|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| IOT REDBOOK  Hardware Reconnaissance |
| |  |  |  | | --- | --- | --- | | Navneet Mishra, Jacob Victor |  |  | |

**Table of Contents**

[Hardware Reconnaissance 2](#_Toc534807687)

[UART 5](#_Toc534807688)

[Identify UART Headers 5](#_Toc534807689)

[Identify UART Pinouts 7](#_Toc534807690)

[Identify GND (Multimeter continuity test) 7](#_Toc534807691)

[Identify Vcc (Multimeter Voltage test) 7](#_Toc534807692)

[Identify Tx (Multimeter Voltage test) 8](#_Toc534807693)

[Identify Rx (Multimeter Voltage test) 8](#_Toc534807694)

[Connect UART Pins to Laptop 9](#_Toc534807695)

[Identify Baud Rate 11](#_Toc534807696)

[JTAG 12](#_Toc534807697)

[Identify JTAG Headers 12](#_Toc534807698)

[Identify JTAG pinouts 14](#_Toc534807699)

[Connect JTAG Pins to Laptop 15](#_Toc534807700)

[Serial Peripheral Interface (SPI) 17](#_Toc534807701)

[Inter-Integrated Circuit (I2C) 17](#_Toc534807702)

[SPI and I2C Identification 17](#_Toc534807703)

[Serial Wire Debug (SWD) 19](#_Toc534807704)

[Reference 19](#_Toc534807705)

# Hardware Reconnaissance

Apart from the visual exterior analysis, reconnaissance consists of two steps –

1. opening the device and looking at the various chips present. Interesting components to look for on the PCB -

* UART
* JTAG
* I2C
* SPI
* SWD
* EEPROM
* NAND Flash

2. finding information from its datasheet

The complexity of opening up device can range from being extremely simple to highly complex depending on the device you are working with. In some of the devices, the screws are hidden beneath rubber pads on the legs, while in other cases they will be largely exposed, and in others still, the two different sections might be welded together.

Depending on how the device has been put together, use the appropriate tools to take apart the different sections. It is also recommended to have a good set of screwdrivers along with you for the entire hardware exploitation process, as varying devices will have many different kinds of screws used in them.

Once you have opened up the device, the next step is to look at the PCB and identify all the various chips present. To read the labels of the chip, use a USB microscope or your smartphone's flashlight while tilting the chip slightly. It is also recommended to have holders, which can hold the device steadily while you read the names of the various chips.

Once you have figured out the name of a chip, head over to Google and search for its manufacturer, followed by the model number and the word, "datasheet". Datasheet is a complete encyclopediaof a component. Once you have the datasheet with you, you can use the information present there to figure out all the various properties of the target chip, including the pinout, which would prove to be extremely useful during the hardware exploitation process.

Important information contained in a datasheet

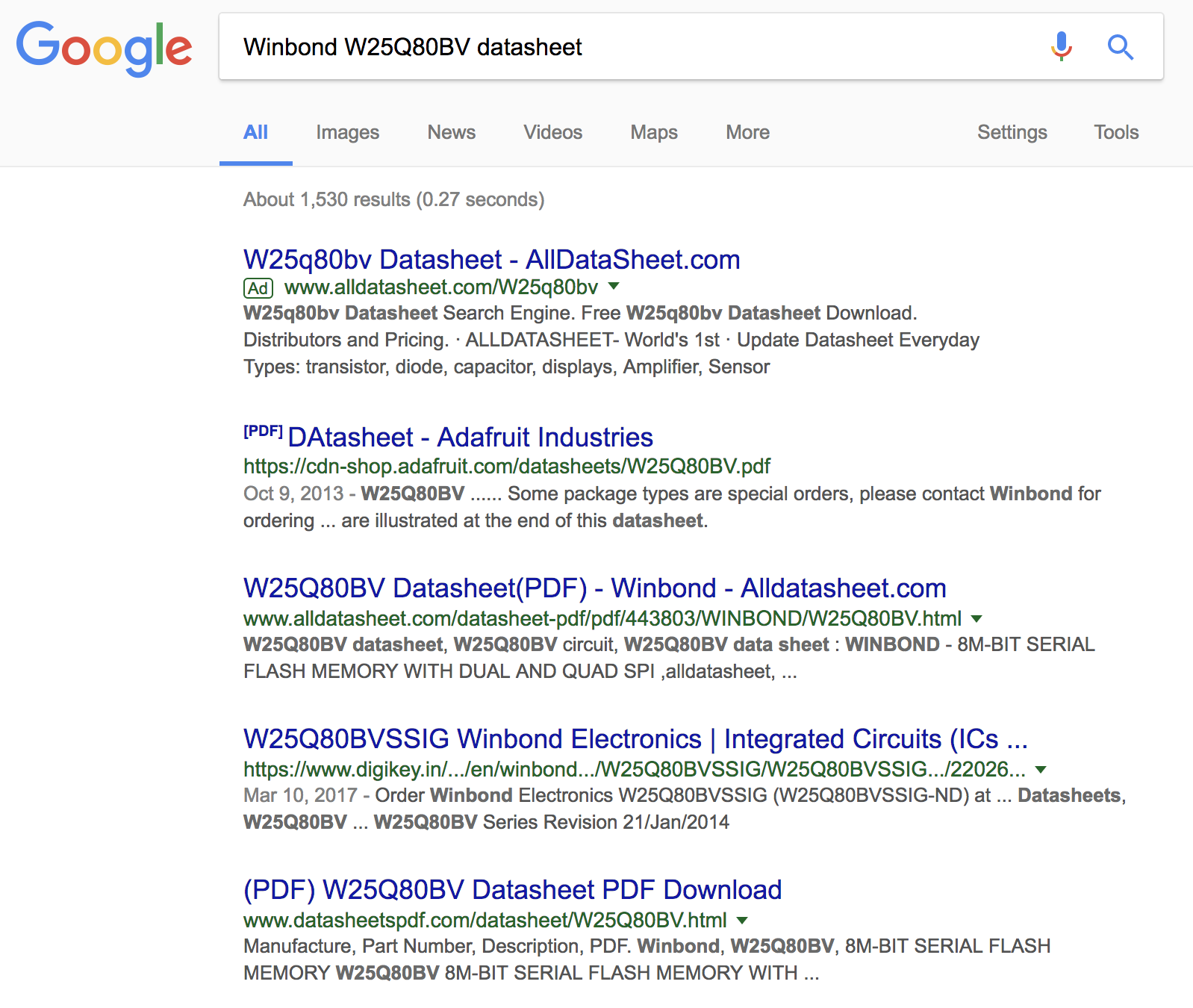
* Size (storage), Speed, Voltage
* Package specifications
* Pin Specifications
* Memory specifications
* Protocols supported (with communication specs at times)
* Applications
* Security/encryption information

