JTAG Switcher

Alexander Merkle - Lauterbach GmbH

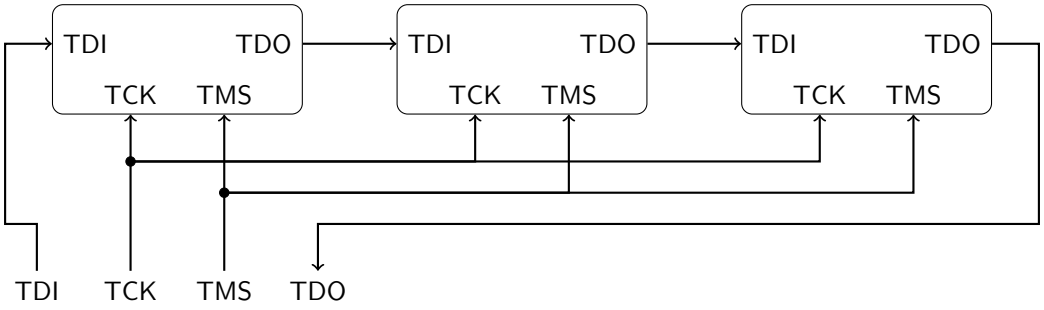
18 October 2019

Overview

Abbreviations

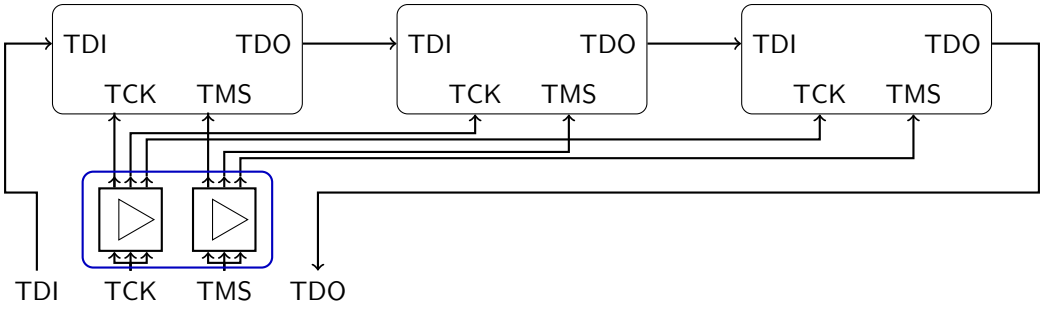
- TAP: Test-Access-Port, JTAG communication unit, consists of Instruction- & multiple Data-Registers selected by a state-machine
- Daisy-Chain: connection of multiple TAPs via a single interface
- JTAG Signals:
  - `TCK`: Test Clock, clocks all D-flip-flops, which are part of the JTAG architecture
  - `TMS`: Test Machine State, controls the JTAG TAP Controller state machine
  - `TDI`: Test Data In, serial data send into the JTAG architecture
  - `TDO`: Test Data Out, serial data received from the JTAG architecture
  - `TRST*`: Test Reset, optional, will *asynchronously* reset the JTAG TAP Controller state machine, putting it into the "Test-Logic-Reset" state

Simple Daisy-Chain



- A Daisy-Chain is formed by connecting TDO to TDI of the next TAP controller
- TCK & TMS are shared signals

