Playing With A $7 AI-Thinker ESP32-CAM IoT Development Board

Henry Dietz

Keeping Current talk, 16:30, March 17, 2021

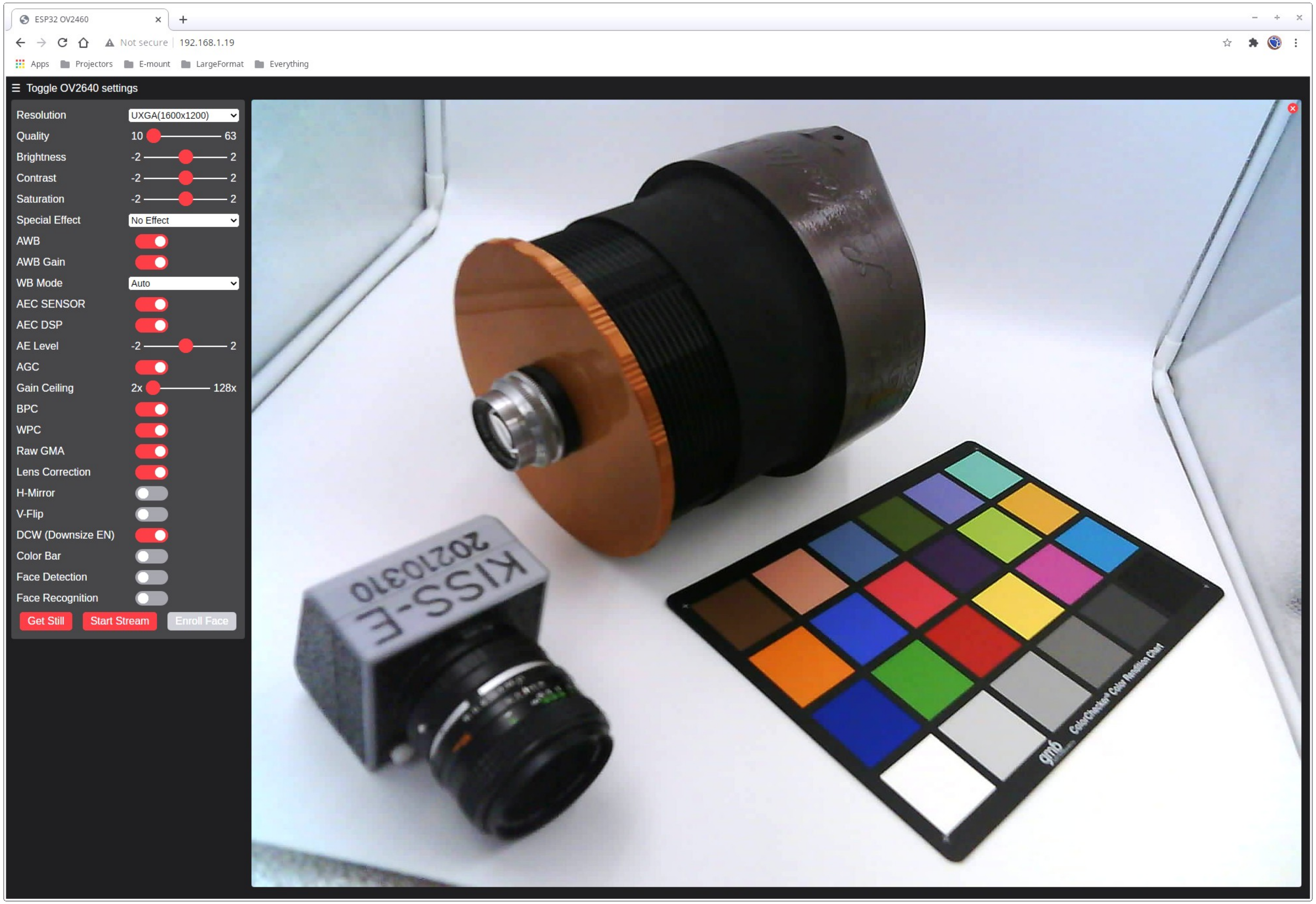
University of Kentucky
Electrical & Computer Engineering

University of Kentucky

Abstract

Abstract: *The Arduino IDE and C++ libraries make microcontrollers easy to use — and some compatible systems are now surprisingly small, cheap, and powerful. The $7 AI-Thinker ESP32-CAM development board is designed to allow Internet-of-Things (IoT) devices to use face recognition and other computationally demanding algorithms. In 27×40mm, this board provides a 240MHz dual-core 32-bit processor, 2MP camera, 802.11b/g/n WiFi and BlueTooth, as well as wired I/O interfaces, a microSD slot and low-power modes. I will present an overview of the ESP32-CAM, discuss how to work around some of its quirks, and briefly show a few sample applications. For example, a single ESP32-CAM was used to implement Lafodis160, the LArge FOrmat DIgital Scanning camera I presented at Electronic Imaging 2021.*

The ESP32-CAM As An IoT Camera

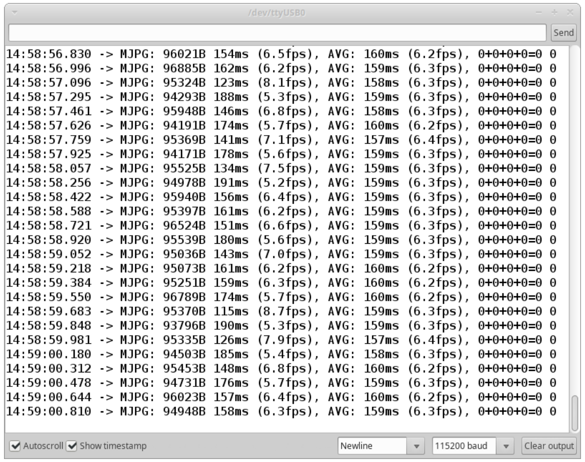


☰ Toggle OV2640 settings

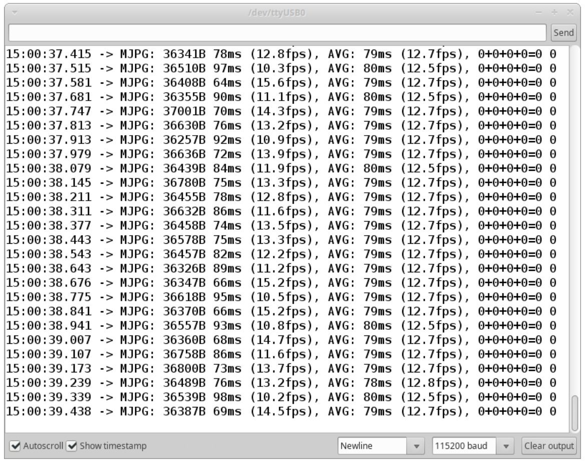
Resolution	UXGA(1600x1200)
Quality	10 ● 63
Brightness	-2 ● 2
Contrast	-2 ● 2
Saturation	-2 ● 2
Special Effect	No Effect
AWB	●
AWB Gain	●
WB Mode	Auto
AEC SENSOR	●
AEC DSP	●
AE Level	-2 ● 2
AGC	●
Gain Ceiling	2x ● 128x
BPC	●
WPC	●
Raw GMA	●
Lens Correction	●
H-Mirror	○
V-Flip	○
DCW (Downsize EN)	●
Color Bar	○
Face Detection	○
Face Recognition	○

