OpenOCD - Beyond Simple Software Debugging

Oleksij Rempel – o.rempel@pengutronix.de
Why I use OpenOCD?

- Reverse engineering and for fun
  - This is the main motivation behind this talk
- Debugging
- Testing
My reverse engineering rules

- Investigate public materials
  - Standards
  - Documentation
  - Patterns
- Try to apply gained knowledge to similarly purposed systems
  - New technology is expensive and vendors are trying to reuse as much as possible
- Assumptions are OK!
The target group

- Everyone who used OpenOCD for software debugging or reverse engineering
- Everyone who has time to use OpenOCD on unsupported or untested HW
- Everyone who is interested in exploring HW from JTAG perspective
History of JTAG

- 1986 - Philips forms Joint European Test Action Group
- 1990 - IEEE Standard 1149.11990 published
boundary scan
Boundary Scan

BS!!! :D

Test-Data In  TDI  Bypass  TDO  Test-Data Out
Identification Register
Instruction Register
TMS  TCK  TAP Controller
Test-Mode Select
Test Clock

Fig. 1. IEEE Standard 1149.1-1990 architecture.
History

- Now 2018, 28 years later...
- We are still using this technology but have no idea how to use it for the original purpose – boundary scan!
- Let's go back to the roots!!! ;)

What is BSDL

- Boundary Scan Description Language
- 1149.1b-1994 “Supplement to IEEE Std 1149.1-1990, IEEE standard test access port and boundary-scan architecture”
- 1149.1-2001 “IEEE standard test access port and boundary-scan architecture”
entity STM32F302_F303_B_C_LQFP100 is
-- This section identifies the default device package selected.
generic (PHYSICAL_PIN_MAP: string:= "LQFP100_PACKAGE");
-- This section declares all the ports in the design.
port ( BOOT0 : in bit;
      JTDI : in bit;
      JTMS : in bit;
      JTCK : in bit;
      JTRST : in bit;
      JTD0 : out bit;
      NRST : in bit;  -- modification to add COMPLIANCE PATTERNS
attribute INSTRUCTION_LENGTH of STM32F302_F303_B_C_LQFP100: entity is 5;

-- Specifies the boundary-scan instructions implemented in the design and their opcodes.

attribute INSTRUCTION_OPCODE of STM32F302_F303_B_C_LQFP100: entity is
   "BYPASS (11111)," &
   "EXTEST (00000)," &
   "SAMPLE (00010)," &
   "PRELOAD (00010)," &
   "IDCODE (00001);"

-- Specifies the bit pattern that is loaded into the Instruction register when the TAP controller
-- passes through the Capture-IR state. The standard mandates that the two LSBs must be "01". The
-- remaining bits are design specific.

attribute INSTRUCTION_CAPTURE of STM32F302_F303_B_C_LQFP100: entity is "XXX01";

-- Specifies the bit pattern that is loaded into the DEVICE_ID register during the IDCODE
-- Instruction when the TAP controller passes through the Capture-DR state.

attribute IDCODE_REGISTER of STM32F302_F303_B_C_LQFP100: entity is
   "XXXX" & -- 4-bit version number
   "0110010001000010" & -- 16-bit part number
   "000001000000" & -- 11-bit identity of the manufacturer
   "1"; -- Required by IEEE Std 1149.1

-- This section specifies the test data register placed between TDI and TDO for each implemented
-- instruction.

attribute REGISTER_ACCESS of STM32F302_F303_B_C_LQFP100: entity is
   "BYPASS (BYPASS)," &
   "BOUNDARY (EXTEST, SAMPLE, PRELOAD)," &
   "DEVICE_ID (IDCODE)";
attribute BOUNDARY_LENGTH of STM32F302_F303_B_C_LQFP100: entity is 250;

-- The following list specifies the characteristics of each cell in the boundary scan register from
-- TDI to TDO. The following is a description of the label fields:
-- num: Is the cell number.
-- cell: Is the cell type as defined by the standard.
-- port: Is the design port name. Control cells do not have a port name.
-- function: Is the function of the cell as defined by the standard. Is one of input, output2,
-- output3, bidir, control or controlr.
-- safe: Specifies the value that the BSR cell should be loaded with for safe operation
-- when the software might otherwise choose a random value.
-- ccell: The control cell number. Specifies the control cell that drives the output enable
-- for this port.
-- disval: Specifies the value that is loaded into the control cell to disable the output
-- enable for the corresponding port.
-- rslt: Resulting state. Shows the state of the driver when it is disabled.

