OpenOCD - Beyond Simple Software Debugging

Oleksij Rempel – o.rempel@pengutronix.de

https://www.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
- JOURNAL OF ELECTRONIC TESTING: Theory and Applications, 2, 1125 (1991)

boundary scan

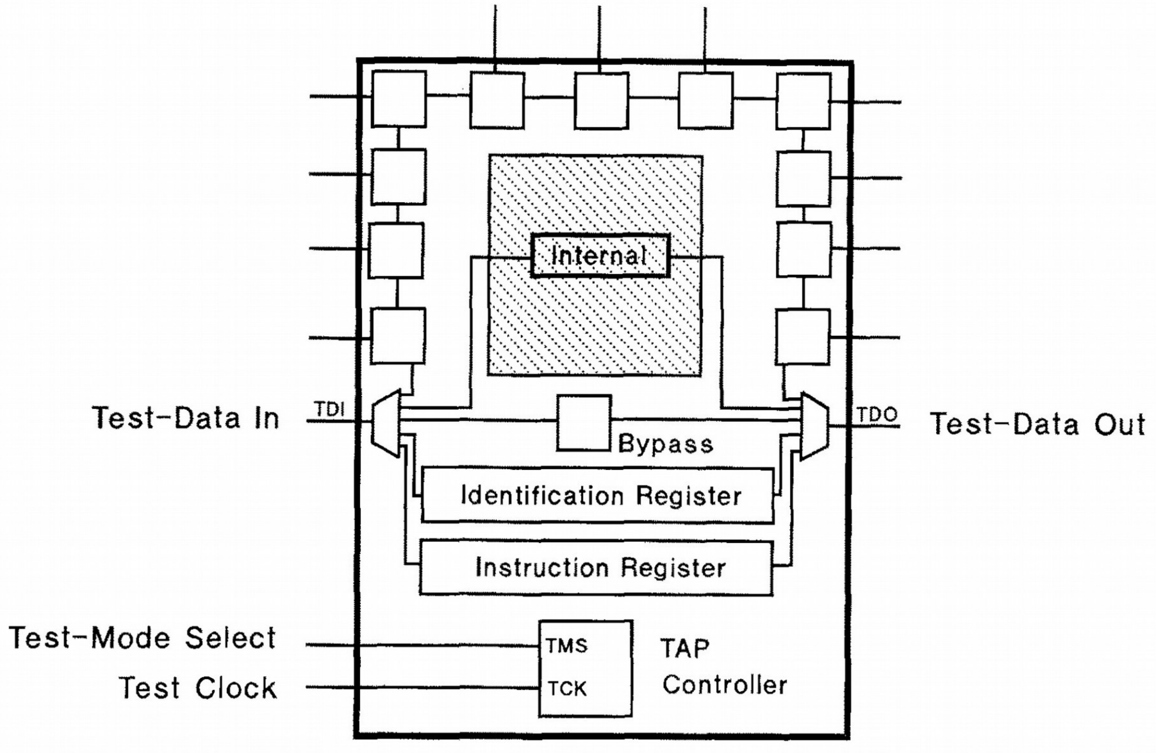


Fig. 1. IEEE Standard 1149.1-1990 architecture.

Boundary Scan

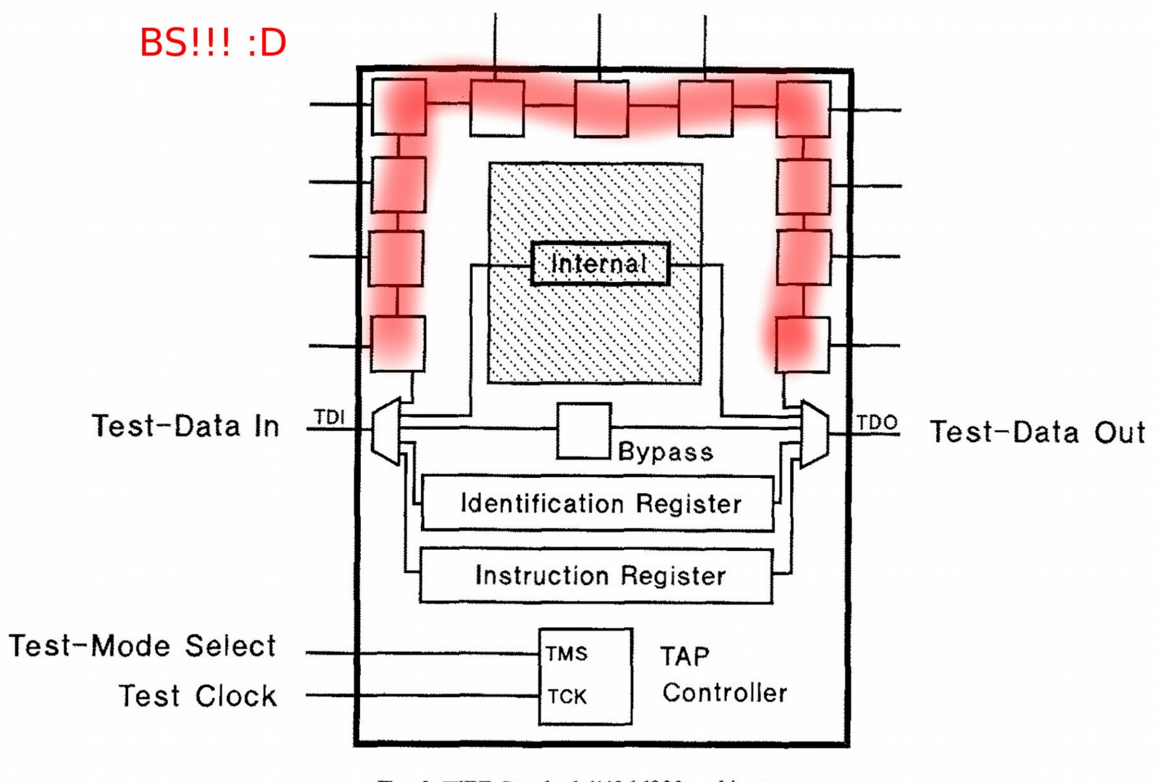


Fig. 1. IEEE Standard 1149.1-1990 architecture.

