Practical JTAG: From 0 to 1

HyperChem
Tencent’s Xuanwu Lab
http://xlab.tencent.com  @XuanwuLab
> #whoami

- Security Researcher
- Used to doing Chemistry;
- Interested in:
  - Console Hacking;
  - Embedded Device Security;
  - Firmware Reverse and Emulation;
  - Unpacking and Un-virtualizing;
  - Geek Stuff: RFID, lock-picking, Device hacking;
• 1. Theory On JTAG;
• 2. Use JTAG in the Wild;
• 3. Let’s Do It;
> #cat /theory/jtaginfo

Let's Make a JTAG Jasper Xbox 360!
MrMario2011 • 18万次观看 • 10个月前
What is this, 2009?! No typo right there, this is not an ROH or even an R-JTAG, this is a legitimate original JTAG modification being

JTAG repair for Samsung I9000 Galaxy S (Unbrick, repair)
Multi-COM • 26万次观看 • 7年前
More information about this product? / Więcej informacji na temat tego produktu? / PL version:
http://multi-con.pl/ahrt-VGQsTJMju

使用JTAG和CrossWorks for ARM调试 Rapsberry Pi Shiıyòng JTAG hé
CrossWorks for ARM tiáoshi Rapsberry Pi
Rowley Associates • 720万次观看 • 6个月前
目录: 00:00 - 介绍 00:07 - 需要物品 00:54 - 安装Raspberry Pi 04:11 - JTAG接线 Mū: 00:00 - Jiēshào 00:07 - Xìyào ...
> # cat /theory/jtaginfo

- JTAG=Join Test Action Group;
- ISO IEEE 1149.1;
- Designed for IC chip test;
- Access logic signal inside IC chip and pins;
- Three major features:
  - Debugging;
  - Storing firmware;
  - Boundary scan testing;
> #cat /theory/benefits

- Bug check for IC chip and peripherals;
- Flashing firmware in system;
- Full Accessing to address space;
- Online Debugging;
- Full Accessing to Registers and Internal Bus Line;
- Dumping firmware;
- Full Controlling CPU execution;
- Talking to peripherals;
> #cat /theory/shift_register

**Integrated Circuit**

**peripherals**

- **Shift data out 1bit/clk**
- **Boundary Scan Chain**
- **Shift data in 1bit/clk**

**Digital Output Pin**
> #cat /theory/electric_structure

- Five Pins for JTAG:
  - TMS-Test Mode Select;
  - TCK-Test Clock;
  - TDI-Test Data Input;
  - TDO-Test Data Output;
  - TRST-Test Reset;

- Serial Connection and Communication;

Connect to the head and tail of Scan Chain
> #cat /theory/tap_controller

```plaintext
> #cat /theory/tap_controller

---

TAP Controller

Selection Circuity

Bypass Register

Instruction Register

IDCode Register

Boundary Scan Chain

TDO

TDI

TMS

TCK

Test Access Port

TENCENT'S XUANWU LAB
```
> #cat /theory/state_machine

Change TMS signal by TCK
> #cat /theory/jtag_steps

- Core: Manipulating TMS signal along with Clock;
- Common Procedures:
  - Select IR;
  - Set IR value to do specific operation;
  - Select a DR;
  - Shift in /out DR value;
  - Update DR value /Parse DR value;

How to debug and access memory!?
JTAG was designed to assist with device, board, and system testing, diagnosis, and fault isolation;

No feature for debugging in the original version of JTAG;

Debugging need to halt, run and step CPU;

Memory access need to access internal memory bus;

New Boundary Scan Chain and Instructions are added for this;
> #cat /semi-theory/arm7tmdti

