OULUN AMMATTIKORKEAKOULU

Pauli Rekilä

Implementation of a Memory Interface

Implementation of a Memory Interface

Pauli Rekilä
Bachelor’s thesis
Autumn 2023
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Device and Product Design

---

Author: Pauli Rekilä
Title of the thesis: Implementation of a memory interface
Thesis supervisors: Teemu Leppänen and Teemu Pitkänen
Term and year of thesis completion: Autumn 2023 Pages: 44

The topic of the thesis was to develop an interface between the double data rate three (DDR3) memory controller provided by the field programmable gate array (FPGA) supplier and Detection Technology Plc's own logic. The need for the thesis occurred because the new product that the company was developing changed to a different vendor. The aim was to develop a new memory controller and create testbenches for it, which will be added to the existing test automation pipeline. The external memory interface block is part of a larger memory interface subsystem. It is possible that the memory interface can be used in future projects where the product of the same vendor is used, either as such or refined further.

The thesis briefly describes the very high-speed integrated circuit hardware description language (VHDL), the basics of field programmable gate array devices and double data rate three memory. The work includes a short literature review of the above-mentioned issues, as well as the reflections on them. Memory controller design process is described, and results are assessed.

A functional memory controller was created, which is utilized as a part of the company's product. Future use of the memory controller is a possibility as it is, or with small changes. In addition, several tests for the company's test automation pipeline were created, with the possibility of further refinement. Studies conducted during the project work also ease the use of newer double data rate standards in the future.

Keywords: FPGA, RTL, DDR3, SDRAM and VHDL.

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikkan tutkinto-ohjelma, tuote- ja laitesuunnittelu

Tekijä: Pauli Rekilä
Opinnäytetyön nimi: Muistirajapinnan kehittäminen
Työn ohjaajat: Teemu Leppänen ja Teemu Pitkänen
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 44

Opinnäytetyön aiheena oli kehittää rajapinta ohjelmoitava porttimatriisi toimittajan tarjoaman DDR3 muistikontrollerin ja Detection Technology oyj:n oman logiikan välille. Työlle oli tarve, koska uudelle tuotteelle, mitä yritys kehittää vaihtui laitetoimittaja. Tavoitteena oli kehittää uusi muistinohjain sekä luoda sille testit, jotka lisätään olemassa olevaan testiautomaatioon. Muistiohjain on osa isompaa muistirajapinta kokonaisuutta. On mahdollista, että muistiohjainta voidaan käyttää tulevissa projekteissa, missä on saman valmistajan tuote käytössä, joko sellaisenaan tai jatkokehitettynä.

Työssä kuvataan lyhyesti VHDL-laitteistonkuvauskieltä, ohjelmoitavien porttimatriisien sekä DDR3-muistin perusteita. Työssä esitetään lyhyt kirjallisuuskatsaus edellä mainituista asioista, sekä kirjoittajan omia mietteitä niistä. Työssä kuvataan sen jälkeen muistiohjaimen suunnitteluprosessi ja arvioidaan sen tuloksia.

Opinnäytetyön tuloksena saatiin valmiiksi toimiva muistiohjain, joka on käytössä yhtiön tuotteessa. Muistiohjainta on myös mahdollisuus käyttää sellaisenaankin, tai pienillä muutoksilla myös tulevissa projekteissa. Lisäksi syntyi useita testejä yhtiön testiautomaatioon, joita on mahdollista vielä jatkojalostaa. Myös uudempien DDR-standardien käyttöönotto helpottuu työssä tehdyn tutkimuksen perusteella.

Asiasanat: FPGA, RTL, DDR3, SDRAM ja VHDL.

ABBREVIATIONS

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
AXI	Advanced eXtinsible Interface
CDC	Clock Domain Crossing
CDR	Clock Data Recovery
CERN	Conseil Européen pour la Recherche Nucléaire
CI	Continuous Integration
CMOS	Complementary Metal-Oxide Semiconductor
DC	Direct Current
DDR3	Double Data Rate 3 (memory type)
DFE	Decision Feedback Equalizer
DLL	Delay Locked Loop
DSP	Digital Signal Processing
EMIF	External Memory Interface
FFE	Feed Forward Equalizer
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
Gbps	Giga bits per second
GSR	Global Set/Reset
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IP	Intellectual Property
JTAG	Join Test Action Group
LUT	Look-Up Table
LVDS	Low Voltage Differential Signaling
MAC	Multiply, Add and Accumulate
MST	Maximal Sustainable Throughput
MT/s	Mega Transfers per Second
PCB	Printed Circuit Board

PFU	Programmable Functional Unit
PHY	Physical layer
PLL	Phase-Locked Loop
QA	Quality Assurance
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SDRAM	Synchronous Dynamic Random-Access Memory
SERDES	Serializer/Deserializer
SRAM	Synchronous Random-Access Memory
SOP	Sum of Products
TCL	Tool Command Language
TFT	Thin Film Transistor
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit

TABLE OF CONTENTS

1 INTRODUCTION

The main objective of this thesis was to develop an external memory interface EMIF block, for the subscriber company Detection Technology Plc. The documentation of the project was also a vital part for the sake of future use of the EMIF block. The purpose of EMIF-block is to initialize DDR3 in powerup and take care of all reading and writing operations. It is so called glue logic between the vendor provided memory controller and proprietary logic of Detection Technology. EMIF block is part of the bigger ongoing development project of the company. The secondary goal of the project is to complement the authors learning in the DDR3 domain. This is accomplished by the literature presented in the thesis and with the help of colleagues and supervisors of the thesis.

Detection Technology is a global provider of X-ray solutions. It has a wide portfolio of different products including photo diodes, X-ray detectors including multi-energy and photon counting, ASICs, software, and algorithms. The author is working on Detection Technology as a firmware design engineer and the need for the thesis arose from the need to develop a new product. When the thesis started the author was quite novice about DDR3, but with the help of senior colleagues it was possible to tackle this problem. (15.)

The core part of the development was done using VHDL in all synthesizing parts of the design. This was done in the Lattice Semiconductor provided software package called Diamond. Simulations and testing were done using Make and TCL based scripts to run a Questa simulator developed by the Mentor Graphics corporation. Project management is done in Scrum style and software for that was Jira Atlassian and project was documented in Confluence which is also Atlassian product. Version control, continuous integration and quality assurance pipelines of the project was done with Gitlab software.

In this thesis the development process from specification to verification is described for the external memory interface block. It also illustrates the basics of the VHDL and basic flow of creating a working product from VHDL-code to the bitstream. It also explains shortly what Field programmable gate arrays are and what kind of building blocks they are made of. There is also introduced basic principles of DDR3 memory. The development process was a vast learning process for the author, and it could not have been finished without the extensive help of colleagues.

2 BACKGROUND

In this section there is a short literature review about the VHDL and how the basic workflow goes with it. Then we have a brief introduction about the FPGAs and what they are used for. The different components that they may contain are discussed next. The last part there is a brief introduction to the DDR3 SDRAM memory technology.

2.1 Very High-Speed Integrated Circuit Hardware Description Language

The VHSIC Hardware Description Language (VHDL) is a language for describing digital electronic systems. It arose out of the United States government's Very High-Speed Integrated Circuits (VHSIC) program in the 1980's. It is heavily influenced in both syntax and concept by the Ada programming language. (1.) After the VHSIC-program VHDL was subsequently developed by IEEE. VHDL is part of the IEEE standard 1076. (12.) Originally VHDL was created for simulating integrated circuits but nowadays it is used for both simulation and synthesis.

