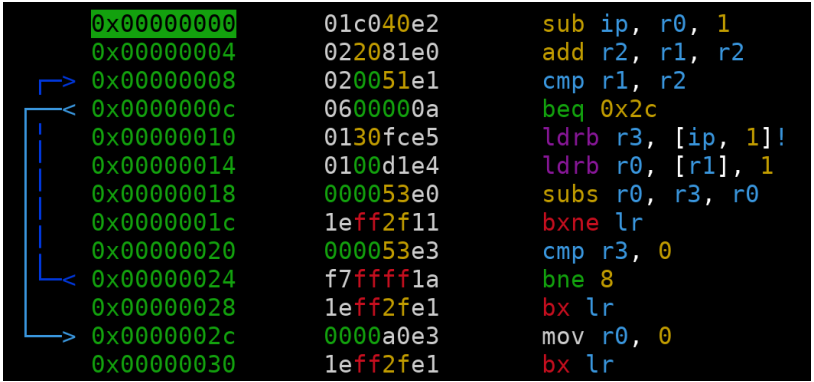
ENGR-E 399/599: Embedded systems reverse engineering

Lecture 10: JTAG

Austin Roach
ahroach@iu.edu

March 24, 2022

Mystery function



- How many arguments does the function take?
- What are the types of the arguments?
- What does the function do?
- What does the function return?

Mystery function revealed

```
int strncmp(const char *s1, const char *s2, size_t n)
```

Description

The strncmp() function is similar to strcmp(), except it compares only the first (at most) n bytes of s1 and s2.

Return value

The strncmp() function returns an integer less than, equal to, or greater than zero if s1 (or the first n bytes thereof) is found, respectively, to be less than, to match, or be greater than s2.

Today's plan

- JTAG overview
  - History
  - JTAG scan chains
  - JTAG TAP registers
  - JTAG TAP state machine
- Identifying JTAG test access ports
  - JTAGulator
- Use of processor debug access for reverse engineering
  - openocd
- Demonstration on WiFi router
- ST Nucleo F429ZI debug access port demo

The JTAG terminology problem

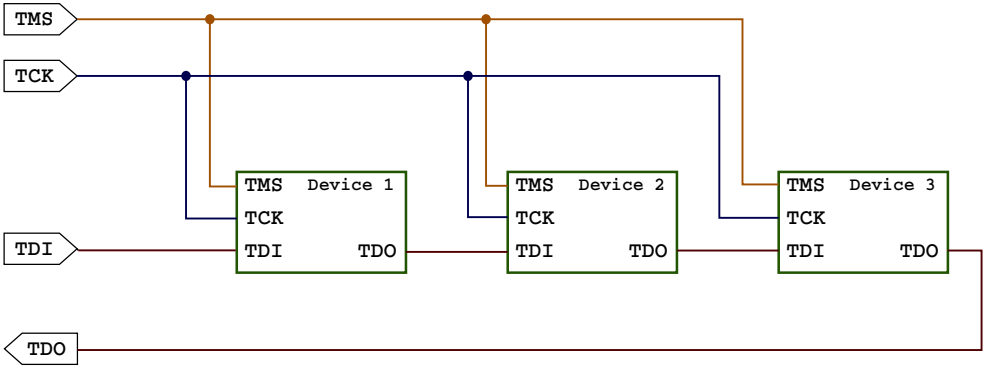
- **EE:** "You should design in a complete JTAG boundary scan chain to allow for board test before shipping."
- **1337 h4x0r:** "That thing has JTAG!!11!1" **Other 1337 h4x0r:** "h00r4y! pwn3ed!"
- **Instructable for gamers:** "How to JTAG Your Xbox 360 and Run Homebrew"

- How are all of these people using the same term in such different ways?
- What do all of these things have in common?

The JTAG terminology resolution

- JTAG (IEEE 1149.1) is a standard for a test access port supporting boundary scan
- Originally designed by the **J**oint **T**est **A**ction **G**roup
- Standard defines a protocol for manipulating a set of debug registers
- On top of this standard, some processor manufacturers built debug access ports allowing production and control of internal processor state
  - This can be very useful for reverse engineering
- Some of the earliest modifications for game consoles exploited processor debug access ports
  - Among some gamers, "a JTAG" is a generic term used to refer to any hardware modification

JTAG scan chains



This is a typical configuration, but standard allows flexibility.