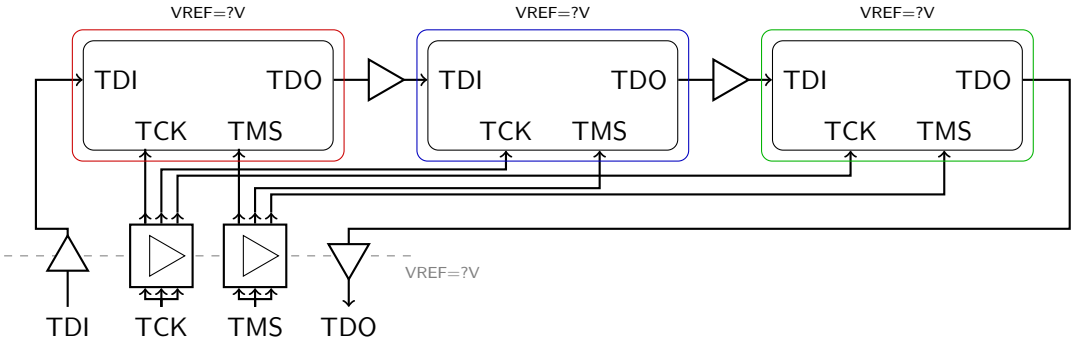
Simple Daisy-Chain



Recommendation

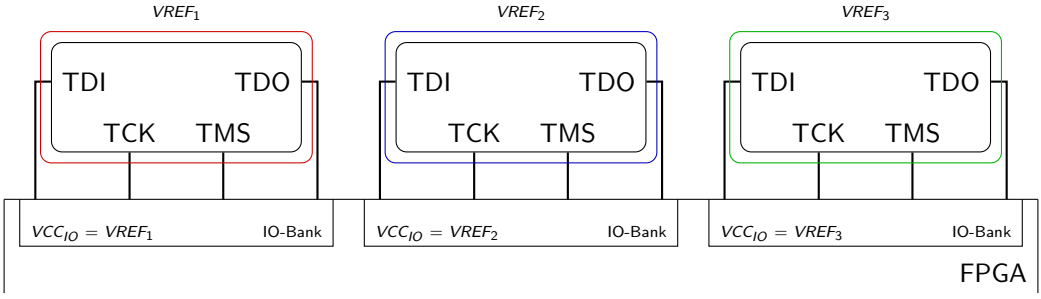
Do not route TCK/TMS as a STAR (stub wires cause reflections).
Use a buffer with one output per TAP for signal integrity.

Real world Daisy-Chain



- In case of different I/O Voltages multiple level shifters are required

FPGA Implementation



- IO-Banks of a (small) FPGA may be also used as level shifters

FPGA Implementation

Listing 1: Daisy-Chain VHDL

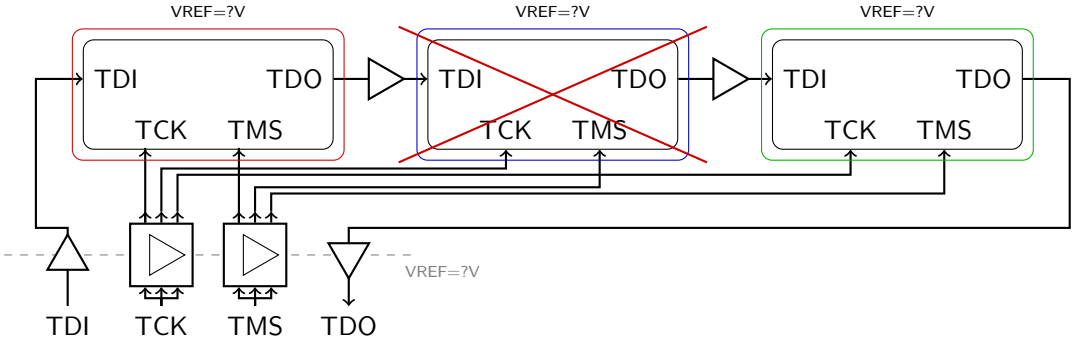
```
oTDI1 <= iTDI;  -- TDI from probe to first TAP
oTDI2 <= iTDO1;
oTDI3 <= iTDO2;
oTDO  <= iTDO3; -- TDO from last TAP to probe
-- wire TCK as a STAR
oTCK1 <= iTCK;
oTCK2 <= iTCK;
oTCK3 <= iTCK;
-- wire TMS as a STAR
oTMS1 <= iTMS;
oTMS2 <= iTMS;
oTMS3 <= iTMS;
```

Daisy-Chaining

- JTAG Daisy-Chaining is simple as long as traces are short and the I/O Levels are matching
- The TCK (=Clock) Signal must have proper integrity
  $\Rightarrow$ use single buffer, use one output per TAP to avoid stubs
- Having multiple I/O Levels requires level shifters
  $\Rightarrow$ influences maximum TCK frequency
- The slowest TAP controller limits the maximum TCK frequency of the complete system
- All TAP controllers must be always functional
  $\Rightarrow$ next slide

