Master-Thesis Colloquium

A Hardware-Supported Boot and Debug Unit for RISC-V + eFPGA SoCs

Stephan Proß

March 16, 2023

Motivation

FABulous Framework

- Embedded FPGA generation
- Open Source
- Modular & customizable
- Target: ASIC/FPGA
- Bit-Stream generation
- Case Study: RISC-V + eFGPA

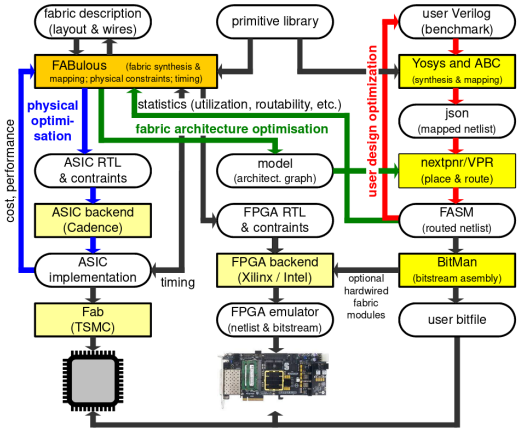


Figure: FABulous Framework[3]

Combining RISC-V & eFPGA

Case Study

RISC-V + eFPGA hybrid:

- 32-bit RISC-V CPU
- eFPGA as custom instruction extension

$\Rightarrow$ No independence

Independence Benefits

- Application versatility
  - Cooperative, Boss-Worker, Parallel
- Robustness

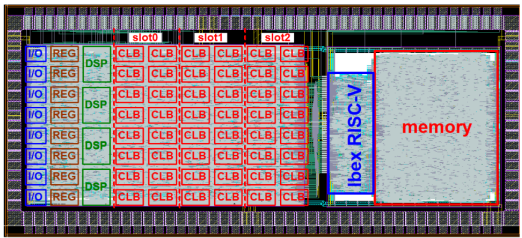


Figure: RISC-V + eFPGA hybrid[3]

FABulous SoC

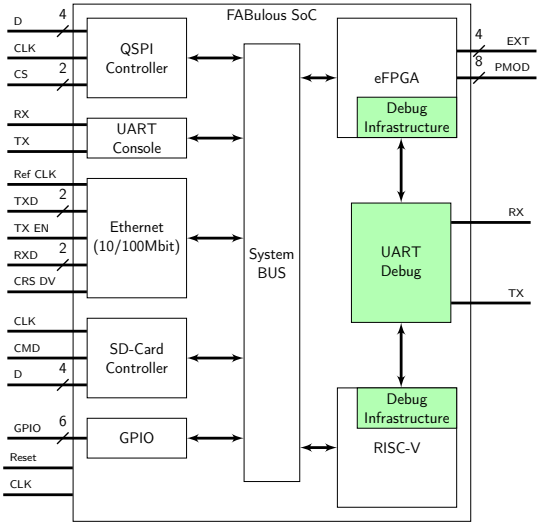
FABulous SoC

CVA6 RISC-V + eFPGA:

- 32-bit RISC-V CPU
- Linux support
- Independent eFPGA

Goals:

- Academic usage
- Versatility
- Traceability



Design Goals

Traceability

- Single step debugging of CPU
- Debugging from reset
- System bus activities
- Memory access

Versatility / Utility

- Features exceeding debug

Robustness

- Independence from system state
- Re-use of existing solutions

How to bring up CPU & FPGA ?

$\Rightarrow$ Anatomy of a RISC-V CPU

RISC-V CPU

RISC-V ISA

- Extensions
  - 32-bit Base Integer
  - FPU,Vector ...

CVA6 Implementation

- Design Goal: OS-Support (Linux)
- Single Core, Single Hart
- Customizable
  - Data width (32,64)
  - FPU, MMU ...

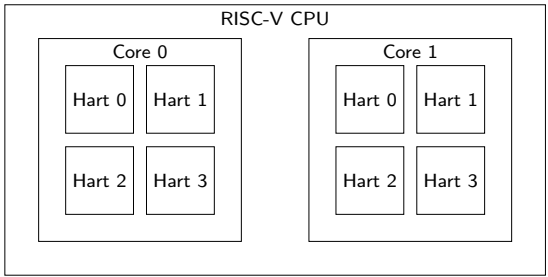


Figure: RISC-V CPU Example

Booting RISC-V

Power-On/Reset

Hard-coded Preliminary Boot-Code

- Harts begin at Reset-Vector (0x100)
- Setup of registers:
  - Hart-ID
  - Flattened-Device-Tree (FDT) address
  - Start-Address

Boot-Code

Resides in (Flash-)Memory

- Setup of hardware using FDT, application environment
- Handover to
  - Bare-metal program
  - OS

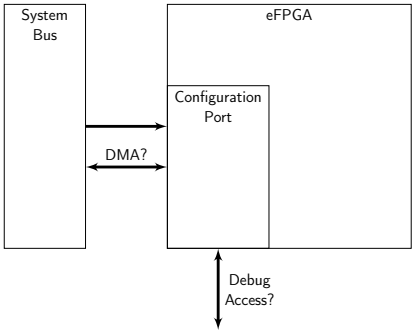
Requirements

Access to system (flash-)memory

Booting eFPGA

Power-On/Reset

- Loading configuration
  - External write (CPU, debug infrastructure)
  - DMA from memory (flash, RAM)
- Fabric enable



