Brandon Tang’s 4 Week Internship at DSTA 2019

Analysis of Syslink E1200v2 Router

Section 0: Overview of Internship Work

During this internship, I look at various hardware debugging interfaces and attempt the 3 Cs of “connect”, “communicate”, “control” on a practise target, an old network router.

Section 1.0: External Information Gathering

To understand the avenues of gaining access to the router, we need to first learn as much about it as possible. The most basic method of which is to do external analysis without dissassembly of the router.

There are 3 main avenues of external information gathering of a router.

1. Physical examination of the router and its box (if provided), including whatever labels and printing are on the router
2. Analysis of the administrative webpage of the router, found by connecting to its wifi and going to the ip address of the router.
3. Online searching for information about the router. Main areas to search are on Google, FCC.io, the syslink support website.

Information from Physical Examination
Brand: Cisco
Model: Linksys E1200 (v2)
Serial Number: 10820C63242832
MAC Address: 20AA4B3CCEB4
Router Pin for WPS : 8722-7482
FCC ID: Q87-E1200V2
IC: 3839A-E1200V2

Information from Router Admin Page
http://192.168.1.1/
Username: admin
Password: admin
Firmware Version: 2.0.01 build 1 Nov 10, 2011
Firmware Verification: f7fffb6734c2effc66bd181bb3544c31
Internet MAC Address: 20:AA:4B:3C:CE:B4
Device Name: Cisco42832

Router IP address: 192.168.1.1
Subnet Mask: 255.255.255.0
IPv6 Link-Local Address: fe80::22aa:4bff:fe3c:ceb3

Network Name (SSID):   Cisco42832
Channel Width: 20MHz
Channel: 1

Information from Online Sources
Firmware Download: https://www.linksys.com/us/support-article?articleNum=148523 User Manual: https://downloads.linksys.com/downloads/userguide/E_Series_UG_E900Rev_3425-01486_Web.pdf Internal Photos https://fccid.io/Q87-E1200V2/Internal-Photos/Internal-Photos-1564096 Information about the internals http://en.techinfodepot.shoutwiki.com/wiki/Linksys_E1200_v2

Section 1.1: Internal Information Gathering

To actually get into the internals of the device, we need to first disassemble it. However, we need to be careful to not break or cut into the printed circuit board (PCB) while doing so. Furthermore, it is good practise to keep the parts in a state where they can be reassembled.

We refered to the internal photos from FCC.io (refer to Fig 1) to know where the position of the screws and the PCB was. This aided us in disassembling the router more efficiently and without fear of damaging the internals.

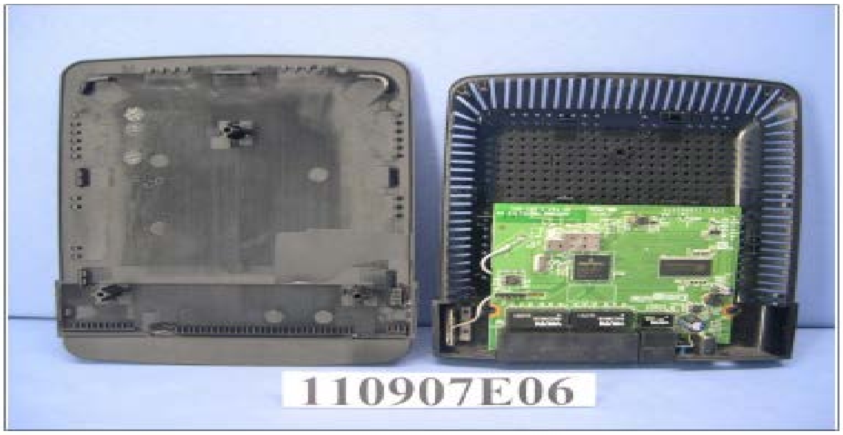


Figure 1: Photo from FCC ID shows screw and PCB positions

From there, we were able to successfully disassemble the router. The next step is to identify the different components attached to the PCB and how they connect to each other.

This was first done by reading the part number off the various components and then searching the internet for information about the part. Ideally, we should aim to find the datasheet for the part as it will contain anything and everything there is to know about the part. Some parts (such as the flash chip) can be very small and thus it is difficult to read their part numbers. This can be overcome by shining more light with a flashlight, using the digital zoom of a phone camera, or if necessary, using a magnifying glass or microscope.

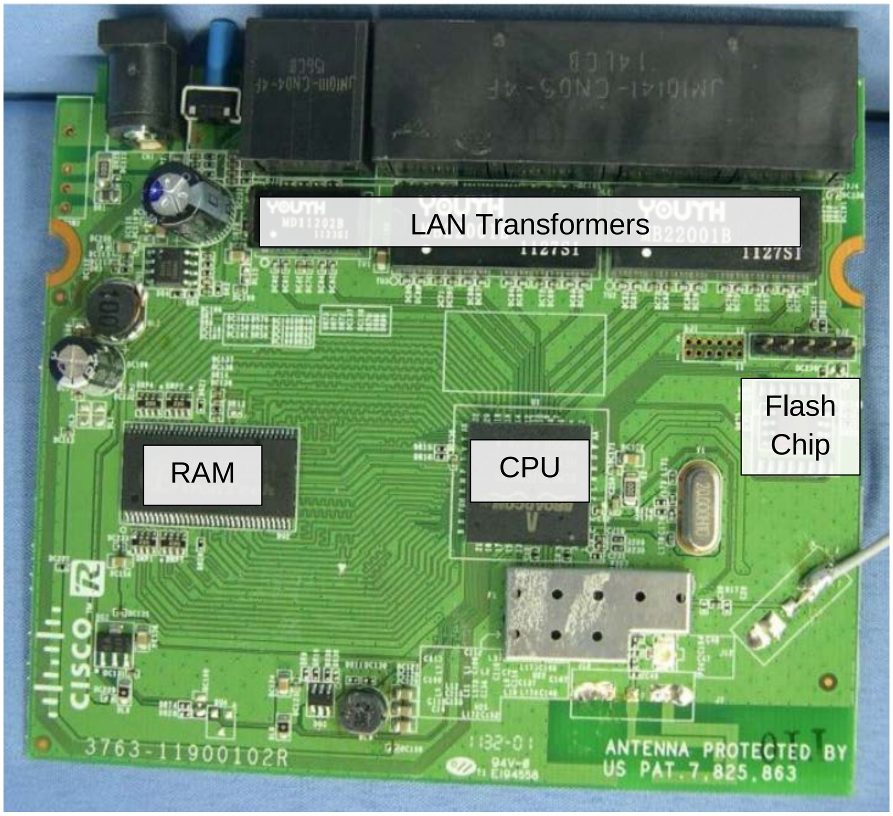


Figure 2: Relevant Internal Components of E1200v2 PCB

Component (Part Number)	Function	Documentation
FPE H12106DK-R	10/100base-T 1:1 transformer	http://www.fpe.com.cn/ch_tw/pdf/PDF/10-100-9.pdf
Youth MB22001B	10/100 Base-T Dual LAN Transformer	
FR9882	Step-down DC/DC converter	http://www.dzsc.com/uploadfile/company/704680/2011217115251271.pdf
W9425G6JH-5	CMOS Double Data Rate synchronous dynamic random access memory (DDR SDRAM)	https://www.acalbfi.com/uk/Semiconductors/Memory-Storage/SDRAM/p/4M-x-4-Banks-x-16-Bits-DDR-SDRAM/00000004K6 http://c1170156.r56.cf3.rackcdn.com/UK_WND_W9425G6JH-5_DS.pdf
25Q64BVSIG	SPI Flash Chip	https://www.datasheetq.com/datasheet-download/853500/1/Winbond/25Q64BVSIG

BCM5357	Wireless LAN (WLAN) router System-on-a-Chip (SoC)	https://www.broadcom.com/products/wireless/wireless-lan-infrastructure/bcm5357

Table 1: List of Components on PCB of E1200v2

From the list of components found, we drew a logic block diagram and noted to potential points of entry.

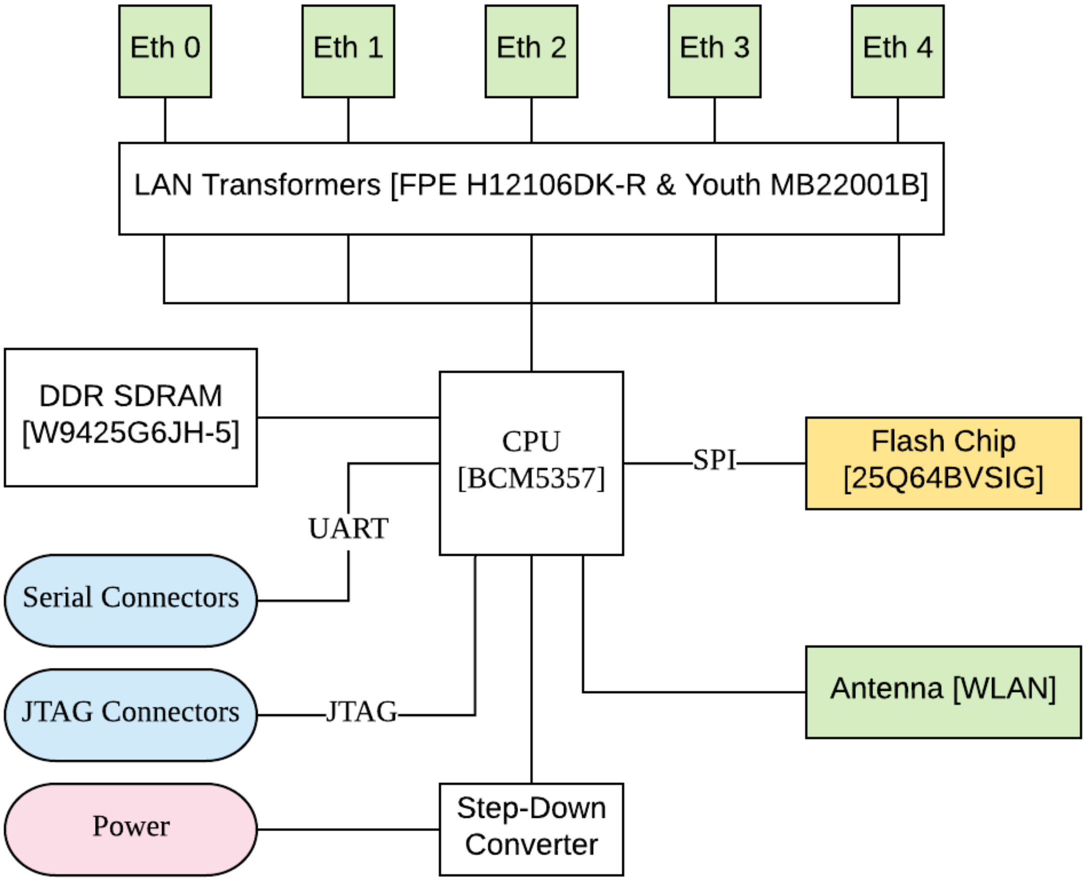


Figure 3: Logic Block Diagram of PCB Components and Ports

[Points of Entry are highlighted in colour]

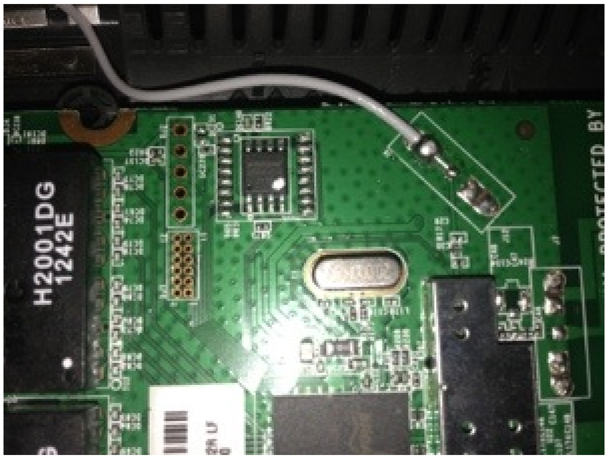
Section 2.0: Preparing Universal Asynchronous Receiver/Transmitter (UART) Port for Connection

During internal information gathering, we identified the serial connector on the PCB that uses UART to communicate. Now we will attempt to tap into the port to communicate with the CPU.

