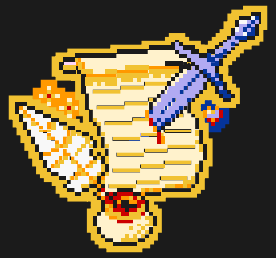
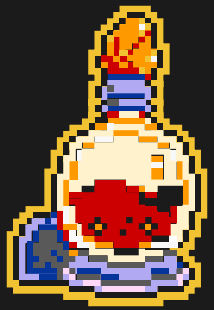


Crash Course:
IoT and Hardware Hacking

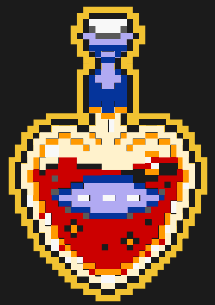
Road to your first CVE! ;P

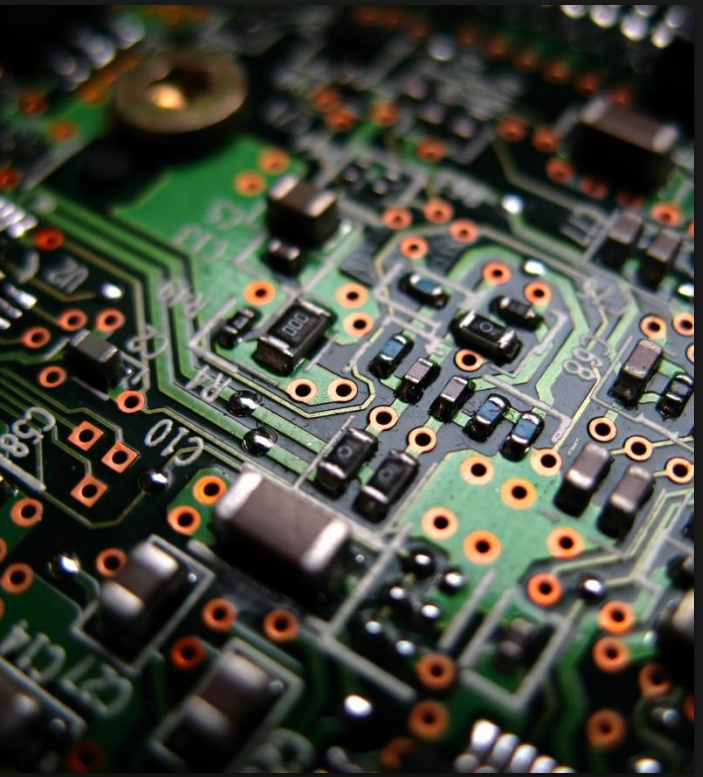
Malaysia CyberSecurity Camp 2024



01

Electronic Basic



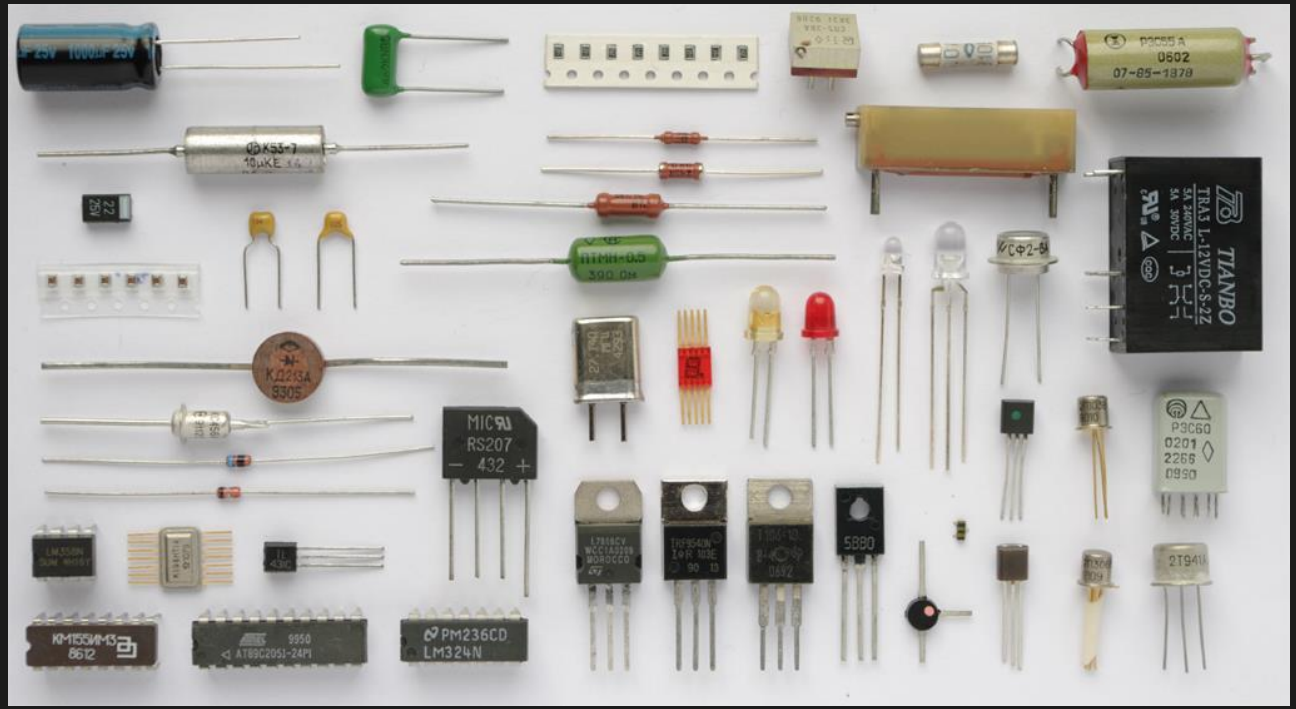


What You'll Learn

- Electronic Component
- Electronic Symbol
- Read electronic Schematic
- Tools you need
- Datasheet.

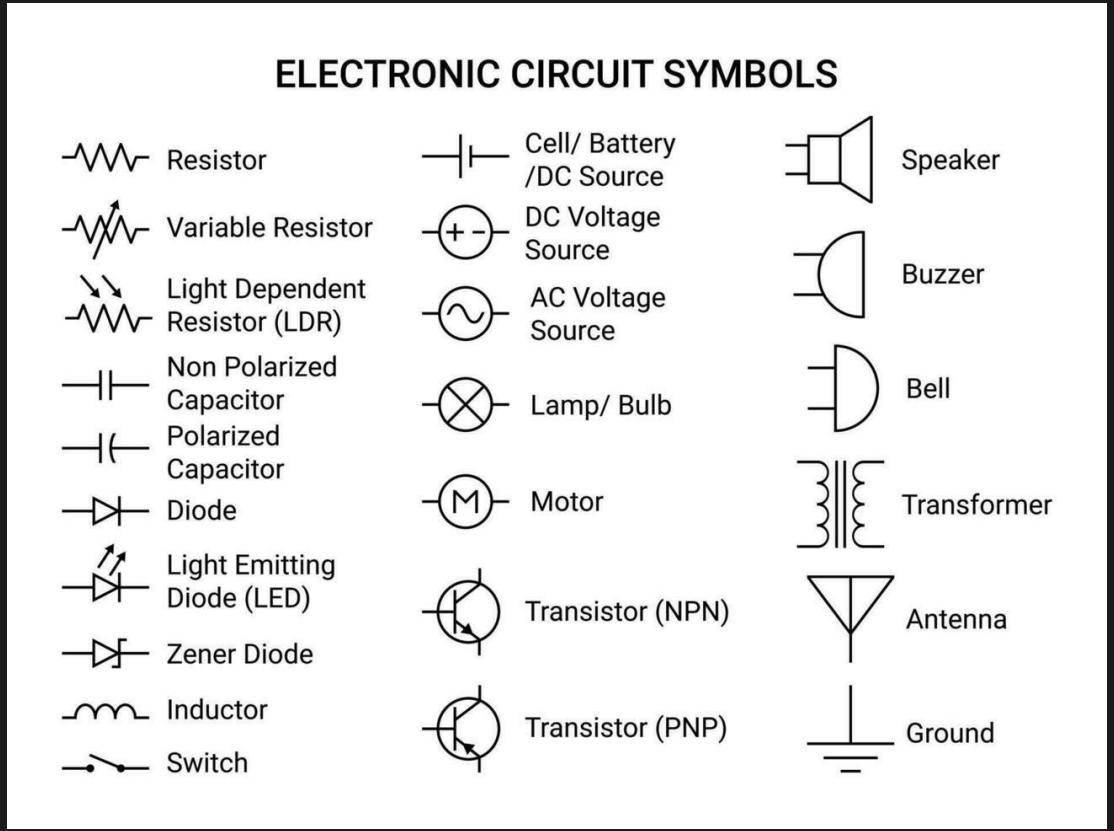
Hardware Basics

Electronic Components:



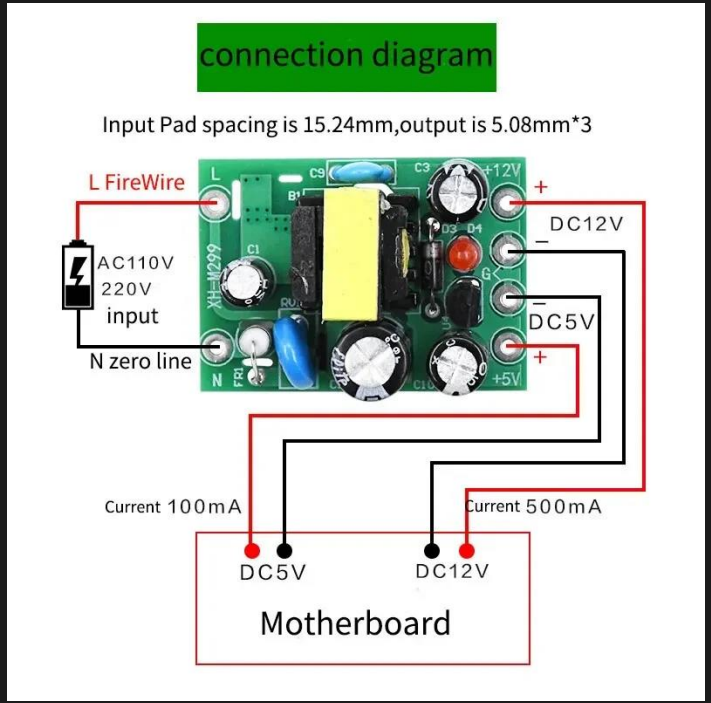
Hardware Basics

Electronic Symbol:



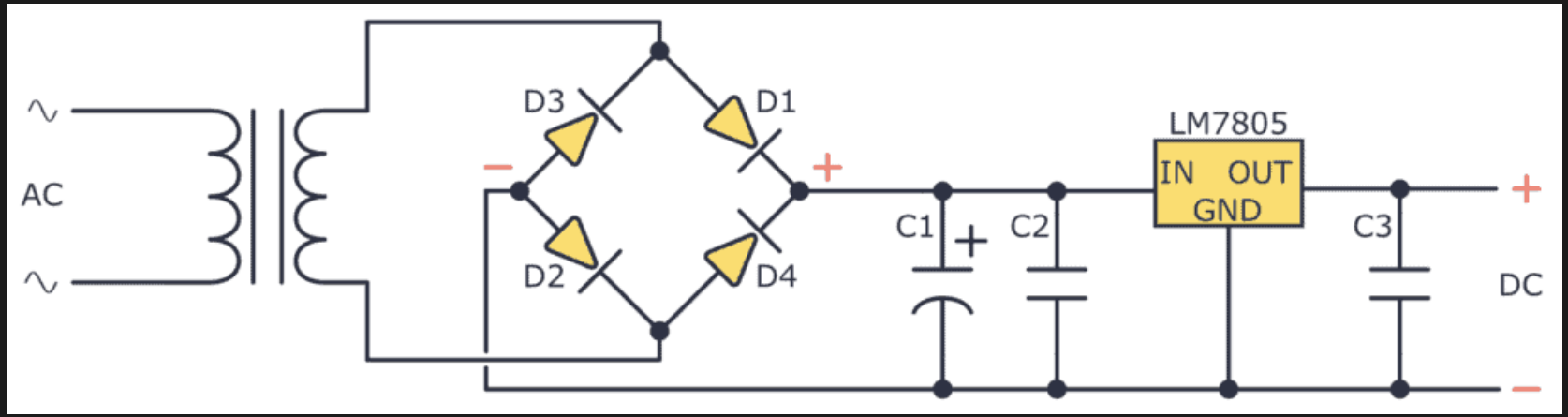
Hardware Basics

Power Supply Circuit:



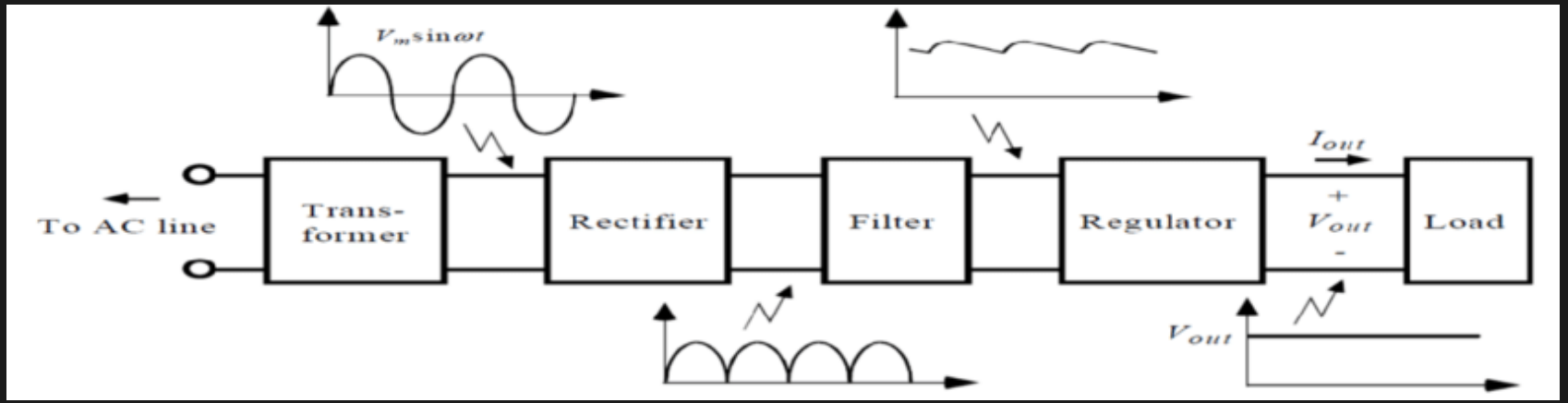
Hardware Basics

Power Supply Schematic:



Hardware Basics

Block Diagram:

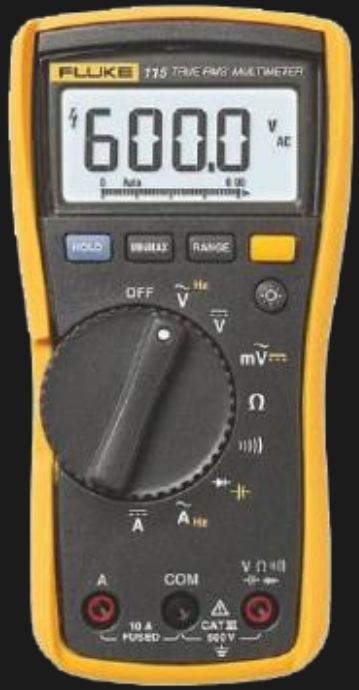


Hardware Basics

Tools - Soldering iron:

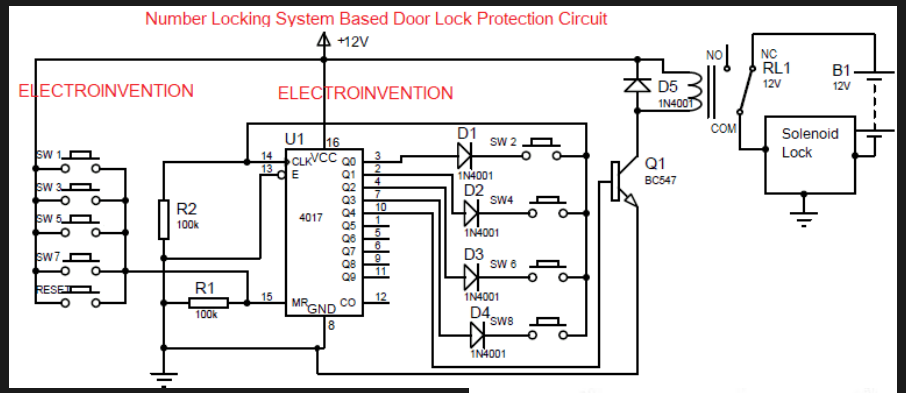
Hardware Basics

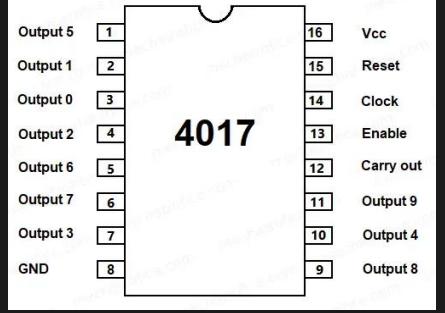
Multimeter :



Hardware Basics

Datasheet:





FAIRCHILD SEMICONDUCTOR™

October 1987
Revised January 1999

CD4017BC · CD4022BC
Decade Counter/Divider with 10 Decoded Outputs · Divide-by-8 Counter/Divider with 8 Decoded Outputs

CD4017BC · CD4022BC Decade Counter/Divider with 10 Decoded Outputs · Divide-by-8 Counter/Divider with 8 Decoded Outputs

General Description

The CD4017BC is a 5-stage divide-by-10 Johnson counter with 10 decoded outputs and a carry out bit.