The original purpose of the VHDL was to model the behavior of existing hardware and not specify the functionality of proposed hardware. The behavior of the hardware can be modeled on multiple different levels. It ranges from system level behavior all the way down to gate level modeling. When VHDL was originally standardized in 1987 there were no automatic synthesis tools available in widespread use. Because of this the meaning of different VHDL constructs in hardware terms was derived a few years after the initial standardization of the language. The consequence of this is that some parts of the VHDL are not suitable for synthesis. (16.)

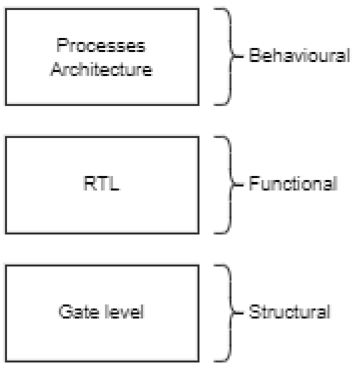


FIGURE 1. Levels of abstraction in VHDL

VHDL is meant to be used in every phase of the design cycle from the system specification to netlist. This results VHDL being large and heavy language. VHDL is in a way hybrid language containing features to all design phases that there is during design cycle. In the idealized design process, there are three phases. These are: system modeling or specification phase, register-transfer level or design phase and netlist or implementation phase. (11.)

2.1.1 Register Transfer Level

Register-transfer level design is a rather simple concept where a circuit is described as a set of registers and transfer functions. Registers are used to store the data and they are implemented as flip-flops. Transfer functions describe how data is processed between the registers and they are implemented as combinational logic. Translating the design into registers and transfer functions is a vital part of the RTL design process and an important part of RTL designer's work. RTL design can be separated into different stages.

The first stage of the design is to specify at the system level what is needed to achieve the wanted behavior of the design. The system level in this context is a higher level than RTL level. Then on the RTL level this is implemented as arithmetic and logic operations on data coming at the primary inputs of the design. At this stage there is no hardware design in mind, but the purpose is to create a simulation model that can later be used as formal specification of the design. With this model it can be verified with the customer that the specification is understood correctly.

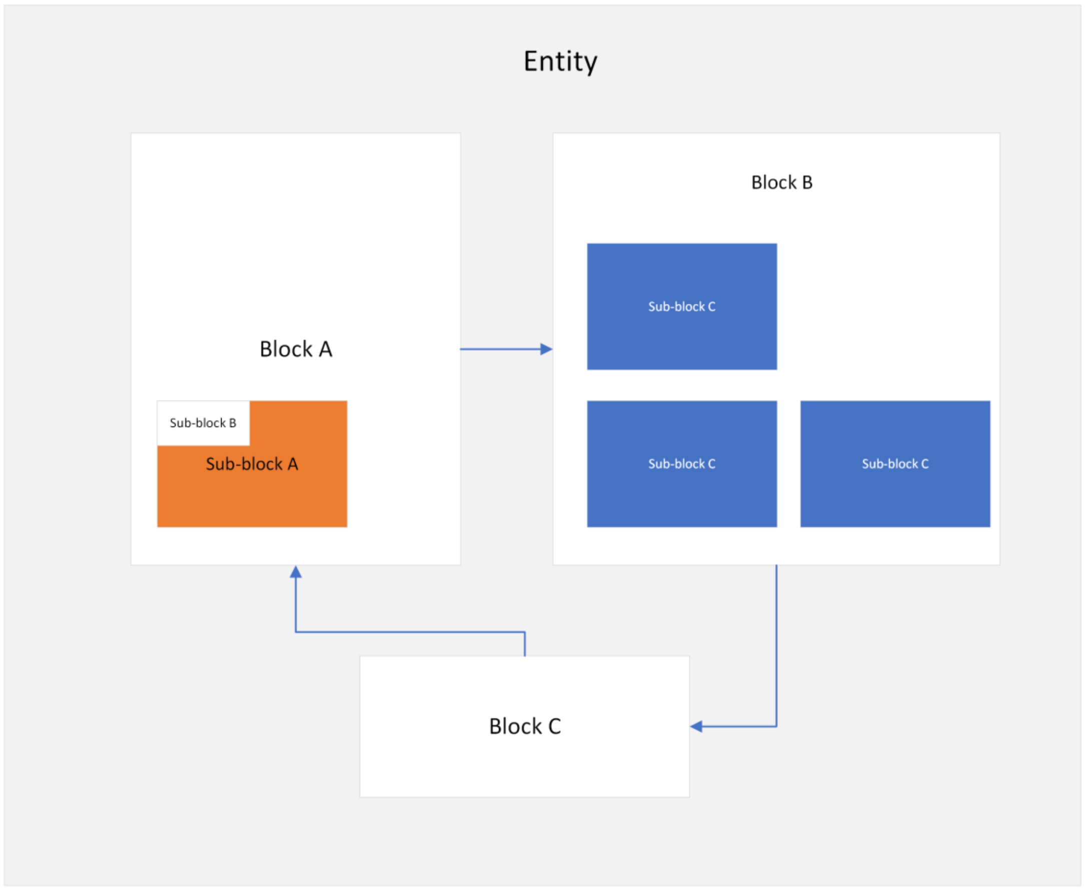


FIGURE 2. Visualized system level block diagram example

The second stage of the flow is to create an RTL design out of the system level model. Some of the basic steps are to identify the different data operations, determine the type and precision of the operations, decide what data processing recourses to provide, allocate operations to recourses, allocate registers of intermediate results, design the controller, and design a resetting scheme. After this comes the final stage of synthesizing the design. (11.)

2.1.2 Synthesis

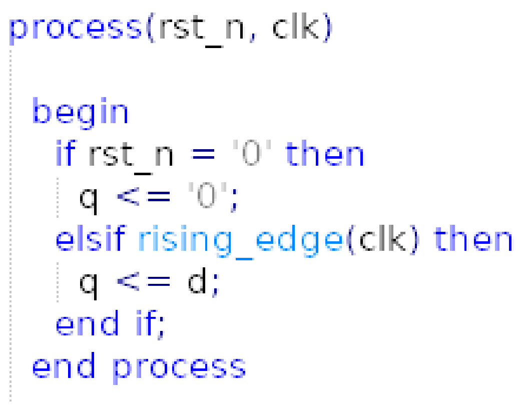
When the RTL design is finished the next step is to synthesize it. Everything in VHDL is not synthesizable and some of the standard types are shown in table one. In synthesis process the VHDL source code is converted into a netlist. A netlist is a list of logical elements and connections describing how these elements are wired together. A netlist is independent of vendor or technology where it is used. During the synthesis the tools provide a long list of warnings. Many of the presented warnings just tell what the designer intended to do but sometimes design errors can be found in these reports. (17.)

TABLE 1. VHDL standard types

Type	Class	Synthesizable
Boolean	enumeration type	yes
Bit	enumeration type	yes
Character	enumeration type	yes
Severity level	enumeration type	no
Integer	integer type	yes
Natural	subtype of integer	yes
Positive	subtype of integer	yes
Real	floating point type	no
Time	physical type	no
String	array of character	yes
Bit vector	array of bit	yes

VHDL was originally designed as a simulation language. Because of this matter synthesizing tools need to make an interpretation about the language. This interpretation is based on mappings of special VHDL constructs onto hardware with equivalent behavior. These constructs are known as templates. For example, there are templates for combinational logic such as registers with synchronous or asynchronous reset, latches, RAMs, ROMs, tristate drivers, or finite state machines. (11.)