Requirements

Configuration port access

Debugging Embedded Systems

General Debugging

- R/W access
  - Process memory, registers
- Control of execution state
  - Halt, resume
- Code execution / patching
  - Breakpoints

Requirement: System Calls

- Unix: ptrace

Issue: Embedded Systems

Without OS:

- System calls unavailable
- Remote target

Requirements

- Remote communication
- Hardware debug support

Debugging RISC-V

Debugger GDB

- Remote debug using TCP/IP

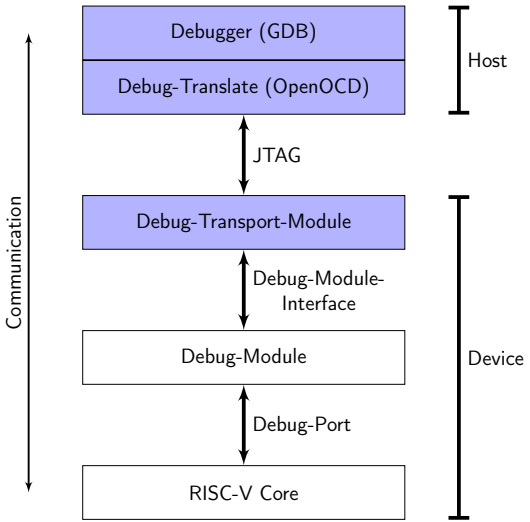
Debug Translation

Open On-Chip Debugger (OpenOCD)

- GDB interface (gdbserver)
- Translates to target (RISCV, ARM, ...)
- Handles Transport (JTAG, SPI..)

Debug Transport Module

Access to Debug-Module-Interface



Debug Module

Debug Module

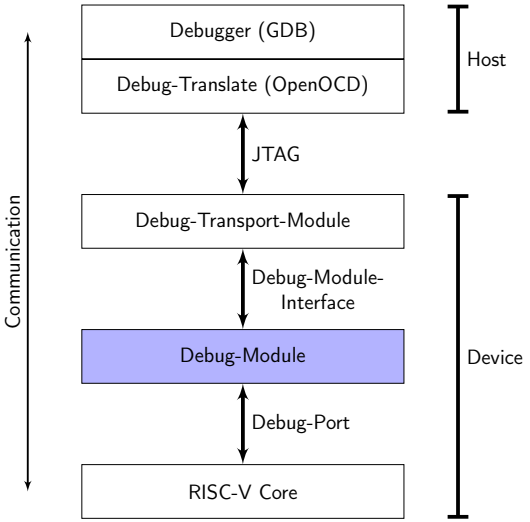
Implements:

- Reset, halt, resume of Harts
- Arbitrary command execution
- Register & memory access

Implementation Options

One of:

- Abstract commands
- Program buffer
- System bus access



Existing Infrastructure

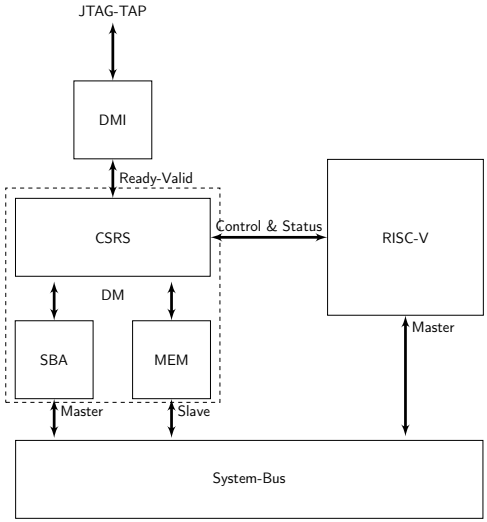
RISC-V Debug Project

Integrated into CVA6
Features:

- Register access
- System bus access
- JTAG as transport

Debug Module

CSRS Debug registers
MEM Program buffer
SBA System bus access



Remaining Work

OpenOCD

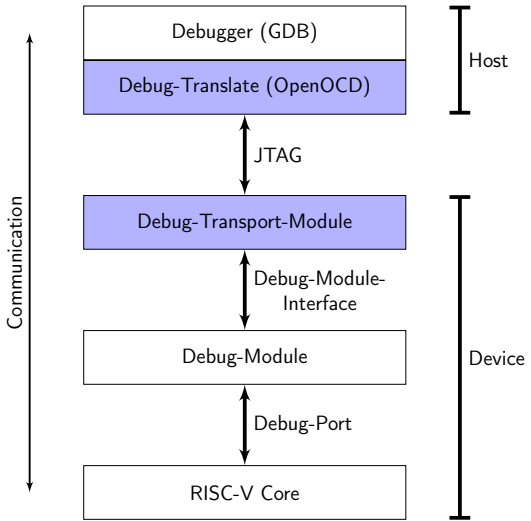
Available transports

- JTAG, SPI ...

No UART support!