The CD4022BC is a 4-stage divide-by-8 Johnson counter with 8 decoded outputs and a carry-out bit.

These counters are cleared to their zero count by a logical "1" on their reset line. These counters are advanced on the positive edge of the clock signal when the clock enable signal is in the logical "0" state.

The configuration of the CD4017BC and CD4022BC permits medium speed operation and assures a hazard free counting sequence. The 10/8 decoded outputs are normally in the logical "0" state and go to the logical "1" state only at their respective time slot. Each decoded output remains high for 1 full clock cycle. The carry-out signal completes a full cycle for every 10/8 clock input cycles and is used as a ripple carry signal to any succeeding stages.

Features

- Wide supply voltage range: 3.0V to 15V
- High noise immunity: 0.45 V_{DD} (typ.)
- Low power Fan out of 2 driving 74L
  TTL compatibility: or 1 driving 74LS
- Medium speed operation: 5.0 MHz (typ.)
  with 10V V_{DD}
- Low power: 10 μW (typ.)
- Fully static operation

Applications

- Automotive
- Instrumentation
- Medical electronics
- Alarm systems
- Industrial electronics
- Remote metering

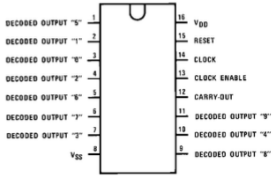
Ordering Code:

Order Number	Package Number	Package Description
CD4017BCM	M16A	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-012, 0.150" Narrow
CD4017BCSJ	M16D	16-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
CD4017BCN	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide
CD4022BCM	M16A	16-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-012, 0.150" Narrow
CD4022BCN	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

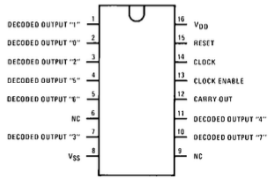
Connection Diagrams

Pin Assignments for DIP, SOIC and SOP
CD4017B



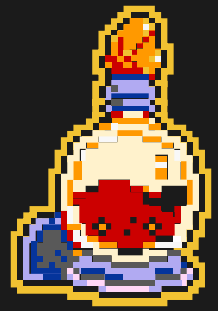
Top View

Pin Assignments for DIP and SOIC
CD4022B



Top View

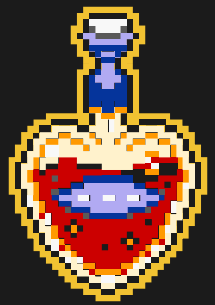
 DS005950.prf www.fairchildsemi.com



02

Internet of Things

IoT



IoT

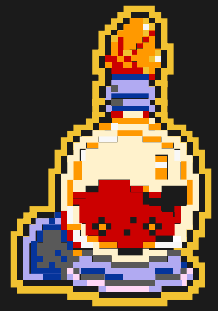
- **IoT (Internet of Things)** refers to a network of physical objects or devices that are embedded with sensors, software, and other technologies, enabling them to connect and exchange data with each other and with systems over the internet. These "things" can range from everyday household items like smart lights and thermostats to industrial tools, vehicles, and even medical devices.

IoT

- **Connectivity**: Devices are connected to the internet and communicate with e other.
- **Sensors**: They collect data from their environment (e.g., temperature, motion humidity).
- **Automation**: IoT systems can perform tasks automatically based on predefine rules or real-time data.
- **Data Exchange**: Data collected by devices is analyzed to optimize operations, improve efficiency, or provide insights.
- **Remote Control**: Devices can be controlled from anywhere using a smartpho or computer.

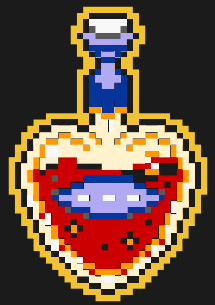
IoT Controller

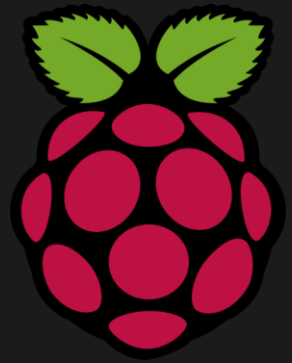
- Arduino uno / ESP32 / ESP8266
- Raspberry Pi.
- NodeMCU
- PLC
- LoRaWAN Controller
- Alexa/Google Next Hub.



03

Raspberry Pi





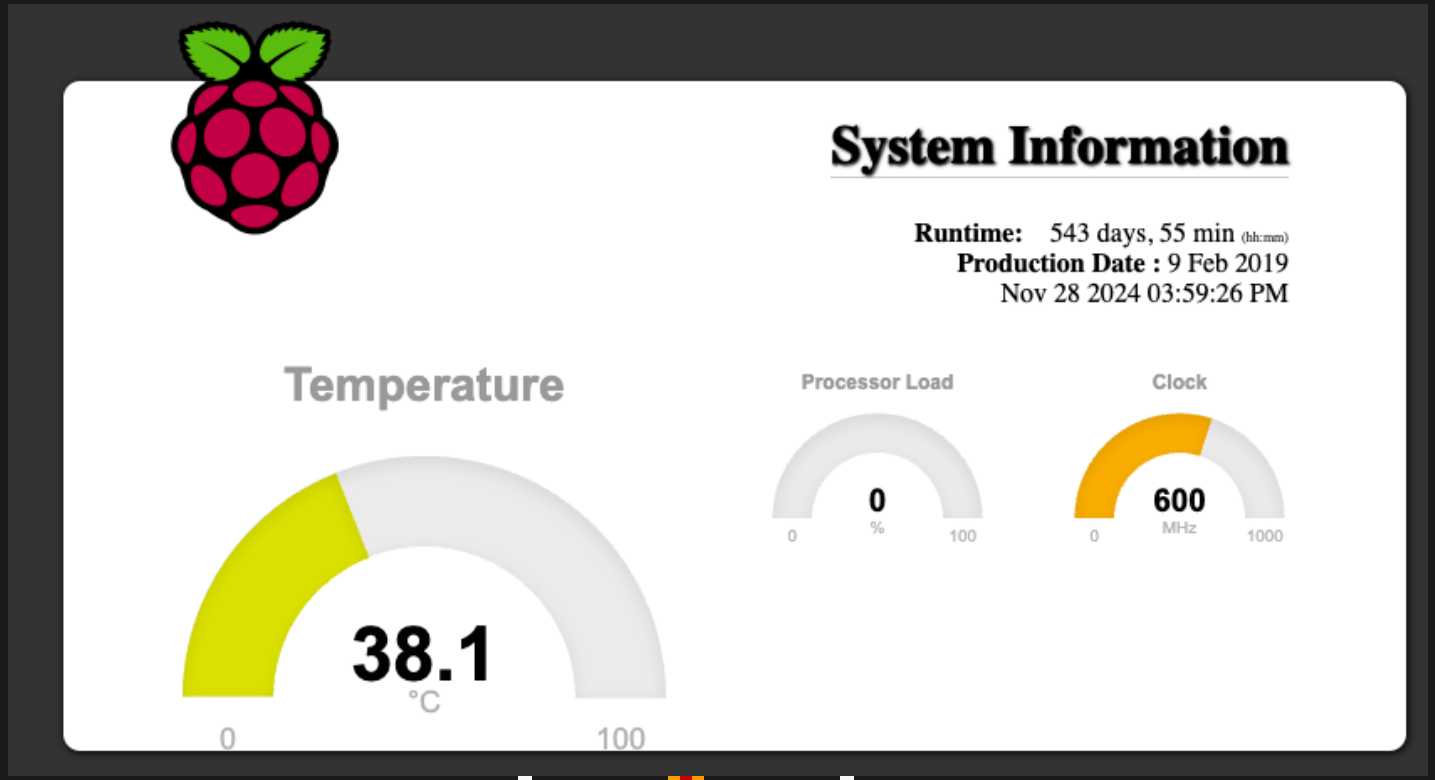
Raspberry Pi

What Is Raspberry Pi:

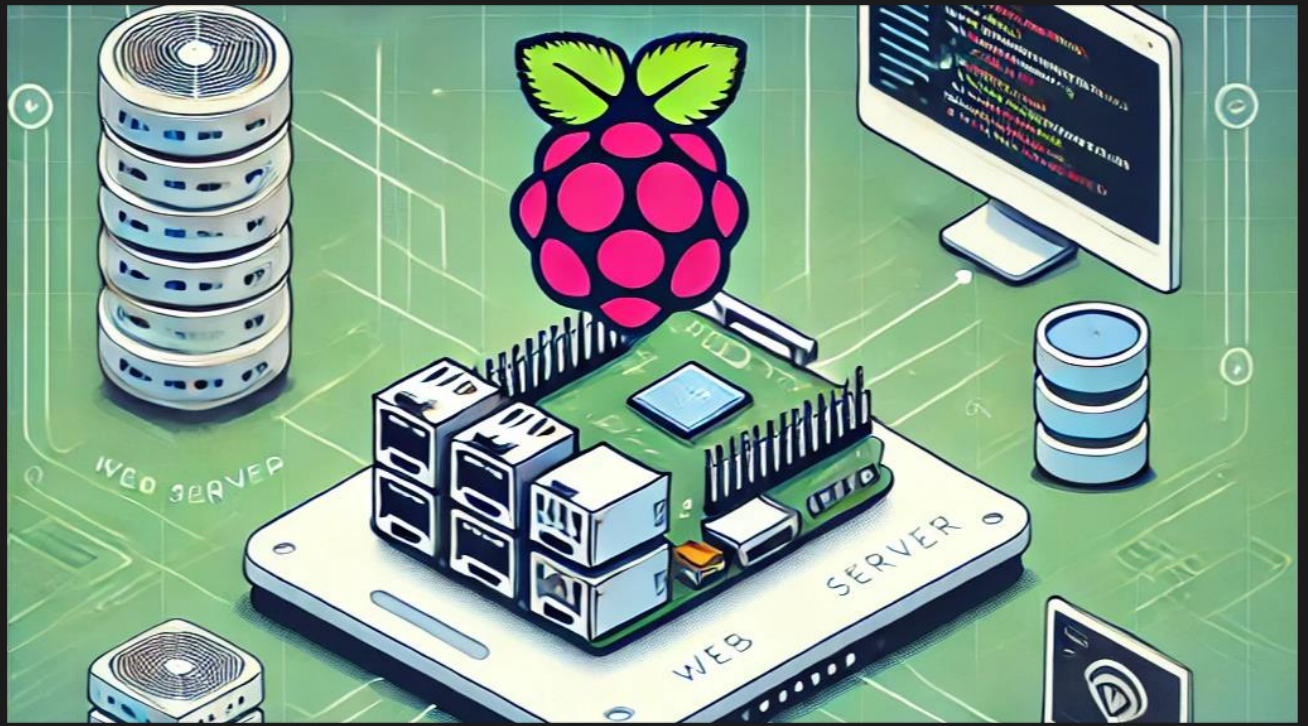
A **small, affordable single-board computer** developed by the Raspberry Pi Foundation, primarily used for learning programming, DIY electronics projects, and as a versatile embedded computer.

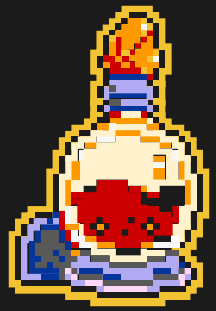
- **Compact Size**: Small enough to fit in your pocket.
- **Low Cost**: Budget-friendly, suitable for hobbyists and educators.
- **GPIO Pins**: Allows interaction with sensors, LEDs, motors, and more.
- **Customizable**: Runs various operating systems, mainly Linux-based (e.g., Raspberry Pi OS).

Raspberry Pi



Raspberry Pi

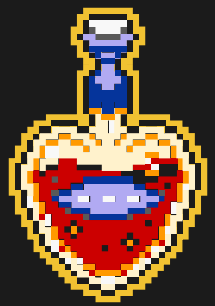




04

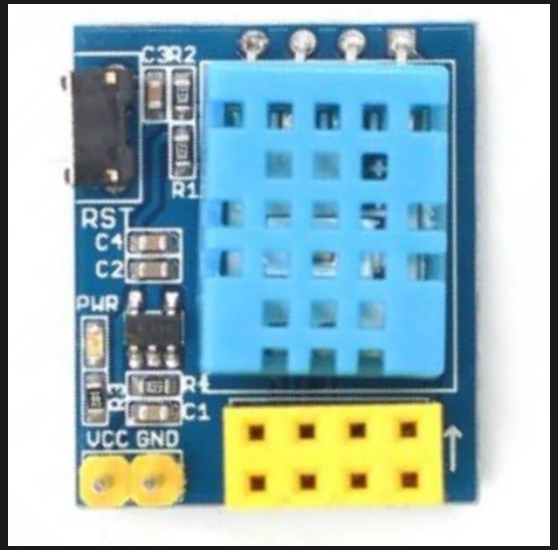
ESP01S with DHT11 Module

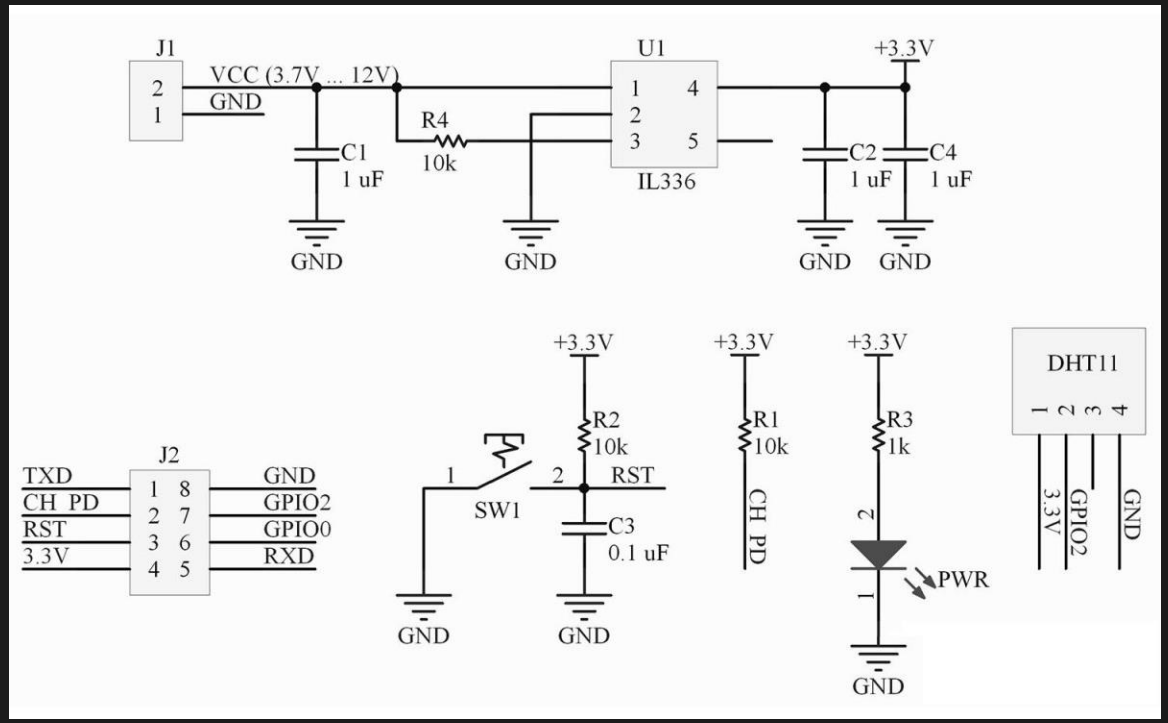
ESP8266



ESP01S With DHT11 Module

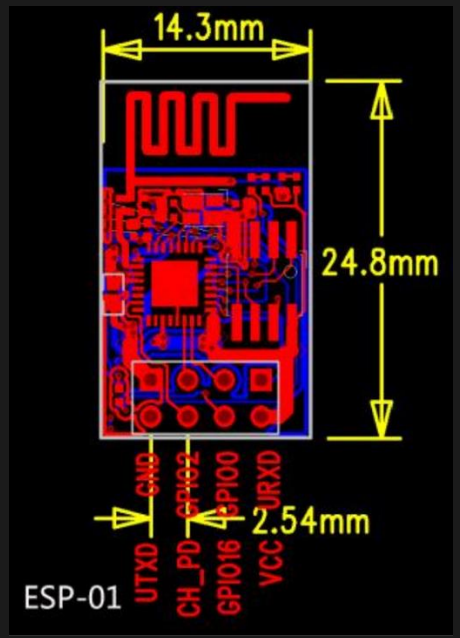
DHT 11 Module:

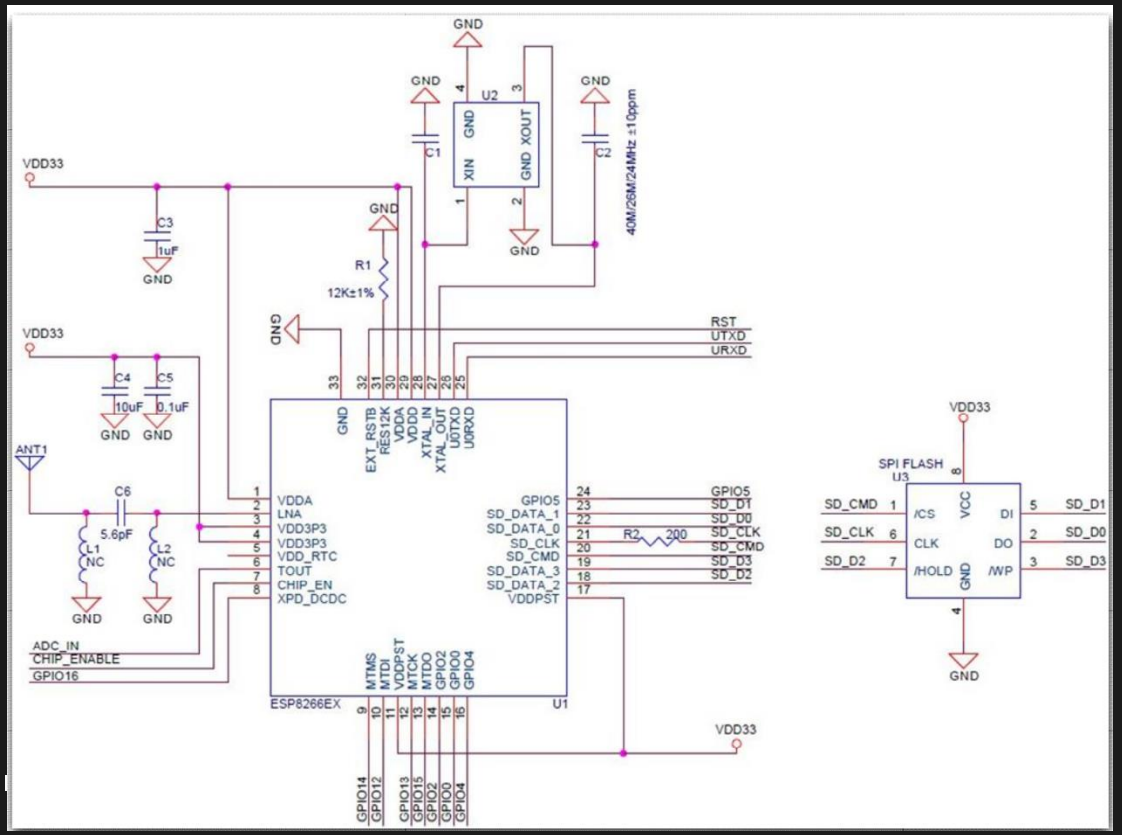


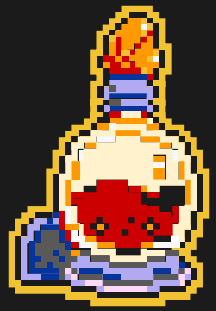


ESP01S With DHT11 Module

ESP01 Module:

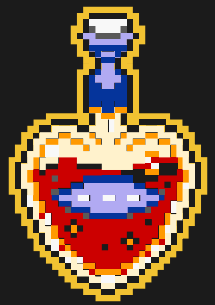






05

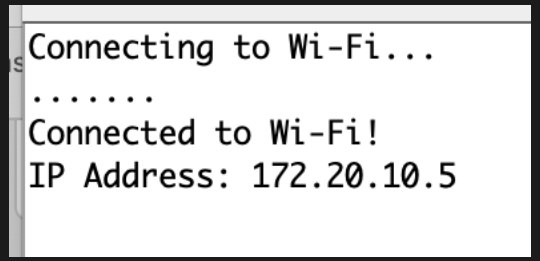
LAB



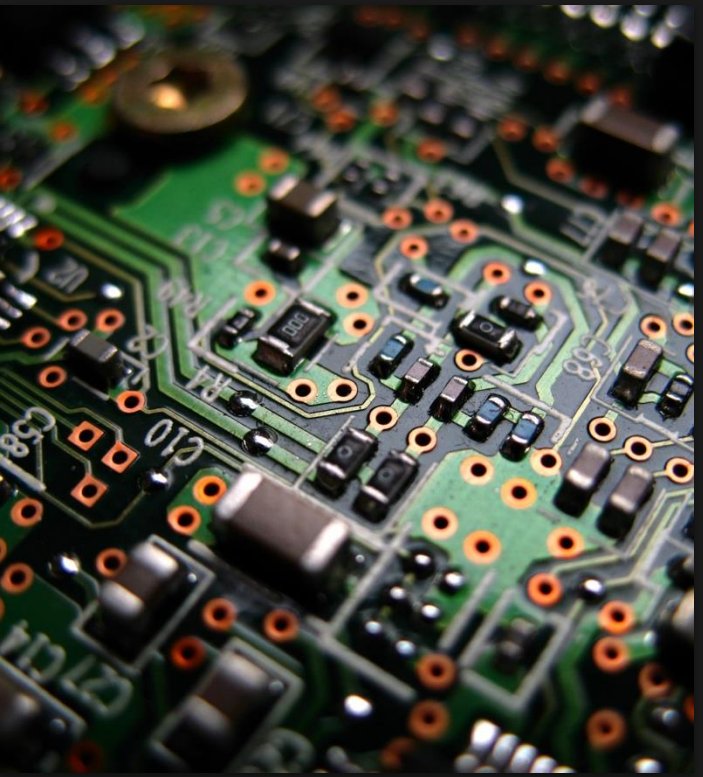
ESP8266 Lab

```
#include <ESP8266WiFi.h>

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);
  delay(2000);

  // Display MAC Address
  String macAddress = WiFi.macAddress();
  Serial.println("ESP-01 MAC Address:");
  Serial.println(macAddress);
}

void loop() {
  // Nothing to do here
}
```

ESP8266 Lab

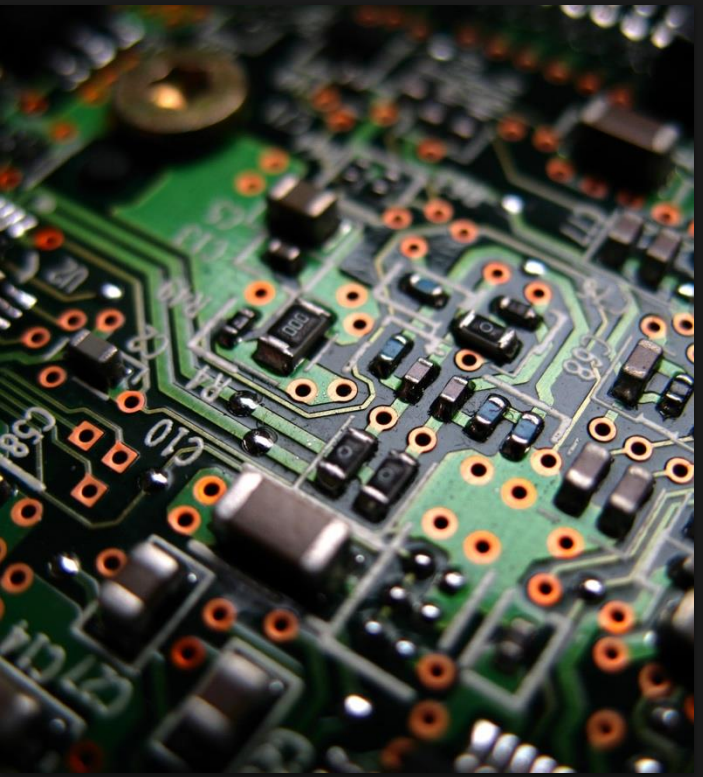



```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

// Replace with your Wi-Fi credentials
const char* ssid = "SSID";      // Wi-Fi network name (SSID)
const char* password = "PASSWORD"; // Wi-Fi network password

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);
  delay(2000);

  // Start Wi-Fi connection
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);

  // Wait until the ESP connects to Wi-Fi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Connection successful
  Serial.println("\nConnected to Wi-Fi!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP()); // Print the assigned IP address
}
```



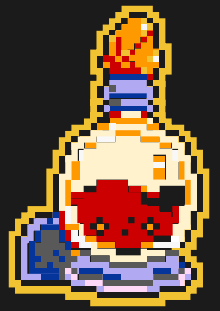
What You'll Learn

- Basic hardware components and their functions
- Pin identification and signal tracing
- Common IoT communication protocols
- Firmware extraction and analysis
- Safe and ethical hardware hacking practices



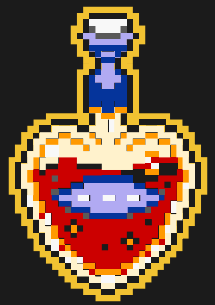
Prerequisites

- Basic understanding of electronics
- Familiarity with Linux commands
- Programming basics
- Willingness to learn and experiment safely



06

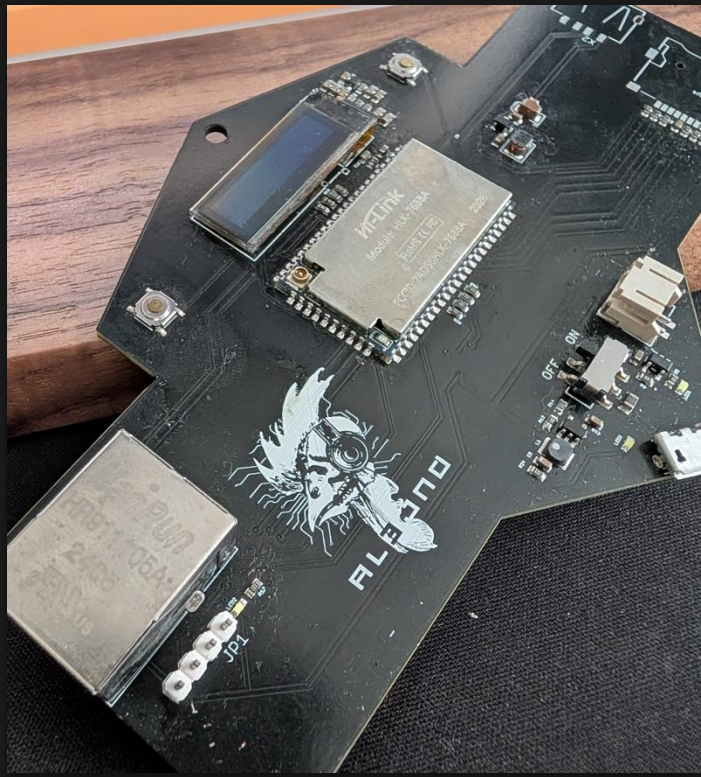
Hardware Basics

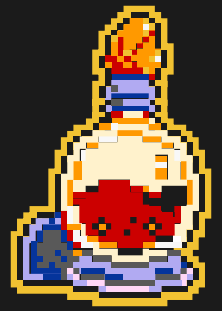


Hardware Basics

Common Components:

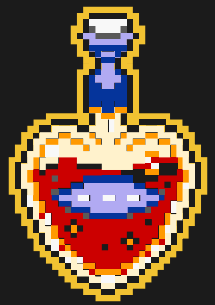
Component	Function	Examples
Microcontrollers (MCU)	Main processing unit	ESP32, ARM
Memory chips (Flash, RAM)	Store firmware/data	W25Q32, MX25L
EEPROM	Configuration storage	24C02, AT24C256
Voltage Register	Power management	AMS1117, LM317
Input/Output	Input/Output ports	HDMI, Ethernet, LED

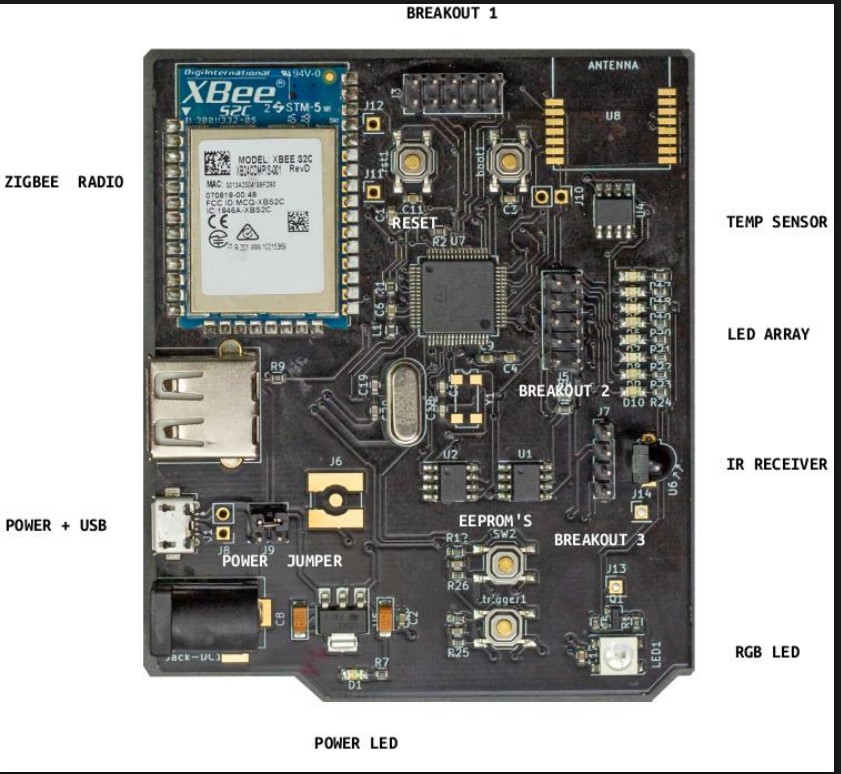




07

Understanding Pins





Pin Types

Type	Pins Required	Typical Voltage
UART/Serial	TX, RX	3.3V/5V
SPI	MOSI, MISO, SCK	3.3V
I2C	SDA, SCL	3.3V
GPIO	Input, Output	0V or VCC

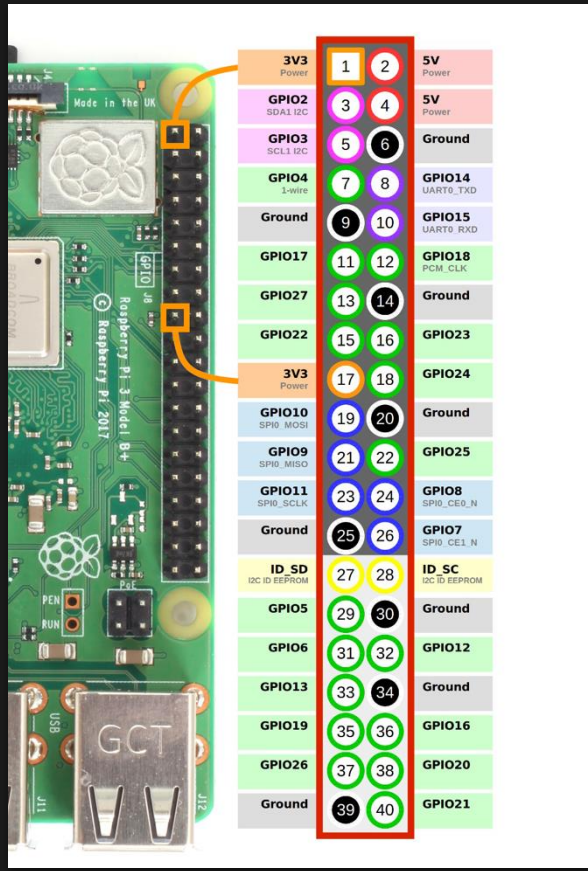
GPIO Pins

Basic GPIO Functions:

- Digital Input: Read logic levels (0V or VCC)
- Digital Output: Set logic levels (0V or VCC)
- Configurable pull-up/pull-down resistors

Example code

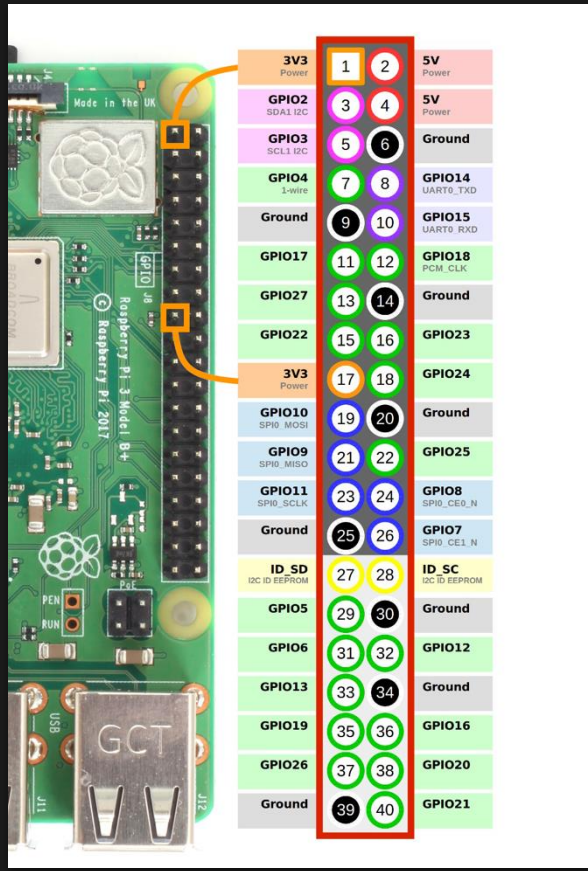
```
import RPi.GPIO as GPIO

# Setup
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)  # Set pin 18 as output

# Output example
GPIO.output(18, GPIO.HIGH)  # Set pin high
GPIO.output(18, GPIO.LOW)  # Set pin low

# Input example
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)# Input with pull-up
value = GPIO.input(17)  # Read input
```



GPIO Pins

- Advanced GPIO Features:

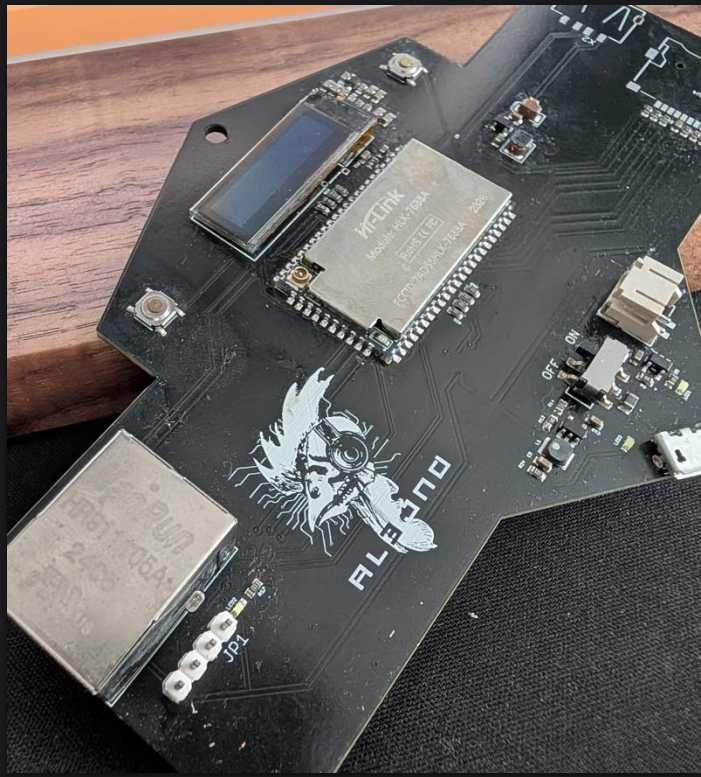
Feature	Description	Common Use
PWM	Pulse Width Modulation	Motor control
Interrupts	Event detection	Button press
ADC	Analog reading	Sensors
DAC	Analog output	Audio output



Communication Protocol Pins

- UART (Universal Asynchronous Receiver/Transmitter)

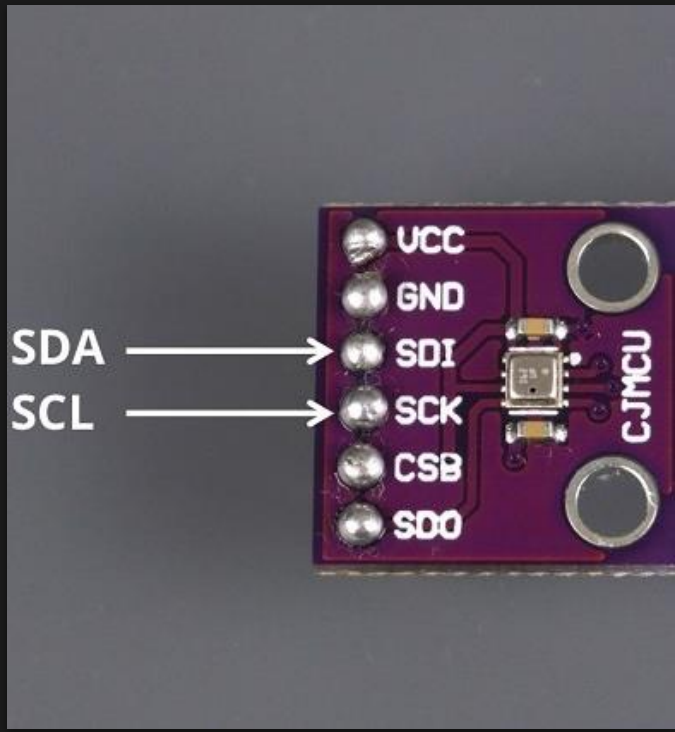
Pin	Direction	Description
TX	Output	Transmit Data
RX	Input	Receive Data
GND	Ground	Ground Reference
VCC	Voltage	Voltage



Communication Protocol Pins

- I2C (Inter-Integrated Circuit)

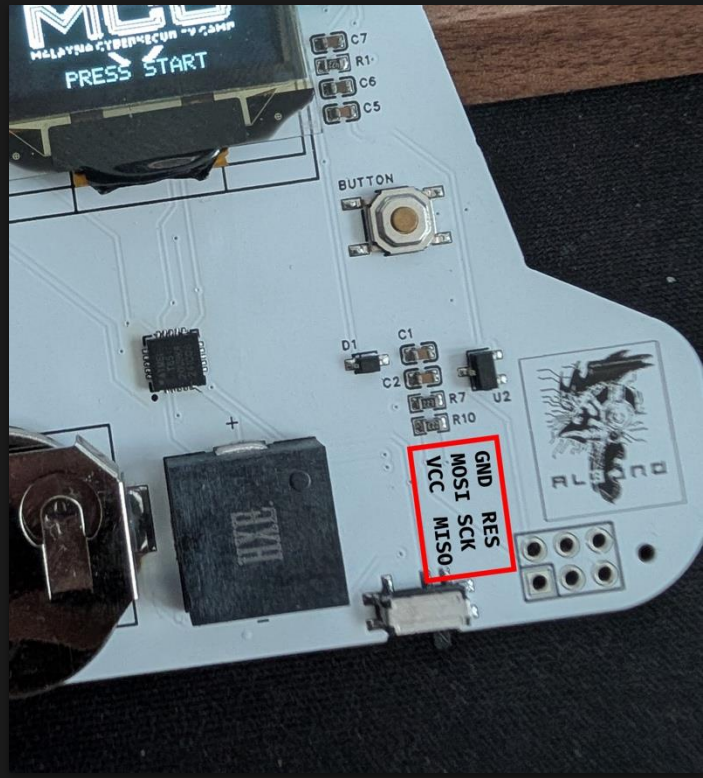
Pin	Type	Description
SCL	Output	Clock Line
SDA	Bidirectional	Data Line



Communication Protocol Pins

- SPI (Serial Peripheral Interface)

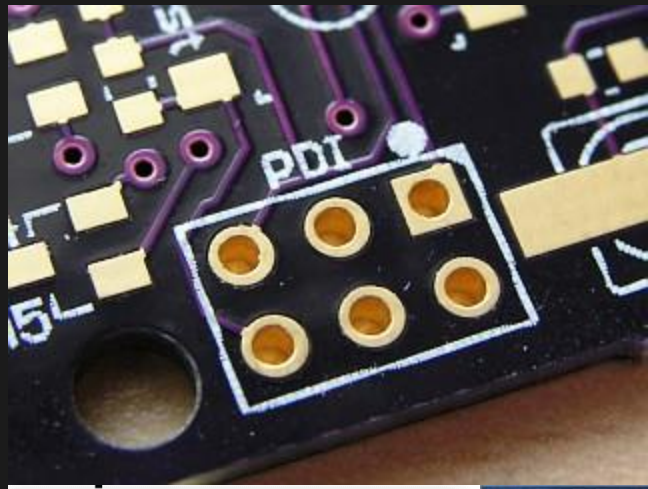
Pin	Direction	Description
MOSI	Output	Master Out
MISO	Input	Master In
SCK	Output	Clock
CS	Output	Chip Select

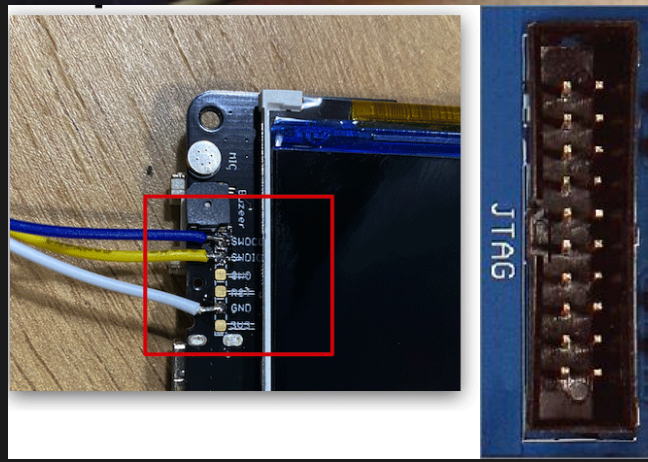


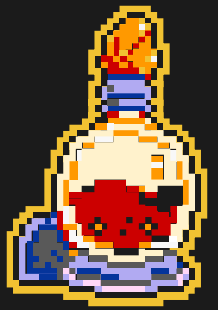
Special Function Pins

- Common Debug Interface Pins

Pin	Description
JTAG	Full debug capability
SWD	Simplified 2-wire debug
PDI	Atmel programming interface

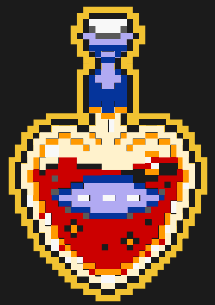






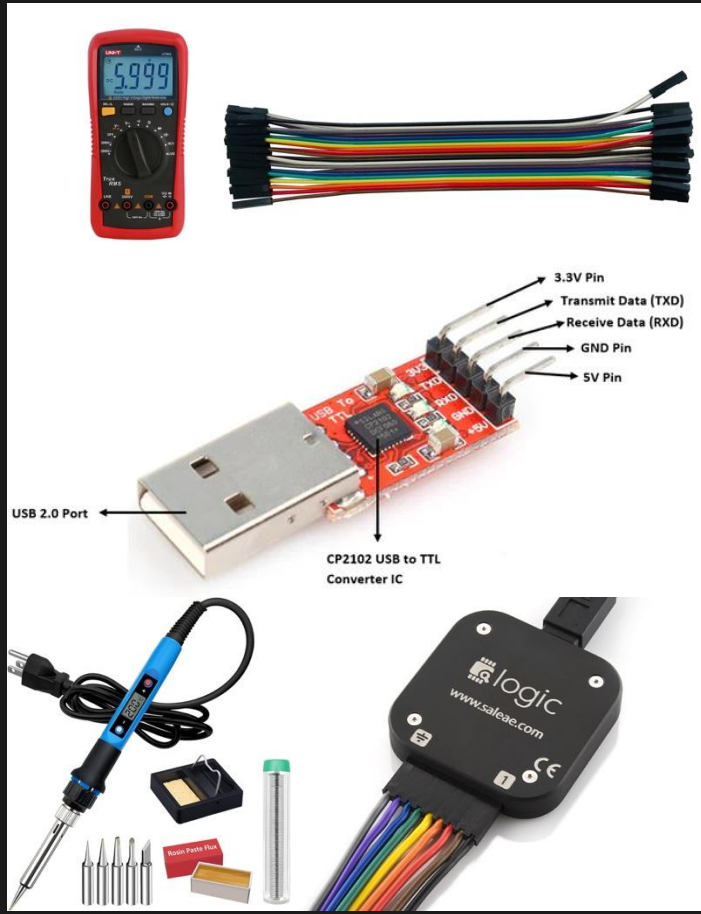
08

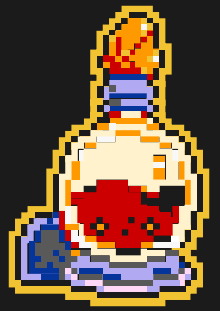
Essential Tools



Essential Tools

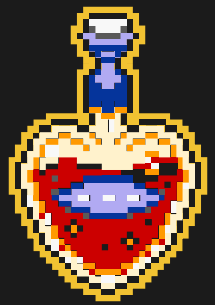
- Your Hardware Hacking Toolkit (simplified):
  - Multimeter
  - USB-to-TTL adapter
  - Jumper wires
  - Soldering iron
  - Software tools (PuTTY, screen, minicom) aka Laptop
  - Logic analyzer





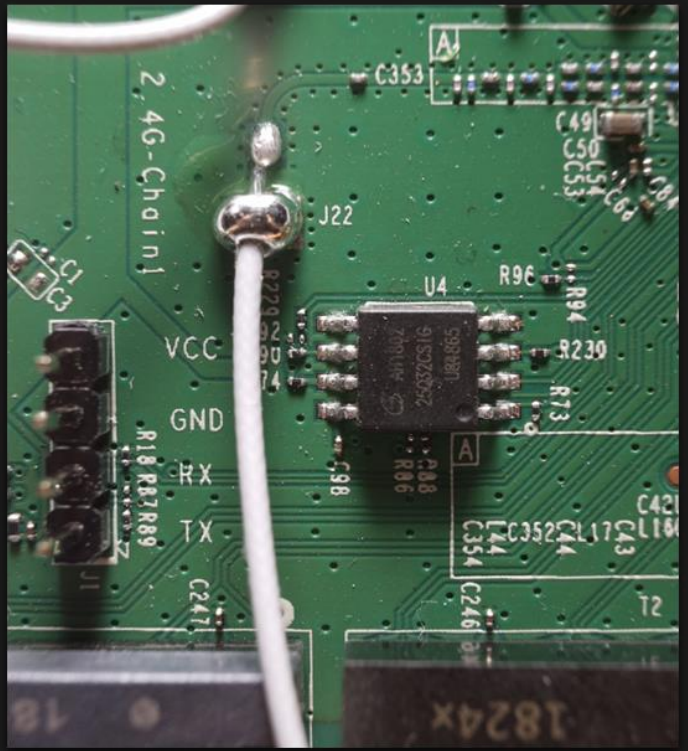
09

Signal Tracing



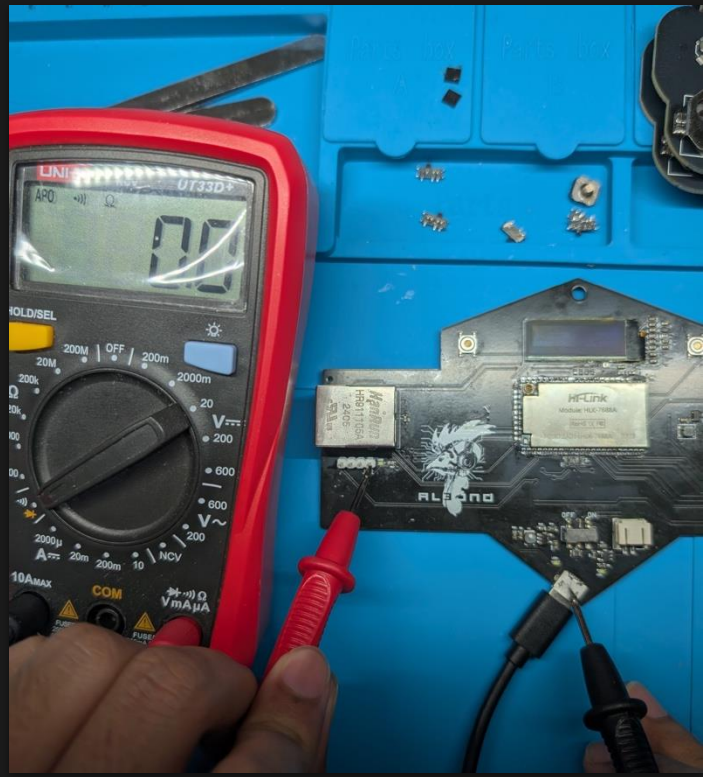
Signal Tracing

- Pin Identification
  - Use multimeter in continuity mode
  - Identifying active pins
  - Common test points
  - Logic analyzer basics
  - Signal pattern recognition



Signal Tracing (Multimeter)

1. GND identification
    - Set multimeter to continuity mode
    - Probe between suspected ground points (USB shield/wifi antenna, etc.) to the pins
    - When you hear beep sound, we can confirm the GND pin.

