# Group Members:

1. Priyanka Abhijit Tamhankar

2. Mohit Palliyil Sathyaseelan

# Lab 2: Establishing a JTAG connection

## Understanding JTAG and Debugging:

1. Briefly describe all required connections for JTAG and what they do

Ans)

JTAG(Joint Test Action Group) is a serial bus with four signals. Test Clock (TCK), Test Mode Select (TMS), Test Data Input (TDI), and Test Data Output (TDO). The bus is used as a test bus for the 'Boundary-Scan' of [ICs](http://www.interfacebus.com/Semiconductor.html), as in Design-For-Testability ([DFT](http://www.interfacebus.com/Design_For_Test.html)).

TCK:  The Test Clock

* used to load the test mode data from the TMS pin and the test data on the TDI pin [on the rising edge].
* On the falling edge test clock outputs the test data on the TDO pin.
* may require buffering or be fanned out by multiple drivers depending on the distance and number of devices in the chain.
* Using multiple drivers would also require a termination resistor on each TCK line.

TMS: Test Mode Select Input

* controls the operation of the test logic, by receiving the incoming data.
* The value at the input on the rising edge of the clock controls the movement through the states of the TAP controller.
* Has an internal pull-up, so the input is high with no input.
* Bused to all ICs in the JTAG chain.
* Should have a 10k [pull-up resistor](http://www.interfacebus.com/IC_Output_Input_Pullup_Resistor_Values.html) on the line.

TDI: Test Data Input

* receives serial input data which is either feed to the test data registers or instruction register.
* has an internal pull-up, so the input is high with no input
* signal is fed to the TDI pin of the first IC in the JTAG chain.
* The TDO signal from that IC is then sent to the TDI pin of the next IC in the chain or sent back out to the JTAG header.
* line should have a 10k pull-up resistor on the line.

TDO: Test Data Output

* outputs serial data which comes from either the test data registers or instruction register.
* Data applies to the TDI pin will appear at the TDO pin but may be shifted of a number of clock cycles, depending on the length of the internal register.
* high-Impedance.
* The TDO signal is the output from a JTAG device that feed the TDI input of another JTAG device.
* should have a 10k pull-up resistor on the line.
* should also include a 22 ohm series resistor placed near the last device in the JTAG chain.

TRST: Test Rest

* will asynchronously reset the JTAG test logic.
* The logic is reset (with TRST) regardless of the state of TMS or TCLK.
* bused to all ICs in the JTAG chain.
* should include a pull-down resistor when possible to reduce the chance the signal floats.

1. What are the ‘two paths’ of JTAG? What does each path do?

Data registers :

Consist of boundary scan registers, bypass registers and device ID registers.

* Boundary scan registers: allow for testing by interchange of data between I/O pins.
* The bypass register: provides a single bit scan path between TDI and TDO.
* Device ID register: A 32-bits that storea information to identify the device manufacturer, part number and version.

Instruction register:

Minimum 2 bits long and can hold the current instruction.

Determines the action that will be performed and which DR to access.

It consists of:

* Hold register: stores previous instructions
* Shift register: loads the next instruction.

1. How does JTAG debugging relate C/higher level code to assembly?

The JTAG interface allows us to

* pause and step through operations
* inspect memory
* write bytes directly into memory
* set watch-points and break-points
* inject code into the process or process memory.

This technique is called “Hardware-based Software Debugging.” We are manipulating the hardware to perform traditional software debugging tasks.

1. What are breakpoints?

Breakpoint is an intentional user imposed break or stop in software using for debugging purpose.

They are used added to investigate state of variables/program at a certain point. The programmer can set breakpoints on essential variables, flags, memory and register values and verify the correct functionality.

1. How can the contents of on-chip memory be viewed?

JTAG can be used to access memory and registers. There are 4 steps to extract on-chip memory using JTAG

1. Identify the JTAG connection pins.
2. Test the connection with a JTAG adapter.
3. Gather information about the memory mapping of the chip.
4. Extract the firmware from the flash memory.

Compilers convert the high level code to binary(machine code) which involves setting and manipulating values on physical locations on memory. JTAG can be used to check values in registers and physical memory

1. Place a breakpoint on line 95 of console.c, to view the contents of the user supplied serial input. Take a picture displaying the inputs as both a variable and in memory.

## Attacking IoT Devices with JTAG:

1. Briefly describe the process of reverse engineering a binary.

• We first gain some information about the binary file by importing it to suitable tools, such as Ghidra.

• We will note some basic details such as type of file, endianness, version history, language etc.

• Then, we can extract a list of symbols, which are references to some type of data like an import, a global variable or a function.