attribute BOUNDARY REGISTER of STM32F302_F303_B_C_LQFP100: entity is

-- num cell port function safe [ccell disval rslt]

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>249</td>
<td>(BC_1, *),</td>
<td>CONTROL,</td>
<td>1),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>(BC_1, PE2,</td>
<td>OUTPUT3,</td>
<td>X,</td>
<td>249,</td>
<td>1,</td>
<td>Z),</td>
<td></td>
</tr>
<tr>
<td>247</td>
<td>(BC_4, PE2,</td>
<td>INPUT,</td>
<td>X),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>(BC_1, *),</td>
<td>CONTROL,</td>
<td>1),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>(BC_1, PE3,</td>
<td>OUTPUT3,</td>
<td>X,</td>
<td>246,</td>
<td>1,</td>
<td>Z),</td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>(BC_4, PE3,</td>
<td>INPUT,</td>
<td>X),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>(BC_1, *),</td>
<td>CONTROL,</td>
<td>1),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>(BC_1, PE4,</td>
<td>OUTPUT3,</td>
<td>X,</td>
<td>243,</td>
<td>1,</td>
<td>Z),</td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>(BC_4, PE4,</td>
<td>INPUT,</td>
<td>X),</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The road map

- How to get JTAG access on modern SoCs.
- Exploring different TAPs and seeking BS register
- Reading BSDL files.
- Unfriendly vendor and no BSDL file, trying to reverse engineer it.
- Practical example.
- Combine CPU and BS tests? Is it possible?
Exploring JTAG port

- In the perfect world, we would have a dedicated JTAG connector in accordance with some valid specification, working all the time from power on till power off.

- The reality is different:
  - In many cases JTAG pins are enabled by the SoC ROM, with some delay after power on (or power cycle)
  - The pins have JTAG functionality only limited time after some event
  - Many TAPs and DAPs with some differences from default or well-known specifications

Welcome to the JTAG zoo!
Getting JTAG access

- There are two states:
  - It just works!
  - Go with me, I’ll show you how some vendors do it! :D
Exploring JTAG port (time frames)
Exploring JTAG port (Allwinner JTAG/SD)

- Most of the Allwinner SoCs have JTAG multiplexed with SD card signals. It is not a secret, but not well-documented.

- This port can be used only within a short time frame:
  - Some $X$ milliseconds after power on JTAG gets enabled
  - $X+Y$ms after power on this port is switched from JTAG to SD, so we have just a small window to access JTAG.
Exploring JTAG port (Allwinner JTAG/SD)

- Remote controllable bench power supply and logic analyser are your friends
- Use `adapter_nsrst_delay`
- Increase `adapter_khz` speed to fit to narrow time frame
- Add some pull-up resistor to the TDI line and measure it
Exploring JTAG port (Allwinner JTAG/SD)

- 1. no pull-ups, 2. pull-ups on 1,2,3,4
Exploring JTAG port (Allwinner JTAG/SD)
Exploring JTAG port (Open Sesame)
Exploring JTAG port (Open Sesame)

- Nicely documented JTAG/ICSP interfaces made by Microchip for PIC32xx series
Exploring the internals
Exploring the internals

- Let’s assume we got access to the SoC, what can we explore?
- TAP – test access port
- Typical instructions provided by a TAP:
  - IDCODE
  - Boundary scan
  - Bypass
Exploring the internals

- The times they are a-changin', after 28 years internals are a bit more complicated
- Let’s take as example STM32 and do following steps:
  - Find the right TAP
  - Find the right Instruction
  - Find the right Bits
Find the right TAP

Figure 405. JTAG TAP connections

STM32 MCU

NJTRST

JTMS

SW-DP Selected

TMS nTRST

Boundary scan TAP

TDI TDO

IR is 5-bit wide

TMS nTRST

Cortex-M4 TAP

TDI TDO

IR is 4-bit wide

JTDO

JTDI
attribute INSTRUCTION_LENGTH of STM32F302_F303_B_C_LQFP100: entity is 5;

-- Specifies the boundary-scan instructions implemented in the design and their opcodes.