Not bad for a **$7 programmable camera with WiFi!**

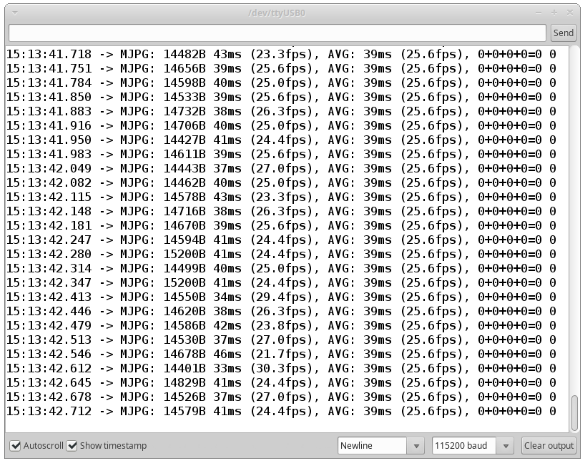
The ESP32-CAM As An IoT Camera



```
14:58:56.830 -> MJPG: 96021B 154ms (6.5fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:58:56.996 -> MJPG: 96885B 162ms (6.2fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:57.096 -> MJPG: 95324B 123ms (8.1fps), AVG: 158ms (6.3fps), 0+0+0+0=0 0
14:58:57.295 -> MJPG: 94293B 188ms (5.3fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:57.461 -> MJPG: 95948B 146ms (6.8fps), AVG: 158ms (6.3fps), 0+0+0+0=0 0
14:58:57.626 -> MJPG: 94191B 174ms (5.7fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:58:57.759 -> MJPG: 95369B 141ms (7.1fps), AVG: 157ms (6.4fps), 0+0+0+0=0 0
14:58:57.925 -> MJPG: 94171B 178ms (5.6fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.057 -> MJPG: 95525B 134ms (7.5fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.256 -> MJPG: 94978B 191ms (5.2fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.422 -> MJPG: 95940B 156ms (6.4fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.588 -> MJPG: 95397B 161ms (6.2fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.721 -> MJPG: 96524B 151ms (6.6fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:58.920 -> MJPG: 95539B 180ms (5.6fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:59.052 -> MJPG: 95036B 143ms (7.0fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:59.218 -> MJPG: 95073B 161ms (6.2fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:58:59.384 -> MJPG: 95251B 159ms (6.3fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:58:59.550 -> MJPG: 96789B 174ms (5.7fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:58:59.683 -> MJPG: 95370B 115ms (8.7fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:59.848 -> MJPG: 93796B 190ms (5.3fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:58:59.981 -> MJPG: 95335B 126ms (7.9fps), AVG: 157ms (6.4fps), 0+0+0+0=0 0
14:59:00.180 -> MJPG: 94503B 185ms (5.4fps), AVG: 158ms (6.3fps), 0+0+0+0=0 0
14:59:00.312 -> MJPG: 95453B 148ms (6.8fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:59:00.478 -> MJPG: 94731B 176ms (5.7fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
14:59:00.644 -> MJPG: 96023B 157ms (6.4fps), AVG: 160ms (6.2fps), 0+0+0+0=0 0
14:59:00.810 -> MJPG: 94948B 158ms (6.3fps), AVG: 159ms (6.3fps), 0+0+0+0=0 0
```



```
15:00:37.415 -> MJPG: 36341B 78ms (12.8fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:37.515 -> MJPG: 36510B 97ms (10.3fps), AVG: 80ms (12.5fps), 0+0+0+0=0 0
15:00:37.581 -> MJPG: 36408B 64ms (15.6fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:37.681 -> MJPG: 36355B 90ms (11.1fps), AVG: 80ms (12.5fps), 0+0+0+0=0 0
15:00:37.747 -> MJPG: 37001B 70ms (14.3fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:37.813 -> MJPG: 36630B 76ms (13.2fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:37.913 -> MJPG: 36257B 92ms (10.9fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:37.979 -> MJPG: 36636B 72ms (13.9fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.079 -> MJPG: 36439B 84ms (11.9fps), AVG: 80ms (12.5fps), 0+0+0+0=0 0
15:00:38.145 -> MJPG: 36780B 75ms (13.3fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.211 -> MJPG: 36455B 78ms (12.8fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.311 -> MJPG: 36632B 86ms (11.6fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.377 -> MJPG: 36458B 74ms (13.5fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.443 -> MJPG: 36578B 75ms (13.3fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.543 -> MJPG: 36457B 82ms (12.2fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.643 -> MJPG: 36326B 89ms (11.2fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.676 -> MJPG: 36347B 66ms (15.2fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.775 -> MJPG: 36618B 95ms (10.5fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.841 -> MJPG: 36370B 66ms (15.2fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:38.941 -> MJPG: 36557B 93ms (10.8fps), AVG: 80ms (12.5fps), 0+0+0+0=0 0
15:00:39.007 -> MJPG: 36360B 68ms (14.7fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:39.107 -> MJPG: 36758B 86ms (11.6fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:39.173 -> MJPG: 36800B 73ms (13.7fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
15:00:39.239 -> MJPG: 36489B 76ms (13.2fps), AVG: 78ms (12.8fps), 0+0+0+0=0 0
15:00:39.339 -> MJPG: 36539B 98ms (10.2fps), AVG: 80ms (12.5fps), 0+0+0+0=0 0
15:00:39.438 -> MJPG: 36387B 69ms (14.5fps), AVG: 79ms (12.7fps), 0+0+0+0=0 0
```



```
15:13:41.718 -> MJPG: 14482B 43ms (23.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.751 -> MJPG: 14656B 39ms (25.6fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.784 -> MJPG: 14598B 40ms (25.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.850 -> MJPG: 14533B 39ms (25.6fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.883 -> MJPG: 14732B 38ms (26.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.916 -> MJPG: 14706B 40ms (25.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.950 -> MJPG: 14427B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:41.983 -> MJPG: 14611B 39ms (25.6fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.049 -> MJPG: 14443B 37ms (27.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.082 -> MJPG: 14462B 40ms (25.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.115 -> MJPG: 14578B 43ms (23.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.148 -> MJPG: 14716B 38ms (26.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.181 -> MJPG: 14670B 39ms (25.6fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.247 -> MJPG: 14594B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.280 -> MJPG: 15200B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.314 -> MJPG: 14499B 40ms (25.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.347 -> MJPG: 15200B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.413 -> MJPG: 14550B 34ms (29.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.446 -> MJPG: 14620B 38ms (26.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.479 -> MJPG: 14586B 42ms (23.8fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.513 -> MJPG: 14530B 37ms (27.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.546 -> MJPG: 14678B 46ms (21.7fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.612 -> MJPG: 14401B 33ms (30.3fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.645 -> MJPG: 14829B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.678 -> MJPG: 14526B 37ms (27.0fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
15:13:42.712 -> MJPG: 14579B 41ms (24.4fps), AVG: 39ms (25.6fps), 0+0+0+0=0 0
```

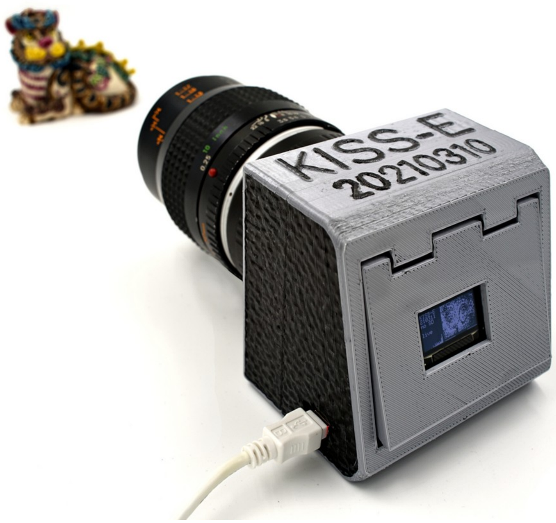
- How fast is it? **Not video speeds**... but not terrible:

1600x1200	UXGA	~**6.2 FPS** (frames per second)
800x600	SVGA	~**12.7 FPS**
400x296	CIF	~**25.6 FPS**

- Rolling shutter at ~**1/50s**
  - Typical for webcam, slow for a hand-held camera (1/*f* s rule)
  - Tiny lens can effectively magnify camera shake
  - Can show distortions within a frame