Debug-Transport-Module

- JTAG only
- Additional features



JTAG vs. UART

JTAG

Wires:

- TDI, TDO: Serial Data I/O
- TMS: State Machine Control
- TCK: Clock from host

Properties:

- Synchronous
- Instruction (IR) and data (DR) registers
- TMS can trigger reset

UART

Wires:

- RX, TX: Serial Data I/O

Properties:

- Asynchronous
- Transmission in frames:
  1. Start (1 bit)
  2. Data (5-9 bits)
  3. Parity (0-2 bits)
  4. Stop (1-2 bits)

UART-Integration

JTAG-UART Adapter

Host software:

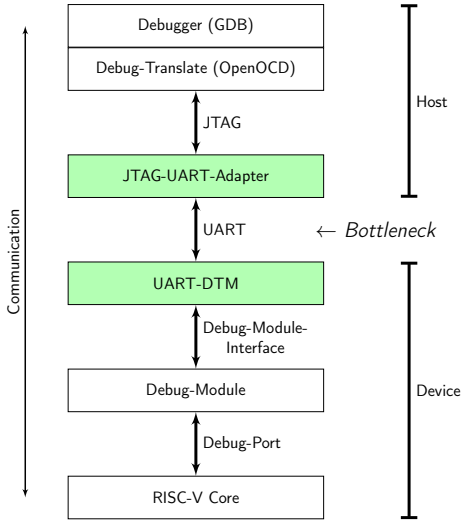
- Maintainability
- Speed

Purpose:

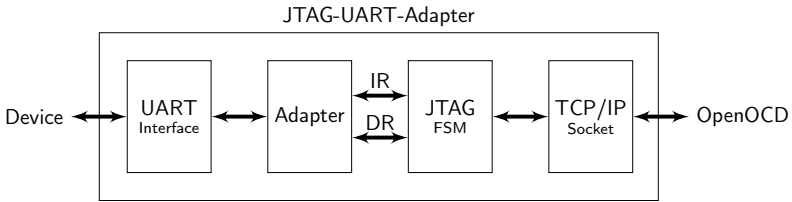
- Functional translation

UART DTM

- UART-Interface
- (Legacy) features



JTAG-UART Adapter



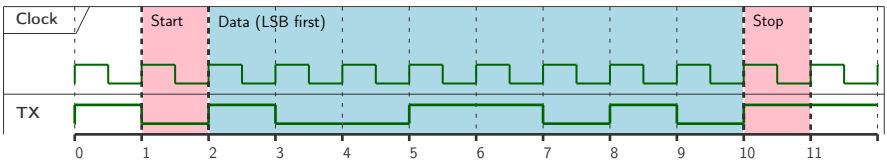
Components

- JTAG-FSM
  - (De-)Serialization
- UART-Interface
  - Byte-wise exchange
- Adapter
  - Functional translation

Issue

Distinction between data and instruction

UART Communication



Common Case 8-bits Data, 0-bits Parity, 1-bit Stop

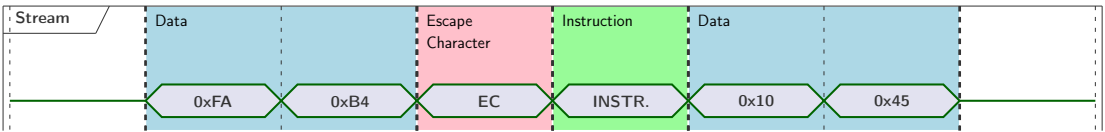
Data Rate

- Symbol/Baud rate f_s
- Serial $\Rightarrow$ f_s = raw bitrate f_b
- Common Case: effective data rate $f_{eff} = f_s$ 80%

Debug Protocol

- Mark instruction/data
- Low overhead

Protocol: Instruction Escape



Protocol

- Escape Character (EC) before instr.
- EC as data $\Rightarrow$ repeat

Properties

- $40\%\ f_s \leq f_{eff} \leq 80\%\ f_s$
- Lin. Dist. : $p = 1/256$
  - $f_{eff} = 79.7\%\ f_s$

Break even point: $p = 1/7 \approx 37/256$

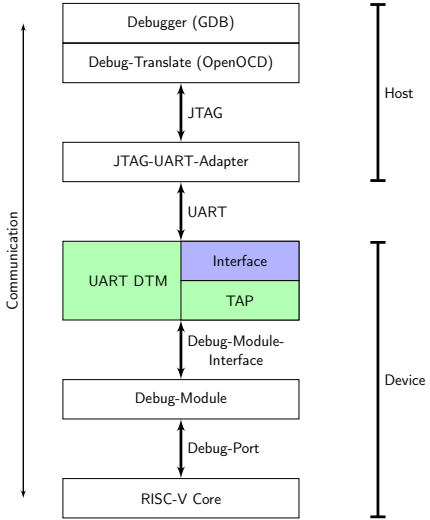
UART-DTM

UART-Interface

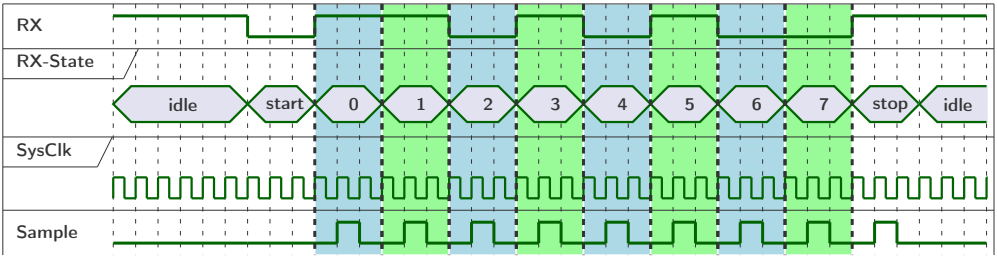
- Sampling of RX
- Un-/packing data from/to frames

