AXI Task OPGAL

RDD

Requirements and Design Document

written by:*Liron Shenkar*

Revision History

#	Revision	Date	Notes	Writer
(1)	V1.0	04/17/20	Initial version	Liron Shenkar

Shortcuts and abbreviations

#	*Abbreviation*	*Full Name*
1.	AXI	Advanced Extensible Interface
2.	AMBA	Advanced Microcontroller Bus Architecture

Table of Contents

1 Scope

This document describes the AXI interface implementation according to the OPGAL AXI task. The document details the various logic entities, timing consolidations, state machines and all other important issues regarding the slogic design .

2 Applicable documents & Reference websites

1. https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface
2. UG1037 - Vivado Design Suite: AXI Reference Guide (v4.0)
3. UG761 - AXI Reference Guide (v14.3)
4. AMBA AXI and ACE ProtocolSpecification
5. https://www.xilinx.com/training/customer-training/axi-introduction
6. https://www.xilinx.com/training/customer-training/connecting-axi-ip-demo
7. https://www.xilinx.com/video/hardware/how-to-use-the-3-axi-configurations
8. https://www.xilinx.com/video/hardware/axi4-lite-interfaces-for-vivado
9. https://www.xilinx.com/products/intellectual-property/axi/axi4_ip
10. Dillon Huff-What is a AXI?
11. Dillon Huff-What is a AXI LITE?
12. Introduction to AXI4-Lite

3 General

3.1 AXI Interface

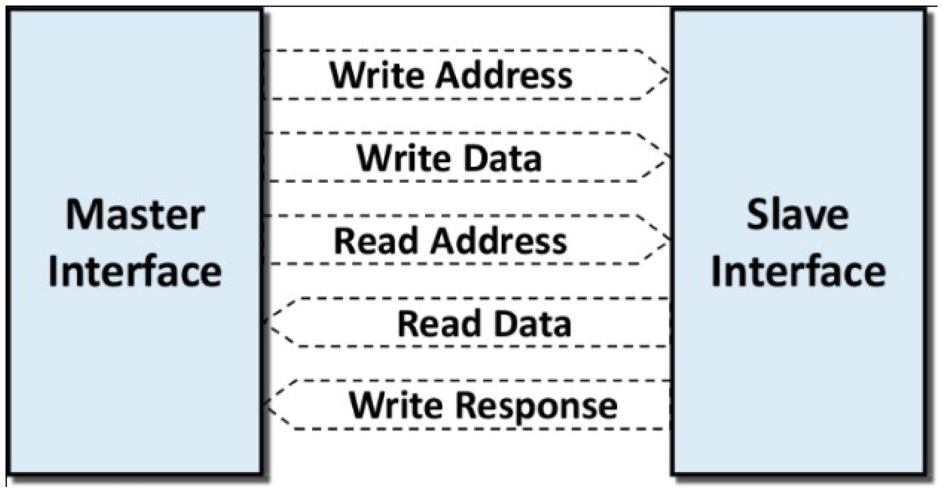
Advanced eXtensible Interface 4 (AXI4) is a family of buses defined as part of the fourth generation of the ARM Advanced Microcontroler Bus Architectrue (AMBA) standard. AXI was first introduced with the third generation of AMBA, as AXI3, in 1996.

The AMBA specification defines 3 AXI4 protocols:

- AXI4: A high performance memory mapped data and address interface. Capable of Burst access to memory mapped devices.
- AXI4-Lite: A subset of AXI, lacking burst access capability. Has a simpler interface than the full AXI4 interface.
- AXI4-Stream: A fast unidirectional protocol for transferring data from master to slave.

AXI4-Lite Interface Signals

The AXI4-Lite interface consists of five channels: Read Address, Read Data, Write Address, Write Data, and Write Response all shown in Drawing 1.



