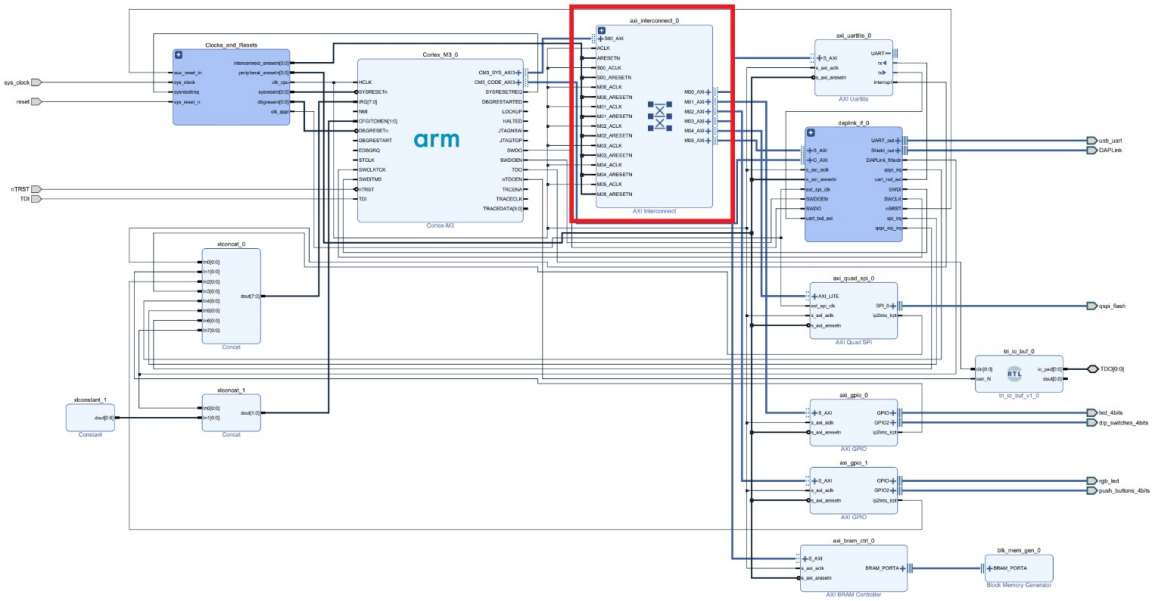
POLITECNICO
MILANO 1863

Embedded Systems AA [2020-21]

Colombi, Corbetta, Consonni



AXI Project

Prof. William Fornaciari
PhD Davide Zoni
PhD Std Andrea Galimberti

Deliverable:	Report
Title:	Project Report
Authors:	Marco Colombi 10631973 marco3.colombi@mail.polimi.it, Giorgio Corbetta 10570080 giorgio1.corbetta@mail.polimi.it, Cesare Consonni 10476810 cesare.consonni@mail.polimi.it
Version:	1.0
Date:	23-January-2021
Download page:	CCC-AXI git repo

Contents

1 Introduction

1.1 AXI Protocol

The AMBA AXI protocol supports high-performance, high-frequency system designs for communication between multi masters and multi slaves components. **AXI4** is widely adopted, providing **benefits** to **productivity** (standardization simplifies the work of developers), **flexibility** (there are slightly different protocols, eachone of them with their peculiarity), and **availability** (it's an industrial standard, and there is a world wide community that uses and support it).

AXI4-Lite, the procol that we studied, is a **subset of AXI4** for communication with simpler control register style interfaces within components.

Some **key features** of the AXI4-Lite protocol are: **separate address/control and data phases**, support for unaligned data transfers, using byte strobes, **separate read and write data channels**, all **transactions** are of burst **length 1**, all data accesses are non-modifiable, non-bufferable and use the full **width of the data bus** (the supported busses are the ones with width of **32-bit** (in our case) or 64-bit), exclusive accesses are not supported.