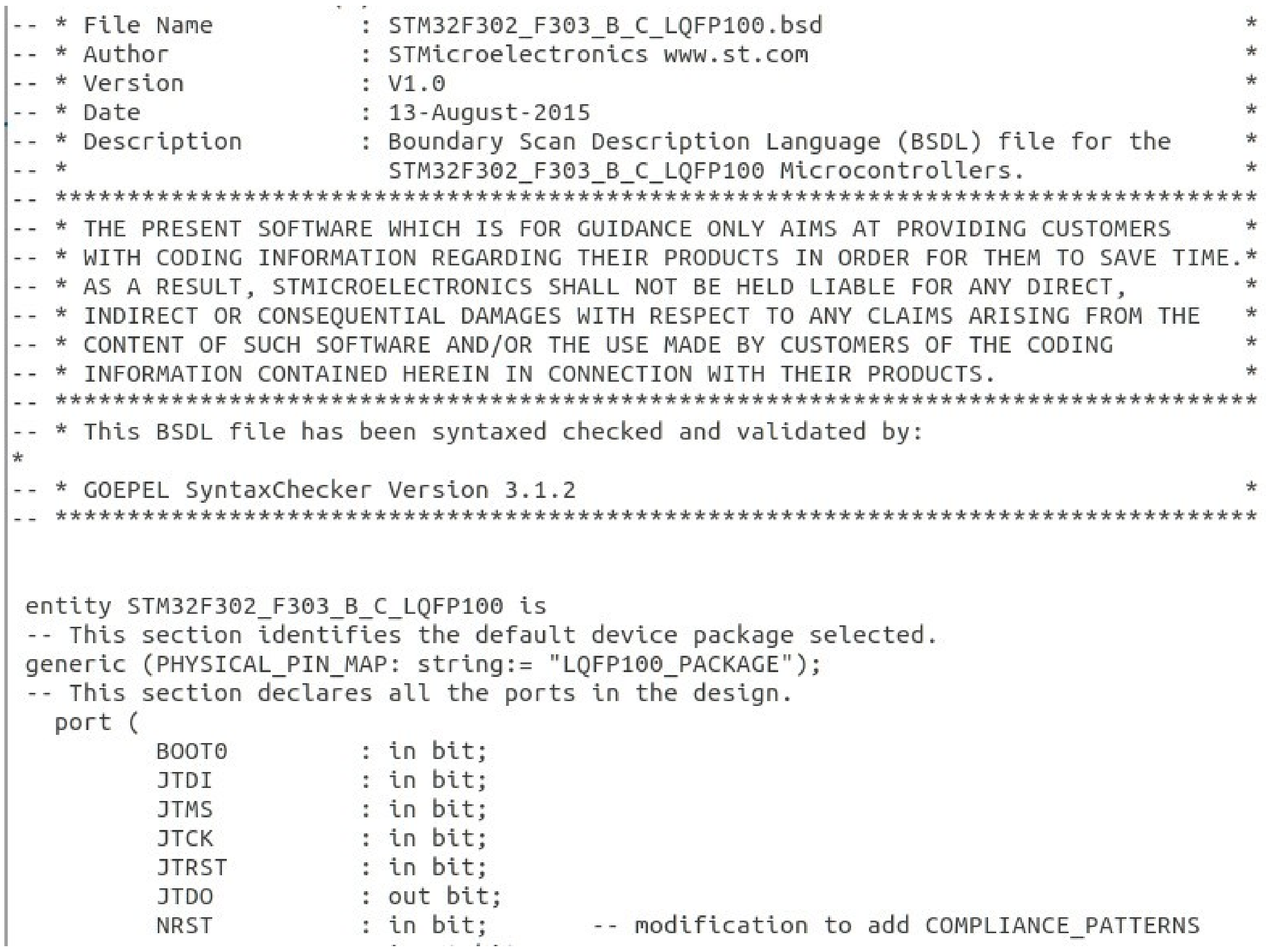
History

- 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

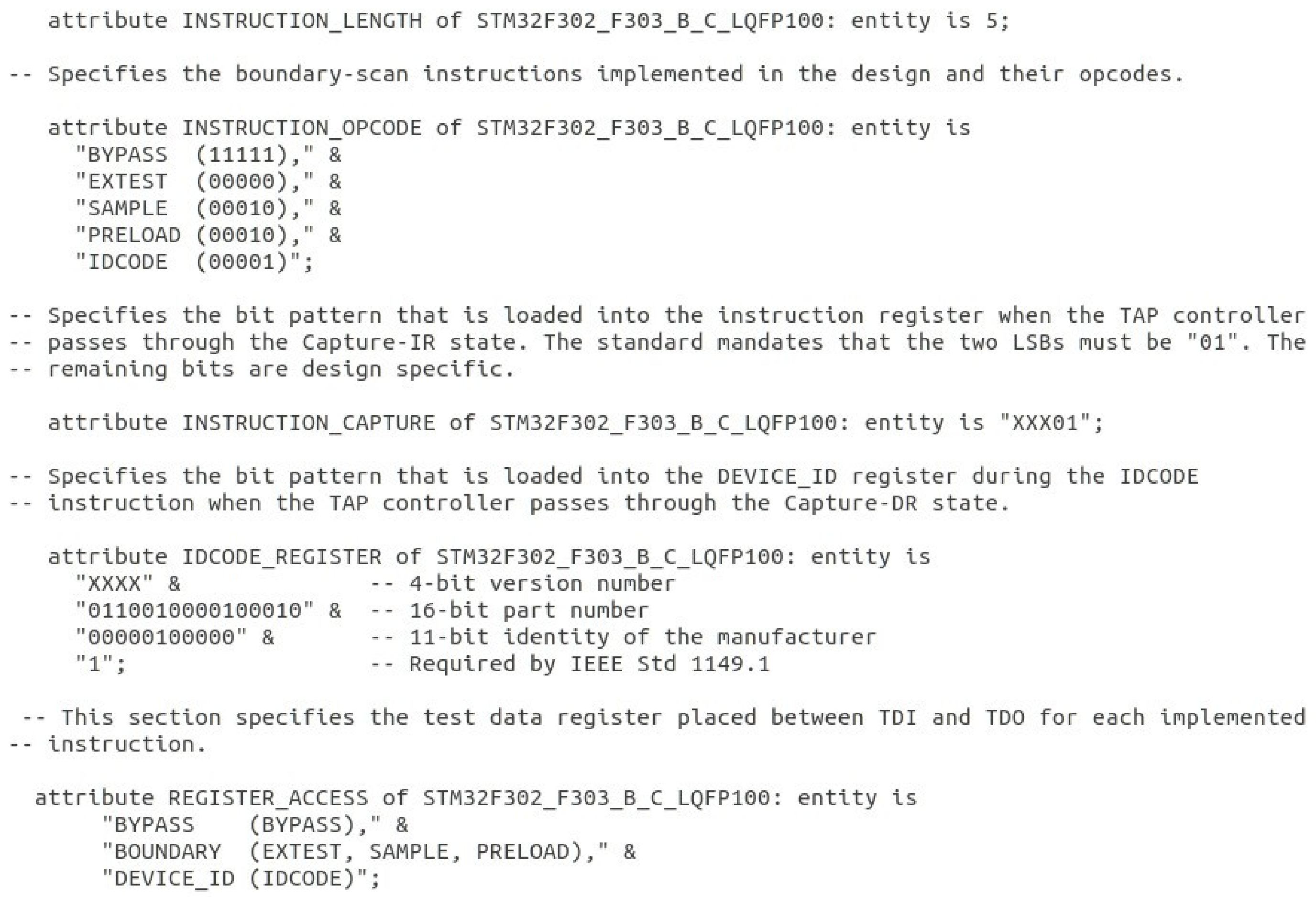
- Boundary Scan Description Language
- 1149.1b-1994 “Supplement to IEEE Std 1149.1-1990, IEEE standard test access port and boundary-scan architecture”
- 1149.1-2001 “IEEE standard test access port and boundary-scan architecture”