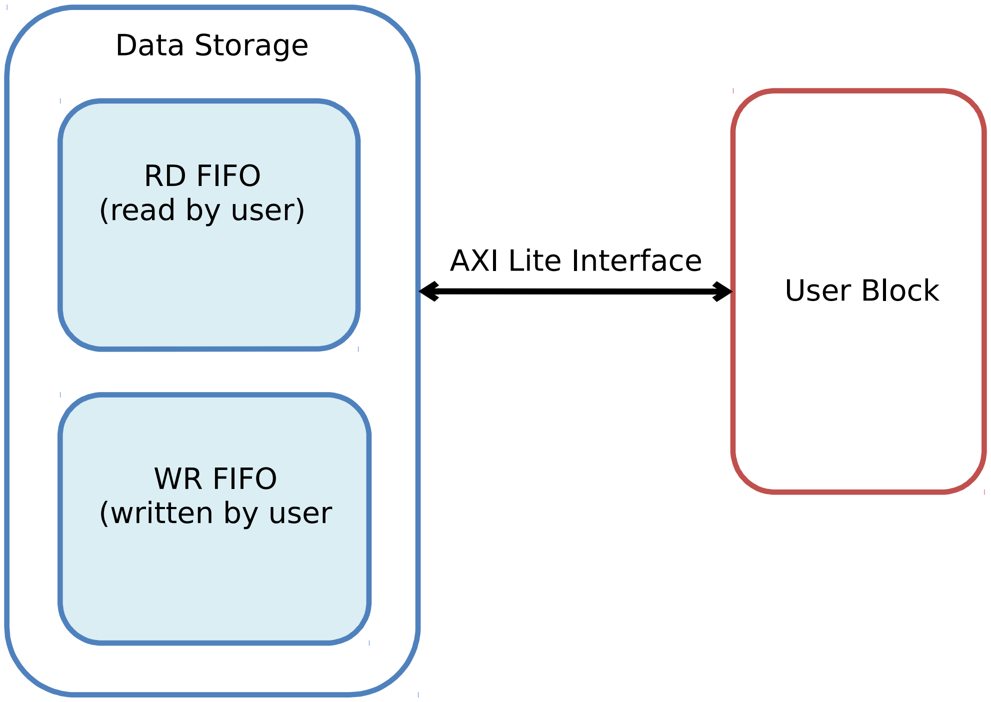
Drawing 1: AXI Channels

For detailed information about the AXI and AXI Lite interface see 2

3.2 Requirements

The Following section is copied from the task document:

- General Blocks View:



Drawing 2: The FIFO Interface Diagram

Name	Direction	Description
clk	IN	clock
rst	IN	Reset-active high
WR_EN	IN	The data in the Din bus is written to the FIFO when WR_EN=’1’
Din	IN (32 bit)	Data to FIFO
RD_EN	IN	Dout bus consist the read data from the FIFO when RD_EN=’1’
Dout	OUT (32 bit)	Data from FIFO
Empty	OUT	FIFO is empty
Full	OUT	FIFO full

AXI Lite is a known ARM AMBA AXI Protocol, the protocol is available on the internet.

- Tasks:
    1. Please create schematic design for FIFO Interface to AXI Lite interface (and vice versa) converting block/s. Schematic design can be FSM, Block, States diagram etc.
    2. Please write VHDL code for the RD FIFO channel

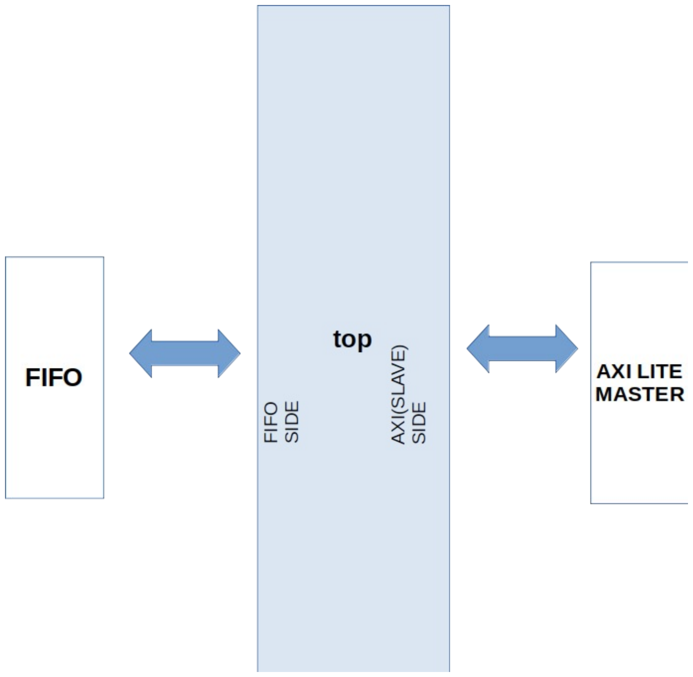
3.3 Micro architecture (MAS) Design

The following section includes a detailed description of the carious components of the design.