A focus on the boundary scan register

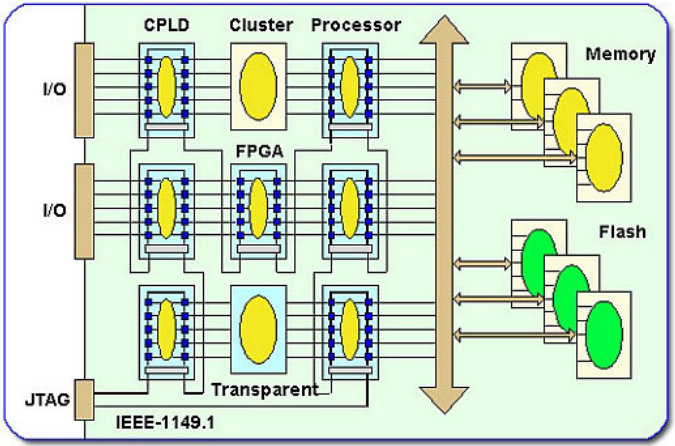
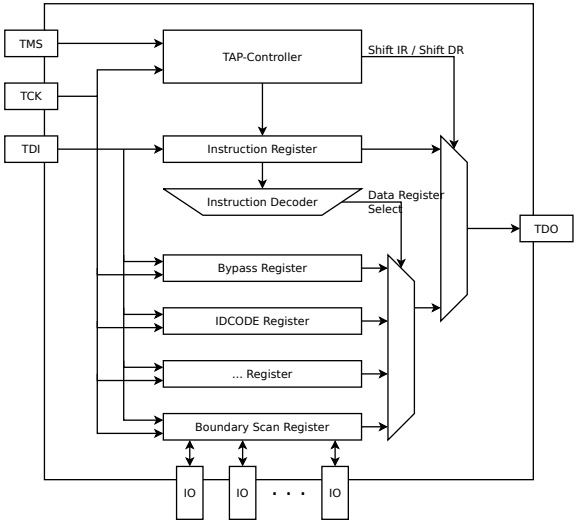


Image credit: Corelis boundary scan tutorial. https://www.corelis.com/education/tutorials/boundary-scan/

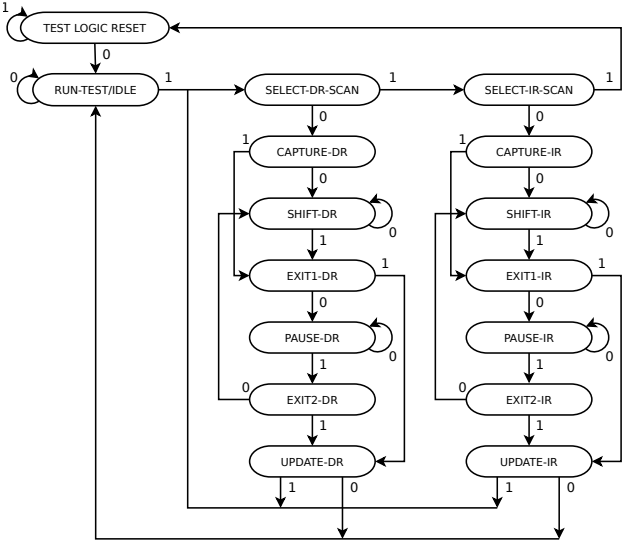
Internal JTAG device registers



- Instruction and data registers implemented as shift registers
- TAP state machine and/or current instruction determine which register connected across TDI/TDO
- Instruction register length *not* standardized
- Vendors can customize available data registers (adding custom instructions)

Image credit: Rudolph H. "File:JTAG Register.svg," Wikimedia Commons, https://commons.wikimedia.org/w/index.php?title=File:JTAG_Register.svg&oldid=241195158.

