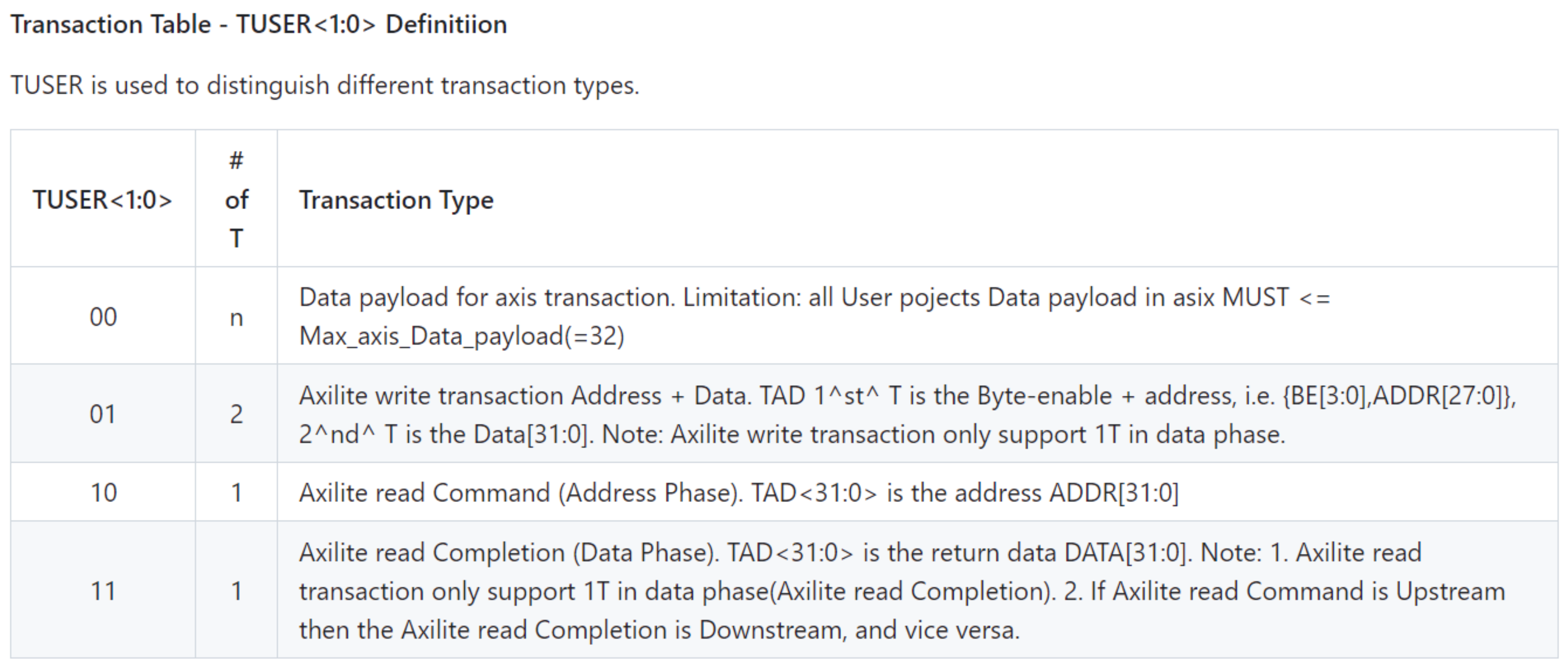
Full Stack IC (fsic) Designer Development

Axilite-Axis (AA) module implementation

Jiin Lai

Axilite-Axis (AA) module functions

- This module does Axilite to Axis, Axis-to-Axilite transaction conversion.
- FSIC-AXIS specification defines Axilite bus overlays on Axis bus.
  - https://github.com/bol-edu/fsic_fpga/blob/main/fsic-spec-dev/modules/FSIC-AXIS%20interface%20specification.md



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.

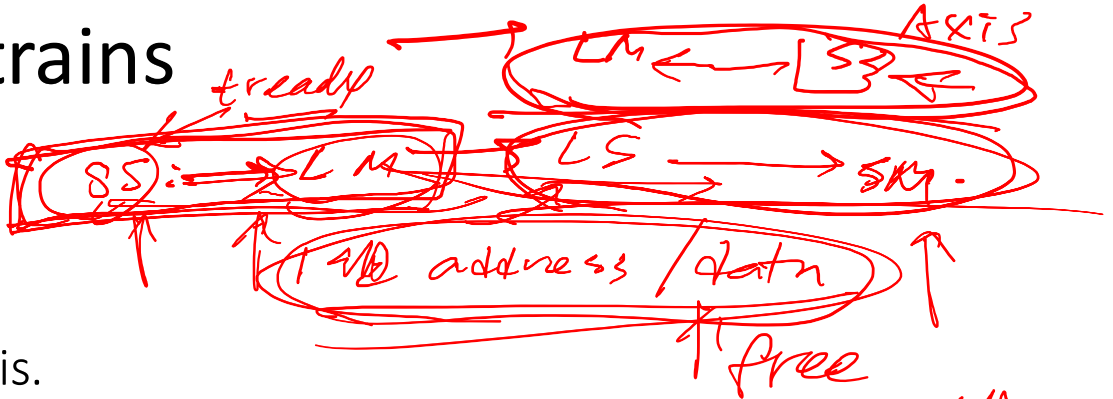
AA ignore

ss_wr, sm_wr

ss_rd, sm_rd

ss_rs, sm_rs

Implementation Features and Constrains



This is a different implementation scheme.