It is important that designers write VHDL using these templates so the synthesis tool can synthesize it properly. Even if VHDL behaves correctly in simulation does not mean it is synthesizable if it is not written in a way that the tool understands it. In figure 3 there are a few different templates in use in the example code snippet. There is a rising edge triggered flipflop with low active asynchronous reset. There is a sensitivity list template which tells the process what it is sensitive to, in this case low active reset and clock signal referred as “rst_n” and “clk”. There is also an asynchronous reset template where the output value of flip-flop is set to zero when “rst_n” has a value zero i.e., low active reset is asserted. If reset is not asserted in the “if” clause then the state of the flip-flop is changed every rising edge of the clock, which can be seen in the “elsif” statement. (11.)



```
process(rst_n, clk)

 begin
  if rst_n = '0' then
   q <= '0';
  elsif rising_edge(clk) then
   q <= d;
  end if;
 end process
```

FIGURE 3. VHDL template example

2.1.3 Translate and Map

In the translate phase all netlists generated in synthesis are put to one big list for whole design and verify that every constraint maps to signals. If something is missing, it is detected in this phase. In Mapping phase, the netlist generated in translate phase is getting compared to resources of the FPGA. It is checked that there is enough and adequate recourses on the FPGA to proceed to the next stage. (17.)

Nowadays the performance of the chip is a big factor. Because of this, some modern mappers can change some components to better performing ones and pre-place some silicon recourses for better performance but keeping the same functionality. (17.)

2.1.4 Place and Route

Place and Route is a process where the algorithm first tries to place the silicon recourse and then routes the signals to other recourses. This process is iterative and is done for as long as the timing is met in the design. The constraints for the design come from the designer and they tell the maximum time it is allowed to spend between the recourses the algorithm is trying to achieve.

One of such algorithms is Pathfinder algorithm which was developed in the 90s. It was the first negotiation-based algorithm and to this day there are negotiation-based algorithms in use in different place and route tools.

The key approach of the Pathfinder algorithm is to over assign signals to nets that it routes. Then it negotiates the signals to different nets even if they have higher delay. This is done by using cost function where congested areas are given higher cost. This way signals can negotiate and find new ways with lower congested areas where they then get placed. The negotiation is done systematically, and the congestion is eventually eliminated. (18.)

2.2 Field Programmable Gate Array

Field-programmable gate arrays (FPGA) are integrated circuits (IC) which can be programmed numerous times. Most popular FPGA technology nowadays is SRAM based devices where the FPGA configuration is volatile. Meaning that when the device is powered off the configuration is lost. During every powerup this configuration is fetched from a non-volatile memory, for example flash memory and configured to the FPGA.

FPGA architecture is typically that on the edges there are I/O pins which connect the FPGA to the outside world. These pins can be either inputs, outputs or bi-directional. At the center there is distributed programmable interconnection matrix which provides interconnection of the logic blocks and connection to I/O pins. There may also be memory elements or DSP units. Simplified block level diagram of Lattice Semiconductor's ECP5 family FPGA is shown on figure 4. (20.)

Unlike conventional chips that have a fixed functionality, FPGAs can be programmed to adapt to different applications and requirements. They can also perform a great deal of parallel tasks and when that kind of requirement is needed, FPGA is a good solution. But if the performance of the circuit is critical then application specific integrated circuit (ASIC) might be a better solution. FPGA always loses to the ASIC because this interconnection matrix introduces plenty of routing delay compared to ASIC.

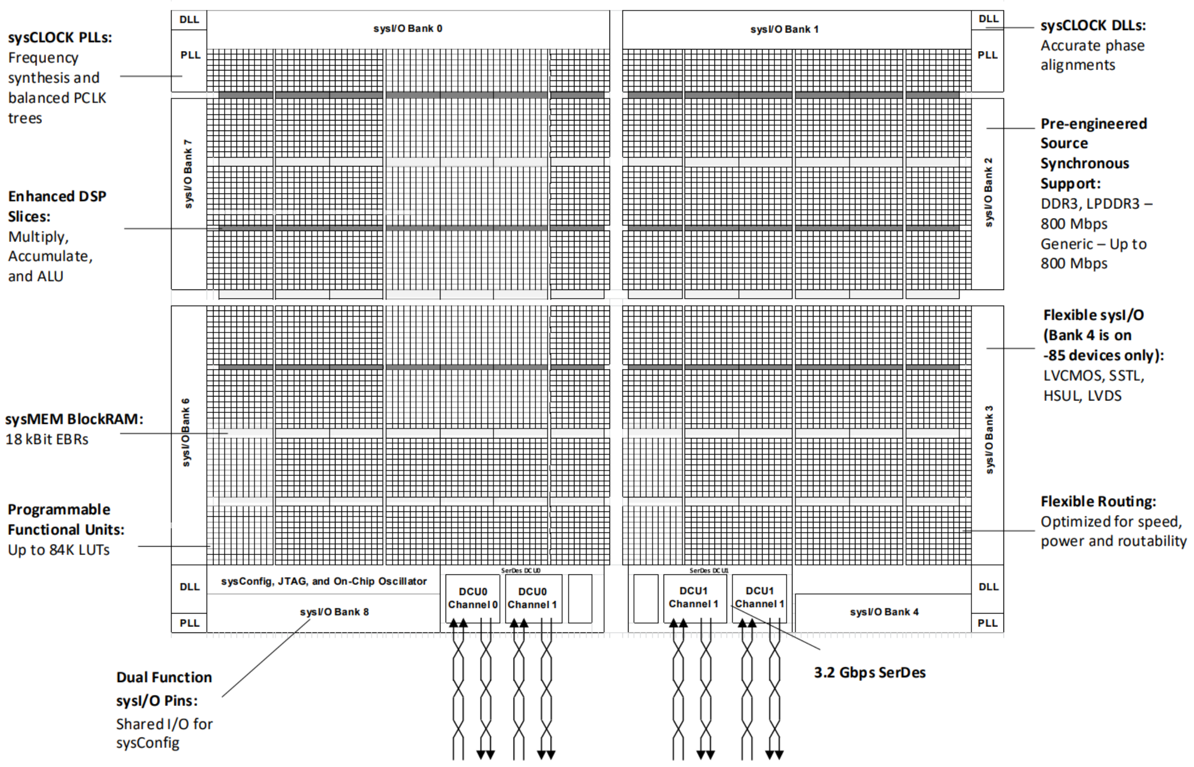

sysCLOCK PLLs: Frequency synthesis and balanced PCLK trees
Enhanced DSP Slices: Multiply, Accumulate, and ALU
sysMEM BlockRAM: 18 kBit EBRs
Programmable Functional Units: Up to 84K LUTs
Dual Function sysI/O Pins: Shared I/O for sysConfig
DLL
PLL
sysI/O Bank 0
sysI/O Bank 1
sysI/O Bank 7
sysI/O Bank 6
sysI/O Bank 2
sysI/O Bank 3
sysConfig, JTAG, and On-Chip Oscillator
sysI/O Bank 8
sysI/O Bank 4
DCU0 Channel 0
DCU0 Channel 1
DCU1 Channel 1
DCU1 Channel 1
3.2 Gbps SerDes
sysCLOCK DLLs: Accurate phase alignments
Pre-engineered Source Synchronous Support: DDR3, LPDDR3 – 800 Mbps Generic – Up to 800 Mbps
Flexible sysI/O (Bank 4 is on -85 devices only): LVCMOS, SSTL, HSUL, LVDS
Flexible Routing: Optimized for speed, power and routability

FIGURE 4. Simplified block diagram, LFEUM/LFEUM5G device (19.)