The following is what an SPI flash chip looks like:



[Image source](https://cdn-shop.adafruit.com/1200x900/1564-00.jpg)

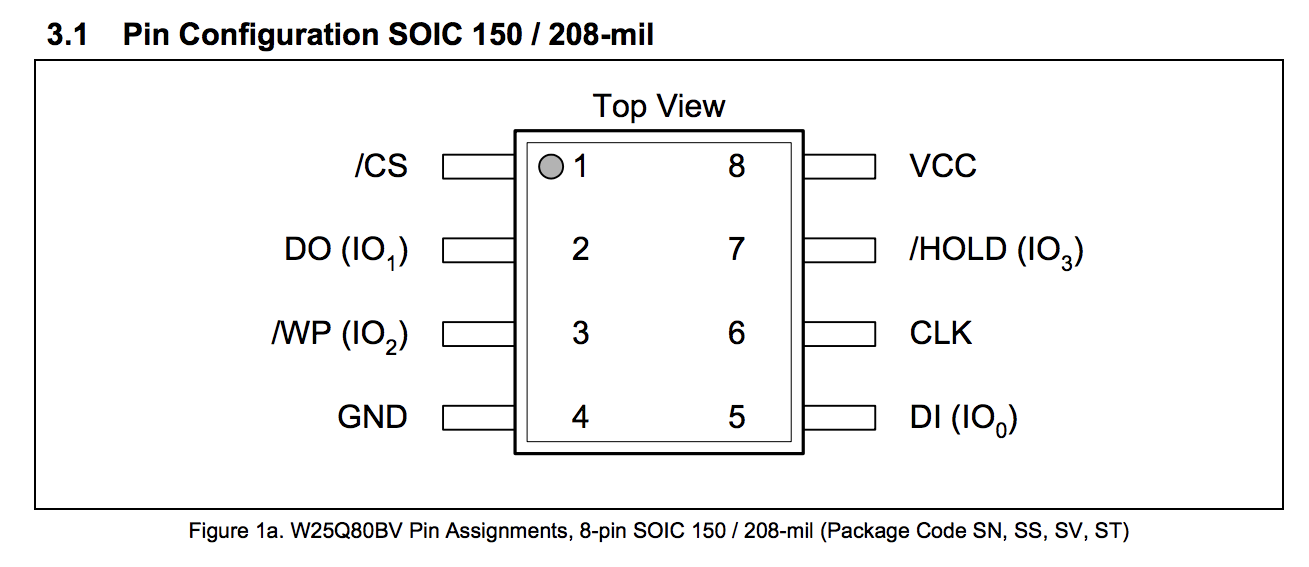
The flash chip in the preceding image has the label, Winbond W25Q80BV, which means that now we can look up its datasheet and identify its various properties-even without knowing that it's an SPI flash chip.



Open up any of the datasheet PDFs found in the search result. At the very start of the datasheet itself, we will find the following:



This means that our chip is an SPI flash chip with 8 MB of storage. As we go further in the datasheet, we also find out its pinouts, as shown in the following screenshot, which tells us the exact pinouts of the given SPI flash chip:



Thus, we have been able to correctly identify what that chip is meant for, what its properties are, and what are its pinouts.

**Note** – It is always recommended to get 2 hardware devices from the vendor as one of the device is mostly rendered unusable during opening up and testing the device.

Additionally, we can search through third-party sites such as https://fccid.io. An FCC ID is a product ID that is assigned by the FCC in order to keep track of wireless products in the market. Fccid.io is awesome and provides us with loads of detailed information on devices! The FCC publishes various design documents, datasheets, internal images, external images, test reports, various manuals, wireless frequencies, and more.

# UART

UART essentially converts the parallel data that it receives into a serial bit stream of data, which could be easier to interact with. It is one of the most common communication protocols found in embedded devices. It is one of the most common ways to gain access to devices. Manufacturers use UART for diagnostics, log messages, and as a debug console for verifying configurations when deploying devices, which makes it one of the most common sources of input in firmware. Since it's used for debugging, root access is commonly granted once connected. However, there are times when UART access is password protected, which may add extra time for brute-forcing.