Problem

- If one TAP controller is inaccessible the complete Daisy-Chain is broken



Failure Situations

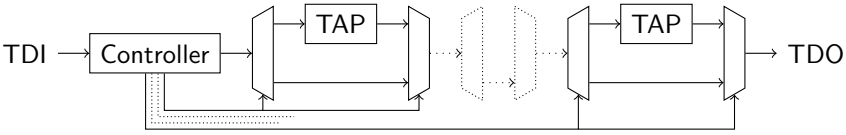
Scenarios that may trigger this problem

- SoC internal or Board level power-saving
- Redundant systems may power down
- CHIP-SECURITY
  - A Secure boot must be never corrupted
  - JTAG is an attack possibility already in early bootstages
  - Some SoCs disable JTAG while booting completely e.g. JTAG pins tristated
  - Some SoCs modify their internal daisy-chain while booting
  - Some SoCs offer possibilities to disable JTAG with FUSEs
  - most times not *properly* documented
- $\Rightarrow$ We might lose control over early-stage debugging (Flash programming)
  $\Rightarrow$ We might be locked out completely in later product lifecycles

Overview

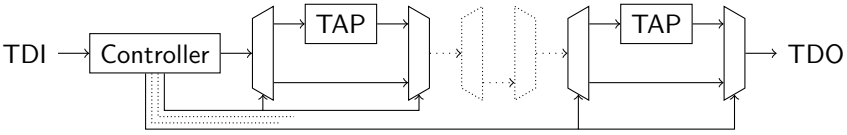
JTAG Switcher Principle

- A possible solution is to mux-in/mux-out malicious/unused TAP controllers from the chain

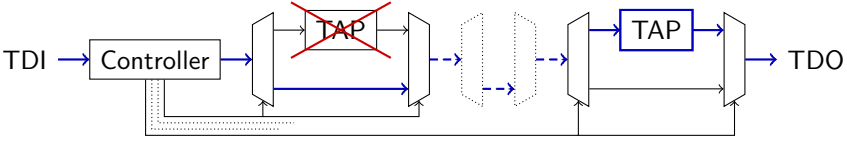


JTAG Switcher Principle

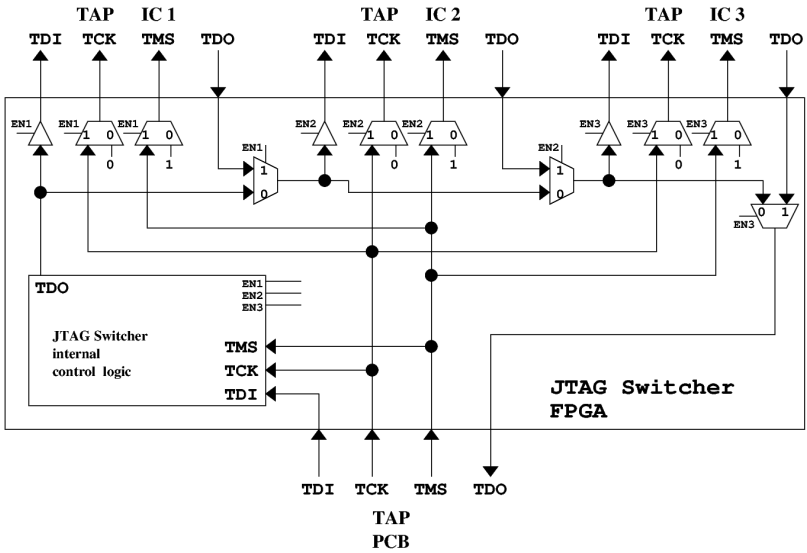
- A possible solution is to mux-in/mux-out malicious/unused TAP controllers from the chain



