Advanced SOC Design

FSIC Architecture & Design

Jiin Lai

Agenda

- FSIC Architecture
- FSIC-AXIS Interface Specification
- System Clocking Scheme
- Module Specification

Markdown documents:

https://drive.google.com/drive/folders/1rDidHtD9xPSkbfLaLT7Mx2P97Wgqvc9x?usp=sharing

FSIC Architecture

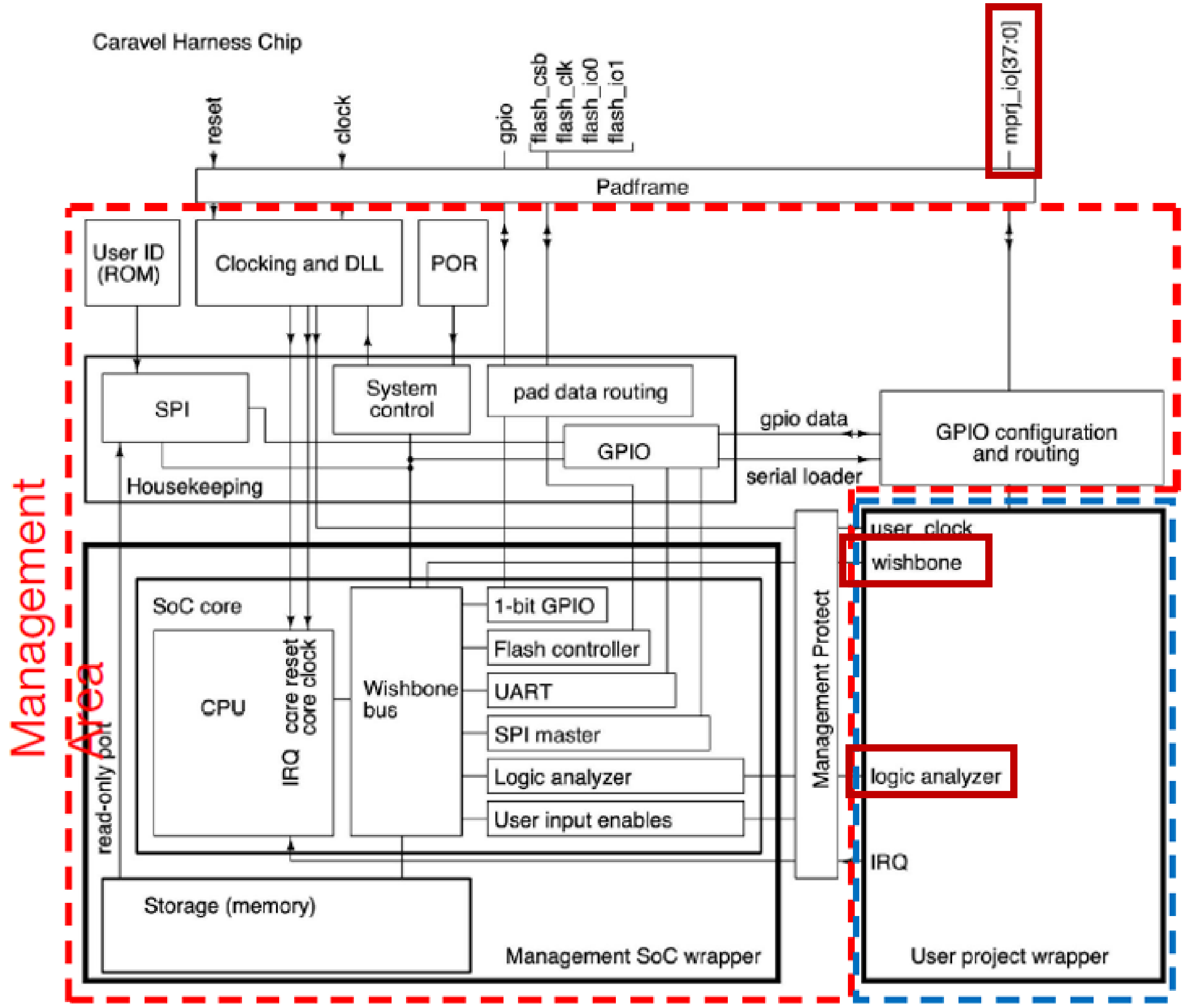
FSIC （Full-Stack IC）is an architecture to implement an IC validation system based on Caravel SOC.

A series of laboratories are designed based on it for the course "SOC Design".

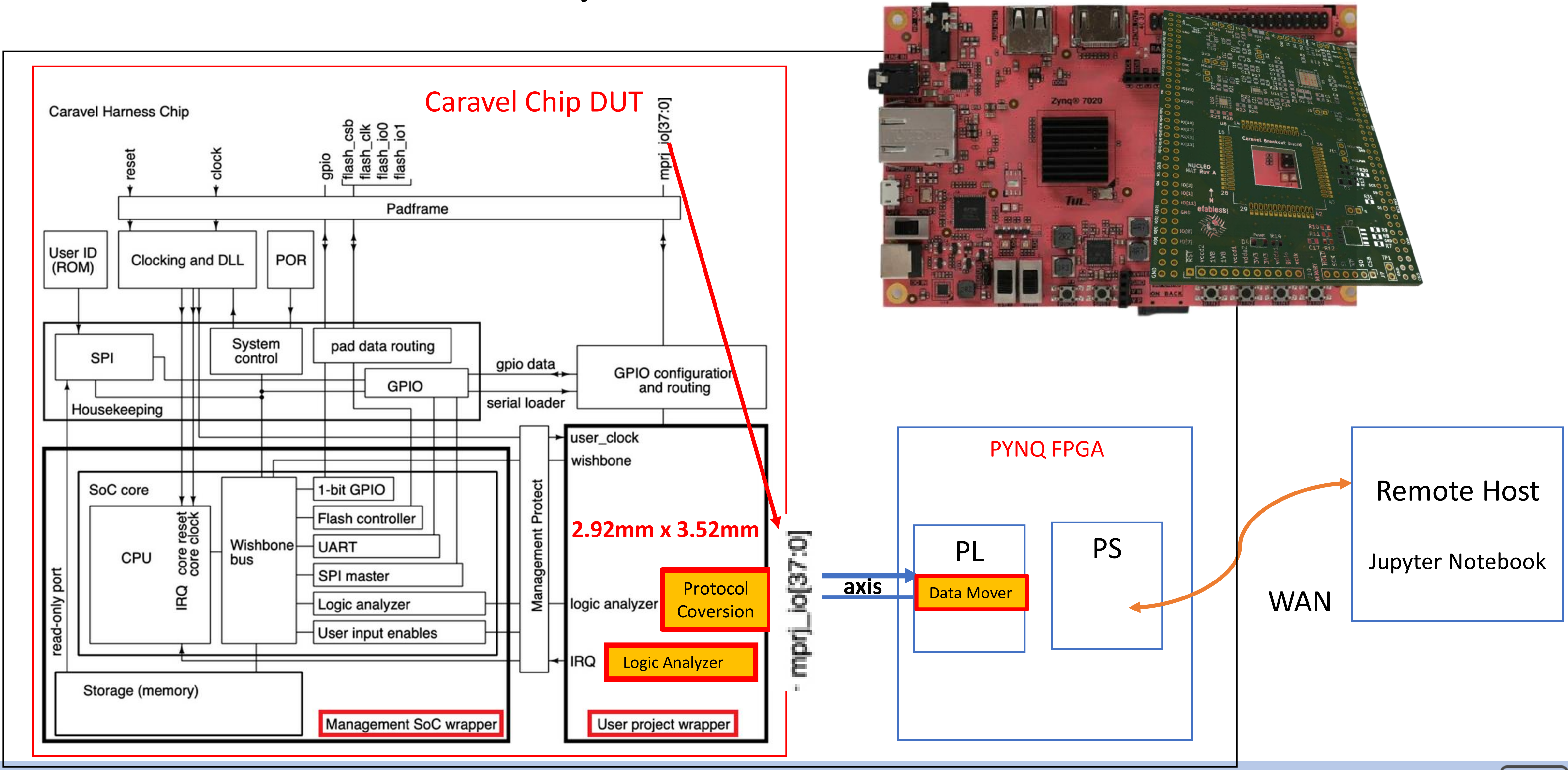
It trains the following disciplines:

- IC Design
- FPGA Design
- Embedded Programming

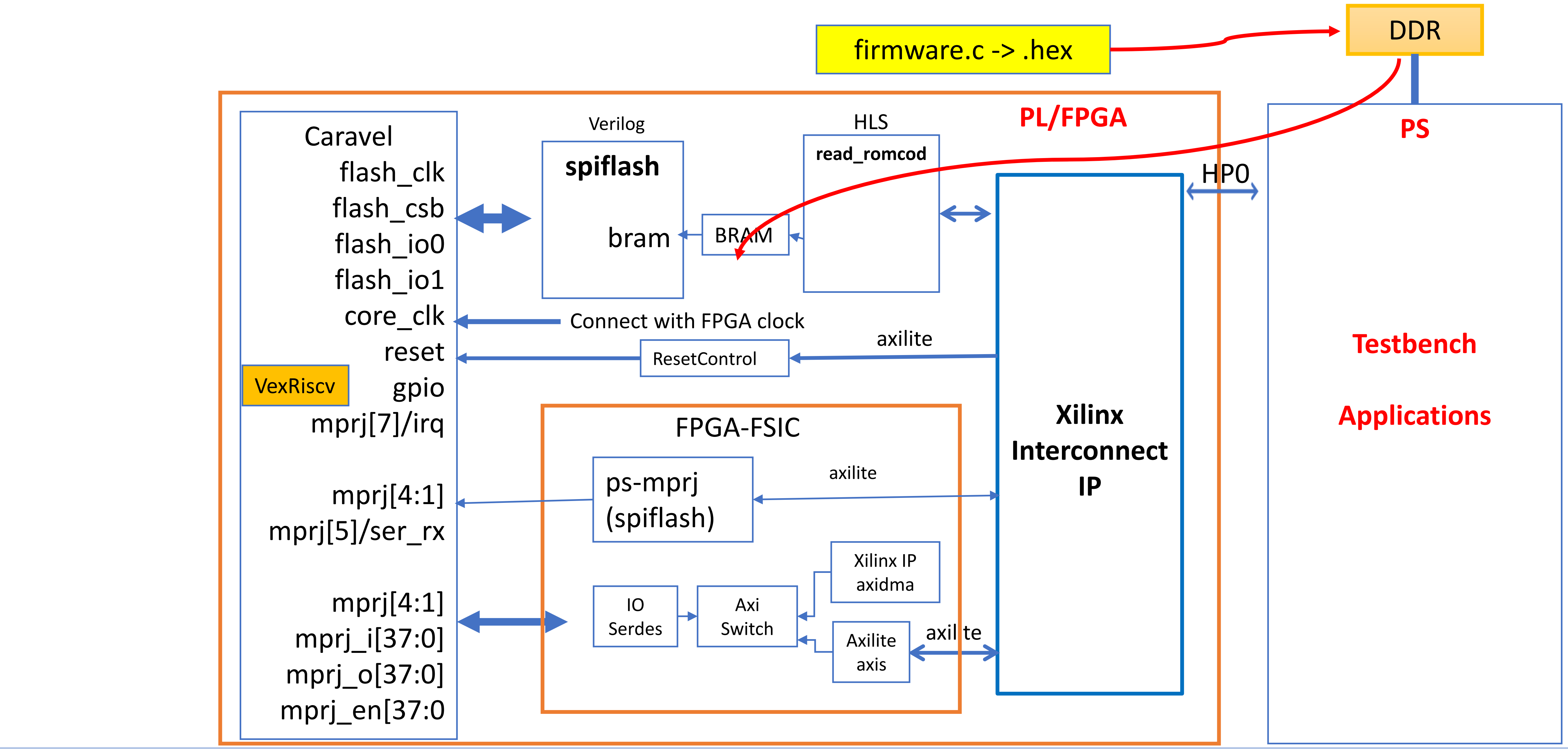
Caravel Harness



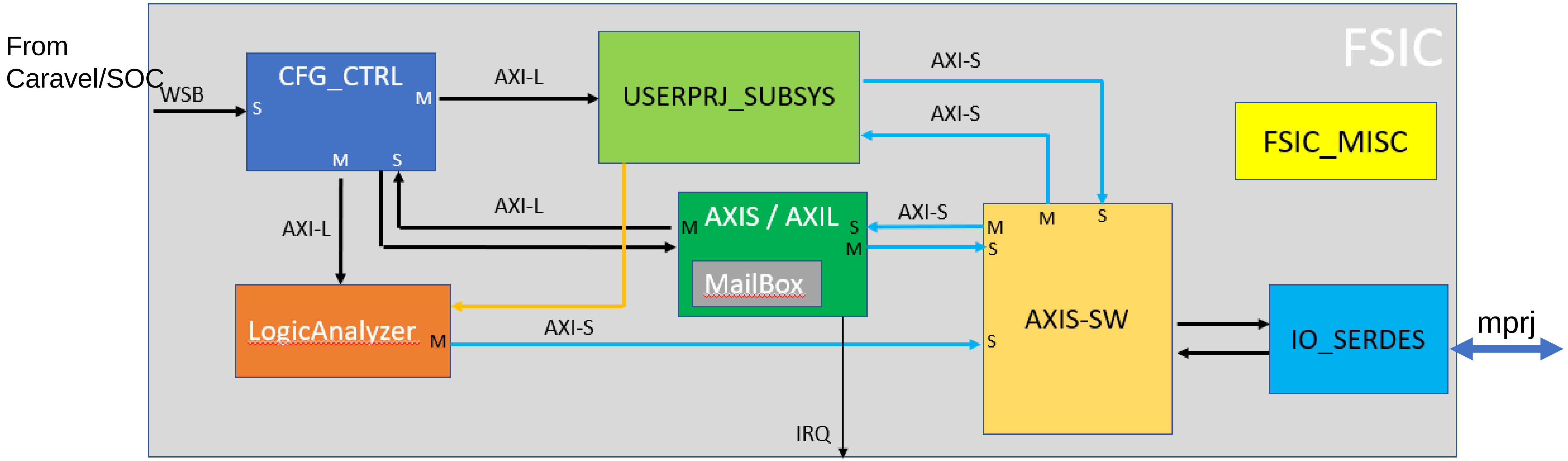
FSIC - IC Validation System



Caravel-FSIC + FPGA-FSIC

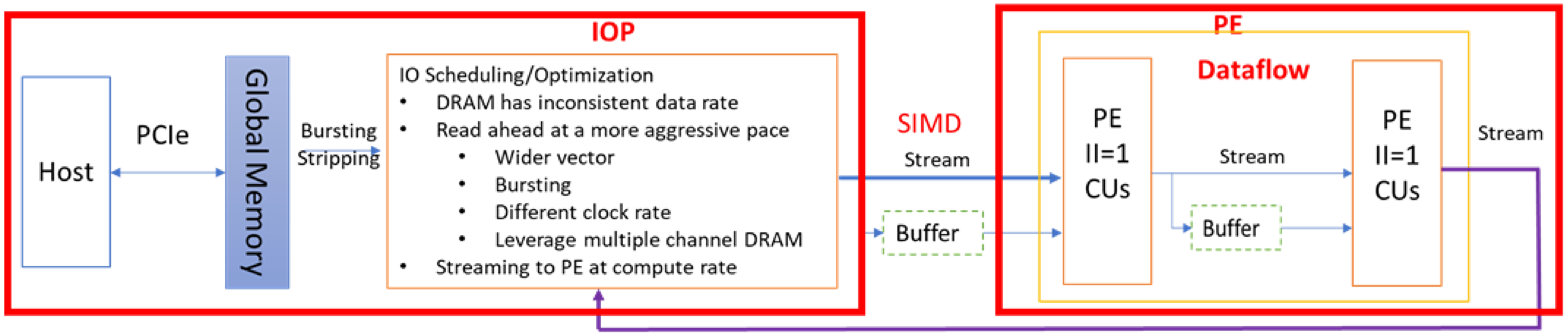


User Project Wrapper



https://hackmd.io/ny1VCNWGQE--AkNyWRdrOw

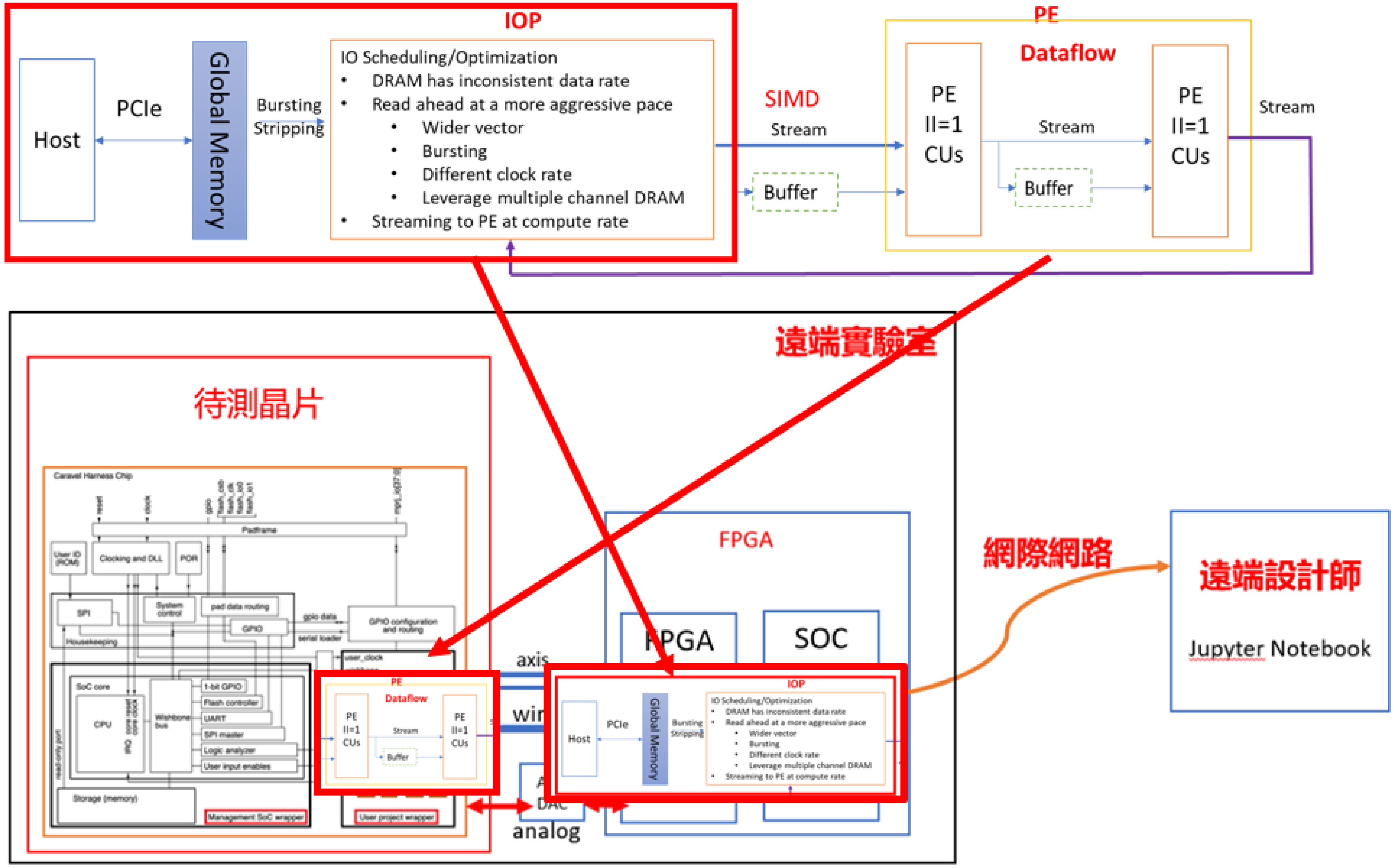
Architecture for Application Accelerator Design



IO Scheduling/Optimization

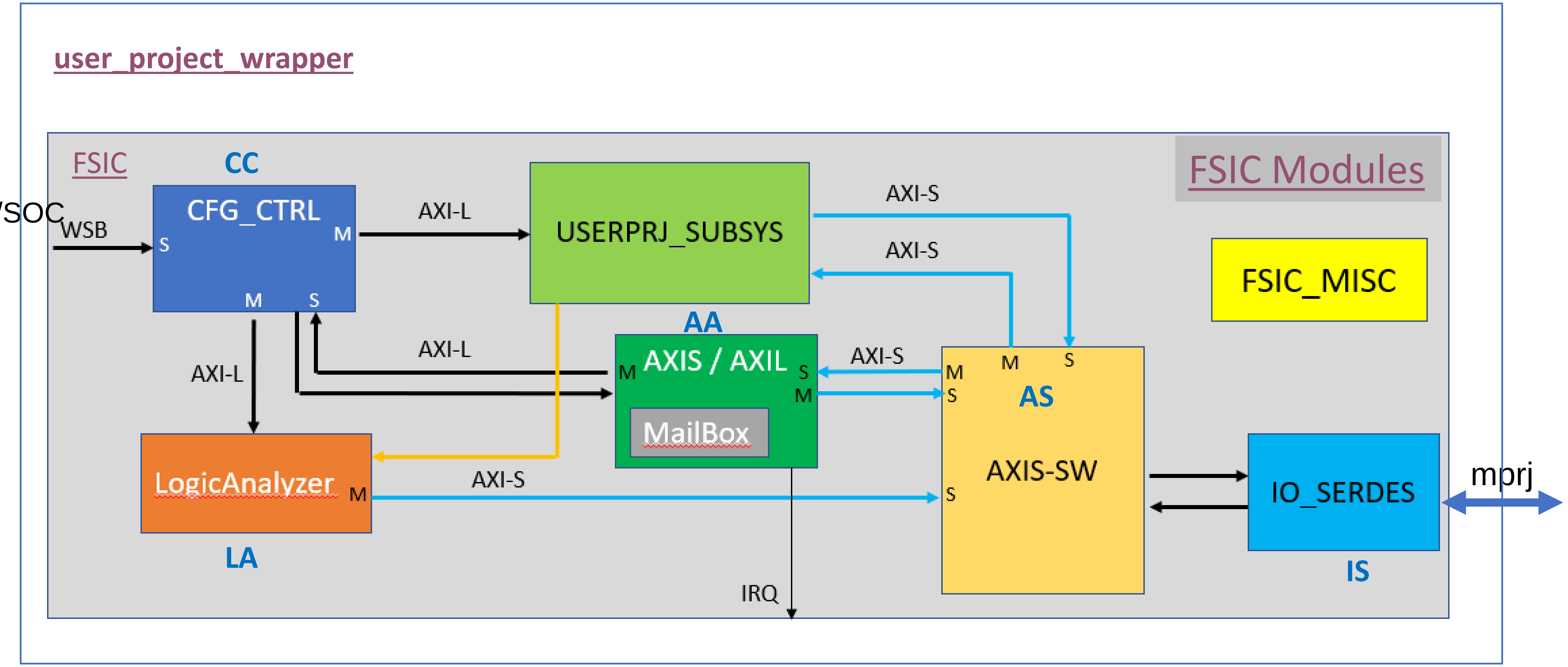
- DRAM has inconsistent data rate
- Read ahead at a more aggressive pace
  - Wider vector
  - Bursting
  - Different clock rate
  - Leverage multiple channel DRAM
- Streaming to PE at compute rate

Embed Application Accelerator in FSIC



User Project Wrapper

From
Caravel/SOC



Design Sources: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user

FSIC-AXIS interface specification

https://hackmd.io/iH10geiITr2WySU_dV3k_Q

FSIC-AXIS interface specification

Downstream:	FPGA	->	Caravel SOC
Upstream :	Caravel SOC	->	FPGA

Extension of Axis specification to include

- Axilite configuration transaction (Axilite Overload)
- Data payload for Axis transaction

Axilite configuration transaction (Axilite Overload)

- Axilite configuration transaction can be Downstream/Upstream.
- Downstream: Remote Host (FPGA/SOC) access configuration registers in FSIC (user logic, mail box, etc) via IO_Serdes.
- Upstream Caravel/RISCV access axilite register (mailbox) in FPGA via IO_Serdes.

Data payload for Axis transaction

- Downstream Data payload is from axis master in remote host to user logic (axis slave) of FSIC.
- Upstream Data payload is from axis master(user logic, Logic Analyzer) of FSIC output data to axis slave in remote host.

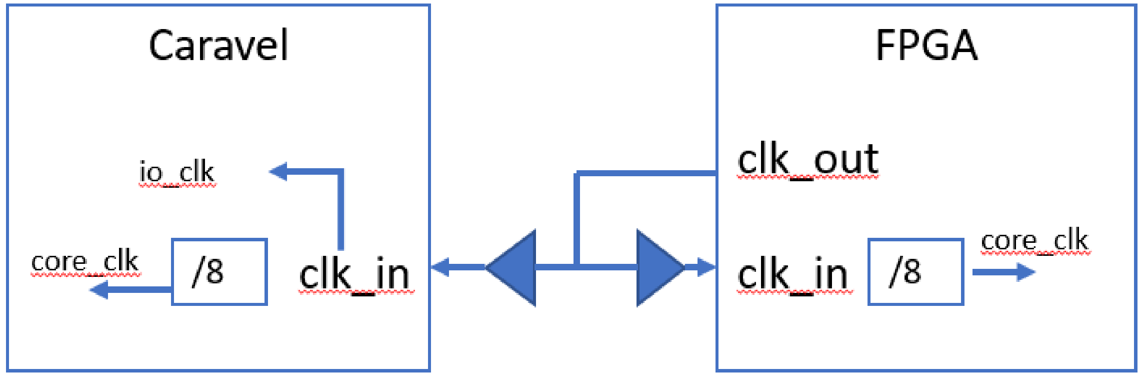
Transaction Table - TUSER<1:0> Definition

TUSER<1:0>	# of T	Transaction Type
00	n	Data payload for axis transaction. Limitation: all User pojects Data payload in axis MUST <= Max_axis_Data_payload(=32)
01	2	Axilite write transaction Address + Data. (TAD) 1st T is the Byte-enable + address, i.e. {BE[3:0],ADDR[27:0]}, 2nd T is the Data[31:0]. Note: Axilite write transaction only support 1T in data phase.
10	1	Axilite read Command (Address Phase). TAD<31:0> is the address ADDR[31:0]
11	1	Axilite read Completion (Data Phase). TAD<31:0> is the return data DATA[31:0]. 1. Axilite read transaction only support 1T in data phase (Axilite read Completion). 2. If Axilite read Command is Upstream then the Axilite read Completion is Downstream, and vice versa.

Routing: TID<1:0> Definition
(used by Axis-switch AS)

Direction	TID[1:0]	Source Module	Destination Module
Downstream	00	User DMA (M_AXIS_MM2S) in remote host (option extended user project)	User Project - the current active user project
Downstream	01	Axilite Master R/W in remote host (include Mail box write)	Axis-Axilite (include Mail box)
Upstream	00	User Project - the current active user project	User DMA (S_AXIS_S2MM) in remote host (option extended user project)
Upstream	01	Axis-Axilite (for Mail box)	Axilite slave in remote host (for mail box write)
Upstream	10	Logic Analyzer	Logic Analyzer data receiver - DMA (S_AXIS_S2MM) in remote host

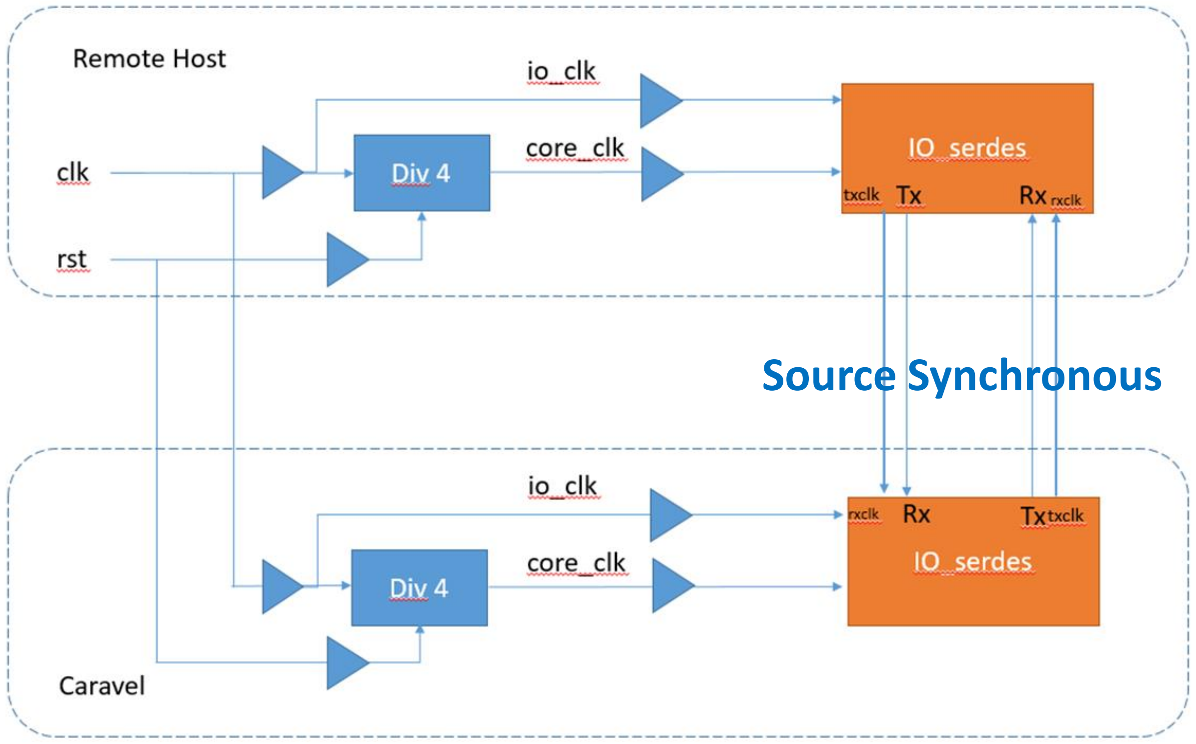
System Clocking Scheme



Synchronous Design

IO_SERDES

Method : with clock from Tx to Rx



https://hackmd.io/eOVkrhPRSJG0HfFP0x6HrQ

Clock Skew Control

The Caravel chip and FPGA runs synchronously both on core_clk and io_clk, i.e. the skew of the core clock separately in Caravel chip and FPGA is controlled, so is the io_clk.

