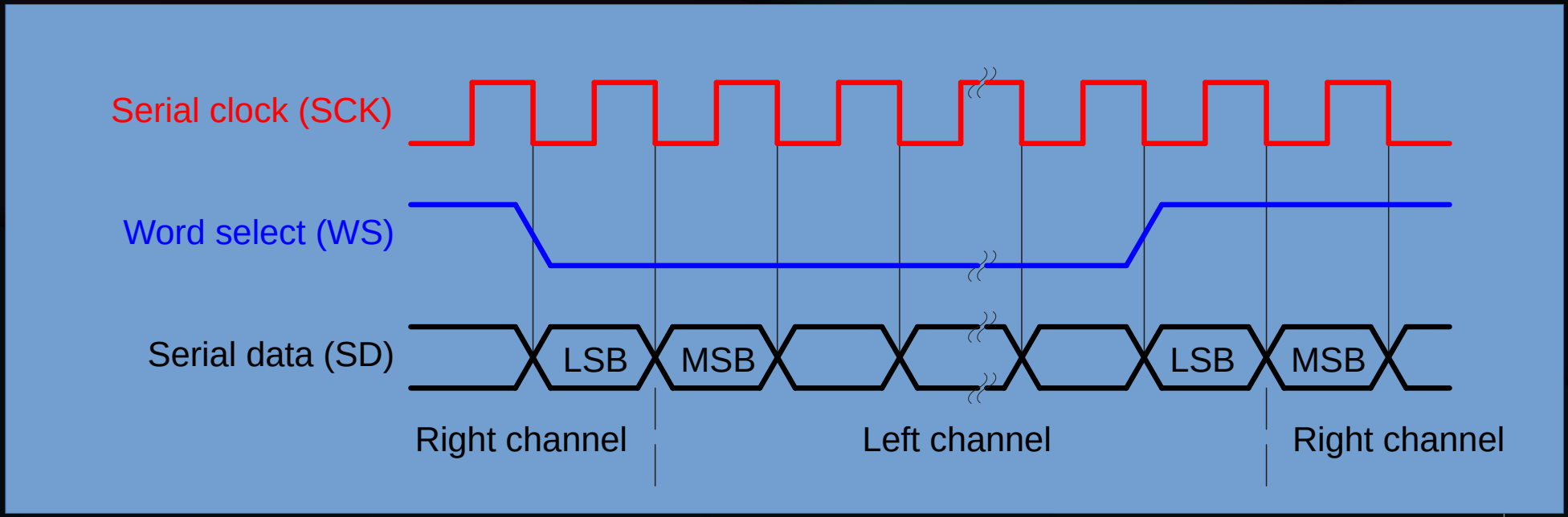
- IO-expanders (character-LCD’s, relay-boards)
- Eeproms
- Real-time-clock chips
- Temperature-chips
- PWM-chips (led-drivers)

I2C Code

```
#include <Wire.h>
void setup() {
  Wire.begin()
  delay(100)
  Wire.beginTransmission(address)
  Wire.write(register_address)
  Wire.write(data)
  Wire.endTransmission()
}
void loop() {
  // do some transfer to chip to get it to prepare data
  Wire.requestFrom(address,amount)
  if (Wire.available()<=2) {
    for (i=0,i<amount,i++) {
     bla=Wire.read()
    }
}
```

Peripheral Zoo, I2S, 1+1 vs 2

- I2S , Inter-IC Sound, I-Squared-S



Peripheral Zoo, I2S

- Meant for digital audio (S/PDIF)
- Useful for high-speed serial I/O
- Often used for ‘digital’ led-strips

Peripheral Zoo, UART, 2 vs 3

- Universal Asynchronous Receiver Transmitter
- Rx/Tx
- CTS/DTS
- IrDA (over infrared)
- RS232/RS485 : different types of signals

Peripheral Zoo, UART

- Used for programming and monitoring
- Useful for communication between modules
- Useful for long-distance communication
- Max232 for ‘rs232’ levels
- RS485 also available

UART Code

```
void setup() {
  Serial.begin(9600)

}
void loop() {
  Serial.print("This is a stupid example")
  delay(100)
}
```

Peripheral zoo, DAC 0 vs 2

- ESP32 , GPIO 25 + 26
- 8-bit.
- Can interoperate with i2s-peripheral

Peripheral Zoo, Touch, 0 vs 10

- Capacitive pads, current-less
- Can trigger interrupt even in sleep-mode
- Can be single-channel ‘button’
- Multiple-channels can be combined into ‘slider’ or ‘ring’ with careful design

Touchpin Code

```
// ESP32 Touch Test – Randomnerdtutorials.com
// Just test touch pin - Touch0 is T0 which is on GPIO 4.

void setup() {
  Serial.begin(115200);
  delay(1000); // give me time to bring up serial monitor
  Serial.println("ESP32 Touch Test");
}

void loop() {
  Serial.println(touchRead(4));  // get value of Touch 0 pin = GPIO 4
  delay(1000);
}
```

Peripheral Zoo, PCNT, 0vs8

- ESP32 has 8 pulse-counters of 16bits
- Can connect to any gpio*
- Can count up, down, do interrupts

Peripheral Zoo, RMT, 0 vs 8

- Remote Receiver or Transmitter
- Each channel can either be Rx or Tx
- Takes care of modulation (38khz, 48khz, etc)
- Can be used for other purposes (sound, leds)

