Lecture 15

ESP32 and Arduino

Agenda

- Lab 4 Mobile Bases
- ESP32
- Arduino
- PicoKit ESP32 Hardware
- DEMO of TA MOBILE BASE

01

Lab 4: Mobile Bases

Mobile Bases

- Skid steer
  - Easy to make, hard to steer

- Ackerman steering
  - Hard to make, constrained motion

- Holonomic
  - Expensive, high mobility

Mobile Bases

- DIfferential Drive
  - Castor or skid in front
  - Easy to make
  - Lower cost
  - Recommended for simplicity
  - Compared to skid steer:
    - Similar control for steering
    - Less motor power required to turn

02

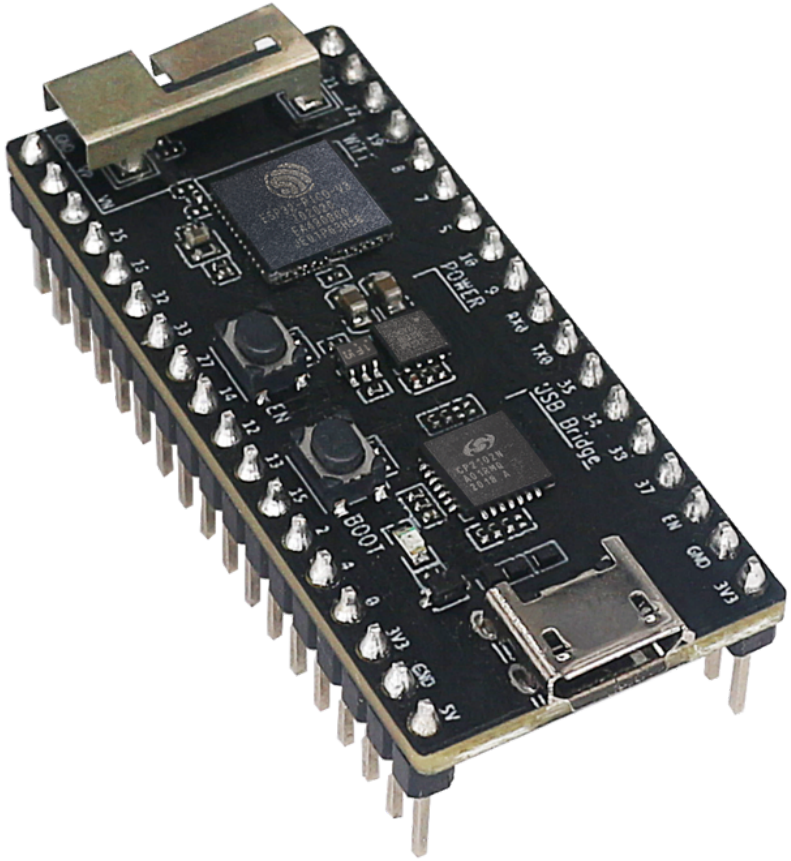
ESP32 Overview

Typical important functions on a microcontroller

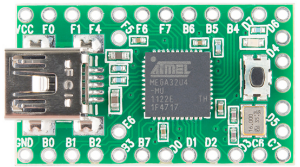
- Computation
- Memory
- Inputs
- Outputs
- Timers
- Analog to Digital Converter
- Communication

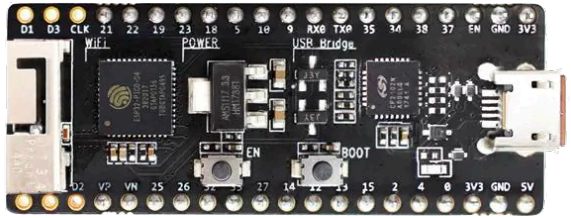
ESP32 Pico-Kit

- ESP32 *Wifi and Bluetooth* Module
  - ~50 foot range
- Operating voltage = 3.3V
- Voltage sources 3.3 – 12V
  - Through VIN pin or 5v through USB
- USB interface to PC or Mac
- 520 KiB SRAM, 4MB of flash memory
- 160 MHz clock speed (dual core)



Teensy VS NodeMCU

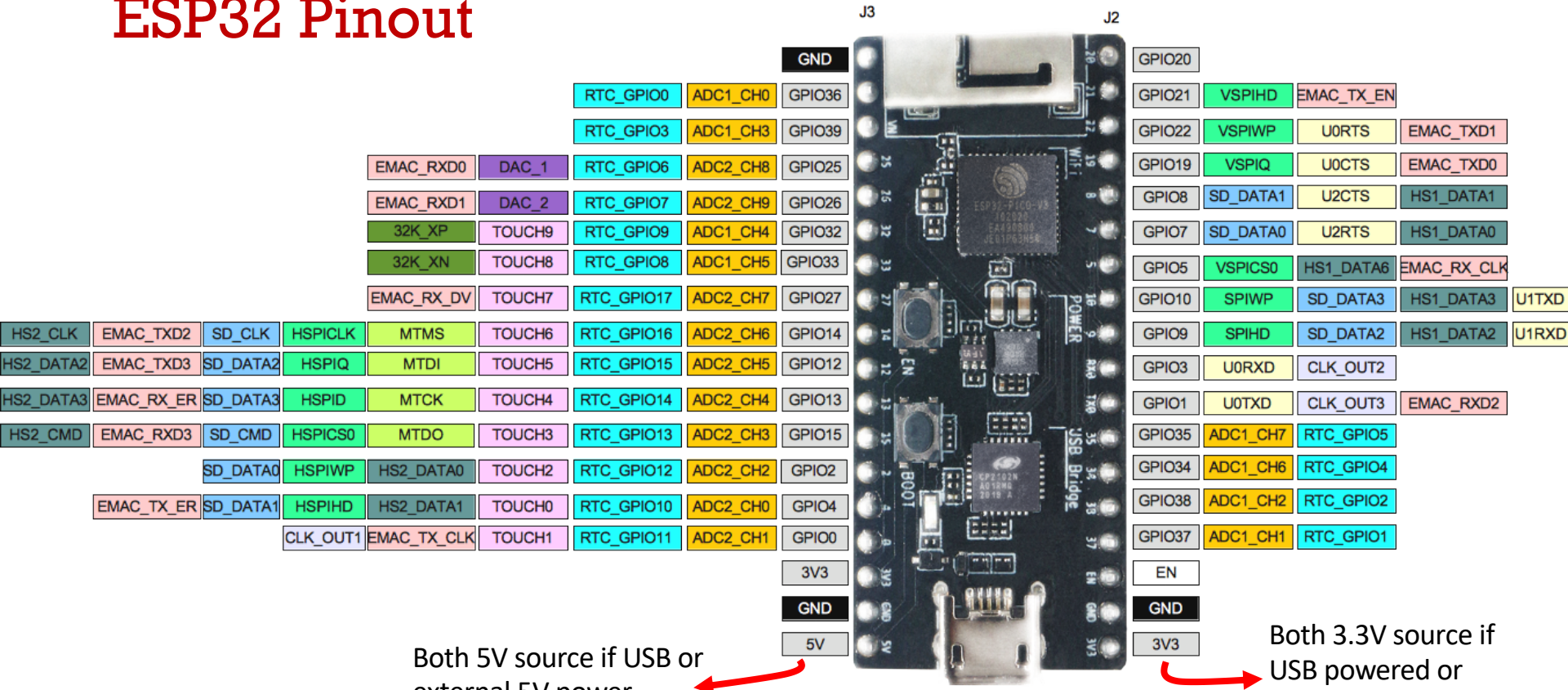




	ATMEGA 32U4	ESP32	ESP32 Difference
Processor Speed	16MHz, 8bit RISC	160Mhz, 32bit RISC	~20x faster comp
Memory	32k Flash; 2.5K RAM	4M Flash; 512K RAM	>100x's more
GP Input/Output	21 @ 5V	17 to 32 @ 3.3V	Similar
IO Current	20 mA (40 mA max)	10mA (20-40mA max)	Uncertain
ADC	12 channels 10-bit	16 channels 12-bit	33% More hi-res
Communication	SPI/UART/I2C	SPI/I2S/UART/I2C/BT/…	Much more
cost	$20	$10	half the price