UART contains about eight data lines with control pins and also has two serial wires which are the receive data and transmit data wires (RX/TX). No external clock is needed for UART. UART pinouts on the PCB are TX, RX, Vcc (voltage), and GND (ground). In order to connect to a UART, the TX, RX, and GND must be located using a multimeter. Sometimes, locating UART may be more difficult on some devices, than others. Some manufacturers may remove the UART header pins from the PCB, requiring soldering to take place. Manufacturers may also cover UART header pins with various layers of silkscreen and cover the headers with another integrated circuit which may be a bit of a pain.

Since the focus here is on reducing the number of lines, there is no clock present in a UART communication. Instead, UART relies on **baud rate**, which is the rate of data transfer. The two different components present in a UART communication will agree on a specified baudrate to ensure that the data is received in a proper format.

Additionally, in a UART communication, another bit called the **parity bit** is also added to the communication to facilitate error detection. Thus, a typical UART communication would have the following bits in order:

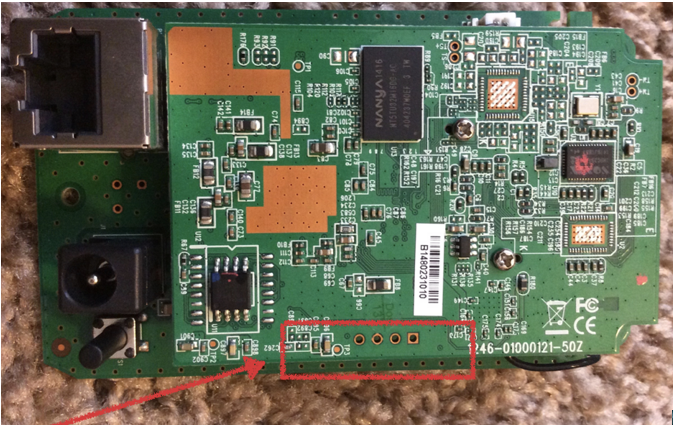
* **Start bit**: This indicates that this is the start of a UART communication.
* **Data bits**: This is the actual data that needs to be transmitted.
* **Parity bit**: This is used for error detection.
* **Stop bit**: This is used to indicate the end of the UART data stream.
* Multimeter
* Terminal emulator (minicom etc)

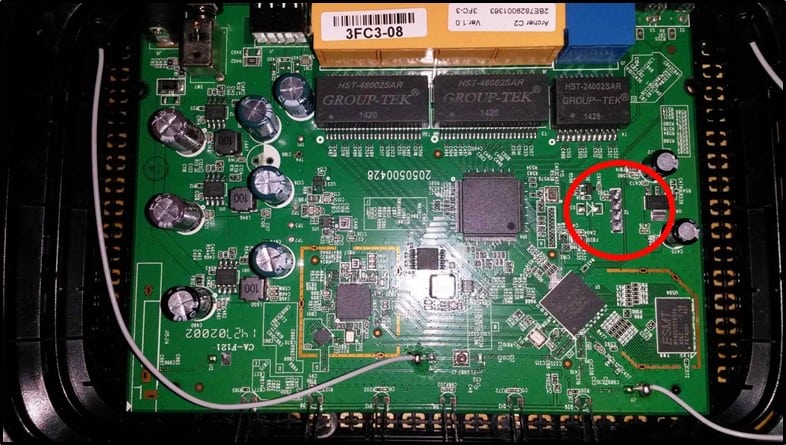
## Identify UART Headers

UART usually has 4 pins

* GND – Ground
* Rx – Receive
* Tx – Transmit
* Vcc – Voltage

If we find a set of 4 pins together, we can hope it’s a UART pins. As you can see in the following images, there are four pins next to each other, which can be UART pins.





Once we have identified 4 potential UART pins, next step is to identify Rx, Tx, GND and VCC.

## Identify UART Pinouts

### Identify GND (Multimeter continuity test)

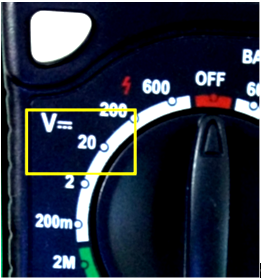
* Ensure that your device is turned off.
* Make sure that the pointer on the multimeter points to the speaker symbol, as shown in the following image:



* Identify any metallic sheet area
* Place the black probe on a ground surface, this could be any metallic surface on the device.
* Put the red probe on the pin to be tested
* If the multimeter makes continuous beep it is a GND pin
* If not, repeat with other pins till you find one

### Identify Vcc (Multimeter Voltage test)

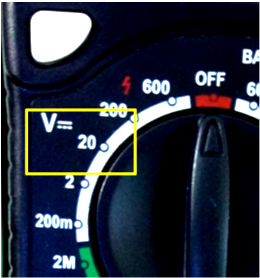
* Power on the device
* Put the multimeter pointer to the V - 20 position (assuming voltage under 20), as now we are going to measure the voltage.