Key assumption:

1. No pipeline support for axis-2-axilite, axilite-2-axis.
2. Axis-2-axilite, axilite-2-axis can be concurrent, i.e. LS, SS can accept transaction
3. If 2 is too complicated, we can make it LS, SS exclusive by holding either tvalid, or tready

SS tready LS

2 W addr/data

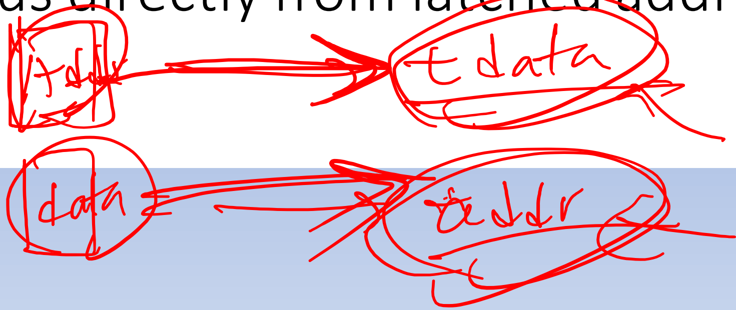
32 bit 32

64 bit

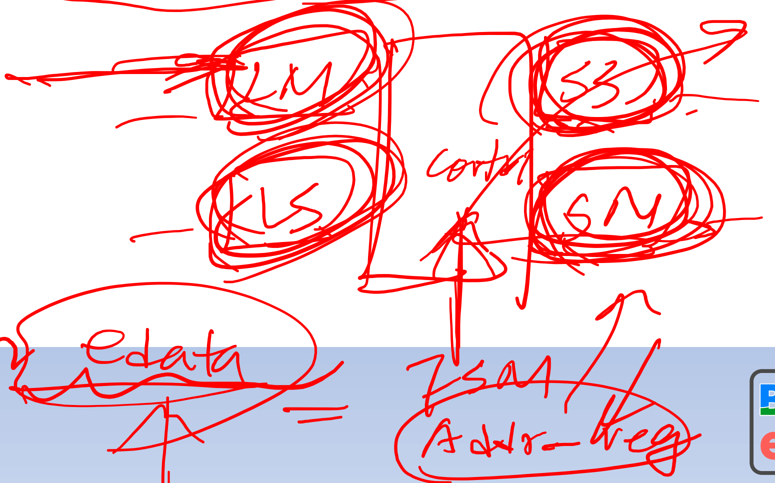
128 reg

1. No FIFO, except necessary address/data latches
2. Take advantage dont-care cycle to save gatecount, i.e. bus signals is only valid when it is sampled, and dont-case in other cycle.
3. Use FSM to link SS, SM, LS, LM bus transaction, mainly the tvalid, tready generation
4. Interface signal directly generated from FSM
5. Data/Address bus directly from latched address/data