There are 5 pins on the connector, however, there are only 3 pins that are necessary for the connection.

1. The transmitter (TX)
2. The receiver (RX)
3. Common ground (GND)

From an online resource[1], we found the following pinout for the serial port.



() ?

() TX

() RX

() ?

() ground

Figure 4: Layout of Serial Port

However, as the information was merely found from a blog online, we needed to verify this pinout. To verify ground, we used the multimeter to check for connectivity between the various pins and connections we know are at common ground. Areas at common ground are generally metal shieldings (which the E1200 has one) and the negative terminal of capacitors. To verify the TX port, we use the oscillosope. As the CPU boots up, we should see output on the TX pin. The pin with a “square” copper plating is generally for voltage (VCC). That just leaves the RX pin and some other pin. The RX pin is high while waiting for the start bit of a data packet, so we use the multimeter to figure out which of the remaining pins is at high voltage.

After the analysis, we figure that the online pinout is correct and we can now solder on jumper wire headers onto the holes. These jumper wire headers enable us to efficiently plug our jumper cables into the port when we want to connect.

[1] "Linksys E900 serial port pinout – Going on my way…." Tomcsanyi, July 26, 2013, https://domonkos.tomcsanyi.net/?p=398. Accessed December 18, 2019.

Figure 5: Male-Male Jumper Wire Headers

Section 2.1: Communicating and Controlling with UART

To actually communicate with the UART chip on the target board, we needed a device that was able to act as an interface to allow our computer to utilise the UART protocol. For this section, we decided to use the UM232H-B USB to Serial/Parallel Break-Out Module from Future Technology Devices International Ltd (FTDI). For future reference, the UM232H-B uses the FT232H Single Channel HiSpeed USB to Multipurpose UART/FIFO IC.

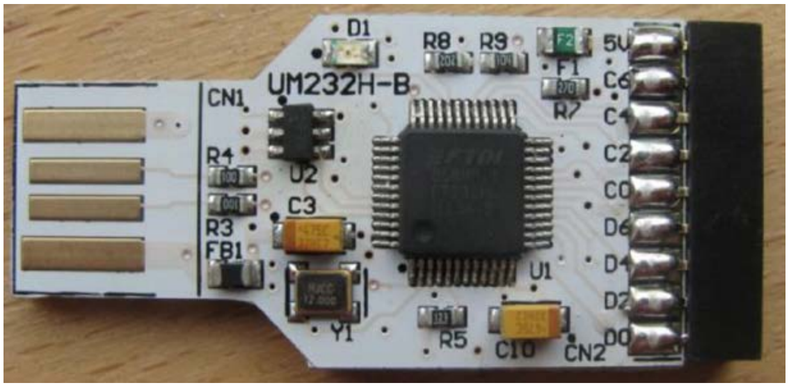


Figure 6: UM232H-B Module

By refering to the datasheet[2], we know that the D0 and D1 ports on the UM232H-B correspond to the TX and RX port when the UM232H-B is used for UART communication. From there, we hook up the serial port on the target board with the UM232H-B via 3 male-female jumper cables and the hardware set-up is finish.

On the software side, we need a UART client to be able to interface with the UM232H-B. For windows, there is Putty[3] and TeraTerm[4], of which I choose to user TeraTerm

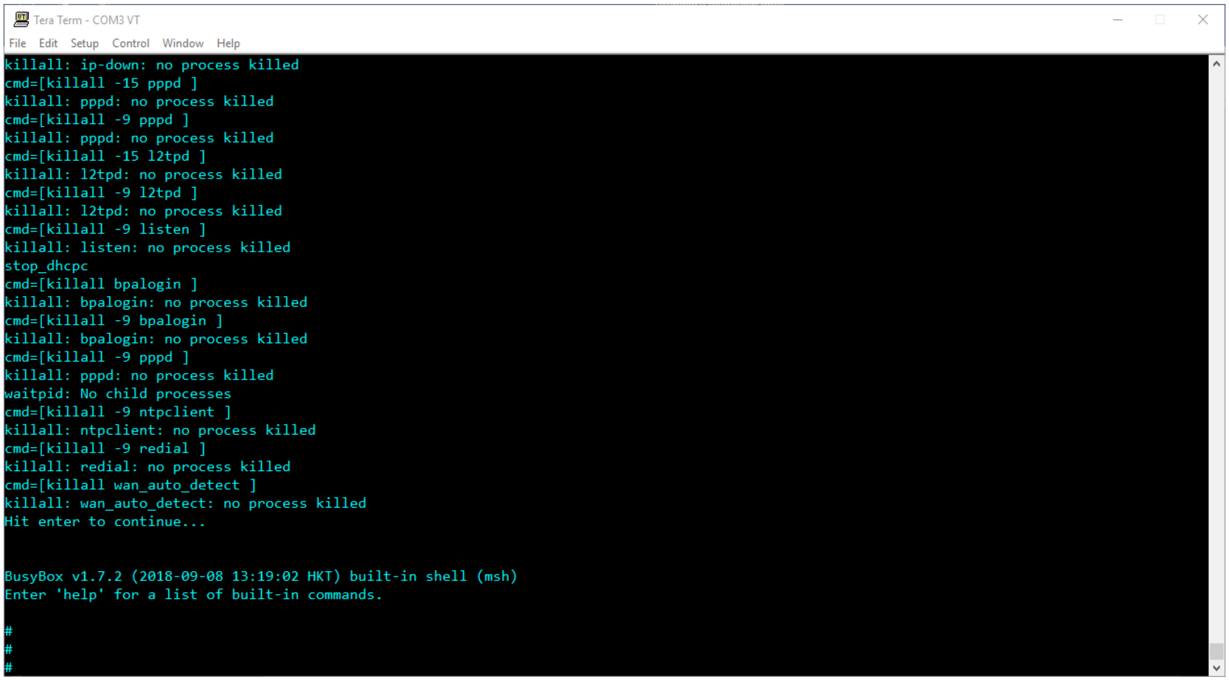
[2] https://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UM232H-B.pdf

[3] "Download PuTTY." a free SSH and telnet client for Windows, https://www.putty.org/. Accessed December 18, 2019.

[4] “Tera Term - Terminal Emulator for Windows.” Tera Term Open Source Project. Accessed December 18, 2019. https://ttssh2.osdn.jp/index.html.

(personal preference). For linux users, there is minicom[5] and picocom[6]. I do not recommend the use of minicom as we were able to receive transmissions from our target board but were unable to send keystrokes through minicom.

We ended up using TeraTerm for most of the UART sessions. To configure the UART communication, we have to specify the baud rate, the number of databits, number of parity bits and the number of stop bits per packet of data transferred. Details for these parameters were found on the same website as in Figure 4. It is good to note that the baud rate can also be determined by taking the reciprocal of the bit time found from the oscilloscope trace of the TX pin.



```
killall: ip-down: no process killed
cmd=[killall -15 pppd ]
killall: pppd: no process killed
cmd=[killall -9 pppd ]
killall: pppd: no process killed
cmd=[killall -15 l2tpd ]
killall: l2tpd: no process killed
cmd=[killall -9 l2tpd ]
killall: l2tpd: no process killed
cmd=[killall -9 listen ]
killall: listen: no process killed
stop_dhcpc
cmd=[killall bpalogin ]
killall: bpalogin: no process killed
cmd=[killall -9 bpalogin ]
killall: bpalogin: no process killed
cmd=[killall -9 pppd ]
killall: pppd: no process killed
waitpid: No child processes
cmd=[killall -9 ntpclient ]
killall: ntpclient: no process killed
cmd=[killall -9 redial ]
killall: redial: no process killed
cmd=[killall wan_auto_detect ]
killall: wan_auto_detect: no process killed
Hit enter to continue...


BusyBox v1.7.2 (2018-09-08 13:19:02 HKT) built-in shell (msh)
Enter 'help' for a list of built-in commands.

#
#
#
```

Figure 7: BusyBox Rootshell via UART After Kernal Loads

From the TeraTerm terminal, we turn the target power off and on and watch the boot process. We can then see the boot log[7] and we are eventually dropped into a root shell. From there we proceed to look around and collect more information[8].

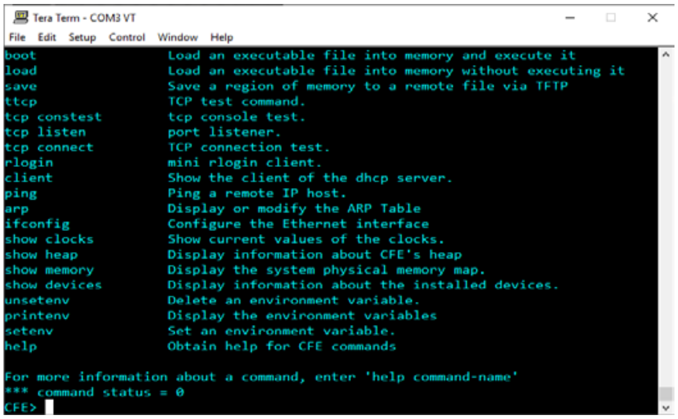
[5] "minicom(1)." Linux man page, https://linux.die.net/man/1/minicom. Accessed December 18, 2019.

[6] "picocom(8)." Linux man page, https://linux.die.net/man/8/picocom. Accessed December 18, 2019.

[7] Refer to Annex A

[8] Refer to Annex B

To see what else we can do, we try to get into the bootloader. By spamming control-c or escape as the system boots up, we manage to escape the boot sequence and we are dropped into a Common Firmware Environment (CFE) shell[9].



```
boot                   Load an executable file into memory and execute it
load                   Load an executable file into memory without executing it
save                   Save a region of memory to a remote file via TFTP
ttcp                   TCP test command.
tcp constest           tcp console test.
tcp listen             port listener.
tcp connect            TCP connection test.
rlogin                 mini rlogin client.
client                 Show the client of the dhcp server.
ping                   Ping a remote IP host.
arp                    Display or modify the ARP Table
ifconfig               Configure the Ethernet interface
show clocks            Show current values of the clocks.
show heap              Display information about CFE's heap
show memory            Display the system physical memory map.
show devices           Display information about the installed devices.
unsetenv               Delete an environment variable.
printenv               Display the environment variables
setenv                 Set an environment variable.
help                   Obtain help for CFE commands

For more information about a command, enter 'help command-name'
*** command status = 0
CFE>
```

Figure 8: CFE Shell via UART

Section 3.0: Preparing to Communicating with the Flash Chip Over Serial Pheripheral Interface (SPI)

The next task we decided to embark on was to communicate with the W25Q64BVSIG SPI flash chip on the PCB to see if we could dump the firmware of the flash chip.