- Example: Select only last TAP, first TAP in error state

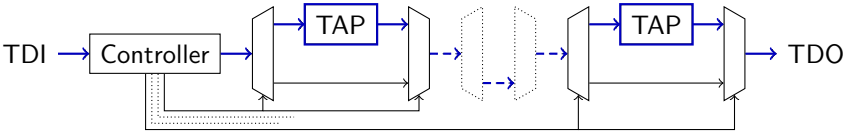


Schematical view of the JTAG Switcher FPGA IP



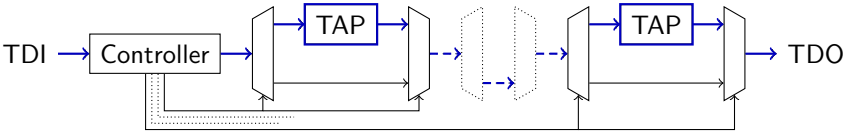
Static Chaining

Usecase 1: Form a classic daisy-chain by writing the configuration *once*

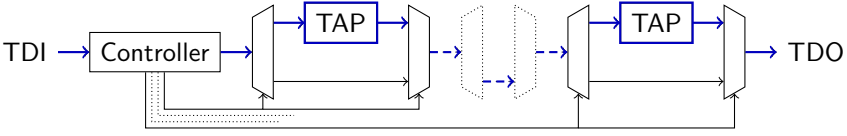


Static Chaining

Usecase 1: Form a classic daisy-chain by writing the configuration *once*

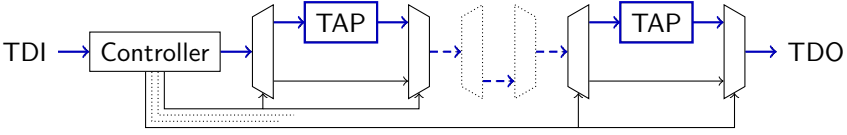


Problem:

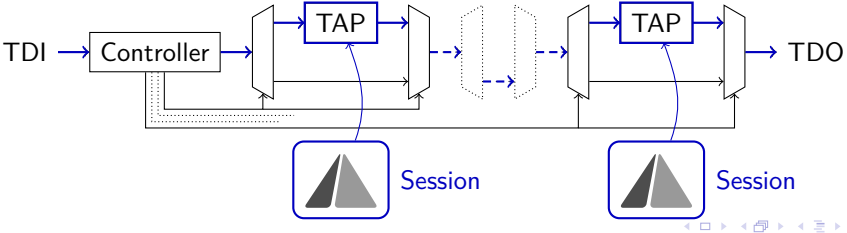


Static Chaining

Usecase 1: Form a classic daisy-chain by writing the configuration *once*

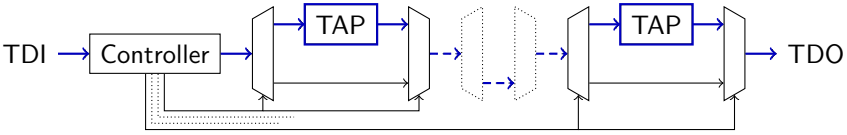


Problem:

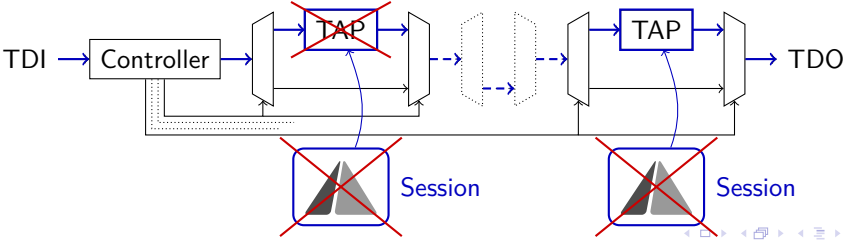


Static Chaining

Usecase 1: Form a classic daisy-chain by writing the configuration *once*



Problem:



Static Chaining

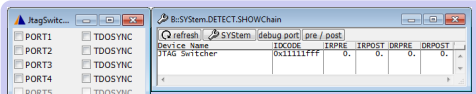
PRO

- Probe requires only Daisy-Chaining support
  $\Rightarrow$ wide tool support
- Industry standard technique

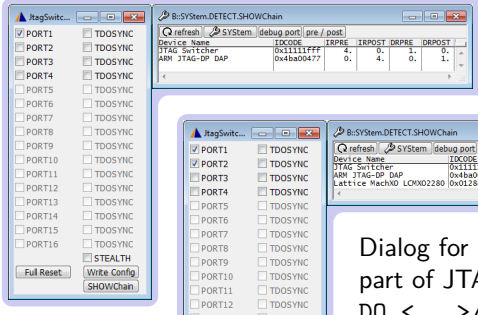
CON

- Configuration needs to be written manually e.g. SVF, TRACE32 RemoteAPI, `JTAG.SHIFT`, `JTAG.SEQuence`
- Not robust against dynamic errors of TAPs
- Performance limitation depending on Daisy-Chain length and TAP performance