- valid

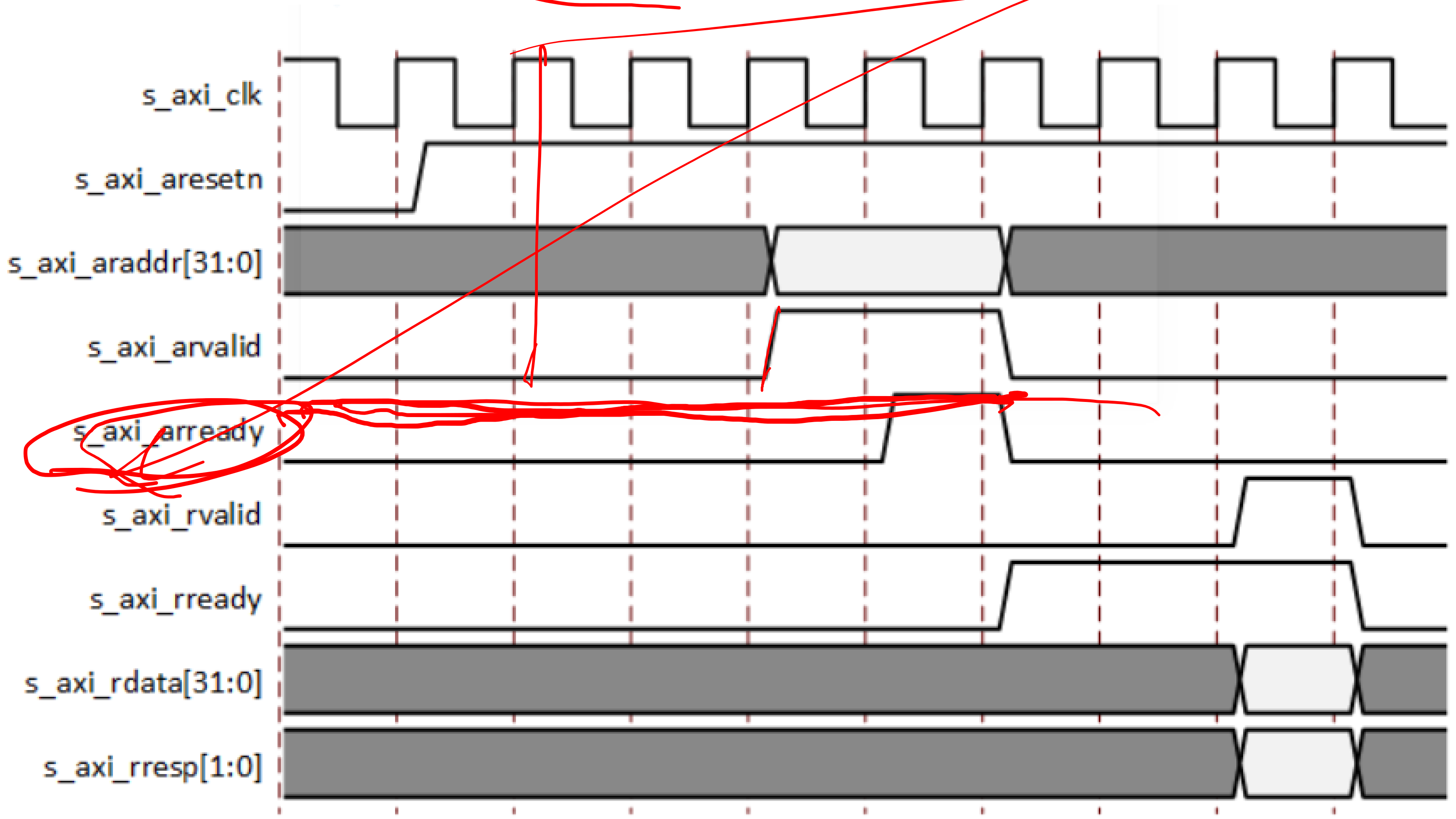


assign edata =

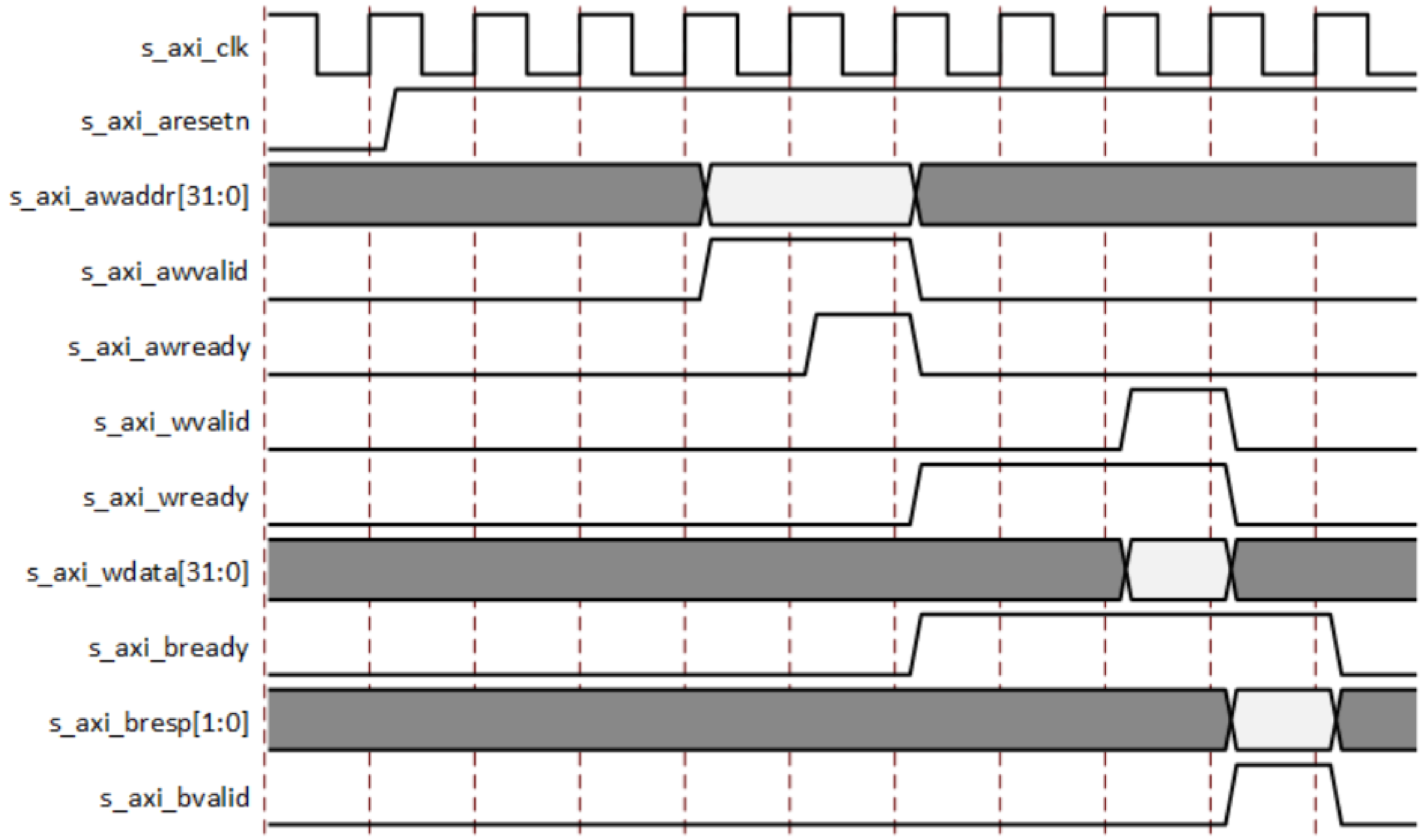


AXI4-Lite Read Transaction

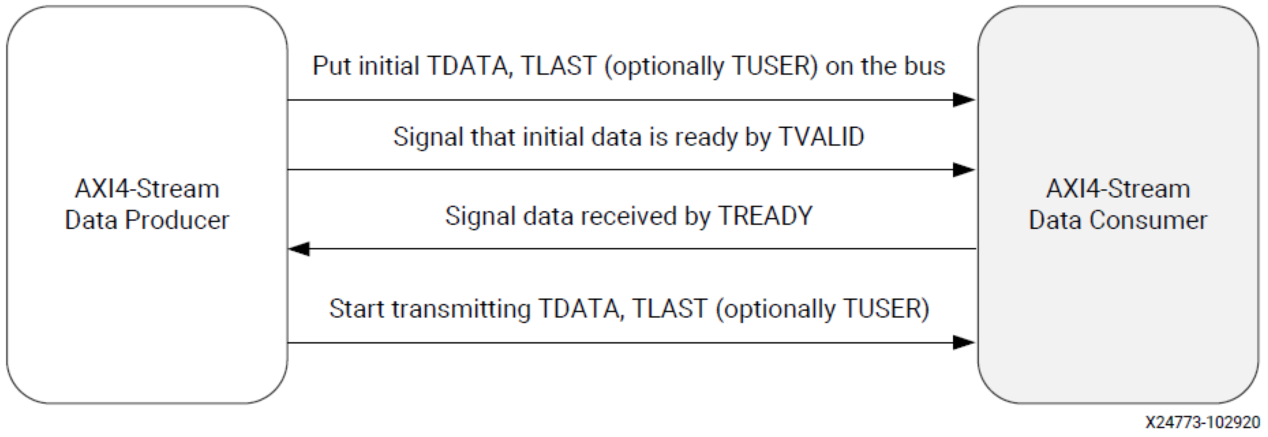
addre / data latch = ~~em~~ free status

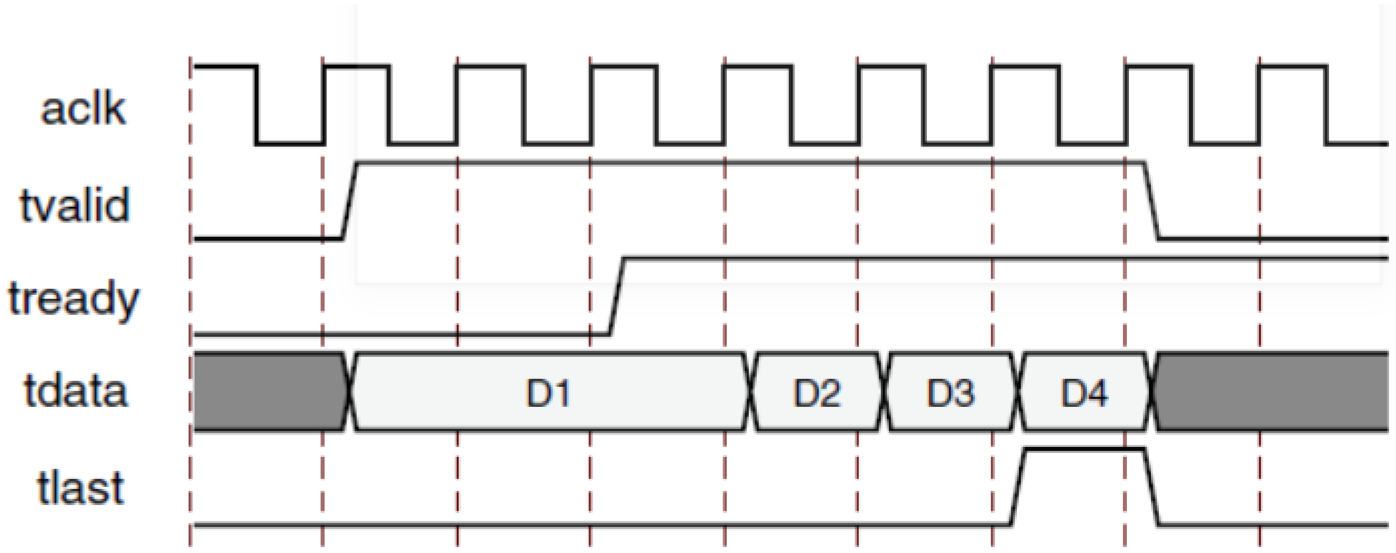


AXI4-Lite Write Transaction



AXI4-Stream Transfer Protocol

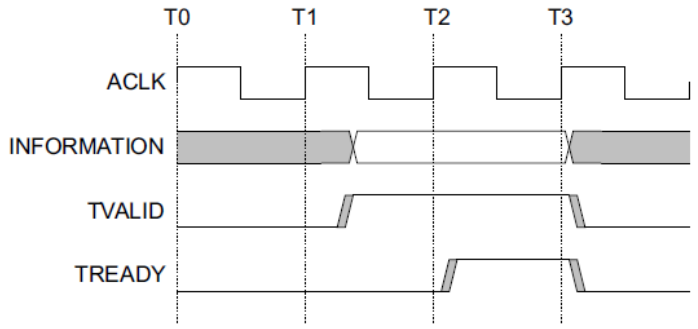

AXI4-Stream Data Producer
Put initial TDATA, TLAST (optionally TUSER) on the bus
Signal that initial data is ready by TVALID
Signal data received by TREADY