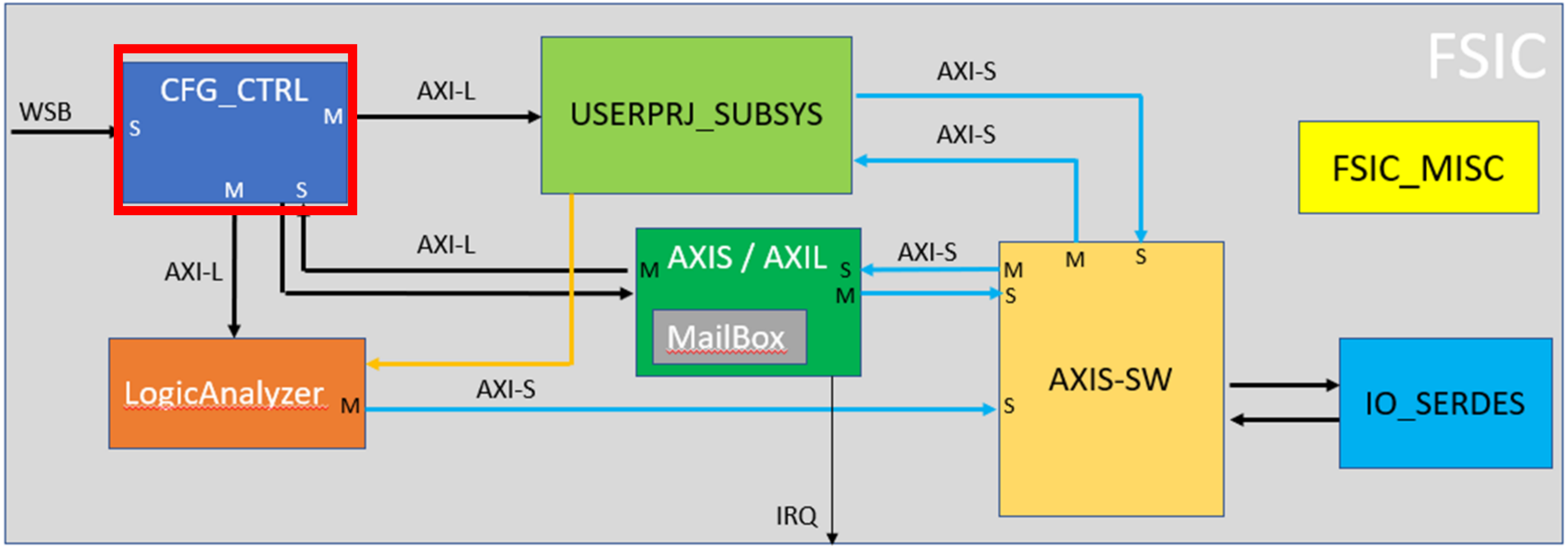
The skew is controlled by board layout, and matched IO buffer delay. Within Caravel and FPGA, the skew between core_clk and io_clk is also controlled by balanced clock tree layout.

A detailed post-layout static timing analysis guarantee there is no hold time violation when signal is passed between core_clk and io_clk. A synchronizer using negative io_clk edge is a method to ensure safe hold time.

System Initialization Sequence

1. FPGA download Caravel flash content from host
2. Reset Exit Sequence : FPGA -> Caravel
    1. FPGA GPIO output (default: assert to reset) to control Caravel reset, and power circuit
    2. After Caravel power-on, FPGA bit-stream download
    3. Caravel reset released after Caravel firmware download
3. FPGA release Caravel reset-pin, and wait for Caravel chip initialization done
4. Caravel chip initialization (By firmware) – Initialize IO_SERDES

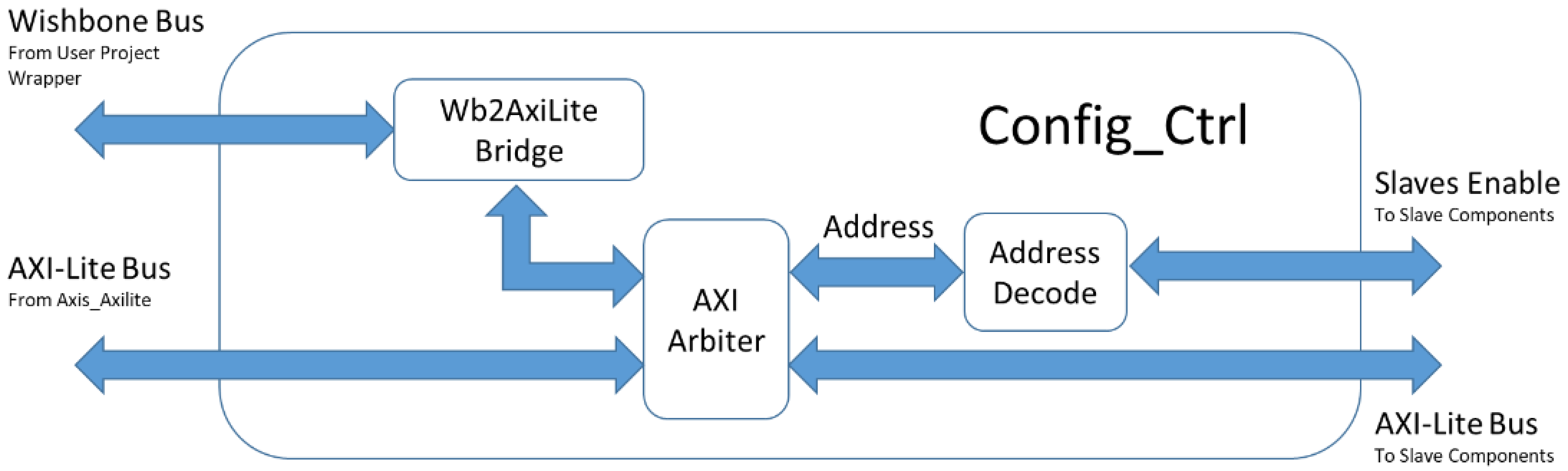
Config_Ctrl (CC)



HackMD: https://hackmd.io/MNIEVTPVSLqNRjcRQWGkgw

Design Source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/config_ctrl/rtl

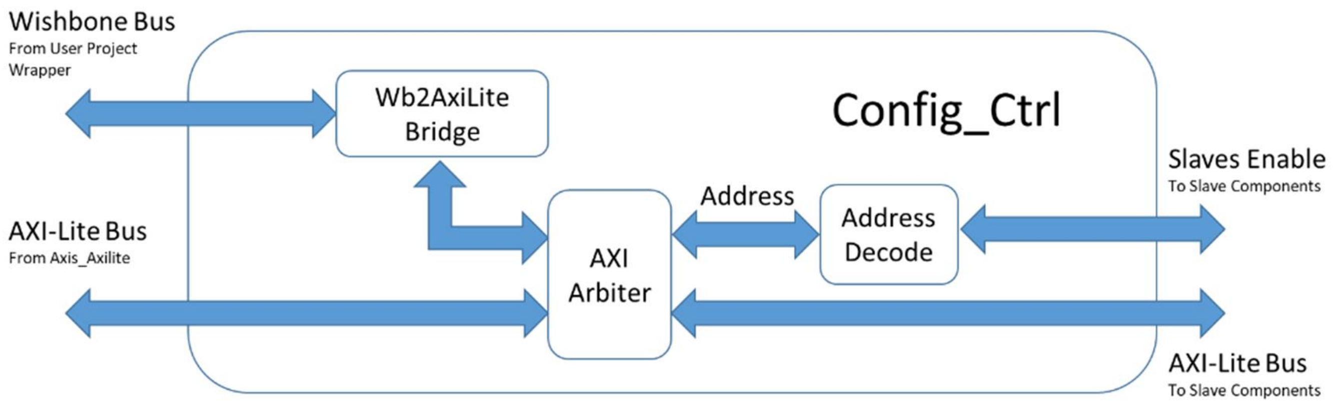
Config_Ctrl (CC)



Hackmd: https://hackmd.io/MNIEVTPVSLqNRjcRQWGkgw

Config_ctrl: (CC) – Configuration Control

- Generate Axilite transaction to configure all modules in user project wrapper
- Function:
  - Wishbond to Axilite Converstion
  - Arbitrate configuration from Caravel and FPGA side
  - Target decoding, and generate cc_target?_enable signal
- Each module has 4K address range 32’h3000_?xxx



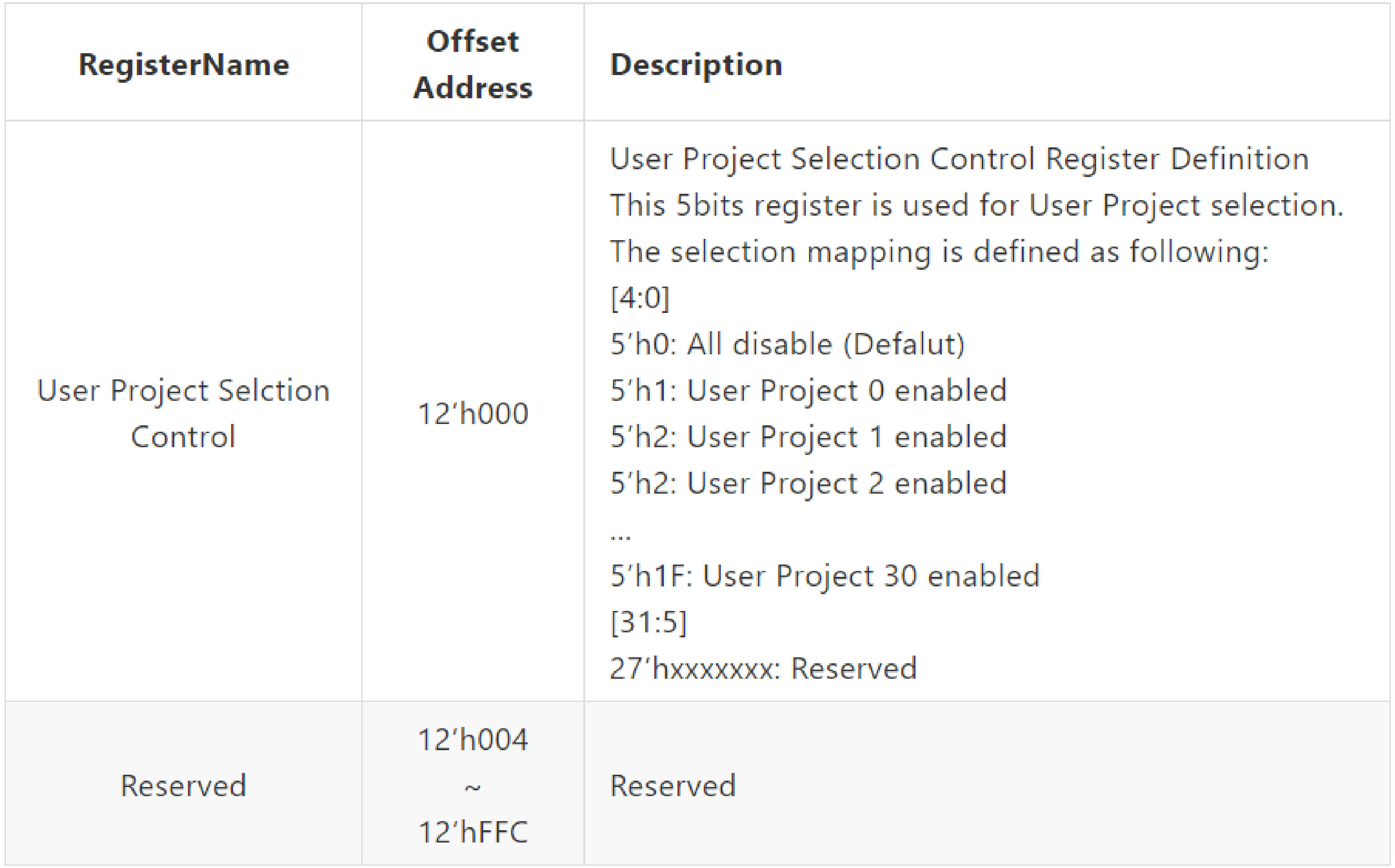
Address Map

Target Module	Address range	Enable signal
User Projects	32’h3000_0xxx	cc_up_enable
Logic Analyzer	32’h3000_1xxx	cc_la_enable
Axis_Axilite	32’h3000_2xxx	cc_aa_enable
IO Serdes	32’h3000_3xxx	cc_is_enable
Axis_Switch	32’h3000_4xxx	cc_as_enable
Config_Control	32’h3000_5xxx	

Mailbox: 15’h2000 – 15’h201F
AA: 15’h2100 – 15’h2107

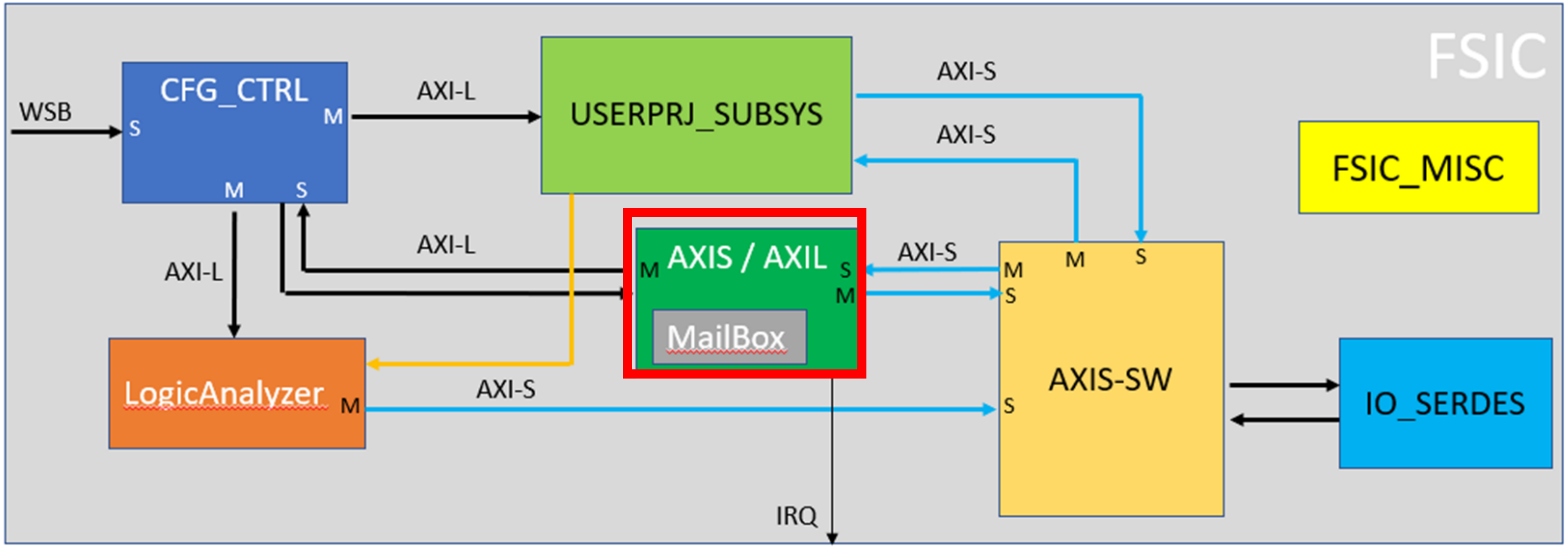
CC – Configuration Register

Configuration Control Group: 32’h3000_5000~32’h30000_5FFF



RegisterName	Offset Address	Description
User Project Selction Control	12’h000	User Project Selection Control Register Definition This 5bits register is used for User Project selection. The selection mapping is defined as following: [4:0] 5’h0: All disable (Defalut) 5’h1: User Project 0 enabled 5’h2: User Project 1 enabled 5’h2: User Project 2 enabled … 5’h1F: User Project 30 enabled [31:5] 27’hxxxxxxx: Reserved
Reserved	12’h004 ~ 12’hFFC	Reserved