However, to ensure that we dumped the firmware correctly, we first updated the firmware from “2.0.01 build 1” to “Ver. 2.0.11 (build 1)” such that we are able to compare the firmware we eventually dump to the firmware we downloaded from the syslink support page for the E1200[10].

As similarly to UART, we begin by identifying which legs of the flash chip correspond to which connections in the SPI protocol.

The SPI protocol involves 4 basic data tranfer lanes:

1. Chip Select
2. Clock
3. Master In - Slave Out
4. Master Out - Slave In

[9] Information found is in Annex B

[10] “Linksys Official Support." E1200 Downloads, https://www.linksys.com/us/support-article?articleNum=148523. Accessed December 18, 2019.

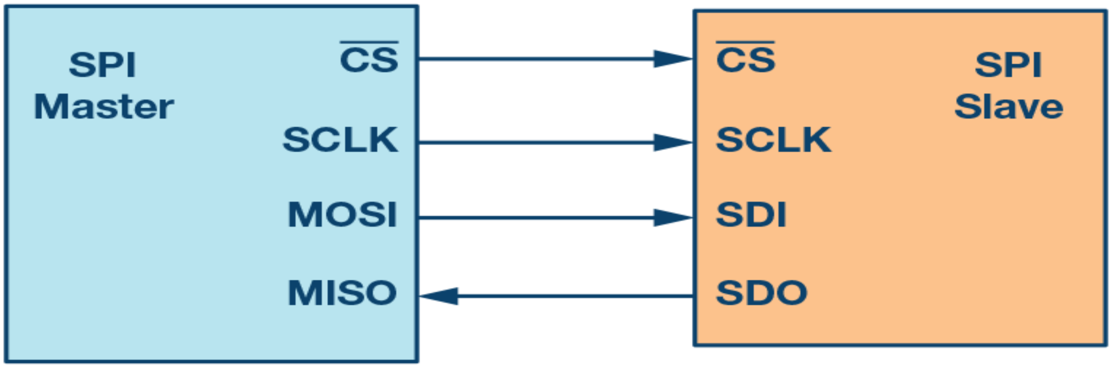


Figure 9: Connections Involved in SPI

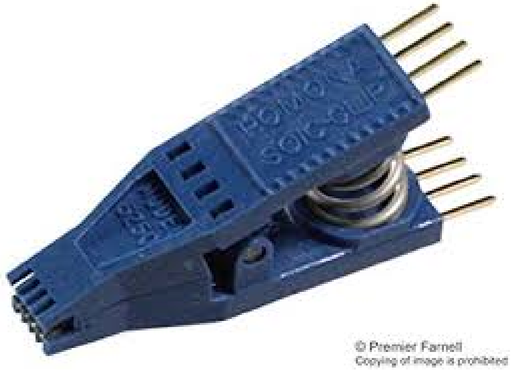
By reading the manual for W25Q64BVSIG, we determine the following pin connections.

Pin No	Pin Name	I/O	Function	Intepretation
1	/CS	I	Chip Select Input	Slave Select
2	DO(I01)	I/O	Data Output	MISO
4	GND		Ground	
5	DI (IO0)	I/O	Data Input	MOSI
6	CLK	I	Serial Clock Input	Clock
8	VCC		Power Supply	

Table 2: Legs of the W25Q64BVSIG in SPI Communication

*Note that Pins 3 & 7 are only used for Quad SPI Instructions

With UART, we had solded on pin headers to the ports in order to connect them our UM232H-B. However, as the SPI chip is soldered onto the board, that was not possible. That being said, we were able to use a Model 5250 Pomona SOIC-Clip to allow us to use jumper wires to communicate with the flash chip.



© Premier Farnell
Copying of image is prohibited

Figure 10: Model 5250 Pomona SOIC-Clip

Now what was left hardware-wise was to connect the SOIC-Clip to the UM232H-B. By referring to its datasheet, we got the following connections.

Port	Function
D0	Clock (CLK)
D1	Data Out (MISO)
D2	Data In (MOSI)
D3	Chip Select (CS)
Gnd	Ground (GND)

Table 3: UM232H-B Port Functions for SPI

After making the relevant connections from the flash chip to the UM232H-B and plugging it into the laptop, the hardware set-up is complete.

Section 3.1: Attempting to Communicate with the SPI Flash Chip

To call the flash chip to dump its memory, we read the datasheet and find the read command. We see that to get the chip to output its data, we dive the CS pin low and shift in the 03h instruction into the data input (DI) pin, followed by a 24 bit memory address of where we want to start reading from. From there the data wiill be put out through the data output (DO) pin until the chip select pin is high again.

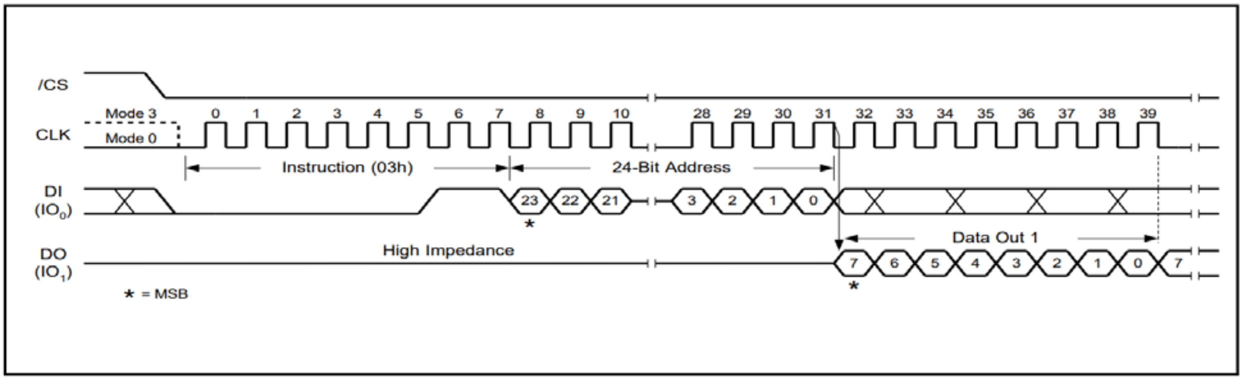


Figure 11: Read Data Instruction of W25Q64BV

In terms of actual implementation, there were 2 main pieces of software that we tried to use.

1. Flashrom[11]
2. Libmpsse[12]

Firstly we attempted to use flashrom to communicate with the flash chip. This at first seems promising because the W25Q64BV chip on the target board is in the list of officially supported devices for flashrom[13]. Furthermore, our FT232H chip that we are using is in the list of supported programmers[14]. For you information, the programmer is the interface that flashrom uses to communicate with the physical interface (here being the UM232H-B), allowing flashrom to access the SPI flash chip through the physical interface[15].

However, problems start almost immediately as we receive an error upon supplying the “type=FT232H” parameter of the “ft2232_spi” programmer, suggesting that the there is no such type. This could be because we conducted this test on a workstation running Ubuntu 14.04[16] which was rather old. To overcome this problem, we downloaded the flashrom source code[17] and built it from source. This allowed flashrom to detect our FT232H chip, but then it was unable to detect the W25Q64BV flashchip.

```
# Command used to interface with SPI chip via flashrom
./flashrom -p ft2232_spi:type=FT232H -i
```

[11] "flashrom." Flashrom, https://www.flashrom.org/Flashrom. Accessed December 18, 2019.

[12] "GitHub." devttys0/libmpsse: Open source library for SPI/I2C control via FTDI chips, https://github.com/devttys0/libmpsse. Accessed December 18, 2019.

[13] "Supported hardware." flashrom, https://www.flashrom.org/Supported_hardware. Accessed December 18, 2019.

[14] "Supported programmers." flashrom, https://www.flashrom.org/Supported_programmers. Accessed December 18, 2019.

[15] "flashrom(8)." Linux man page, https://linux.die.net/man/8/flashrom. Accessed December 18, 2019.

[16] "Ubuntu 14.04.6 LTS (Trusty Tahr)." Ubuntu, http://releases.ubuntu.com/14.04/. Accessed December 18, 2019.

[17] "GitHub." flashrom/flashrom, https://github.com/flashrom/flashrom. Accessed December 18, 2019.

In our attempts to debug the problem, we replaced our UM232H-B with the Attify Badge[18]. The pinouts for the Attify badge are clearly labelled on the badge itself, making it more convenient to use.

Pin	Code	Function
D0	SCK	Serial Clock
D1	MISO	Master In Slave Out
D2	MOSI	Master Out Slave In
D3	CS	Chip Select

Table 4: Labelled SPI Pinout of Attify Badge

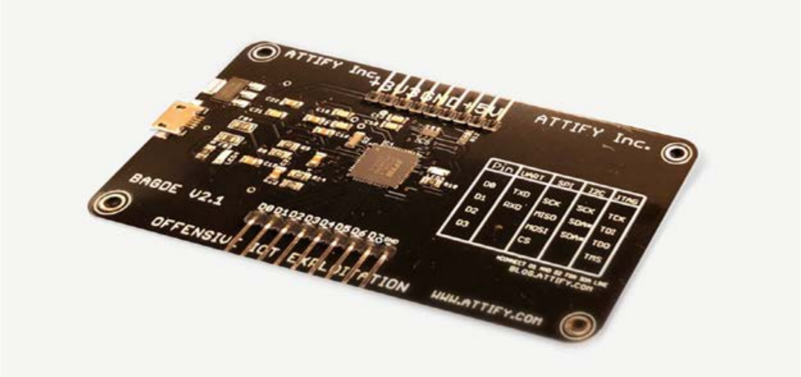


Figure 12: Attify Badge

However, even with the Attify Badge, flashrom was still unable to detect the chip. We were unable to overcome this issue. On hindsight, this is likely because the CPU is constantly querying the flashchip, making it ignore other commands from our FT232H chip.

As flashrom was not working, we tried to use libmpsse to read the flashchip. However, there were also several problems faced. Firstly,it required the use the “libftdi[19]” and “libmpsse[20]” libraries. We installed the former using the “apt” package manager on our Kali Virtual Machine (VM) and installed the latter using the the “pip” python 2 package manager. Note that libmpsse only supports python 2. However, after doing so, running “python spi_flash.py -r dump.bin -s 10000” resulted in a segmentation fault error. It turns out that the issue was that there were some incomptible libaries, resulting in libmpsse crashing. To resolve this, we ran the installation file of the Attify Badge graphical user

[18] "Attify Store - Attify Badge." UART JTAG SPI I2C (pre-soldered headers) | Attify Store, https://www.attify-store.com/products/attify-badge-uart-jtag-spi-i2c-pre-soldered-headers. Accessed December 18, 2019.

[19] "libftdi package : Ubuntu." Launchpad, https://launchpad.net/ubuntu/+source/libftdi. Accessed December 18, 2019.

[20] "libmpsse · PyPI." Pypi, May 03, 2017, https://pypi.org/project/libmpsse/. Accessed December 18, 2019.

interface (GUI)[21] which would install all the libraries required to operate the Attify Badge properly. After doing so, we were able to run the spi_flash.py file, however the dump.bin file was merely all “F”s (1s in binary).

We tested our set-up with another target board that was guaranteed to work, however this also resulted in all “F”s. At this point, one of our collegues mentioned that the labelling of the Attify Badge was wrong with the MISO and MOSI swapped (assuming the attify badge is the master and the connected device is the slave)[22].