- Instruction Register length: 4;
- JTAG Instructions:
  - IDCODE: 1110, Read 32-bit ID for Core;
  - SCAN_N: 0010, Select one of four Scan Chains;
  - BYPASS: 1111, pass through test, 1 clock delay;
  - INTEST: 1100, Set Selected Scan Chain to Internal Test Mode;
  - RESTART: 0100, Set ARM CPU back to normal mode;
• Four Scan Chains:
  • Scan Chain 0:
    • Len: 113 bits, include: data bus, address bus, core signal and debug control signal;
  • Scan Chain 1:
    • Len: 33 bits, include: 32 bits data bus, 1 bit BREAKPT signal;
  • Scan Chain 2:
    • Len: 38 bits, debug control registers, to set breakpoints, watch points etc
  • Scan Chain 3:
    • For accessing external scan chains;
CPU DO

- Read Instruction From Memory To Data Bus
- Decode Instruction to determine what to do
- Execute Instruction: set registers, calculation, memory access
  Value of registers appears on Data Bus

We DO

- Change Instruction to Memory writing on Data Bus
- Wait...
- Change registers, wait for accessing, read result.
Jtag_SelectScanN(1);
Jtag_ShiftIR(ARMJTAG_INTEST,UpdateIR); //INTEST means no outer access.
Jtag_ChangeData(0xe8900003,FALSE); //LDMIA r0, {r0, r1}
Jtag_ChangeData(ARM_INSTR_NOP,FALSE); //NOP
Jtag_ChangeData(address,FALSE); //set r0=address
Jtag_ChangeData(data,FALSE); // set r1=data
Jtag_ChangeData(ARM_INSTR_NOP,FALSE); //NOP
Jtag_ChangeData(ARM_INSTR_NOP,FALSE); //NOP
    // set brkpt to enable outer access for once
Jtag_ChangeData(ARM_INSTR_NOP,TRUE);
Jtag_ChangeData(0xe4801000,FALSE,FALSE);//STR r1, [r0] // do mem write
> # cat /semi-theory/jtag_in_real_world

- Many IP Core provide internal signals via JTAG;
- Utilizing JTAG accessing memory and registers needs knowledge about ISA;
- Need tricks to improve efficiency for debugging and mem dumping;
- Tools available for doing these jobs;

Are you ready for JTAG debugging?
Where are the PINs of JTAG?
> #cat /practice/jtag/pins

- Identify PINs for JTAG: TDI, TDO, TMS, TCK;

- Ways:
  - Silk Print On PCB;
  - Read Data Sheet;
  - Enumerate possible PINs;
  - Search for Group PINs
> #cat /practice/jtag/pins

• Sometimes, No obvious silk print on pcb, we have to guess;
• Find out possible pins for JTAG, test every combination;
• Criteria for right JTAG:
  • No capacitor to GND or Vcc;
  • Usually 10K pull-up or down;
  • Usually IDCODE shows up while JTAG reset;
  • IDCODE have a format;
Do I need CPU info for JTAG?
> #cat /practice/jtag/cpu_cfg

- CPU info is necessary for jtag connection and debugging;
- When Pins Found & JTAG enable → read CPU IDCODE;
- IDCODE Register (32bits) is connected between TDI and TDO when JTAG reset;
- Shift it out and Get IDCODE; Jtagulator does this too
- Example: 0x2548217F, BCM5482

<table>
<thead>
<tr>
<th>Ver.</th>
<th>Design Center</th>
<th>Core Number</th>
<th>Chip Derivative</th>
<th>Manufacturer ID</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.. 28</td>
<td>27..22</td>
<td>21..17</td>
<td>16..12</td>
<td>11..1</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>010101</td>
<td>00100</td>
<td>00010</td>
<td>00010111111</td>
<td>1</td>
</tr>
</tbody>
</table>
>`#cat /practice/jtag/cpu_cfg`

- Find Detail Info by IDCODE:
  - Manufacturer website;
  - `bsdl.info`;
  - OpenOCD directory;

- Looking for:
  - IP Core Type;
  - JTAG Instructions;
  - Boundary Scan Chains;
  - Instruction Register Length;

- OpenOCD makes life easier;
Who help me translate Binary to signal?
> #cat /practice/jtag/adapter

- Connected between computer and your testing pcb board;
- Converting logic data to digital signal to manipulate JTAG pins;
- Exposed interface for higher level:
  - Parallel Port;
  - Serial /COM Port;
  - TCP/IP Port;
  - Private Interface in commercial product;
  - FTDI chip Port;
> #cat /practice/jtag/adapter