Pico-kit
ESP32 Pinout

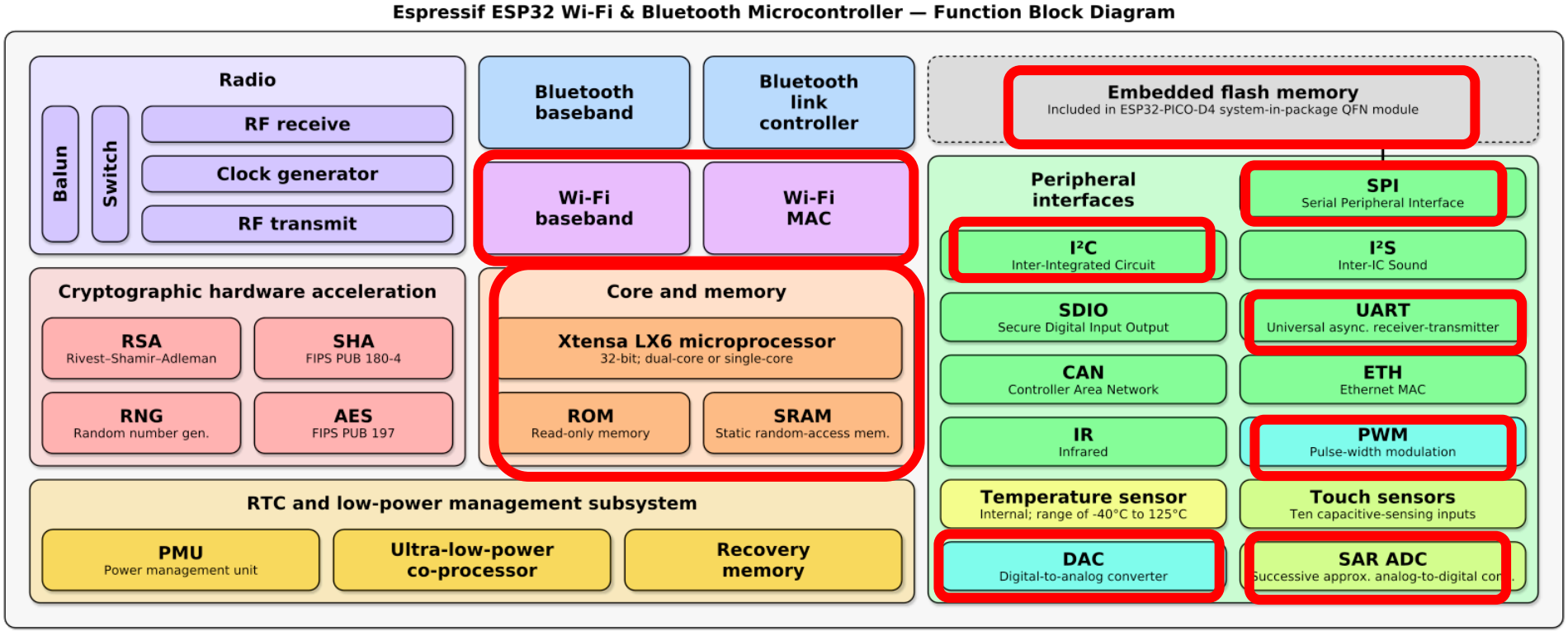
GPIO = General Purpose Input/Output
GND = ground or negative power line
VIN = positive power (5V typically)



Both 5V source if USB or external 5V power

Both 3.3V source if USB powered or external 3.3 power

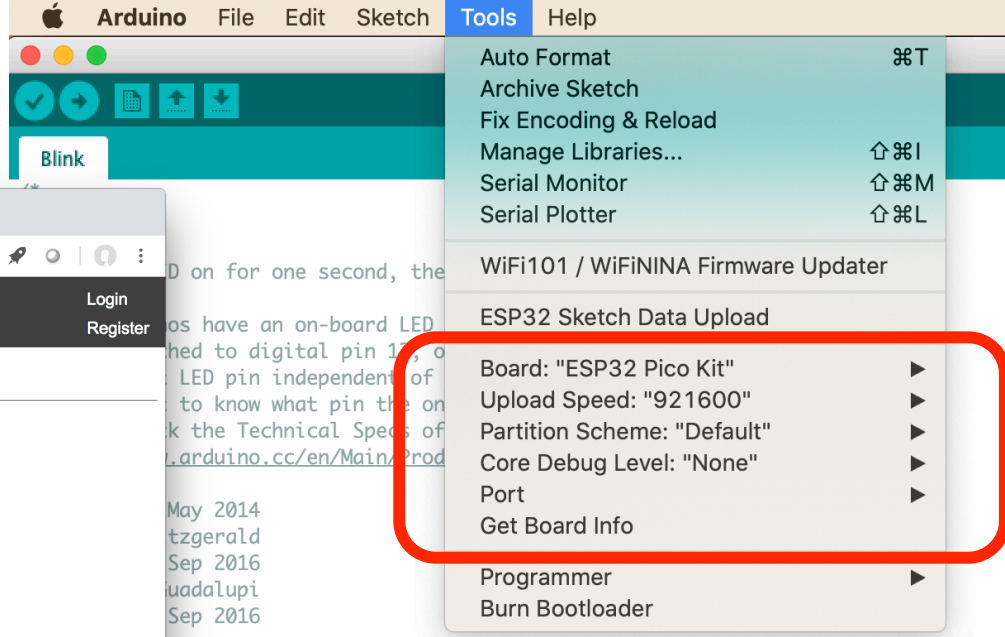
ESP32 Block Diagram

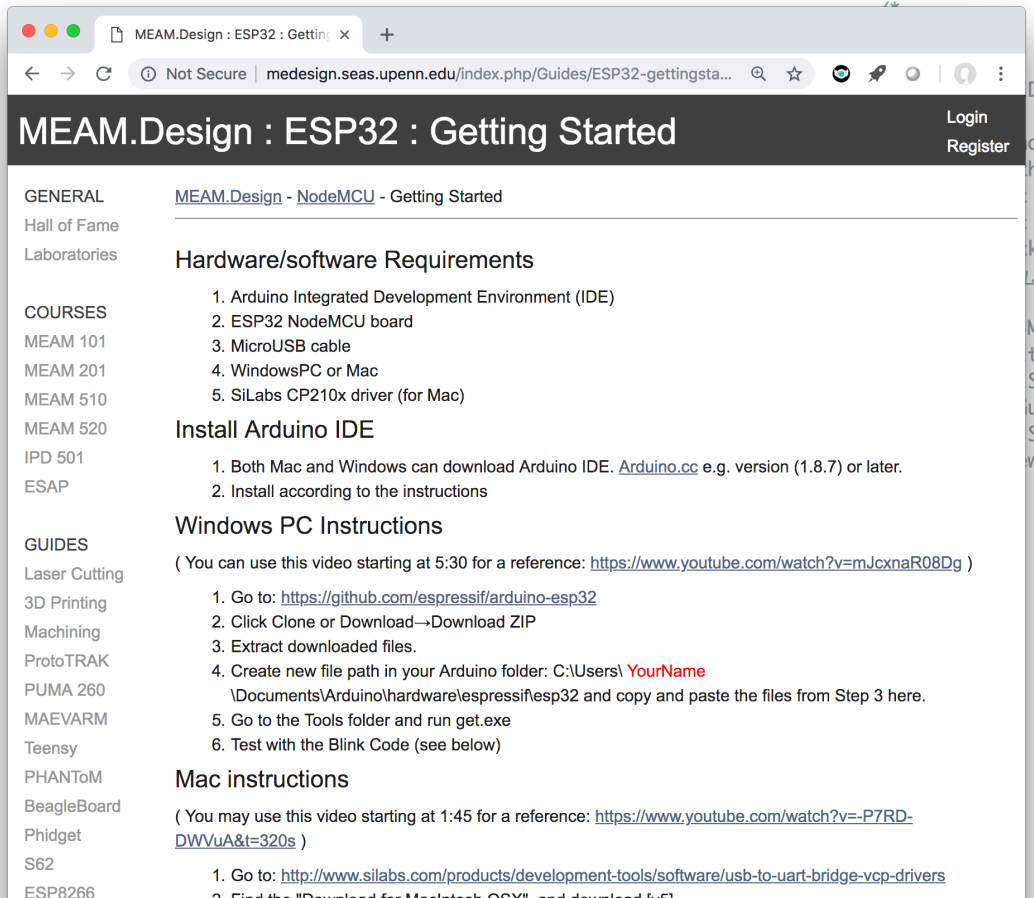


03

Software - Arduino

Arduino Set-up





http://medesign.seas.upenn.edu/index.php/Guides/ESP32-gettingstarted

Arduino Code Breakdown

Every Arduino "Sketch":