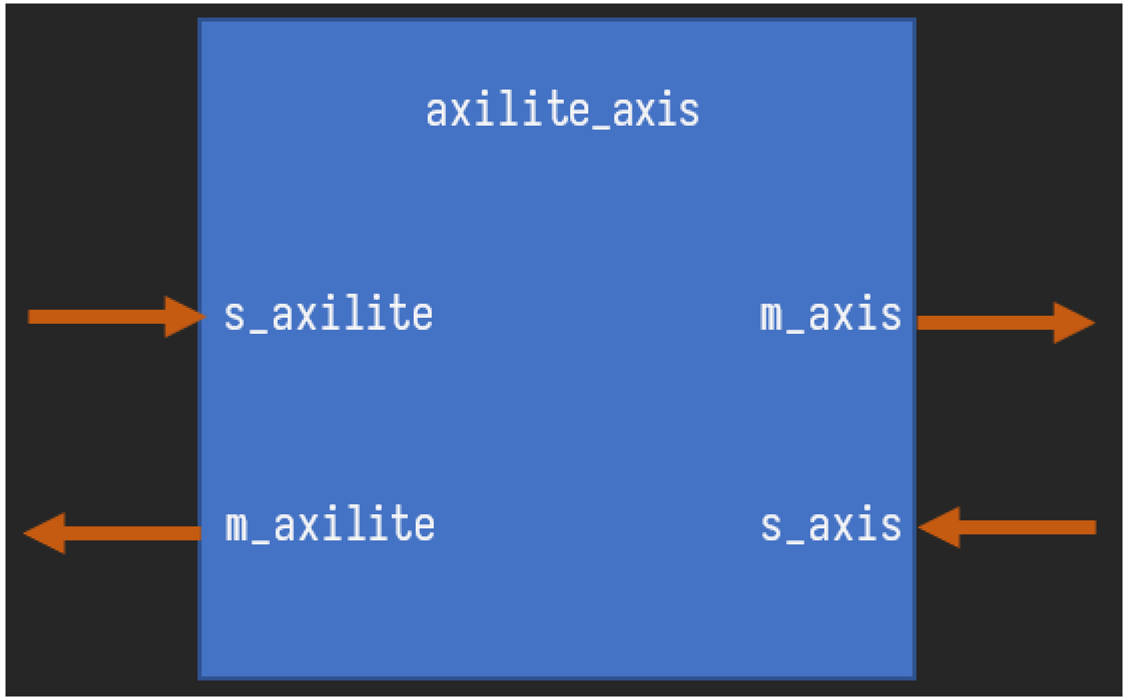
AXILite-AXIS (AA)



HackMD: https://hackmd.io/7m3m0X0vSI2_FoCVdVu1dg

Design Source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/axilite_axis/rtl

Protocol Conversion between AXILite <-> AXIS (SystemVerilog)

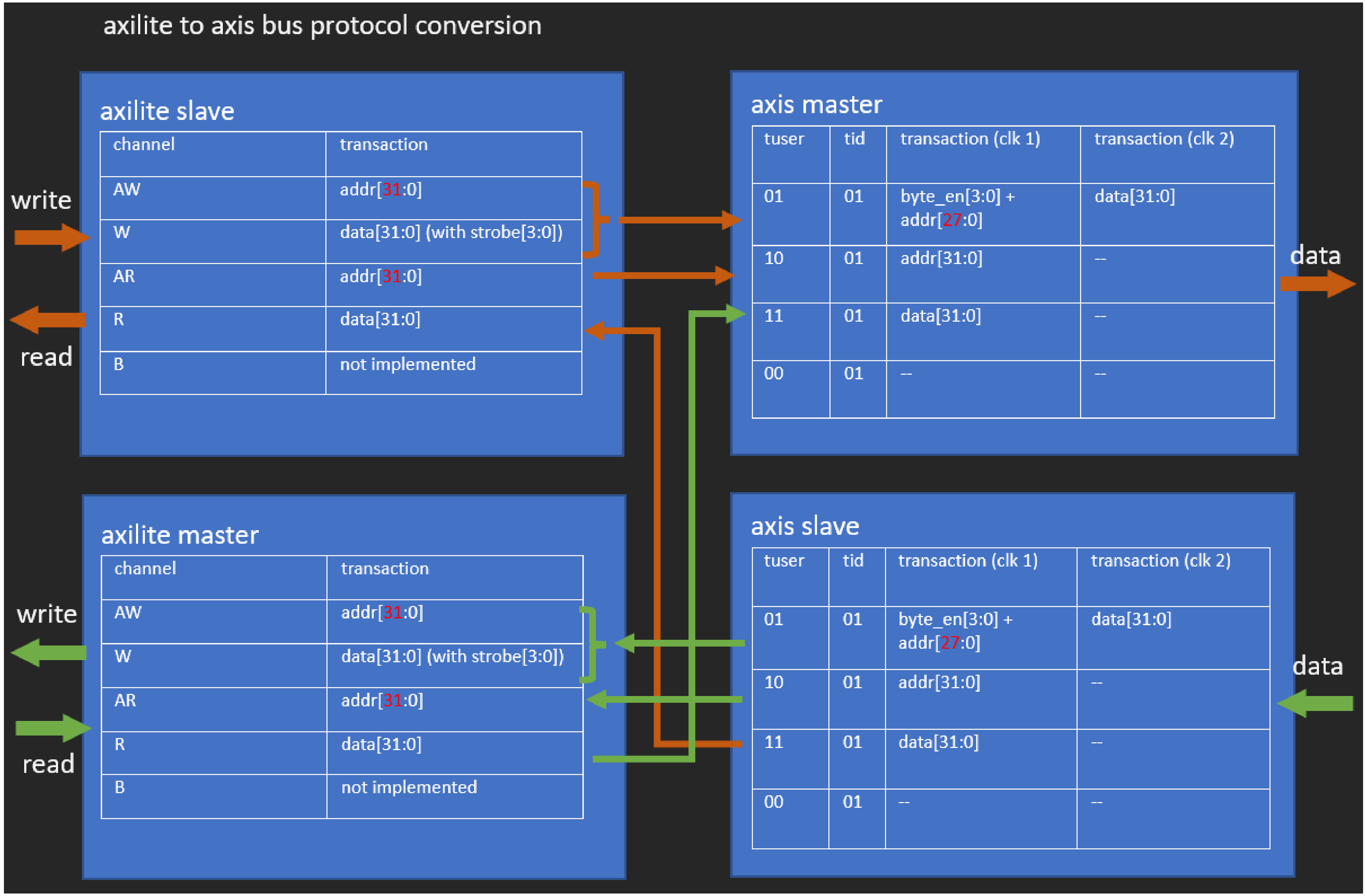


Hackmd: https://hackmd.io/7m3m0X0vSI2_FoCVdVu1dg

Features

1. Convert axilite transaction to modified axis transaction, and convert axis back to axilite.
2. Syncing mailbox in caravel and fpga, can generate interrupt with status if mailbox is written by transaction from other side.
3. Maintain the access of modules in caravel user_wrapper from fpga/ps.
4. In case of the mailbox base address in address map may change at fpga side, this module implements **remap control register / remap addr register** for software configuration.

Bus Protocol Conversion



axilite slave

channel	transaction
AW	addr[31:0]
W	data[31:0] (with strobe[3:0])
AR	addr[31:0]
R	data[31:0]
B	not implemented

axis master

tuser	tid	transaction (clk 1)	transaction (clk 2)
01	01	byte_en[3:0] + addr[27:0]	data[31:0]
10	01	addr[31:0]	--
11	01	data[31:0]	--
00	01	--	--

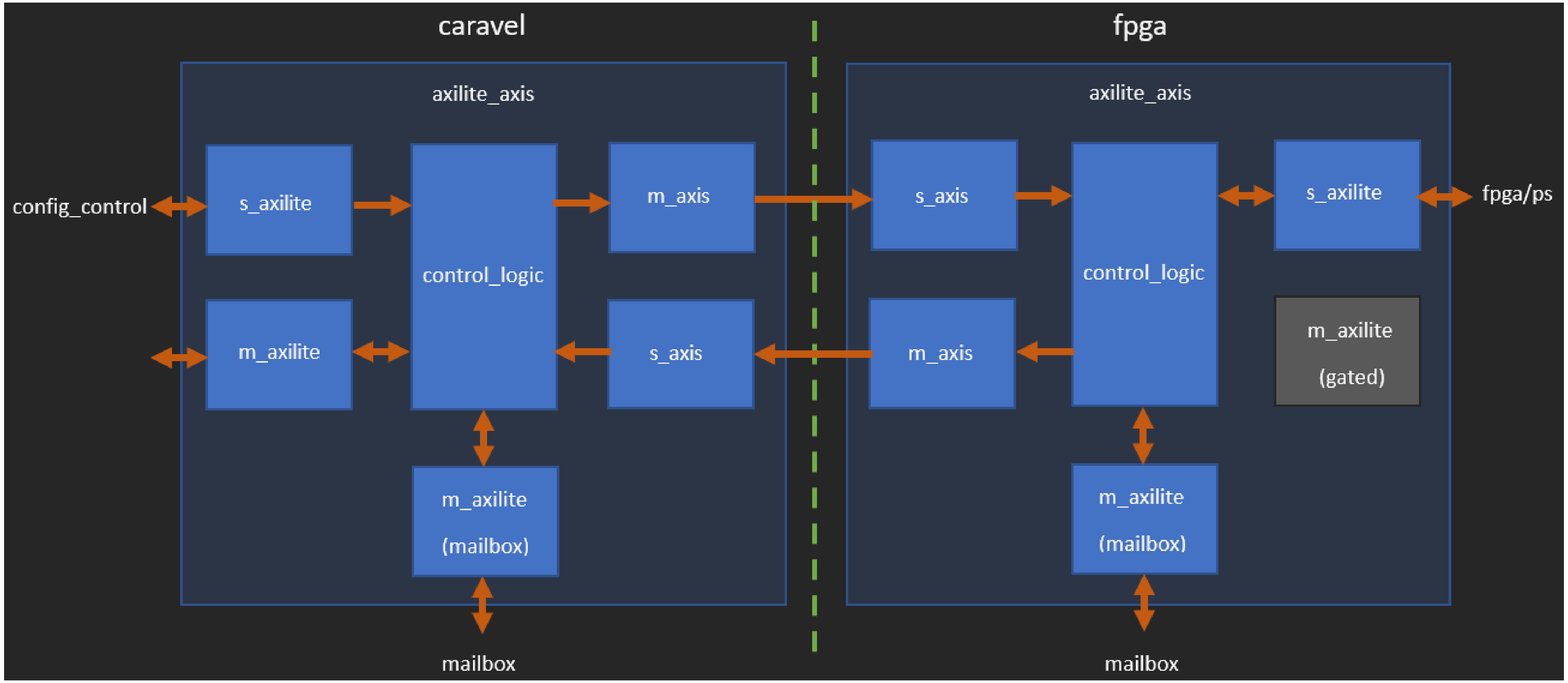
axilite master

channel	transaction
AW	addr[31:0]
W	data[31:0] (with strobe[3:0])
AR	addr[31:0]
R	data[31:0]
B	not implemented

axis slave

tuser	tid	transaction (clk 1)	transaction (clk 2)
01	01	byte_en[3:0] + addr[27:0]	data[31:0]
10	01	addr[31:0]	--
11	01	data[31:0]	--
00	01	--	--

Block Diagram



Design source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/axilite_axis/rtl

```
# addr is the address received by s_axilite, could be from config_control or fpga/PS

if addr in range(0x3000_3000, 0x3000_3FFF): # mailbox address space
        if caravel:
                if read:    # path 1
                        => read caravel mailbox
                elif write: # path 2
                        => write caravel mailbox
                        => write fpga mailbox (raise interrupt in fpga)

        elif fpga:
                fpga_mb_addr = addr
                if remap:
                        fpga_mb_addr += remap_base_addr
                if read:    # path 3
                        => read fpga mailbox using fpga_mb_addr
                elif write: # path 4
                        => write fpga mailbox using fpga_mb_addr
                        => write caravel mailbox using addr (raise interrupt in caravel)

elif addr in range(0x3000_0000, 0x3FFF_FFFF): # caravel address space
        if caravel:
                => NOP (handled by config_control)

        elif fpga:
                if read:    # path 5
                        => read caravel
                elif write: # path 6
                        => write caravel (no interrupt in caravel)
```

Transaction Table - TUSER<1:0> Definitiion

TUSER is used to distinguish different transaction types.

TUSER<1:0>	# of T	Transaction Type
00	n	Data payload for axis transaction. Limitation: all User pojects Data payload in asix MUST <= Max_axis_Data_payload(=32)
01	2	Axilite write transaction Address + Data. TAD 1^st^ T is the Byte-enable + address, i.e. {BE[3:0],ADDR[27:0]}, 2^nd^ T is the Data[31:0]. Note: Axilite write transaction only support 1T in data phase.
10	1	Axilite read Command (Address Phase). TAD<31:0> is the address ADDR[31:0]
11	1	Axilite read Completion (Data Phase). TAD<31:0> is the return data DATA[31:0]. Note: 1. Axilite read transaction only support 1T in data phase(Axilite read Completion). 2. If Axilite read Command is Upstream then the Axilite read Completion is Downstream, and vice versa.

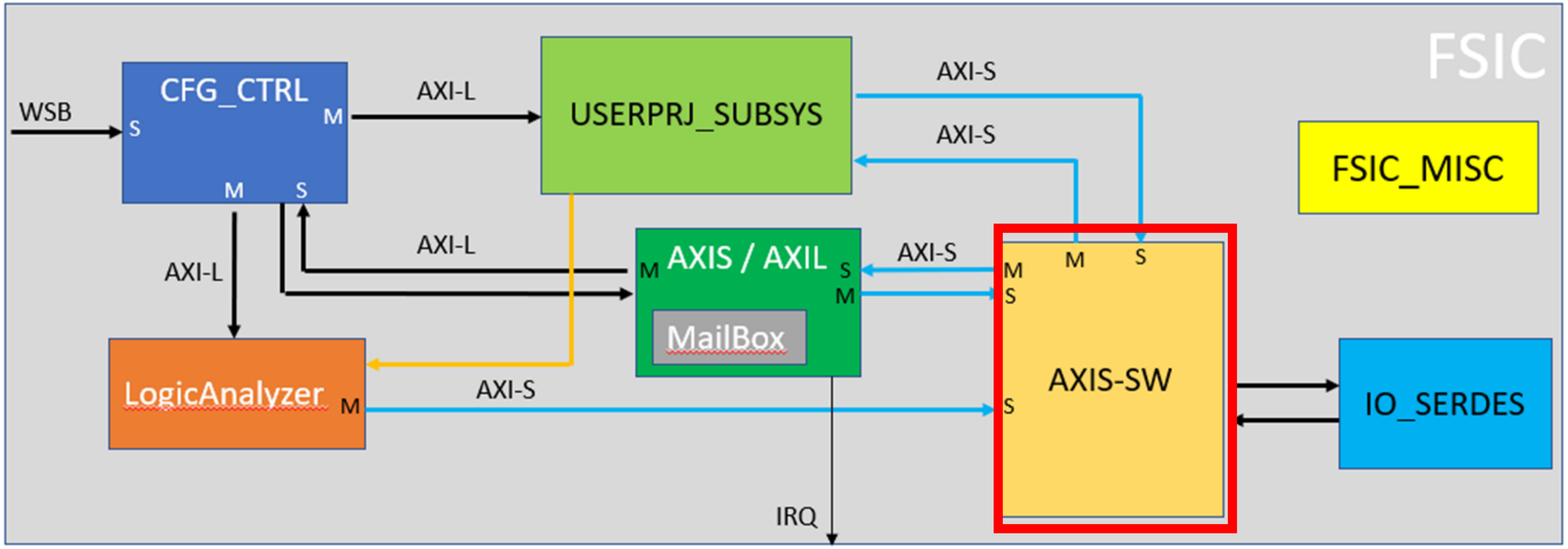
Register Map

RegisterName	Offset Address	Description
remap_enable	'h0	enable remap at fpga side, this module will send address plus base address offset in **remap addr register** 1’b0: (default) remap disabled 1’b1: remap enabled
remap_base_addr	'h0	base address for address map at fpga side. 32’h0: (default)
interrupt_enable	'h0	enable interrupt if write transaction from other side send to mailbox 1’b0: (default) disabled 1’b1: interrupt enabled
mb_wr	'h0	(write one clear) mailbox is written by the transaction from other side. 1’b0 (default): no other side write transaction 1’b1: mailbox is written

Reference for SystemVerilog Testbench

- https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/axilite_axis/testbench

AXI-SWITCH (AS)