* Put the red probe on the pin to be tested
* Put the black probe on the identified GND pin or GND of input supply
* If the multimeter displays fairly constant voltage (for ex. 3.3) it is Vcc
* If not, repeat with other pins till you find one

### Identify Tx (Multimeter Voltage test)

* Power on the device and immediately do the multi-meter test
* Point the rotary switch to V (20) Assuming voltage under 20

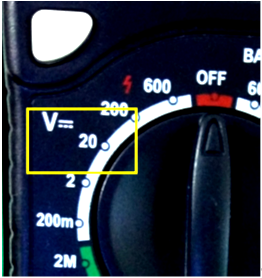


* Put the black probe on the metallic area
* Put the red probe on the pin to be tested
* If the multimeter displays varying voltage it is likely a Tx pin
* If not, repeat with other pins till you find one

**Reason**: Since the Tx pin transmits data, the voltage will keep on changing as and when bits are transferred. When the device is powered on it transmits a lot of boot up/log messages andhence we are likely to find variation in voltage (for ex. 3.3-2.5-1.6-3.2 etc)

### Identify Rx (Multimeter Voltage test)

* Power on the device and immediately do the multi-meter test
* Point the rotary switch to V (20) Assuming voltage under 20



* Put the black probe on the metallic area
* Put the red probe on the pin to be tested
* In some cases, it will show constant voltage either low or high (Generally, we get constant low voltage)
* In some cases, it will show varying voltage
* If not found, repeat with other pins, if any left, till you find one

**Note**

* If we identify other 3 pins successfully, and there are only 4 pins then the one left is Rx ;)
* Little difficult to identify Rx as there are no specific traits

## Connect UART Pins to Laptop

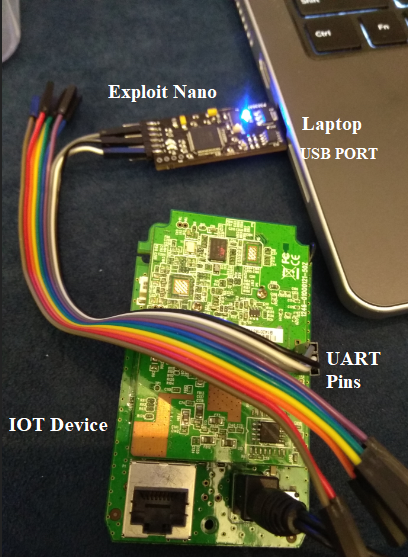
Hardware required

* Expliot Nano ([Payatu](https://payatu.com/))
* Cables, breakaway headers

1. Now that we have identified the pinouts, the next step is to connect the pinouts to Expliot Nano using wires and breakaway headers.

* Expliot Nano TX <------------ > RX UART Pin on board
* Expliot Nano RX <------------ >TX UART Pin on board
* Expliot Nano GND <------------> GND UART Pin on board
* Don’t connect Vcc.





**Note** - You can also use other devices in place of Expliot Nano like Attify Badge, USB-TTL or Adafruit FT232H.

## Identify Baud Rate

Once we have made all the connections, the next step is to figure out the baudrate on which the IOT device operates.

1. Most devices have default baudrate of 115200.

2. The port can be accessed via /dev/ttyUSB0.

3. Power on the machine and immediately run baudrate.py ([Github](https://github.com/devttys0/baudrate/blob/master/baudrate.py))

baudrate.py -p /dev/ttyUSB0

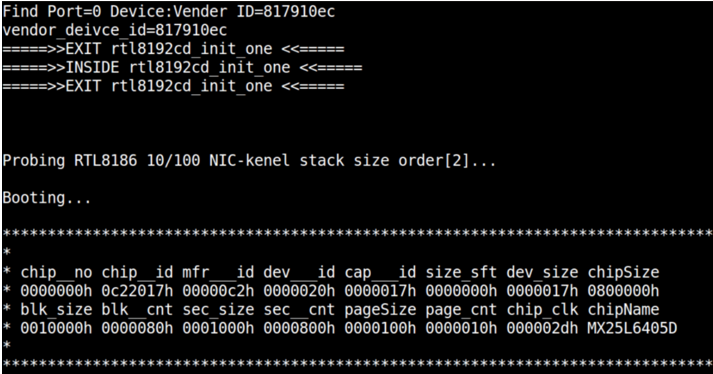
4. Down arrow key will cause baudrate.py to shift to a lower baudrate.

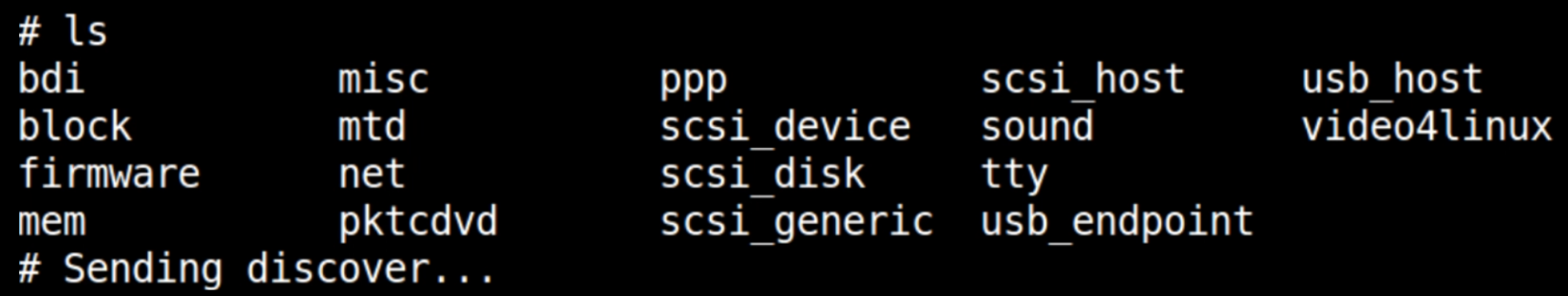
5. Up arrow keywill cause baudrate.py to shift to a higher baudrate.

6. It starts with 115200. if you see garbage data press down arrow and test with lower baudrate, repeat it till you find the correct baudrate.

**Note** – You can also run Baudrate.py -a (auto detect mode). So you dont have to press Up/Down arrow keys.

It should look something like the following screenshot -



Next, hit Ctrl + C, which will take you to the minicom utility using the identified settings. Hitting Enter here would grant you shell access, given that your target device has a UART-based shell:

To exit from a minicom session Press Ctrl+A and then X.