JTAG TAP controller state machine



- TMS sampled on rising edge of TCK to govern state transitions
- Clocking five 1s returns to `Test-Logic-Reset` from any other state
- May also include a `TRST` signal to reset the TAP controller

Image credit: Rudolph H. "File:JTAG TAP Controller State Diagram.svg", Wikimedia Commons, https://commons.wikimedia.org/w/index.php?title=File:JTAG_TAP_Controller_State_Diagram.svg&oldid=179981764.

Sampling of standard electrical test instructions

Name	Required?	Description
BYPASS	Yes	TDI/TDO connected through single-bit bypass register
SAMPLE	Yes	All pin I/O states stored in BSRs in `Capture-DR` state without interrupting device operation
PRELOAD	Yes	Access BSR to shift in data without interrupting device operation
EXTEST	Yes	`Capture-DR` state: input I/O pin values stored in BSRs `Update-DR` state: output I/O pin values set by BSRs
IDCODE	No	Access Device Identification register (32-bit register)

- Other optional instructions clamp output values, set impedance states, allow programming user-programmable IDs, etc.
- Encoding of all instructions except `BYPASS` set by vendor
  - `BYPASS` instruction is all 1s

What is boundary scan used for?

- Test electrical connections
  - No opens or shorts
- Drive pin states to write external memory
  - No need for additional flying lead contact points
- Sample pin states during operation to debug
  - Low speed logic analyzer

Electrical test information contained in **B**oundary **S**can **D**escription **L**anguage (BSDL) files, to aid in the development of electrical tests.

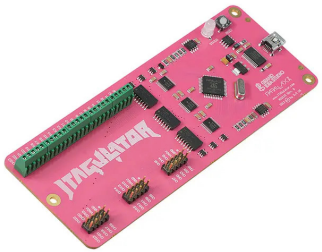
Identifying pinouts of JTAG TAPs

Some handy requirements from IEEE 1149.1:

- If `IDCODE` data register is present it must be selected in `Test-Logic-Reset` state
  - Otherwise, `BYPASS` data register must be selected
- The LSB of the `IDCODE` data register must be 1 (first bit shifted out)
- The `BYPASS` instruction must be all 1s shifted into the `IR`

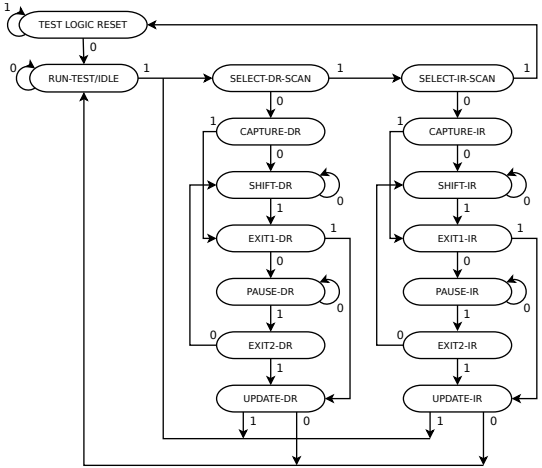
Tools for discovering JTAG interfaces

- JTAGenum (https://github.com/cyphunk/JTAGenum)
  - Arduino or Raspberry Pi host
  - Implement algorithms that we discuss on the following slides
  - Hardware platforms have poor electrical protection
- JTAGulator (http://www.grandideastudio.com/jtagulator/) (https://github.com/grandideastudio/jtagulator/)
  - Similar algorithms to JTAGenum
  - Thoughtful input protection
  - $$
- UrJTAG (http://www.urjtag.org)
  - Uses a variety of JTAG-enabled debug pods
  - Low level access to JTAG interaction



Algorithms: ID code scan

- Try all possible permutations of candidates for TCK, TMS, TDO
- Attempt to enter `Shift-DR` state from `Test-Logic-Reset`
  - Standard specifies that, if present, `IDCODE` register selected in `Test-Logic-Reset` state
- Attempt to shift out data register
- Successful match if:
  - First bit is 1
  - The contents are not all 0xffff...
- Attempt to return to `Test-Logic-Reset`