`setup()` runs once [first]

`loop()` repeats over and over.

```
void setup() {
    // put your setup code here, to run once:
    // These are comments "//"
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

"Hello World" in Arduino

1. Open:
<Files-> Examples -> 01.Basic -> BareMinimum>
2. Add the lines:
`Serial.begin(115200);`
in `setup()` section.
`Serial.println("hello world");`
in `loop()` section.
3. Click (✓) <Verify> to compile
4. Click (➔) <Upload> to Flash code to Arduino.
5. Click (magnifying glass) <serial monitor> (upper rt corner)

Q1 In public chat: list 5 functions we use with Teensy pins

Command Summary

Teensy	ESP32 Arduino
set/clear (DDRx, pin#)	• pinMode(pin#, INPUT/OUTPUT)
set/clear (PORTx, pin#)	• digitalWrite(pin#, HIGH/LOW)
bit_is_clear/set(PINx,pin#)	• digitalRead(pin#)
m_usb_init()	• Serial.begin()
m_usb_tx_uint(X)	• Serial.println()
_delay_ms()	• delay()
ADC routines…	• analogRead(pin#)
PWM timer routines…	• ~~analogWrite(pin#)~~ not for ESP32
	• ledc routines… (will cover later)
	• dacWrite(pin#)

Arduino Blink Sketch

LED_BUILTIN doesn't exist on pico

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED_BUILTIN pin as an output
}


// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, LOW);    // Turn the LED on (Note write LOW)
                                     // but actually the LED is on;
                                     // LED it is active low on the ESP

  delay(1000);                       // Wait for a second
  digitalWrite(LED_BUILTIN, HIGH);   // Turn the LED off by making pin HIGH
  delay(2000);                       // Wait for two secs
}
```

Functions: in Blink

`pinMode(pin,OUTPUT);`

- Tells the micro that `pin` is an output pin.
- Modes can be INPUT, OUTPUT, INPUT_PULLUP, INPUT_PULLDOWN

Available on ESP32, not most others

`digitalWrite(pin,HIGH);`

- Sets the state of digital pin # `pin` to HIGH (3.3V), LOW=0V,
- `pin` is the GPIO number, can be 0, 1, 2… 39 for ESP32
- Similar to `set(PORT,PIN);` and `clear(PORT,PIN);`

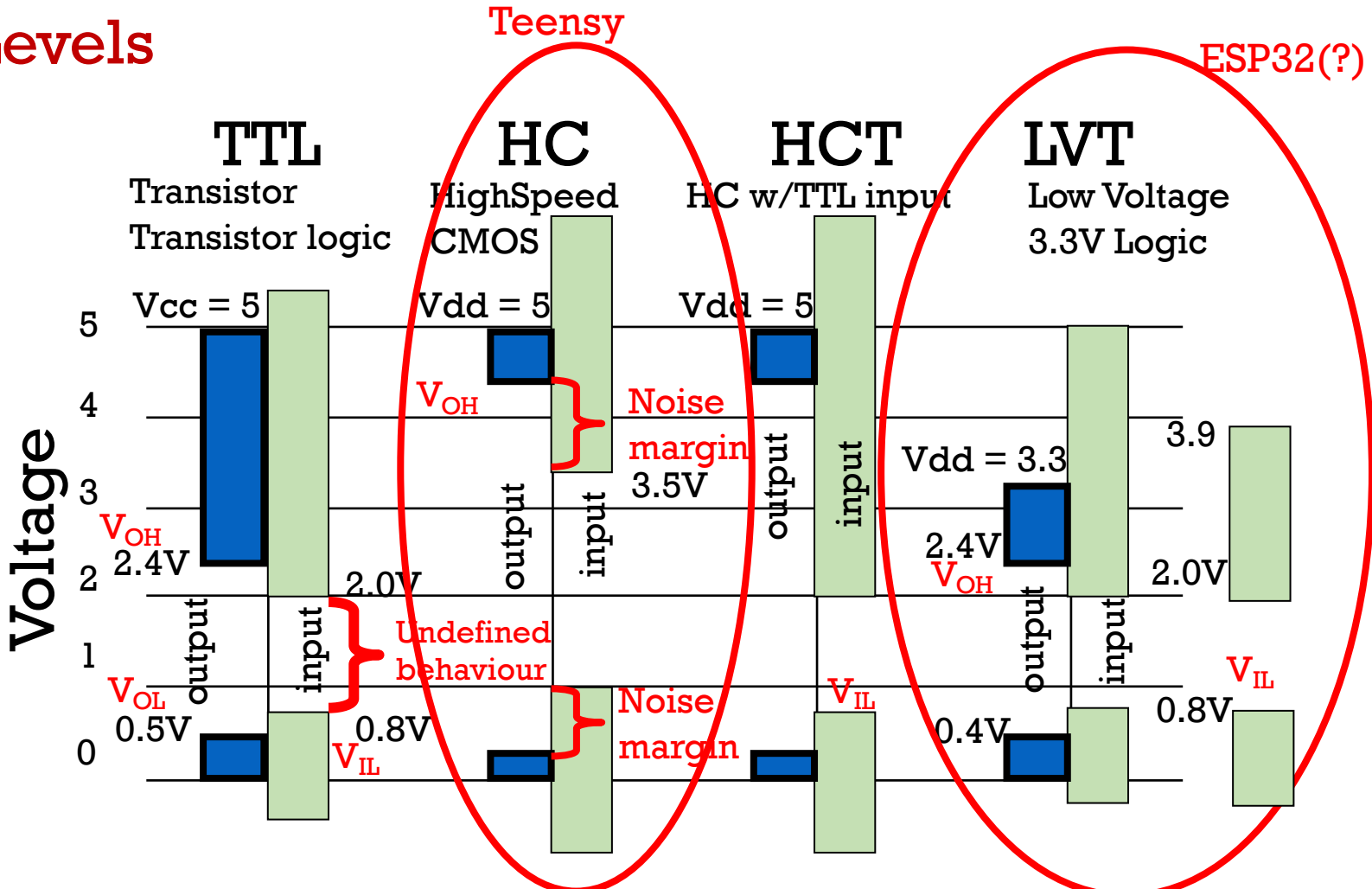
`delay(1000);`

- Delays further execution of code for 1000 milliseconds. (Busy waits)

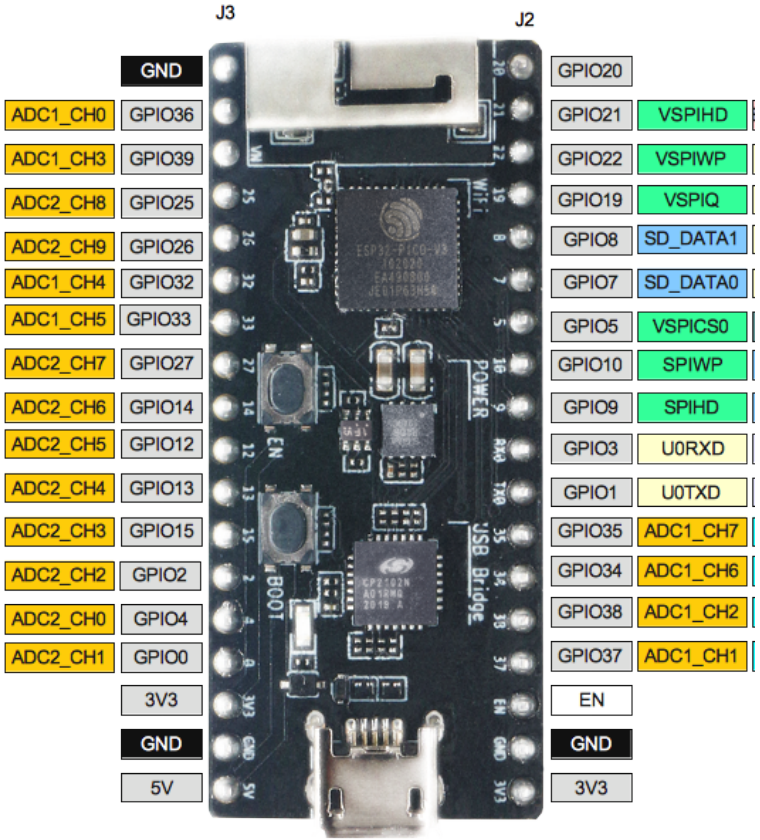
Logic Levels

Valid Output Levels

Valid Recognized Input Levels



How do we know what to call a pin?