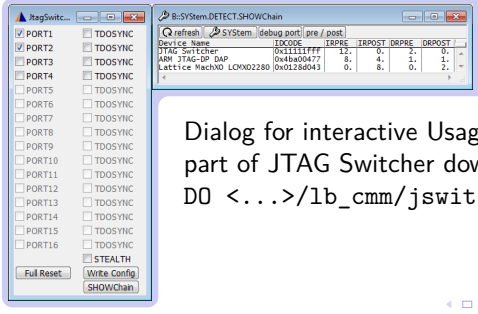
Usage



No PORT/TAP selected



First PORT/TAP selected



PORT/TAP **1 & 2** selected

Dialog for interactive Usage
part of JTAG Switcher download
`DO <...>/lb_cmm/jswitch_dialog.cmm`

Usage

```
IF !(JTAG.SEQuence.EXIST(TLRJSwitchReset)&&JTAG.SEQuence.EXIST(TLRJSwitchWriteReg))
(
  DO <...>/lb_cmm/jswitch_jtagsequence_lib.cmm
)
; optional
; JTAG.SEQuence.Execute TLRJSwitchReset

JTAG.SEQuence.Execute TLRJSwitchDisableAll
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 <Bank1 Value> <Bank2 Value>
```

Register SELECT

PORT/TAP 1-8

PORT/TAP 9-16

JTAG Switcher Register Bit mapping

Slave TAP	16		15		14		13		12		11		10		9	
Slave TAP	8		7		6		5		4		3		2		1	
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

No Operation	0	0
Set Bit	0	1
Clear Bit	1	0
Clear Bit	1	1

Examples

- Example: Enable PORT1

  ```
  JTAG.SEQuence.Execute TLRJSwitchDisableAll
  JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 0x1 0x0
  ```

  or (including disable all other ports pattern)

  ```
  JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 0x1|0xAAA8 0xAAAA
  ```

TLRJSwitchDisableAll

The JTAG Sequences `TLRJSwitchDisableAll` & `TLRJSwitchReset` are robust against 128 (remaining) IR-Bits in the chain (~32 ARM-SoCs).

TLRJSwitchWriteAllBanks

All JTAG Sequences `TLRJSwitchWrite...` must not be called when a debug session is active. The chain must include *ONLY* JTAG Switcher.

Examples

- Example: Enable PORTs 4,5,9

```
JTAG.SEQuence.Execute TLRJSwitchDisableAll
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 0x140 0x1
```

- Example: Reset JTAG Switcher including all miscellaneous registers

```
JTAG.SEQuence.Execute TLRJSwitchReset
```

- Example: Enable PORTs 4,5,9 - enable TDOSYNC for Port 4,5

```
JTAG.SEQuence.Execute TLRJSwitchDisableAll
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x2 0x140 0x2
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 0x140 0x1
```

Reminder

`TLRJSwitchDisableAll` does only reset the SELECT(=`0x1`) register, others e.g. TDO-SYNC(=`0x2`) are not affected.

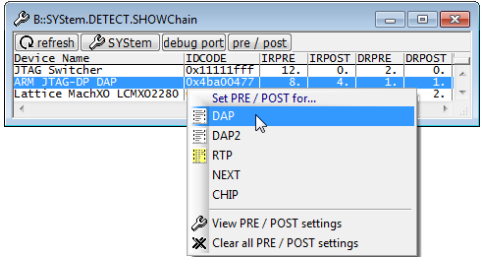
Usage

Debug Session configuration

```
SYStem.CPU <SoC-Cluster>
SYStem.CONFIG IRPRE  <>
SYStem.CONFIG DRPRE  <>
SYStem.CONFIG IRPOST <>
SYStem.CONFIG DRPOST <>
SYStem.Mode <Attach|Up>
```

ARM Coresight based System

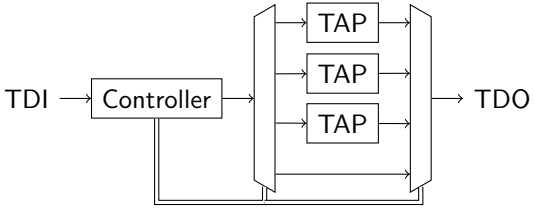
```
SYStem.CPU <SoC-Cluster>
SYStem.CONFIG DAPIRPRE  <>
SYStem.CONFIG DAPDRPRE  <>
SYStem.CONFIG DAPIRPOST <>
SYStem.CONFIG DAPDRPOST <>
SYStem.Mode <Attach|Up>
```