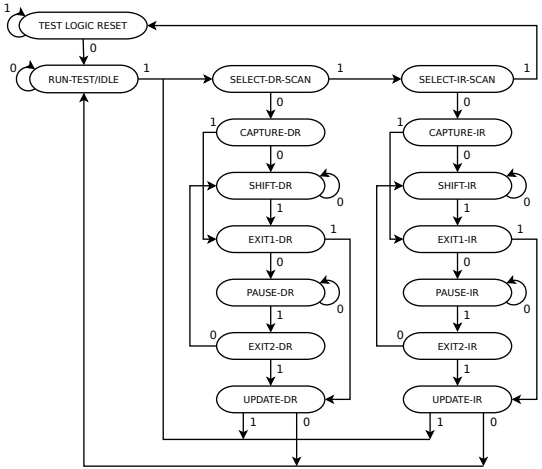


IDCODE_scan() from JTAGulator.spin

```
jTDI := g#PROP_SDA    ' TDI isn't used when we're just shifting data from the DR. Set TDI to a temporary
num := 0      ' Counter of possible pinouts
ctr := 0
repeat jTDO from chStart to chEnd   ' For every possible pin permutation (except TDI and TRST)...
  repeat jTCK from chStart to chEnd
    if (jTCK == jTDO)
      next
    repeat jTMS from chStart to chEnd
      if (jTMS == jTCK) or (jTMS == jTDO)
        next
...
      jtag.Config(jTDI, jTDO, jTCK, jTMS, jTCKSpeed)   ' Configure JTAG
      jtag.Get_Device_IDs(1, @value)        ' Try to get the 1st Device ID in the chain (if it exists) by
      if (value <> -1) and (value & 1)      ' Ignore if received Device ID is 0xFFFFFFFF or if bit 0 != 1
        Display_JTAG_Pins                   ' Display current JTAG pinout
        num += 1                            ' Increment counter
        xtdo := jTDO                        ' Keep track of most recent detection results
        xtck := jTCK
        xtms := jTMS
        jPinsKnown := 1                     ' Enable known pins flag
...
```

Algorithms: BYPASS scan

- Try all possible permutations of TCK, TMS, TDO, *and TDI*
- Attempt to enter `Shift-IR` from `Test-Logic-Reset` state
- Shift in a ton of 1s (selecting `BYPASS` instruction)
- Attempt to enter `Shift-DR` state
- Shift in a ton of 1s (filling `BYPASS` DRs with 1s)
- Shift in a zero on TDI, count clock cycles until it emerges on TDO
- Shift in additional random patterns and observe output to gain confidence



BYPASS scan implementation

- See `BYPASS_scan()` in JTAGulator.spin
- Refers to:
  - `Detect_Devices()` in propJTAG.spin
  - `Bypass_Test()` in propJTAG.spin

What can go wrong?

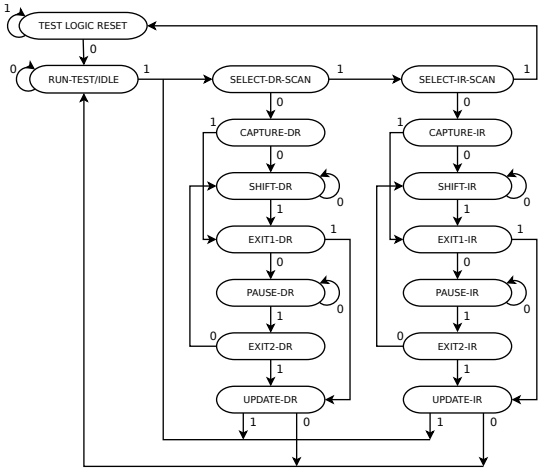
Possibility of false positives and false negatives:

- Device does not contain Device ID register
- More devices than expected on the scan chain
- Unexpectedly large instruction register lengths
- TDO candidate is some other output pin that just happens to output something like expected result

Further JTAG TAP properties: Determine instruction register length

Assumes a single device on the scan chain

- Enter `Shift-IR` state
- Shift in a lot of 1s
- Shift in a 0 on TDI
- Count number of clock cycles until 0 observed on TDO