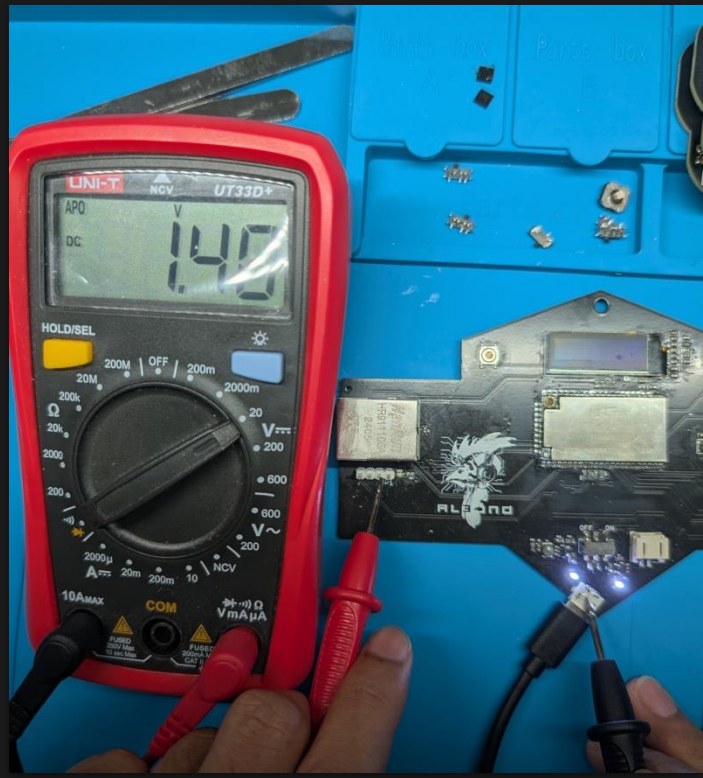


Signal Tracing (Multimeter)

2. TX Pin
    - Power on device
    - Set multimeter to DC voltage
    - Connect black probe to confirmed ground
    - Test suspected pins
    - Monitor voltage during boot

TX Pin Characteristic

State	Expected Behaviour
Boot/startup	Rapid voltage fluctuation
Idle	Steady high (3.3V/5V)
Active/transmit	Voltage fluctuations

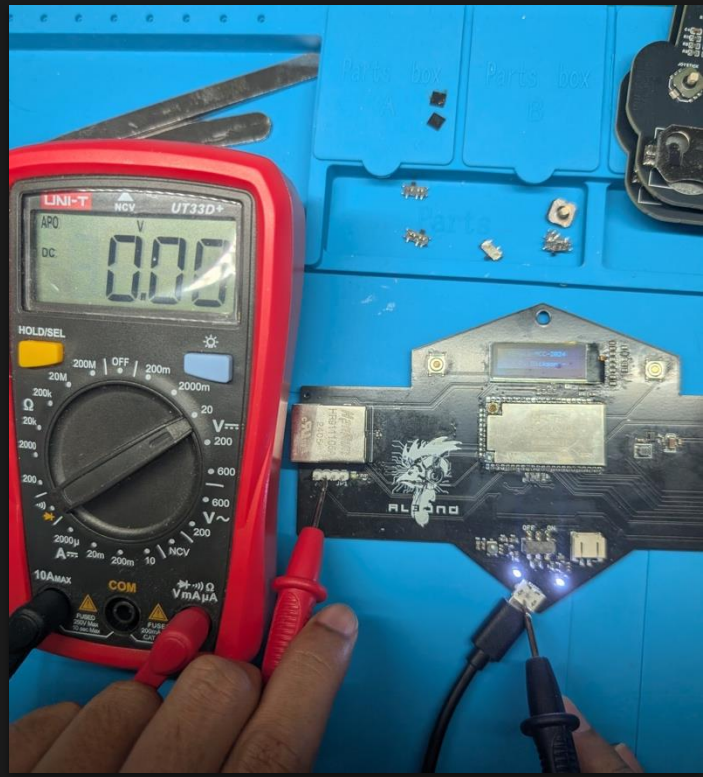


Signal Tracing (Multimeter)

2. RX Pin
    - Power on device
    - Set multimeter to DC voltage
    - Connect black probe to confirmed ground
    - Test suspected pins
    - Monitor voltage during boot

TX Pin Characteristic

- Usually either high voltage or low voltage

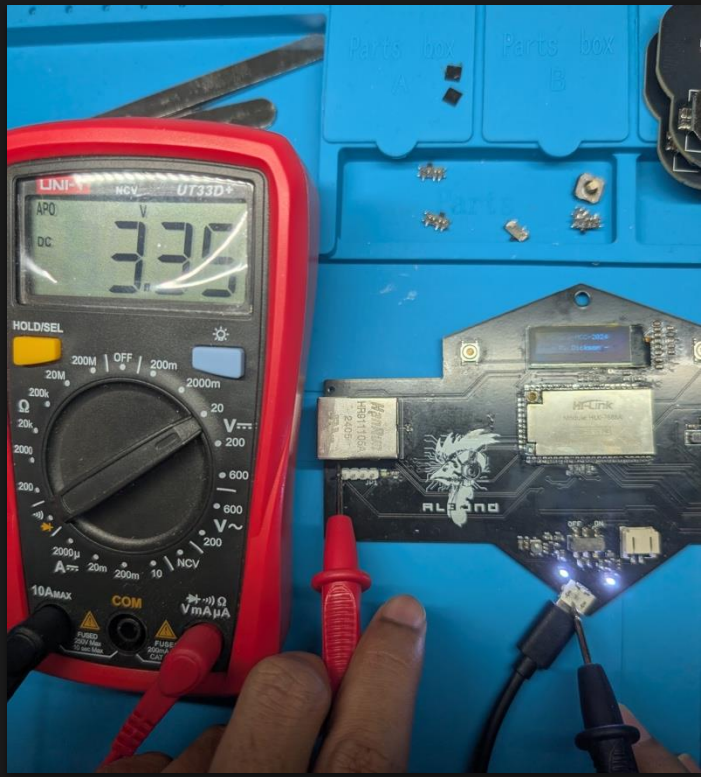


Signal Tracing (Multimeter)

2. VCC Pin
    - Power on device
    - Set multimeter to DC voltage
    - Connect black probe to confirmed ground
    - Test suspected pins
    - Monitor voltage during boot

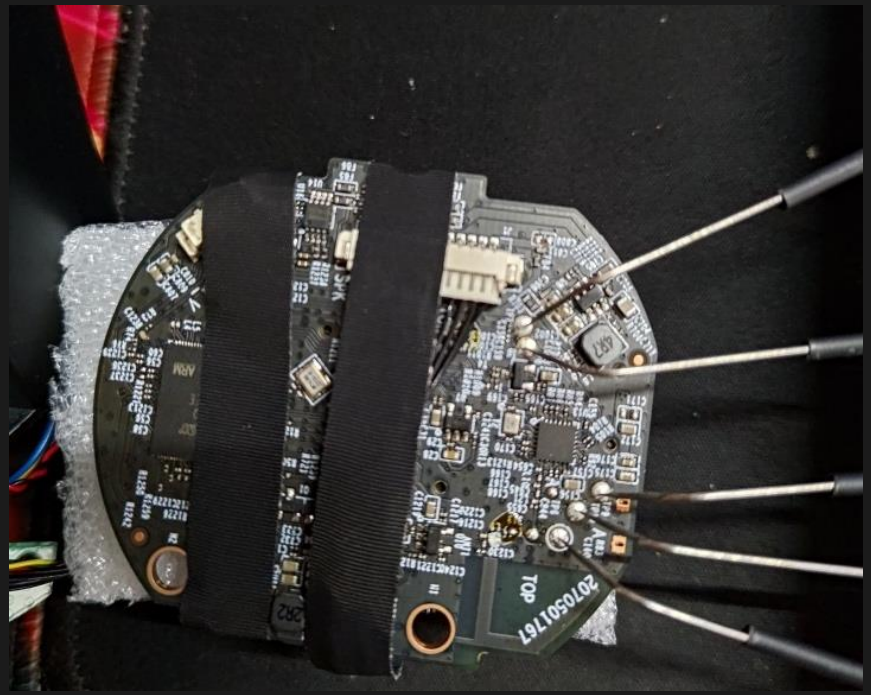
TX Pin Characteristic

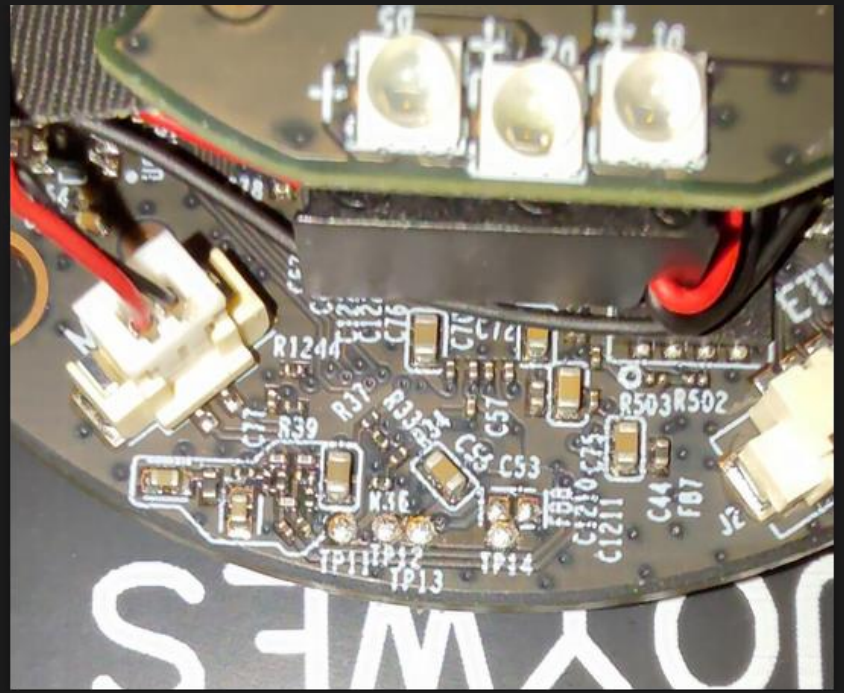
- Always steady high voltage (3.3V/5V)



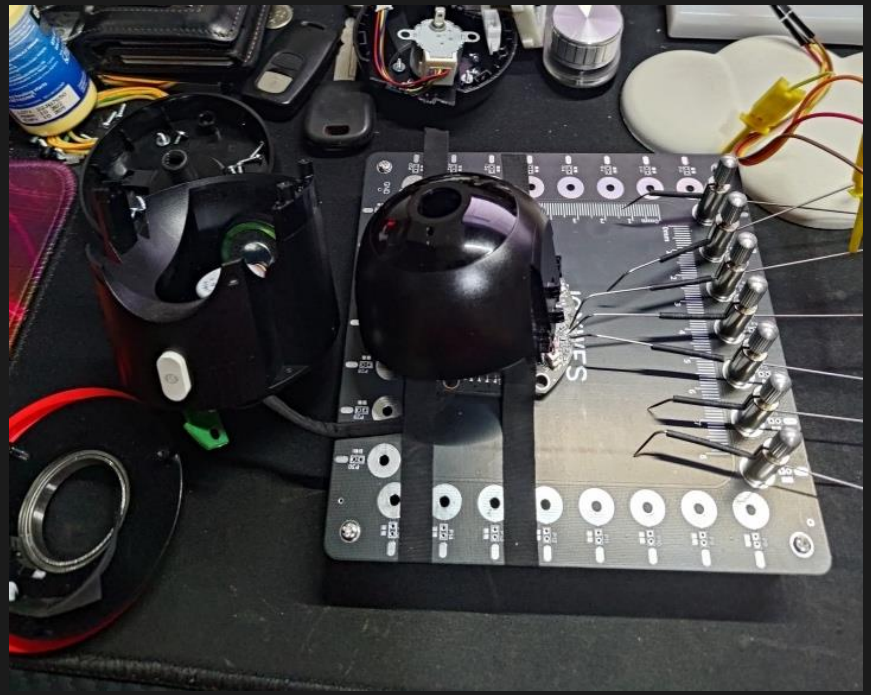
Signal Tracing (Multimeter)

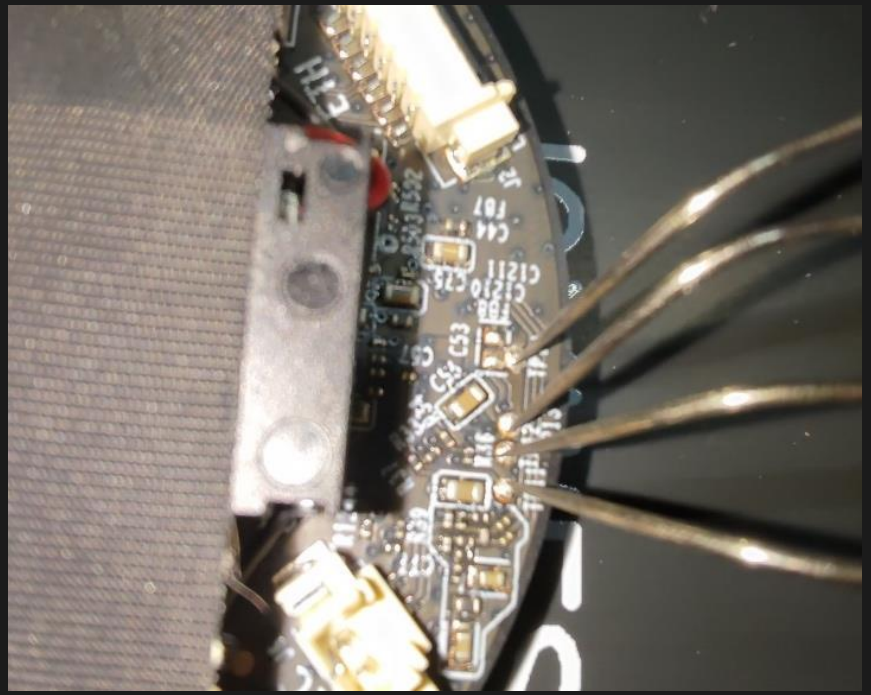
Signal Tracing??