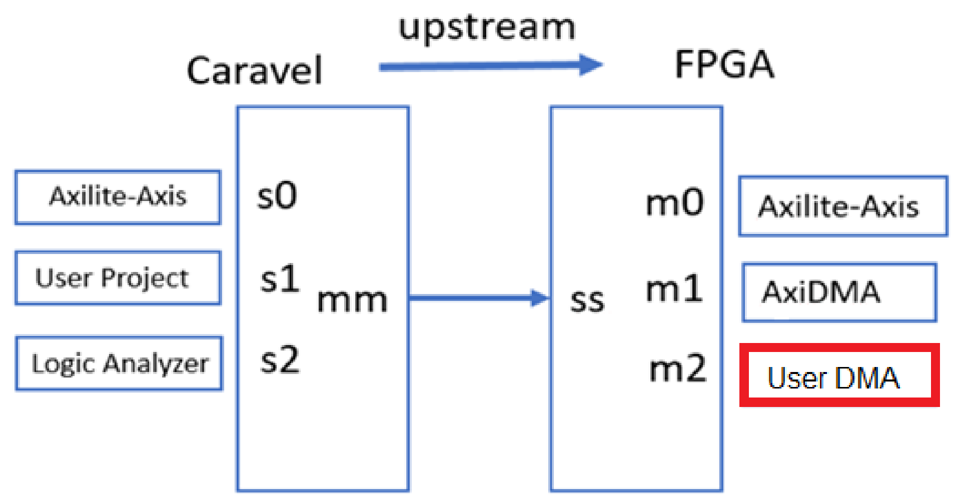


- HackMD: https://hackmd.io/YtuWTM78T1aNogiuNzM80A
- Design Source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/axis_switch/rtl

Path & Port Definitions

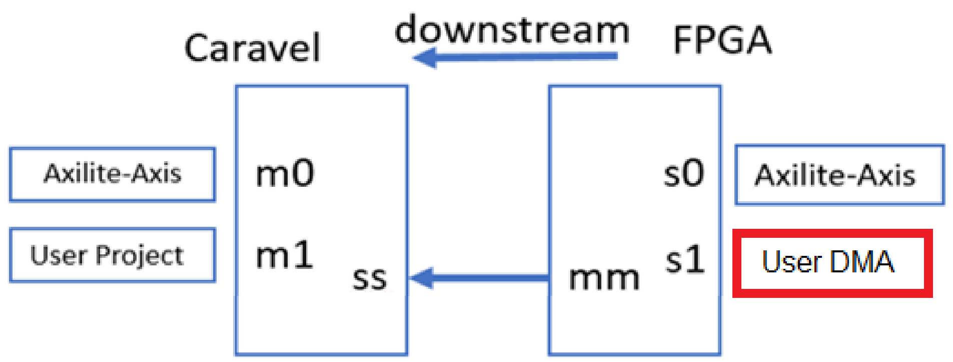
Data flow directions of AXI stream.

- **Upstream**: data flows from Caravel to FPGA/Memory
- **Downstream**: data flows from FPGA/Memory to Caravel

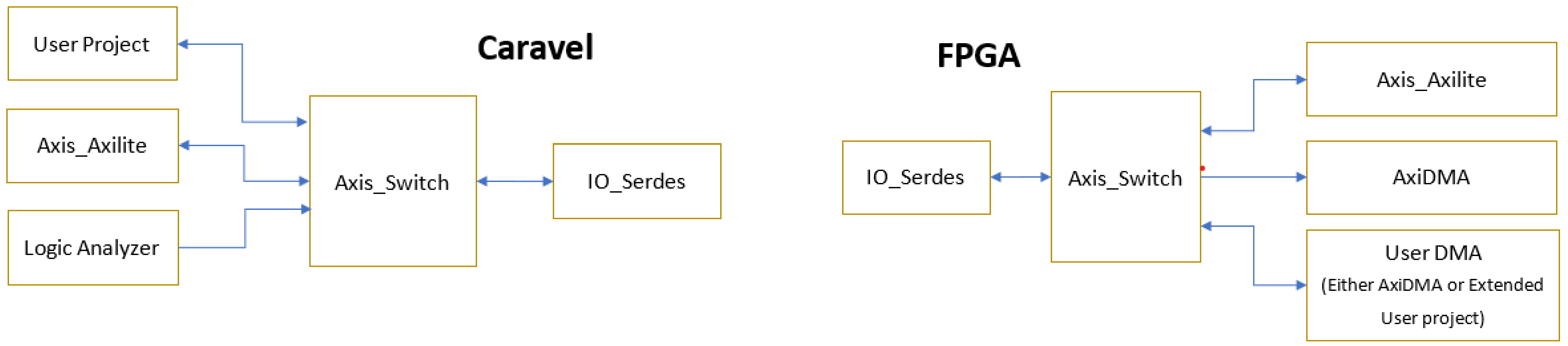


Data producers and consumers include

- **User Project** (in Caravel)
- Extended User Project (in FPGA)
- Logic Analyzer (in Caravel)
- Axilite-Axis (in both Caravel, FPGA)
- AxiDMA (in FPGA)



Interface Blocks



TID Definition

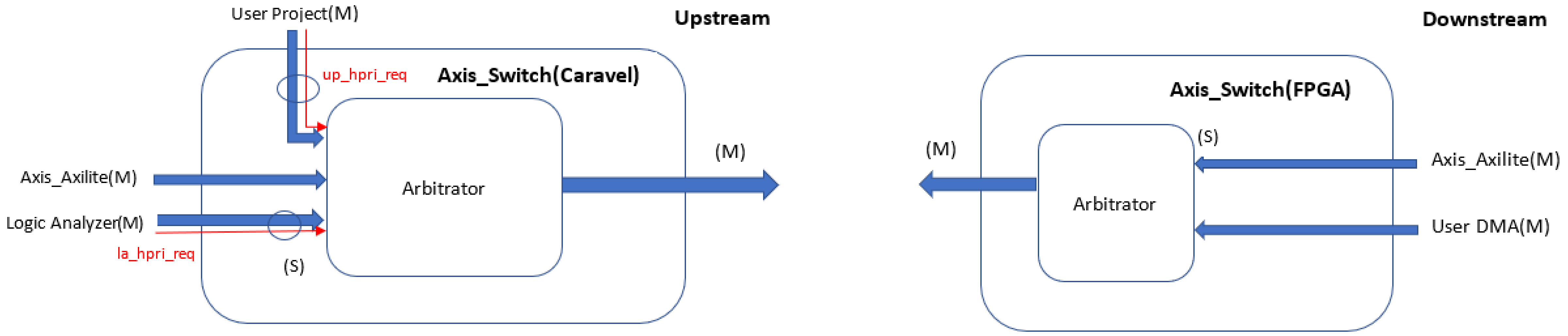
Direction	TID[1:0]	Source Module	Destination Module
Downstream	00	User DMA (M_AXIS_MM2S) in remote host (option extended user project)	User Project - the current active user project
Downstream	01	Axilite Master R/W in remote host (include Mail box write)	Axis-Axilite (include Mail box)
Upstream	00	User Project - the current active user project	User DMA (S_AXIS_S2MM) in remote host (option extended user project)
Upstream	01	Axis-Axilite (for Mail box)	Axilite slave in remote host (for mail box write)
Upstream	10	Logic Analyzer	Logic Analyzer data receiver - DMA (S_AXIS_S2MM) in remote host

Configuration Register Definition

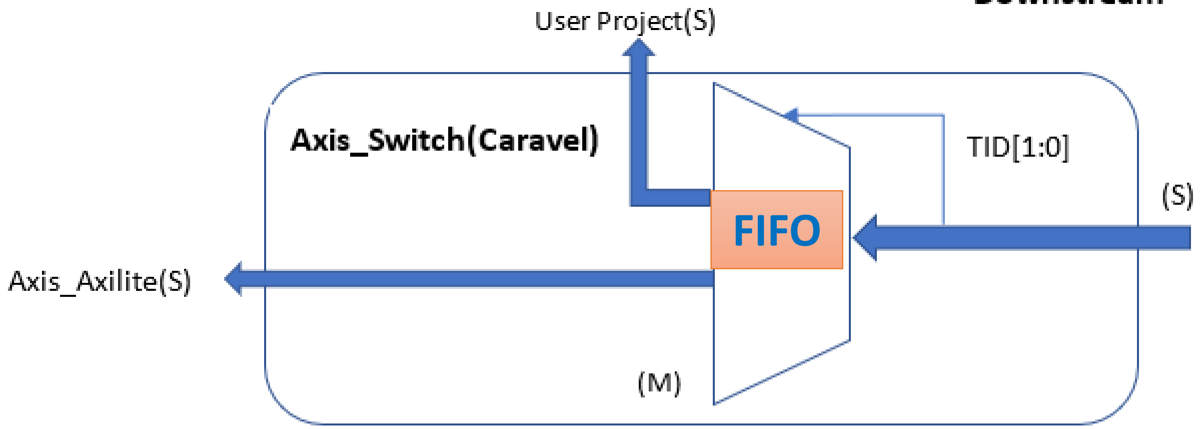
FIFO Threshold Register

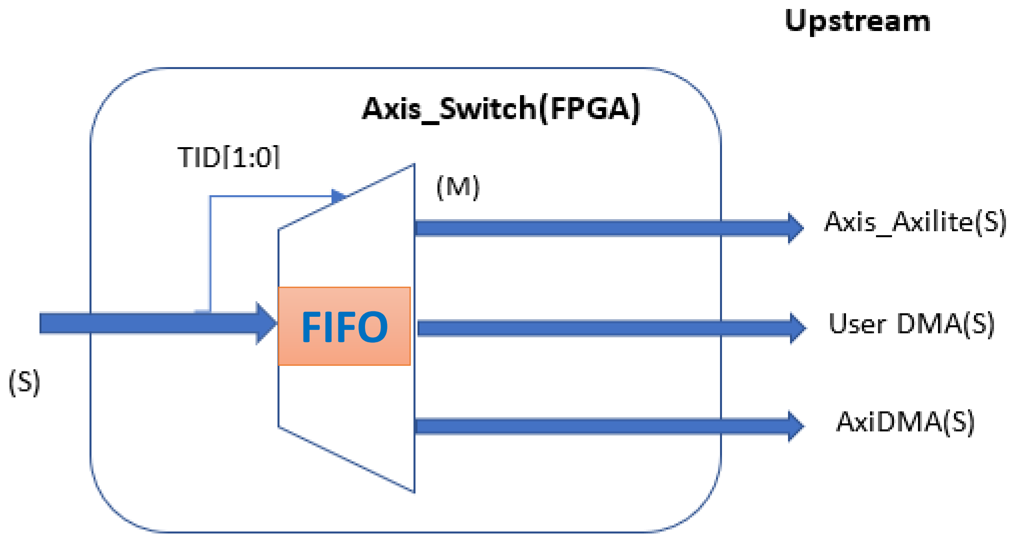
RegisterName	Offset Address	Description
FIFO Threshold	'h0	Demux FIFO Threshold Control Register This 4-bits register is used for as_is_axis_tready de-assertion when current free slot of FIFO is less than this register

Round-Robin Arbitration - MUX



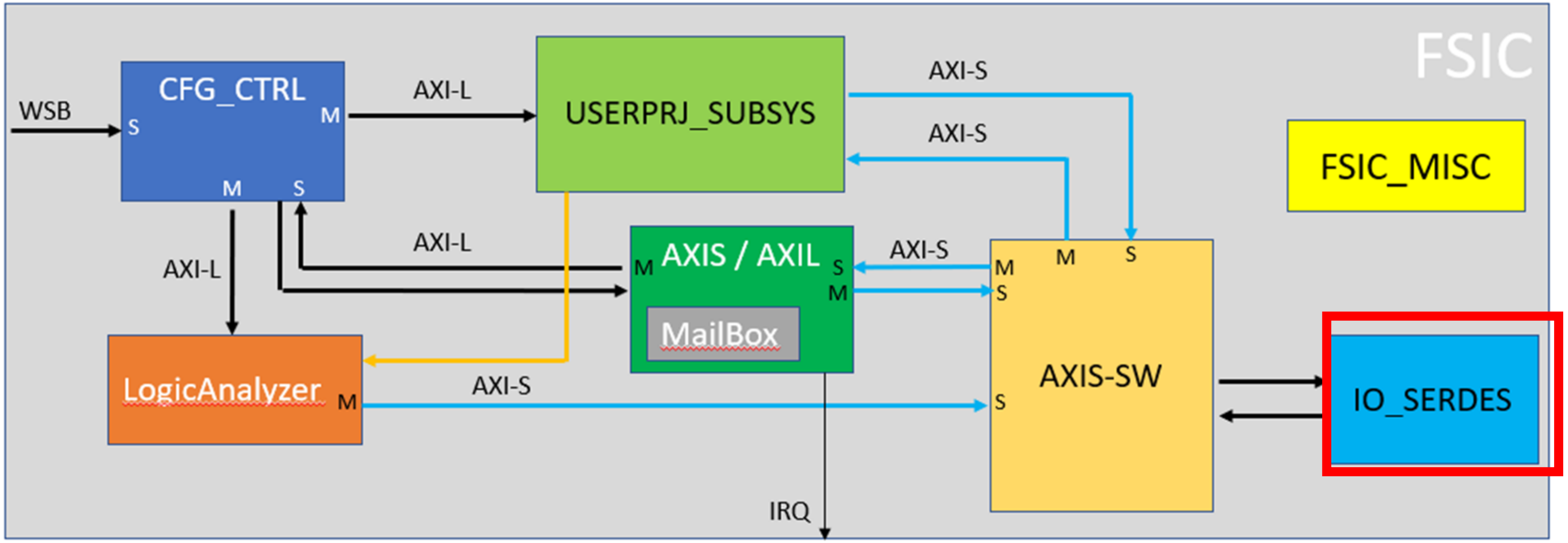
Demultiplexer with TID





FIFO depth = 16

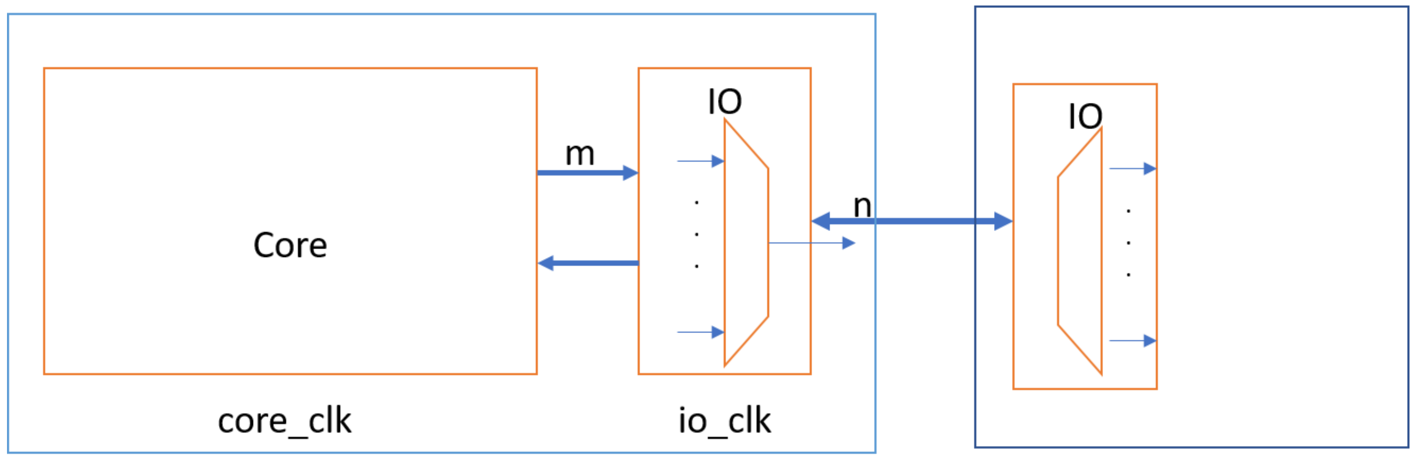
IO_SERDES (IS)



- HackMD: https://hackmd.io/4Kd9drL4QwWJwhFshsFSRQ
- Design Source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/io_serdes/rtl

Function

- The purpose of this module is to virtually increase the number of IO pins by ratioing the core clock and io clock. In the following diagram, there are m* core signals to IO, and there are n * io pins. To match its throughput, it needs to meet the equation, **m x core_clk = n x io_clk**.