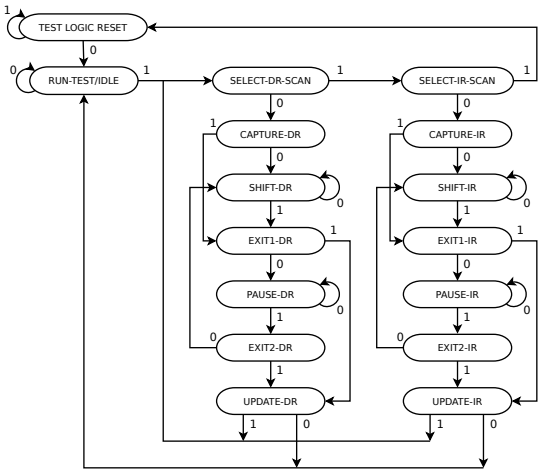
See `Detect_IR_Length()` in PropJTAG.spin

Further JTAG TAP properties: Determine data register lengths

Assumes a single device on the scan chain

For each possible instruction:

- Enter `Shift-DR` state
- Shift in a lot of 1s
- Shift in a 0 on TDI
- Count number of clock cycles until 0 observed on TDO
- Gives an idea of what instruction values are implemented (DR length > 1) versus implemented (DR length $== 1$)
  - Unused instruction register codes equivalent to the `BYPASS` instruction



See `Detect_DR_Length()` in PropJTAG.spin

Further JTAG TAP properties: ID codes

- IEEE 1149.1 specifies ID code fields (32 bit value)
  - Version
  - Part number
  - Manufacturer ID
  - LSB 1
- Can learn about devices by comparing JEDEC manufacturing IDs
- Sometimes just Googling the ID code yields useful information

Processor debug access ports

JTAG provides:

- Widely adopted debug interface
- General mechanism for reading to and writing from data registers

Can build processor debug interfaces on top of this:

- Read and write memory
- Halt processor
- Read and modify registers
- Set breakpoints
  - Triggered when program counter reaches target address (instruction access)
- Set watchpoints
  - Triggered by read/write memory access at target address (data access)
- Reset processor
- Configure core parameters

These features are specific to a particular type of processor core

Standard debug access port connectors

Some standard physical interfaces:

- http://www.keil.com/support/man/docs/ulink2/ulink2_hw_connectors.htm
- http://www2.lauterbach.com/pdf/app_arm_jtag.pdf
- http://www.jtagtest.com/pinouts/

But many interfaces implemented as test points or with custom headers

Variations

- Serial Wire Debug (SWD)
  - Uses only two signal wires (SWDIO, SWCLK)
  - Uses TMS, TCK signal wires on devices that also provide full JTAG interface
- Multiple cores on scan chain
- JTAG Route Controller (JRC) to dynamically insert or remove devices from scan chain
- Any number of manufacturer/processor-specific variations

A cautionary note about IEEE 1149.1 compliance

From the ARM Debug Interface Architecture Specification:

3.3.3 IMPLEMENTATION DEFINED extensions to the IR instruction set

The eight IR instructions 0b0000 to 0b0111 are reserved for IMPLEMENTATION DEFINED extensions to the DAP.

These instructions can be used for accessing a boundary scan register, for IEEE 1149.1 compliance. The instructions required to do this are listed in Table 3-3. All these instructions select the boundary scan data register.

Note

This extension describes only the boundary scan instructions described by IEEE 1149.1-1990 and IEEE 1149.1-2001. Other editions of IEEE 1149.1 might define additional instructions. The extension only supports up to eight additional instructions.

ARM recommends that:

- Separate JTAG TAPs are used for boundary scan and debug.
- The instructions listed in Table 3-3 are not implemented.

Recommends non-compliance with IEEE 1149.1

If the IR register is set to an IR instruction value that is not implemented, or reserved, then the BYPASS register is selected.