3.3.1 Top

File Name :top.vhd

In order to implement the FIFO INTERFACE to AXI LITE translator, the following top entity is implemented:



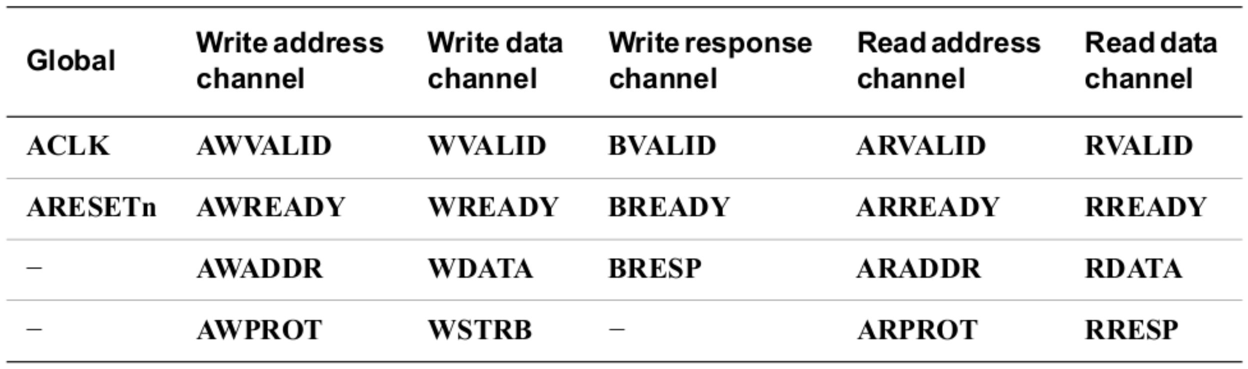
Drawing 3: The top module interfaces

As a translator, the top module has two sides:

1. The FIFO side - The translator is the ***master*** of the FIFO, and controls all the read and writes operations from/to the FIFO.
2. The AXI side – The translator is an AXI slave which communicates with an AXI master.

The FIFO interface signals are according to the table in the requirements (see 3.2) table. The AXI LITE 5- channel interface are fully described in chapters A2 and B1 in the AMBA AXI and ACE Protocol Specification(Link in 2). the following table shows the full AXI LITE interface:

Table B1-1 AXI4-Lite interface signals



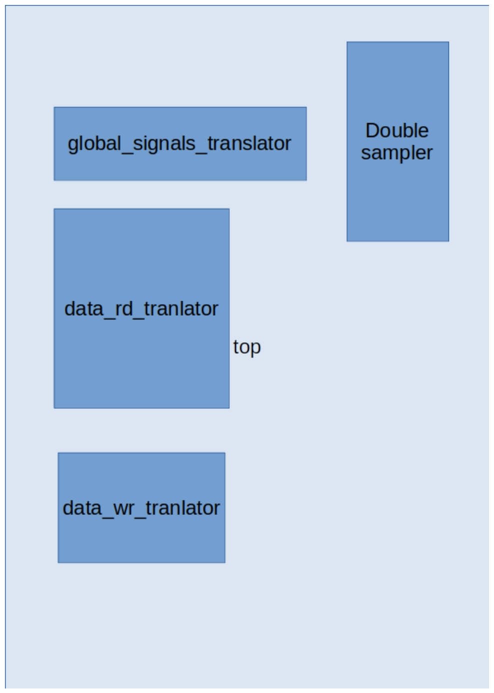
Global	Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
–	AWADDR	WDATA	BRESP	ARADDR	RDATA
–	AWPROT	WSTRB	–	ARPROT	RRESP

The AXI interface is designed for memory map applications, targeted to RAM-like access including address and data. However, since the FIFO has no address, only the following signals are needed:

- Global signals(ACLK, ARESETn)
- Read data channel (RRESP is optional)
- Write data channel
- Write response channel (optional)

The top module includes 4 components :

1. global_signals_translator – generating the resets signals for the entire design. Includes reset bridge.
2. double_sampler – performs double sampling to all inputs of the design. The output of the modules are the samples signals. *Note- alternatively ,double sampling can be performed inside the components themselves, not in a separate component.*
3. data_rd_translator – performs the read operation.
4. data_wr_tranlator – performs the write operation.