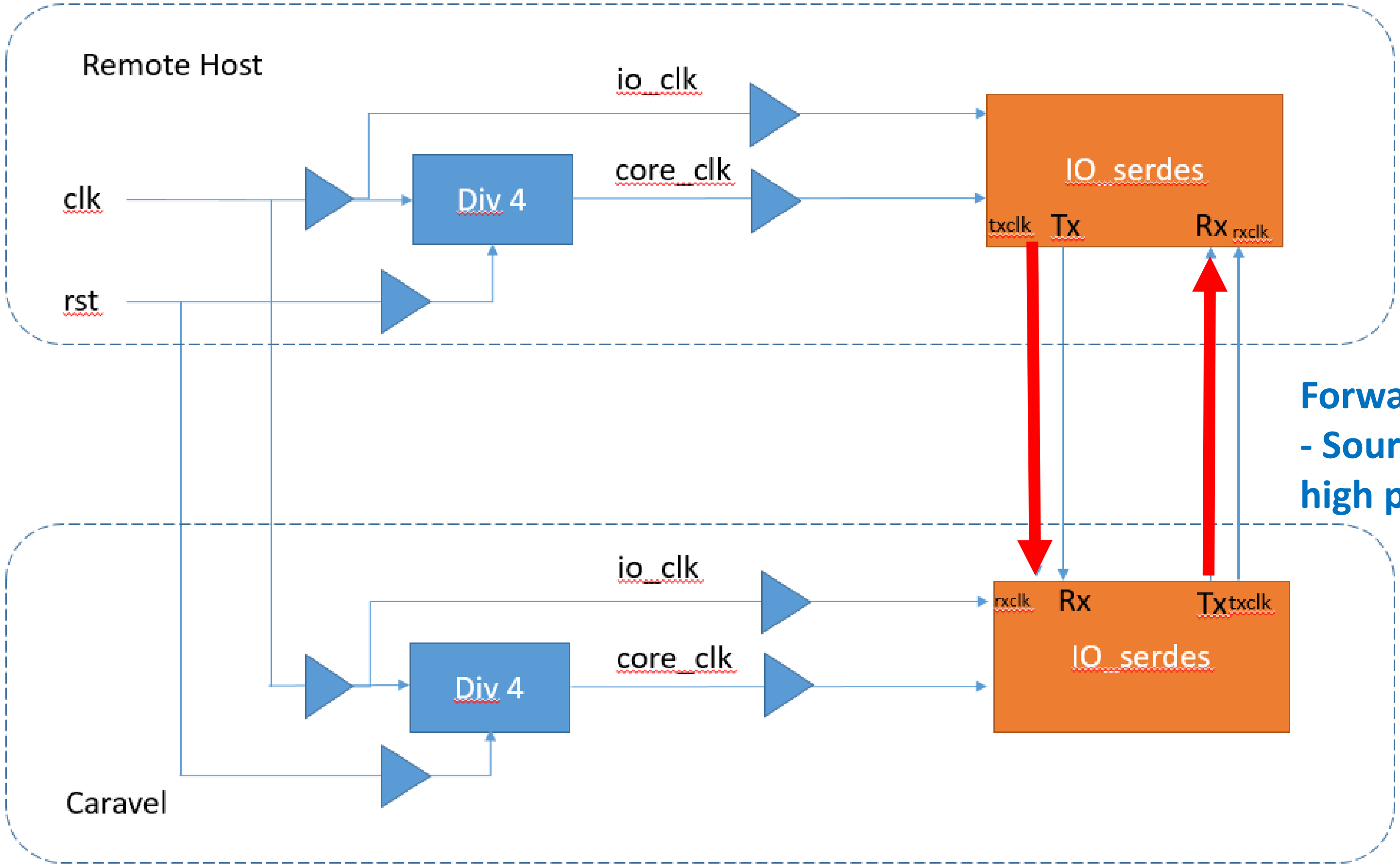
m * core_clk = n * io_clk

Clock and Reset

3 clocks

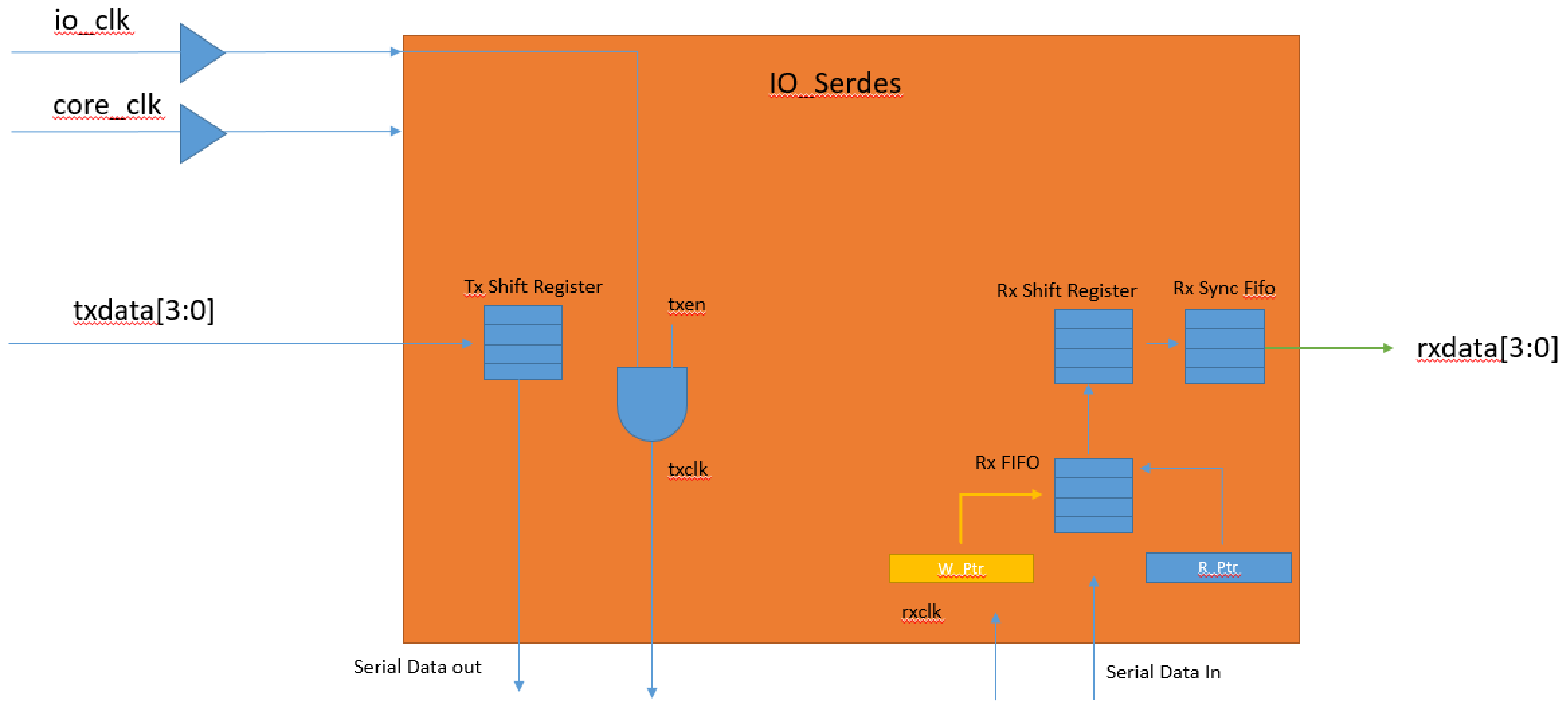
- **rxclk/txclk (forwarding clock)**
- **io_clock**
- **core_clock**

Method : with clock from Tx to Rx



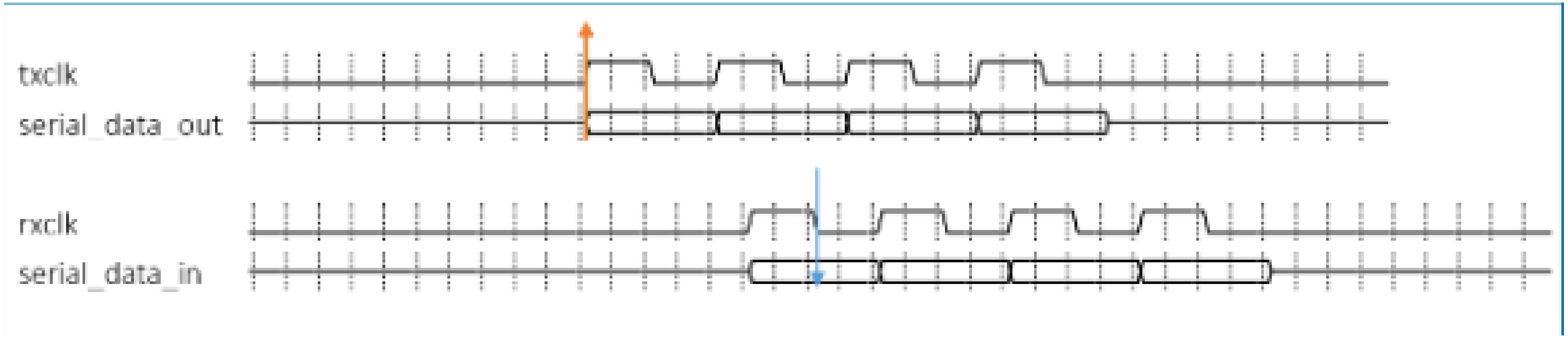
Forwarding clock
- Source Synchronous scheme for high performance

Block Diagram



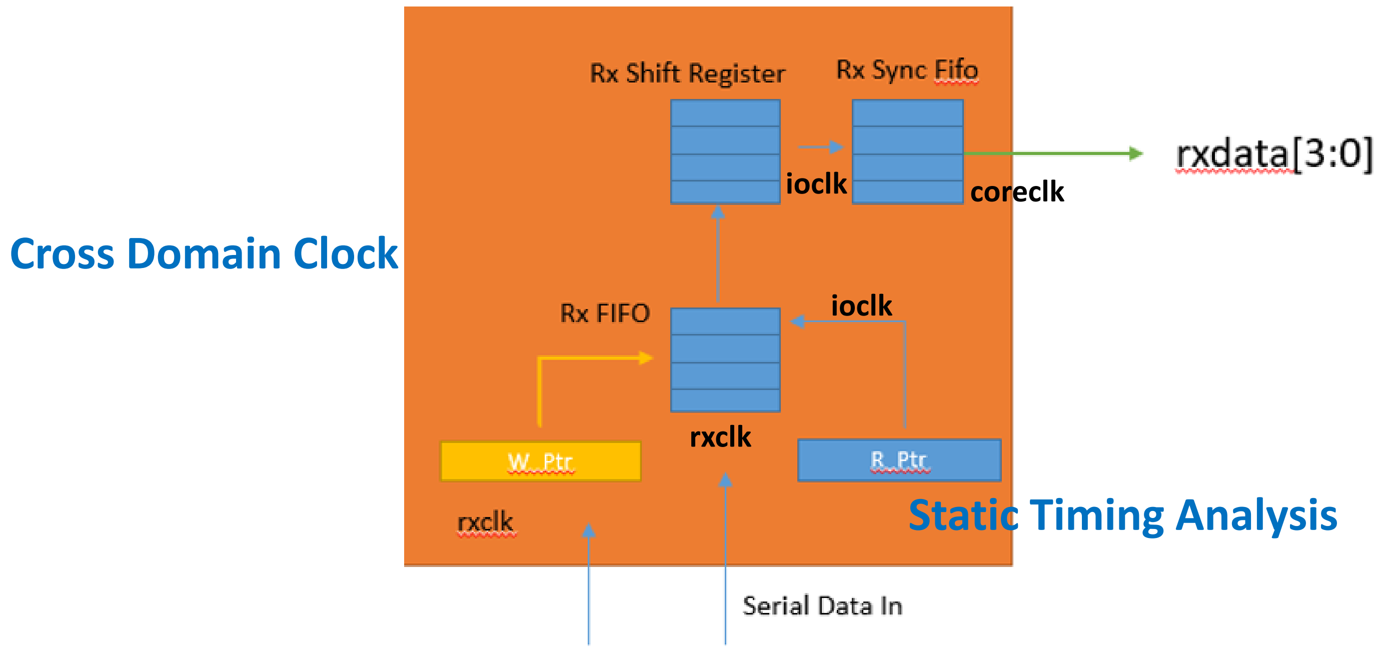
Transmit / Receive Timing

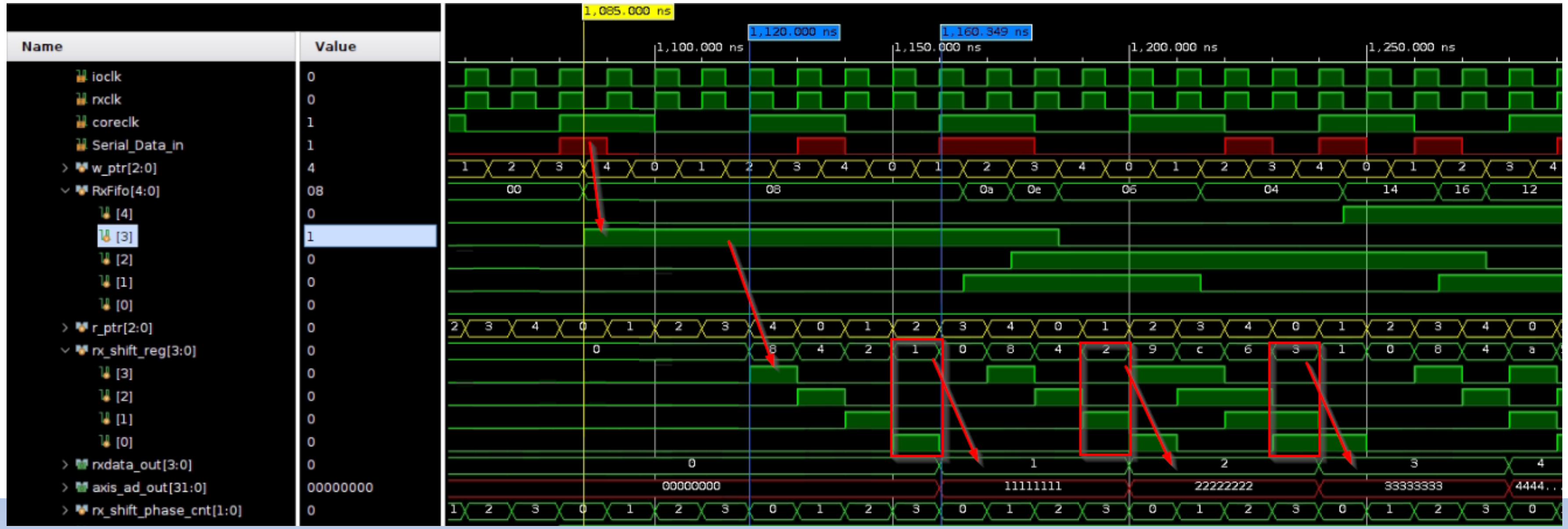
- Tx side output serial_data align with txclk rising edge.
- Rx side use rxclk negative edge to sample the serial_data



Serial-In, Parallel-Out

- Serial_Data_in to RxFIFO – use w_ptr
- RxFIFO to rx_shift_reg – use r_ptr
- rx_shift_reg to rx_data_out
  - move when rx_shift_phase_cnt = 3
- Static Timing Analysis issues
  - Cross-clock domain - rxclk -> core_clk
  - Source Synchronous - rxclk & data