Table 3-3 Recommended IMPLEMENTATION DEFINED IR instructions for IEEE 1149.1-compliance

IR instruction value	Instruction	Required by IEEE 1149.1?
0b0000	EXTEST	See note in main text.
0b0001	SAMPLE	Yes
0b0010	PRELOAD	Yes
0b0011	Reserved	-
0b0100	INTEST[a]	No
0b0101	CLAMP[a]	No
0b0110	HIGHZ[a]	No
0b0111	CLAMPZ[a]	No. See *The CLAMPZ instruction*.

a. Reserved, if the instruction is not implemented.

If you require a boundary scan implementation, you must implement the instructions that are required by IEEE 1149.1. The other IR instruction values listed in Table 3-3 are reserved encodings that must be used if that function is implemented in the boundary scan. If implemented, these instructions must behave as required by the IEEE 1149.1 specification. If not implemented, they select the BYPASS register.

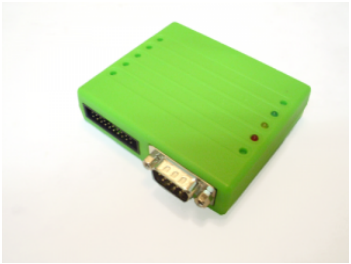
Processor debug access for reverse engineering

What do we want to accomplish?

- Read memory to extract firmware
- Dynamic analysis
  - Set breakpoints and watchpoints
  - Observe execution
- Modify memory or register states
  - Force execution path that may be hard to reach otherwise

Tools for JTAG interfaces

- Segger J-link
- Flyswatter 2
- MPSSE cable
- Other debug adapter pods
- Hardware hacker tools
  - Bus Pirate, etc.



Software for debug access

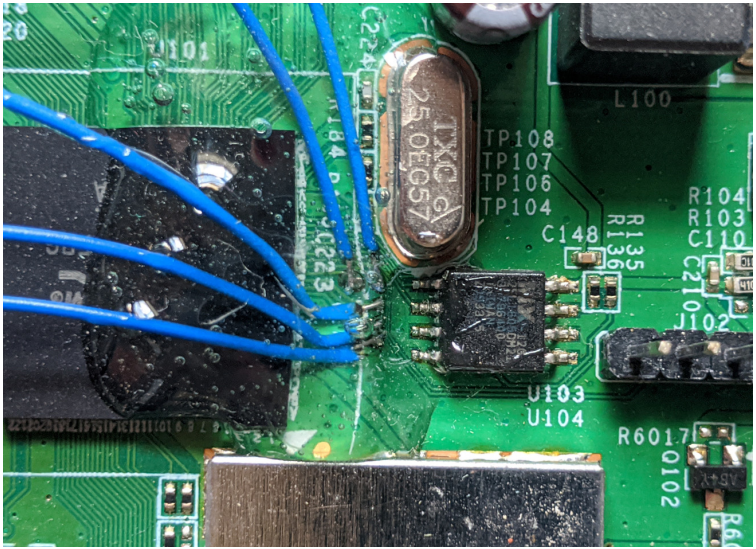
- openocd (http://openocd.org/)
- Other proprietary or specialized software products

If you really care about what's going on behind the scenes with ARM debug access ports:
ARM Cortex Debug Interface Architecture
https://developer.arm.com/documentation/ihi0031/latest

Find the JTAG interface: WiFi router board

Find the JTAG interface: WiFi router board closeup

Electrical connection to probe points



JTAGulator ID code scan

JTAGulator BYPASS scan

urjtag device detect

```
user@t470:~$ jtag

UrJTAG 0.10 #2007
Copyright (C) 2002, 2003 ETC s.r.o.
Copyright (C) 2007, 2008, 2009 Kolja Waschk and the respective authors

UrJTAG is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
There is absolutely no warranty for UrJTAG.

warning: UrJTAG may damage your hardware!
Type "quit" to exit, "help" for help.

jtag> cable Flyswatter
Connected to libftdi driver.
jtag> frequency 500000
Setting TCK frequency to 500000 Hz
jtag> detect
IR length: 5
Chain length: 1
Device Id: 00010000000010001100000101111111 (0x1008C17F)
  Manufacturer: Broadcom (0x17F)
  Unknown part! (0000000010001100) (/usr/share/urjtag/broadcom/PARTS)
jtag>
```