- FT232H/FT2232H, USB 2.0 Hi-Speed (480Mb/s) to UART/FIFO IC;
- Frequently used in open source adapter: bus pirate;
- Configurable on host to simplify serial protocol (JTAG) communication;
- Easy to use, supported by many software;

Much Cheaper

~5$

Canada
> #cat /practice/target

Define an adapter?
> # cat /practice/jtag/adapter_cfg

- Define Type of Adapter, like ftdi;
- Tell software know how to communicate with adapter;
- Set initial voltage level for adapter Pins;
- Key for talking to adapter;

```c
interface ftdi
ftdi_device_desc "Dual RS232-HS"
ftdi_vid_pid 0x0403 0x6014

ftdi_layout_init {levelbits} {direction bits}
ftdi_layout_signal nTRST-data 0x0100 -noe 0x0400
ftdi_layout_signal nSRST-data 0x0200 -noe 0x0800
```
Write a cfg:
- Determine type;
- Find out vid:pid;
- Set Adapter Name;
- Set bits for Pins: direction and level

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Direction</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCK</td>
<td>output</td>
<td>low</td>
</tr>
<tr>
<td>TDI</td>
<td>output</td>
<td>low</td>
</tr>
<tr>
<td>TDO</td>
<td>input</td>
<td>low</td>
</tr>
<tr>
<td>TMS</td>
<td>output</td>
<td>low</td>
</tr>
<tr>
<td>GPIOL0</td>
<td>output</td>
<td>high</td>
</tr>
<tr>
<td>GPIOL1</td>
<td>input</td>
<td>low</td>
</tr>
<tr>
<td>GPIOL2</td>
<td>input</td>
<td>low</td>
</tr>
</tbody>
</table>

LevelBits: 0x0010
Direction Bits: 0x001b
>`#cat /practice/target`

Who knows My adapter?
Connect and communicate with adapter;
Unify cmds for different architectures and cores;
Generate JTAG signals depends on IP core at lower level;
Commercial: Jlink, Xjtag;
Open Source: GDBs, OpenOCD, UrJTAG
> #cat /practice/target

Anything left?

- JTAG PINs
- CPU Info
- Adapter
- Adapter cfg
- Software
> #cat /practice/jtag/reset

- TRST pin has to be set, to avoid some unexpected situation
> #cat /practice/target

Seems Done!??
<table>
<thead>
<tr>
<th>Actions</th>
<th>What for?</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG PINs</td>
<td>Get Pins for connection</td>
</tr>
<tr>
<td></td>
<td>I know how to connect</td>
</tr>
<tr>
<td>CPU Info</td>
<td>Know CPU Core to inject code</td>
</tr>
<tr>
<td></td>
<td>I know your ISA</td>
</tr>
<tr>
<td>Adapter</td>
<td>Signal Conversion</td>
</tr>
<tr>
<td></td>
<td>You Know my language</td>
</tr>
<tr>
<td>Adapter cfg</td>
<td>Let Software recognize Adapter</td>
</tr>
<tr>
<td></td>
<td>Power on my translator</td>
</tr>
<tr>
<td>Software</td>
<td>Pack JTAG sequence to readable cmds</td>
</tr>
<tr>
<td></td>
<td>Talk in English</td>
</tr>
<tr>
<td>Reset</td>
<td>Some tricky pins</td>
</tr>
<tr>
<td></td>
<td>Reset Connection</td>
</tr>
</tbody>
</table>

Hello World!
>#!/demo

- Are you looking for something?
JTAG theory is a little bit complicated;

JTAG is architecture-free by design;

But architecture-dependent by use;

Fortunately most work have been done by open source software: OpenOCD;

Set up JTAG connection to an unknown device ab initio is not easy;

Connect to 127.0.0.1:3333 by GDB to debug code;

Anyway, JTAG is a very powerful tool for hardware hacking.
>#!/acknowledge

- TombKeeper;
- Dragos and CanSecWest
- My colleague: salt
• Any Questions?