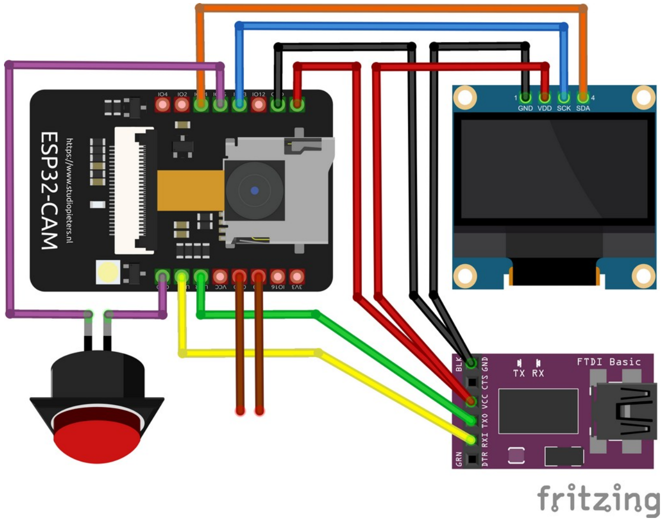
KISS-E: Kentucky's Interchangeable-lens Small Sensor E-mount camera

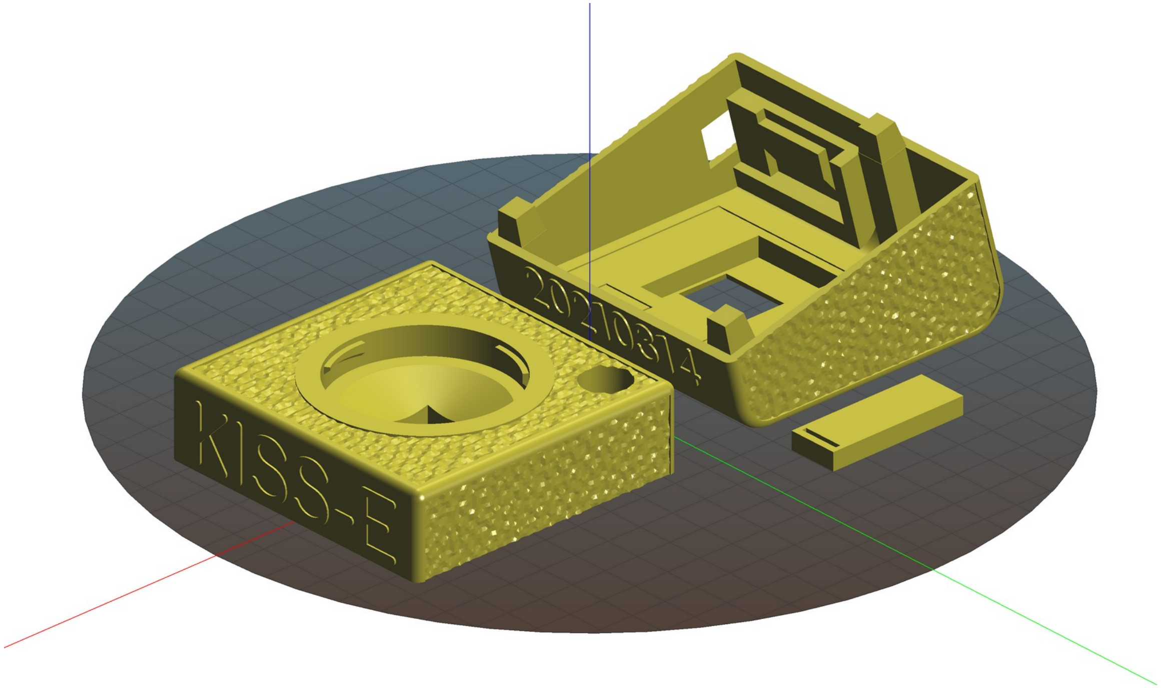
KISS-E
20210310



- Interchangeable-lens camera with Sony E-mount
  - Use stand alone, capture to TF card using OLED live view
  - Use tethered via USB (not UVC protocol… yet)
  - Use as IoT webcam via 802.11 or Bluetooth
- 1600x1200 native resolution, RGB color, no integrated NIR filter
- 9.8X crop factor: 50mm lens gives view of 491mm on FF

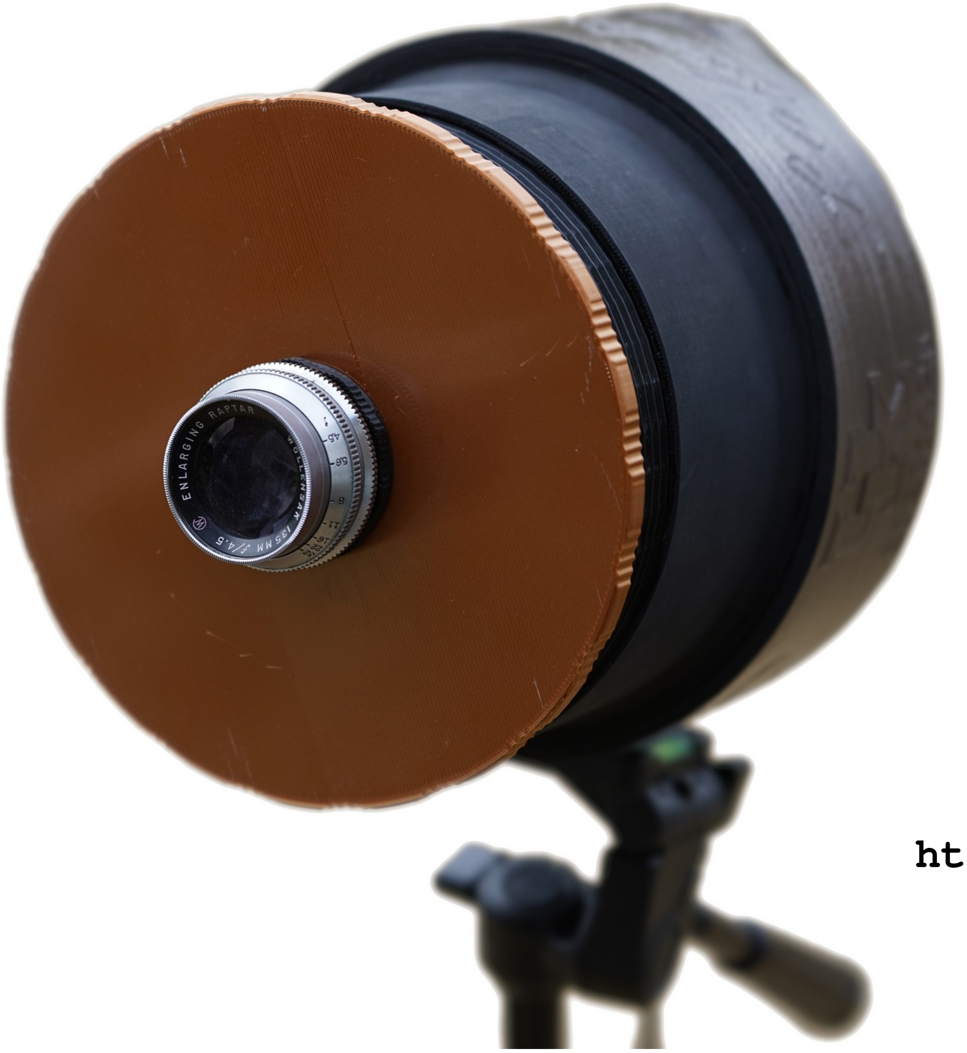
KISS-E **Build <$25** (without lens)