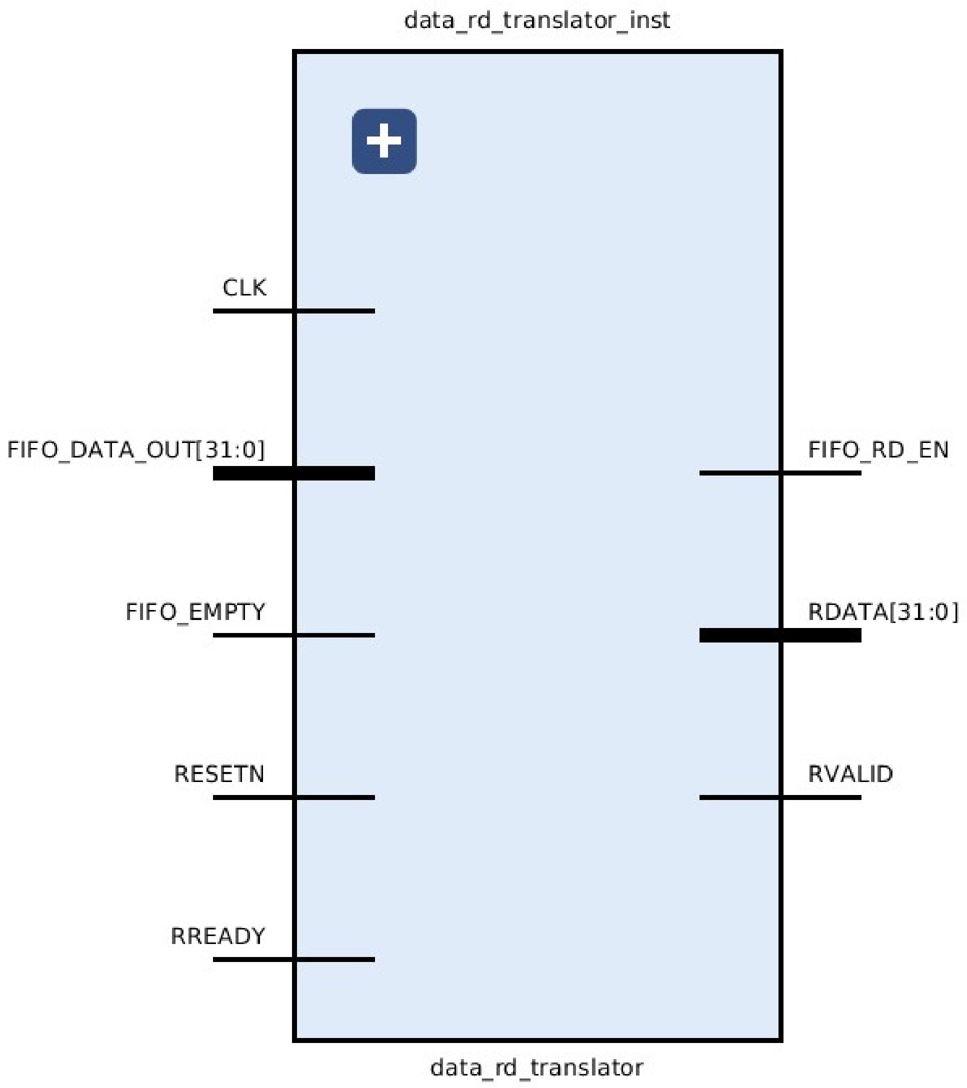
Drawing 4: top module components

3.3.2 data_rd_translator

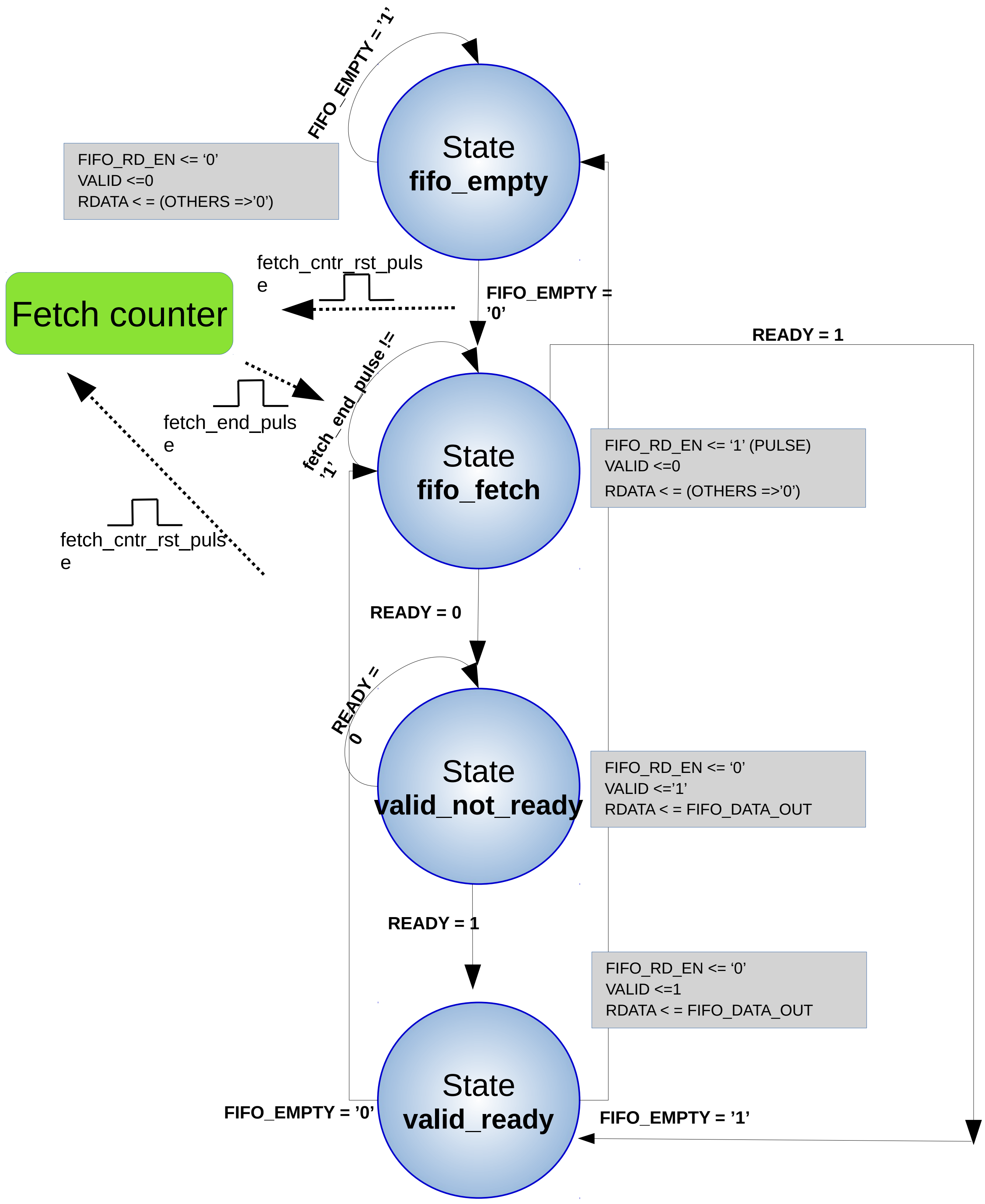
File Name :data_rd_translator.vhd

This component performs the reading operations from the FIFO and transferring the read data to the AXI master. This entity is implemented in VHDL as required in 3.2.

The implementation of the reading is done by the read FSM :



Drawing 5: The data read translator



Drawing 6: read FSM

The state machine works according to the following guidelines of the AXI spec:

- A source is not permitted to wait until READY is asserted before asserting VALID.
- Once VALID is asserted it must remain asserted until the handshake occurs, at a rising clock edge at which VALID and READY are both asserted.

In addition:

- Both ready before valid and valid before ready are possible.
- One data transaction occurs on each VALID/READY handshake.

The machine works as follows:

1. As long is the FIFO is empty – no data can be fetched from the FIFO. This is the **st_fifo_empty**_state.
2. As the FIFO_EMPTY goes 'LOW':
   I. the machine enters the **st_fifo_fetch** state.
   II. A single clock pulse of FIFO_RD_EN is sent to the FIFO in order to fetch new data from the FIFO.