BSDL Example 1



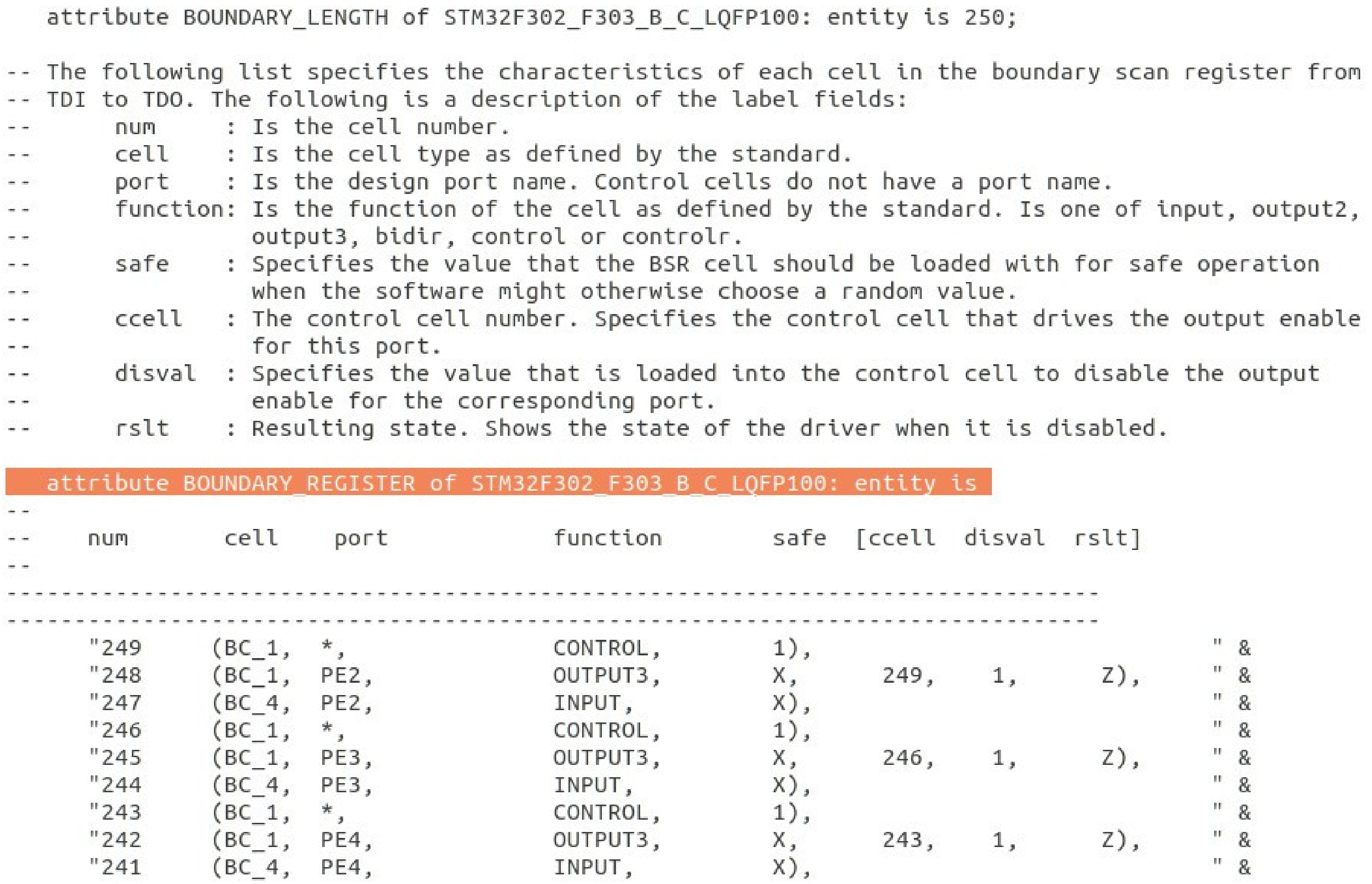
```
-- * File Name           : STM32F302_F303_B_C_LQFP100.bsd                        *
-- * Author              : STMicroelectronics www.st.com                          *
-- * Version             : V1.0                                                   *
-- * Date                : 13-August-2015                                         *
-- * Description         : Boundary Scan Description Language (BSDL) file for the *
-- *                       STM32F302_F303_B_C_LQFP100 Microcontrollers.           *
-- ********************************************************************************
-- * THE PRESENT SOFTWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS   *
-- * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.*
-- * AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT,      *
-- * INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE *
-- * CONTENT OF SUCH SOFTWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING       *
-- * INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.               *
-- ********************************************************************************
-- * This BSDL file has been syntaxed checked and validated by:
*
-- * GOEPEL SyntaxChecker Version 3.1.2                                            *
-- ********************************************************************************



 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;
          JTDO           : out bit;
          NRST           : in bit;       -- modification to add COMPLIANCE_PATTERNS
```

BSDL Example 2



```
   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
     "0110010000100010" &   -- 16-bit part number
     "00000100000" &        -- 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)";
```

BSDL Example 3



```
   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]
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
      "249     (BC_1,  *,              CONTROL,        1),                              " &
      "248     (BC_1,  PE2,            OUTPUT3,        X,      249,    1,      Z),      " &
      "247     (BC_4,  PE2,            INPUT,          X),                              " &
      "246     (BC_1,  *,              CONTROL,        1),                              " &
      "245     (BC_1,  PE3,            OUTPUT3,        X,      246,    1,      Z),      " &
      "244     (BC_4,  PE3,            INPUT,          X),                              " &
      "243     (BC_1,  *,              CONTROL,        1),                              " &
      "242     (BC_1,  PE4,            OUTPUT3,        X,      243,    1,      Z),      " &
      "241     (BC_4,  PE4,            INPUT,          X),                              " &
```

The road map

- How to get JTAG access on modern SoCs.
- Exploring diferent 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

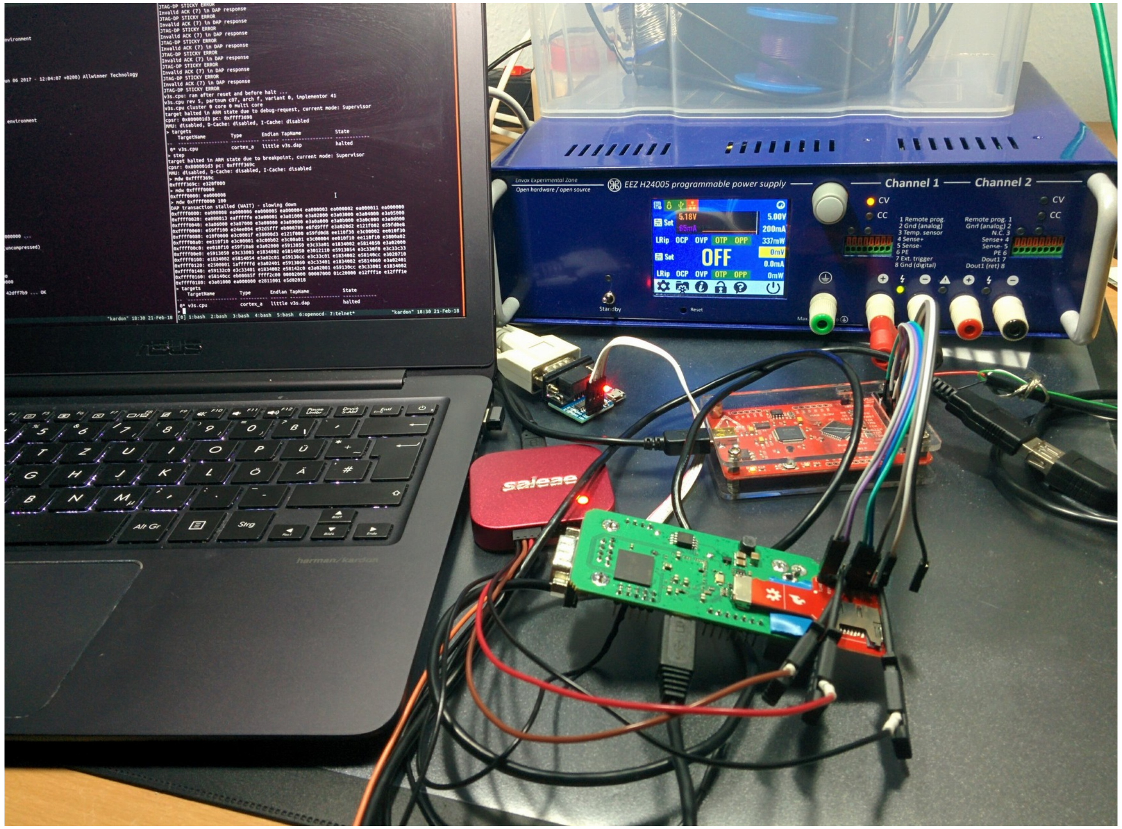
Exploring JTAG port (time frames)

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 millisecs after power on JTAG gets enabled
  - X+Yms 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)

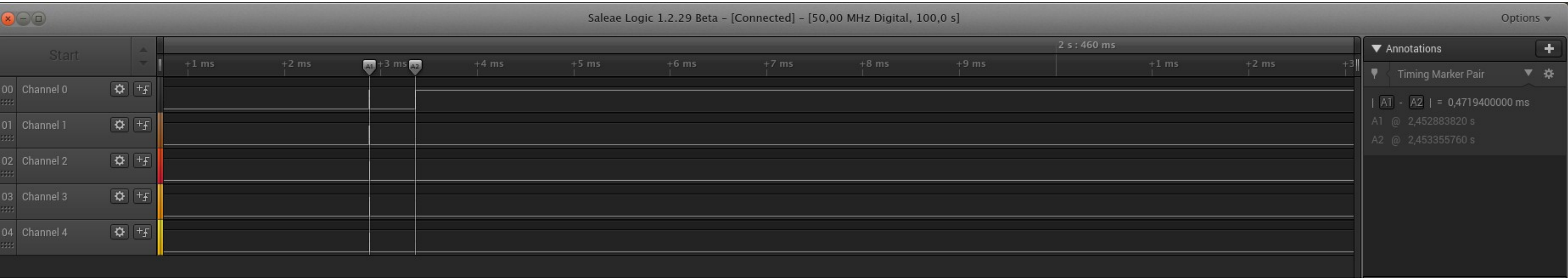
- Remote controllable bench power supply and logic analyser are your friends
- Use adapter_nsrst_delay
- Increase adapter_khz speed to fit to narrow time frame
- Add some pull-up resistor to the TDI line and measure it