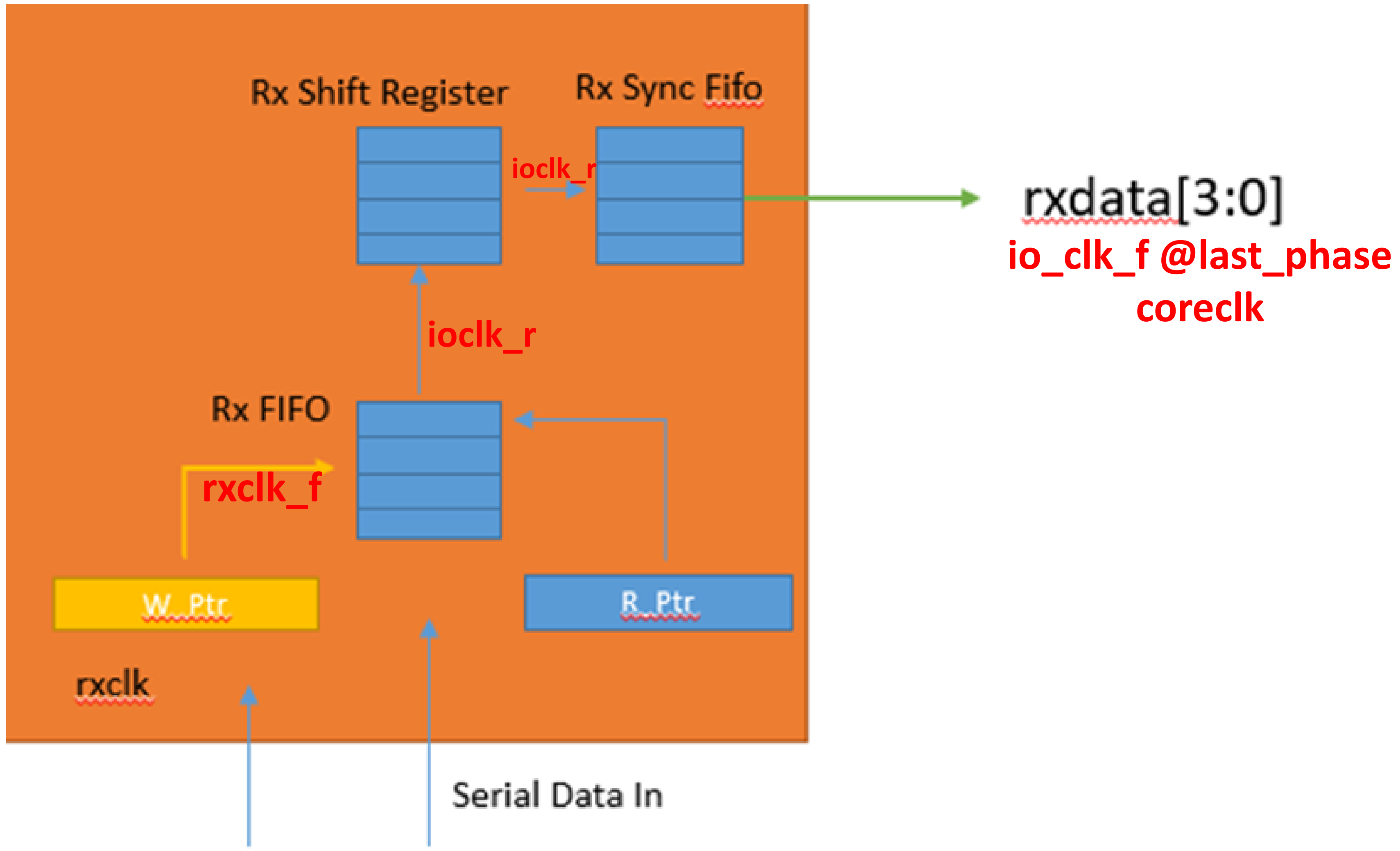
Name	Value
ioclk	0
rxclk	0
coreclk	1
Serial_Data_in	1
w_ptr[2:0]	4
RxFifo[4:0]	08
[4]	0
[3]	1
[2]	0
[1]	0
[0]	0
r_ptr[2:0]	0
rx_shift_reg[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
rxdata_out[3:0]	0
axis_ad_out[31:0]	00000000
rx_shift_phase_cnt[1:0]	0

Initialization (Enable RX first, then, enable TX)

- Step 1. system power on and reset de-assert
- Step 2. rxen = 1 within 100ms after reset de-assert
    - rxen for avoid glitch issue when GPIO setting changed.
- Step 3. Tx side provide txclk with data when txen =1.
    - Must wait enough time for Rx side reset de-assert and rxen=1
    - Use negedge to generate txen to avoid glitch in txclk.
    - For example : Tx side wait 200 ms after reset de-assert then set txen =1.
- Step 4. Rx side read Serial_data_in as below
    - Step 4-1. Use rxclk to sample Serial_data_in to RxFIFO(deep=5) by w_ptr
        - Need 2 filp-flops for avoid metastable issue when read w_ptr (Gray code).
    - Step 4-2. Use ioclk to read RxFIFO and push to Rx_Shift_Reg
    - Step 4-3. Use ioclk to move Rx_Shift_Reg to Rx_Sync_FIFO when Rx_Shift_Reg_Valid (4 bits data ready in Rx_Shift_Reg)
    - Step 4-4. Use coreclk to get Rx_Sync_FIFO when Rx_Sync_FIFO_Valid

Txen is software program -> after mprj program

Clock Domain Crossing



Configuration Registers

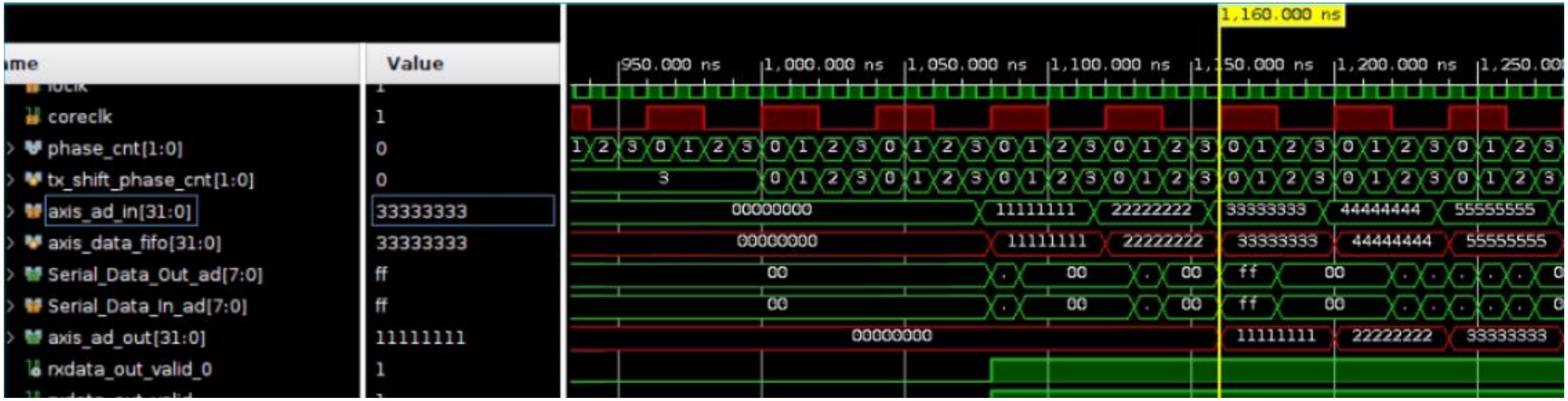
RegisterName	Offset Address	Description
IO Serdes Control	'h0	Control Register block Definition **bit 0: rxen_ctl** FSIC FW set this bit to 1 when GPIO config setting is done for io serdes. set this bit within 100ms after reset de-assert **bit 1: txen_ctl** FSIC FW set this bit to 1 when remote side had enable rx. set this bit after 200ms after reset de-assert
Status	'h4	Status register block definition NA

Programming Guide

- SOC side
  - step 1. init mprj_io setting for io serdes
  - step 2. set rxen_ctl=1
  - step 3. wait 100 ms
  - step 4. set txen_ctl=1
- FPGA side
  - step 1. Do some init for FPGA
  - step 2. set rxen_ctl=1
  - step 3. wait 100 ms
  - step 4. set txen_ctl=1

Design Issues: Flow Control by axis_switch

- issue: the ready signal from local side need 2T to observed in remote side
  - in below figure the data in axis_data_fifo in local side take 2T to axis_ad_out in remote side
- axis_switch need implement a buffer, base on the thresh hold (for example, available slot in buffer <= 2) to generate ready signal to remote side.
- Use **as_is_tready, is_as_tready** to do flow control



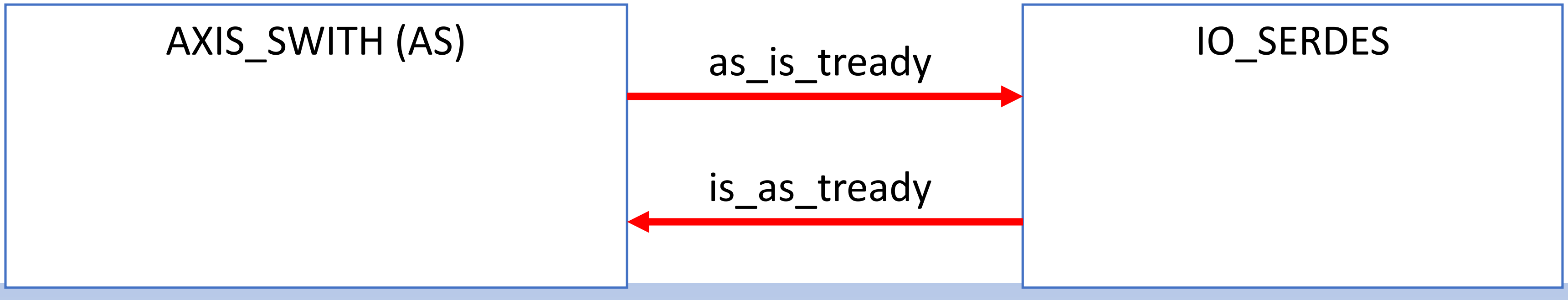
me	Value
coreclk	1
phase_cnt[1:0]	0
tx_shift_phase_cnt[1:0]	0
axis_ad_in[31:0]	33333333
axis_data_fifo[31:0]	33333333
Serial_Data_Out_ad[7:0]	ff
Serial_Data_In_ad[7:0]	ff
axis_ad_out[31:0]	11111111
rxdata_out_valid_0	1

FIFO

as_is_tready

as_is_tready: when local side axis switch Rxfifo size <= threshold then as_is_tready=0, this flow control mechanism is for notify remote side do not provide data with is_as_tvalid=1

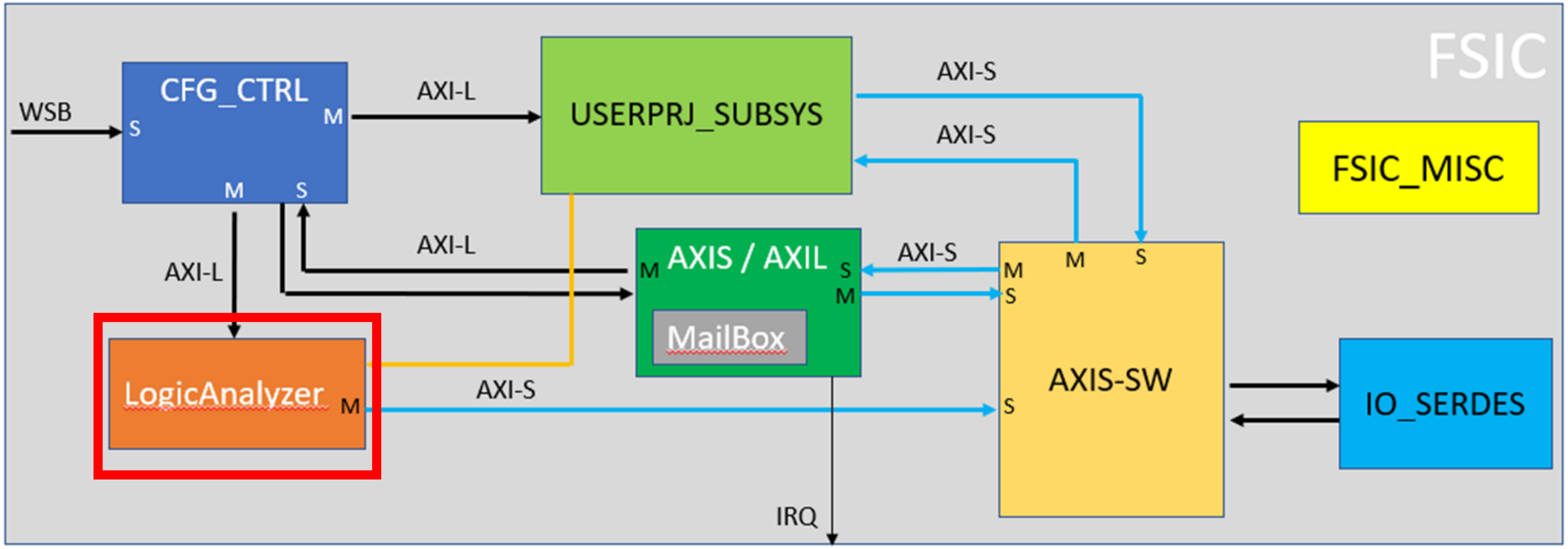
is_as_tready: when remote side axis switch Rxfifo size <= threshold then is_as_tready=0, this flow control mechanism is for notify local side do not provide data with as_is_tvalid=1



Design Consideration

- MPRJ_IO PIN PLANNING Usage (pSERIALIO_WIDTH=13)
  - RX : [20: 8]   Input   RXD   ; [21]   Input   RXCLK
  - TX: [34:22] Output TXD ; [35] Output   TXCLK
  - [36] Input   IO_CLK
- Skew - Io_clock CLK & Core_clock within SOC and FPGA
  - Control Clock-Tree Synthesis
- Skew – rxclk & RXD
  - Control the skew
- Skew – IOCLK to SOC and FPGA

LOGIC_ANALYZER (LA)



- HackMD: https://hackmd.io/DHmJkokLTKq3TNaF4midjA
- Design Source: https://github.com/bol-edu/fsic_fpga/tree/main/rtl/user/logic_analyzer/rtl

Function

1. Monitor signals provided by user project. Support up to 24 monitoring signals
2. Support signal conditioning to trigger signal logging (Currently, done by host program)
3. Compress (Waveform compression, e.g. Run-Length-Encoding RLE) the logged signals and sent them to remote users using the Axis port. Waveform can be displayed in remote environments.
   - Waveform log is designed to be saved in .vcd file format, it can be open by gtkwave.

Note: Use Waveform compression (Run-Length-Encoding RLE) with customized coding scheme to save data bandwidth and internal data buffers

Note: The LogicAnalyzer function is different from the Caravel LogicAnalyzer function (signals: la_xx). In Caravel LogicAnzlyzer signals are used for general-purpose gpio signals controlled by RISC-V.