# JTAG

JTAG is another serial communication under IEEE 1149.1. It was created for chip-and system level testing. Manufacturers use JTAG as a source of debugging, similar to UART. There is the ability to password protect JTAG access, but the BYPASS mode should still work. Firmware can be dumped for analysis or upgraded using JTAG. It provides a direct interface to hardware on the board which means it can access devices connected to it, such as flash or RAM. JTAG connects to an on-chip test access port (TAP) which regulates a state when accessing registers on chips. Similar to UART, manufacturers may obfuscate header pins or traces.

JTAG is a simplified way of testing pins and debugging them. It allows device developers and testers to ensure that each of the pins in the various chips on the device are functional, interconnected, and operational as intended.

For penetration testers, JTAG serves a number of purposes, ranging from giving us the ability to read/write data and even debug running processes, and modifying the program execution flow.

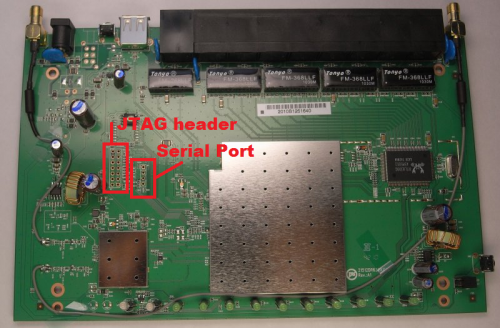
The four most important pins in JTAG are -

* test data in (TDI)
* test data out(TDO)
* test clock (TCK)
* test mode select (TMS)

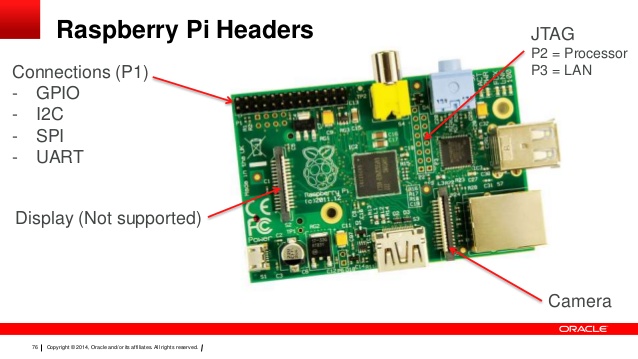
## Identify JTAG Headers

Before identifying these individual pinouts, we must first identify where the JTAG headers are located on the device.

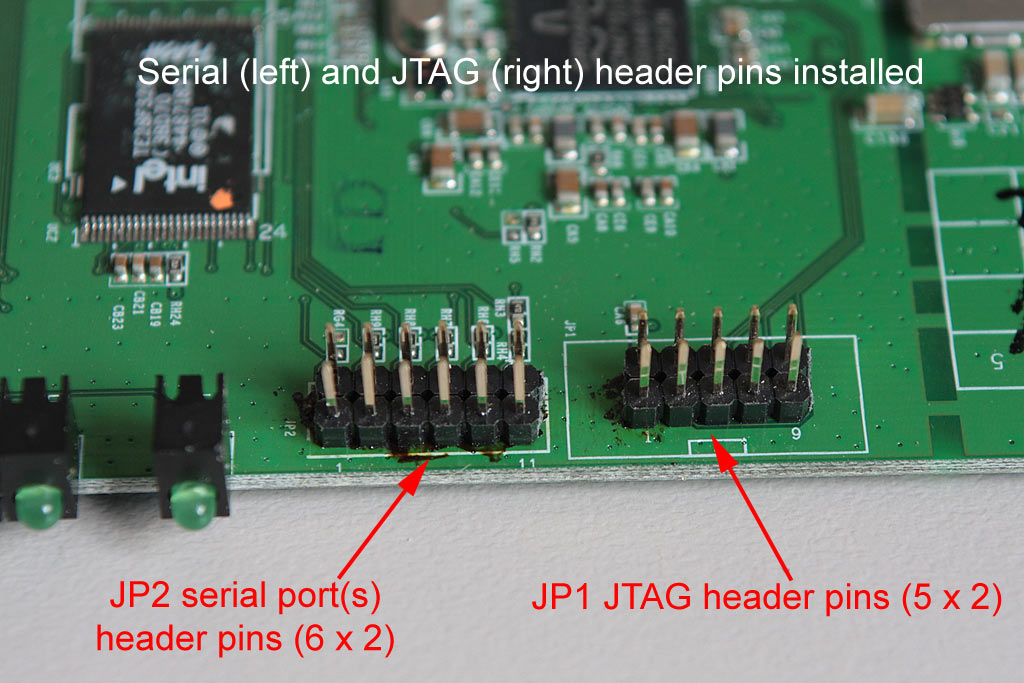
To make things easier, JTAG comes in a couple of standard interface options such as 13 pins, 14 pins, 20 pins, and so on. The following are some of the images of JTAG interfaces in real-world devices:



[Image source](https://www.dd-wrt.com/wiki/images/thumb/9/99/DLINK-DIR632_Board.png/500px-DLINK-DIR632_Board.png)



[Image Source](https://image.slidesharecdn.com/rpi-java8-uni-141123214849-conversion-gate02/95/raspberry-pi-with-java-8-76-638.jpg?cb=1416779506)



[Image source](http://www.karldawson.com.au/header_pins_installed.jpg)

**Note** - Even though you might be able to find the JTAG laid out in the standard header format, in some of the real-world devices, you will find the JTAG pins scattered all across the board instead of being at a single location. In these cases, you will need to solder headers/jumpers on them and connect them to a JTAGulator to be able to identify if they are JTAG pinouts, and which pin corresponds to what JTAG pin.