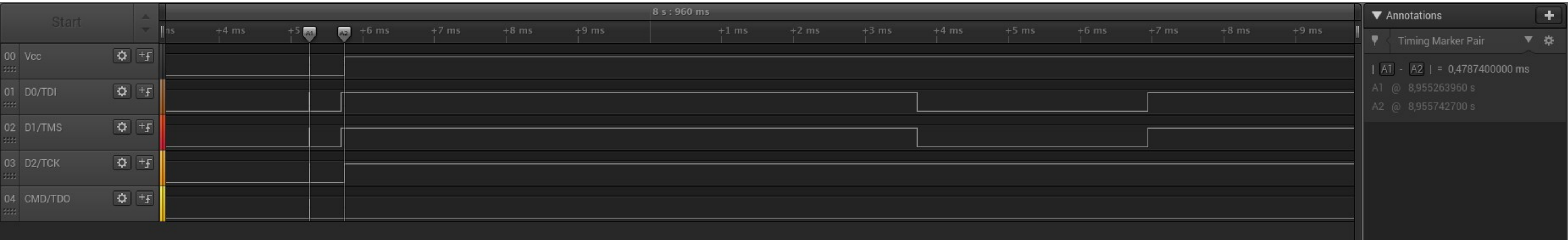
```
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
Invalid ACK (7) in DAP response
JTAG-DP STICKY ERROR
v3s.cpu: ran after reset and before halt ...
v3s.cpu rev 5, partnum c07, arch f, variant 0, implementor 41
v3s.cpu cluster 0 core 0 multi core
target halted in ARM state due to debug-request, current mode: Supervisor
cpsr: 0x800001d3 pc: 0xffff3698
MMU: disabled, D-Cache: disabled, I-Cache: disabled
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* v3s.cpu            cortex_a   little v3s.dap            halted
> step
target halted in ARM state due to breakpoint, current mode: Supervisor
cpsr: 0x800001d3 pc: 0xffff369c
MMU: disabled, D-Cache: disabled, I-Cache: disabled
> mdw 0xffff369c
0xffff369c: e320f000
> mdw 0xffff0000
0xffff0000: ea000008
> mdw 0xffff0000 100
DAP transaction stalled (WAIT) - slowing down
0xffff0000: ea000008 ea000006 ea000005 ea000004 ea000003 ea000002 ea000011 ea000000
0xffff0020: ea000013 eafffffe e3a00001 e3a01000 e3a02000 e3a03000 e3a04000 e3a05000
0xffff0040: e3a06000 e3a07000 e3a08000 e3a09000 e3a0a000 e3a0b000 e3a0c000 e3a0d000
0xffff0060: e59ff100 e24ee004 e92d5fff eb000789 e8fd9fff e3a020d2 e121f002 e59fd0e8
0xffff0080: e10f0000 e3c0001f e38000d3 e121f000 e59fd0d8 ee110f30 e3c00002 ee010f30
0xffff00a0: ee110f10 e3c00001 e3c00b02 e3c00a01 e3c00004 ee010f10 ee110f10 e3800a02
0xffff00c0: ee010f10 e59f10a8 e3a02000 e5913050 e3c33a01 e1834002 e5814050 e3a02000
0xffff00e0: e5913050 e3c33003 e1834002 e5814050 e3012110 e5913054 e3c330f0 e3c33c33
0xffff0100: e1834002 e5814054 e3a02c01 e59130cc e3c33c01 e1834002 e58140cc e3020710
0xffff0120: e2500001 1afffffd e3a02401 e5913060 e3c33401 e1834002 e5814060 e3a02401
0xffff0140: e59132c0 e3c33401 e1834002 e58142c0 e3a02001 e59130cc e3c33001 e1834002
0xffff0160: e58140cc eb00085f ffff2c00 00002000 00007000 01c20000 e12fff1e e12fff1e
0xffff0180: e3a01000 ea000000 e2811001 e5d02018
> targets
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* v3s.cpu            cortex_a   little v3s.dap            halted
>
[0] 1:bash  2:bash  3:bash  4:bash  5:bash  6:openocd- 7:telnet*      "kardon" 18:30 21-Feb-18
```

Exploring JTAG port (Allwinner JTAG/SD)