Pin	Code	Function
D0	SCK	Serial Clock
D1	MOSI	Master Out Slave In
D2	MISO	Master In Slave Out
D3	CS	Chip Select

Table 5: True SPI Pinout of Attify Badge

Trying again, we are able to read the CHIPID of the guaranteed to work board. However, we are unable to do so with the target board. Even more, we are unable to read from the contents of the flash chip on the guaranteed to work board.

We eventually realised that it was because that board was not receiveing sufficient power from the 3.3v supplied by the Attify Badge to the chip. As such, we used a variable power supply to increase the voltage to 4 volts. We were now able to read from the memory of the flash chip.

However, things still weren’t working with the target board. We were stll reading all “F”s. At this point, we deduded that the CPU was probably keeping the SPI flash chip busy and that we would have to desolder the chip to be able to dump its memory via SPI. Thus we decided to put a pause to this venture and start on the Joint Test Action Group (JTAG) section first.

Section 4.0: Preparations for JTAG Communication

As with the other 2 protocols used earlier, it is good to know what connections we need to make before starting. For a successful JTAG connection, there are 4 mandatory connections and 1 optional one[23].

They are:

[21] "GitHub." attify/attify-badge: Attify Badge GUI tool to interact over UART, SPI, JTAG, GPIO etc., https://github.com/attify/attify-badge. Accessed December 18, 2019.

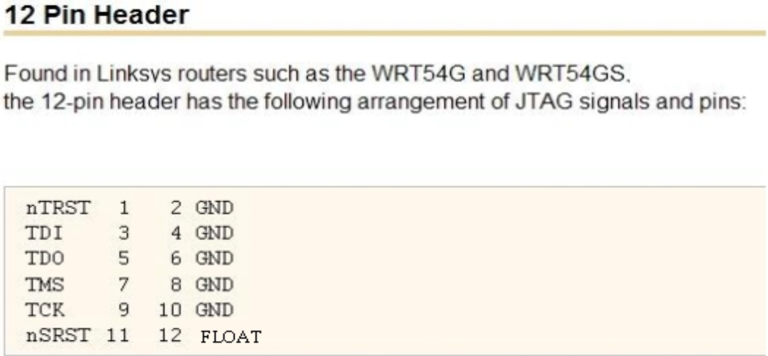
[22] "Intro to Hardware Hacking." Dumping your First Firmware, https://nvisium.com/blog/2019/08/07/extracting

[23] "Technical Guide to JTAG." XJTAG Tutorial, https://www.xjtag.com/about-jtag/jtag-a-technical-overview/. Accessed December 18, 2019.

1. **TCK** (Test Clock)
2. **TMS** (Test Mode Select)
3. **TDI** (Test Data In)
4. **TDO** (Test Data Out)
5. **TRST** (Test Reset) [Optional]

The collection of the mandatory ports make up a test access port (TAP).

In the internal information gathering stage (refer to section 1.1), we identified a 12 pin JTAG connector on the PCB. We now proceed to identify the pin out, via a quick Google search, we find the following pin configuration.



12 Pin Header

Found in Linksys routers such as the WRT54G and WRT54GS,
the 12-pin header has the following arrangement of JTAG signals and pins:

```
nTRST  1    2 GND
TDI    3    4 GND
TDO    5    6 GND
TMS    7    8 GND
TCK    9   10 GND
nSRST 11   12 FLOAT
```

Figure 13: Pinout Found Online[24]

Thus we know that we only need 6 wires, 1 for each data transfer wires and 1 for ground. However, at this stage, the connector is merely 12 very small holes on the PCB, we then spend a considerable amount of time soldering connectors to the holes.

We initially try to put straight headers[25] into the holes. However, we find that we are unable to put rows adjacent to each other. We then try to put a 2x6 T shaped header into the ports, but this led to the issue of the headers being difficult to attach to wires as the header pins were very close to each other, thus it was very easy for solder to short 2 pins.

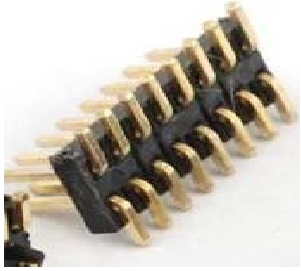


Figure 14: T-shaped Headers

[24] https://forum.dd-wrt.com/phpBB2/files/jtag_pin_out_142.jpg
[25] Refer to Figure 5

We eventually settled on directly shoving wires through the holes on the PCB and soldering them directly there. However, there is the problem of the multi-stranded wires being too soft to shove throught the holes. We avoided this problem by only putting 3 out of 5 of the individual strands into each hole. However there is nowo the problem of the connection being relatively weak and the stray strands touching other connections, resulting in flaky connections later on.

That being said, the connection could be made (no matter how flaky), and as long as we didn't touch it too much, it would be fine.'

We proceed to verify the above pinout in Figure 11 using the JTAGulator[26]. We connected up the JTAG port to our JTAGulator and connected to the JTAGulator over UART using picocom. However, upon doing the IDSCAN and BYPASS scan, there are no results returned. While we were very puzzled, looking back, it is probably because of the flaky hardware connection described above.

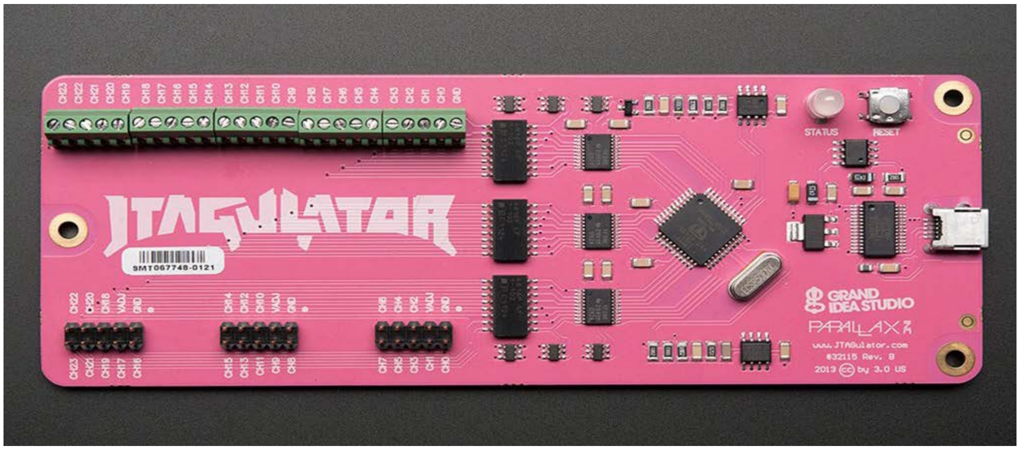


Figure 15: JTAGulator

We then assumed that the Figure 11 pinout was correct and proceeded to connect the board to our Attify Badge.We attach the 4 mandatory wires and ignore the optional reset pin.

Section 4.1: Communication via JTAG

The standard for communicating via JTAG and issuing high level commands to the CPU is Open On-Chip Debugger (OpenOCD)[27]. Fortunately for us, it was already installed on my Kali VM after running the Attify Badge GUI install script in section 3.1.

To use OpenOCD, we need to specify configuration files ideally for these 3 components

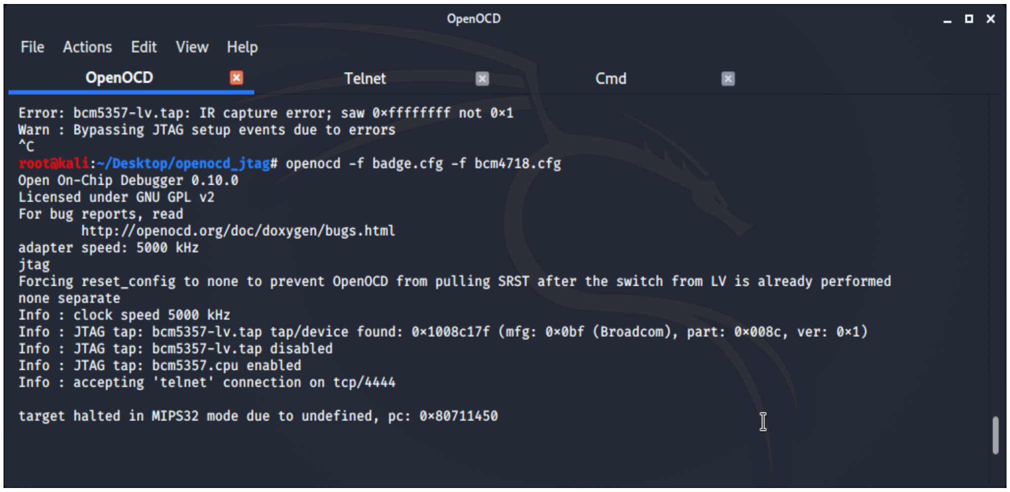
[26] Industries, Adafruit. "JTAGulator by Grand Idea Studio." adafruit industries blog RSS. Accessed December 18, 2019. https://www.adafruit.com/product/1550.

[27] "Open On-Chip Debugger." Openocd, http://openocd.org/. Accessed December 18, 2019.

1. The interface we use (our Attify Badge)
2. Our target CPU (BCM5357)
3. Our taret board (E1200v2)

While there is a configuration file for the Attify Badge, we are unable to find a config file for our target board and our target CPU. Thus we start by using the autoprobe function[28] to detect the test access points (TAPs) on the CPU.

This results in OpenOCD being able to identify a TAP and returning the instruction register length and the expected device ID . We then modify the BCM4718.cfg file by changing the LVTAP ID to the one that OpenOCD autoprobe had detected. After which we ran openOCD again but were prompted with the error of an unexpected CPUID. However, the error gave the CPUID it found, so we just modified our BCM4718.cfg[29] to expect that CPUID. And just like that, we are in.



```
Error: bcm5357-lv.tap: IR capture error; saw 0xffffffff not 0x1
Warn : Bypassing JTAG setup events due to errors
^C
root@kali:~/Desktop/openocd_jtag# openocd -f badge.cfg -f bcm4718.cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 5000 kHz
jtag
Forcing reset_config to none to prevent OpenOCD from pulling SRST after the switch from LV is already performed
none separate
Info : clock speed 5000 kHz
Info : JTAG tap: bcm5357-lv.tap tap/device found: 0x1008c17f (mfg: 0x0bf (Broadcom), part: 0x008c, ver: 0x1)
Info : JTAG tap: bcm5357-lv.tap disabled
Info : JTAG tap: bcm5357.cpu enabled
Info : accepting 'telnet' connection on tcp/4444

target halted in MIPS32 mode due to undefined, pc: 0x80711450
```

Figure 16: Successful Start of the OpenOCD Server

From there, we test the various functions of OpenOCD to ensure we have control over the CPU. We test poll which shows the TAP as being enabled, and we test halt which returns that the CPU has been halted. However, when we run “targets” to check the status of the CPU, we see that it is still running. We then try spamming halt multiiple times which results in the CPU being actually halted. We theorise that there is a watchdog process running that reboots the CPU when it is halted. This is further suggested by the fact that the CPU always halts properly when we send 2 halt commands fast and in succession meaning that the first halt causes a reboot and the second halt stops the CPU before the watchdog process is start up.

[28] "OpenOCD User’s Guide: TAP Declaration." Openocd, http://openocd.org/doc/html/TAP-Declaration.html. Accessed December 18, 2019.