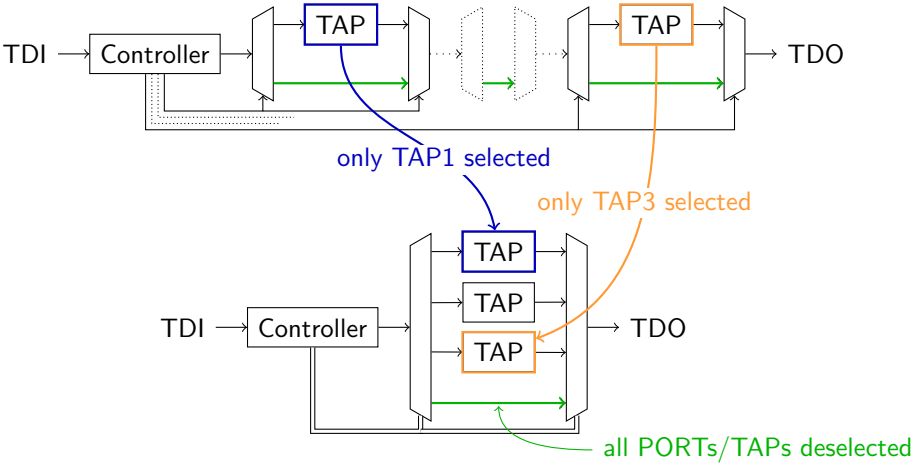
Dynamic Chaining

Usecase 2: Dynamically control the Daisy-Chain from the probe

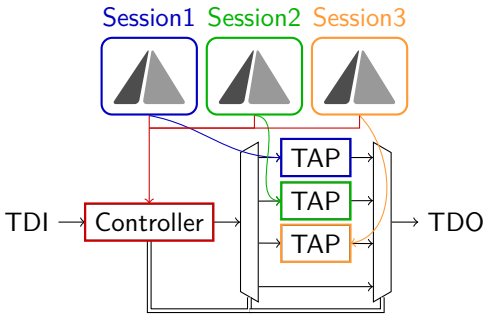
- Idea: Activate only one PORT/TAP at a time
- $\Rightarrow$ Chain consists only of JTAG Switcher and current TAP of *interest*
- $\Rightarrow$ Errors from other TAPs do not propagate
- Thus we may use JTAG Switcher as one logical multiplexer



Dynamic Chaining

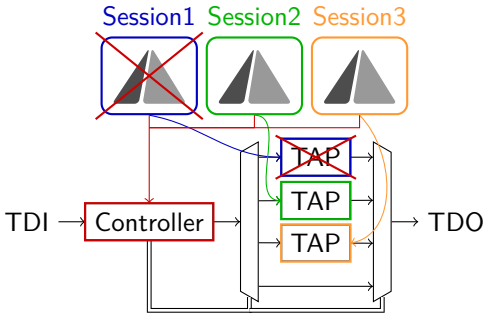


Dynamic Chaining



- Every Debug session connects (ideally) to only one TAP
- The probe (e.g. TRACE32) needs also to control the JTAG Switcher
- $\Rightarrow$ We can dynamically start debug-sessions without prior Daisy-Chain configuration

Dynamic Chaining



- In case one TAP has an (unlikely) error while the communication is active only this session is affected
- Example: TAP 1 in error state
  $\Rightarrow$ Session 2 & Session 3 remain active

Dynamic Chaining

PRO

- No *pre-initialization* required, configuration can be written by probe on demand
- We can dynamically *add* sessions
- Speed limitiations (maximum TCK) of one TAP does not propagate
- Better performance due to shorter Daisy-Chain length

CON

- Probe requires support for JTAG Switcher

Dynamic Chaining

Sequence - initial Connect
(TRACE32: `SYStem.CONFIG.MULTITAP JtagSEQuence Attach`)

- Bring possible remaining Daisy-Chained TAPs into a safe state (JTAG BYPASS)
- Disable all possibly activated JTAG Switcher PORTs (Reset all SELECT registers=`0x7`)

optional Write JTAG Switcher miscellaneous configuration registers (e.g. TDOSYNC)

- Write SELECT Register (Set Bit) and apply configuration
- Test-Logic-Reset of Slave-Port-TAP

Dynamic Chaining

Sequence - activate PORT
(TRACE32: `SYStem.CONFIG.MULTITAP JtagSEQuence SELect`)

- Write SELECT Register (Set Bit) and apply configuration
- Probe operation ...