- 1. no pull-ups, 2. pull-ups on 1,2,3,4

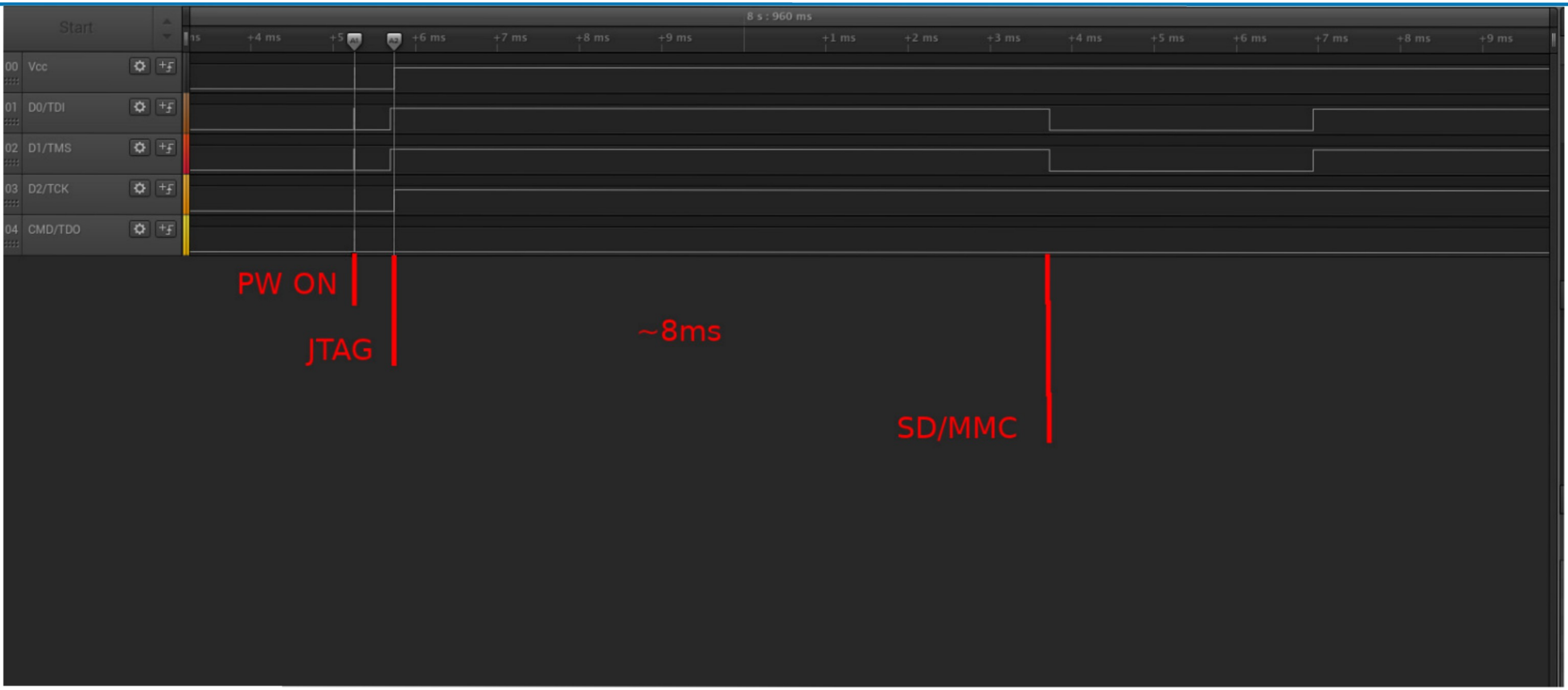


| A1 - A2 | = 0,4719400000 ms
A1 @ 2,452883820 s
A2 @ 2,453355760 s

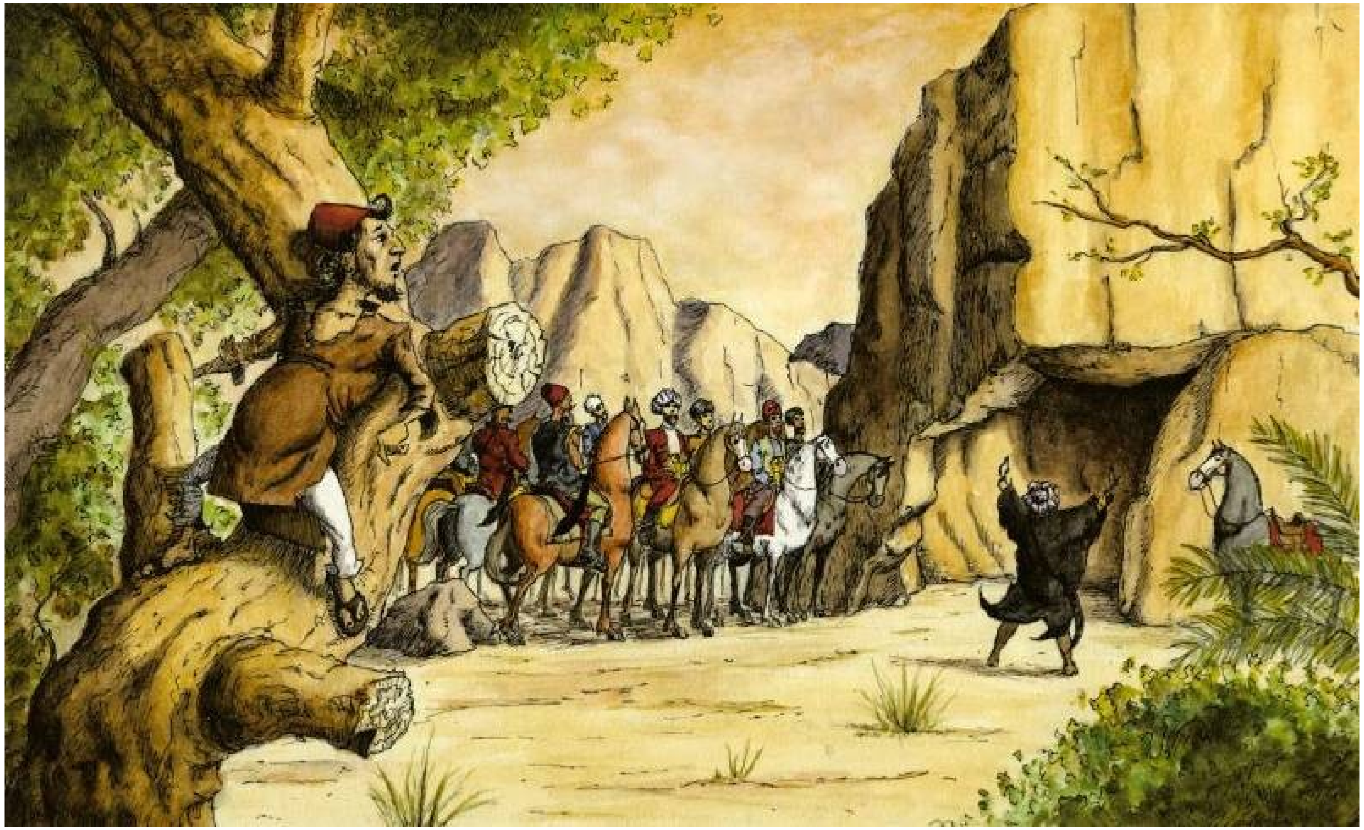


| A1 - A2 | = 0,4787400000 ms
A1 @ 8,955263960 s
A2 @ 8,955742700 s

Exploring JTAG port (Allwinner JTAG/SD)



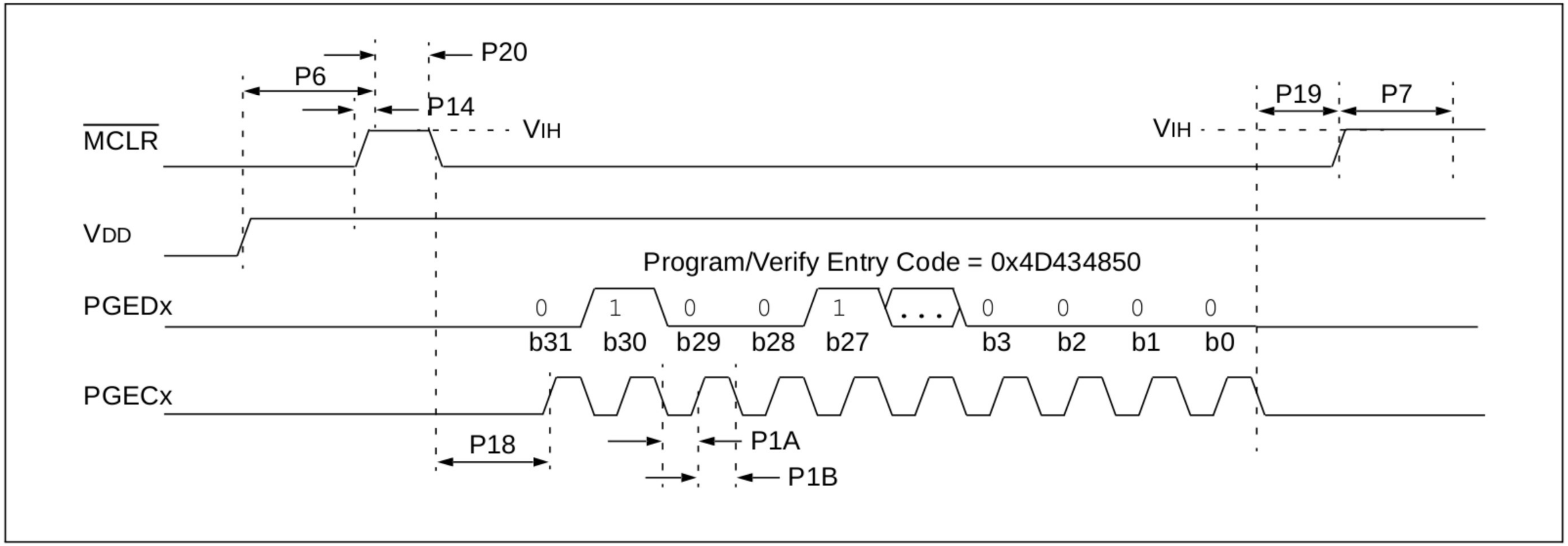
Exploring JTAG port (Open Sesame)



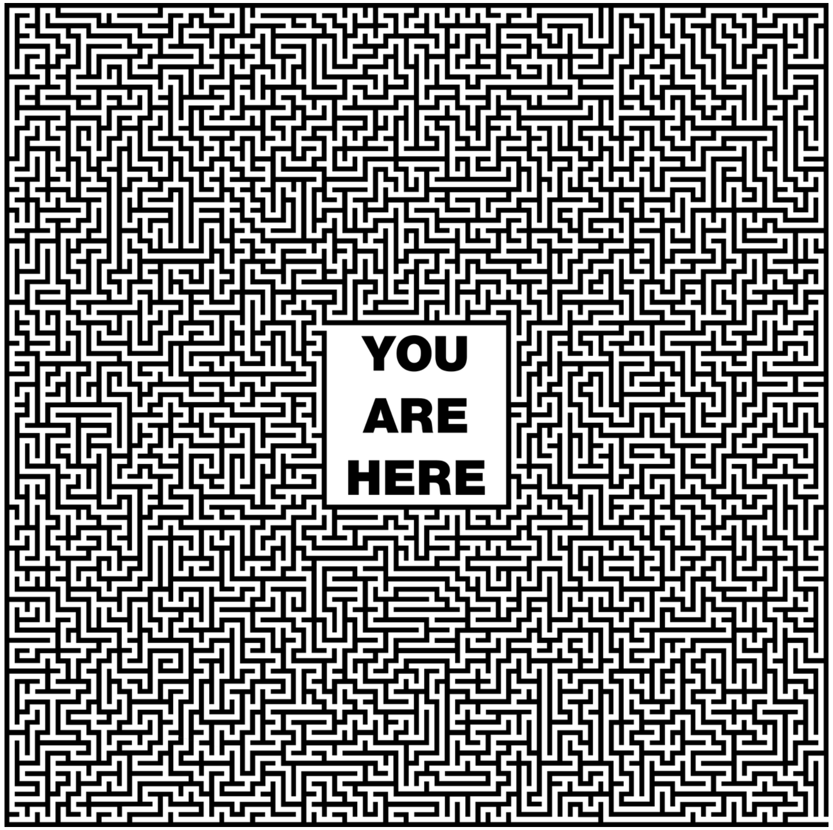
Exploring JTAG port (Open Sesame)