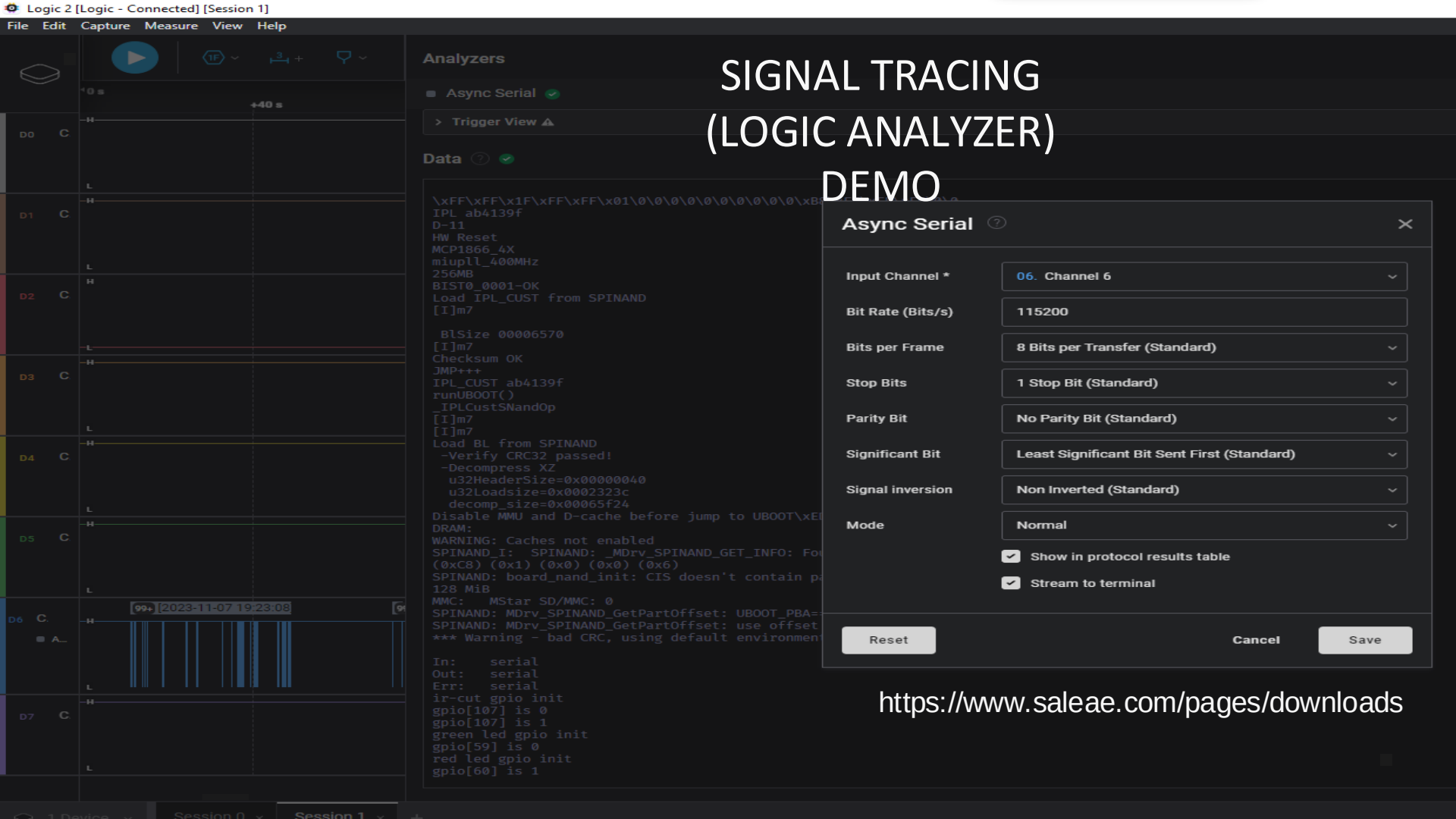




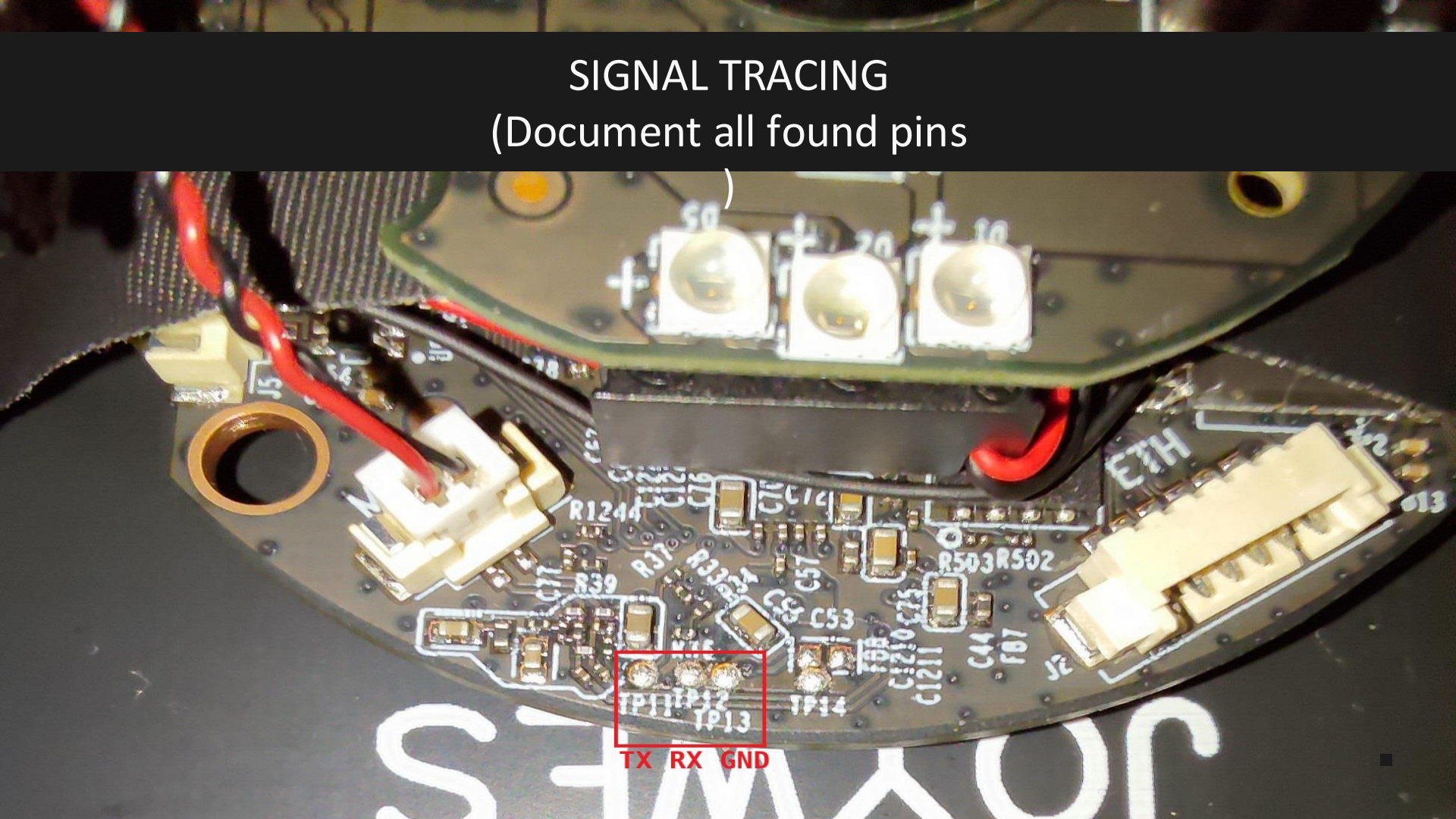
Signal Tracing??

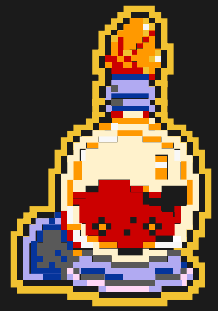





Logic 2 [Logic - Connected] [Session 1]
File Edit Capture Measure View Help
Analyzers
Async Serial
Trigger View
Data
SIGNAL TRACING
(LOGIC ANALYZER)
DEMO
Async Serial
Input Channel *
06. Channel 6
Bit Rate (Bits/s)
115200
Bits per Frame
8 Bits per Transfer (Standard)
Stop Bits
1 Stop Bit (Standard)
Parity Bit
No Parity Bit (Standard)
Significant Bit
Least Significant Bit Sent First (Standard)
Signal inversion
Non Inverted (Standard)
Mode
Normal
Show in protocol results table
Stream to terminal
Reset
Cancel
Save
https://www.saleae.com/pages/downloads

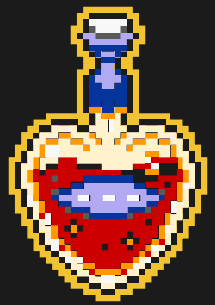
SIGNAL TRACING
(Document all found pins
)





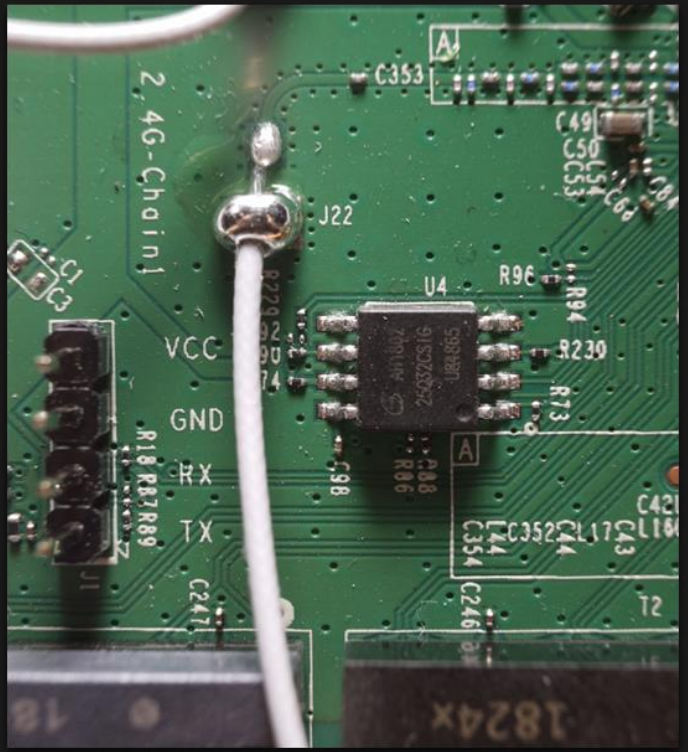
05

UART Communication



UART Communication

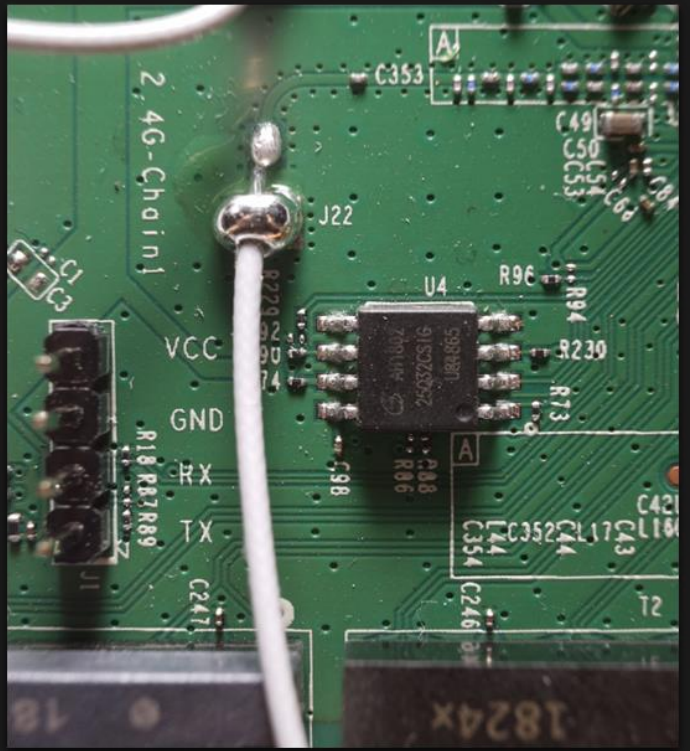
- Working with UART:
  - Identifying UART pins ✔
  - Finding baud rate
  - Accessing serial console
  - Connecting safely



UART Communication

- Working with UART:
    - Finding baud rate

```
#!/bin/bash
BAUDRATES=("1200" "2400" "4800" "9600" "19200" "38400" "57600"
"115200")

DEVICE="/dev/ttyUSB0"
current=0

while true; do
    clear
    echo "Current baud rate: ${BAUDRATES[$current]}"
    screen $DEVICE ${BAUDRATES[$current]}
    ((current=(current+1)%${#BAUDRATES[@]}))
done
```



UART Communication

- Working with UART:
  - Finding baud rate

Using picocom:

```
# Install picocom
sudo apt-get install picocom

# Start directly with specific settings
picocom --baud 9600 /dev/ttyUSB0
```

Picocom Hotkeys for Baud Rate

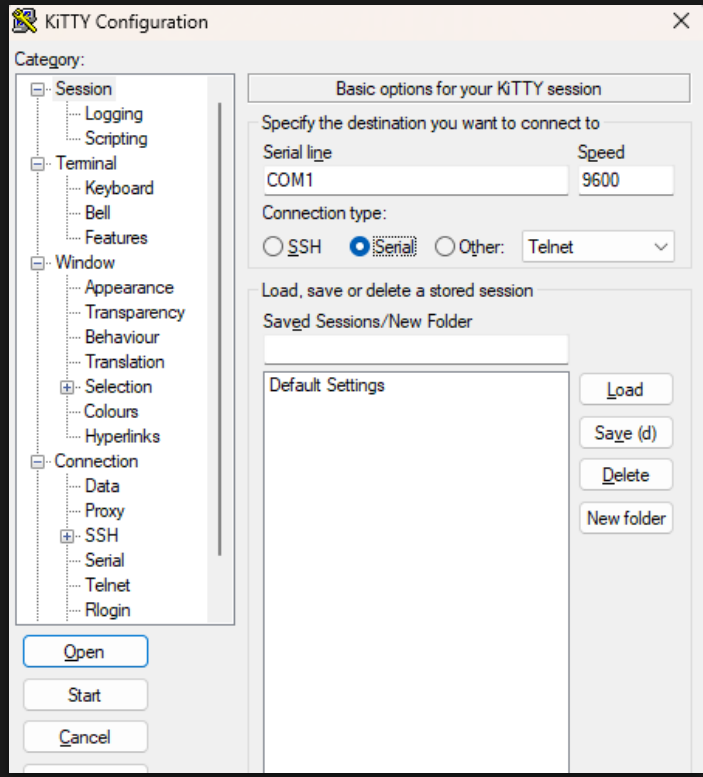
- Ctrl-a Ctrl-h - Show help
- Ctrl-a Ctrl-u - Increase baudrate (baud-up)
- Ctrl-a Ctrl-d - Decrease baudrate (baud-down)
- Ctrl-a Ctrl-x - Exit picocom

```
└─$ sudo picocom --baud 9600 /dev/ttyUSB0
picocom v3.1

port is        : /dev/ttyUSB0
flowcontrol    : none
baudrate is    : 9600
parity is      : none
databits are   : 8
stopbits are   : 1
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
hangup is      : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready
```

UART Communication

- Working with UART:
    - Accessing serial console

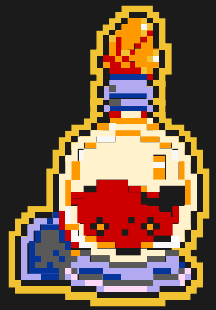
1. Windows
    - PuTTY, KiTTY, MobaXTerm, etc.
2. Linux
    - Screen, Picocom, Minicom, etc.
3. Mac
    - Screen, Picocom, Minicom, etc.