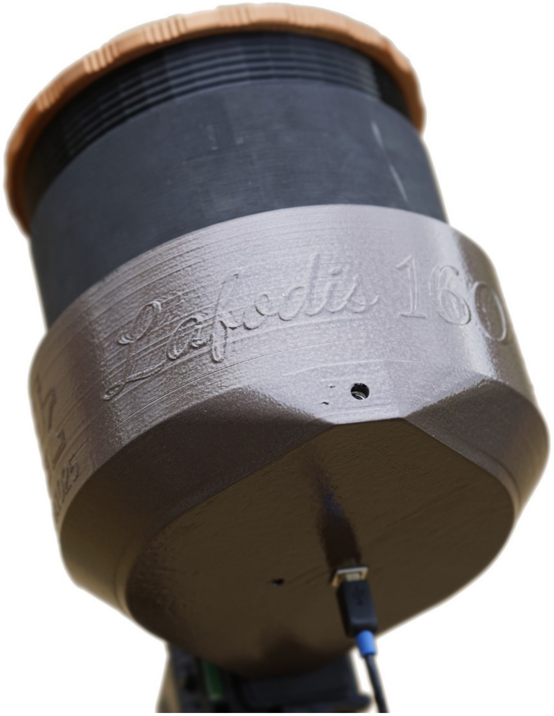




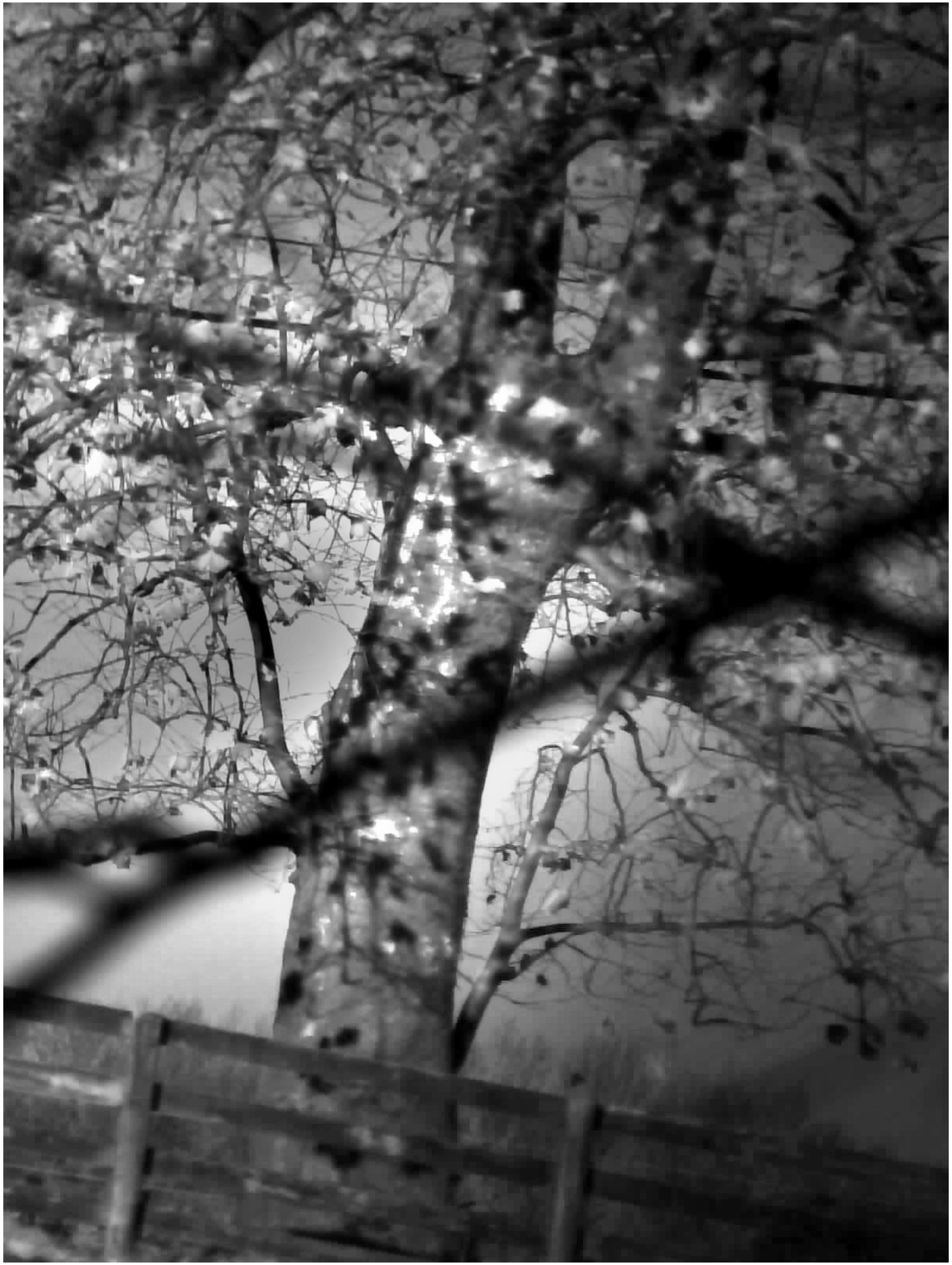
- $7 **ESP32-CAM**
- $3 **SSD1306 OLED**, 128x64
- $0.50 **Switch**, 12mm momentary SPST push button
- Power supply; either
  - $2.50 **FT232RL** with USB
  - $2 **CR123**/**16340** battery + $3 5V boost converter

Lafodis160: LArge FOrmat DIgital Scanning, 160mm coverage circle





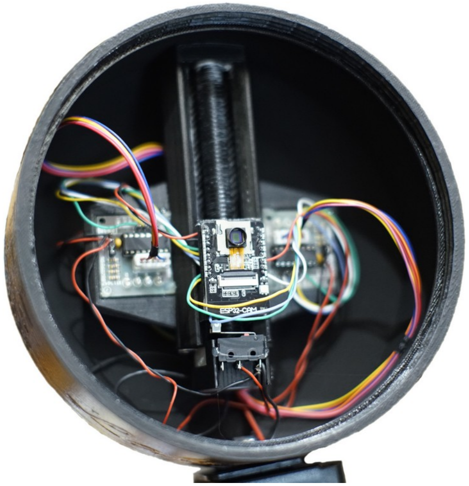
http://aggregate.org/DIT/Lafodis160

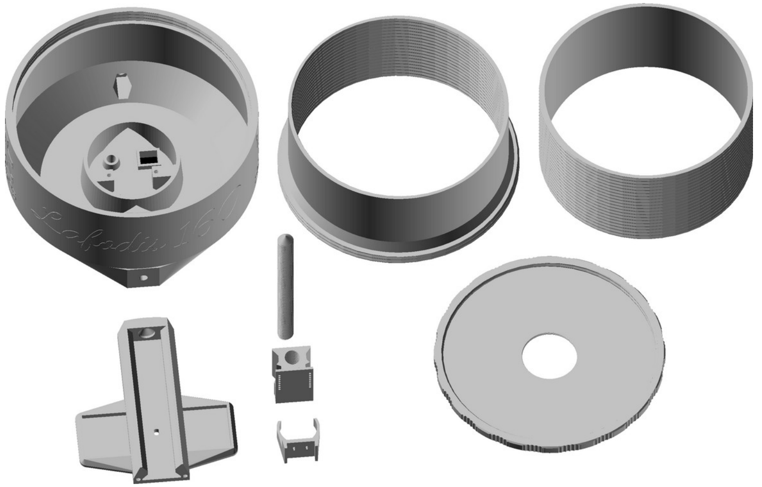


Sample B&W Capture

- One exposure, 1600x1200
- OV2640 JPEG
- Shallow DoF from 4x5 lens

Lafodis160 Build <$50 (without lens)