• The tool displays symbol tree, assembly code with instructions, addresses, bytes, operands and commands. The decompiled output is available which converts assembly back to high level code.

• Then, we navigate around the binary and look for individual functions. For example, the main function of a program can be searched in the symbol tree view.

• If the binary does not have symbols, we can look for arguments that functions are calling.

• We can analyze the program by editing a function to see how the program responds.

1. Briefly describe three ways an attacker can use JTAG to violate CIA.

Pin modification:

* Uses boundary scan operation, for serially testing all different modules of a PCB .
* The TDI pin injects test signals or secret data into x number of devices.
* These devices can be serially connected memories or combinational circuits etc.
* If the user injects some secret data to the TDI pin, the data can be sniffed by a device such as memory, controller etc., that is already present in that series of devices.

JTAGulator tool:

* attacker checks for open pins indirectly available on PCB to normal user.
* Uses brute force attack.
* Once the pinout from PCB board is recognized, the attacker can easily get access to the firmware or bootloader.
* Attackers access controller’s internal memory leading to the manipulation of the register values.
* Eg: if a controller read protection is implemented and the attacker manipulates register values it can potentially bypass the protection layer.

1. Using the breakpoint on line 95 of console.c, input an incorrect password and step into the assembly. In the assembly code, identify the outcome of the string comparison. Alter the values in the registers to have the code grant authorization/pass the check, despite the password being incorrect.

Note: As discussed with Max in office hours the platformio update error leads to the debugger not hitting the break point.

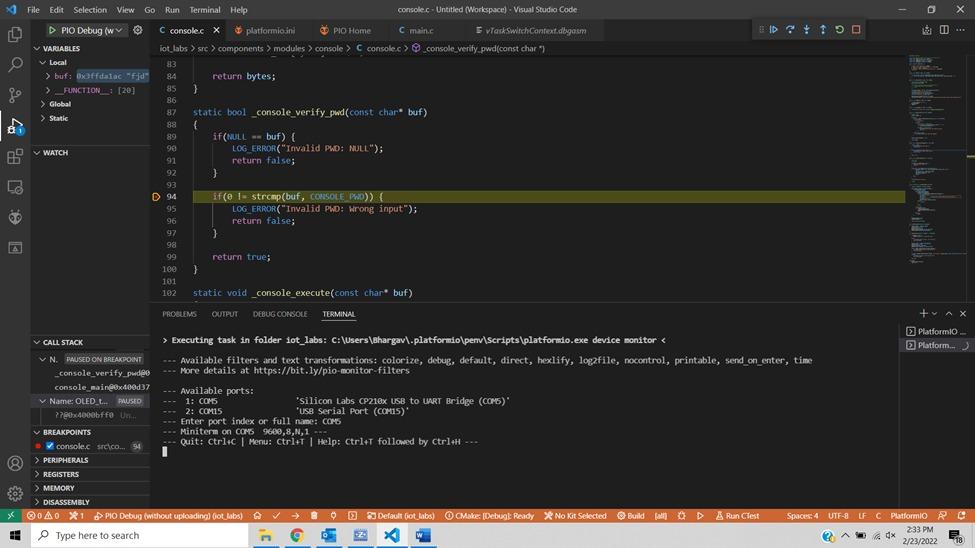


Fig 1 : Breakpoint set at line number 94

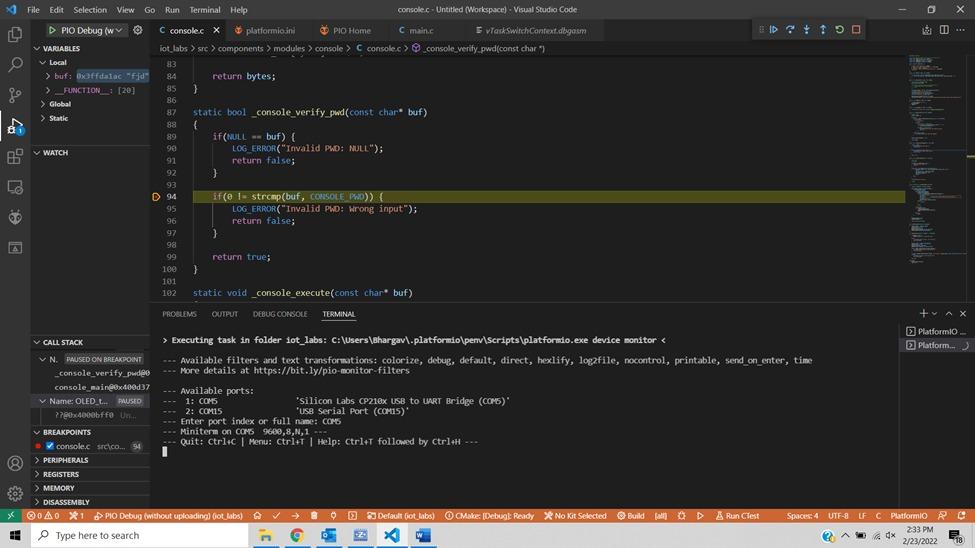


Fig 2 : Serial Monitor

## Securing JTAG:

1. How can JTAG be secured on the ESP32?

ESP32 supports irreversible and one-time hardware disable by writing to the JTAG\_DISABLE eFuse.