In Arduino/hardware/espressif/esp32/variants/pico32/

pins_arduino.h

```
#ifndef Pins_Arduino_h
#define Pins_Arduino_h

#include <stdint.h>

#define EXTERNAL_NUM_INTERRUPTS 16
#define NUM_DIGITAL_PINS        40
#define NUM_ANALOG_INPUTS       16

#define analogInputToDigitalPin(p)   (((p)<20
#define digitalPinToInterrupt(p)     (((p)<40
#define digitalPinHasPWM(p)          (p < 34

static const uint8_t TX = 1;
static const uint8_t RX = 3;

static const uint8_t SDA = 21;
static const uint8_t SCL = 22;
```

Other Functions

`digitalRead(pin);`

- Returns the state of digital pin # `pin` as HIGH or LOW,
- `pin` can be 0, 1, 2 ... 39

`delayMicroseconds(x);`

- Delays further execution of code for `x` microseconds. (Busy waits)

`Serial.begin(115200);`

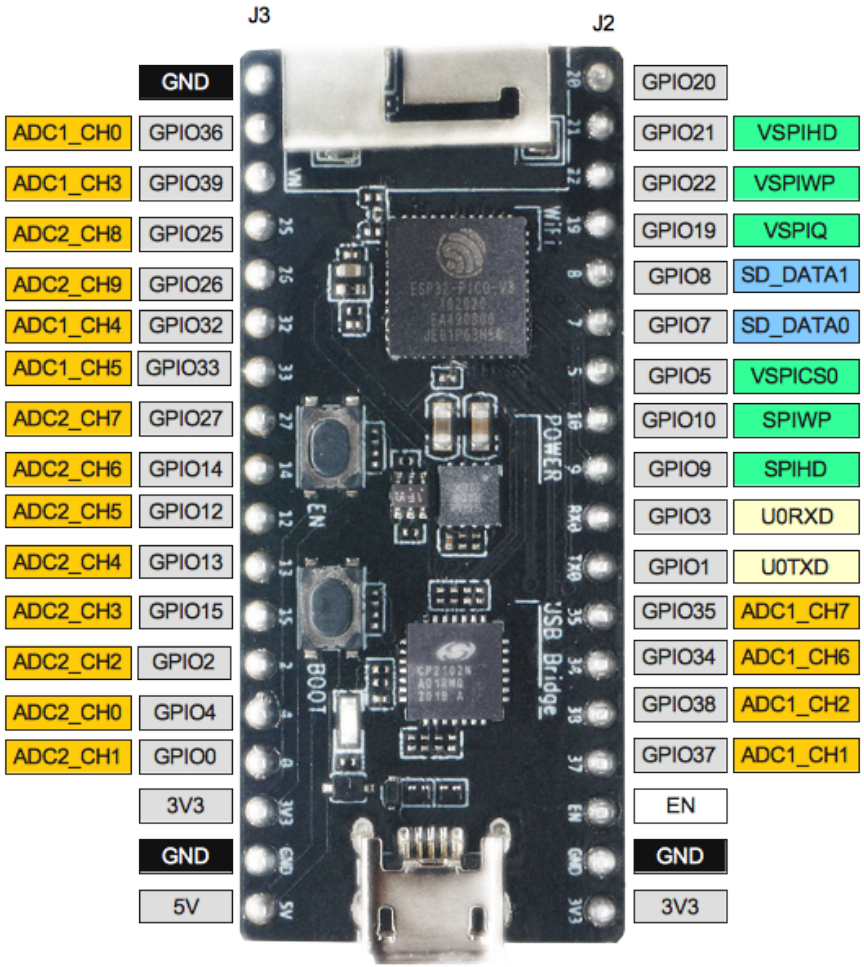
- Needed for serial comms.
- 115200 is the baudrate. 9600 is another standard rate.

`Serial.println(x);` `Serial.print("string");`

- `println()` with carriage return, `print()` without carriage return.

Analog to Digital Converter

- Reads voltage at pin
- 18 available ADC pins
- 12bit SAR ADC
- Sample rate (about 6kHz)
- Can be referred to the ADC channel number A#, or GIPO number
  - ADC channel 0-> A0 = 36



analogRead(pin)

- Reads the ESP32 ADC channel associated with pin.
- Returns an integer from 0 – 4095 (12 bit)

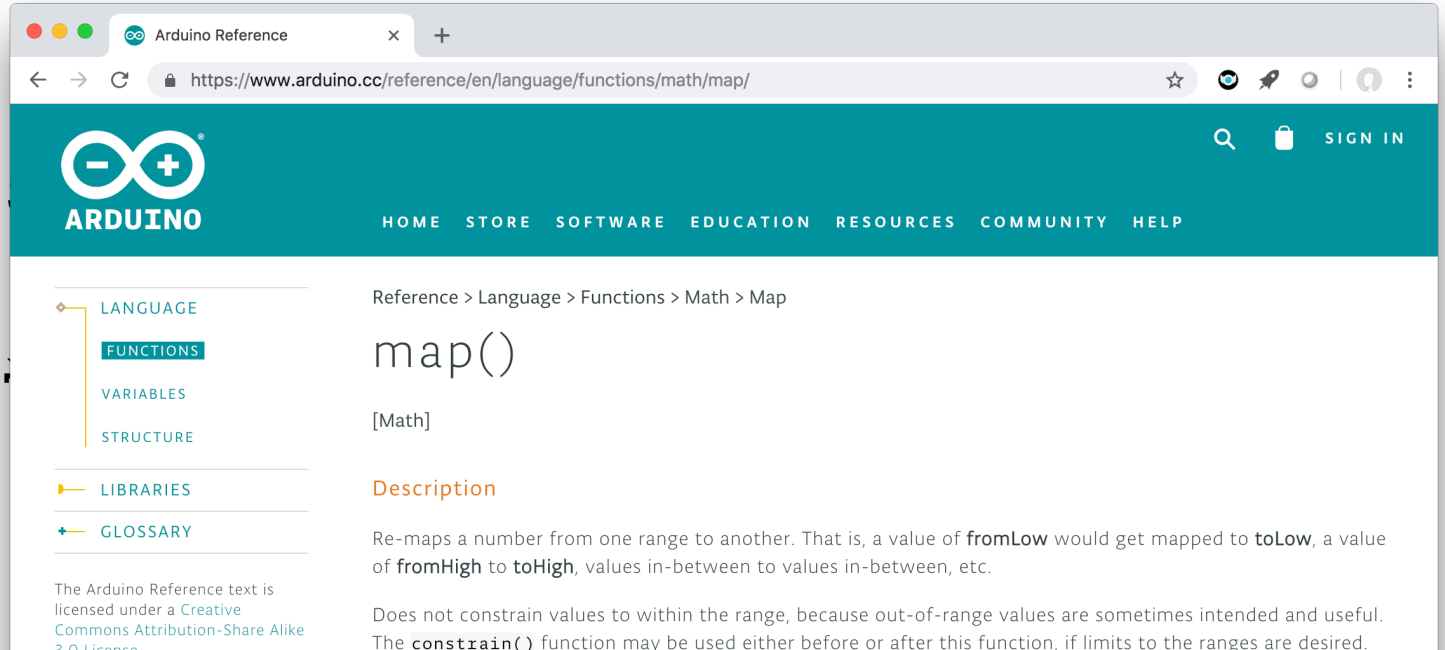
```
void setup() {
  Serial.begin(115200);
}

void loop() {
  Serial.println(analogRead(A0));
  delay(200);
}
```

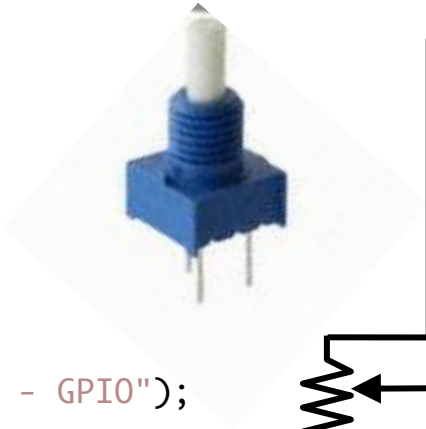
`analogRead()`, `map(value,inL,inH,outL,outH)`