2.2.1 Lookup Tables

Lookup Tables or LUTs are a type of memory that can be programmed to perform a sum of product (SOP) logical functions. Organization of an LUT contains 2^n number of memory cells where n is the number of inputs. Basic AND, OR and exclusive OR (XOR) and their negative counterparts can be seen as two input lookup tables.

For example, a three-input lookout table can select eight different memory cells and therefore produce eight different SOP expressions. Consider a Boolean expression:

$$Q = \overline{A_2}\,(\overline{A_1}A_0 + A_1\overline{A_0}) + A_2A_1A_0 \qquad \text{EQUATION 1}$$

$Q =$ Output

$A_0 =$ First input

$A_1 =$ Second input

$A_2 =$ Third input

And convert it to sum of products term:

$$Q = A_2A_1A_0 + \overline{A_2A_1}A_0 + \overline{A_2}A_1A_0 + \overline{A_2}A_1\overline{A_0}$$

In the figure below is shown a three-input LUT programmed to produce SOP function described in equation 1.

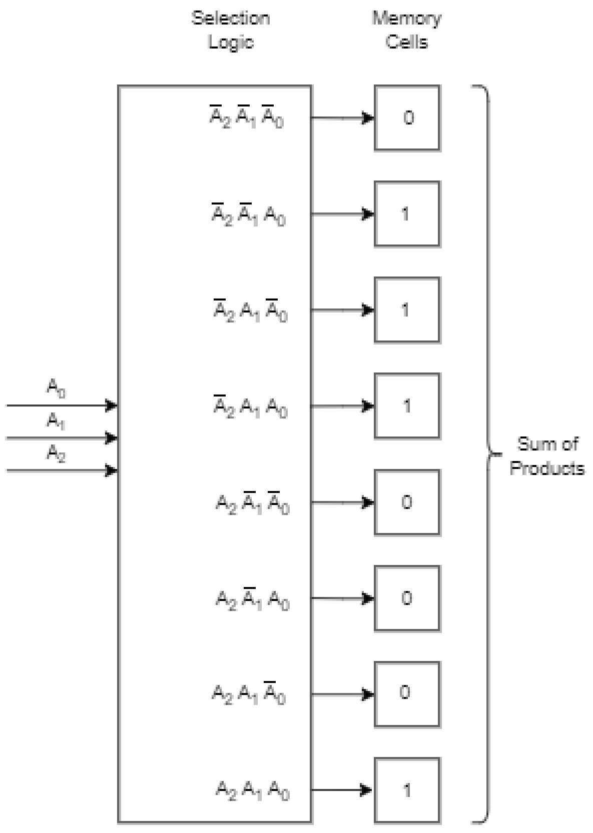

Selection Logic
Memory Cells
$\overline{A_2}\,\overline{A_1}\,\overline{A_0}$
$\overline{A_2}\,\overline{A_1}\,A_0$
$\overline{A_2}\,A_1\,\overline{A_0}$
$\overline{A_2}\,A_1\,A_0$
$A_2\,\overline{A_1}\,\overline{A_0}$
$A_2\,\overline{A_1}\,A_0$
$A_2\,A_1\,\overline{A_0}$
$A_2\,A_1\,A_0$
0
1
1
1
0
0
0
1
A_0
A_1
A_2
Sum of Products

FIGURE 5. LUT3 producing SOP for equation 1

2.2.2 Registers

There are two types of registers flip-flops and latches. They are bistable multivibrator devices which have two states SET and RESET which they can keep indefinitely. Registers are the simplest form of memory, and they can hold information of one bit. Registers can retain these states infinitely until the power is turned down. The main difference between flip-flop and latch is how they change their state. Flip-flops are edge-triggered meaning that they change state when specific conditions on input are met, for example a rising edge of the clock. Latches are level triggered and they change state when the input signal changes. (20.)

TABLE 2. Truth table for low active S-R latch

INPUTS		OUTPUTS		INFO
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
1	1	NC	NC	No chance
0	1	1	0	SET
1	0	0	1	RESET
0	0	1	1	Invalid state

TABLE 3. Truth table for rising edge triggered S-R flip-flop (note that falling edge has same truth table except for the falling edge of the clock)

INPUTS			OUTPUTS		INFO
S	R	CLOCK	Q	$\overline{Q}$	
0	0	Don't care	Q_0	$\overline{Q_0}$	No chance
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	?	?	Invalid state

2.2.3 Slices and Programmable Functional Units

Slices are small sub-blocks of which most of the FPGA area is used for. They contain different sized LUTs, registers, multiplexers, and routing elements. It is possible that they contain adders or shift registers also. They can be programmed in various ways during the FPGA powerup.

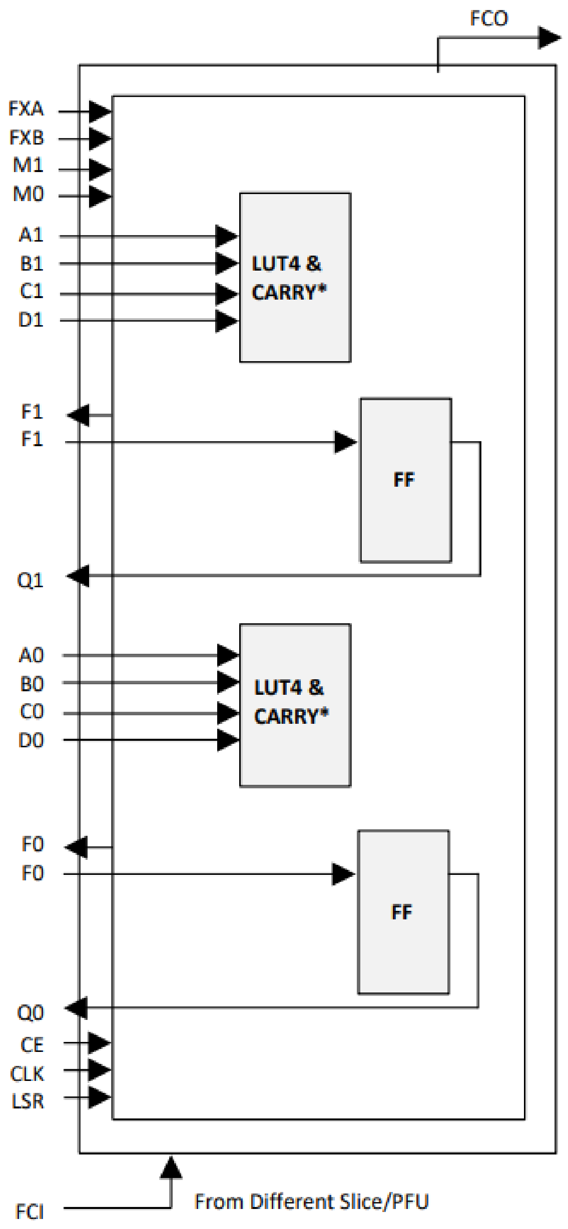


FIGURE 6. Lattice ECP5 Family Slice (19.)