We can implement a secure boot configuration. Hardware secure boot support generates a device secure bootloader key (generated via hardware RNG, then stored read/write protected in efuse), and a secure digest. The digest is derived from the key, an IV, and the bootloader image contents.The secure digest is flashed at offset 0x0 in the flash.Depending on Secure Boot Configuration, efuses are burned to disable JTAG and the ROM BASIC interpreter (it is strongly recommended these options are turned on.)Bootloader permanently enables secure boot by burning the ABS\_DONE\_0 efuse. The software bootloader then becomes protected (the chip will only boot a bootloader image if the digest matches.)On subsequent boots the ROM bootloader sees that the secure boot efuse is burned, reads the saved digest at 0x0 and uses hardware secure boot support to compare it with a newly calculated digest. If the digest does not match then booting will not continue. The digest and comparison are performed entirely by hardware, and the calculated digest is not readable by software.

1. What are the three methods of securing JTAG in general? When might each be used?

**Disable**:

**Disabled JTAG**: This mode provides maximum security. All security-sensitive JTAG features are permanently blocked, preventing any debugging. This mode is not recommended as boundary scan is currently used by Digi on its manufacturing process and could affect the RMA procedure.

**Disabled debugging**: This mode disables debugging but leaves the boundary scan functionality enabled. This is the recommended mode for maximum security.

**Enabled**: This mode provides low security. This is the default mode of operation.

**Secure**: This mode provides high security. JTAG use is regulated by a 56-bit secret key-based challenge/response authentication mechanism.

References :

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/secure-boot-v1.html>

<https://www.digi.com/resources/documentation/digidocs/90001546/concept/trustfence/c_secure_jtag_android.htm>

## ESP-Prog Setup Guide:

1. To tell plaformio what debugger you want to use navigate to the platformio.ini there you will see the following lines.

[env:esp32dev]

platform = espressif32@1.4.0

board = esp32dev

framework = espidf

debug\_tool = esp-prog

...

There are a few things going on here but lets focus on the debug\_tool, this tells platformio what we want to use for debugging, in our case the esp-prog. We have to have this extra piece of hardware because the ESP32 does not have a built-in debugger even though we can flash it via our USB connection. Another thing that we can add is the upload\_protocol field which will tell platformio that we want to use the tool specified to flash the board. A full list of supported debuggers can be found [here](https://docs.platformio.org/en/latest/plus/debugging.html#tools-debug-probes).

Also be sure to slow the debug connection down by setting debug\_speed to 500.

2. It’s time to install the drivers for the esp-prog tool. (Ignore wiring examples as that will be covered in the next step.)

* **Windows**: [Step-by-step guide: Drivers, Zadig, Wiring](https://www.hackster.io/brian-lough/use-the-platformio-debugger-on-the-esp32-using-an-esp-prog-f633b6)  
  [Video tutorial](https://www.hackster.io/brian-lough/use-the-platformio-debugger-on-the-esp32-using-an-esp-prog-f633b6)
* **Mac**: macOS contains default FTDIUSBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) [AN134: FTDI Drivers Installation guide for MAC OS X](http://www.ftdichip.com/Support/Documents/AppNotes/AN_134_FTDI_Drivers_Installation_Guide_for_MAC_OSX.pdf)
* **Linux**: Please install “udev” rules [99-platformio-udev.rules](https://docs.platformio.org/en/latest/faq.html#faq-udev-rules). If you already installed them before, please check that your rules are up-to-date or repeat steps.

1. Now we need to connect the JTAG debugger to our ESP32. This can be done using the wires found in the bag with the JTAG debugger. Note only one wire will fit both the header on the IoT board and the JTAG debugger. This wire can also only be connected to either the header in one direction, as indicated by the slight protrusion which should serve as a guide.  
   JTAG Pins
2. Now that everything is wired up we need to tell platformio we want to debug our code, there are a few buttons we can press to do this:
   * Debug: Start debugging in the top menu.
   * Start Debuging option in the Quick Access menu.
   * Run in the debug view.  
     Text

     Description automatically generatedGraphical user interface, text

     Description automatically generated  
     As you can see above there are buttons to start, stop, step over, step out, restart, and stop.