   Note
   The fetch state is needed since it might takes several clocks from the rise of FIFO_RD_EN until the data is valid on the RDATA ports. The number of clocks is determined by a constant and can be modified to our needs (fore example- more sampling stages of FIFO_DATA_OUT). It is assumed that there is a 1 clock delay from the end of the FIFO_RD_EN pulse until the new data gets out of the FIFO .

3. As the data is available on the RDATA ports (after counter reached the desired value):
   I. If READY = '0' the machine enters the state **st_valid_not_ready**. This is a "valid before ready" case. In this state the machine waits for the READY = '1' signal.
   II. If READY = '1' the machine enters the state **st_valid_ready**. This arrow could be ommitted for the sake of simplicity in the price of an extra clock).

   In either case the VALID goes HIGH and the new data is available to the master.

4. The **st_valid_ready** state is a 1-clock state, where both VALID and READY are 'HIGH' and a transaction takes place.

5. After the transaction takes place, the machine shall try to fetch new data from the FIFO, if possible (meaning, FIFO_EMPTY='0'). If it is not the case, the machine shall enter the **fifo_empty** state.

Here is a small example form the simulation showing the 'ready before valid" flow of the FSM. The state transition and counter are shown.

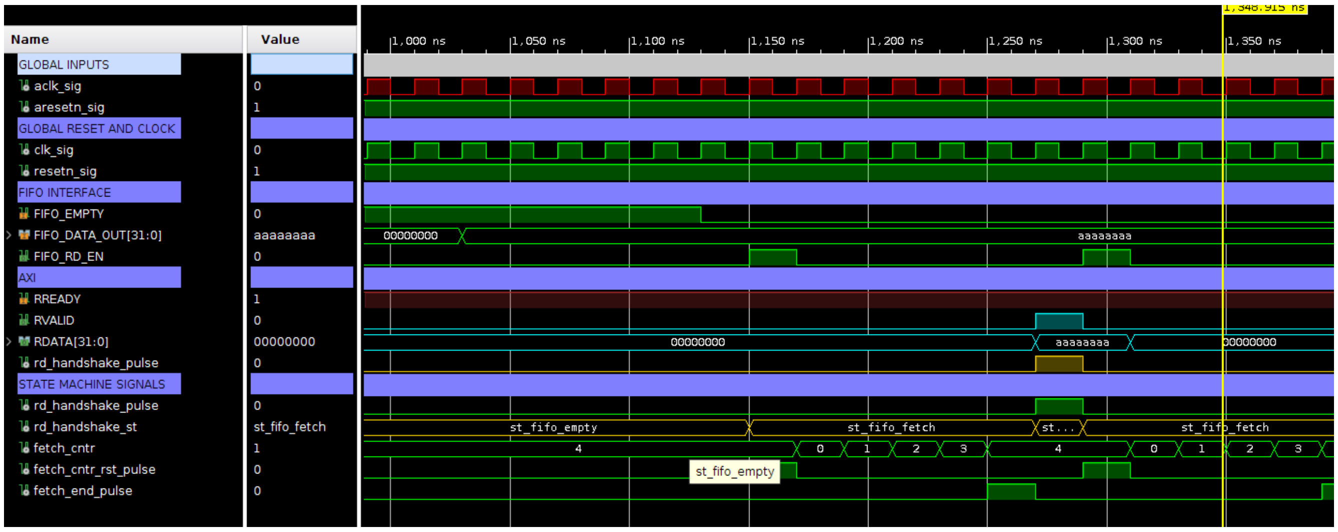


Illustration 1: valid before ready FSM flow

3.3.3 data_wr_translator

File Name :data_wr_translator.vhd

This component performs the writing operations to the the FIFO and transferring the read data from the AXI master.

The implementation of the writing is done by the following state machine (see write FSM).

The Machine works as follows:

1. As long is the FIFO is full the machine stays at **st_fifo_full** state.

   WREADY signal is set to LOW.