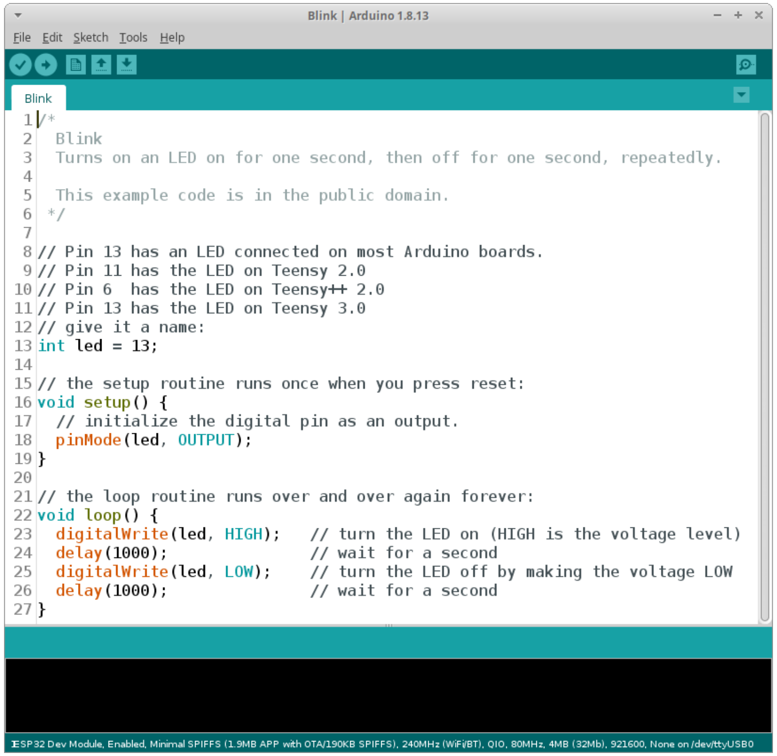




- **Scan resolution:** default **500MP @ 4x5 inch**; 2.6GP max
- **Dynamic range:** 8-10EV; HDR to 20EV
- **Color:** RGB CFA, no integrated NIR filter
- **Scan speed: currently <1MP/s**; theoretical peak ~10MP/s
- **Electronics: ESP32-CAM**, two **28BYJ-48 with ULN2003**
- **Capture control:** wireless via Bluetooth (it’s an IoT device!)
- **Firmware update:** wireless via 802.11 WiFi
- **Power:** 5V via USB connector from external source
- **Build equip.:** 180mm dia. x120mm tall 3D printer, wire wrap

Arduino IDE

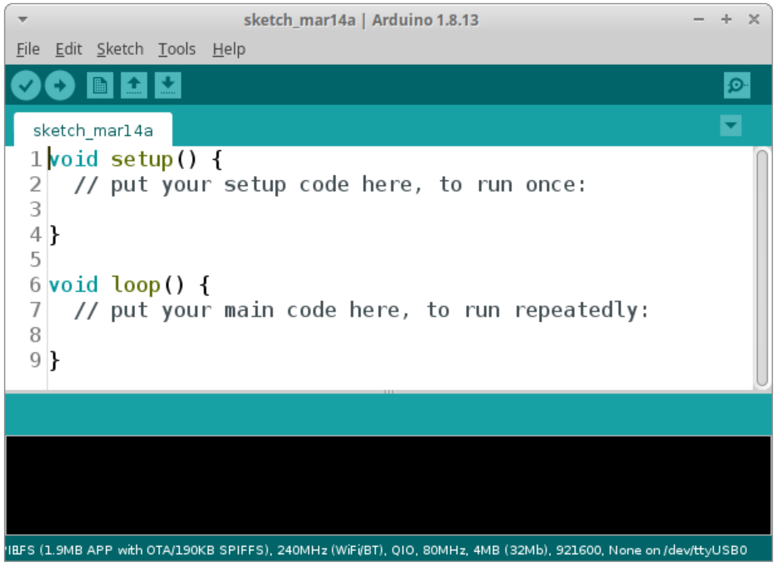
- Arduino started in 2005 at Interaction Design Institute Ivrea, Italy (aka, IDII)
- Open-source HW/SW
- Cross development IDE
  - C/C++ “sketches”
  - **`setup()`**
  - **`loop()`**



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// Pin 11 has the LED on Teensy 2.0
// Pin 6  has the LED on Teensy++ 2.0
// Pin 13 has the LED on Teensy 3.0
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

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

ESP32 Dev Module, Enabled, Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on /dev/ttyUSB0

Embedded Code Is Magic!



```
void setup() {
  // put your setup code here, to run once:

}

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

}
```

```
IBFS (1.9MB APP with OTA/190KB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on /dev/ttyUSB0
```

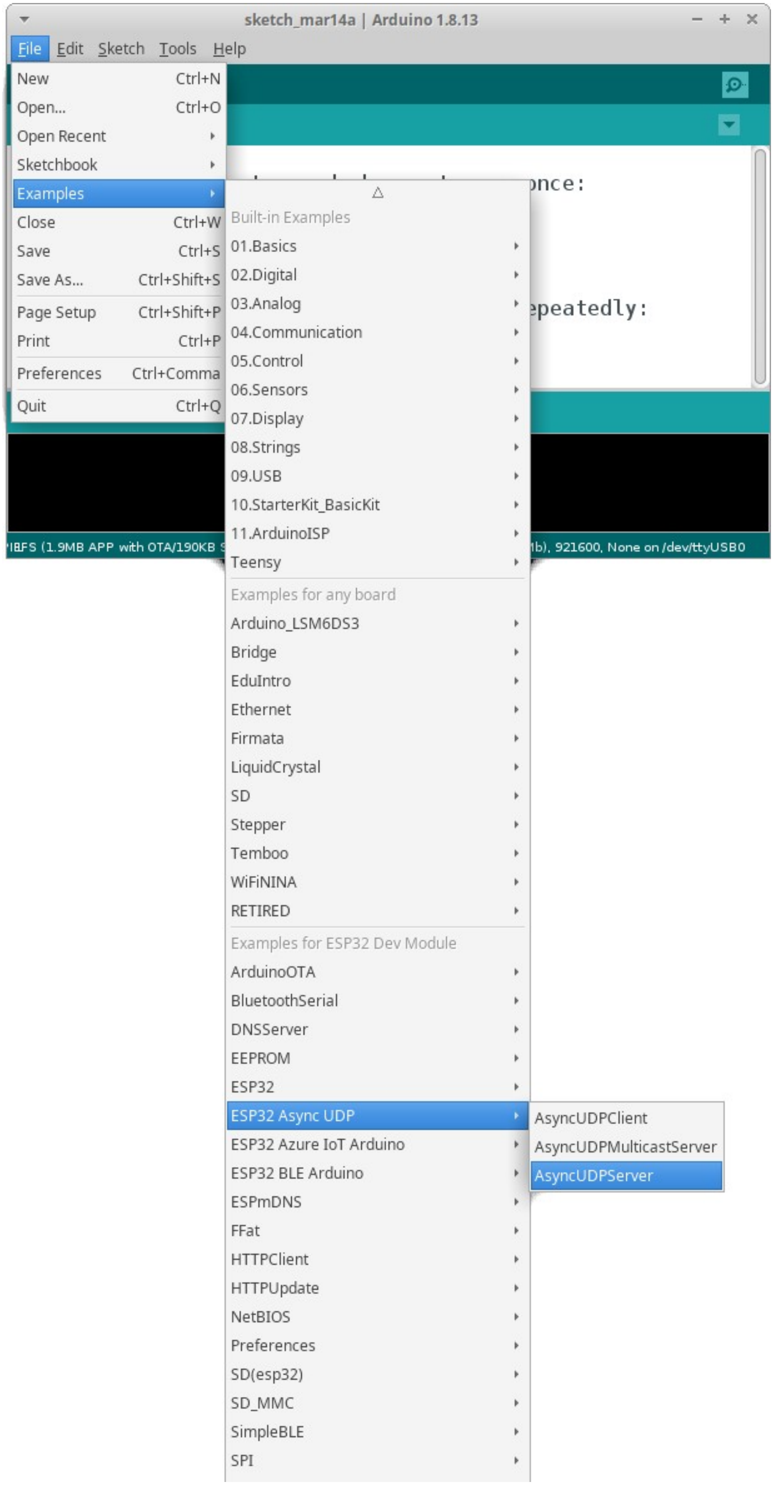
- Embedded systems are full of magic details…
  - **What pins?**
  - **Hardware registers**
  - **Real-time issues**

- **Difficult to get started**

- **Really difficult to debug** embedded systems

Embedded Code Is Magic!