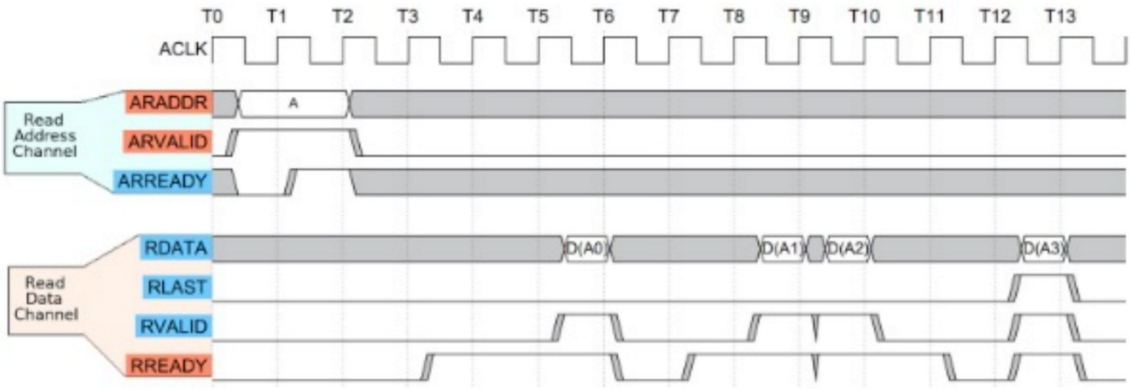


Figure 1: AXI4 (**non lite**) read

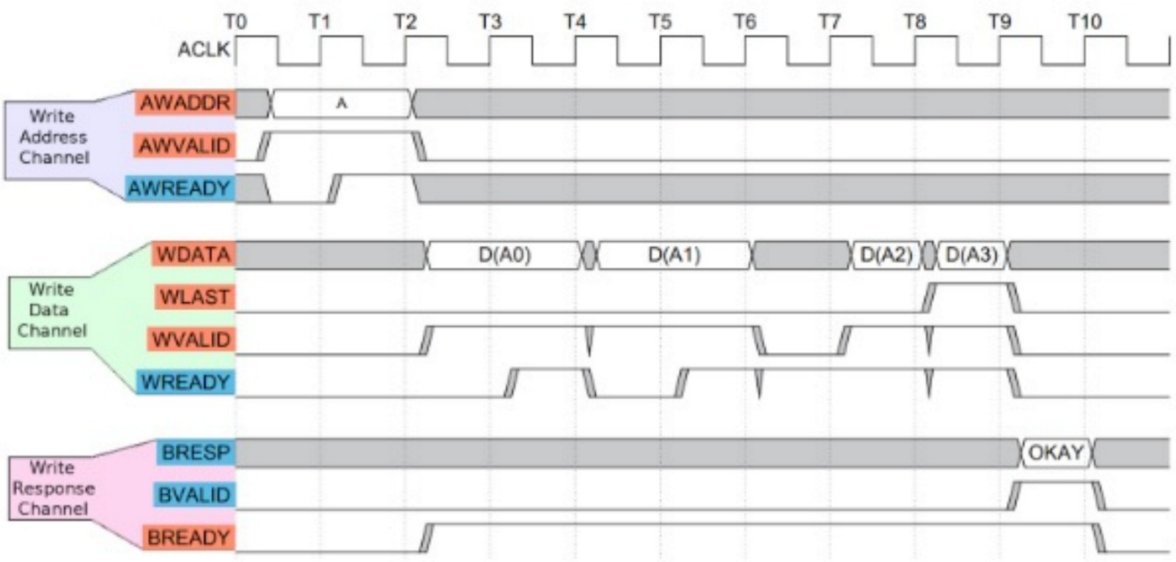


Figure 2: AXI4 (**non lite**) write

WRITE SIGNALS		
	AW group	
AWADDR	[31:0]	//where the master want to write
AWVALID	[0:0]	//the address line is valid
AWPROT	[2:0]	//access permissions
AWREADY	[0:0]	//the slave is ready to recieve the address
	W group	
WDATA	[31:0]	//what the master want to write
WSTRB	[3:0]	//which bytes of the WDATA are meaningful
WVALID	[0:0]	//the data line is valid
WREADY	[0:0]	//the slave is ready to recieve the data
	B group	
BREADY	[0:0]	//the master is ready to recieve the response
BRESP	[1:0]	//slave's response about the operation
BVALID	[0:0]	//the response line is valid
READ SIGNALS		
	AR group	
ARADDR	[31:0]	// where the master want to read
ARVALID	[0:0]	//the address line is valid
ARPROT	[2:0]	//access permissions
ARREADY	[0:0]	//the slave is ready to recieve the address
	R group	
RREADY	[0:0]	//the master is ready to recieve the data
RDATA	[31:0]	//the data requested by the master
RVALID	[0:0]	//the data line is valid
RRESP	[1:0]	//slave's response about the operation
MASTER controlled		
SLAVE controlled		

1.2 Project Scope

The scope of the project is to understand **how works the AXI4-Lite** communication protocol, and **design by ourselves** an "AXI interconnect" **component** to **replace** the real one inside the *Arm Cortex-M3 DesignStart FPGA-Xilinx edition* and **observe** its **behaviour** and the **differences** between them.

In our case there is only one master with multiple slaves (we don’t need arbitrer) and there is no need for clock gating (because the clock is shared between all components)

2 Our Design

Here we will explain how we developed our AXI interconnect, first explaining how it works as a stand-alone component, and then how we inserted it into the Processor.

2.1 Block Design

In the figure **??** is reported the Block Diagram of the AXI interconnect we developed.