attribute INSTRUCTION_OPCODE of STM32F302_F303_B_C_LQFP100: entity is
"BYPASS (11111)," &
"EXTTEST (00000)," &
"SAMPLE (00010)," &
"PRELOAD (00010)," &
"ICODE (00001);"

-- Specifies the bit pattern that is loaded into the instruction register when the TAP controller
-- passes through the Capture-IR state. The standard mandates that the two LSBs must be "01". The
-- remaining bits are design specific.

attribute INSTRUCTION_CAPTURE of STM32F302_F303_B_C_LQFP100: entity is "XXX01";

-- Specifies the bit pattern that is loaded into the DEVICE_ID register during the IDCODE
-- instruction when the TAP controller passes through the Capture-DR state.

attribute IDCODE_REGISTER of STM32F302_F303_B_C_LQFP100: entity is
"XXXX" & -- 4-bit version number
"8110010000100010" & -- 16-bit part number
"0000010000000" & -- 11-bit identity of the manufacturer
"1"; -- Required by IEEE Std 1149.1

-- This section specifies the test data register placed between TDI and TDO for each implemented
-- instruction.

attribute REGISTER_ACCESS of STM32F302_F303_B_C_LQFP100: entity is
"BYPASS (BYPASS)," &
"BOUNDARY (EXTTEST, SAMPLE, PRELOAD)," &
"DEVICE_ID (ICODE)";
Find the right Bits

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
<th>Value4</th>
<th>Value5</th>
</tr>
</thead>
<tbody>
<tr>
<td>154</td>
<td>(BC_4, PE7)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>(BC_1, PE8)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>153</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>151</td>
<td>(BC_4, PE8)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>(BC_1, PE9)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>150</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>148</td>
<td>(BC_4, PE9)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>(BC_1, PE10)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>147</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>145</td>
<td>(BC_4, PE10)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>(BC_1, PE11)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>144</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>142</td>
<td>(BC_4, PE11)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>(BC_1, PE12)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>141</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>139</td>
<td>(BC_4, PE12)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>(BC_1, PE13)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>138</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>136</td>
<td>(BC_4, PE13)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>(BC_1, PE14)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>135</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>133</td>
<td>(BC_4, PE14)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>(BC_1, PE15)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>132</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>130</td>
<td>(BC_4, PE15)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>(BC_1, *)</td>
<td>CONTROL</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>(BC_1, PB10)</td>
<td>OUTPUT3</td>
<td>X</td>
<td>129</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>127</td>
<td>(BC_4, PB10)</td>
<td>INPUT</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exploring JTAG port (BS on STM32)

- Video demonstration of using JTAG boundary scan on STM32F3
- The bsr.tcl script by Paul Fertser
  - Init BS TAP
  - Scan for floating PADs
  - Scan for changed PADs after adding pull-up/down.
  - Test related control bits for given PAD. For example:
    - Bit 142 - read input state
    - Bit 143 - set output state
    - Bit 144 - switch between input and output mode.
Exploring JTAG port (BS on STM32)
Crazy idea:
What if we configure a pin from GPIO peripheral and test it with BS?
Is it possible with JTAG BS to read a PAD which was configured by GPIO peripheral? Yes! At least on some SoCs

Steps made in following video:
- Start JTAG and halt CPU.
- Enable CLK for GPIO controller.
- Measure PAD with GPIO, then switch the PAD to output mode
- Switch to the JTAG BS mode and read out PAD state
Exploring JTAG port (GPIO + BS on STM32)
Exploring JTAG port (GPIO + BS on PIC32)

- Same test on PIC32
- Suddenly it needed more work than expected
- PIC32xx has multiple taps but not connected in chain so BYPASS instruction is not applicable. We have here two vendor instructions: switch to MTAP and switch to ETAP
- The BS is available on MTAP
Exploring JTAG port (GPIO + BS on PIC32)
Exploring JTAG port (GPIO + BS on iMX6)

- Same test on iMX6
- BS is implemented on SJC TAP
- This was fast, the BS instruction is directly connected to reset controller. Executing BS will automatically put CPU in reset state
- BS should still be possible with correctly configured bootstrap pins (see the SoC manual)
Exploring JTAG on iMX6

- Implemented and tested TAPs for iMX6:
  - MPCore, Cortex-A9
- Not implemented or not upstreamed parts:
  - Everything else :)
Thank you!

Questions?