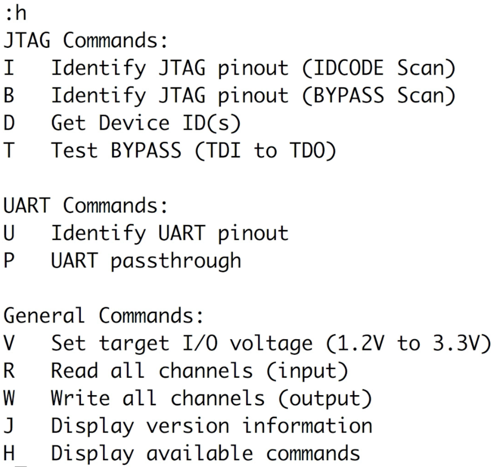
## Identify JTAG pinouts

Once you have connected all the JTAGulator channels to the expected JTAG pinouts on the target device, additionally connecting the GND to GND.

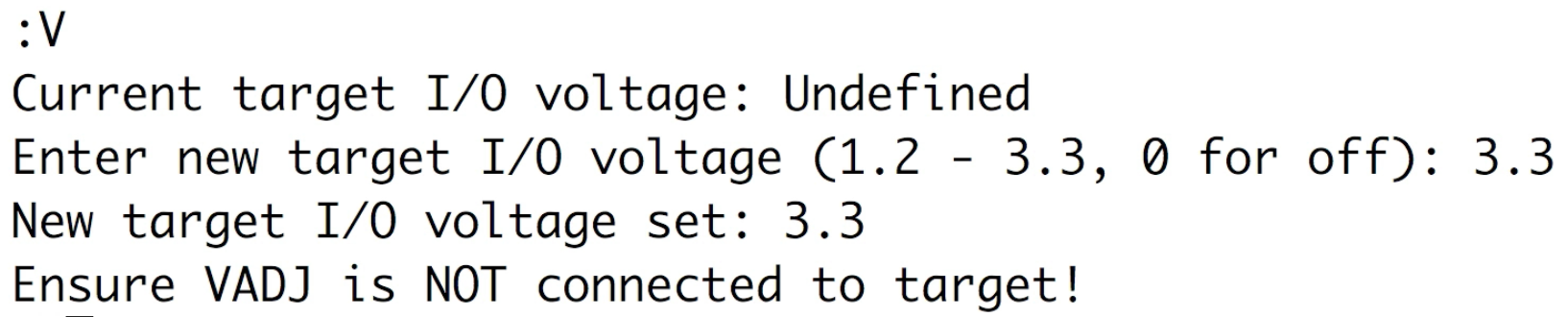
1. Launch the screen using the following command

sudo screen /dev/ttyUSB0 115200

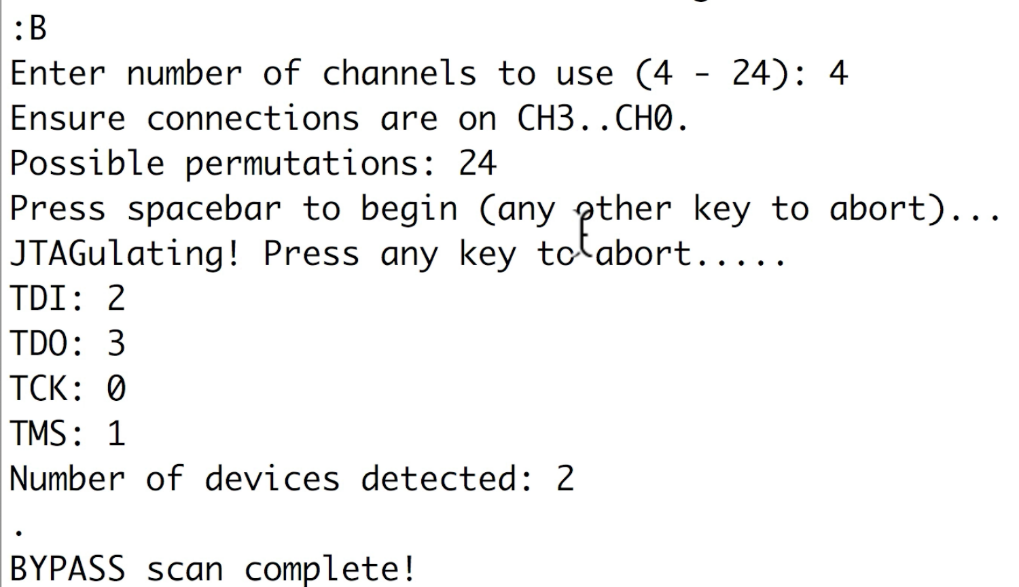
2. Then, you will be granted with a JTAGulator prompt, as shown in the following screenshot:



3. The first thing that we will do here is set our target device's voltage, which in the current scenario is 3.3. To do this, simply type V followed by 3.3 as shown in the following screenshot:



4. Once we have set the target voltage, we can then run a bypass scan by hitting *B* to figure out the JTAG pins in our current connection.



As you can see, JTAGulator was able to identify the JTAG pinouts and tell us what the individual pins correspond to.