- Can use arduino `map()` to convert linear units. E.g. from ADC counts (0-4095) to millivolts (0-3300).

https://www.arduino.cc/reference/en/language/functions/math/map/

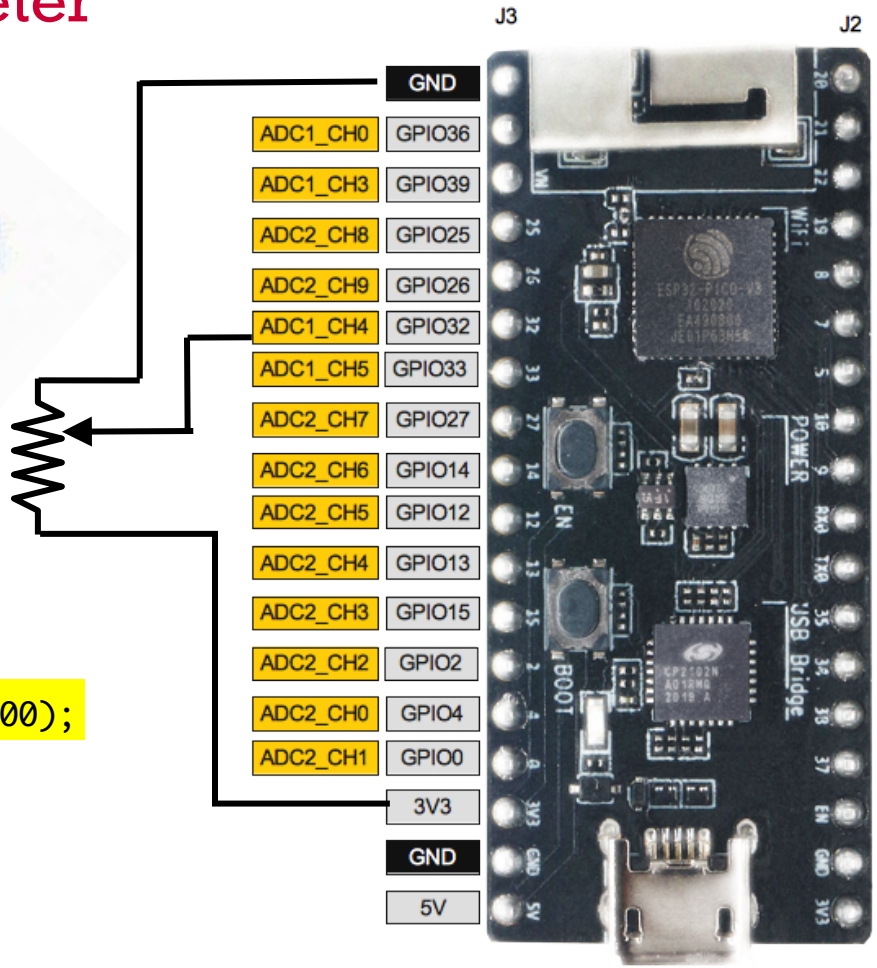


Example reading a potentiometer



```
#define ANALOG_PIN A4
void setup() {
  Serial.begin(115200);
  Serial.print("ESP32 Analog Pin - GPIO");
}

void loop() {
    int mv;
    mv = map(analogRead(ANALOG_PIN),0,4095,0,3300);
    Serial.println (mv);
       // prints 0-4095 <-> 0 – 3300mV
    delay(500);
}
```



Q2 Circle what we would change to make the code read ADC channel 3 and print value in Volts (not mv)

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

```
#define ANALOG_PIN A4
void setup() {
  Serial.begin(115200);
  Serial.print("ESP32 Analog Pin - GPIO");
}

void loop() {
    int mv;
    mv = map(analogRead(ANALOG_PIN),0,4095,0,3300);
    Serial.println (mv);
       // prints 0-4095 <-> 0 – 3300mV
    delay(500);
}
```

Q4 (in Chat) What would write in this line to print Volts?

`analogWrite()` (actually pseudo "analog")

- In Arduino analogWrite is Actually a modulated wave whose pulse width varies to set the effective voltage (PWM).
- A real analog write would be a DAC – digital to analog converter. It would output smooth voltage, not a modulated wave form.

- NOTE: This Arduino library function is not yet available for ESP32!

dacWrite() (a real "analog" Write)

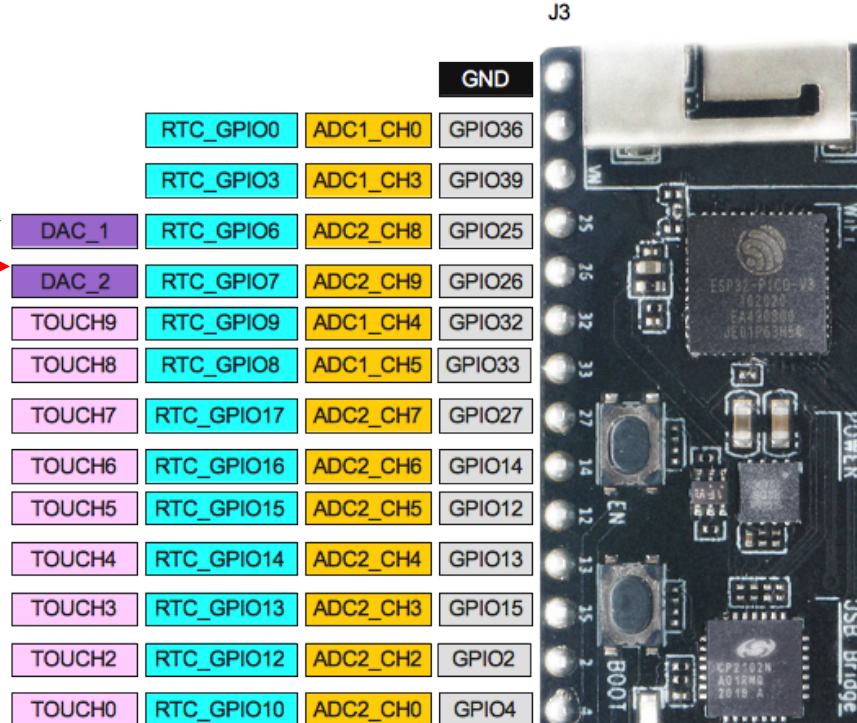
- **ESP32 has two Digital to Analog converters (GPIO 25 or 26)**

```
dacWrite(GPIOpin#, [0:255]); // 0 = 0V, 255=3.3V
```