[29] Refer to Annex C

To really confirm that we are controlling the CPU, we look into the boot process using UART with the UM232H-B and as the CPU boots up, we halt the CPU, and indeed we see the reboot caused by the watchdog process and then we see that the CPU halts after the second halt command. We are also able to step through the instructions and call for the processor to resume operation, thus we conclude that we have attained control over the CPU via JTAG.

Section 4.2: Flash Dumping with OpenOCD

To do a flash dump, we first need to declare the parameters of the flash storage to openOCD. This is done through the flash bank command[30] in the config file. As we didn't know many of the parameters of the flash memory such as the "chip_width" or "bus_width", we looked at the config file of another syslink router board (linksys-wrt54gl.cfg) and modified the flash bank command from there. We knew that the size of the flash chip on our target board was 8MB so that was the only parameter we changed. Details for the modified command are found in Annex C). An important parameter that we needed to modify but didn't know what to modify the value to was the base_address of the flash chip.

Eventually, by looking at the UART logs, we notice a line

```
CMD: [boot -raw -z -addr=0x80001000 -max=0x6ff000 flash0.os:]
```

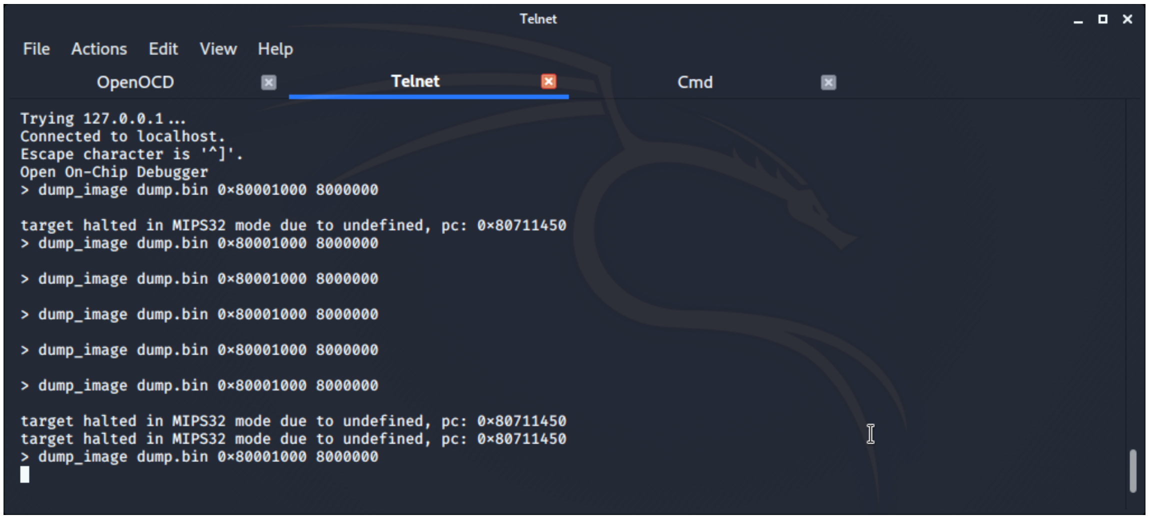
and use 0x80001000 as the base address.
Now there are 2 defined functions to read from flash memory

- flash read
- dump_image

As flash read threw some errors, we just went with using dump_image. However, dump_image requires an address to start from which we don't really know. We tried dumping 712944 bytes from 0x80700000, which were values we got from the UART boot logs, however, a binwalk on the output didn't yield anything interesting. That being said, we were able to perform "strings" on the bin file and found that we were dumping some interesting strings, meaning that we were dumping near the correct area.

Refering back to the avoid boot command in the UART log, we try to dump 8MB from 0x80001000 to extract the entire flash disk of contents. However, due to the flaky connection, it was difficult to dump the entire disk without the connection being dropped. The entire transfer was expected to take 8 hours, thus we decided to run it overnight.

[30] "OpenOCD User's Guide: Flash Commands." Openocd, http://openocd.org/doc/html/Flash-Commands.html. Accessed December 18, 2019.



```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> dump_image dump.bin 0×80001000 8000000

target halted in MIPS32 mode due to undefined, pc: 0×80711450
> dump_image dump.bin 0×80001000 8000000

> dump_image dump.bin 0×80001000 8000000

> dump_image dump.bin 0×80001000 8000000

> dump_image dump.bin 0×80001000 8000000

> dump_image dump.bin 0×80001000 8000000

target halted in MIPS32 mode due to undefined, pc: 0×80711450
target halted in MIPS32 mode due to undefined, pc: 0×80711450
> dump_image dump.bin 0×80001000 8000000
```

Figure 17: Connection Repeatedly Dropped While Dumping 8MB

The dumping took 10.9 hours, however, we managed to attain a 7.7MB file as a result. Performing a binwalk on the file, we get the result in Annex D.

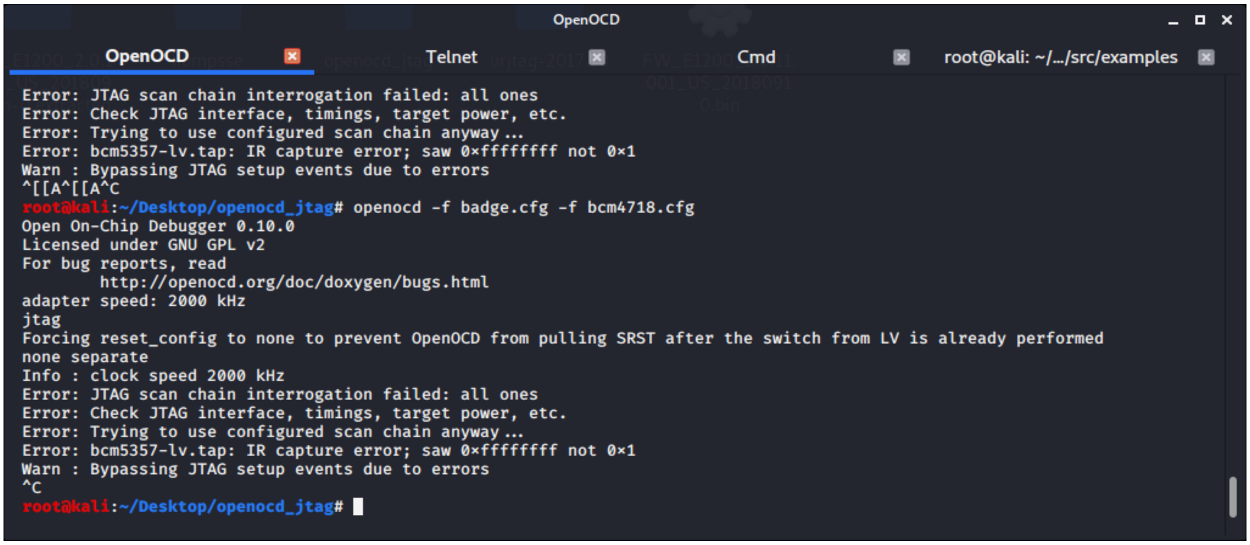
Where there are indeed many different components of the bin file that are identifiable by binwalk, there is no squashfs file system found and we unable to extract the firmware with binwalk -e.

Eventually, we realised that dump_image dumps from the system's RAM rather than flash. Thus we spend a large amount of time trying to configure the "flash bank" command but to no avail.

We also try to halt the CPU with JTAG and dump the flash through SPI. But that also didn't work, proably because the CPU is halted in a position that still holds control over the SPI chip.

Lastly, we try to go into the CFE through UART and use the "load" command to load the OS into the RAM. After which, we would use dump_image to access the RAM. However, the Attify Badge I had been using died before we could try this.

We tried to use another Attify Badge to connect with JTAG again there were some errrors and we were unable to successfully control the CPU again. We hypothesize that the boundary scan cells were destroyed when the first Attify Badge was also destroyed. As such we are unable to continue with our JTAG ventures.



```
Error: JTAG scan chain interrogation failed: all ones
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: bcm5357-lv.tap: IR capture error; saw 0xffffffff not 0x1
Warn : Bypassing JTAG setup events due to errors
^[[A^[[A^C
root@kali:~/Desktop/openocd_jtag# openocd -f badge.cfg -f bcm4718.cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 2000 kHz
jtag
Forcing reset_config to none to prevent OpenOCD from pulling SRST after the switch from LV is already performed
none separate
Info : clock speed 2000 kHz
Error: JTAG scan chain interrogation failed: all ones
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: bcm5357-lv.tap: IR capture error; saw 0xffffffff not 0x1
Warn : Bypassing JTAG setup events due to errors
^C
root@kali:~/Desktop/openocd_jtag# 
```