UART Communication

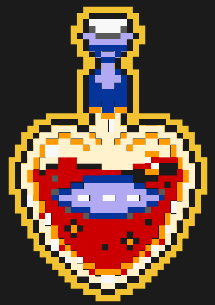
- Working with UART:
  - Accessing serial console

```
Ralink UBoot Version: 1.0
ASIC 7628_MP (Port5<->None)
DRAM component: 1024 Mbits DDR, width 16
DRAM bus: 16 bit
Total memory: 128 MBytes
Flash component: SPI Flash
Date:May 23 2020  Time:14:40:12
icache: sets:512, ways:4, linesz:32 ,total:65536
dcache: sets:256, ways:4, linesz:32 ,total:32768
RESET MT7628 PHY!!!!!!

 _   _  _       _  __  _____   __    ___   ___      ^
| | | || |     | |/ / |___ /  / /   / _ \ / _ \    / \
| |_| || |     | ' /    |_ \ / /_  | (_) | (_) |  / ^ \
|  _  || |     | . \   ___) | '_ \  > _ < > _ < / /___\ \
| | | || |___  | |\ \ / __/  (_) | | (_) | (_) / _____ \
|_| |_||_____| |_| \_\_____|\___/  \___/ \___/ /_/     \_\

            https://github.com/gnubee-git
            7688A build May 23 2020 14:40:12

Please choose the operation:
   1: Load system code to SDRAM via TFTP.
   2: Load system code then write to Flash via TFTP.
   3: Boot system code via Flash (default).
   4: Entr boot command line interface.
   5: Load system code then write to Flash via Httpd.
   7: Load Boot Loader code then write to Flash via Serial.
   9: Load Boot Loader code then write to Flash via TFTP.
 9
You chose 3

 0

3: System Boot system code via Flash.
## Booting image at bc050000 ...
   Image Name:   MIPS OpenWrt Linux-4.14.206
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    1263324 Bytes =  1.2 MB
   Load Address: 80000000
   Entry Point:  80000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
No initrd
## Transferring control to Linux (at address 80000000) ...
## Giving linux memsize in MB, 128
```



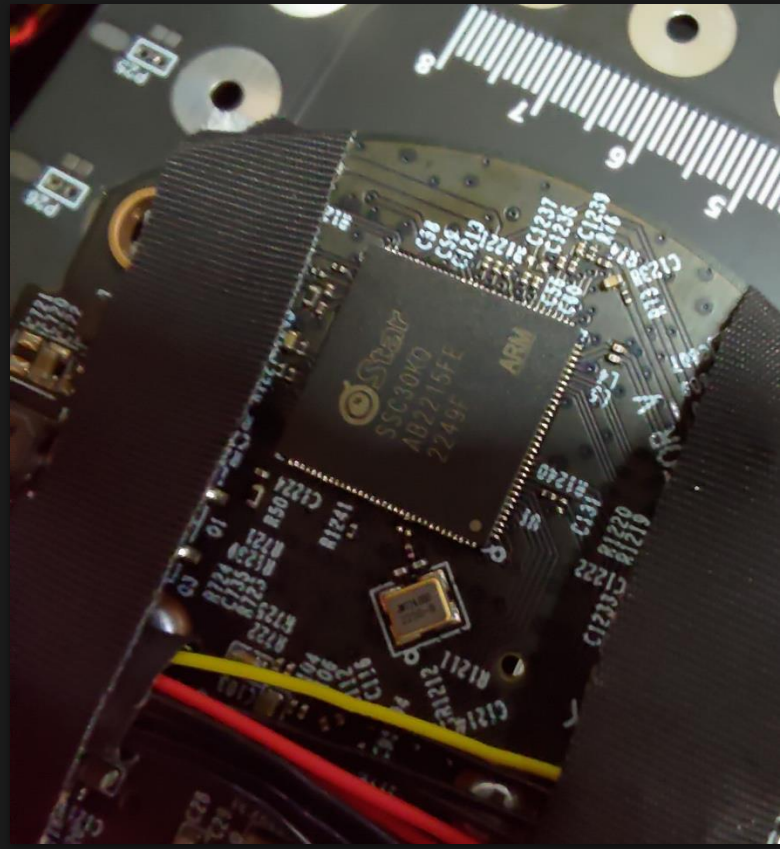
10

Identifying Components



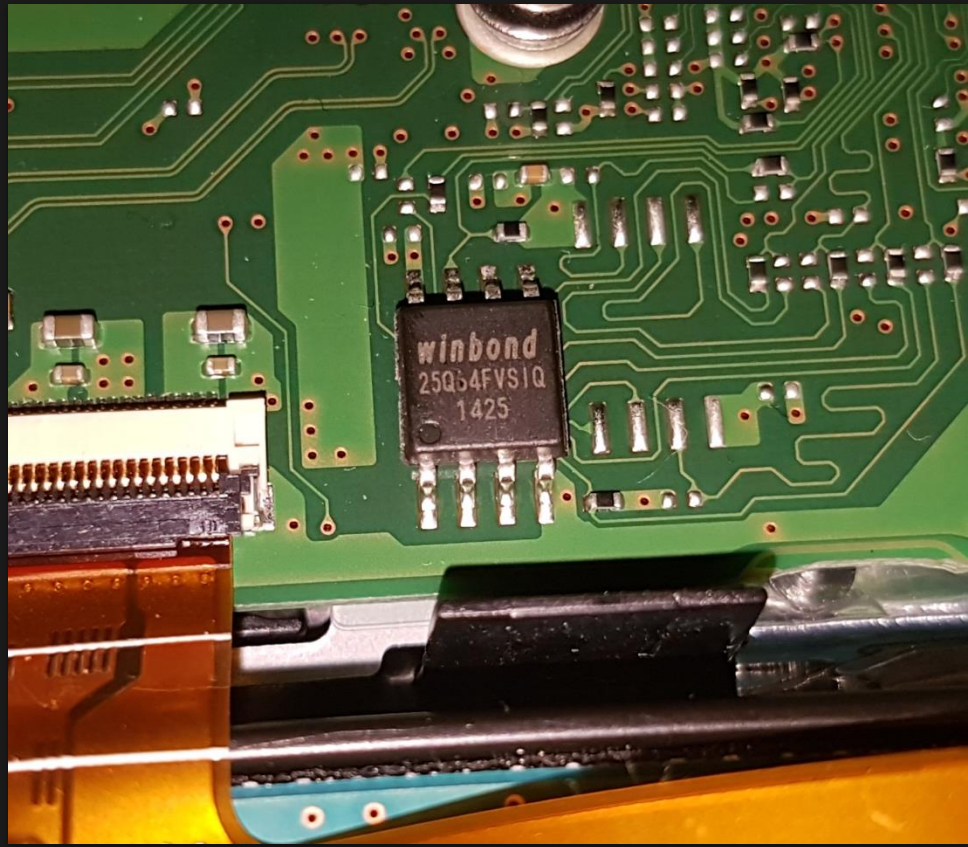
Identifying Components

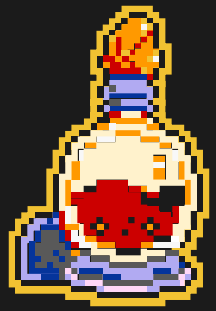
- Identifying Processor:
  - Physically look at the component
  - Example: SSC30KQ
  - GOOGLE “SSC30KQ datasheet”
  - READ THE DATASHEET!



Identifying Components

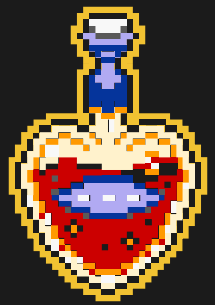
- Identifying Flash:
  - Physically look at the component
  - Example: 25Q64FVSIQ
  - GOOGLE “25Q64FVSIQ datasheet”
  - READ THE DATASHEET!





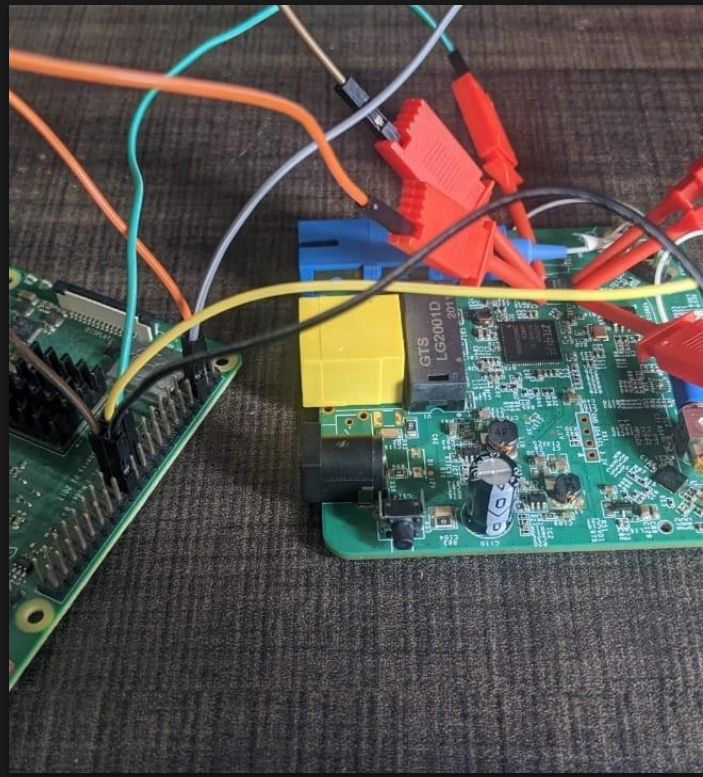
11

Firmware Extraction



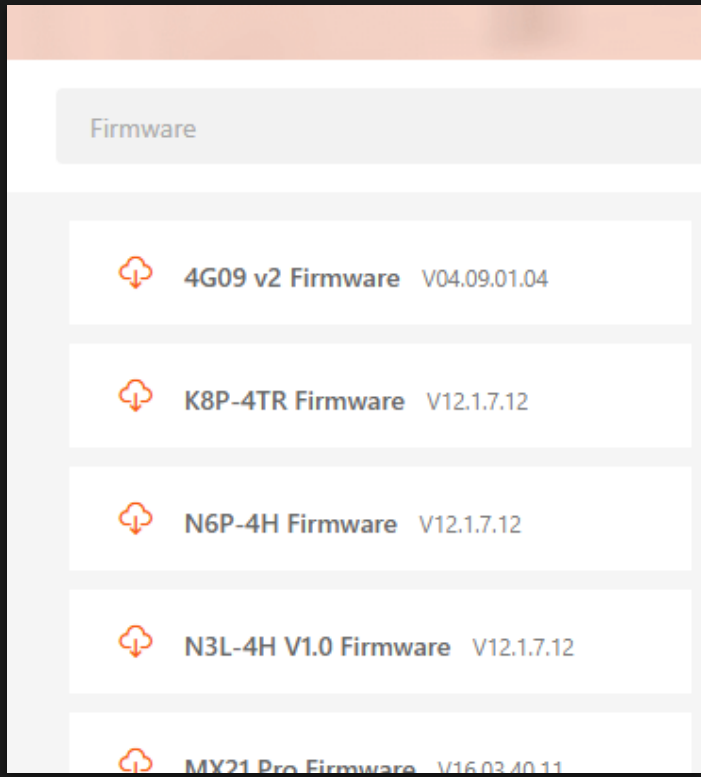
Firmware Extraction

- Common method:
  - Download from manufacturers
  - Dump from bootloader using UART
  - Dump from shell
  - SPI dumping



Firmware Extraction

- Download from manufacturers
  - Fast method to get firmware



Firmware Extraction

- Dump from bootloader using UART/Serial
  - Common U-boot commands to dump
    - spi - spi command
    - save - save a region of memory
    - md - memory display
  - These commands may vary depending on the manufacturer and flash, as they might customize bootloader for their devices.
  - This method is slow, but it works.

```
MT7628 # spi read 0×bc050000 100
read len: 256
27 5 19 56 34 fe e6 18 5f bc cc 5f 0 13 46 dc 80 0 0 0 80 0 0 0 35 ea e6 39 5 5 2
 fd ff ff a3 b7 7f 8e 57 19 13 b1 6d 37 ef d9 aa 5b 9b f8 33 72 d7 b4 cb 8f 26 e4
 be 8 f5 9c 65 c5 f3 97 f9 3b 8e 87 4 ea 63 24 51 73 2f 80 a 9e 49 87 53 74 1c a4
19 93 e3 3f 43 21 9e 9 13 14 8e 75 f6 4c 99 2c 73 89 d6 a5 90 6c df 78 56 b1 bd 1
MT7628 # md.b
Usage:
md      - memory display

MT7628 # md.b 0×bc050000 100
bc050000: 27 05 19 56 34 fe e6 18 5f bc cc 5f 00 13 46 dc    '..V4..._.._..F.
bc050010: 80 00 00 00 80 00 00 00 35 ea e6 39 05 05 02 03    ........5..9....
bc050020: 4d 49 50 53 20 4f 70 65 6e 57 72 74 20 4c 69 6e    MIPS OpenWrt Lin
bc050030: 75 78 2d 34 2e 31 34 2e 32 30 36 00 00 00 00 00    ux-4.14.206.....
bc050040: 6d 00 00 80 00 dc 08 3c 00 00 00 00 00 00 00 6f    m......<.......o
bc050050: fd ff ff a3 b7 7f 8e 57 19 13 b1 6d 37 ef d9 aa    .......W...m7...
bc050060: 5b 9b f8 33 72 d7 b4 cb 8f 26 e4 7f bf d3 b5 1f    [..3r....&......
bc050070: 7c 80 db ed 16 2c 08 f8 90 08 92 5b c1 d0 8d e5    |....,.....[....
bc050080: b6 38 5d 41 93 63 b4 b1 3b 5e 8a cd ae 95 62 18    .8]A.c..;^....b.
bc050090: be ad ff a0 6c 88 a1 be 08 f5 9c 65 c5 f3 97 f9    ....l......e....
bc0500a0: 3b 8e 87 04 ea 63 24 51 73 2f 80 0a 9e 49 87 53    ;....c$Qs/...I.S
bc0500b0: 74 1c a4 08 a1 a5 c3 e5 c0 c9 5b 19 bb d8 da 61    t.........[....a
bc0500c0: fa 35 78 5d 3d 9a 2c 08 5c 43 ed 8d 84 0c 56 65    .5x]=.,.\C....Ve
bc0500d0: b9 f3 93 5e 78 14 ad db 8c 68 bc 33 7a c7 9a 19    ...^x....h.3z...
bc0500e0: 93 e3 3f 43 21 9e 09 13 14 8e 75 f6 4c 99 2c 73    ..?C!.....u.L.,s
bc0500f0: 89 d6 a5 90 6c df 78 56 b1 bd 1e aa 0a a2 b6 bc    ....l.xV........
```

Firmware Extraction

- Dump from shell
  - During boot, most of the time serial console will display information of partition table
  - These information can be used to dump the whole flash

```
[    0.506143] m25p80 spi0.0: w25q256 (32768 Kbytes)
[    0.515541] 4 fixed-partitions partitions found on MTD device spi0.0
[    0.528129] Creating 4 MTD partitions on "spi0.0":
[    0.537629] 0x000000000000-0x000000030000 : "u-boot"
[    0.548432] 0x000000030000-0x000000040000 : "u-boot-env"
[    0.559890] 0x000000040000-0x000000050000 : "factory"
[    0.570812] 0x000000050000-0x000000800000 : "firmware"
[    0.619359] 2 uimage-fw partitions found on MTD device firmware
[    0.631124] 0x000000050000-0x00000018471c : "kernel"
[    0.641857] 0x00000018471c-0x000000800000 : "rootfs"
[    0.652582] mtd: device 5 (rootfs) set to be root filesystem
[    0.665355] 1 squashfs-split partitions found on MTD device rootfs
[    0.677661] 0x0000004e0000-0x000000800000 : "rootfs_data"
```

Firmware Extraction

- Dump from shell
  - Having this information, we can check `/proc/mtd` with `cat /proc/mtd`

```
root@OpenWrt:/# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00030000 00010000 "u-boot"
mtd1: 00010000 00010000 "u-boot-env"
mtd2: 00010000 00010000 "factory"
mtd3: 007b0000 00010000 "firmware"
mtd4: 0013471c 00010000 "kernel"
mtd5: 0067b8e4 00010000 "rootfs"
mtd6: 00320000 00010000 "rootfs_data"
```

```
[    0.506143] m25p80 spi0.0: w25q256 (32768 Kbytes)
[    0.515541] 4 fixed-partitions partitions found on MTD device spi0.0
[    0.528129] Creating 4 MTD partitions on "spi0.0":
[    0.537629] 0x000000000000-0x000000030000 : "u-boot"
[    0.548432] 0x000000030000-0x000000040000 : "u-boot-env"
[    0.559890] 0x000000040000-0x000000050000 : "factory"
[    0.570812] 0x000000050000-0x000000800000 : "firmware"
[    0.619359] 2 uimage-fw partitions found on MTD device firmware
[    0.631124] 0x000000050000-0x00000018471c : "kernel"
[    0.641857] 0x00000018471c-0x000000800000 : "rootfs"
[    0.652582] mtd: device 5 (rootfs) set to be root filesystem
[    0.665355] 1 squashfs-split partitions found on MTD device rootfs
[    0.677661] 0x0000004e0000-0x000000800000 : "rootfs_data"
```

Firmware Extraction

- Dump from shell
  - We can also check `/proc/partitions` with `cat /proc/partitions`
  - This shows the mapping of flash partitions to mtdblock virtual devices.

```
root@OpenWrt:/# cat /proc/partitions
major minor  #blocks  name

  31     0      192 mtdblock0
  31     1       64 mtdblock1
  31     2       64 mtdblock2
  31     3     7872 mtdblock3
  31     4     1233 mtdblock4
  31     5     6638 mtdblock5
  31     6     3200 mtdblock6
```

```
[    0.506143] m25p80 spi0.0: w25q256 (32768 Kbytes)
[    0.515541] 4 fixed-partitions partitions found on MTD device spi0.0
[    0.528129] Creating 4 MTD partitions on "spi0.0":
[    0.537629] 0x000000000000-0x000000030000 : "u-boot"
[    0.548432] 0x000000030000-0x000000040000 : "u-boot-env"
[    0.559890] 0x000000040000-0x000000050000 : "factory"
[    0.570812] 0x000000050000-0x000000800000 : "firmware"
[    0.619359] 2 uimage-fw partitions found on MTD device firmware
[    0.631124] 0x000000050000-0x00000018471c : "kernel"
[    0.641857] 0x00000018471c-0x000000800000 : "rootfs"
[    0.652582] mtd: device 5 (rootfs) set to be root filesystem
[    0.665355] 1 squashfs-split partitions found on MTD device rootfs
[    0.677661] 0x0000004e0000-0x000000800000 : "rootfs_data"
```

Firmware Extraction

- Dump from shell
  - Transferring data over serial console will take forever
  - If the device can mount USB or SD card, this will become an advantage
  - After inserting USB/SD card, check /proc/partitions again and make sure you see a new partitions.