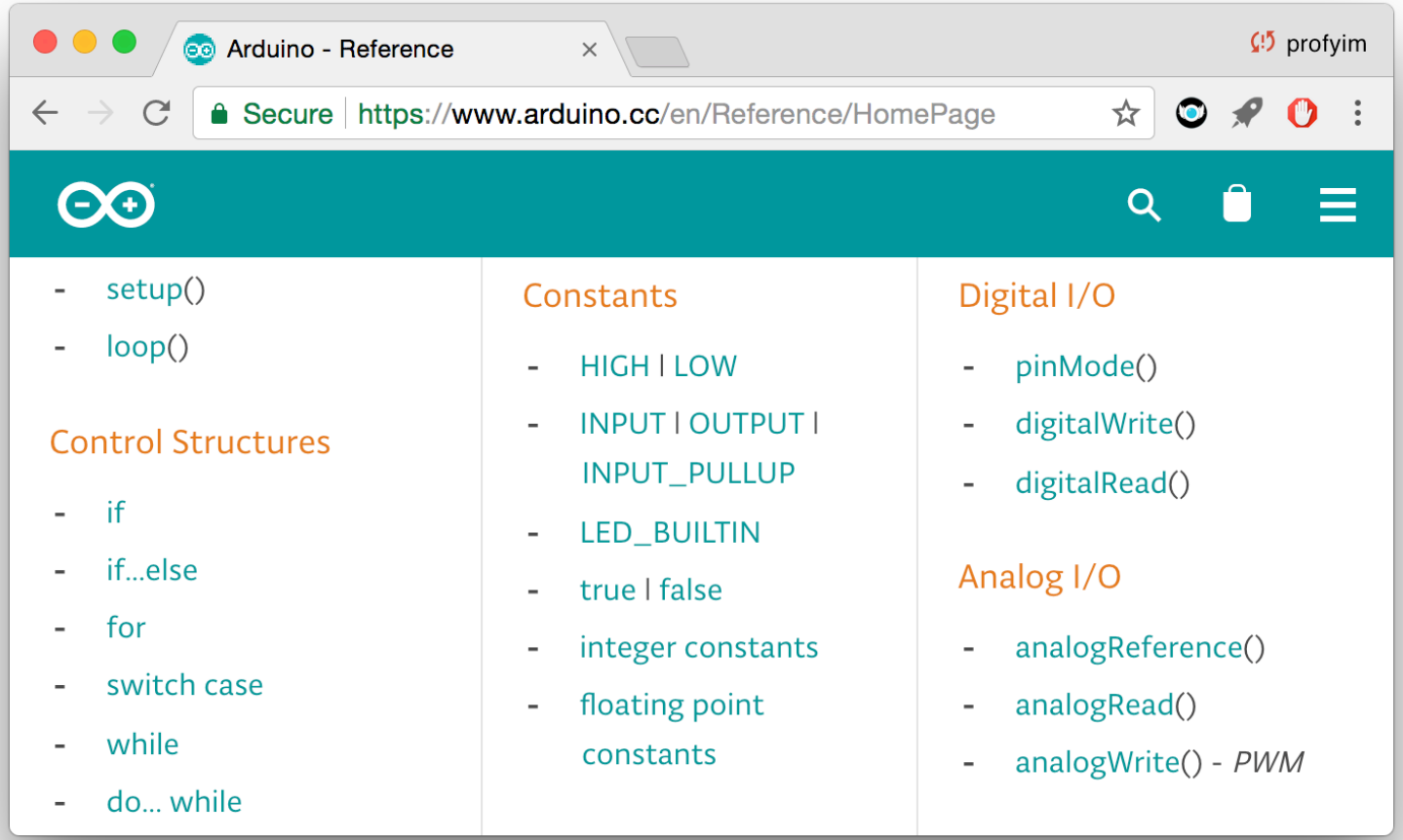
```
void setup() {
  Serial.begin (115200);
}

void loop() {
   int i;
   for( i=0;i<256;i++)) {
      dacWrite(25,i);
      Serial.println(i);
      delay(100);
   }
}
```

Ex: this one

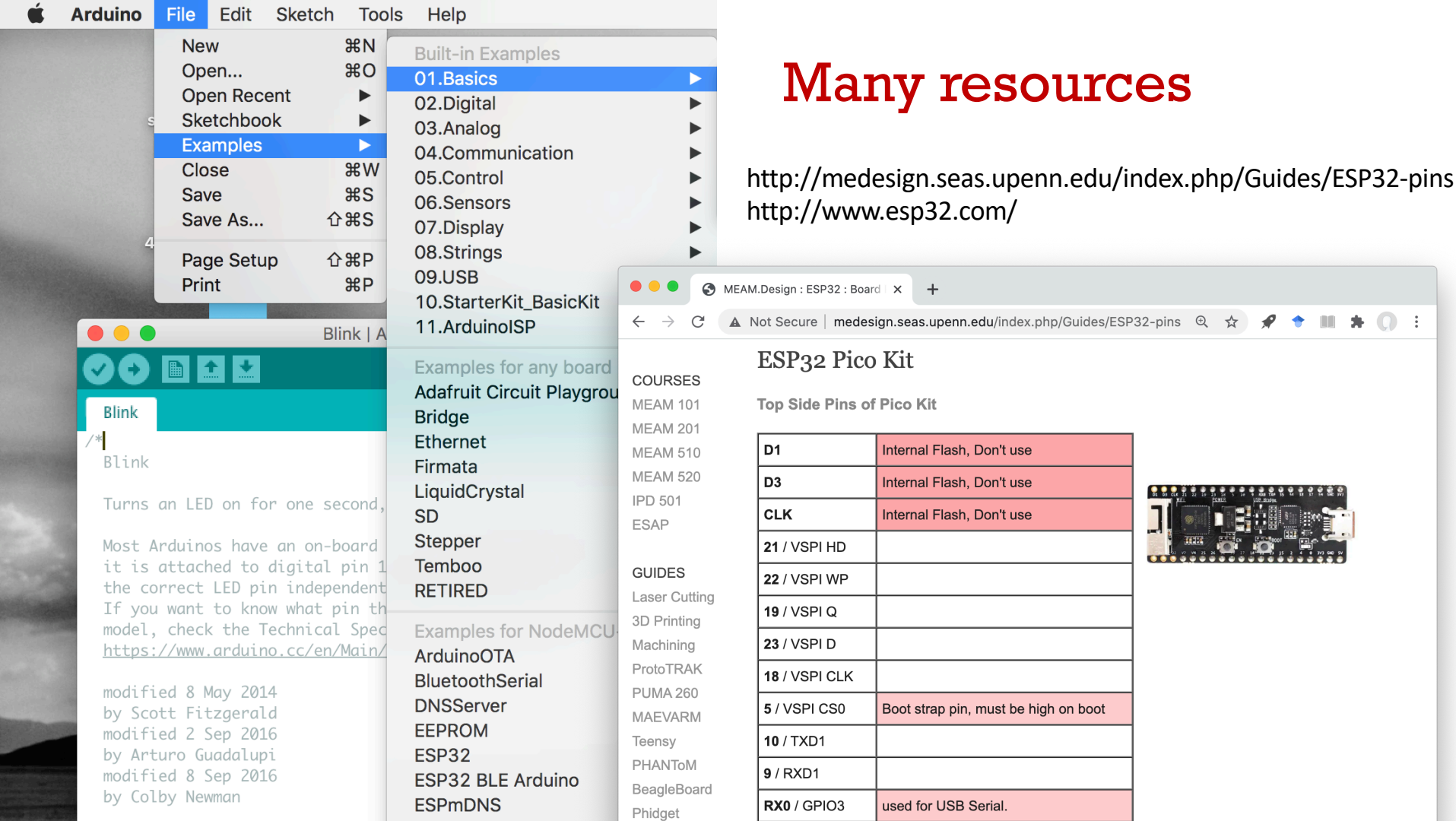


References online (www.arduino.cc)