Figure 18: OpenOCD Unable to Communicate with the CPU Properly

Annex A: UART Boot Log and Entering CFE via UART

```
Decompressing...done
Start to blink diag led ...


CFE version 5.100.138.11 based on BBP 1.0.37 for BCM947XX (32bit,SP,LE)
Build Date: 11/23/11 12:16:38 CST (wzh@cybertan)
Copyright (C) 2000-2008 Broadcom Corporation.

Initializing Arena
Initializing Devices.

No DPN
This is a Serial Flash
Boot partition size = 262144(0x40000)
Found an ST compatible serial flash with 128 64KB blocks; total size 8MB
Partition information:
boot    #00   00000000 -> 0003FFFF  (262144)
trx     #01   00040000 -> 0004001B  (28)
os      #02   0004001C -> 007EFFFF  (8060900)
nvram   #03   007F0000 -> 007FFFFF  (65536)
Partition information:
boot    #00   00000000 -> 0003FFFF  (262144)
trx     #01   00040000 -> 007EFFFF  (8060928)
nvram   #02   007F0000 -> 007FFFFF  (65536)
BCM47XX_GMAC_ID
et0: Broadcom BCM47XX 10/100/1000 Mbps Ethernet Controller 5.100.138.11
CPU type 0x19749: 300MHz
Total memory: 32768 KBytes

CFE mem:    0x80700000 - 0x807AE0F0 (712944)
Data:       0x80743360 - 0x80747440 (16608)
BSS:        0x80747440 - 0x807480F0 (3248)
Heap:       0x807480F0 - 0x807AC0F0 (409600)
Stack:      0x807AC0F0 - 0x807AE0F0 (8192)
Text:       0x80700000 - 0x80743360 (275296)

Boot version: v5.3.7
The boot is CFE
mac_init(): Find mac [20:aa:4b:3c:ce:b3] in location 0
Nothing...

### CLKDIV= 0x8080842, SFlashClkDiv=8 clkdivsf=2 ###

### Change it to 0x2080842 (2) ###
CMD: [ifconfig eth0 -addr=192.168.1.1 -mask=255.255.255.0]
Device eth0:  hwaddr 20-AA-4B-3C-CE-B3, ipaddr 192.168.1.1, mask 255.255.255.0
        gateway not set, nameserver not set
CMD: [go;]
Check CRC of image1
  Len:     0x771000   (7802880)      (0xBC040000)
  Offset0: 0x1C              (28)           (0xBC04001C)
  Offset1: 0x14FF14   (1376020)      (0xBC18FF14)
  Offset2: 0x0 (0)    (0xBC040000)
  Header CRC:    0xE66A894E
  Calculate CRC: 0xE66A894E
Image 1 is OK
Try to load image 1.
Waiting for 3 seconds to upgrade ...
CMD: [load -raw -addr=0x807ae0f0 -max=0x1851f10 :]
```

```
Loader:raw Filesys:tftp Dev:eth0 File:: Options:(null)
Loading: _tftpd_open(): retries=0/3
_tftpd_open(): retries=1/3
_tftpd_open(): retries=2/3

### Start=417500943 E=735697857 Delta=318196914 ###
Failed.
Could not load :: Timeout occured
CMD: [boot -raw -z -addr=0x80001000 -max=0x6ff000 flash0.os:]
Loader:raw Filesys:raw Dev:flash0.os File: Options:(null)
Loading: ...... 3334532 bytes read

### Start=740727873 E=921456100 Delta=180728227 ###
Entry at 0x80001000
Closing network.
Starting program at 0x80001000
Linux version 2.6.22 (hhm@sw3) (gcc version 4.2.3) #44 Sat Sep 8 13:15:31 HKT 20
18
prom_init:123: memory size is (2000000) by automatily calculating!
prom_init:190: mem:2000000, actually, test by seal!
CPU revision is: 00019749
Found an ST compatible serial flash with 128 64KB blocks; total size 8MB
Determined physical RAM map:
 memory: 02000000 @ 00000000 (usable)
Zone PFN ranges:
  Normal          0 ->     8192
  HighMem      8192 ->     8192
early_node_map[1] active PFN ranges
    0:        0 ->     8192
Built 1 zonelists.  Total pages: 8192
Kernel command line: root=/dev/mtdblock2 console=ttyS0,115200 init=/sbin/preinit
Primary instruction cache 32kB, physically tagged, 4-way, linesize 32 bytes.
Primary data cache 32kB, 4-way, linesize 32 bytes.
Synthesized TLB refill handler (20 instructions).
Synthesized TLB load handler fastpath (32 instructions).
Synthesized TLB store handler fastpath (32 instructions).
Synthesized TLB modify handler fastpath (31 instructions).
PID hash table entries: 256 (order: 8, 1024 bytes)
CPU: BCM53572 rev 1 at 300 MHz
Using 150.000 MHz high precision timer.
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory: 27916k/32768k available (2546k kernel code, 4836k reserved, 505k data, 2
04k init, 0k highmem)
Mount-cache hash table entries: 512
NET: Registered protocol family 16
PCI: no core
PCI: no core
PCI: Fixing up bus 0
NET: Registered protocol family 2
Time: MIPS clocksource has been installed.
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 1, 8192 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
squashfs: version 3.2-r2 (2007/01/15) Phillip Lougher
fuse init (API version 7.8)
io scheduler noop registered (default)
HDLC line discipline: version $Revision: 1.1.1.1 $, maxframe=4096
N_HDLC line discipline registered.
```

```
Serial: 8250/16550 driver $Revision: 1.1.1.1 $ 4 ports, IRQ sharing disabled
serial8250: ttyS0 at MMIO 0x0 (irq = 8) is a 16550A
loop: module loaded
PPP generic driver version 2.4.2
NET: Registered protocol family 24
Register DIAG LED in /proc/sys/diag_blink.
The DIAG LED GPIO is 6.
Register DIAG LED success in /proc/sys/diag_blink.
pflash: found no supported devices
sflash: squash filesystem with lzma found at block 1599
Creating 4 MTD partitions on "sflash":
0x00000000-0x00040000 : "boot"
0x00040000-0x007f0000 : "linux"
0x0018ff14-0x007f0000 : "rootfs"
mtd: partition "rootfs" doesn't start on an erase block boundary -- force read-o
nly
0x007f0000-0x00800000 : "nvram"
u32 classifier
nf_conntrack version 0.5.0 (256 buckets, 2048 max)
edward ====to register conntrack protocol helper for esp:
nf_conntrack_rtsp v0.6.21 loading
nf_nat_rtsp v0.6.21 loading
edward =======nf_nat_proto_esp_init
ip_tables: (C) 2000-2006 Netfilter Core Team
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 10
lo: Disabled Privacy Extensions
ip6_tables: (C) 2000-2006 Netfilter Core Team
NET: Registered protocol family 17
Ebtables v2.0 registered
802.1Q VLAN Support v1.8 Ben Greear <greearb@candelatech.com>
All bugs added by David S. Miller <davem@redhat.com>
VFS: Mounted root (squashfs filesystem) readonly.
Freeing unused kernel memory: 204k freed
Warning: unable to open an initial console.
Failed to execute /init
ctmisc: module license 'unspecified' taints kernel.
Register /dev/ctmisc device, major:10 minor:255
cmd=[/sbin/hotplug2 --coldplug & ]
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
```

```
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
/dev/nvram: No such file or directory
[sighandler]: No more events to be processed, quitting.
[cleanup]: Waiting for children.
[cleanup]: All children terminated.
hahaha enter wl_nvram_convert!
The boot is CFE
Algorithmics/MIPS FPU Emulator v1.5
/dev/: cannot create
cmd=[misc -t get_mac -w 3 ]
type = [get_mac]ctmisc_ioctl: cmd=0x11, buffer size=404

get_data(): cmdata_init(): base = 0xbc03ee00
d=0x11 count=8 ldata_init(): location = [1], mydatas index = 1
en=18
ctmisc_ioctl: index=1
tallest:=====(ctmisc ioctl done...)=====
get_data(): Get MAC count is [1]
get_data(): MAC 0: [20:aa:4b:3c:ce:b3ÿ]
get_data name get_mac write_to_nv 3
get_data(): done
cmd=[misc -t get_wsc_pin -w 3 ]
type = [get_wsc_ctmisc_ioctl: cmd=0x26, buffer size=404
pin]
get_data()data_init(): base = 0xbc03f400
: cmd=0x26 countdata_init(): location = [1], mydatas index = 1
=8 len=8
ctmisc_ioctl: index=1
tallest:=====(ctmisc ioctl done...)=====
get_data(): Get WSC count is [1]
get_data(): WSC 0: [87227482]
get_data name get_wsc_pin write_to_nv 3
get_data(): done
cmd=[misc -t get_sn -w 3 ]
type = [get_sn]ctmisc_ioctl: cmd=0x15, buffer size=404

get_data(): cmdata_init(): base = 0xbc03fe32
d=0x15 count=8 ldata_init(): location = [1], mydatas index = 1
en=20
ctmisc_ioctl: index=1
tallest:=====(ctmisc ioctl done...)=====
get_data(): Get SN count is [1]
get_data(): SN 0: [10820C63242832ÿÿÿÿÿÿ]
get_data name get_sn write_to_nv 3
get_data(): done
cmd=[misc -t get_flash_type -w 1 ]
type = [get_flasctmisc_ioctl: cmd=0x17, buffer size=404
h_type]
```

```
get_flasflash_init: sflash type 0x100
sh_type(): cmd=0sflash_init: sflash type 0x16
x17 count=0 len=Flash Type: SFLASH 8192 kB
0
tallest:=====(ctmisc ioctl done...)=====
Get FLASH TYPE is [SFLASH 8192 kB]
cmd=[misc -t get_pa0idxval -w 3 ]
type = [get_pa0ictmisc_ioctl: cmd=0x28, buffer size=404
dxval]
get_datadata_init(): base = 0xbc03efe0
(): cmd=0x28 coudata_init(): location = [0], mydatas index = 0
nt=8 len=24
ctmisc_ioctl: index=0
tallest:=====(ctmisc ioctl done...)=====
get_data(): Get PA0IDXVAL count is [0]
get_data name get_pa0idxval write_to_nv 3
get_data(): done
Using default PA0 value
cmd=[misc -t get_pa1idxval -w 3 ]
type = [get_pa1ictmisc_ioctl: cmd=0x2a, buffer size=404
dxval]
get_datadata_init(): base = 0xbc03ef20
(): cmd=0x2a coudata_init(): location = [0], mydatas index = 0
nt=8 len=24
ctmisc_ioctl: index=0
tallest:=====(ctmisc ioctl done...)=====
get_data(): Get PA1IDXVAL count is [0]
get_data name get_pa1idxval write_to_nv 3
get_data(): done
Using default PA1 value
Cannot find lang from /proc/mtd
ret = -1
www -> /www
mount: No such file or directory
cmd=[insmod emf ]
cmd=[insmod igs ]
cmd=[insmod ctf ]
Needed modules: et wl ip6table_mangle ip6table_filter ip6t_rt ip6t_frag ip6t_ipv
6header ip6t_REJECT ip6t_LOG ip6t_ipv6range tunnel4 sit tunnel6 ip6_tunnel nf_co
nntrack_h323.ko nf_nat_h323.ko xt_TCPMSS.ko
cmd=[insmod et ]
cmd=[insmod wl ]
cmd=[insmod ip6table_mangle ]
cmd=[insmod ip6table_filter ]
cmd=[insmod ip6t_rt ]
cmd=[insmod ip6t_frag ]
cmd=[insmod ip6t_ipv6header ]
cmd=[insmod ip6t_REJECT ]
cmd=[insmod ip6t_LOG ]
cmd=[insmod ip6t_ipv6range ]
cmd=[insmod tunnel4 ]
cmd=[insmod sit ]
cmd=[insmod tunnel6 ]
cmd=[insmod ip6_tunnel ]
cmd=[insmod nf_conntrack_h323.ko ]
cmd=[insmod nf_nat_h323.ko ]
cmd=[insmod xt_TCPMSS.ko ]
cmd=[insmod dnshook ]
Hit enter to continue...cmd=[misc -t get_country -w 3 ]
type = [get_country]
get_data(): cmd=0x2c count=30 len=2
get_data(): Get COUNTRY count is [1]
get_data(): COUNTRY 0: [AU]
get_data name get_country write_to_nv 3
get_data(): done
waitpid: No child processes
The chipset is BCM5357 for E1200
cmd=[killall httpd ]
killall: httpd: no process killed
killall: check_http.sh: no process killed
cmd=[killall gn-httpd ]
killall: gn-httpd: no process killed
waitpid: No child processes
cmd=[killall gn-httpd ]
killall: gn-httpd: no process killed
waitpid: No child processes
cmd=[killall wm-httpd ]
killall: wm-httpd: no process killed
waitpid: No child processes
cmd=[et robowr 0x02 0x06 0x001000a0 ]
cmd=[resetbutton ]
cmd=[vconfig set_name_type VLAN_PLUS_VID_NO_PAD ]
cmd=[vconfig add eth0 1 ]
cmd=[vconfig set_ingress_map vlan1 0 0 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 1 1 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 2 2 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 3 3 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 4 4 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 5 5 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 6 6 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan1 7 7 ]
waitpid: No child processes
cmd=[vconfig add eth0 2 ]
cmd=[vconfig set_ingress_map vlan2 0 0 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 1 1 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 2 2 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 3 3 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 4 4 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 5 5 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 6 6 ]
waitpid: No child processes
cmd=[vconfig set_ingress_map vlan2 7 7 ]
waitpid: No child processes
cmd=[brctl addbr br0 ]
cmd=[brctl setfd br0 0 ]
waitpid: No child processes
cmd=[brctl stp br0 dis ]
cmd=[brctl addif br0 vlan1 ]
waitpid: No child processes
```

```
br0: No child processes
cmd=[wlconf vlan1 up ]
vlan1: Operation not supported
Write wireless mac successfully
cmd=[brctl addif br0 eth1 ]
waitpid: No child processes
br0: No child processes
cmd=[wlconf eth1 up ]
eth1: Operation not supported
eth1: Operation not permitted
wlconf: PHYTYPE: 4
eth1: Invalid argument
eth1: Invalid argument
eth1: Operation not supported
eth1: Operation not supported
cmd=[brctl addif br0 eth1 ]
device eth1 is already a member of a bridge; can't enslave it to bridge br0.
waitpid: No child processes
Write wireless mac fail : : No such device
cmd=[brctl addif br0 eth2 ]
interface eth2 does not exist!
eth2: No such device
cmd=[brctl addif br0 eth3 ]
interface eth3 does not exist!
eth3: No such device
Set 1 to /proc/sys/net/ipv6/conf/br0/forwarding ...
cmd=[iptables -t nat -F wlwarning2wan ]
iptables: No chain/target/match by that name
cmd=[iptables -F wlwarningaccept ]
iptables: No chain/target/match by that name
waitpid: No child processes
cmd=[ip6tables -t mangle -F wlwarning2wan ]
ip6tables: No chain/target/match by that name
cmd=[ip6tables -t filter -F wlwarningaccept ]
ip6tables: No chain/target/match by that name
waitpid: No child processes
lo: File exists
Set 66560 to /proc/sys/net/core/rmem_max ...
set 2048 to /proc/sys/vm/min_free_kbytes
cmd=[klogd -c 1 ]
cmd=[syslogd -m 0 -O /var/log/mess ]
cmd=[tftpd -s /tmp -c -l -P E150 ]
cmd=[cron ]
The boot is CFE
tftp server started
tftpd: standalone socket
cmd=[httpd ]
cmd=[touch /tmp/hosts ]
waitpid: No child processes
cmd=[dnsmasq -h -i br0 -c 0 -r /tmp/resolv.conf -u  ]
cmd=[route del -net 224.0.0.0 netmask 240.0.0.0 dev br0 ]
route: ioctl 0x890c failed: No such process
waitpid: No child processes
cmd=[route add -net 224.0.0.0 netmask 240.0.0.0 dev br0 ]
cmd=[cesmDNS -o /tmp/.mdns_host_info -d -h Cisco42832 -l 192.168.1.1 ]
Starting in daemon mode
br0 192.168.1.100  86400
write_dhcpd_conf: file=/tmp/dhcpd-br0.conf, ifname=br0, lan_ip=lan_ipaddr lan_ma
sk=lan_netmask
cmd=[dhcpd -cf /tmp/dhcpd-br0.conf -lf /tmp/dhcpd.leases -df /tmp/udhcpd.leases
-pf /var/run/dhcpd.pid br0 ]
```

```
Internet Systems Consortium DHCP Server 4.1.1-P1
Copyright 2004-2010 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 0 leases to leases file.
Listening on Socket/br0/192.168.1.0/24
Sending on   Socket/br0/192.168.1.0/24
cmd=[upnp -D -W vlan2 ]
cmd=[/bin/eapd ]
UPnP::upnp_device_attach:br0: attach InternetGatewayDevice.xml
ssdp byebye
UPnP::upnp_init:UPnP daemon is ready to run
cmd=[nas ]
cmd=[killall wps_monitor ]
killall: wps_monitor: no process killed
waitpid: No child processes
cmd=[killall wps_ap ]
killall: wps_ap: no process killed
cmd=[killall wps_enr ]
killall: wps_enr: no process killed
cmd=[/bin/wps_monitor ]
cmd=[/usr/sbin/acsd ]
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: scan in progress ...
acsd: selected channel spec: 0x2b01
cmd=[netbios /tmp/samba/lib/netbios.conf ]
cmd=[nlinkd ]
lltd:echo Cisco42832 > /proc/sys/kernel/hostname
LLTD: wireless interface argument is eth1.
cmd=[killall -1 radvd ]
killall: radvd: no process killed
cmd=[/sbin/monitor_cable ]
cmd=[/usr/sbin/arp -c ]
cmd=[touch /tmp/hosts ]
waitpid: No child processes
tallest:=====( wan_or_lan=wan )=====
start_wan_ipv6: [IPV6] vlan2 dhcp
cmd=[touch /tmp/hosts ]
waitpid: No child processes
dhcpc_main,reason[PREINIT]
start_wan_ipv6: [IPV6]  0
Hit enter to continue...cmd=[killall igmpxmld ]
killall: igmpxmld: no process killed
stop_dhcp6c
cmd=[/usr/sbin/ip -6 addr flush dev vlan2 scope global ]
cmd=[ip6tables -t filter -F ]
waitpid: No child processes
cmd=[ip6tables -t filter -Z ]
cmd=[ip6tables -t mangle -F ]
cmd=[ip6tables -t mangle -Z ]
Set 0 to /proc/sys/net/ipv6/conf/all/forwarding ...
```

```
ioctl: No such device
cmd=[killall -9 waninfo ]
stop_wan_ipv6: done
RTNETLINK answers: No such file or directory
Set 0 to /proc/sys/net/ctf/wan_mode ...
cmd=[killall nlinkd ]
killall: cannot kill pid 455: No such process
killall: cannot kill pid 456: No such process
killall: cannot kill pid 457: No such process
killall: cannot kill pid 458: No such process
killall: cannot kill pid 459: No such process
killall: cannot kill pid 460: No such process
killall: cannot kill pid 461: No such process
killall: cannot kill pid 462: No such process
killall: cannot kill pid 463: No such process
killall: cannot kill pid 464: No such process
killall: cannot kill pid 465: No such process
killall: cannot kill pid 466: No such process
killall: cannot kill pid 467: No such process
killall: cannot kill pid 468: No such process
killall: cannot kill pid 469: No such process
killall: cannot kill pid 470: No such process
killall: cannot kill pid 471: No such process
killall: cannot kill pid 472: No such process
killall: cannot kill pid 473: No such process
killall: cannot kill pid 474: No such process
killall: cannot kill pid 475: No such process
killall: cannot kill pid 476: No such process
killall: cannot kill pid 477: No such process
killall: cannot kill pid 478: No such process
killall: cannot kill pid 479: No such process
killall: cannot kill pid 480: No such process
killall: cannot kill pid 481: No such process
killall: cannot kill pid 482: No such process
killall: cannot kill pid 483: No such process
killall: cannot kill pid 484: No such process
killall: cannot kill pid 485: No such process
cmd=[killall -9 nlinkd ]
killall: nlinkd: no process killed
waitpid: No child processes
cmd=[killall -9 qos_bw_detect ]
killall: qos_bw_detect: no process killed
cmd=[killall igmprt ]
killall: igmprt: no process killed
waitpid: No child processes
cmd=[killall pppd ]
killall: pppd: no process killed
cmd=[killall -9 pppd ]
killall: pppd: no process killed
cmd=[killall ip-up ]
killall: ip-up: no process killed
cmd=[killall ip-down ]
killall: ip-down: no process killed
cmd=[killall -15 pppd ]
killall: pppd: no process killed
cmd=[killall -9 pppd ]
killall: pppd: no process killed
cmd=[killall -15 l2tpd ]
killall: l2tpd: no process killed
cmd=[killall -9 l2tpd ]
killall: l2tpd: no process killed
```

```
cmd=[killall -9 listen ]
killall: listen: no process killed
stop_dhcpc
cmd=[killall bpalogin ]
killall: bpalogin: no process killed
cmd=[killall -9 bpalogin ]
killall: bpalogin: no process killed
cmd=[killall -9 pppd ]
killall: pppd: no process killed
cmd=[killall -9 ntpclient ]
killall: ntpclient: no process killed
waitpid: No child processes
cmd=[killall -9 redial ]
killall: redial: no process killed
cmd=[killall wan_auto_detect ]
killall: wan_auto_detect: no process killed
Hit enter to continue...


BusyBox v1.7.2 (2018-09-08 13:19:02 HKT) built-in shell (msh)
Enter 'help' for a list of built-in commands.

#
# reboot
Restarting system.
Decompressing...done
Start to blink diag led ...


CFE version 5.100.138.11 based on BBP 1.0.37 for BCM947XX (32bit,SP,LE)
Build Date: 11/23/11 12:16:38 CST (wzh@cybertan)
Copyright (C) 2000-2008 Broadcom Corporation.

Initializing Arena
Initializing Devices.

No DPN
This is a Serial Flash
Boot partition size = 262144(0x40000)
Found an ST compatible serial flash with 128 64KB blocks; total size 8MB
Partition information:
boot    #00   00000000 -> 0003FFFF  (262144)
trx     #01   00040000 -> 0004001B  (28)
os      #02   0004001C -> 007EFFFF  (8060900)
nvram   #03   007F0000 -> 007FFFFF  (65536)
Partition information:
boot    #00   00000000 -> 0003FFFF  (262144)
trx     #01   00040000 -> 007EFFFF  (8060928)
nvram   #02   007F0000 -> 007FFFFF  (65536)
BCM47XX_GMAC_ID
et0: Broadcom BCM47XX 10/100/1000 Mbps Ethernet Controller 5.100.138.11
CPU type 0x19749: 300MHz
Total memory: 32768 KBytes

CFE mem:    0x80700000 - 0x807AE0F0 (712944)
Data:       0x80743360 - 0x80747440 (16608)
BSS:        0x80747440 - 0x807480F0 (3248)
Heap:       0x807480F0 - 0x807AC0F0 (409600)
Stack:      0x807AC0F0 - 0x807AE0F0 (8192)
Text:       0x80700000 - 0x80743360 (275296)
```

```
Boot version: v5.3.7
The boot is CFE
mac_init(): Find mac [20:aa:4b:3c:ce:b3] in location 0
Nothing...

### CLKDIV= 0x2080842, SFlashClkDiv=2 clkdivsf=2 ###

### Change it to 0x2080842 (2) ###
CMD: [ifconfig eth0 -addr=192.168.1.1 -mask=255.255.255.0]
Device eth0:  hwaddr 20-AA-4B-3C-CE-B3, ipaddr 192.168.1.1, mask 255.255.255.0
        gateway not set, nameserver not set
Automatic startup canceled via Ctrl-C / ESC
CFE> ^C
CFE> ^C
CFE>
```