- Embedded systems are full of magic details…
  - **What pins?**
  - Hardware registers
  - **Real-time issues**
- Open source **Libraries!**
- Open source **Examples!**
- Some **Debug support**

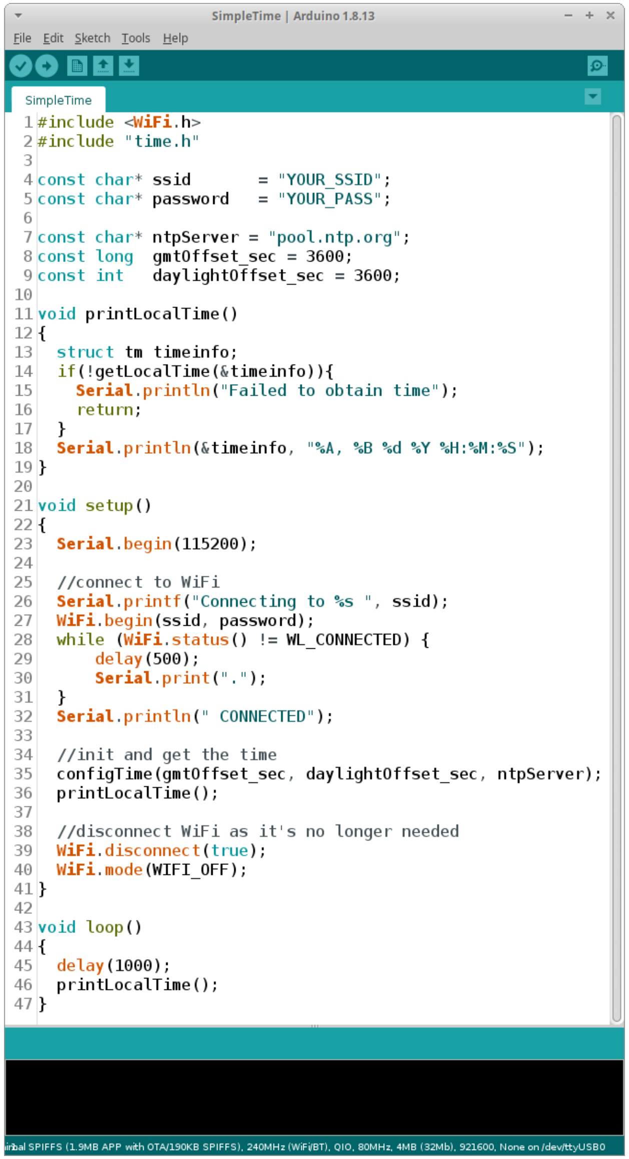


An Example: NTP-Set Clock

- Everything is open source
  - Example code
  - Libraries used

- Lots of magic hidden using C++ classes & overloading… e.g., `Serial.println()`

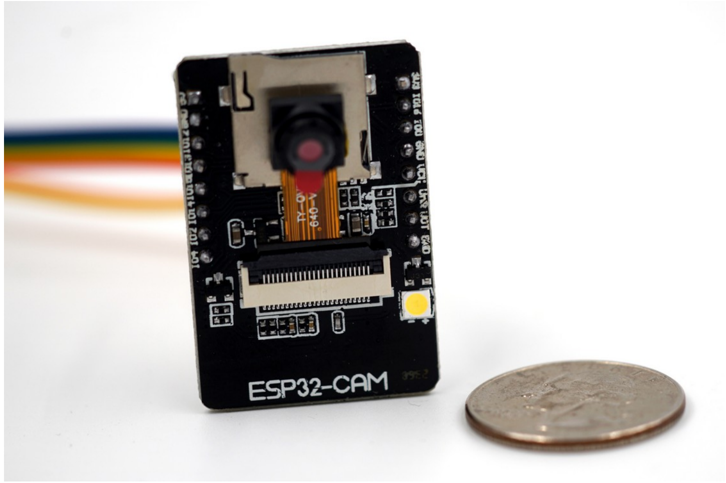
- IDE has some ICE-like features
  - USB programming / debug
  - OTA programming… Over The Air via 802.11

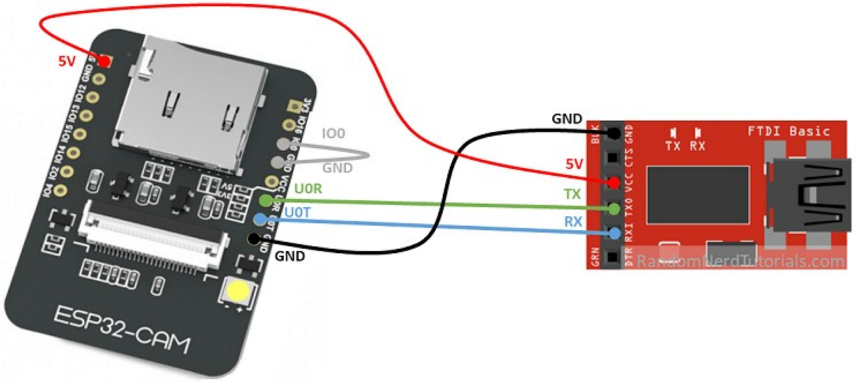


```
#include <WiFi.h>
#include "time.h"

const char* ssid       = "YOUR_SSID";
const char* password   = "YOUR_PASS";

const char* ntpServer = "pool.ntp.org";
const long  gmtOffset_sec = 3600;
const int   daylightOffset_sec = 3600;

void printLocalTime()
{
  struct tm timeinfo;
  if(!getLocalTime(&timeinfo)){
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}

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

  //connect to WiFi
  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      Serial.print(".");
  }
  Serial.println(" CONNECTED");

  //init and get the time
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  printLocalTime();

  //disconnect WiFi as it's no longer needed
  WiFi.disconnect(true);
  WiFi.mode(WIFI_OFF);
}

void loop()
{
  delay(1000);
  printLocalTime();
}
```

imal SPIFFS (1.9MB APP with OTA/190KB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on /dev/ttyUSB0

Arduinos Are Wimpy…?





- **ESP32-CAM isn't wimpy**… nor is it an Arduino
- **Need USB-TTL adapter**, e.g., CP2102 or FT232RL (Future Technology Devices Inc. chip – FTDI)
- Arduino IDE needs add-on for ESP32:

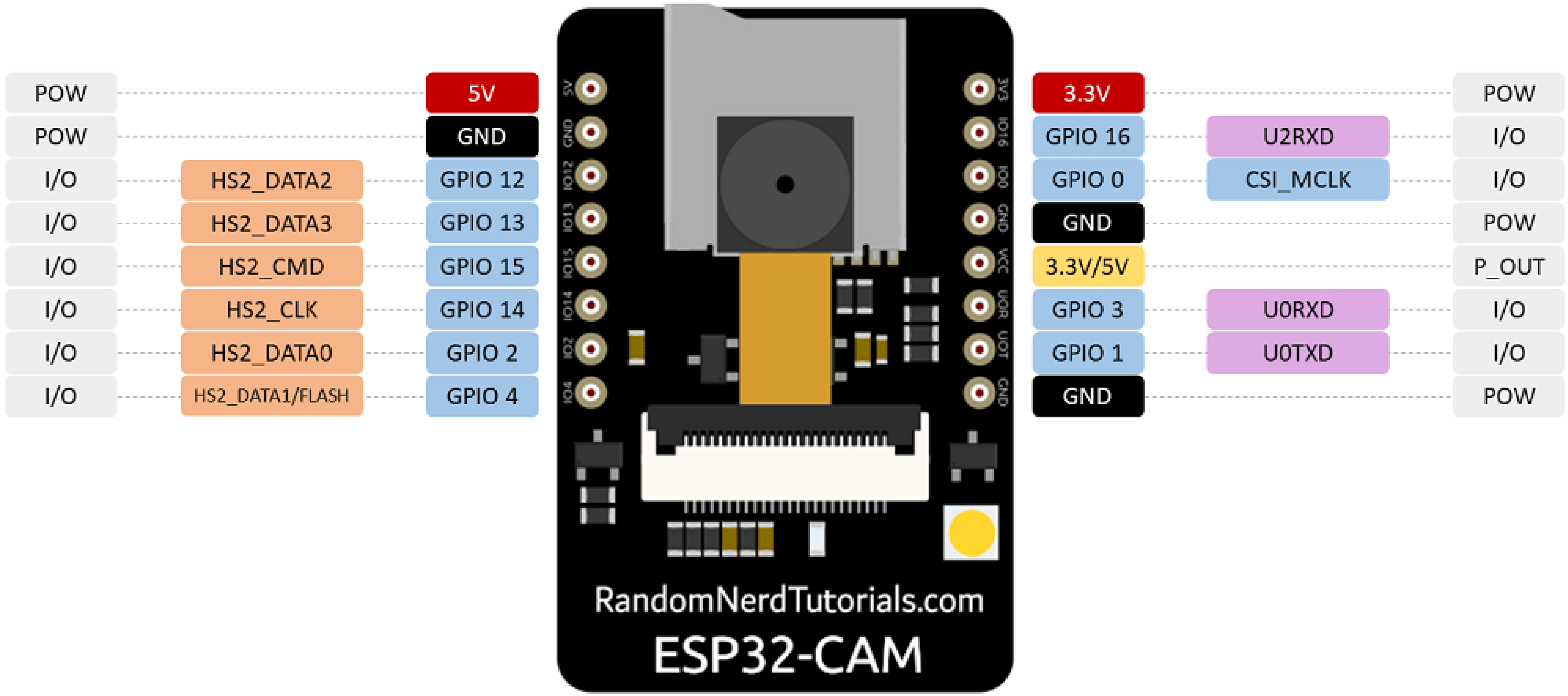
  https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