PFUs are the core of the FPGA devices. They are the higher-level logic blocks which can contain varying amounts of slices depending on FPGA and vendor. PFUs can be used to perform logic, arithmetic, or ROM functions. They can also be used as distributed RAM. They can also be combined to generate bigger entities like LUT8. PFUs interconnected with each other on global row and column interconnects. These interconnects are the main reason for signal delays inside FPGA and the reason why ASICs have higher performance. (20.)

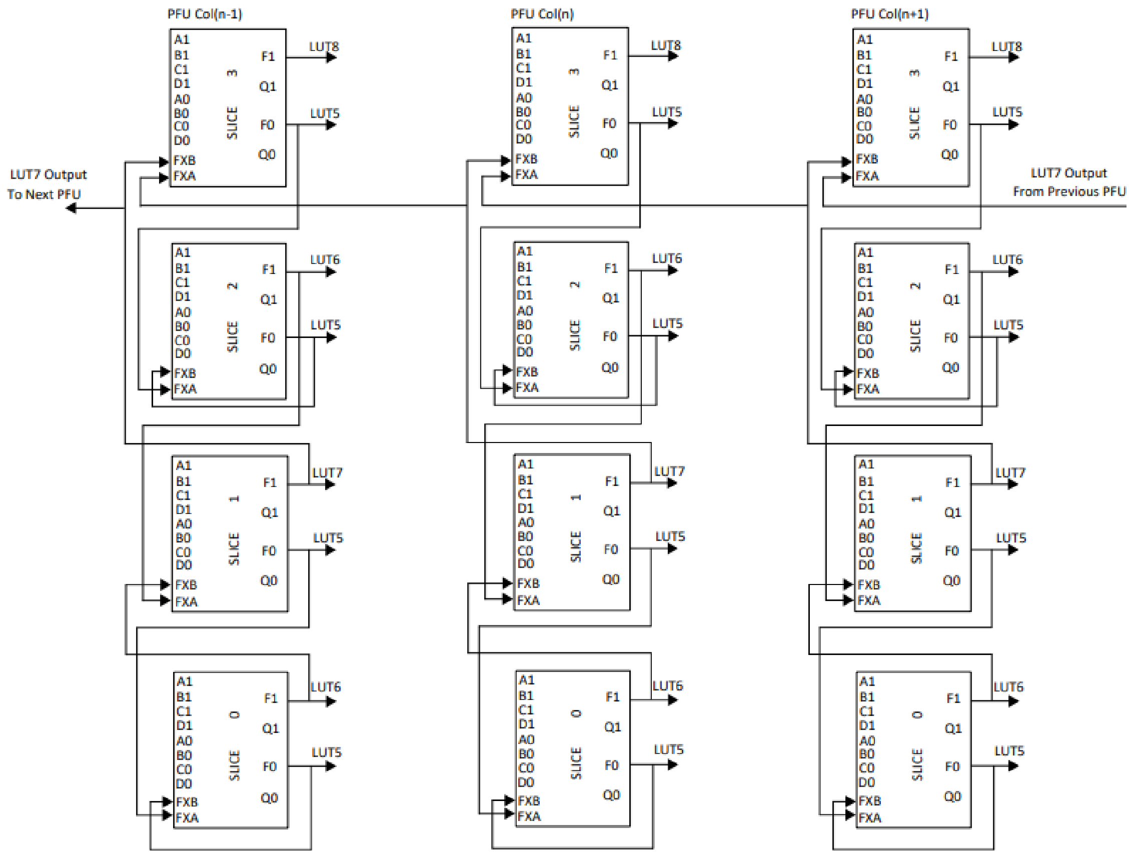


FIGURE 7. Connectivity of ECP5 Family PFUs (19.)

Naming conventions vary from vendor to vendor. Slice and PFU are used by Lattice Semiconductor. For two big players of the FPGA field: Intel and Xilinx (AMD) both have different naming conventions. Intel use adaptive logic modules ALM and logic array blocks LAB. Xilinx use slices and configurable logic blocks CLB. Comparing these building blocks is hard because depending on the manufacturer or even FPGA family these blocks can differ dramatically. They can contain different number of registers and LUTs with varying sizes. Also, other peripherals can vary a great deal. It is noted that the simple comparison of the number of slices or PFUs between FPGAs does not give good comparison between the devices. The designer should know which components are needed or what the final design should look like, so one can decide the best suited FPGA for the design.

2.2.4 Block Random Access Memory

Most modern FPGAs contain dedicated memory elements for storing data which are called block RAM. These special memory elements are needed because even if PFUs inside FPGA can be programmed to be memories they are inefficient if larger memory arrays are needed. For example, storing a 100-element array of 32-bit numbers can consume over 30 percent of midsized FPGAs area. On the other hand, it would consume only one percent of the same devices block RAM. (21.)

Block RAM can be used in different configurations. Most common configurations are single-port RAM, dual port RAM, pseudo dual port RAM, read only memory, RAM based shift register and single or dual clock FIFOs. (22.)

In this thesis the first in first outs (FIFOs) are discussed more closely because they are used in the actual design. The number one rule of FIFO is that empty FIFO should not be read, and full FIFO should not be written to. FIFOs can be used to buffer data if there are bottlenecks in the design, so data is not lost. Dual clock version of FIFO is also a very convenient way of crossing clock domains inside the design because the data can be written and read out with different clock. Usually, FIFOs have some status flags that help the designer’s job like full, empty, almost full, almost empty, read enable, write enable, write ready and read valid. Write and read enable are mandatory signals needed to exchange data with the FIFO, optional read valid and write ready signals are handy for implementing AXI-interface which is described later.

2.2.5 Serializer/Deserializer

There is a need for high-speed communication between chips nowadays because even though silicon density has grown according to Moore’s Law the PIN count of the devices has not kept up. This fact means that saving PIN usage of the device is crucial for the designers. Another thing to avoid parallel data transfer is that when there is a wide data bus and high-speed clock the timing of the design gets harder and more complicated. (23.)

When I/O interfaces start reaching data speeds of 2.5 Gbps and over high-speed serializer/deserializer (SERDES) is dominant implementation. The main difference between high-speed parallel clock and SERDES is that there is a clock and data recovery (CDR) circuit which determines the proper sampling point of the data. So, there is no need for an external clock, but the clock signal is embedded to the data. There are also feed forward equalizers (FFEs) to help with the signal integrity on the transmitter side and decision feedback equalizers (DFEs) on the receiver side. Data needs to be encoded for achieving DC-balance on the transmitting line, this is done by 8b/10b coding or better. After this it can be transmitted with differential pair. (23.)

2.2.6 Digital Signal Processing Blocks

Digital signal processing (DSP) blocks are dedicated components for arithmetic operations. The need for these blocks is a similar concept as why block RAM is used for memory. Even simple multiplying or dividing operations, when the factor is not power of two, consumes lots of logic and slows down maximum operating frequency. This results in the fact that the design needs to be pipelined to be able to keep up a higher frequency, but the pipelining has a disadvantage of consuming extra logic and introducing latency to the design. DSP blocks are a lot faster than logic, so they are needed in applications where near real time calculations are needed. They are used in a wide variety of applications, such as automotive, consumer, graphics/imaging, industrial, instrumentation, medical, military, telecommunication, and voice/speech applications. For example, DSPs can be used to compress data for efficient transfer and storage, recognize and generate speech, and clean up noisy audio or images. (20; 21.)