Annex B: UART Shell Information

```
# cpuinfo
system type             : Broadcom BCMD144 chip rev 1
processor               : 0
cpu model               : MIPS 74K V4.9
BogoMIPS                : 149.50
wait instruction        : no
microsecond timers      : yes
tlb_entries             : 32
extra interrupt vector  : no
hardware watchpoint     : yes
ASEs implemented        : mips16 dsp
VCED exceptions         : not available
VCEI exceptions         : not available
unaligned_instructions : 41
dcache hits             : 2147483648
dcache misses           : 4294426625
icache hits             : 2147483648
icache misses           : 4253020159
instructions            : 2147483648
#cat filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   pipefs
nodev   anon_inodefs
nodev   futexfs
nodev   tmpfs
nodev   inotifyfs
nodev   configfs
nodev   devpts
        squashfs
nodev   ramfs
nodev   autofs
nodev   fuse
        fuseblk
nodev   fusectl
```

```
# cat iomem
00000000-01ffffff : System RAM
00001000-0023a113 : Kernel code
0023a114-002b20bf : Kernel data
# cat version
linux version 2.6.22 (wzh@cybertan) (gcc version 4.2.3) #7 Thu Nov 10 16:04:37 CST
2011
#uname -a
Linux (none) 2.6.22 #7 Thu Nov 10 16:04:37 CST 2011 mips unknown
# cat partitions
major minor  #blocks  name
31     0        256 mtdblock0
  31    1       7872 mtdblock1
  31    2       6653 mtdblock2
  31    3         64 mtdblock3

# cat mtd
        dev:    size  erasesize name
        mtd0: 00040000 00010000 "boot"
        mtd1: 007b0000 00010000 "linux"
        mtd2: 0067f684 00010000 "rootfs"
        mtd3: 00010000 00010000 "nvram"
# CFE Environment Variables
Variable Name       Value
-------------------- --------------------------------------------------
BOOT_CONSOLE         uart0
CFE_VERSION         1.0.37
CFE_BOARDNAME       BCM947XX
CFE_MEMORYSIZE      32768
NET_DEVICE          eth0
NET_IPADDR          192.168.1.1
NET_NETMASK         255.255.255.0
NET_GATEWAY          0.0.0.0
NET_NAMESERVER      0.0.0.0
STARTUP              go;
#### Available Commands [Note that the lists are overlapping]
# Commands from bin
addgroup     delgroup     fgrep        more         pwd          umount
adduser      deluser      grep         mount        rm           uname
busybox      df           kill         msh          rmdir        usleep
cat          dmesg        ln           mv           sh           wps_monitor
chgrp        eapd         login        netstat      sleep
chmod        echo         ls           ping         su
cp           egrep        mkdir        ping6        touch
date         false        mknod        ps           true
# Commands from sbin
6rd_nud             hb_connect           qos_bw_detect
check_all_led       hb_disconnect        rc
check_ps            hotplug              reboot
check_ses_led       hotplug2             redial
check_wps_led       hotplug_2            resetbutton
ddns_checkip        ifconfig             restore
ddns_error          init                 rmmod
ddns_success        insmod               route
detectwan           ipupdated            sendudp
```

```
dhclient            klogd              ses_led
diag_pingbutton     listen              setreg
diag_tracertbutton  logread             stats
diagwpsbutton       lsmod               sulogin
disconnected_pppoe  misc                swapoff
erase               mkfs.minix          swapon
fdisk               monitor_cable       sysctl
filter              ntpd                syslogd
filtersync          pivot_root          udevtrigger
fsck.minix          power_led           wan_auto_detect
generate_md5sum     poweroff            waninfo
getreg              ppp_event           wl_iocmd
getty               preinit             write
gpio_check         process_monitor
halt                qos
# Busybox Functios
addgroup, adduser, basename, cat, chgrp, chmod,
clear, cp, cut, date, delgroup, deluser, df, dirname,
dmesg, du, echo, egrep, env, expr, false, fdisk, fgrep,
find, free, fsck.minix, getty, grep, halt, head, hostid,
id, ifconfig, insmod, kill, killall, klogd, less, ln,
login, logread, ls, lsmod, mkdir, mkfifo, mkfs.minix,
mknod, more, mount, msh, mv, netstat, passwd, ping, ping6,
pivot_root, poweroff, printf, ps, pwd, rdate, reboot,
reset, rm, rmdir, rmmod, route, sh, sleep, su, sulogin,
swapoff, swapon, sysctl, syslogd, tail, telnet, telnetd,
test, tftp, top, touch, true, umount, uname, uptime, usleep,
wget, xargs, yes
#CFE Commands [Note that CFE is accesible via spamming ESC or CTRL-C while rebooting
upgrade             Upgrade Firmware
et                  Broadcom Ethernet utility.
modify              Modify flash data.
nvram               NVRAM utility.
reboot              Reboot.
flash               Update a flash memory device
memtest             Test memory.
f                   Fill contents of memory.
e                   Modify contents of memory.
d                   Dump memory.
u                   Disassemble instructions.
batch               Load a batch file into memory and execute it
go                  Verify and boot OS image.
boot                Load an executable file into memory and execute it
load                Load an executable file into memory without executing it
save                Save a region of memory to a remote file via TFTP
ttcp                TCP test command.
tcp constest        tcp console test.
tcp listen          port listener.
tcp connect         TCP connection test.
rlogin              mini rlogin client.
client              Show the client of the dhcp server.
ping                Ping a remote IP host.
arp                 Display or modify the ARP Table
ifconfig            Configure the Ethernet interface
show clocks         Show current values of the clocks.
show heap           Display information about CFE's heap
show memory         Display the system physical memory map.
show devices        Display information about the installed devices.
unsetenv            Delete an environment variable.
printenv            Display the environment variables
setenv              Set an environment variable.
help                Obtain help for CFE commands
```