UART-TAP

- Debug-Protocol interpretation
- Read/write registers



UART Interface



RX Sampling

- Process start w. falling edge
- Centered in symbol period

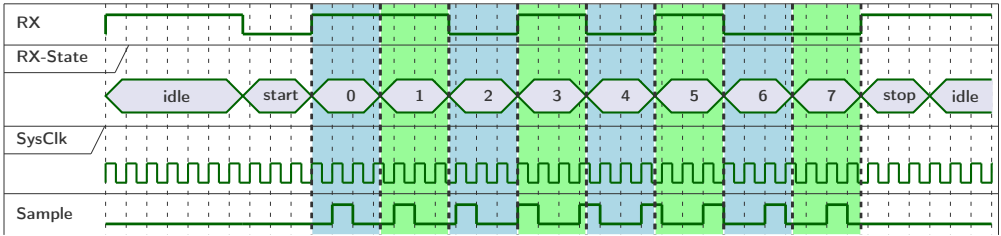
Sampling Interval

- $f_{sys} = 10MHz$
- $f_s = 1MBd$

Clock divider:

- $C := \lfloor f_{sys}/f_s \rfloor = 3$

Sampling Interval



Sampling Interval

- $f_{sys} = 10MHz$
- $f_s = 3MBd$

Clock divider:

- $C := \lfloor f_{sys}/f_s \rfloor = 3$

Issue

Integer Division of f_{sys} f_s
$\Rightarrow$ Remainder causes drift!

Sampling Solutions

1. Solution: Correction w. Edge Detection

Edges in data used as synchronization

- Data dependency!
- Expensive logic

$\Rightarrow$ Only Mitigation

2. Solution: Remainder Counting

"Add" remainder to delay Sampling.

- Data Independent
- Small additional counter

$\Rightarrow$ Actual Prevention

Flow-Control

Issue

Target device might be too slow / busy

Solution

Software Signaling:

- FIFO-Buffer w. fill-level signal
  - Latency is critical
- Halt-Signal to host
  - ASCII: XON/XOFF?
  - Escape Character!

Optimal Buffer Depth ?

- EC transmission time
- TX can be busy
- Host Latency?

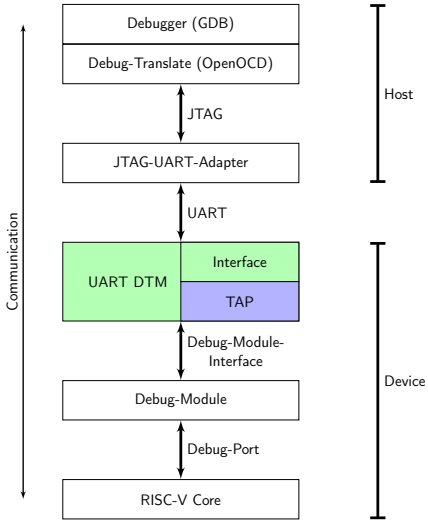
UART-TAP

Requirements

- Legacy Registers
- Peripheral access
- Protocol Interpretation
- R/W Streaming

Goals

- Resource Efficiency
- Utility
- Extendability



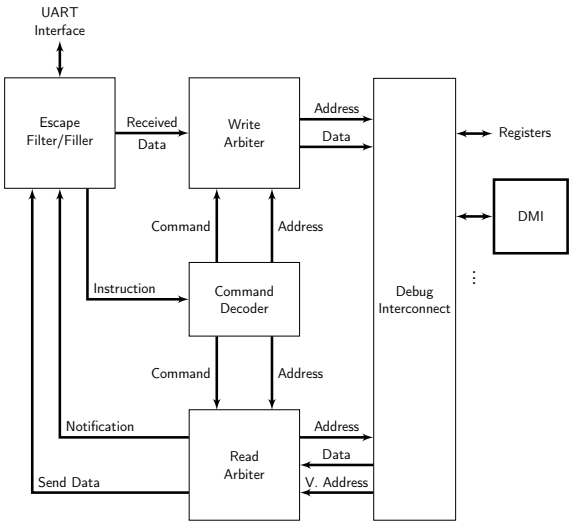
TAP Architecture

Instructions

- Continuous Write
- Single / Continuous Read
- Reset

Additional Features

- Escape Character handling
- Byte-wise (de-)serialization
- Bi-directional streaming
- Host-Notification



Debugging FPGAs

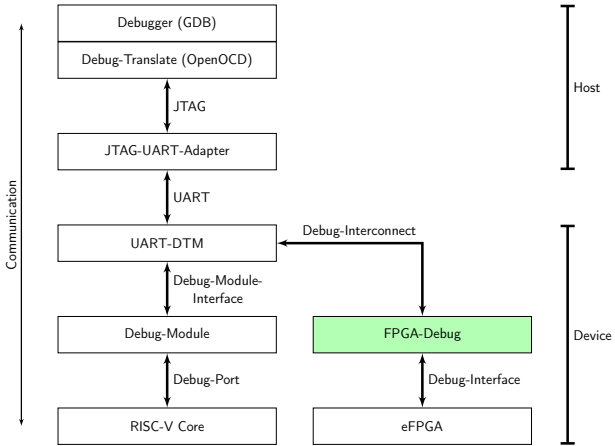
Debug Stack

- JTAG replaced with UART
- Original functionality preserved
- Efficient streaming