## Connect JTAG Pins to Laptop

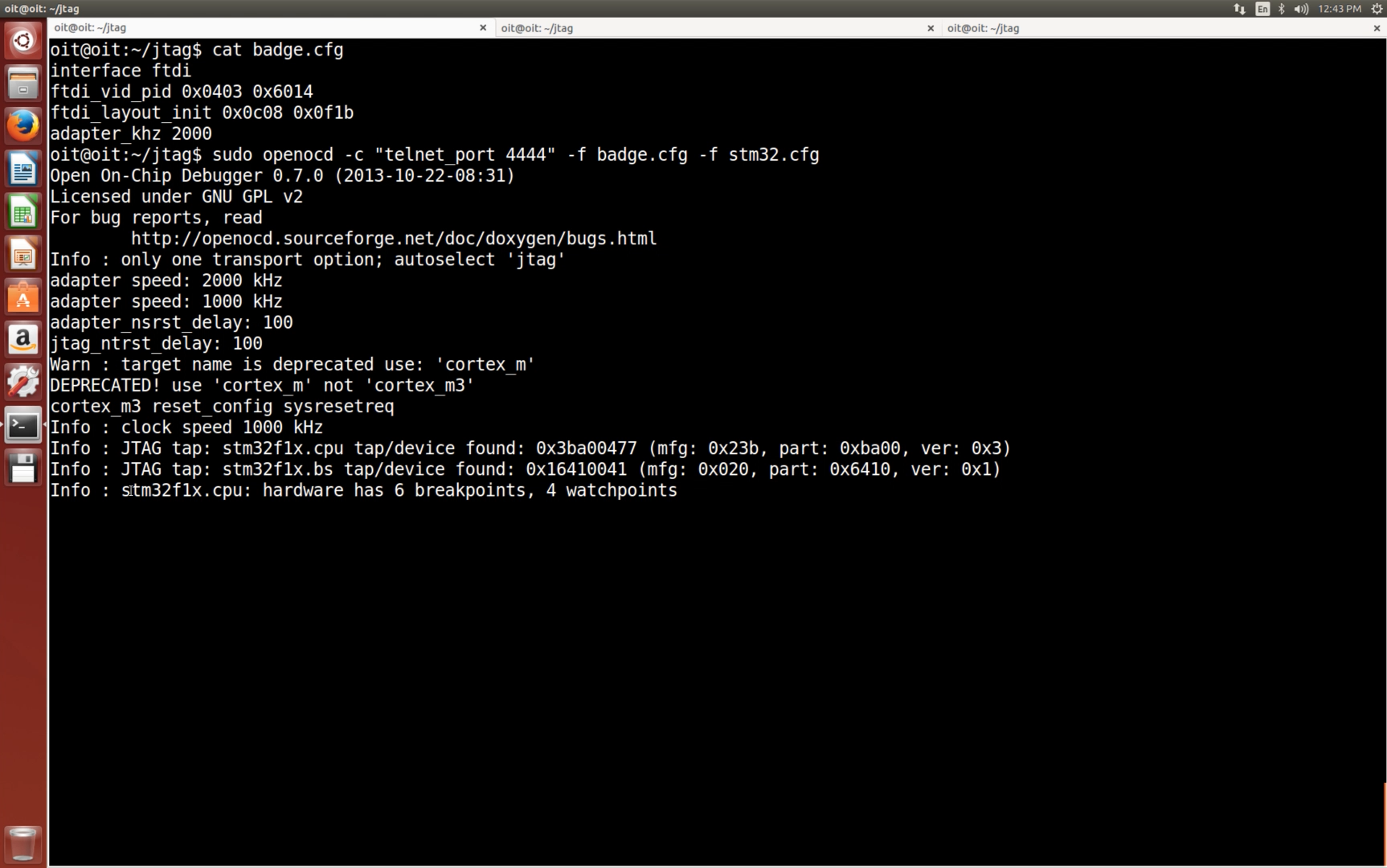
Hardware required - Attify Badge (or FTDI C232HM MPSSE cable)

1. Now that we have identified the pinouts, the next step is to connect the pinouts to as shown next:

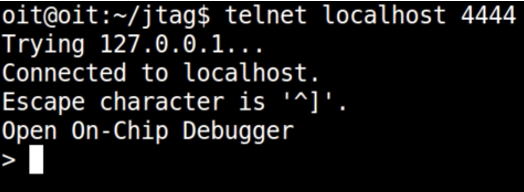
* The TDI of the target goes to the D1(TDI) of Attify Badge (or the Yellow of the FTDI cable)
* The TDO of the target goes to the D2 (TDO) of Attify Badge (or the Green of the FTDI cable)
* The TMS of the target goes to the D3 (TMS) of Attify Badge (or the Brown of the FTDI cable)
* The TCK of the target goes to the D0 (TCK) of Attify Badge (or the Orange of the FTDI cable)

2. Once you have made the required connections, the next step is to run Open On-Chip Debugger ([OOCD](http://openocd.org/)) using the configuration files for Attify Badge (or the FTDI C232HM MPSSE cable) and the target device's chip. The configuration files can be obtained from the OpenOCD directory after installation and are located at openocd/tcl/target.

3. OpenOCD can be run as shown in the following screenshot:



4. As you can see, OpenOCD has identified both the devices in the chain and it has also enabled Telnet on port 4444, which we can now connect to, as shown in the following screenshot:



At this step, you can perform all the various OpenOCD commands, as well as the commands specific to your given chip, in order to compromise the device.