2.2 Coupling

The coupling is achieved by the Muxers and Demuxers, driven by the Decoders which reads the Address and couple the address to the correspective slave.

2.3 Handshake

The handshake's signals are checked by the FSM that will drive the Decoders to signal when the address is valid and when it's not.

2.4 Errors

There are 2 types of error: The ones raised by the slave (and are generated by slave and so pass through the AXI interconnect) The ones raised if the address isn't mapped; in such a case we have an ad-hoc fake slave that sends the error to the master.

2.5 Integration inside the processor

We managed to put our AXI right inside the processor without any fake component.

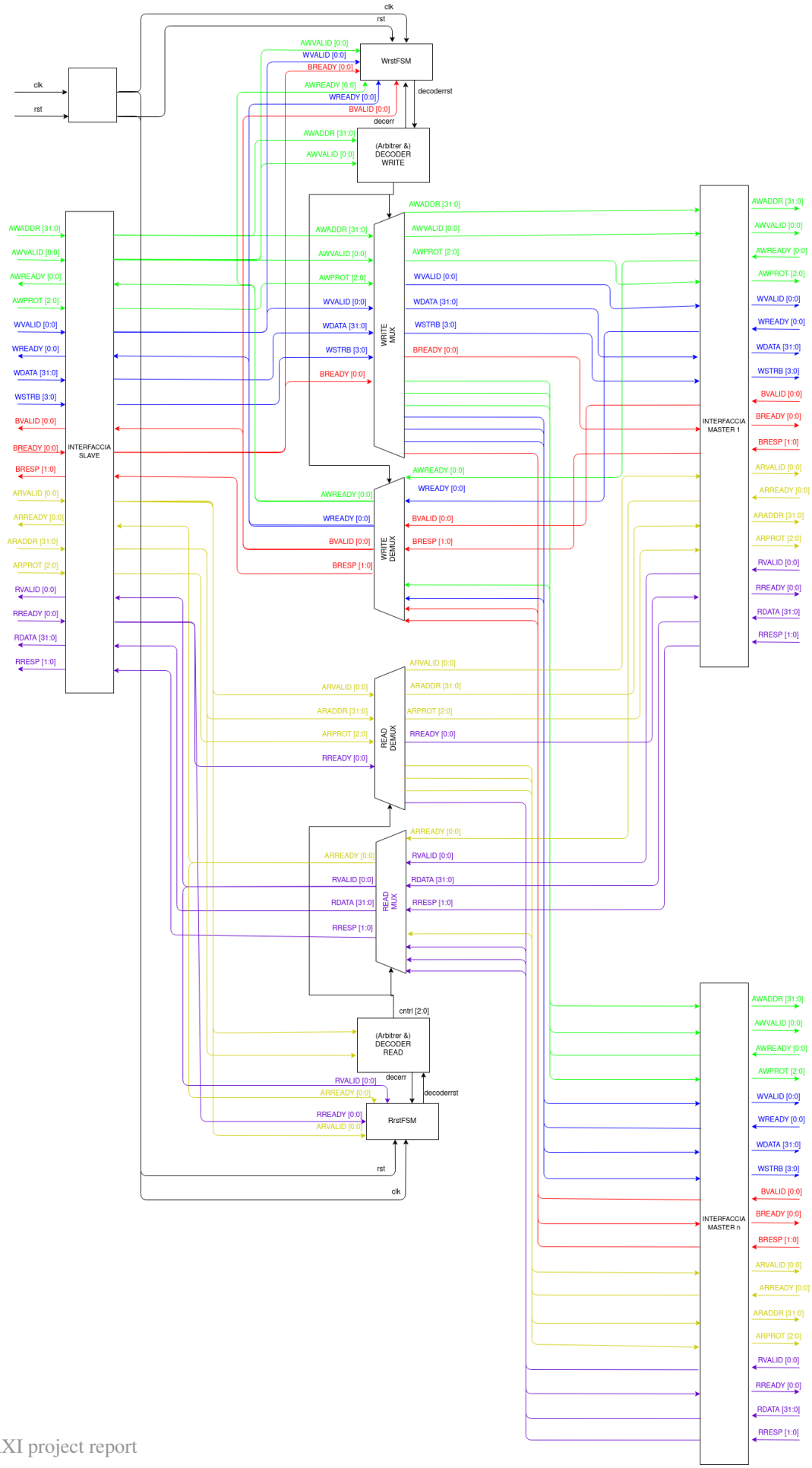


Figure 3: highLevel block diagram

3 Reading The OUTPUT

We read the output of the system by the uart -with hand method and software one-

4 Differences

The main differences that we found are:

4.1 Time Differences

Due to not having couplers

4.2 Signal Propagation

Our implementation is safer but more costly