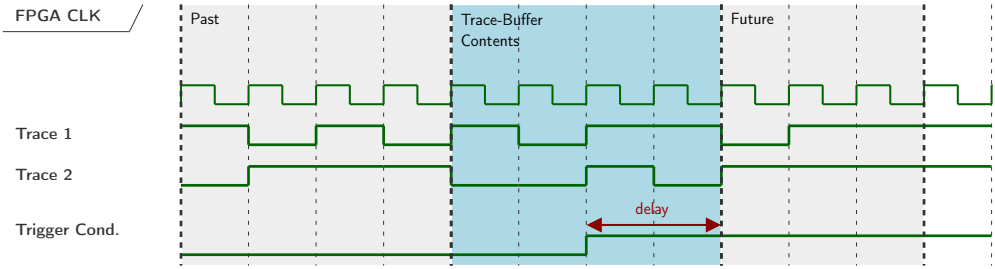
FPGA Debugging

Common Capabilities:

- Trace Capture
- Trigger Logic
- Basic data exchange



Trace Capture



Trace Buffer

- Ring Buffer
- Continuous Signal Capture
- Trigger cond. stops capture

⇒ Logic Analyzer

Trigger Condition

- Arbitrarily complex
- Delay:
  - Ratio of history to pre-history

FPGA Debug Infrastructure

Why?

Debug infrastructure in soft logic:

- Less logic for user design

Large area overhead:

- 1x 1kB SRAM Cell $\approx$ 32x LUT4

$\Rightarrow$ hard logic smaller

Proposed Solution

Streaming Trace Buffer (STB)

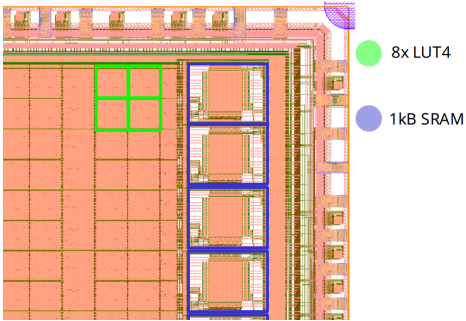


Figure: Caravel with eFPGA[2][1]

Streaming Trace Buffer

Components

- 1kB SRAM cell used as:
  - Trace-Buffer
  - RW-Buffer
- FPGA-Interface
  - Trace-Capture
  - Clock Domain Crossing
- System-Interface
  - Connects to debug interconnect

Modes

- Trace Mode
  - Variable number of trace signals
  - More signals less history
- Stream Mode
  - Bi-directional

Trigger Logic?

- Relegated to user logic
- Single signal is trigger
- Variable delay

Overview

FPGA Interface

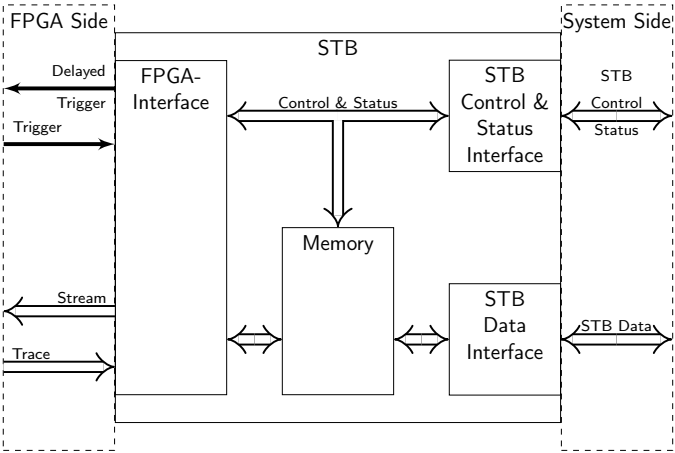
- Trace-Capture
  - De-/Serialization
- Trigger Delay

Memory Controller

- Pointer Handling
- RW Strobing

System Interface

- Control & Status
- 32-bit data exchange



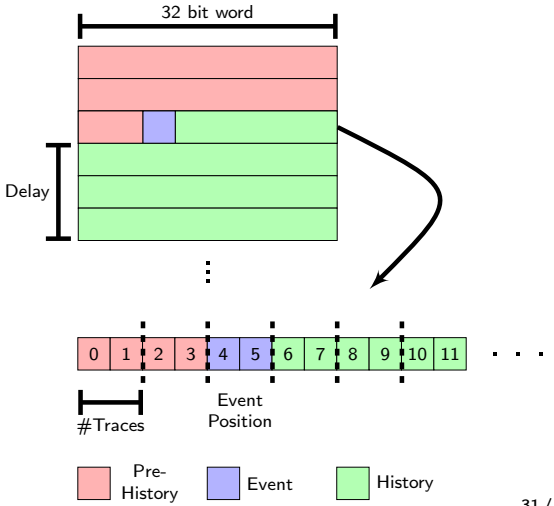
History Reconstruction

Trigger event:

- Full SRAM readout
  - First value = history begin

Required Information

- Control Registers:
  - Delay ⇒ Memory word in readout
  - Number of Traces ⇒ Granularity
- Status:
  - Event Position Status



Implementation

FPGA Emulation

Xilinx Vivado 2021.1 targeting Nexys-4 DDR Board

- System Clock : 25 MHz
- Baud Rate : 3 MBd

ASIC

- SRAM-Macros: OpenHW Skywater 130nm
- Tool-Chain: Yosys
- Available space: 10mm^2

Area Requirements?