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

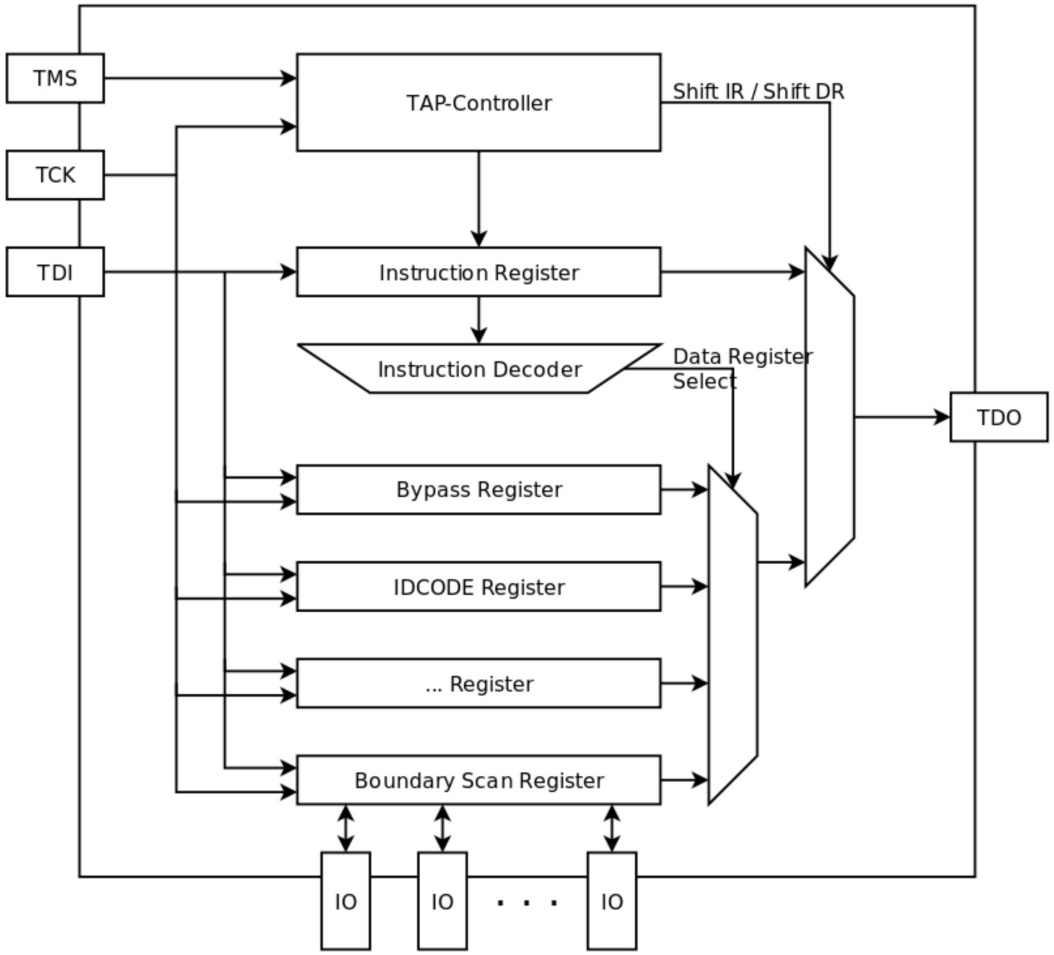
FIGURE 7-1: ENTERING ENHANCED ICSP™ MODE



Exploring the internals



Exploring the internals



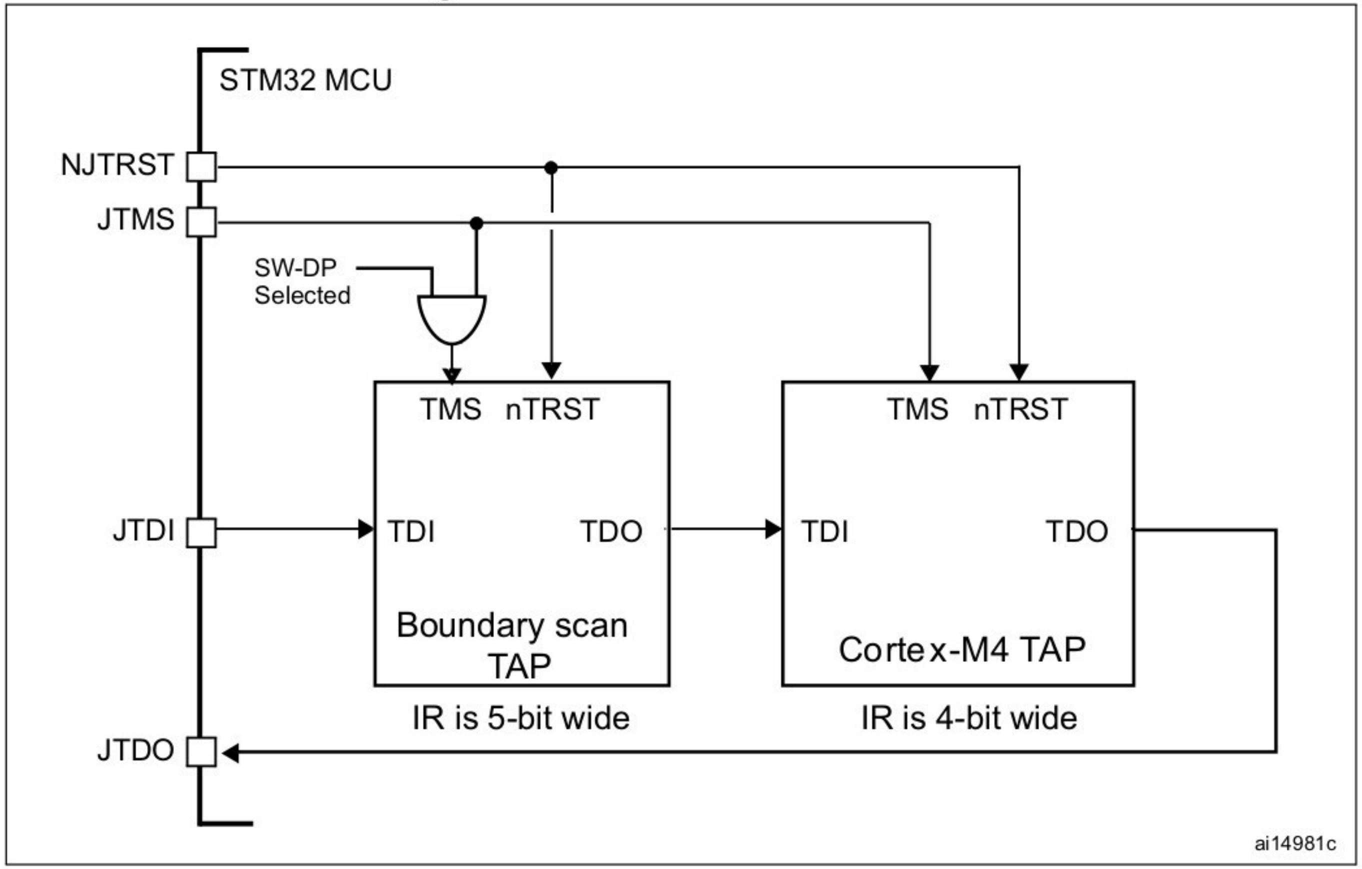
- Let’s assume we got access to the SoC, what can we explore?
- TAP – test access port
- Typical instructions provided by a TAP:
  - IDCODE
  - Boundary scan
  - Bypass