Start transmitting TDATA, TLAST (optionally TUSER)
AXI4-Stream Data Consumer
X24773-102920



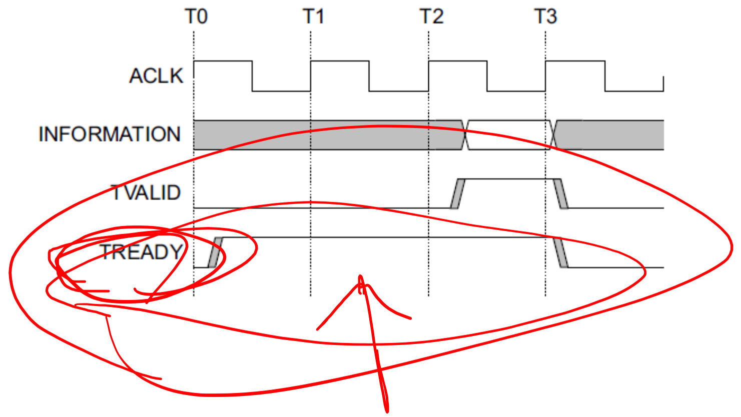
Data Transfer Handshake : TVALID, TREADY

- For a transfer to occur, both **TVALID** and **TREADY** must be asserted
- A Transmitter is not permitted to wait until **TREADY** is asserted before asserting **TVALID**
- Once **TVALID** is asserted, it must remain asserted until the handshake occurs
- A Receiver is permitted to wait for **TVALID** to be asserted before asserting **TREADY**