Sequence - deactivate PORT
(TRACE32: `SYStem.CONFIG.MULTITAP JtagSEQuence DeSELect`)

- Probe operation ...

either Disable all activated JTAG Switcher PORTs
(Reset all SELECT registers=`0x7`)

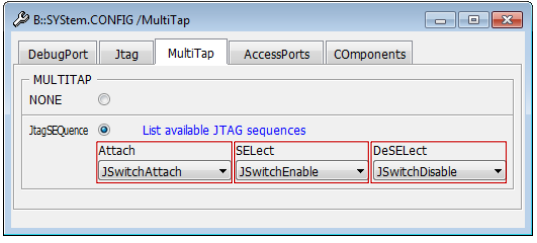
or Write SELECT Register (Clear Bit) and apply configuration

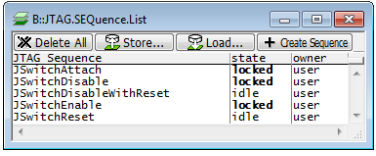
Dynamic Chaining - TRACE32

Usage:

```
SYStem.CPU <SoC-Cluster>
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=<x>
SYStem.Mode <Attach|Up>
```

The script `jswitch_multitap_jtagsequence.cmm` patches all required settings to control JTAG Switcher into the current TRACE32 session.





Examples

Example: STM32F103 on Port 1

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=1
SYStem.Mode <Attach|Up>
```

Example: STM32F103 on Port 1, activate TDOSYNC

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=1 TDOSYNC
SYStem.Mode <Attach|Up>
```

Example: Daisy-Chain on Port 1
JTAG Switcher -> FPGA (IR=5, DR=1) -> STM32F103 -> JTAG Switcher

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=1 IRPOST=5 DRPOST=1
SYStem.Mode <Attach|Up>
```

Examples

Example: Daisy-Chain on Port 5
JTAG Switcher -> STM32F103 -> FPGA (IR=5, DR=1) -> JTAG Switcher

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=5 IRPRE=5 DRPRE=1
SYStem.Mode <Attach|Up>
```

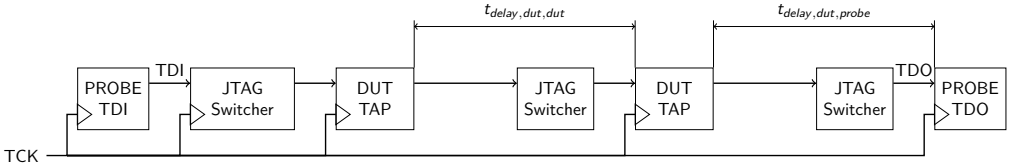
Example: ZYNQ-Ultrascale+ on Port 4
JTAG Switcher -> SoC internal Daisy-Chain -> JTAG Switcher

```
SYStem.CPU ZYNQ-ULTRASCALE+-APU
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=4 IRPOST=12 DRPOST=1
SYStem.Mode <Attach|Up>
```

Overview

TDOSYNC

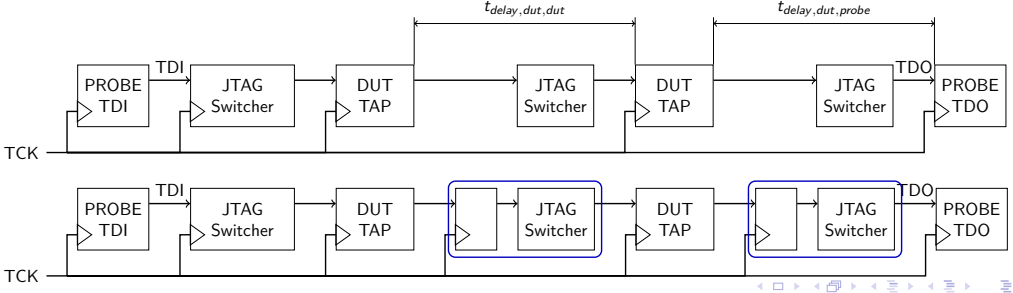
- Simplified timing model of a JTAG Switcher Daisy-Chain with two Slave TAPs/PORTs activated
- Two worst case timings $t_{delay,dut,dut}$ and $t_{delay,dut,probe}$
  Includes: level shifters, signal lines and delay in Slave TAP & FPGA



TDOSYNC

- Simplified timing model of a JTAG Switcher Daisy-Chain with two Slave TAPs/PORTs activated
- Two worst case timings $t_{delay,dut,dut}$ and $t_{delay,dut,probe}$
  Includes: level shifters, signal lines and delay in Slave TAP & FPGA