  http://arduino-er.blogspot.com/2020/06/install-esp32esp8266-to-arduino-ide-on.html

AI-Thinker ESP32-CAM

- Compute (180-310mA, **5µA deep sleep**)
  - **240MHz** ESPRESSIF **32-bit dual-core** LX6
  - Internal **520KB SRAM**, external **4MB PSRAM**
  - **4MB** flash memory; up to **4GB** (32GB?) **TF card**
  - **Crypto HW** for RNG, ECC, RSA, SHA-2, AES
- I/O facilities (including **antenna**)
  - **802.11 b/g/n/e/i**
  - **Bluetooth v4.2 BR/EDR** and **BLE**
  - SPI, I2C, MMC, ADC, PWM…
- Camera (and **white LED light** on board)
  - Omnivision OV2640, **1600x1200 native RGB**
  - 2.2µm square pixels, 10-bit ADC, **HW JPEGs**
  - **Removable lens**

AI-Thinker ESP32-CAM I/O Pins

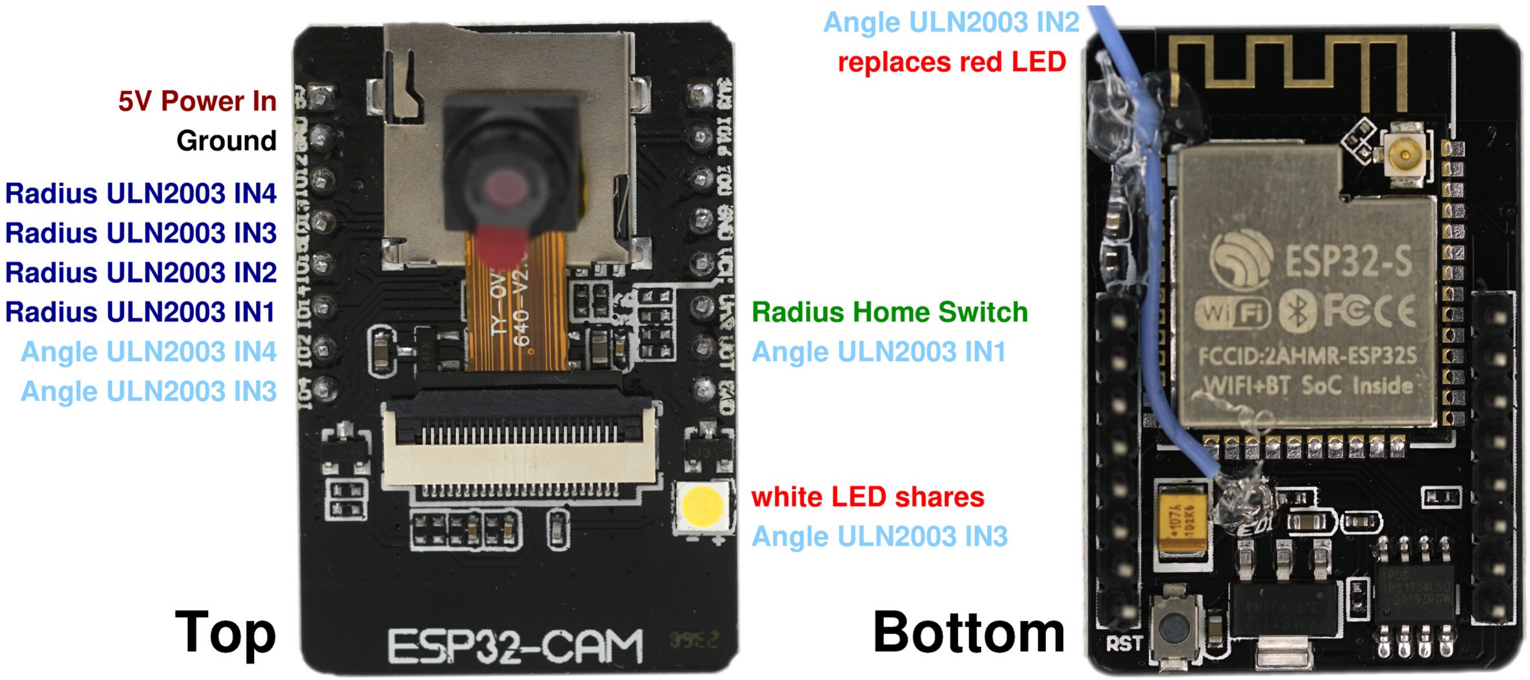


POW		5V	3.3V		POW
POW		GND	GPIO 16	U2RXD	I/O
I/O	HS2_DATA2	GPIO 12	GPIO 0	CSI_MCLK	I/O
I/O	HS2_DATA3	GPIO 13	GND		POW
I/O	HS2_CMD	GPIO 15	3.3V/5V		P_OUT
I/O	HS2_CLK	GPIO 14	GPIO 3	U0RXD	I/O
I/O	HS2_DATA0	GPIO 2	GPIO 1	U0TXD	I/O
I/O	HS2_DATA1/FLASH	GPIO 4	GND		POW

- Pins are **wildly overloaded** with different functions

https://github.com/raphaelbs/esp32-cam-ai-thinker/blob/master/docs/esp32cam-pin-notes.md

How I Got 4+4 Output Pins



5V Power In

Ground

Radius ULN2003 IN4

Radius ULN2003 IN3

Radius ULN2003 IN2

Radius ULN2003 IN1

Angle ULN2003 IN4

Angle ULN2003 IN3

Top

Angle ULN2003 IN2
replaces red LED

Radius Home Switch

Angle ULN2003 IN1

white LED shares
Angle ULN2003 IN3

Bottom

- Overloading: I use Pin 4 as Angle ULN2003 IN3… but it also controls the white LED and TF card
- Desperation: removed red LED and used that for my Angle ULN2003 IN2 signal

More Annoyances

- Can run off 3.3V, but **runs better off 5V** – especially when reprogramming the part
- Brownout detector is a little too aggressive:

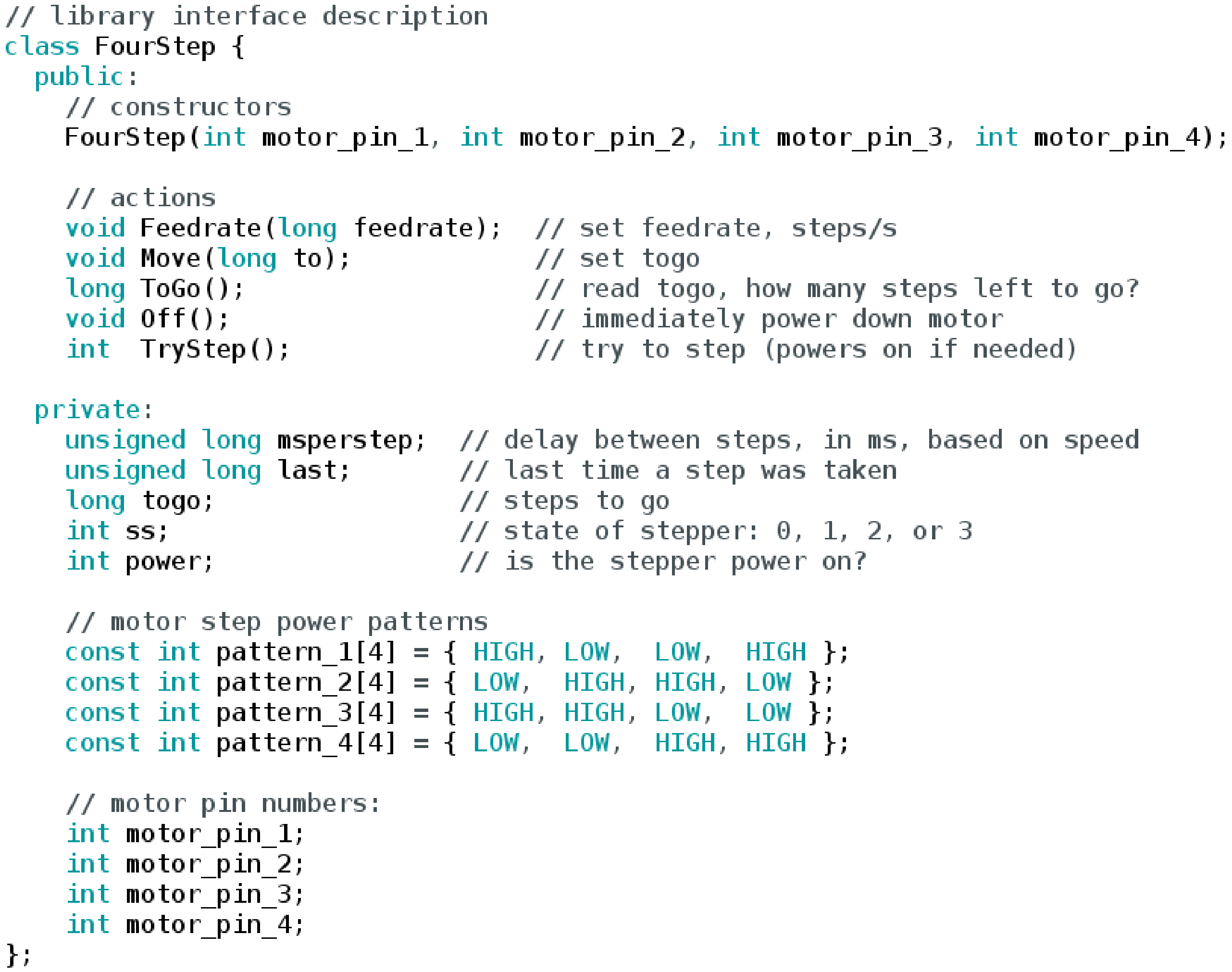
  ```
  #include "soc/soc.h"
  #include "soc/rtc_cntl_reg.h"
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  ```

- Some pins need to be in certain states to boot...
- OTA requires space; “`ESP32 Dev Module`” with: `1.9MB APP with OTA, 190K SPIFFS`
- PSRAM is too slow for large OV2640 images; use JPEG and `img_converters.h` for RGB888

Still, It’s Easy To Use

- The key is to **fully leverage source code for libraries and examples**…
- For Lafodis160, I needed to drive 2 stepper motors (that’s what those 8 output pins were needed for)
  - There are **several libraries, with examples, for driving steppers**… all **open source**
  - None of those was usable because they **always left power on**… including the white LED!
  - Looking at the source, it was easy to understand how the steppers had to be controlled and thus I **wrote my own library that is more efficient** and **implements power management**

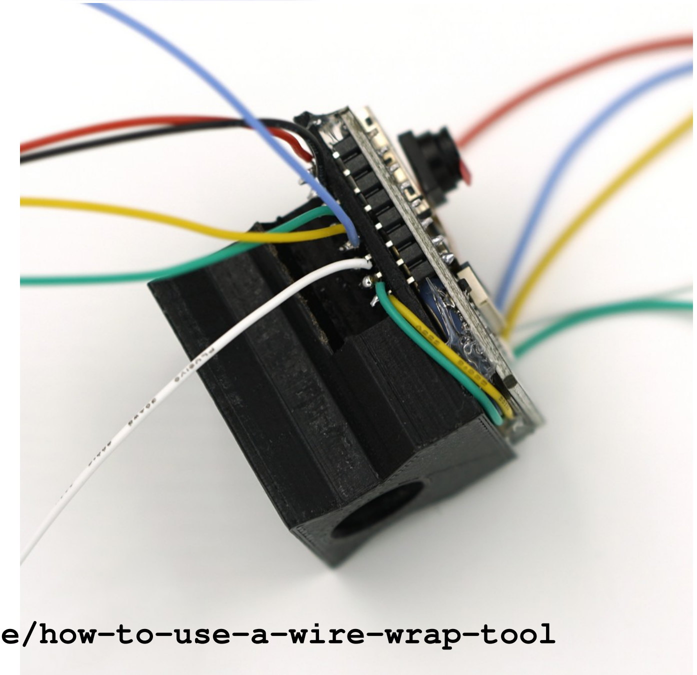
New Stepper Library



```
// library interface description
class FourStep {
  public:
    // constructors
    FourStep(int motor_pin_1, int motor_pin_2, int motor_pin_3, int motor_pin_4);

    // actions
    void Feedrate(long feedrate);  // set feedrate, steps/s
    void Move(long to);            // set togo
    long ToGo();                   // read togo, how many steps left to go?
    void Off();                    // immediately power down motor
    int  TryStep();                // try to step (powers on if needed)

  private:
    unsigned long msperstep;  // delay between steps, in ms, based on speed
    unsigned long last;       // last time a step was taken
    long togo;                // steps to go
    int ss;                   // state of stepper: 0, 1, 2, or 3
    int power;                // is the stepper power on?

    // motor step power patterns
    const int pattern_1[4] = { HIGH, LOW,  LOW,  HIGH };
    const int pattern_2[4] = { LOW,  HIGH, HIGH, LOW };
    const int pattern_3[4] = { HIGH, HIGH, LOW,  LOW };
    const int pattern_4[4] = { LOW,  LOW,  HIGH, HIGH };

    // motor pin numbers:
    int motor_pin_1;
    int motor_pin_2;
    int motor_pin_3;
    int motor_pin_4;
};
```

3D Printing & Wiring Tricks

- ESP32-CAM usually comes with pins installed in the board
- Can **remove pins** & **solder**
- Can use **wire wrap**: a good overview is at
  - 3D-print **cavity for board**
  - 3D-print **traceless PCB** as part of 3D design
- Don't know how to wire-wrap?



`https://learn.sparkfun.com/tutorials/working-with-wire/how-to-use-a-wire-wrap-tool`

Conclusion

- **AI Thinker ESP32-CAM** is remarkably versatile
  - More flexible than Canon PowerShots using CHDK
  - Ignoring the camera, a powerful embedded controller
- I'm posting helpful hints at:

  `http://aggregate.org/DIT/ESP32CAM`
- **Even if you hate IoT**, **HW built to support it is cool**
  - Small, cheap, surprisingly powerful, and versatile
  - You might not even need to solder (**wire wrap!**)
  - **Arduino community** provides the necessary magic