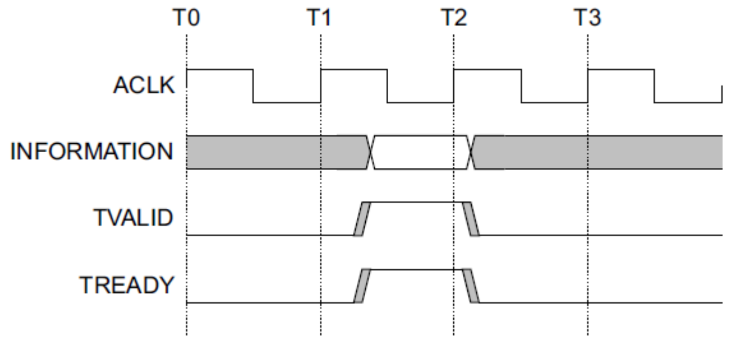
TVALID asserted before TREADY



TREADY asserted before TVALID



TVALID and TREADY asserted simultaneously



Design Steps

- Understand Bus Protocol – Axilite Read/Write (LM, LS), Stream Master/Slave (SM, SS)
- Draw transaction Waveform
  - Identify internal states, control/status signals
- Coding
  - Define internal states, control signals
  - Generate output signal using internal states, control signals
  - FSM, link control signals
- Special case handling, e.g. AA internal register, Mbox

$LS \Longrightarrow SM$

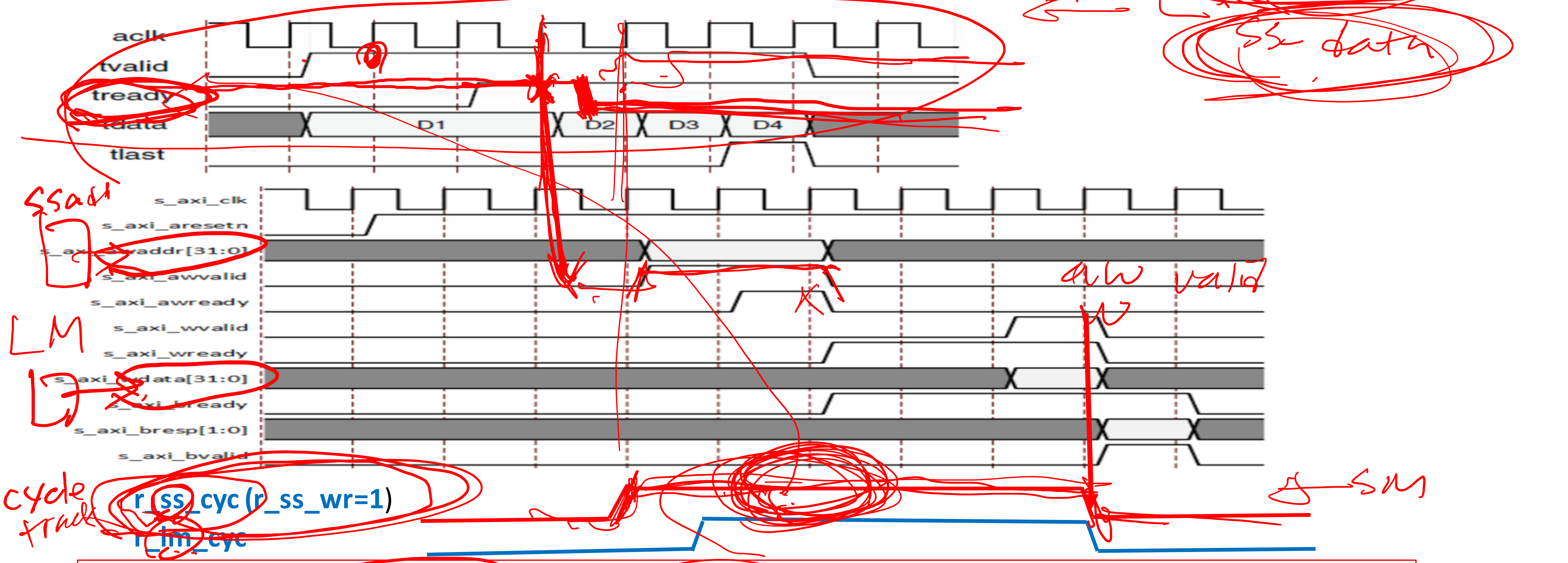
Transaction Type

- SS -> LM Write Request/Data
- SS -> LM Read Request
- SS -> Response/Data
- LS -> S~~S~~M Write / Mbox (W)
- LS -> S~~S~~M Read Request
- LS Read Response/Data
- LS -> AA-Reg (R/W) / Mbox (R) (terminate)