Finding more about the BCM5355

Broadcom

General information | Wireless chipsets | SoC | PLC | Switch [edit]

WiSoC [edit]

bg [edit]

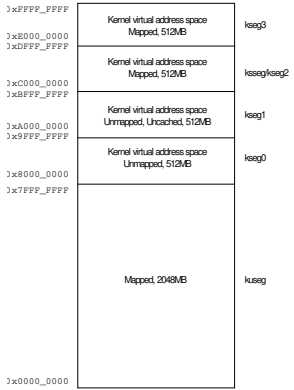
Chip	Radio	WLC rev.	PHY rev.	Arch.	Clock speed	PHY modes	First seen (FCC)	Notes	ESystems	PR / PPage
BCM4712	BCM2050	7/2	G r2	BMIPS3300	200 MHz	bg	2003-11-24		46 devices	PPage RD RG
BCM5350	BCM2050	7/2	G r2	BMIPS3300	200 MHz	bg	2004-07-21	in. FE switch	13 devices	PR RD
BCM5352	BCM2050	9/7	G r7	BMIPS3300	200 MHz	bg	2005-03-29	int. FE switch	5 devices	.
BCM5352E	BCM2050	9/7	G r7	BMIPS3300	200 MHz	bg	2005-05-17	int. FE switch, Encore DSP	16 devices	PPage
BCM5354	BCM2062 r1	13/0	LP r0	BMIPS3300	240 MHz	bg	2007-01-09	int. FE switch, Encore DSP	44 devices	PR RD PPage
BCM5355	...	...	...	MIPS 74K	333 MHz	bg	2009-09-17	Int. FE switch, seems to identify as BCM5356	1 devices	

Source: WikiDevi (RIP) https://web.archive.org/web/20191031174603/wikidevi.com/wiki/Broadcom#tab=SoC

MIPS 74k memory address space

From MIPS 74K Processor Core Family Software User Manual:

Figure 5.6 Kernel Mode Virtual Address Space



Physical memory address 0x0 represented in `kseg0` (cached) and `kseg1` (uncached)

MIPS 74k reset address

From MIPS 74K Processor Core Family Software User Manual:

8.1.5 Fetch Address

Upon Reset, unless the EJTAGBOOT option is used, the fetch is directed to VA 0xBFC00000 (PA 0x1FC00000). This address is in kseg1, which is unmapped and uncached, so that the TLB and caches do not require hardware initialization.

Could this be mapped to the SPI flash memory?

openocd script

```
# OpenOCD config for Belkin F5D7234-4 V5 with Flyswatter interface
# According to https://wikidevi.com/wiki/Broadcom, processor core is
# MIPS 74K, which is the same core as many of the bcm47xx processors.
# We use target/bcm4718.cfg as a template here.

source [find interface/ftdi/flyswatter2.cfg]
adapter_khz 5000
transport select jtag

set _CHIPNAME bcm5355
set _LVTAPID 0x1535617f
set _CPUID 0x0008c17f

source [find target/bcm47xx.cfg]
```

openocd memory dump

```
user@t470:~/lecture-11$ openocd -f belkin_f5d7234_flyswatter.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
Forcing reset_config to none to prevent OpenOCD from pulling SRST after the switch from LV is a
lready performed
none separate
switch_lv_to_ejtag
Info : clock speed 5000 kHz
Info : JTAG tap: bcm5355-lv.tap tap/device found: 0x1535617f (mfg: 0x0bf (Broadcom), part: 0x53
56, ver: 0x1)
Info : JTAG tap: bcm5355-lv.tap disabled
Info : JTAG tap: bcm5355.cpu enabled
```

```
user@t470:~/lecture-11$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
target halted in MIPS32 mode due to debug-request, pc: 0x80007480
> mdb 0xbfc00000 0x10
0xbfc00000: 00 b8 12 3c 50 53 07 24 00 00 51 8e ff ff 0a 34
> dump_image mem.bin 0xbfc00000 0x200000
```