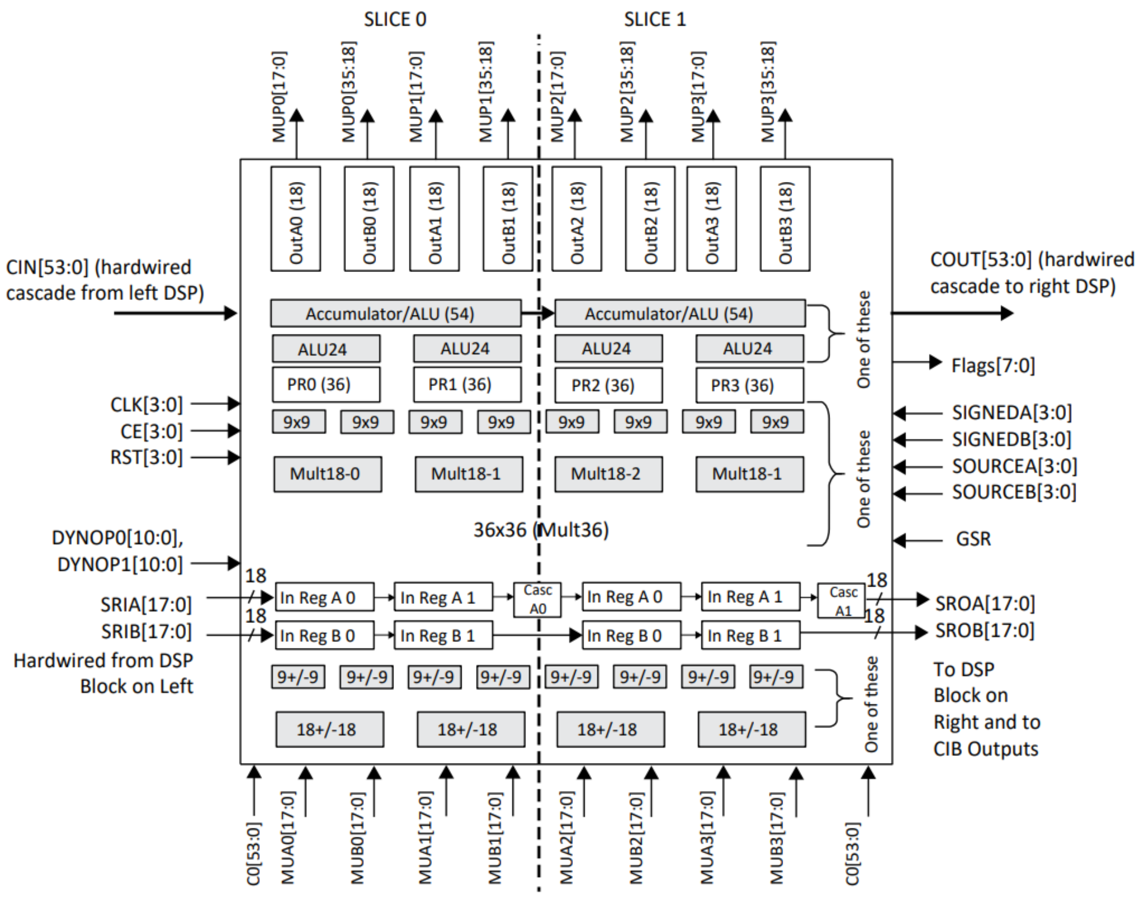


FIGURE 8. Simplified block diagram of Lattice ECP5 series DSP (19.)

Capability of DSP blocks vary between different FPGAs, and they are not necessarily found in every FPGA. DSP blocks can perform multiply, add, and accumulate operations or MAC operations.
Typical architecture of DSP block is that there is a multiplier which is connected to the adder which is connected to the accumulator or the arithmetic logic unit (ALU). Only capabilities of these units differ between FPGA families, for example bigger multipliers or ALUs. In some cases, DSP blocks may cascade to formulate bigger entities. They may be placed near block RAM, or some DSP slices may even contain dedicated block RAMs. Typical functions implemented in DSP blocks are multiply, MAC, multiply add, four-input add, barrel shift, wide-bus multiplexing, magnitude comparator, bitwise logic functions, wide XOR, pattern detect, and wide counter. These functions can create applications such as Finite Impulse Response filters, Fast Fourier Transforms functions, Correlators, Reed-Solomon/Turbo/Convolution encoders, and decoders. (19; 24; 25.)

2.2.7 Intellectual Property

Nowadays FPGA designs are so complex that it is not feasible to write every block from scratch. One solution to this problem is borrowed from traditional programming where one should reuse blocks that someone else has made before. There are three sources where these existing intellectual property (IP) blocks can be obtained. Building them internally inside the company, getting them from the FPGA vendor, or Third-party IP providers. They can be free to use, or they may have some licensing or other costs when acquiring them outside the company. IP blocks can be categorized into three different groups, soft, hard, and firm IP. (25.)

Soft IPs are source-level libraries of high-level functions that can be included in design. Typically, these functions are represented in some hardware description language like VHDL at the register transfer level of abstraction. All soft IP used in the design are incorporated into the main design and subsequently synthesized into a group of programmable logic blocks and may be combined with some hard IP blocks. (25.)

Hard Ips are the opposite of soft IP, and they are pre-implemented blocks such as microprocessor cores, gigabit interfaces or DSP blocks and the like. These blocks are made as efficient as possible, and they are incorporated into the FPGA whether they are used or not. (25.)

Firm IP is in the middle of the previous two. Firm IP comes in the form of high-level functions like the soft IP, but these functions have already been mapped, placed, and routed into a group of programmable logic blocks. They may also be combined with hard IP blocks. These predefined blocks can be instantiated in the design. (25.)

It is common that the IP blocks are also encrypted, this can happen even with the free IP blocks provided by the FGPA vendor to prevent migrating same designs to some other vendors FPGA. All IP providers have their own way of encrypting these IP blocks and there is no standard for it. IP blocks can also come with a generator script or program in which you feed parameters and select features. Then IP block with desired specification is then generated, this ensures an efficient way to create an IP block containing only the desired features. These blocks can be either routed and placed or not, they can also be encrypted. In case they are encrypted often simulation model is also provided with the encrypted block. (25.)

2.2.8 Backpressure

Back pressure is a phenomenon that occurs when there is more data coming that the device or circuit can handle. This can cause corrupt or lost data in the system. There are various ways to handle backpressure. Buffering the data to intermediate storage like RAM or FIFO where it can be consumed later. Regulating the data output to match the consuming speed of data recipient can help with the backpressure. This solution is not feasible in all scenarios if the data generator cannot be throttled due to various reasons.

Backpressure can be used also as a means of controlling the data flow in the system. Adding backpressure to the system also has drawbacks. When introduced to the system it decreases the maximal sustainable throughput MST (10.).

2.2.9 Advanced eXtinsible Interface Four Stream Protocol

AXI4-Stream protocol is the subset of Advanced eXtensible Interface (AXI) developed by Arm Holdings plc which was released in 2003. The AXI4-Stream protocol was introduced later in 2010. Both specifications are free to use, and full specification is provided by Arm (7; 8.). AXI is a widely used open interface standard in many IPs due to it being royalty free.