SS	: Stream Slave
SM	: Stream Master
LS	: Axilite Slave
LM	: Axilite Master

SS -> LM Write Request/Data

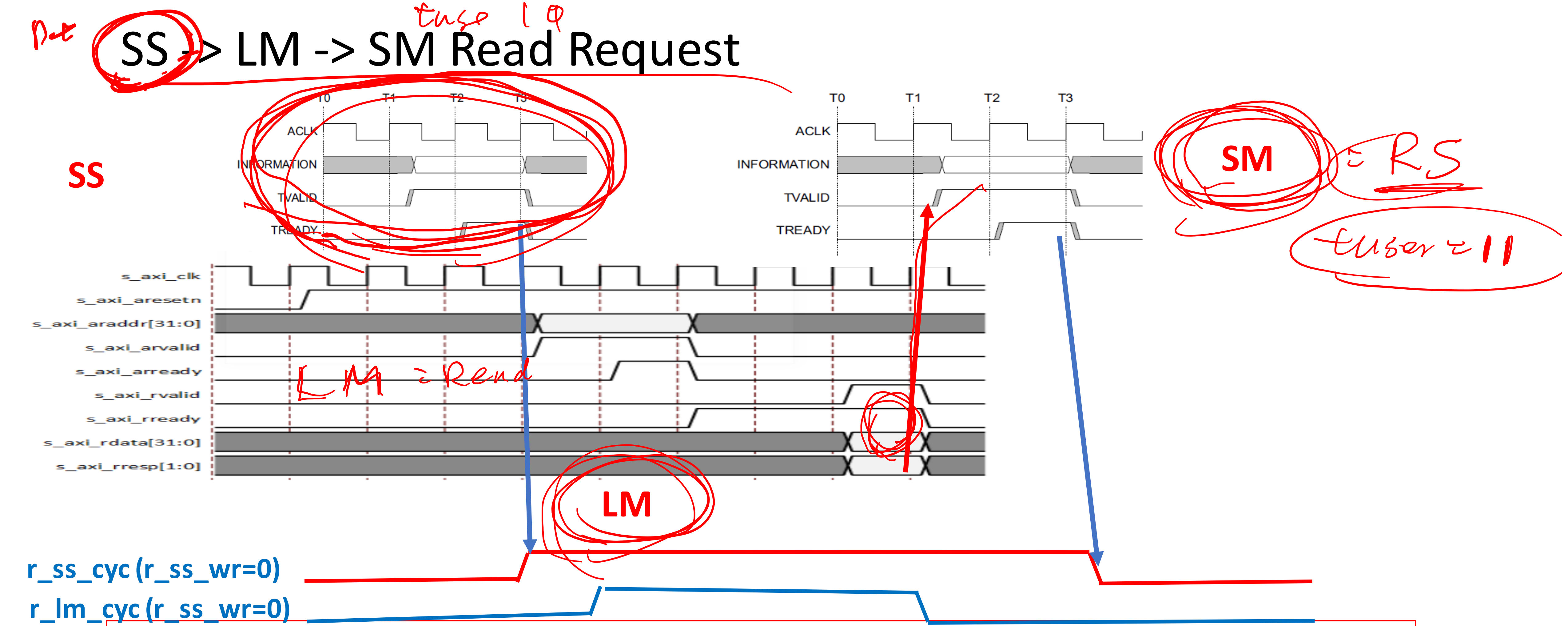
Tuser = 01



ss: ss_addr free
ss_data

1. Latch address (r_ss_rw_addr), data (r_ss_w_data)
2. Cycle-tracker (r_ss_cyc), and status (r_ss_wr) to
   1. Trigger LM by (r_ss_cyc & !rr_ss_cyc), i.e. r_ss_cyc rising edge
   2. Freeze the update on r_ss_rw_addr and r_ss_w_data → Not ~~add~~ valablO
   3. tready is held when r_ss_cyc, tready = ~r_ss_cyc;
3. Cycle-tracker (r_lm_cyc, r_lm_done)

SS -> LM -> SM Read Request

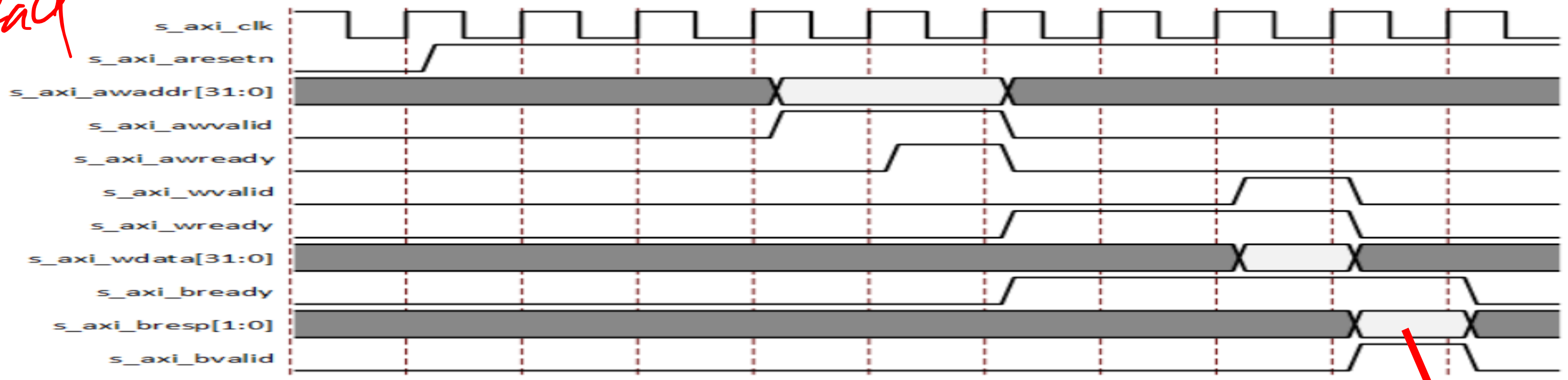


1. Latch address (r_ss_rw_addr), data (r_ss_w_data)
2. SS Cycle-tracker (r_ss_cyc), and status (r_ss_wr) to
   1. Trigger LM by (r_ss_cyc & ~rr_ss_cyc) , i.e. r_ss_cyc rising edge
   2. Freeze the update on r_ss_rw_addr
3. LM cycle-tracker (r_lm_cyc, r_lm_done)
   1. r_lm_done & r_ss_rd trigger SM to perform ss_rs transaction
4. r_ss_sm_rs_cyc = set by lm rs till sm tready => used to generate sm tready