Packet Format

Packet size is 32bits = {RC[7:0],Data[23:0]}

- RC(Repeat count)：Start at 1 and ends at 255.
  - If RC exceed 255, the packet is released. A new count starts.
- Packet = 32'b0, indicate FIFO overflow because RC start at 1.
- Maximum transfer period

  8 bit = 256 bits count

  If clock is 100MHz

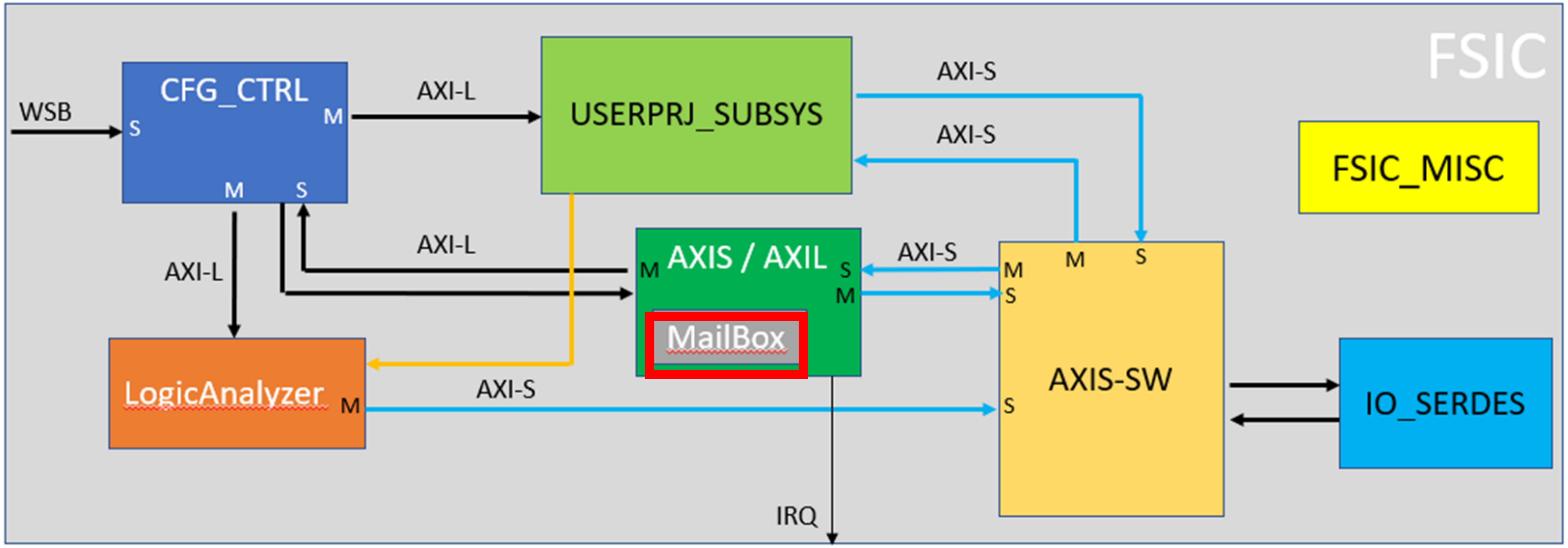
  10ns*256=2.5us/packet

Configuration Register

RegisterName	Offset Address	Description
ctrl	'h00	bit [23:0] - The bitmap maps to up_la_data[23:0]. The 1’b1 means the corresponding up_la_data signal will be monitored. Logic Analyzer signal monitoring is disabled if all bit are 0. Default is 24’h000000.
la_hprj_high_th	'h04	bit[6:0] - Threshold to enable la_hpri_req Default is 7’h40
la_hprj_low_th	'h08	bit[6:0] - Threshold to disable la_hpri_req Default is 7’h10
axis_pkt_len	'h0c	bit[6:0] - A group of bytes that are transported together across an AXI-Stream interface. Default is 7’h08

MAIL_BOX (MB)

Exchange message between Caravel SOC and FPGA



Note:
There is no MailBox module. It is simply registers embedded in axi_ctrl_logic.sv
logic [31:0] mb_regs [7:0]; // 32bit * 8

- HackMD: https://hackmd.io/m-NvQqieQVy0AWxUgM3y7g
- Design Source: https://github.com/bol-edu/fsic_fpga/blob/main/rtl/user/axilite_axis/rtl/axi_ctrl_logic.sv

Functions

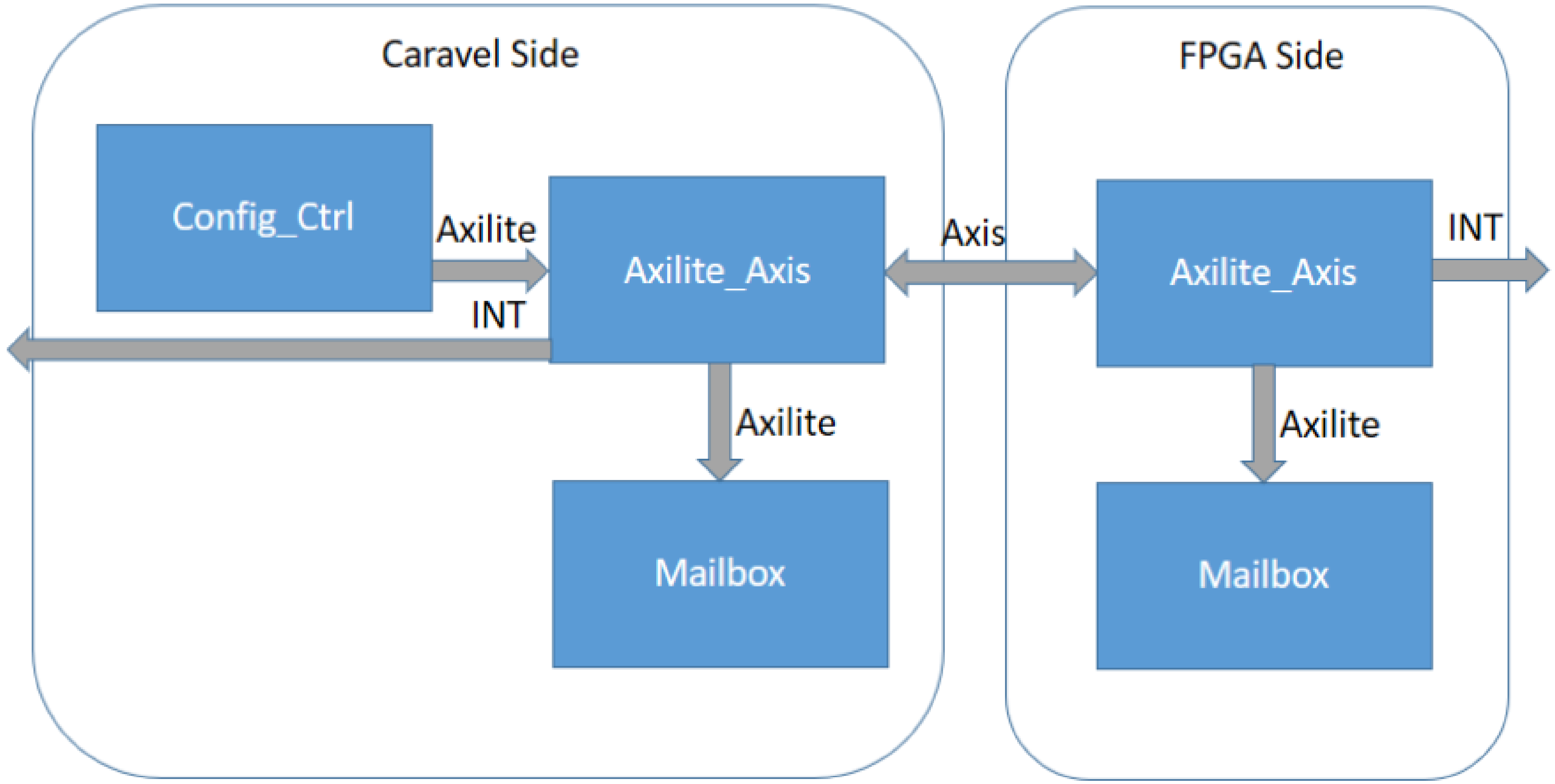
The mailbox is a set of registers which provides a communication channel between Caravel/RISC-V and FPGA/ARM. The operation mechanism is below:

1. The mailbox registers are duplicated in Caravel chip and in FPGA.
2. When one side (either Caravel or FPGA) write to mailbox, the transaction is passed to other side.
3. When the mailbox is updated, an interrupt is generated to the other side.
4. The mailbox addresses is defined in the user address space in 'h3300_000 - 'h33ff_ffff.

Mailbox Operation

Operation	**FPGA side**	**Caravel side**
Caravel Write	Update mailbox, generate interrupt to ARM	Update mailbox, generate message cycle to FPGA
Caravel Read	NOP	Return mailbox content
FPGA Write	Update mailbox, generate message cycle to Caravel	Update mailbox, generate interrupt to Caravel
FPGA Read	Return mail content	NOP

Interface Block

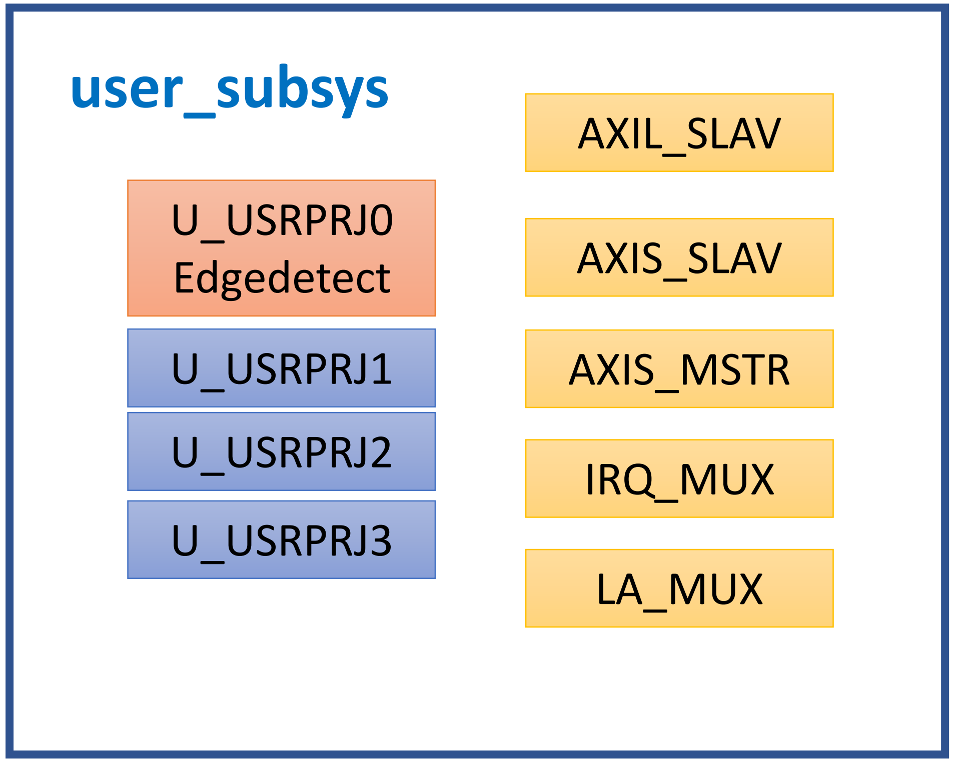


Configuration Registers – 8 x 32-bit Registers

Mailbox: 15’h2000 – 15’h201F

RegisterName	**Offset Address**	**Description**
mb_reg_0[31:0]	'h0	
mb_reg_1[31:0]	'h4	
mb_reg_2[31:0]	'h8	
mb_reg_3[31:0]	'hc	
mb_reg_4[31:0]	'h10	
mb_reg_5[31:0]	'h14	
mb_reg_6[31:0]	'h18	
mb_reg_7[31:0]	'h1c	

User_subsys

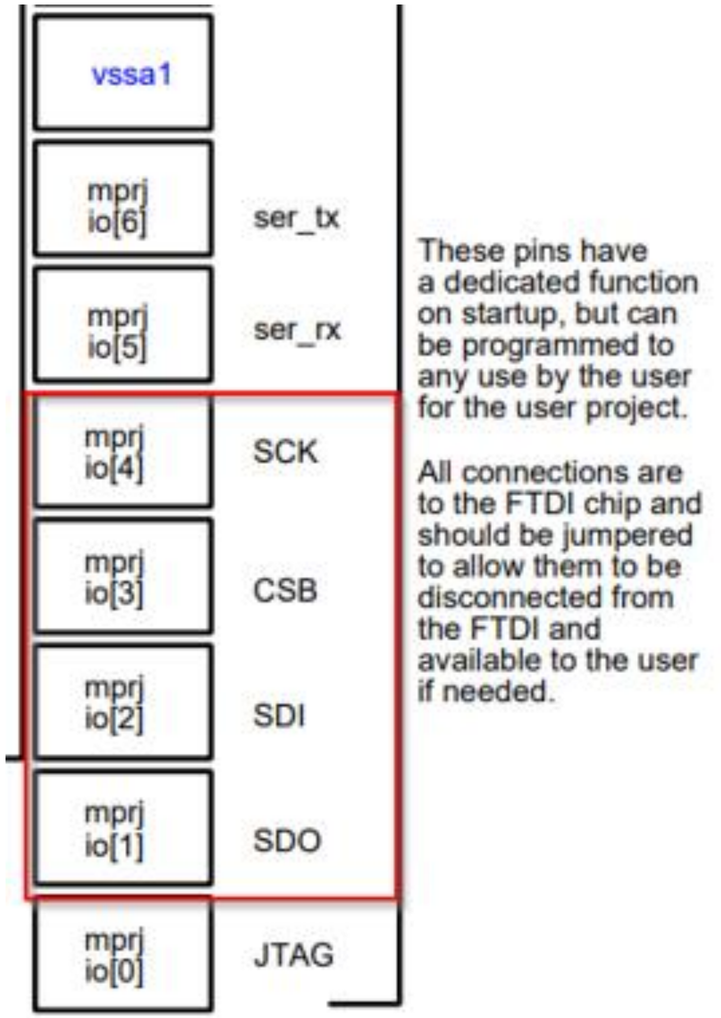


```
../axil_slav/rtl/axil_slav.v
../axis_mstr/rtl/axis_mstr.v
../axis_slav/rtl/axis_slav.v
../irq_mux/rtl/irq_mux.v
../la_mux/rtl/la_mux.v
-f ../user_prj/user_prj0/rtl/rtl.f
-f ../user_prj/user_prj1/rtl/rtl.f
-f ../user_prj/user_prj2/rtl/rtl.f
-f ../user_prj/user_prj3/rtl/rtl.f
../rtl/user_subsys.v
```

MPRJ_IO Usage

[7:0] - Reserved for System Usage

```
// MPRJ_IO PIN PLANNING when pSERIALIO_WIDTH=
// --------------------------------
// [20: 8]  I   RXD
// [    21]  I   RXCLK
// --------------------------------
// [34:22]  O   TXD
// [      35]  O   TXCLK
// --------------------------------
// [      36]  I   IO_CLK
```


vssa1
mprj io[6]
ser_tx
mprj io[5]
ser_rx
mprj io[4]
SCK
mprj io[3]
CSB
mprj io[2]
SDI
mprj io[1]
SDO
mprj io[0]
JTAG
These pins have a dedicated function on startup, but can be programmed to any use by the user for the user project.
All connections are to the FTDI chip and should be jumpered to allow them to be disconnected from the FTDI and available to the user if needed.

fsic_clkrst

```
assign axi_clk = wb_clk;
assign axis_clk = wb_clk;
```

axi_rst_n, axis_rst_n, uck2_rst_n delay 2T from wb_rst

```
assign user_irq[2] = high_pri_irq;
assign user_irq[1] = low__pri_irq;
assign user_irq[0] = mb_irq;
```

https://github.com/bol-edu/fsic_fpga/blob/main/rtl/user/fsic_clkrst/rtl/fsic_clkrst.v

Supplement

User Project SideBand Signals Support ??

Issues with AXIS_SWITCH

// 1. register initialization ?

// 2. demux using FIFO mem - what if sink endpoint inserts wait state, can we allow concurrent mulitple sinks ??

// 3. mux - arbitration - no need for FIFO

// 4. why (* ramstyle = "no_rw_check" *)

// 5. arbitration structure ?