Area STB

Component	Area (μm^2)	Total%	LUTs	Registers
STB	184204.5	100.0	277	340
- FPGA-Interface	12406.6	6.7	248	306
- Memory-Controller	171632.5	93.2	45	76
- - SRAM-Cell	167999.0	91.2		
- System-Interface	217.8	0.1	4	6
STB w.o. SRAM-Cell	16205.5	8.8	277	340

Observation

- SRAM-Cell dominates cost
  - Surrounding logic fractional

Total Area?

Debug Logic Area

Component	LUTs	LUT%	Registers	Register%
STB w.o. SRAM-Cell	277	100.0	340	100.0
UART-DTM	331	119.5	401	117.9
Debug Module	455	164.3	676	198.8
Total	1063	383.8	1417	416.8
CVA6-CPU (32bit)	13529		7402	

Observation

FPGA emulation:

- Costs of DTM & DM comparable to STB

Approximation

- LUTs & registers $\sim$ area

$\Rightarrow$ Total Area $\approx$ 40% of SRAM-Cell

Task Review

Device-to-Host using UART

- UART-DTM ☑
- Host Software ☑
- Debug Protocol ☑

FPGA Infrastructure

- Trace Capture ☑
- Data Exchange ☑
- Configuration Access (☑)
  $\Rightarrow$ Access via System-Bus

Questions

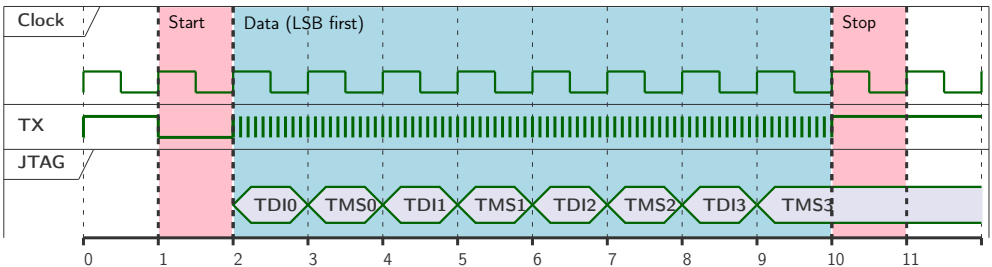
References I

[1] *Chip Gallery — FABulous Dokumentation 0.1 Documentation*. URL: https://fabulous.readthedocs.io/en/latest/gallery/index.html (visited on 03/15/2023).

[2] *GitHub - Nguyendao-Uom/eFPGA_v3_caravel: https://caravel-user-project.readthedocs.io*. URL: https://github.com/nguyendao-uom/eFPGA_v3_caravel (visited on 03/15/2023).

[3] Dirk Koch et al. "FABulous: An Embedded FPGA Framework". In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '21. New York, NY, USA: Association for Computing Machinery, Feb. 2021, pp. 45–56. ISBN: 978-1-4503-8218-2. DOI: 10.1145/3431920.3439302. URL: https://doi.org/10.1145/3431920.3439302 (visited on 01/13/2023).

Protocol: JTAG Over UART



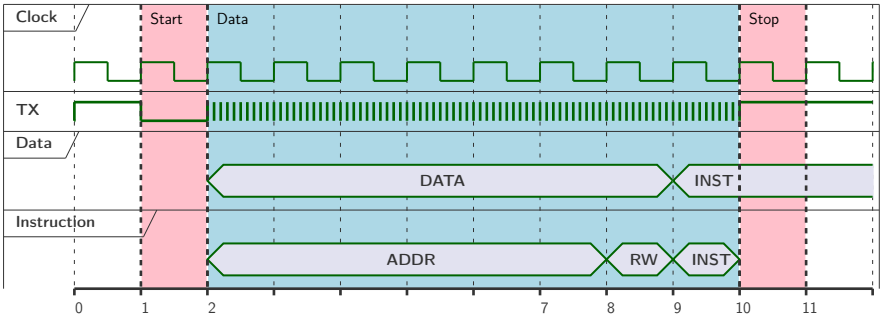
Protocol

- Wrap TDI, TMS in a frame
- TCK generated by receiver

Properties

- $f_{eff} = f_s$ 40%
- JTAG-TAP reusable

Protocol: Instruction Bit



Protocol

- Reserve one bit of data
- Instruction: Address & command

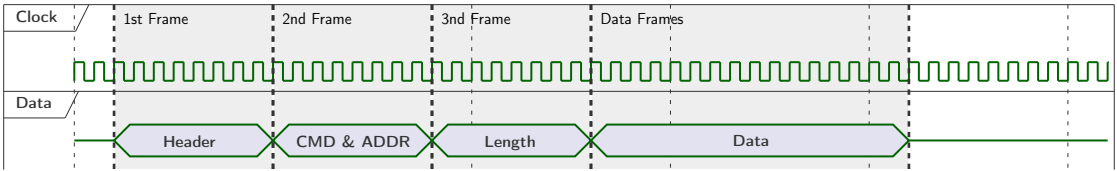
Properties

- $f_{eff} = f_s$ 70%

Worse for 32-bit data:

- 5 Frames $\Rightarrow f_{eff} = f_s$ 64%

Protocol: Instruction Packets



Protocol

Packet Header, instruction, (length), data

Properties

Assuming 4 bytes data:

- $f_{eff} = 53\%\ f_s$
- Break even point: 14 bytes data

Additional Features

Observations

- USB-to-UART chip single channel
- Additional data paths desirable

Direct RX,TX access to eFPGA:

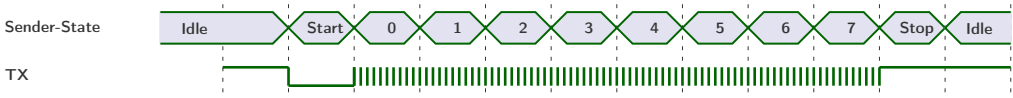
- Simple messaging
- UART-Interface in eFPGA

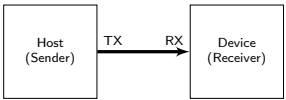
Multi-Channel Support

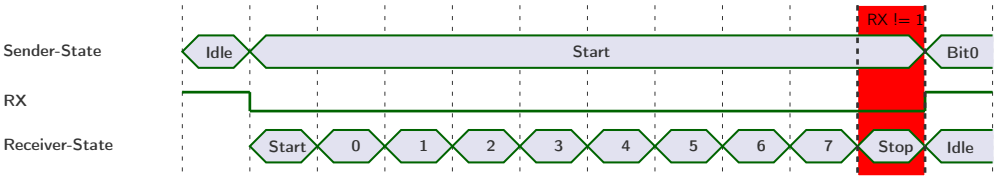
Allow debug infrastructure and eFPGA usage of interface

- Auto detection
- Low protocol overhead

Multi-Channel







Condition

$f_{Receiver} \geq 10 * f_{Sender}$

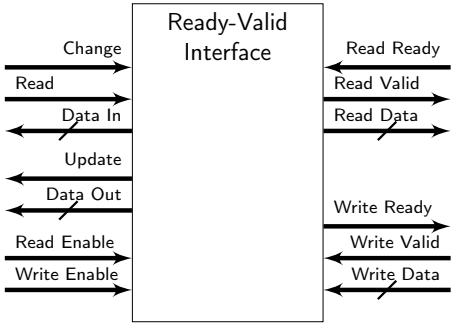
System Side

Modular RV-Register

- Control & Status Interface
- Data Interface
  - 32 bit asynchronous R/W

Control & Status Interface

- Trigger Delay
- Number of Traces
- STB Mode
- Trigger Event
  - Presence
  - Bit Position

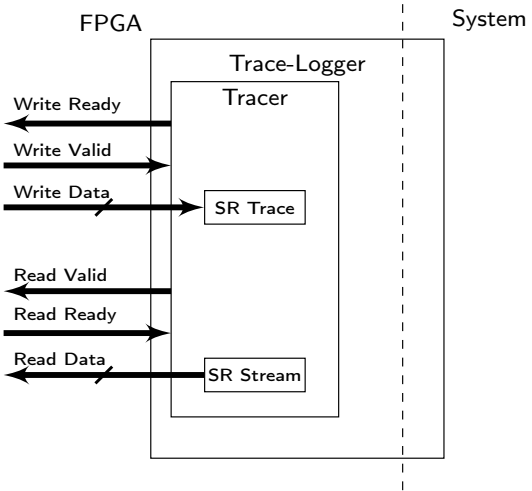


Stream Mode

Properties

- Bi-directional, async. R/W
- Ready-Valid signaling:
  - Trigger → Write Valid
  - Delayed Trg. → Read Ready
- Requirement: Matching I/O rates

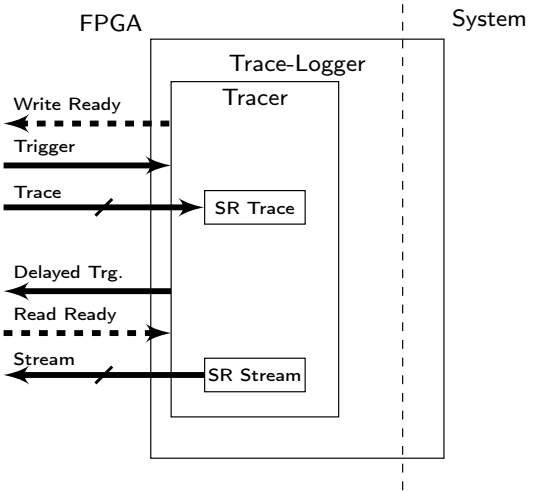
⇒ Uni-directional modes



Trace Mode

Properties

- Traces continuously captured in memory
- Oldest traces output via Stream
  - $\Rightarrow$ STB daisy-chaining
- Trigger halts capture after delay