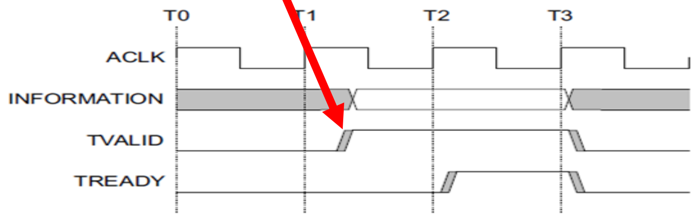
LS -> SM ~~Write~~ Read -> SS RS -> LM RREADY

Read

LS



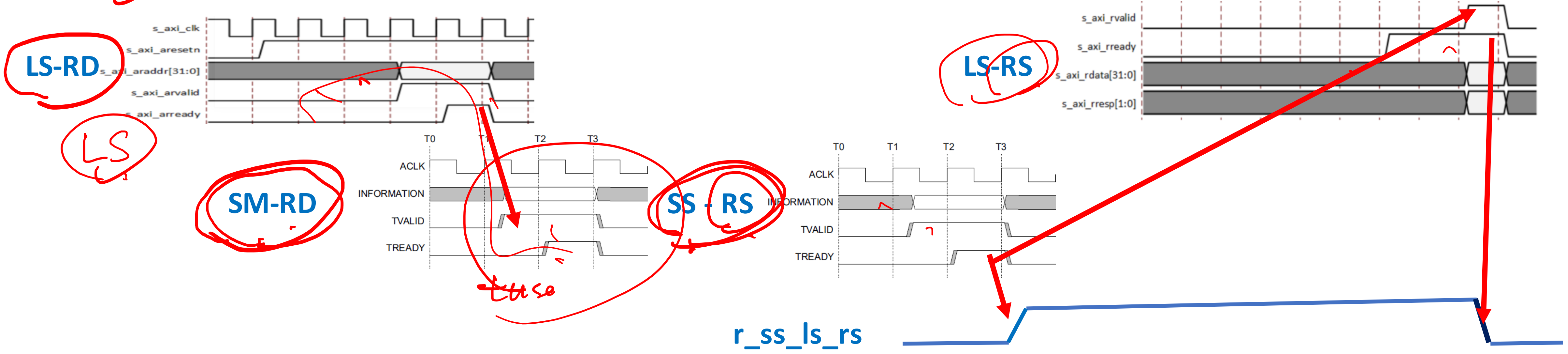
SM



1. Latch address (**r_ls_rw_address**), data (**r_ls_w_data**)
2. **ls_aa_internal** (address is in ‘h3000_2000 (aa), ‘h3000_3000 (mbox, read)
3. LS cycle tracker (**r_ls_cyc, r_ls_wr**)
   1. r_ls_cyc from (awvalid) to **r_sm_done**)
4. SM trigger by r_ls_cyc rising edge (**r_ls_cyc & !rr_ls_cyc & !ls_aa_internal**)

Read

LS -> SM-RD -> SS_RS -> LS-RS



1. Latch address (**r_ls_rw_address**)
2. SS-RS, latch response data (**r_ss_rs_data**)
3. LS cycle tracker (**r_ls_cyc, r_ls_wr**)
4. SM trigger by **r_ls_cyc rising**
5. SS receive RS cycle triggers LS-RS (respond data with **r_ss_r_data**
6. **r_ss_ls_rs – it is used to generate LS s_rvalid**

Code Structure

FSM – SS_FSM – Tracking SS Received Transaction

- SS (Write) -> LM Write (if not AA internal)
- SS (Write) terminate if AA internal
  - If Mbox, generate Interrupt
- SS (Read) -> LM Read -> SM RS (if not AA Internal)
- SS (Read) -> SM RS (if AA internal)
- SS (RS) -> LS (RS)

- Encode cycle tracking signal, e.g. r_ss_cyc, r_lm_done … into FSM
- Encode output signal, tvalid, tready, … into FSM
- The output can be the state qualified by other status signals

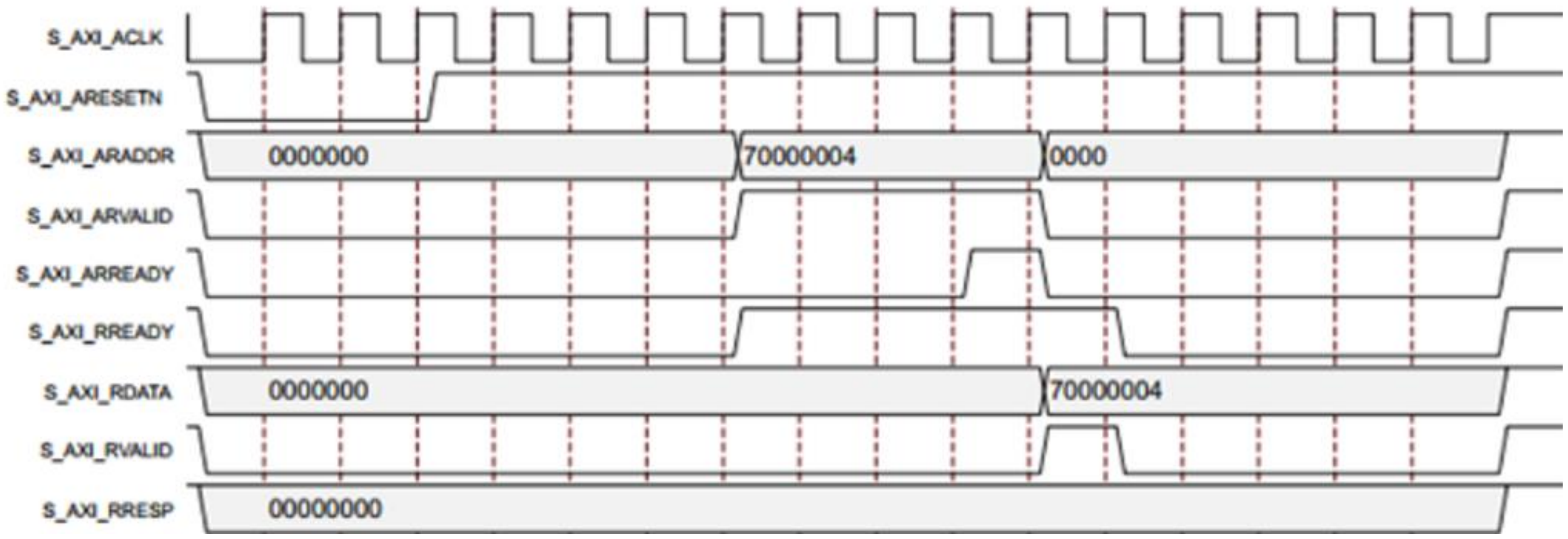
FSM – LS_FSM – Tracking LS Received Transaction