The AXI4-Stream protocol is a standard interface used to connect components for exchanging data. Interface can be used either with single master that creates data for single slave. Or it can be used to connect multiple masters to multiple slaves. Integral part of the AXI-standard is TREADY and TVALID handshake. Both signals need to be high at the same rising edge to exchange data. There are three different ways this can happen (figure 9). TVALID can be ready before TREADY or vice versa and they can be high at the same rising edge.

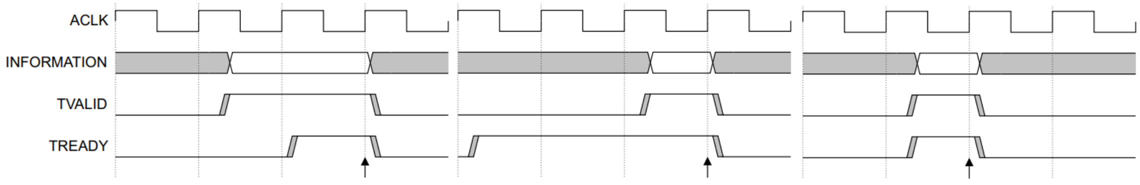


FIGURE 9. AXI handshaking schemes (8.)

2.2.10 Clock Domain Crossing

Metastability in digital systems occurs when two or more asynchronous signals combine in a way that ensuing output goes to an undefined state. A common example of this is a case of data violating the setup and hold specification of a flip-flop (4). Metastability occurs in every clocked system and there is no cure for it. It is not always the case that metastability causes problems in design, but it should still be taken care of. All registers in digital devices such as FPGAs have defined signal timing requirements that allow each register to correctly capture data at its inputs and produce an output signal.

To ensure reliable operation (figure 10), the input to a register must be stable for a minimum time before the clock edge (register setup time t_{SU}) and for a minimum time after the clock edge (register hold time t_H) (6). If these timings are not met signal goes to an undefined state where it is neither of voltage value of logical value zero nor one. (5.)

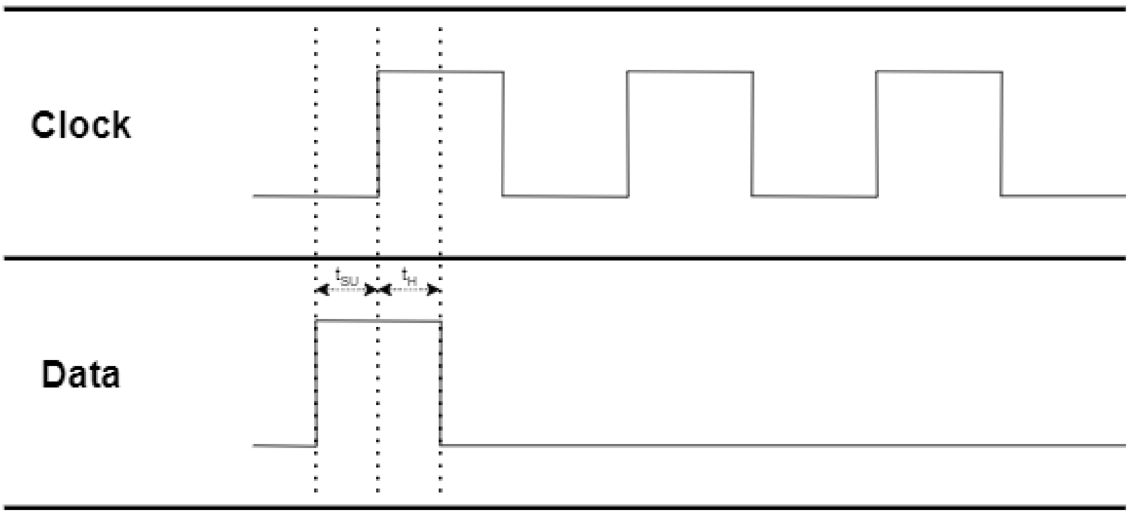


FIGURE 10. Setup and hold time

There is no fixed time interval sufficiently long that can guarantee that flip-flop have reached its defined state (3.). There are engineering techniques to deal with metastability, and the probability of metastable state decreases exponentially with the time the signal has for resolving its state.

$$f_{(r)} = e^{\left(\frac{-t_r}{\tau}\right)} \qquad \text{EQUATION 2}$$

$f_{(r)} =$ probability of nonresolution as a function of resolve time allowed

$t_r =$ resolve time allowed in excess of the normal propagation delay time

$$\tau = \frac{t_{r2} - t_{r1}}{\ln\left(\frac{N1}{N2}\right)} \qquad \text{EQUATION 3}$$

t_{r1} = resolve time 1

t_{r2} = resolve time 2

$N1$ = number of failures relative to t_{r1}

$N2$ = number of failures relative to t_{r2}

2.2.11 Resetting

Resetting is important on hardware to ensure that the device behaves correctly on the startup. In synthesized designs registers, counters, state machines etc. must start in a known state to ensure proper functionality for every startup. Designing reset and initialization is an integral part of the RTL design process and it is in developers' responsibility for proper resetting of the device. Nowadays many FPGAs have global set/reset (GSR) to help resetting. It can be used as both asynchronous and synchronous reset in most of the devices. There are two different ways to reset a device; with asynchronous or synchronous reset. (11.)

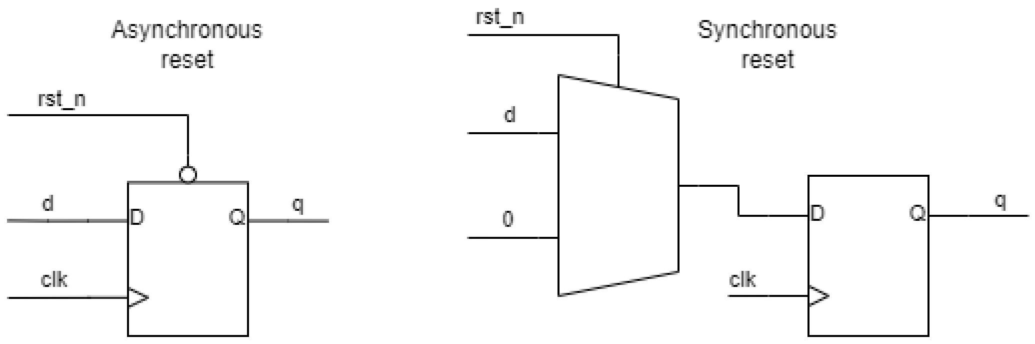


FIGURE 11. Hardware implementation of asynchronous and synchronous reset

Asynchronous resets act immediately, and they do not care about the clock. Usually asynchronous reset is a system-level reset and it should be driven by primary inputs of the device. They are very sensitive to glitches because the reset signal is active immediately. When using asynchronous reset the deassertion of reset should be synchronized to the clock. Reset pin must obey the setup and hold times relative to the clock pin or there is a chance of metastability. (11; 13.)

Synchronous reset takes place in the active clock edge. This results to the fact that there is no need for any special logic for managing the reset. In general, all resets inside the design which are driven by the internal logic should be synchronous resets. Synchronous resets are more resistant to errors because the clock can filter out small glitches which would affect asynchronous resetting practices. (11; 13.)

Both resetting schemes have their pros and cons, and it is for the designer to choose the proper implementation. Resetting schemes should be planned early in the design because it is a vital part of the design.