Peripheral Zoo, CAN 0 vs 1

- Used in automotive industry
- Requires some external circuitry

Peripheral Zoo, Ethernet 0 vs 1

- 100Mbit ‘MAC’ controller
- Requires external ‘PHY’ chip + magnetics
- Communicates via RMII (Reduced Media Independant Interface)
- GPIO 0, 19, 21, 22, 25, 26, 27 are fixed
- 3 more GPIO required, can be chosen by user
- Layout for working ethernet-setup is tricky/picky

Peripheral Zoo, WiFi, 1vs1

- HT20 vs HT40 (wider band, higher speed)
- WiFI, network, TCP/IP CPU intensive
- Modules available with connectors
- Uses same radio + antenn as Bluetooth
- EspNOW mesh-protocol available

ESP Wifi Code

```
#include <ESP8266WiFi.h>        // Include the Wi-Fi library for ESP8266
// #include <WiFi.h>  // Include the Wi-Fi library for ESP32

const char* ssid     = "SSID";         // The SSID (name) of the Wi-Fi network you want to connect to
const char* password = "PASSWORD";     // The password of the Wi-Fi network

void setup() {
  Serial.begin(115200);         // Start the Serial communication to send messages to the computer
  delay(10);
  Serial.println('\n');

  WiFi.begin(ssid, password);             // Connect to the network
  Serial.print("Connecting to ");
  Serial.print(ssid); Serial.println(" ...");

  int i = 0;
  while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
    delay(1000);
    Serial.print(++i); Serial.print(' ');
  }

  Serial.println('\n');
  Serial.println("Connection established!");
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());         // Send the IP address of the ESP to the computer
}

void loop() { }
```

Peripheral Zoo, Bluetooth, 0 vs 1

- Works but limited examples/docs
- BLE ‘sensing’ works
- Audio works
- Input-device works
- Meshing protocol available

Peripheral Zoo, SecureElement 0vs1

- Accelerates crypto : ECDSA for TLS

Cool stuff available esp-idf (esp32)

- IPv6 support
- ‘Thread’ IPv6-based IoT mesh-networking
- ESP-Adf audio framework (i2s, bluetooth, mp3, etc)
- ESP-CSI Use wifi as ‘sensor’
- ESP-DSP Digital Signal Processing
- ESP-Wifi-Mesh Mesh-networking
- ESP-WHO Face Detecting with camera (!)
- ‘EasyConnect’ for WiFi (qr-code data)

ESP, typical application

- Connect to wifi or set up AP with webportal
- Read some sensors
- Drive some outputs
- Listen to commands from network
- Send status to network

ESP, typical application

- khoih-prog/ESP_WiFiManager
- Tons-of-libraries for input
- Tons-of-libraries for output
- me-no-dev/ESPAsyncWebServer
- knolleary/pubsubclient
- bblanchon/ArduinoJson

ESP typical application, caveats

- Define some kind of structured API
- Much of code is ‘boilerplate’, sometimes only small changes between projects
- Extending requires rewrites of API handling
- Interaction with ‘other stuff’ requires custom ‘glue’

Alternative to building your own

- Existing projects
  - Tasmota
  - Espeasy
  - Espurna
- PRO’s: flash once, config on device

Alternative to building your own

- Cons:
  - No ‘code’ to version/backup
  - Multiple versions for different uC’s
  - Not much space left on device
  - Often not very ‘structured’ internally

ESPhome.io

- Based on PlatformIO (cll) framework.
- Designed for HomeAssistant, but useful without
- “Code” is a .yaml file
- Project-creation ‘wizard’
- ‘flashing’ via usb or OTA only what you **need**
- Structured API and ‘webserver’ module
- Esphome ‘dashboard’ for easy maintenance

ESPHome.io

- Provides abstract ‘components’
- A ‘component’ can act upon others
- Tons of components and component-types
- Allows integration of custom code
- Provides ‘filters’ for data-transformation
- Provides ‘automation’ for on-device control

ESPHome.io - Demo

- esphome wizard tutorial.yaml
- esphome run tutorial.yaml
- compile, upload, logs
- esphome dashboard localhost

ESPHome.io, i2c-expander → switch

```
# Example configuration entry
pcf8574:
  - id: 'pcf8574_hub'
    address: 0x21
    pcf8575: false

# Individual outputs
switch:
  - platform: gpio
    name: "PCF8574 Pin #0"
    pin:
      pcf8574: pcf8574_hub
      # Use pin number 0
      number: 0
      # One of INPUT or OUTPUT
      mode: OUTPUT
      inverted: false
```

ESPHome.io IR-receiver → switch

```
# Example configuration entry
remote_receiver:
  pin: GPIO32
  dump: all

binary_sensor:
  - platform: remote_receiver
    name: "Panasonic Remote Input"
    panasonic:
      address: 0x4004
      command: 0x100BCBD
```

ESPHome.io , HX711 digital scale

```
sensor:
  - platform: hx711
    name: "HX711 Value"
    dout_pin: D2
    clk_pin: D1
    gain: 128
    update_interval: 1s
    filters:
        - calibrate_linear:
            - 49000 -> 0.00
            - 117500 -> 2.776
    unit_of_measurement: kg
```

ESPHome.io – Things to note

- logger:

  baud_rate: 0 #disable UART logging

  api:

  reboot_timeout: 0 # disables failsafe bootloop

- Use ‘-’ , not ‘_’ in names for device/yaml

ESP8266 , ESP32 Conclusions

- Versatile platform
- Cheap
- Some ‘polished’ boards are better than Arduino hardware (m5stack)
- Hits sweet spot between 8bit µC and ‘full blown’ Single-board-computer