- LS (Write) -> SM Write
- LS (Read) -> SM Read -> LS RS -> LS RS
- LS (R/W) access AA Internal
  - Read/write AA Internal, Read Mbox - immediate complete
  - Write Mbox – write internal mbox and generate SM Write

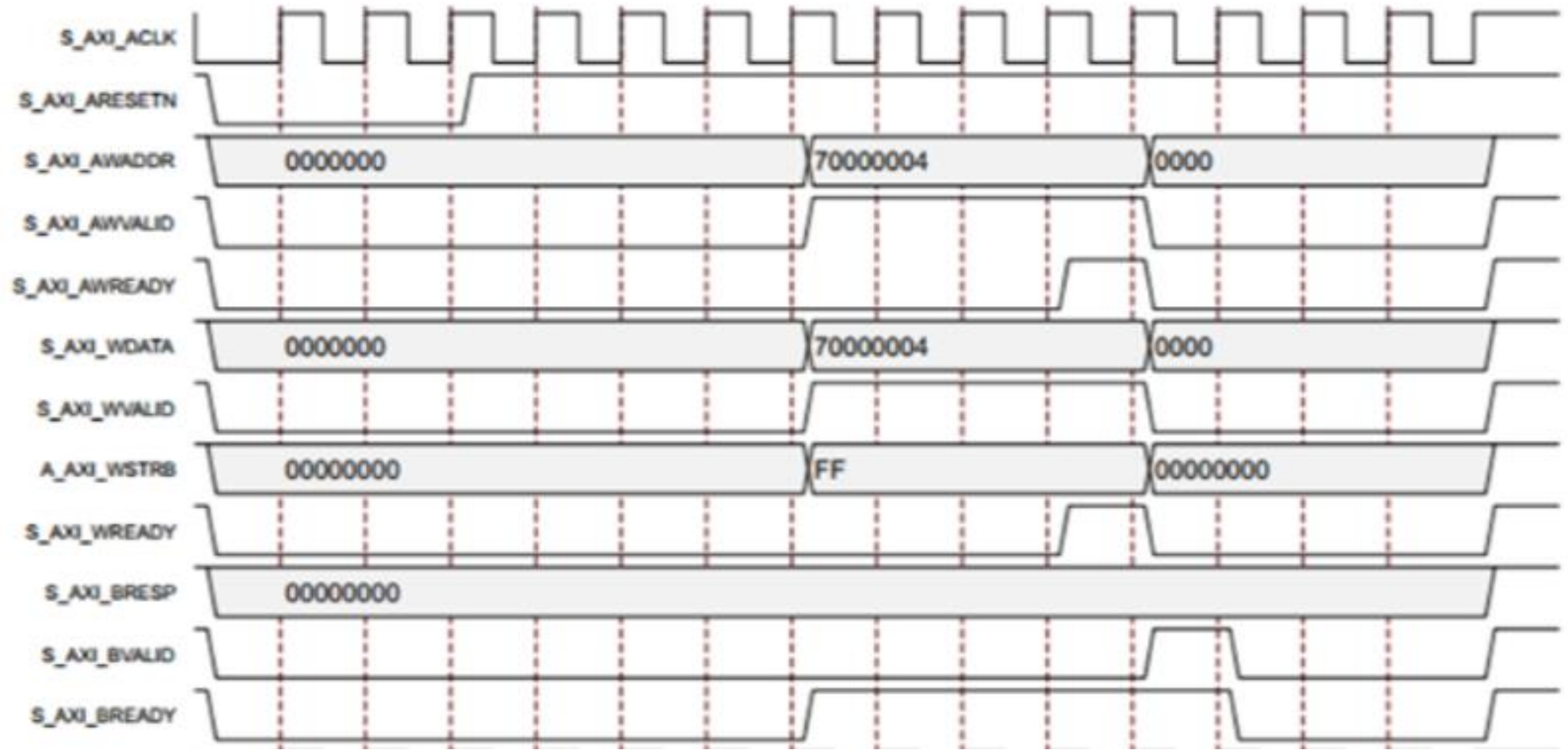
- Encode cycle tracking signal, e.g. r_ss_cyc, r_lm_done … into FSM
- Encode output signal, tvalid, tready, … into FSM
- The output can be the state qualified by other status signals

Supplement

AXI4-Lite Read Transaction



AXI4-Lite Write Transaction


S_AXI_ACLK
S_AXI_ARESETN
S_AXI_AWADDR
0000000
70000004
0000
S_AXI_AWVALID
S_AXI_AWREADY
S_AXI_WDATA
0000000
70000004
0000
S_AXI_WVALID
A_AXI_WSTRB
00000000
FF
00000000
S_AXI_WREADY
S_AXI_BRESP
00000000
S_AXI_BVALID
S_AXI_BREADY