Exploring the internals

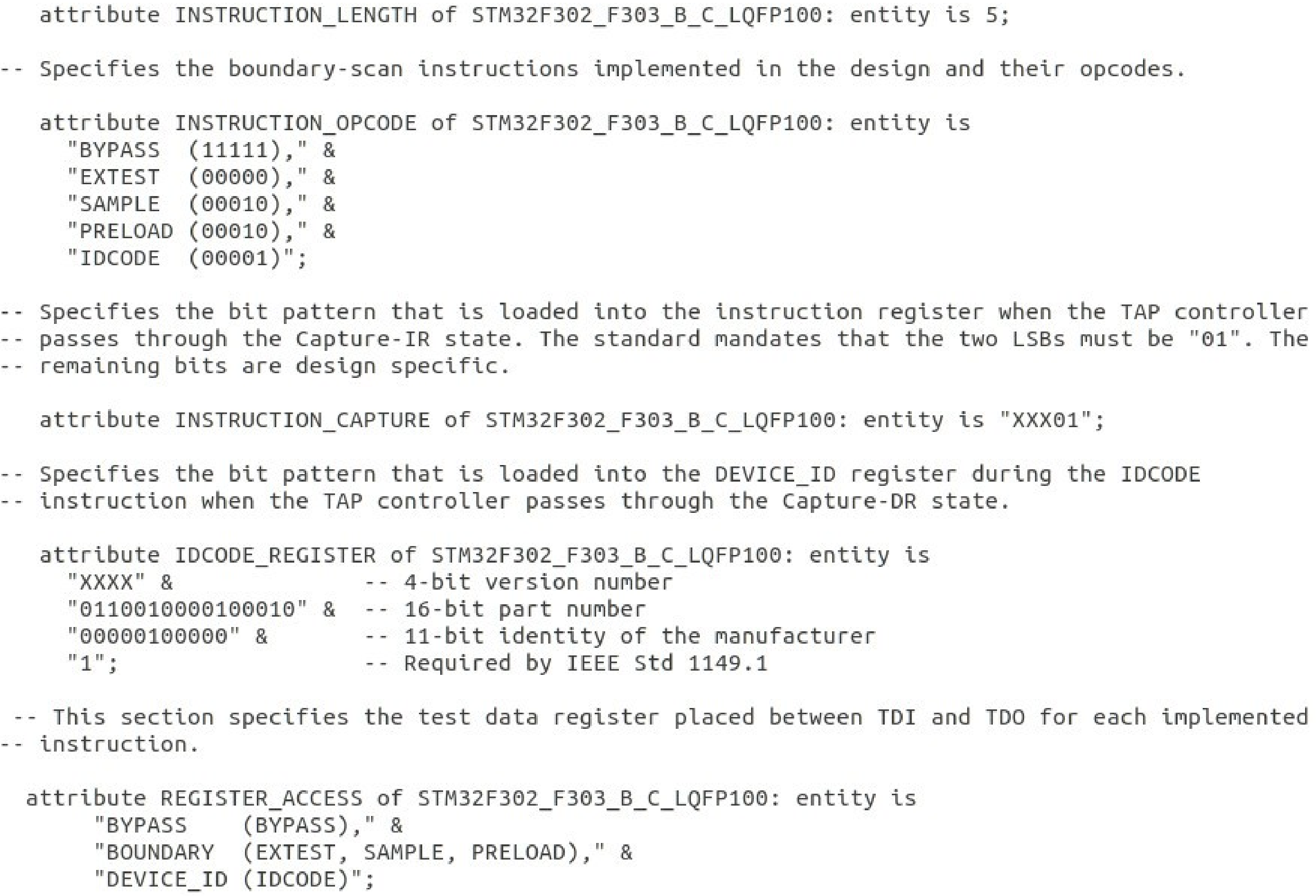
- The times they are a-changin', after 28 years internals are a bit more complicated
- Let’s take as example STM32 and do following steps:
  - Find the right TAP
  - Find the right Instruction
  - Find the right Bits

Find the right TAP

Figure 405. JTAG TAP connections

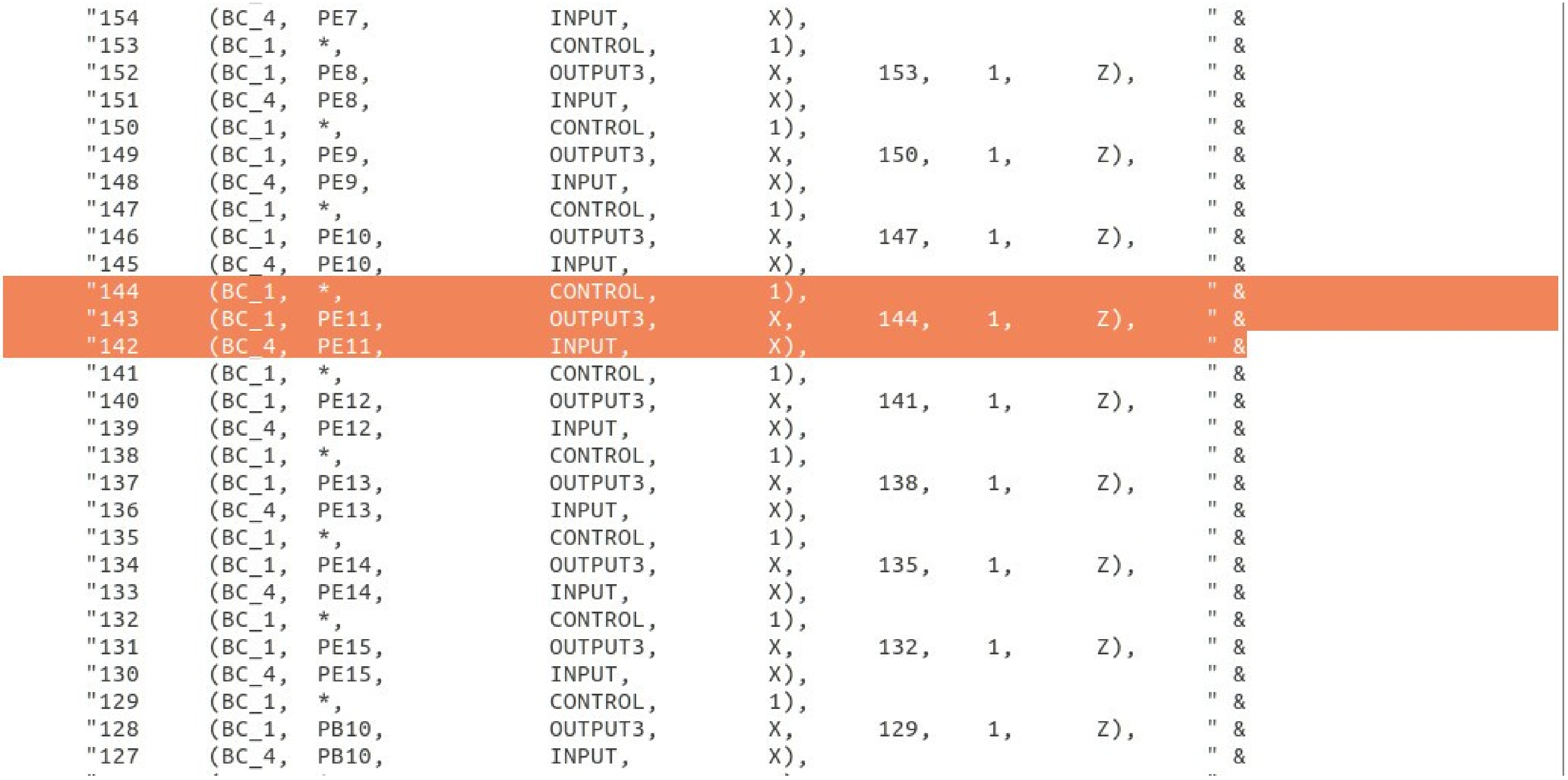


Find the right Instruction



```
   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
     "0110010000100010" &   -- 16-bit part number
     "00000100000" &        -- 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)";
```