2. When the FIFO_FULL signal goes ‘LOW’, the machine enters the **st_rady_not_valid** state.

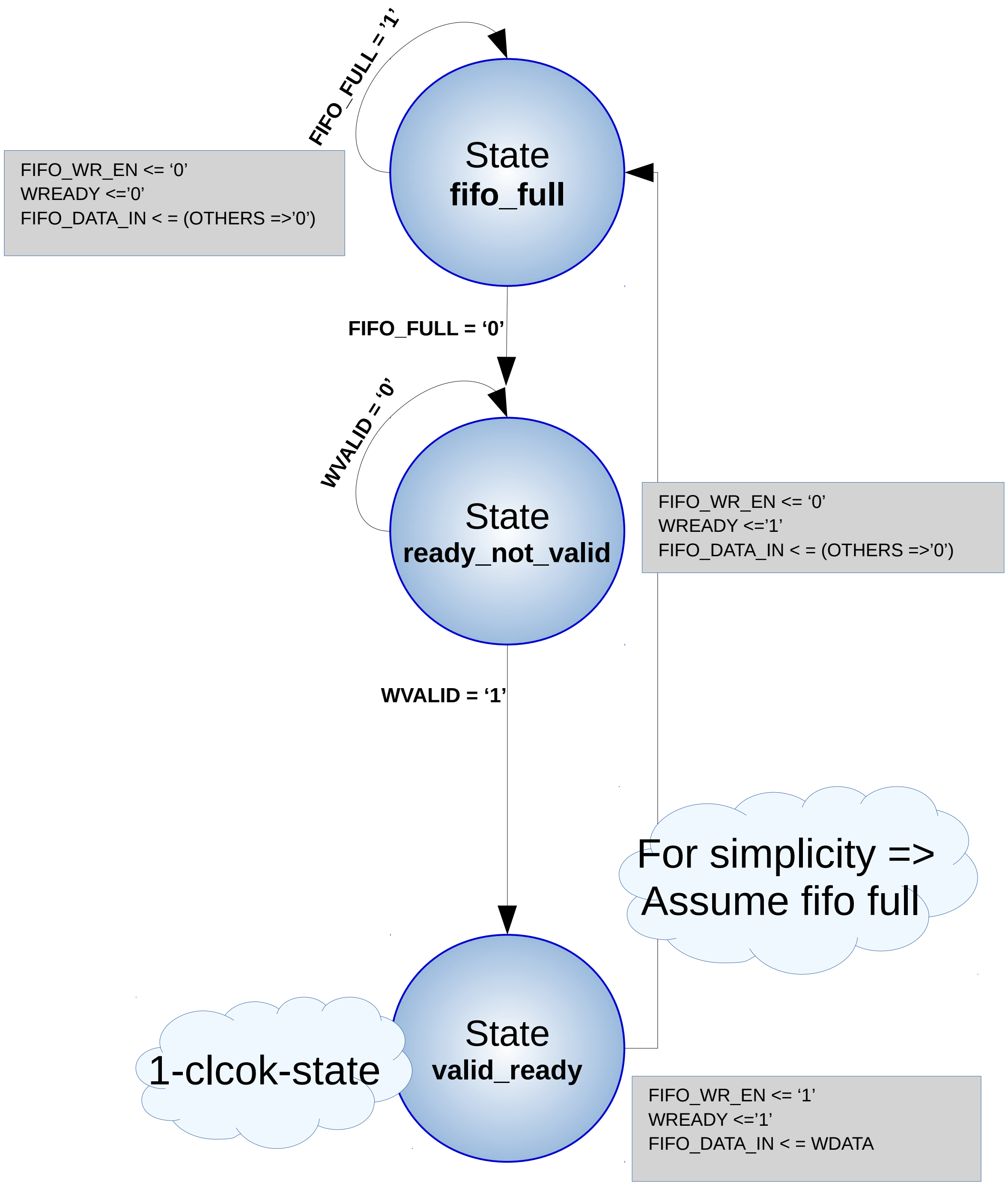
   The WREADY signal is set to ‘HIGH’.

3. If the master sets the WVALID to ‘HIGH’ The write data is available and valid on WDATA.

   The machine enters the **st_valid_ready** state. The FIFO_WR_EN is set to ‘HIGH’ for 1 cock (with the transition to **st_valid_ready).** The WDATA is transferred to the FIFO_DATA_IN.

4. The **st_valid_ready** state is a 1-clock state on which WVALID and WREADY are both ‘HIGH’ and FIFO_DATA_IN gets WDATA. For simplicity, after the transaction the machine enters the **st_fifo_full state** (and ‘WREADY’ goes down).

   An alternative implementation of the design is to let the machine wait until FIFO_FULL is valid, and if it is ‘LOW’ - to enter **st_ready_not_valid** state.



Drawing 7: write FSM