$\Rightarrow$ TDOSYNC - adds a synchronization register (virtual IR/DR Bit)



TDOSYNC Static Chaining

- TDOSYNC is available in the JTAG Switcher Dialog
- TDOSYNC can also be activated using the `jswitch_jtag_sequence_lib.cmm` & `JTAG.SEQuence.Execute` compare examples in Static Chaining section
- The inserted TDOSYNC register needs to be configured in the
  `SYStem.CONFIG [DAP]IRPRE`
  `SYStem.CONFIG [DAP]DRPRE`

TDOSYNC Static Chaining

- TDOSYNC is available in the JTAG Switcher Dialog
- TDOSYNC can also be activated using the `jswitch_jtag_sequence_lib.cmm` & `JTAG.SEQuence.Execute` compare examples in Static Chaining section
- The inserted TDOSYNC register needs to be configured in the
  `SYStem.CONFIG [DAP]IRPRE`
  `SYStem.CONFIG [DAP]DRPRE`

TDOSYNC Static Chaining

Example: enable PORT 1 with STM32F103, activate TDOSYNC

```
IF !(JTAG.SEQuence.EXIST(TLRJSwitchReset)&&JTAG.SEQuence.EXIST(TLRJSwitchWriteReg))
(
  DO <...>/lb_cmm/jswitch_jtagsequence_lib.cmm
)
; optional
; JTAG.SEQuence.Execute TLRJSwitchReset

JTAG.SEQuence.Execute TLRJSwitchDisableAll
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x2 0x1 ; TDOSYNC
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanks 0x1 0x1 ; SELECT

SYStem.CPU STM32F103
SYStem.CONFIG DAPIRPOST 4. ; JTAG Switcher
SYStem.CONFIG DAPDRPOST 1. ; JTAG Switcher
SYStem.CONFIG DAPIRPRE 1.  ; TDOSYNC
SYStem.CONFIG DAPDRPRE 1.  ; TDOSYNC
SYStem.Mode <Attach|Up>
```

TDOSYNC Dynamic Chaining

- TDOSYNC is available as a parameter in `DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm`
- Daisy-Chaining paramters are automatically calculated

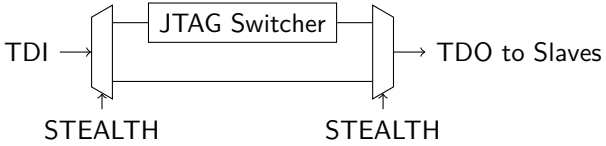
Example: enable PORT 1 with STM32F103, activate TDOSYNC

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=1 TDOSYNC
SYStem.Mode <Attach|Up>
```

STEALTH Mode

- Some SoCs require to be *first* in chain
- Examples: TriCore, C166, possibly others
- In order to daisy-chain these devices, STEALTH mode hides JTAG Switcher in the Daisy-Chain

$\Rightarrow$ First enabled PORT becomes first in chain



STEALTH Usage

Static-Chaining: enable PORT 1 with STM32F103, activate STEALTH

```
IF !(JTAG.SEQuence.EXIST(TLRJSwitchReset)&&JTAG.SEQuence.EXIST(TLRJSwitchWriteReg))
(
  DO <...>/lb_cmm/jswitch_jtagsequence_lib.cmm
)
; optional
; JTAG.SEQuence.Execute TLRJSwitchReset

JTAG.SEQuence.Execute TLRJSwitchDisableAll
JTAG.SEQuence.Execute TLRJSwitchWriteAllBanksStealth 0x1 0x1 ; SELECT + STEALTH

SYStem.CPU STM32F103
SYStem.Mode <Attach|Up>
```

Dynamic-Chaining: enable PORT 1 with STM32F103, activate STEALTH

```
SYStem.CPU STM32F103
DO <...>/lb_cmm/jswitch_multitap_jtagsequence.cmm PORT=1 STEALTH
SYStem.Mode <Attach|Up>
```

Overview

- VHDL Source available under MIT License
- Tested on Lattice MachXO/iCE40/ECP5, Altera MAX V/10, Xilinx CoolRunner
- IP is configured using a single VHDL file `jswitch_config_pkg.vhd`
- Source: https://www.lauterbach.com/jtag_switcher.html

Example to share JTAG probe connector between FPGA and JTAG Switcher