STB Chaining

Dual STBs

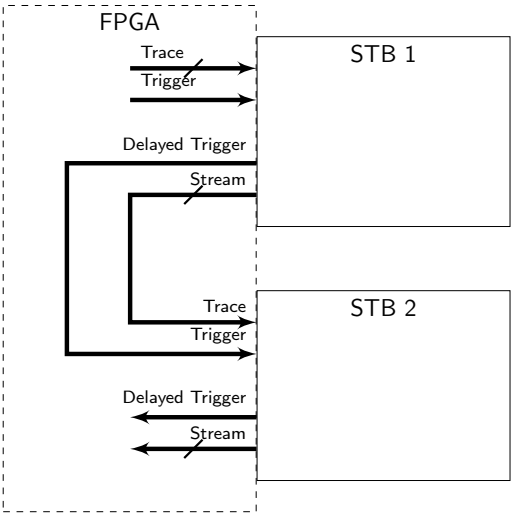
Allow:

- Tracing more signals in parallel
- Increase history depth
  - Chaining

Chaining

Stream outputs oldest trace:

- Second STB captures trace



Stream Throughput

Analysis

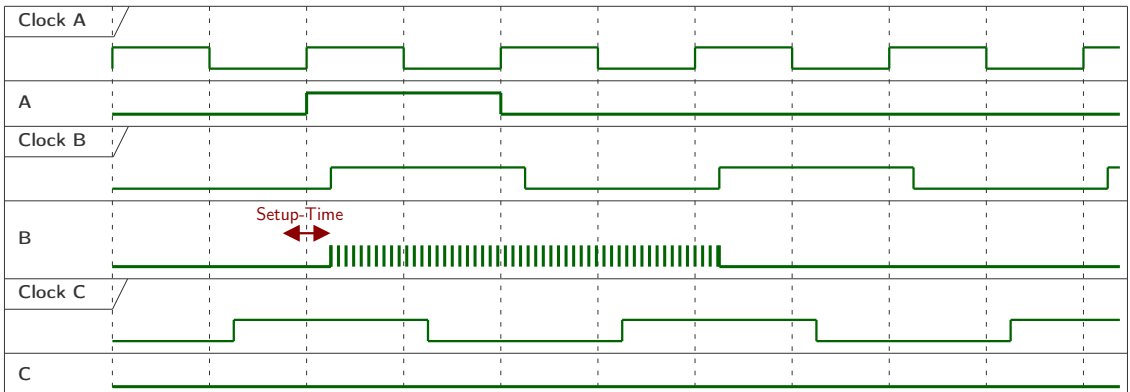
Protocol:

- $f_{eff} \approx$ 3MBd * 80% = 2.4 Mb/s

Hardware Bottlenecks?

- UART-DTM $\times$
- RV-Register $\times$
- FPGA-Interface

Clock Domain Crossing Issue



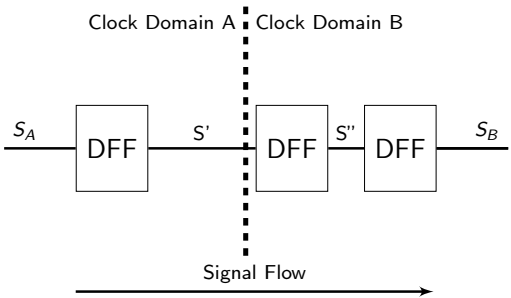
CDC Techniques

Open-Loop Synchronizer

- Basis: FF in source and target domain
- Adds FF to path in target domain
- Eventually Accurate

Stability requirement:

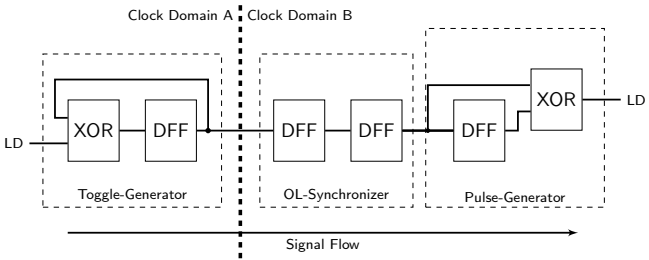
- Source signal stable for 1.5 times the target period.



CDC Techniques ff.

Multi-Cycle-Path Formulation

- Multiple data signals, one synchronizer
- Data registered in both domains
- Sample Accurate
- Sync: Pulse $\Rightarrow$ Toggle $\Rightarrow$ Pulse

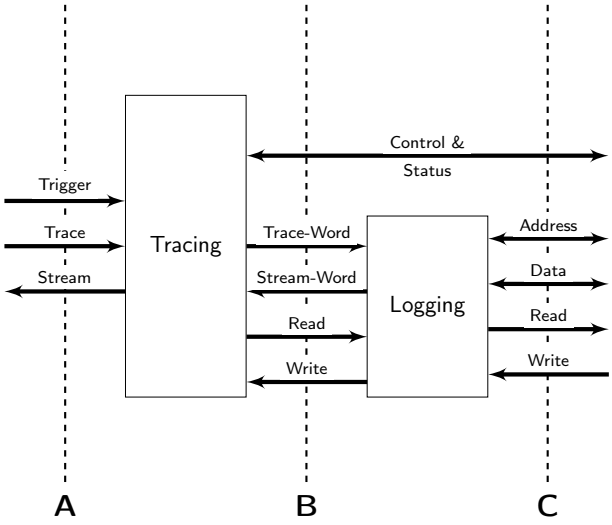


CDC Design

CDC Techniques

- Resource Costs
- Delay

How to partition logic into Clock Domains?



Implementation FPGA-Interface