Find the right Bits



```
"154      (BC_4,   PE7,              INPUT,         X),                                     " &
"153      (BC_1,   *,                CONTROL,       1),                                     " &
"152      (BC_1,   PE8,              OUTPUT3,       X,       153,     1,       Z),          " &
"151      (BC_4,   PE8,              INPUT,         X),                                     " &
"150      (BC_1,   *,                CONTROL,       1),                                     " &
"149      (BC_1,   PE9,              OUTPUT3,       X,       150,     1,       Z),          " &
"148      (BC_4,   PE9,              INPUT,         X),                                     " &
"147      (BC_1,   *,                CONTROL,       1),                                     " &
"146      (BC_1,   PE10,             OUTPUT3,       X,       147,     1,       Z),          " &
"145      (BC_4,   PE10,             INPUT,         X),                                     " &
"144      (BC_1,   *,                CONTROL,       1),                                     " &
"143      (BC_1,   PE11,             OUTPUT3,       X,       144,     1,       Z),          " &
"142      (BC_4,   PE11,             INPUT,         X),                                     " &
"141      (BC_1,   *,                CONTROL,       1),                                     " &
"140      (BC_1,   PE12,             OUTPUT3,       X,       141,     1,       Z),          " &
"139      (BC_4,   PE12,             INPUT,         X),                                     " &
"138      (BC_1,   *,                CONTROL,       1),                                     " &
"137      (BC_1,   PE13,             OUTPUT3,       X,       138,     1,       Z),          " &
"136      (BC_4,   PE13,             INPUT,         X),                                     " &
"135      (BC_1,   *,                CONTROL,       1),                                     " &
"134      (BC_1,   PE14,             OUTPUT3,       X,       135,     1,       Z),          " &
"133      (BC_4,   PE14,             INPUT,         X),                                     " &
"132      (BC_1,   *,                CONTROL,       1),                                     " &
"131      (BC_1,   PE15,             OUTPUT3,       X,       132,     1,       Z),          " &
"130      (BC_4,   PE15,             INPUT,         X),                                     " &
"129      (BC_1,   *,                CONTROL,       1),                                     " &
"128      (BC_1,   PB10,             OUTPUT3,       X,       129,     1,       Z),          " &
"127      (BC_4,   PB10,             INPUT,         X),                                     " &
```

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?

CHALLENGE ACCEPTED

Exploring JTAG port (GPIO + BS on STM32)

- 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

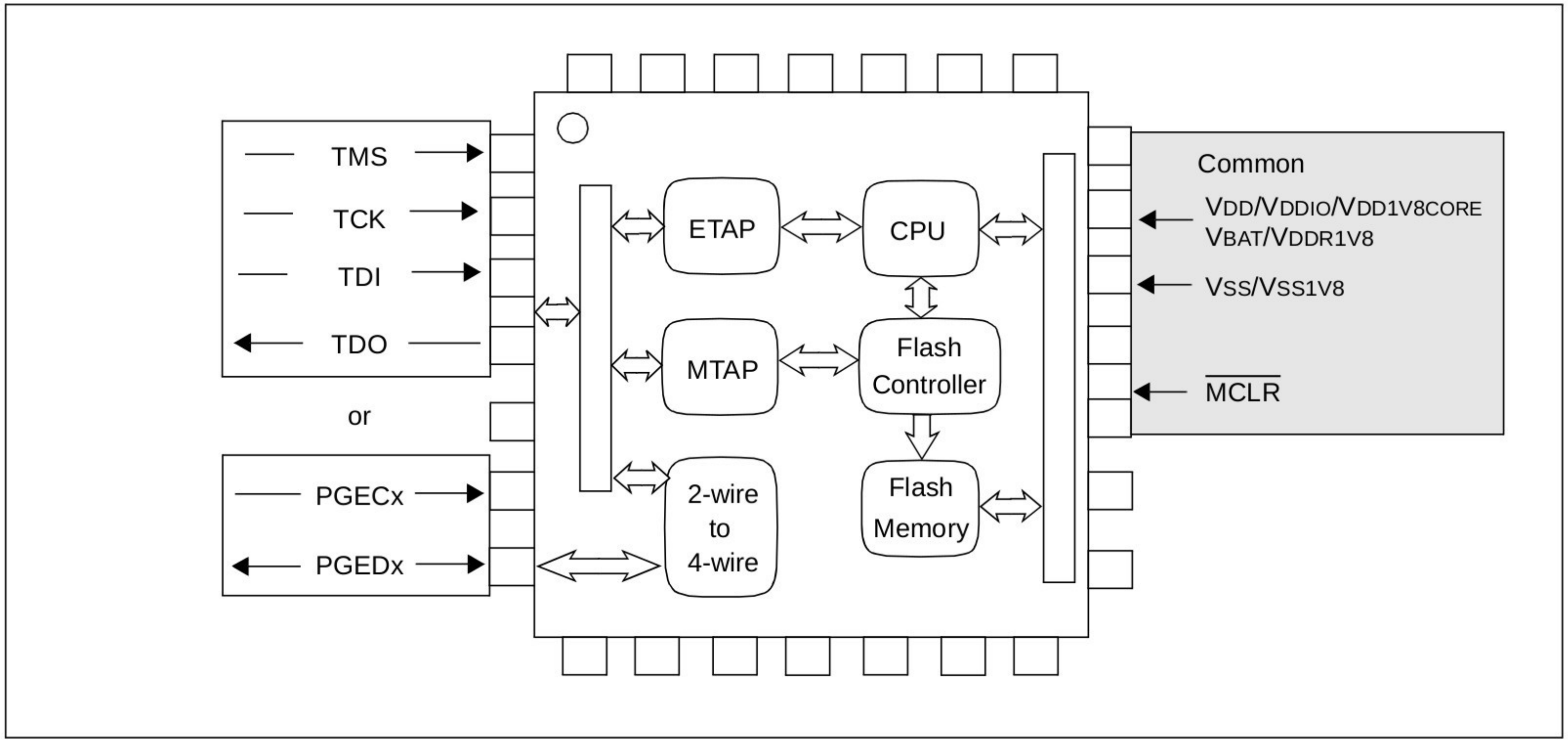
Exploring JTAG port (GPIO + BS on STM32)

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)

FIGURE 5-2: BASIC PIC32 PROGRAMMING INTERFACE BLOCK DIAGRAM



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 :)

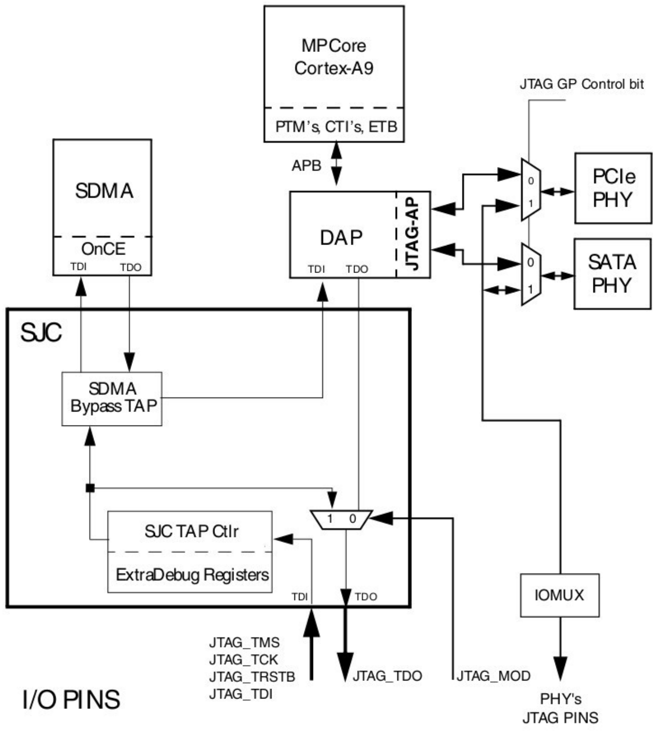


Figure 56-1. System JTAG Controller (SJC) Block Diagram

Thank you!

Questions?

Pengutronix.

https://www.pengutronix.de