Many resources

http://medesign.seas.upenn.edu/index.php/Guides/ESP32-pins
http://www.esp32.com/

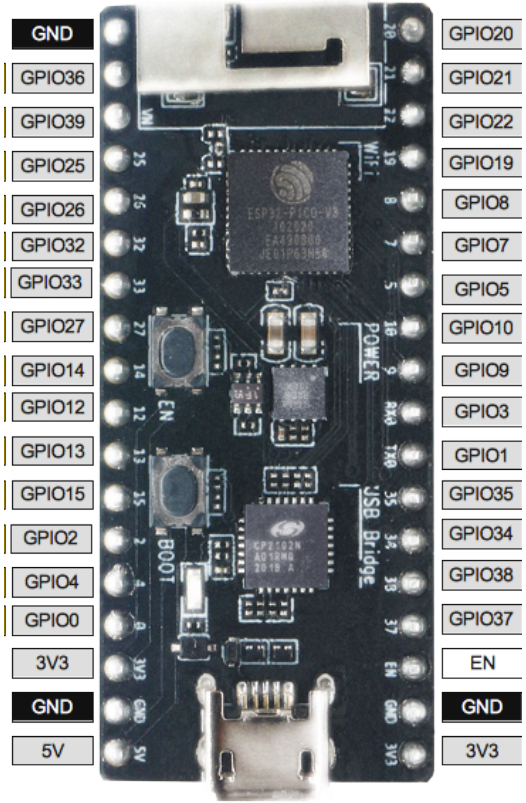


04

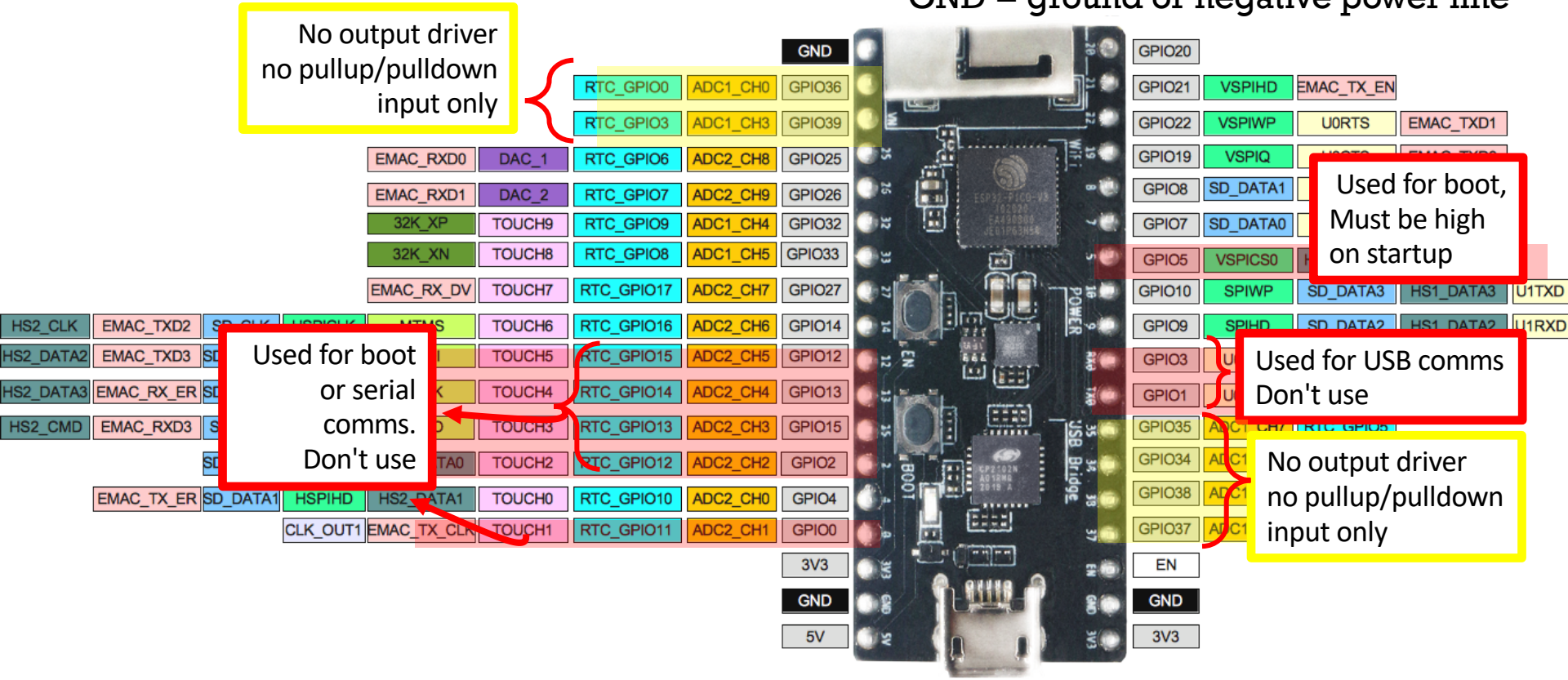
ESP32 PicoKit Hardware

General Purpose Input/Output (GPIO)

- 29 pins labeled as GPIO (only 20 are safe to use, see next slides for explanation)
- 14 GPIO pins can be input or output
- All use 3.3V
  - HIGH = 3.3V
  - LOW = 0V
- Internal pullup or pulldown
- RED LED turns on if powered.

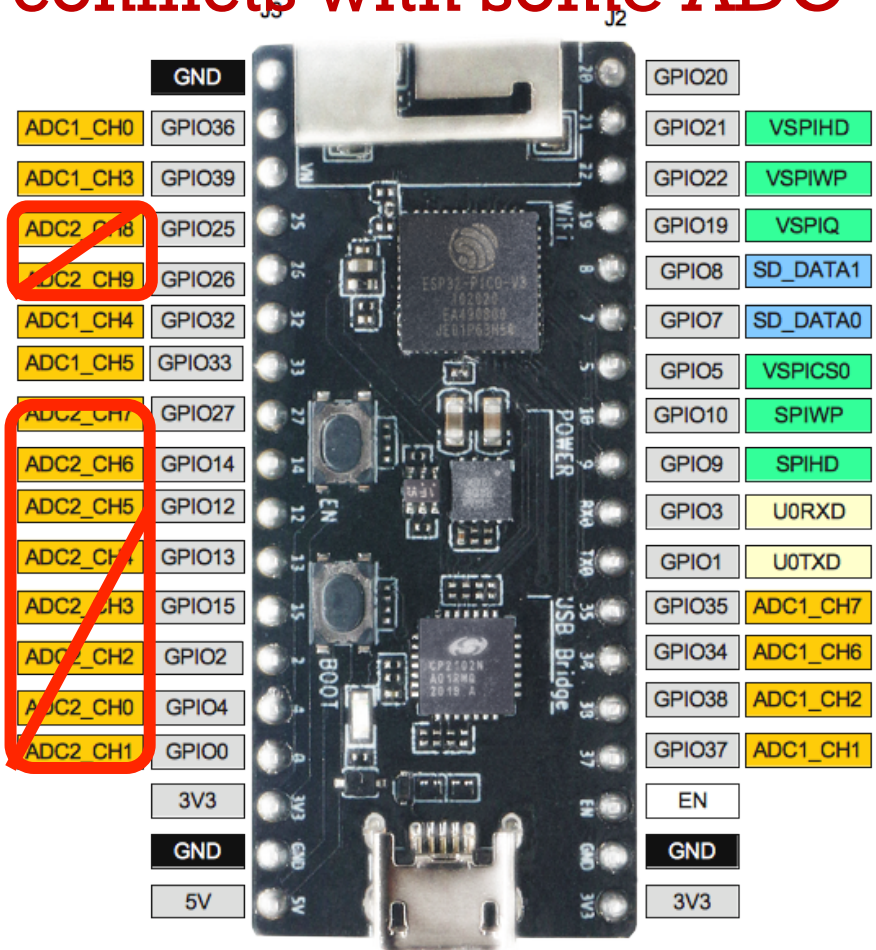


GPIO = General Purpose Input/Output
GND = ground or negative power line



More ESP Issues – WiFi conflicts with some ADC

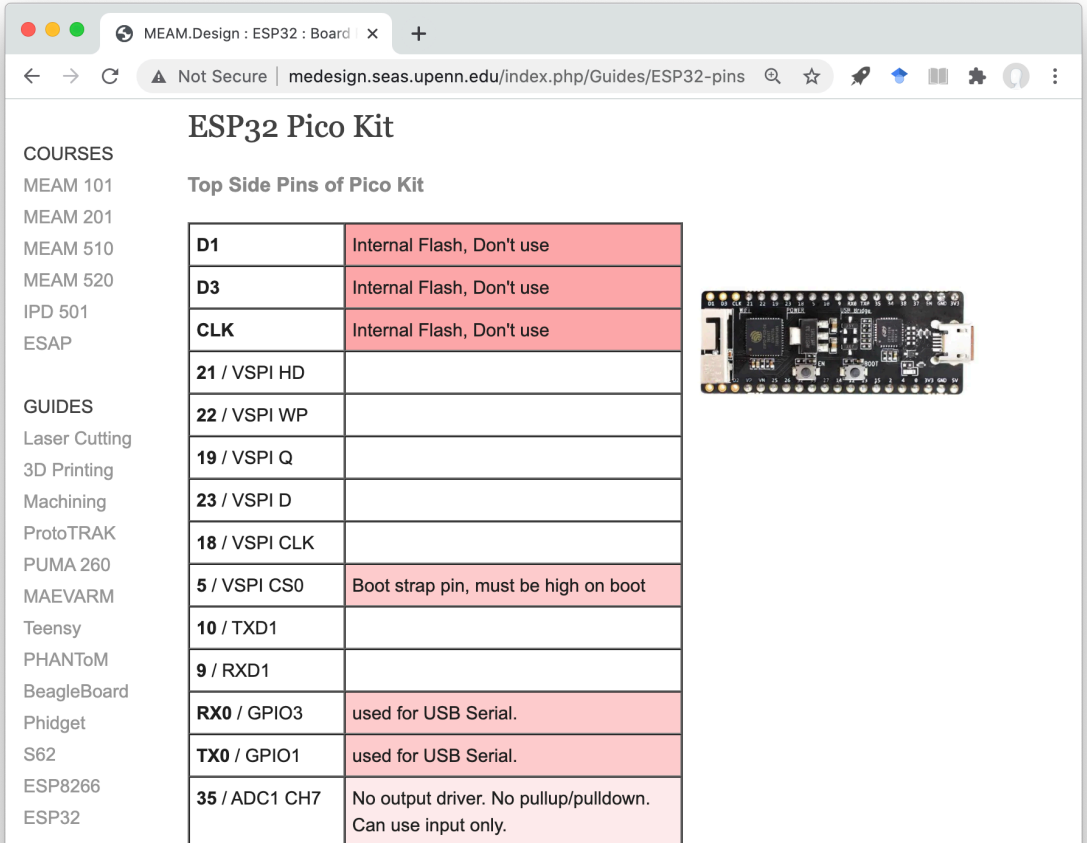
- On PicoKit 18 ADC pins are available.
- 10 of them (ADC#>9) do not work when WiFi is on.