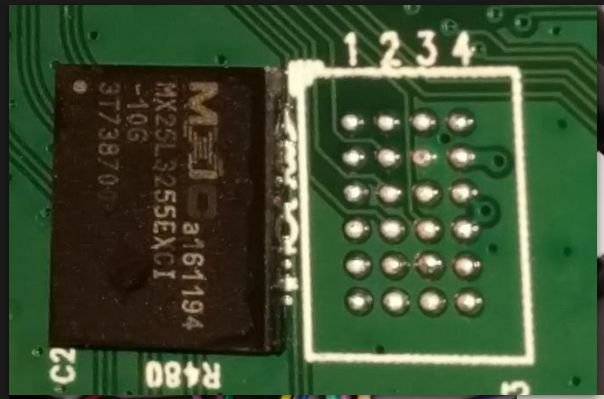
```
root@OpenWrt:/# cat /proc/partitions
major minor  #blocks  name
SNIP...
 31       5       6638 mtdblock5
 31       6       3200 mtdblock6
179       0   15138876 mmcblk0
179       1   15138784 mmcblk0p1
```

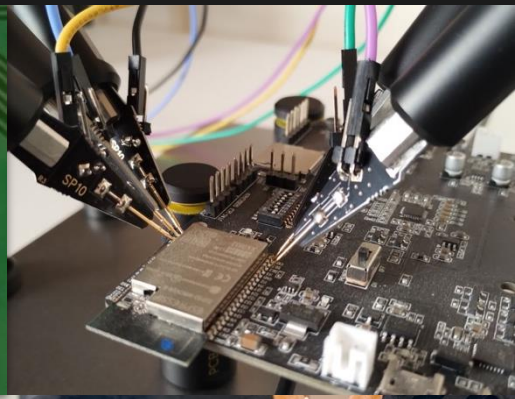
- Step to dump

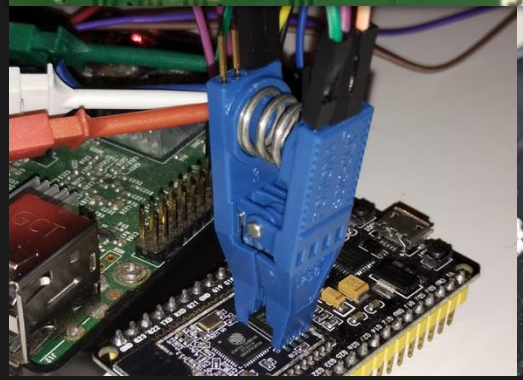
```
# mount tmpfs to /dev
mount -t tmpfs tmpfs /dev -o mode=0755,size=512K
mknod /dev/mmcblk0p1 b 179 1  # mounting device
mount tmpfs /tmp -t tmpfs -o size=20633600,nosuid,nodev,mode=1777
mkdir /tmp/sdcard
mount -t vfat /dev/mmcblk0p1 /tmp/sdcard/

ls /tmp/sdcard/ # checking if sdcard is mounted

#start dumping mtdblock0 to /tmp/sdcard/flashdump.bin
cat /dev/mtdblock0 >> /tmp/sdcard/flashdump.bin
cat /dev/mtdblock1 >> /tmp/sdcard/flashdump.bin #continue
...[SNIP]
cat /dev/mtdblock6 >> /tmp/sdcard/flashdump.bin

ls -al flashdump.bin # check size

umount /tmp/sdcard # unmount sdcard, can trasfer to PC
```

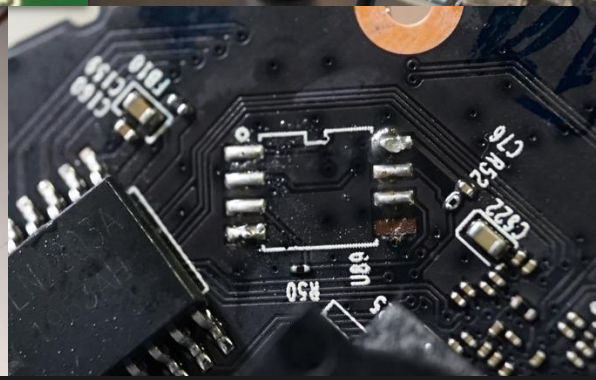
Firmware Extraction

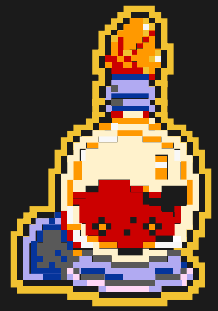
- SPI dumping
  - Desolder chip (Chip-off extraction)
  - Use pogo pin / Test clip
  - Use Test probes
- Personally, I do not prefer to do this as anything could go wrong.
- This is the most challenging method and should be considered a last resort if other methods are not viable.





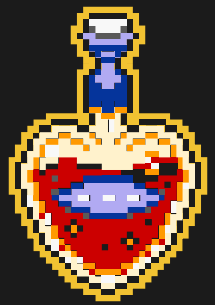






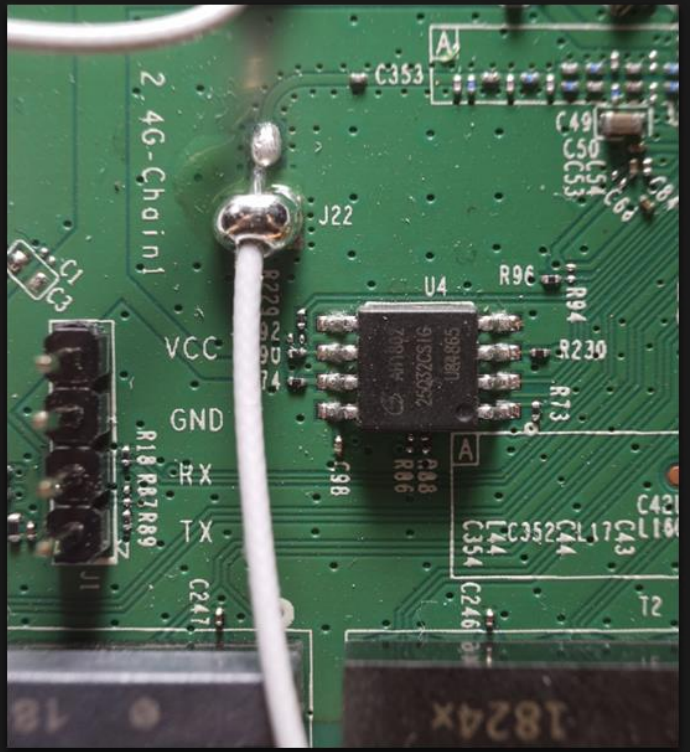
12

Basic Firmware Analysis



Basic Firmware Analysis

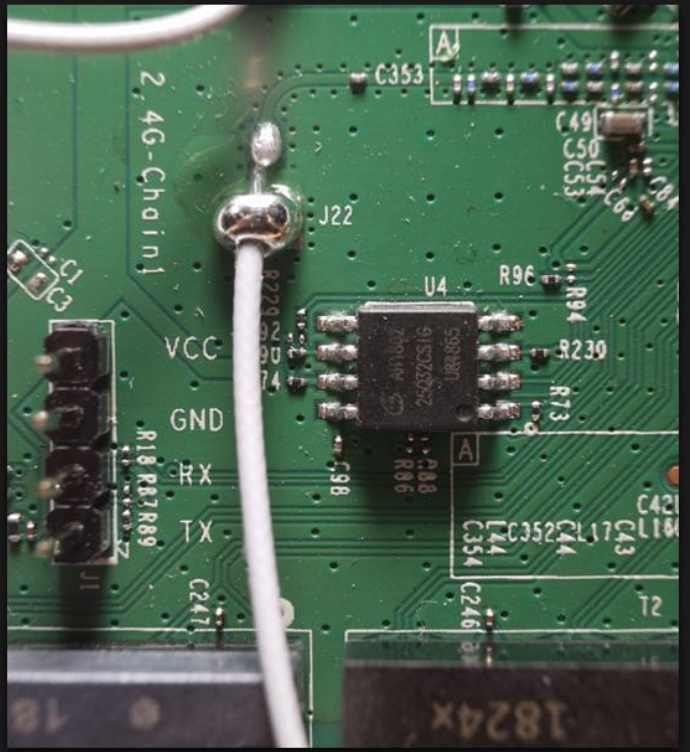
- Firmware Analysis Techniques:
  - File System Extraction
  - Binary Analysis Tools
  - Common Vulnerability Patterns
  - Static Analysis Checklist



Basic Firmware Analysis

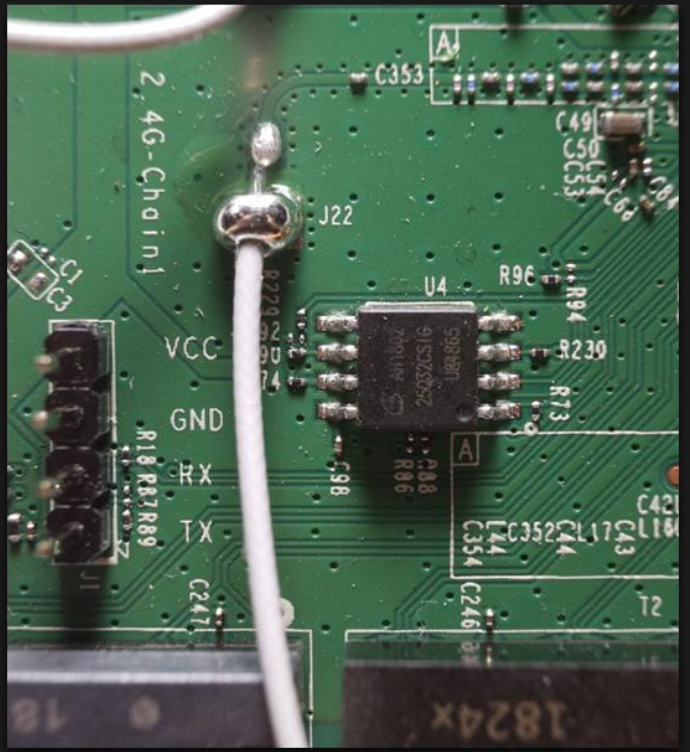
- File System Extraction:

```
# Extract using binwalk
binwalk -e firmware.bin

# Mount filesystem
sudo mount -o loop filesystem.img /mnt/analysis
```



Basic Firmware Analysis

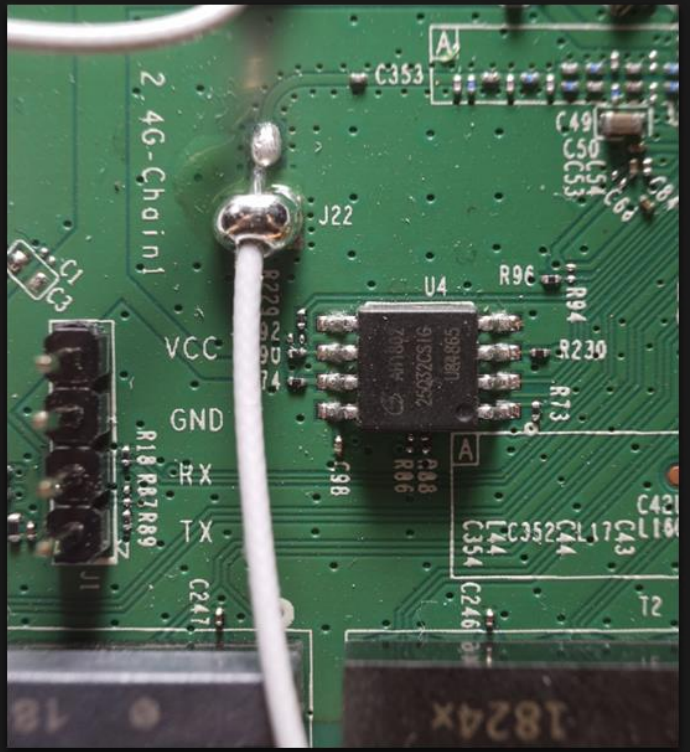
- Binary Analysis Tools:
  - IDA Pro
  - Ghidra
  - GDB
  - JEB
  - r2



Basic Firmware Analysis

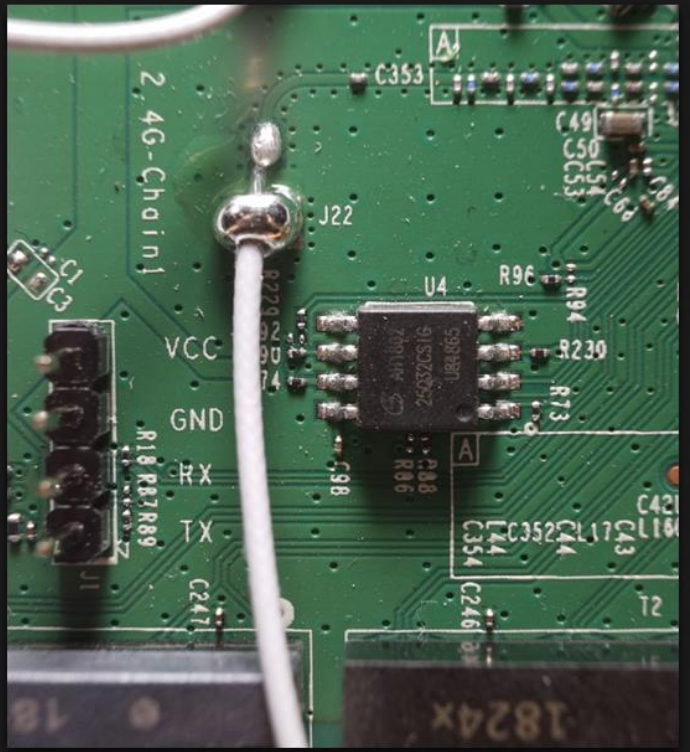
- Common Vulnerability Patterns:

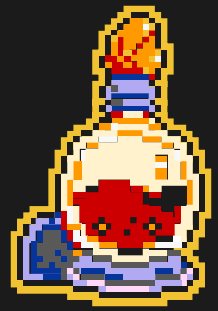
Pattern	Description
strcpy()	Buffer overflow potential
system()	Command injection risk
hardcoded passwords	Authentication bypass
unencrypted storage	Information disclosure



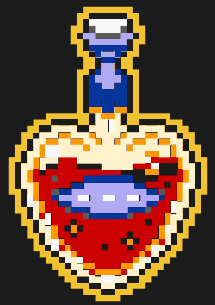
Basic Firmware Analysis

- Static Analysis Checklist:
  - Check for hardcoded credentials
  - Identify encryption keys
  - Look for debug functions
  - Review network services





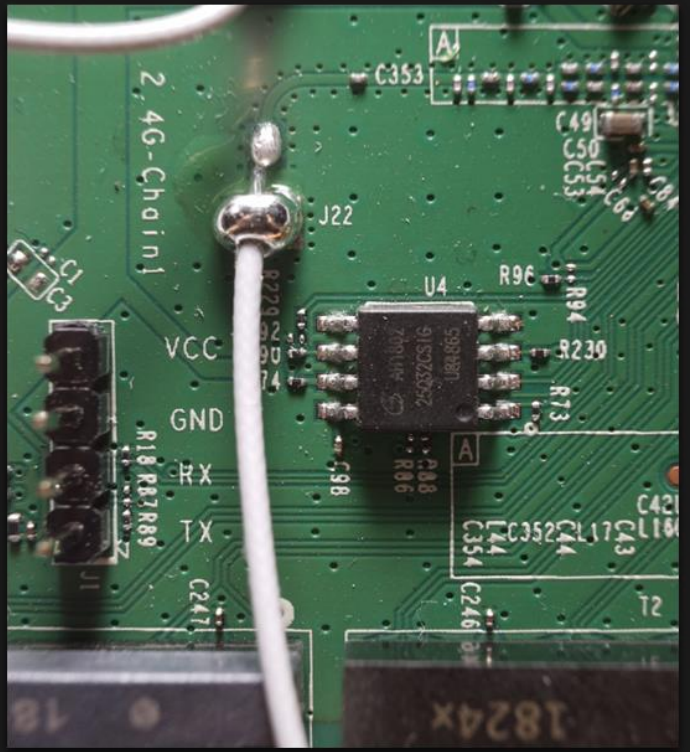
END



Continuing Your Learning Journey

Continuing Your Learning Journey

- Online Resources:
  - https://www.youtube.com/@mattbrwn
  - https://www.youtube.com/@ehacks
- Advanced Topics
  - Fault Injection
  - Reverse Engineering
  - Modified firmware/Custom firmware



“This is a quote, words full of wisdom that someone important said and can make the reader get inspired”

—SOMEONE FAMOUS

THANKS

Do you have any questions?

naja[at]aymjnd.one