# Serial Peripheral Interface (SPI)

It is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

Advantages of SPI:

* It’s faster than asynchronous serial
* The receive hardware can be a simple shift register
* It supports multiple slaves

Disadvantages of SPI:

* It requires more signal lines (wires) than other communications methods
* The communications must be well-defined in advance (you can’t send random amounts of data whenever you want)
* The master must control all communications (slaves can’t talk directly to each other)
* It usually requires separate SS lines to each slave, which can be problematic if numerous slaves are needed.

# Inter-Integrated Circuit (I2C)

The Inter-Integrated Circuit (I2C) Protocol is a protocol intended to allow multiple “slave” digital integrated circuits (“chips”) to communicate with one or more “master” chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information. Communication via I2C is more complex than with a UART or SPI solution. The signaling must adhere to a certain protocol for the devices on the bus to recognize it as valid I2C communications. Fortunately, most devices take care of all the fiddly details for you, allowing you to concentrate on the data you wish to exchange.

# SPI and I2C Identification

SPI and I2C identification is similar to what we saw in the UART communication identification. One of the ways of identifying that the communication protocol being used is SPI or I2C is by using a logic analyzer and looking at the various bits that have been transmitted in the communication.

Both SPI and I2C fall under serial communication, mostly used in Flash and EEPROM. One of the ways to correctly identify the exact protocol being used, along with further details, is to look at the chip name and get the information from the datasheet.

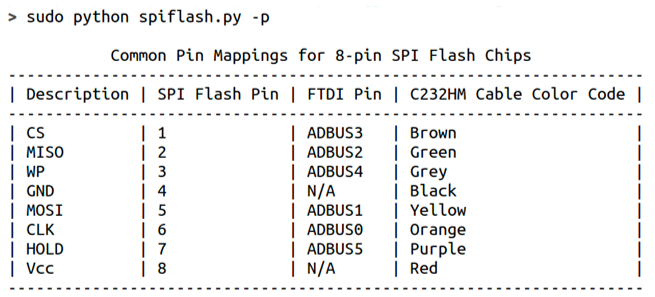
Once we have this information ([see here](#spi_datasheet)), we can connect to the SPI flash chip using Attify Badge.

1. data out (DO) to MOSI in the Attify Badge (or DI green of the FTDI cable)

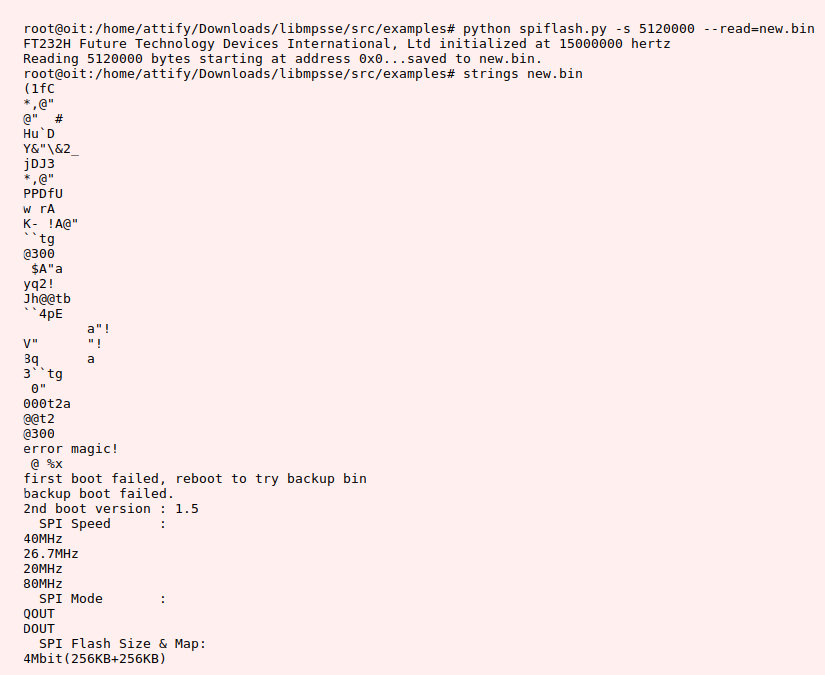
2. data in (DI) to MISO in the Attify Badge (or DO yellow of the FTDI cable)

3. Additionally, also connect the Vcc, GND, WP, and CS of the cable to the same pins on the chip.

The table in the following figure will help you make the connections at this stage:



Once the connections are made, all we have to do is run the spiflash.py utility from the [LibMPSSE library](https://github.com/devttys0/libmpsse)



The size in the preceding syntax was obtained from the datasheet of the flash chip, and the entire content of the flash chip was put into a file called new.bin.

Thus, now you can look at a SPI flash chip, find out its pinouts, and dump data from it, which could be firmware, hardcoded keys, or other sensitive information, depending on the device that you are working with.

# Serial Wire Debug (SWD)

It provides a debug port for severely pin-limited packages, often the case for small package microcontrollers but also complex Application-Specific Integrated Circuit (ASIC), where limiting pin count is critical and can be the controlling factor in device costs.

SWD replaces the 5-pin JTAG port with a clock + single bi-directional data pin, providing all the normal JTAG debug and test functionality plus real-time access to system memory without halting the processor or requiring any target resident code. SWD uses an Arm standard bi-directional wire protocol, defined in the Arm Debug Interface v5, to pass data to and from the debugger and the target system in a highly efficient and standard way. As a standard interface for Arm processor-based devices, the software developer can count on a wide choice of interoperable tools from Arm and third party tool vendors.

# Reference

* <https://www.packtpub.com/networking-and-servers/iot-penetration-testing-cookbook>
* <https://payatu.com/>
* <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>
* <https://learn.sparkfun.com/tutorials/i2c/all>