ST Nucleo F429ZI: openocd init

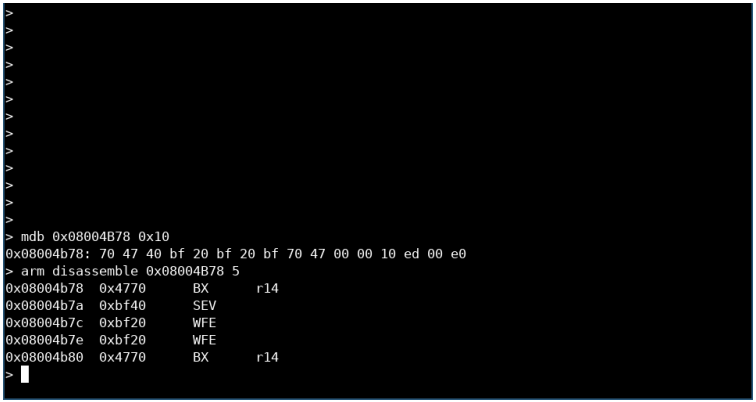
```
user@beaglebone:~$ openocd -f board/st_nucleo_f4.cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ comp
ared to plain JTAG/SWD
adapter speed: 2000 kHz
adapter_nsrst_delay: 100
none separate
srst_only separate srst_nogate srst_open_drain connect_deassert_srst
Info : Unable to match requested speed 2000 kHz, using 1800 kHz
Info : Unable to match requested speed 2000 kHz, using 1800 kHz
Info : clock speed 1800 kHz
Info : STLINK v2 JTAG v29 API v2 SWIM v18 VID 0x0483 PID 0x374B
Info : using stlink api v2
Info : Target voltage: 3.238667
Info : stm32f4x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

```
user@beaglebone:~$ telnet localhost 4444
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
>
```

ST Nucleo F429ZI: openocd view registers

```
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x61000800 pc: 0x08004b78 psp: 0x200006d8
> reg
===== arm v7m registers
(0) r0 (/32): 0xE000ED10
(1) r1 (/32): 0x00000001
(2) r2 (/32): 0x00000000
(3) r3 (/32): 0x00000001
(4) r4 (/32): 0x20000B10
(5) r5 (/32): 0x00000000
(6) r6 (/32): 0x00000000
(7) r7 (/32): 0x00000000
(8) r8 (/32): 0x00000000
(9) r9 (/32): 0x00000000
(10) r10 (/32): 0x00000000
(11) r11 (/32): 0x00000000
(12) r12 (/32): 0x00000000
(13) sp (/32): 0x200006D8
(14) lr (/32): 0x0800CDC7
(15) pc (/32): 0x08004B78
(16) xPSR (/32): 0x61000800
(17) msp (/32): 0x2002FFD8
```

ST Nucleo F429ZI: openocd view memory



ST Nucleo F429ZI: openocd breakpoints

```
>
>
>
>
>
>
> halt
> bp 0x8004b78 2 hw
breakpoint set at 0x08004b78

> resume
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000800 pc: 0x08004b78 psp: 0x200006d8
> resume
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000800 pc: 0x08004b78 psp: 0x200006d8
> rbp 0x8004b78
> resume
>
```

ST Nucleo F429ZI: openocd single stepping

```
>
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x61000800 pc: 0x08004b78 psp: 0x200006d8
> step
target halted due to single-step, current mode: Thread
xPSR: 0x61000000 pc: 0x0800cdc6 psp: 0x200006d8
halted: PC: 0x0800cdc6
> step
target halted due to single-step, current mode: Thread
xPSR: 0x61000000 pc: 0x0800cdca psp: 0x200006e0
halted: PC: 0x0800cdca
> step
target halted due to single-step, current mode: Thread
xPSR: 0x61000000 pc: 0x0800a774 psp: 0x200006e0
halted: PC: 0x0800a774
> step
target halted due to single-step, current mode: Thread
xPSR: 0x61000000 pc: 0x0800a776 psp: 0x200006e0
halted: PC: 0x0800a776
> resume
>
```