Refer to medesign website for summary of caveats

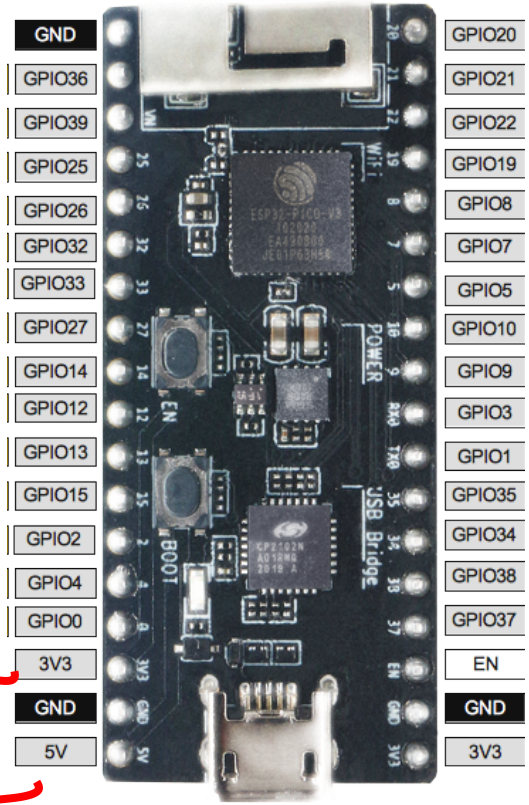
http://medesign.seas.upenn.edu/index.php/Guides/ESP32-pins

- Don't use the pins that are highlighted in red



PicoKit w/external power (e.g. battery)

- The PicoKit has a protection diode for your USB. So you can connect up to 12V on the Vin pin
  - even while USB is plugged in!
- Their design of the protection diode and voltage regulator is not great, so it causes problems we'll talk about later...

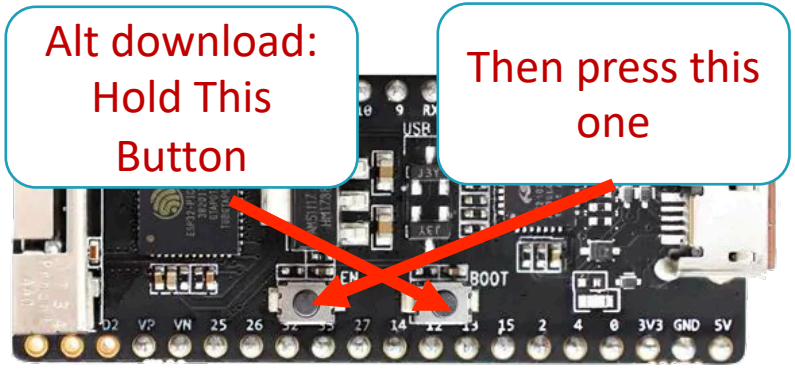


Both 3.3V source if USB powered or supply external 3.3V power

Both 5V source if USB or supply external power up to 12V

ESP32 Issues

- If ESP32 mysteriously stop loading.
  - It is likely dead. We suspect that it is Electrostatic Static Discharge (ESD).
  - **Solution: Be sure to ground yourself always before touching ESP32 including hitting the download button.**
- NodeMCU and PicoKit may not work with all USB cables. The power design is not great.
  - **Use supplied USB cable. Later use external power.**



The Good

1. WiFi on a chip - easy to setup a webpage
2. Full TCP/IP stack firmware w/RTOS
3. Very cheap ($3 for module, $10 for board)
4. Arduino SDK
5. DIY community is eating it up.
6. ~1M available program space
7. Enough I/O to do many things
   A. 18 ADC
   B. 2 SPI
   C. 2 UART
   D. I2S, I2C
   E. ~17 useful GPIO

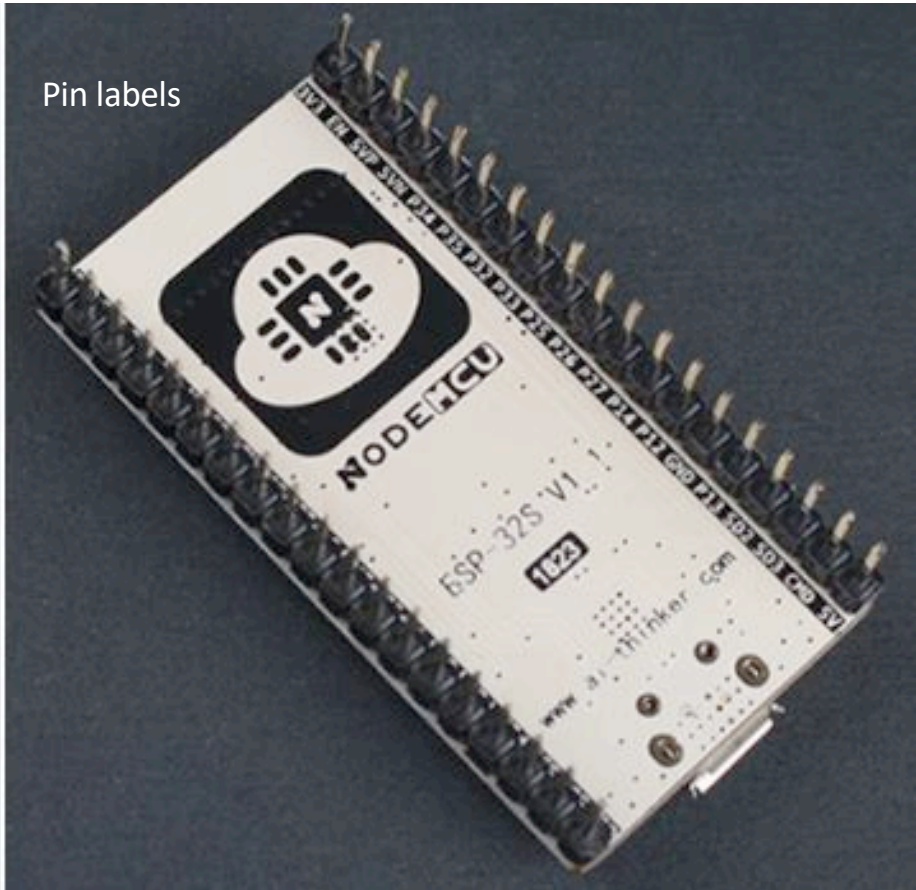
The Bad

1. WiFi only with port forwarding on router.
2. Arduino SDK only partially implemented
   A. No servo library
   B. TCP is slow and timing is inconsistent
   C. WiFi Latency ~130ms per packet
   D. UDP is reasonably fast though ~500 packets /sec.
3. Timing not guaranteed (in arduino, PWM can be glitchy)
4. Downloading is much slower than Teensy
5. Power circuit design is borderline bad
6. QA for hiletgo is bad - ~2% failure out of box

The Ugly

1. 3.3V interface (ports maybe 5V safe?)
   A. Use 5V level shifters from sparkfun (elegant design)
2. Documentation is horrendous (getting better)
3. RTOS adds complication
4. Many ports are multi-use in default conditions (e.g. GPIO used for startup)
5. Ports can probably safely sink/source 10mA (vs 20mA for ATmega)
6. Randomly gets hot and dies… ESD?

NodeMCU version of ESP32
(we have some extras from previous years)



Summary

- ESP32 is much more powerful than teensy.
- ESP32 uses 3.3V logic
- ESP32 documentation is very bad.
- When developing with ESP32 be careful of ESD!
- We will use Arduino SDK.