Annex C: Modified BCM4718.cfg to Connect OpenOCD to BCM5357

```
set _CHIPNAME bcm5357
set _LVTAPID 0x101ca17f
set _CPUID 0x1008c17f

source [find target/bcm47xx.cfg]
set _FLASHNAME winbond_flash.flash

flash bank $_FLASHNAME cfi 0x80001000 0x00800000 2 2 $_TARGETNAME
gdb_memory_map disable
```

Annex D: Binwalk on Dump from 0x80001000 for 8000000 bytes [base_addr=0x80001000]

```
> binwalk -B dump_0x80001000_8000000.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
46708         s0xB674          LZMA compressed data, properties: 0x5D, dictionary
size: 65536 bytes, uncompressed size: 291900 bytes
2617344       0x27F000        Linux kernel version 2.6.22
2641040       0x284C90        CRC32 polynomial table, little endian
2656556       0x28892C        CRC32 polynomial table, little endian
2852300       0x2B85CC        Unix path: /usr/gnemul/riscos/
2854956       0x2B902C        Unix path: /usr/lib/libc.so.1
2927975       0x2CAD67        Neighborly text, "NeighborSolicitsts"
2927999       0x2CAD7F        Neighborly text,
"NeighborAdvertisementsmp6OutDestUnreachs"
2928200       0x2CAE48        Neighborly text, "NeighborSolicitsirects"
2928228       0x2CAE64        Neighborly text, "NeighborAdvertisementssponses"
2930275       0x2CB663        Neighborly text, "neighbor
%.2x%.2x.%.2x:%.2x:%.2x:%.2x:%.2x:%.2x lost on port %d(%s)(%s)"
3182599       0x309007        Unix path: /usr/sbin/dhclient %s %s %s %s %s %s
3182700       0x30906C        Unix path: /usr/sbin/dhclient -r %s -cf %s -sf %s -lf
%s -pf %s %s
3183208       0x309268        Unix path: /usr/sbin/dhclient -6 -dec -sf %s -lf %s -pf
%s %s
3183660       0x30942C        Unix path: /usr/sbin/dhclient -nw -cf %s -sf %s -lf %s
-pf %s -bm %s %s &
3184765       0x30987D        Unix path: /usr/sbin/check_http.sh]
3184792       0x309898        Unix path: /usr/sbin/check_http.sh &
3185476       0x309B44        Unix path: /usr/sbin/ip -6 route del %s/%s
3203680       0x30E260        Unix path: /usr/sbin/ip -6 route show default
3204552       0x30E5C8        Unix path: /usr/sbin/ip -f inet6 addr flush %s scope
global
3205360       0x30E8F0        Unix path: /usr/sbin/ip -6 route flush table 200
```

```
3205768       0x30EA88        Unix path: /usr/sbin/ip -6 route del %s/%d dev %s
3207168       0x30F000        ELF, 32-bit LSB MIPS-I shared object, MIPS, version 1
(SYSV)
3271913       0x31ECE9        HTML document footer
3309568       0x328000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
3336276       0x32E854        Unix path: /var/run/dhcpd.pid br0 ]
3373299       0x3378F3        Linux kernel version 2.6.22
4358048       0x427FA0        Unix path: /var/run/dhcpd.pid
4561936       0x459C10        XML document, version: "1.0"
4563516       0x45A23C        Unix path: /usr/lib/libnvram.so
4624384       0x469000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
4628480       0x46A000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
4687200       0x478560        XML document, version: "1.0"
4688176       0x478930        XML document, version: "1.0"
4698224       0x47B070        SHA256 hash constants, little endian
4698532       0x47B1A4        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/bcmcrypto/random.c
4710968       0x47E238        Unix path: /usr/lib/libnetconf.so
4778076       0x48E85C        Base64 standard index table
4782112       0x48F820        XML document, version: "1.0"
4786580       0x490994        XML document, version: "1.0"
4791056       0x491B10        XML document, version: "1.0"
4813720       0x497398        Unix path: /etc/config/resolv.conf
4911085       0x4AEFED        Unix path: /usr/sbin/upnp
4965272       0x4BC398        Unix path: /etc/config/resolv.conf
4976640       0x4BF000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
4986972       0x4C185C        Base64 standard index table
5056916       0x4D2994        XML document, version: "1.0"
5061392       0x4D3B10        XML document, version: "1.0"
5065744       0x4D4C10        XML document, version: "1.0"
5158944       0x4EB820        XML document, version: "1.0"
5270808       0x506D18        Base64 standard index table
5374656       0x5202C0        XML document, version: "1.0"
5375664       0x5206B0        XML document, version: "1.0"
5396880       0x525990        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/router/nas/nas_wksp.c
5397728       0x525CE0        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/router/nas/nas_wksp_radius.c
5398972       0x5261BC        CRC32 polynomial table, little endian
5505024       0x540000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
5510040       0x541398        Unix path: /etc/config/resolv.conf
5537904       0x548070        SHA256 hash constants, little endian
5538212       0x5481A4        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/bcmcrypto/random.c
5546548       0x54A234        Unix path: /usr/lib/libnvram.so
5632360       0x55F168        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/wps/common/shared/slist.c
5652480       0x564000        ELF, 32-bit LSB executable, MIPS, version 1 (SYSV)
5685230       0x56BFEE        Unix path: /usr/sbin/nas
5689788       0x56D1BC        CRC32 polynomial table, little endian
5754228       0x57CD74        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/wps/brcm_apps/linux/wps_linux_main.c
5856240       0x595BF0        CRC32 polynomial table, little endian
5861956       0x597244        Unix path: /usr/lib/libnvram.so
5873664       0x59A000        ELF, 32-bit LSB MIPS-I shared object, MIPS, version 1
```

```
(SYSV)
5894256       0x59F070        SHA256 hash constants, little endian
5894564       0x59F1A4        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/bcmcrypto/random.c
5907352       0x5A2398        Unix path: /etc/config/resolv.conf
5910528       0x5A3000        ELF, 32-bit LSB MIPS-I shared object, MIPS, version 1
(SYSV)
5914912       0x5A4120        Unix path: /usr/sbin/acsd
6036872       0x5C1D88        Unix path: /home/hhm/work/E1200v2-
0825/e1200v2_2.0.10.001/src/bcmcrypto/bn.c
6157208       0x5DF398        Unix path: /etc/config/resolv.conf
7579956       0x73A934        Copyright string: "Copyright (C) 2000-2008 Broadcom
Corporation."
7598165       0x73F055        HTML document header
7598250       0x73F0AA        HTML document footer
7606864       0x741250        CRC32 polynomial table, little endian
7619392       0x744340        HTML document header
7620498       0x744792        HTML document footer
7620508       0x74479C        HTML document header
7620623       0x74480F        HTML document footer
7620632       0x744818        HTML document header
7621034       0x7449AA        HTML document footer
7621044       0x7449B4        HTML document header
7621468       0x744B5C        HTML document footer
7621476       0x744B64        HTML document header
7622204       0x744E3C        HTML document header
7622922       0x74510A        HTML document footer
```