2.3 Double Data Rate Three

Double Data Rate 3 (DDR3) Synchronous Dynamic Random-Access Memory (SDRAM) is a memory technology that transfers data twice the speed of SDRAM. Data is transferred on both rising and falling edge of the clock. DDR3 is an older standard first introduced in JESD79-3A in September 2007. The latest version of the standard is JESD79-3F published in July 2012. DDR3 is widely used in the industry in various applications such as personal computers, various industrial products, consumer electronics or x-ray detectors. It has been adopted in many different products due to its relatively low cost but reasonable performance and power consumption. (14; 26.)

2.3.1 Double Data Rate Standards

At the time of writing this thesis DDR is in its fifth generation DDR5 which was released in the year 2020. Compared to previous generations every generation is faster and consumes less power than the previous one. It is already announced that the sixth generation of DDR is being developed. There are also sub standards for DDR for example DDRA-BBBB, DDR3-800 or DDR5-6400, where A denotes the generation and BBBB denotes the data transfer rate MT/s. (14.)

DDR3 also has two extensions to it which are DDR3L which stands for low voltage where reference voltage is 1.35 V compared to normal reference of 1.5 V. The second extension DDR3U stands for ultra-low voltage, and it has reference voltage of 1.25 V. Both extensions are fully compatible with DDR3 standard. Note that there is also low power double data rate3 (LPDDR3) which is an entirely different standard and not compatible with DDR3, same goes with different graphics double data rate (GDDR) standards. (14.)

2.3.2 Architecture

There are several different architectures in DDR3, but they are very similar generally speaking. The biggest difference is in the number of data strobes i.e., width of the data interface. In this example a 16-bit interface is used. Memory can be seen as a two-dimensional array of rows and columns and those arrays form a third dimension where there are address banks. The main parts are control logic where all the command signals go, for example row address strobe, column address strobe, chip select, and write enable. Then there is the address register which is divided into three address lines for rows, columns, and banks. Address lines are connected to the respective multiplexers and at the bottom of each row there are sense amplifiers which do the actual reading of the data. Then there are 16 DQ pins or data strobes what are the physical interface to the DDR3 chip.

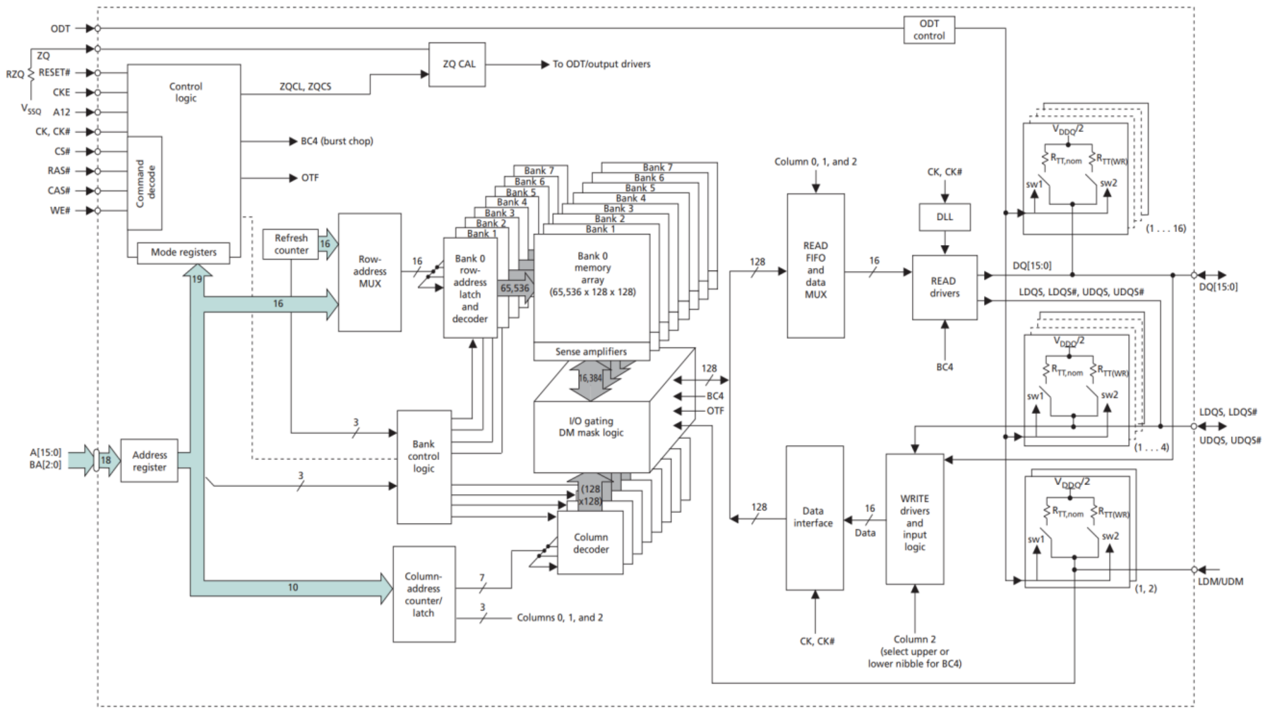


FIGURE 12. Simplified 512 MB x 16 DDR3 Functional Block Diagram (26.)

Due to the demand of high-speed memory, a so called 8n-prefetch architecture is used in DDR3. This means that every read or write operation effectively consists of a single 8n-bit-wide data transfer. This means that in an interface where there are 16 pins the smallest data word that can be accessed from the memory is sixteen times eight which equates to a 128-bit wide data word. (14; 25.)

2.3.3 Need for Dynamic Random Access Memory

Increasing demand for high-speed memory capacity led to the development of dynamic random-access memory (DRAM). The demand to get higher density RAM meaning more bits per chip, chip designers invented memory cells that could use only one transistor per bit. In SRAM technology a memory cell is formed by a D-latch which utilizes four to six transistors per bit (20; 26.).

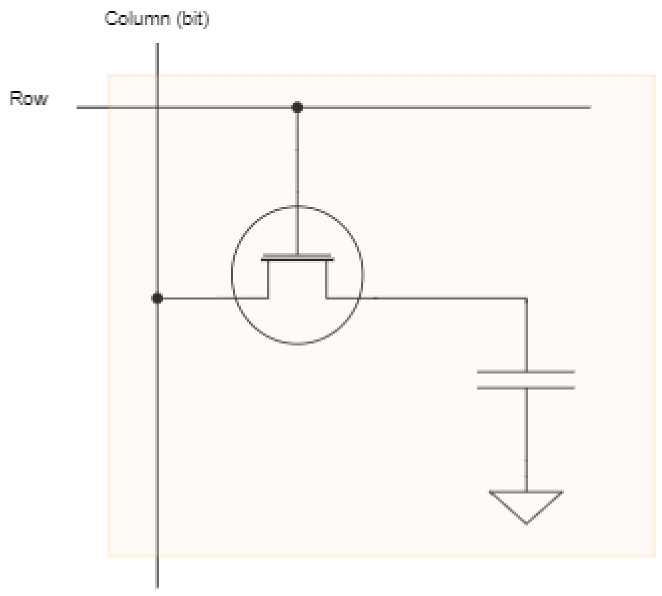


FIGURE 13. One-bit MOSFET DRAM cell

It is not possible to build a bistable multivibrator with just one transistor. Instead, memory cells in DRAM store information in tiny capacitor accessed through metal-oxide-semiconductor field-effect transistor (MOSFET). When the capacitor is charged, it represents the logical HIGH and when it is empty, it represents LOW. This makes DRAM volatile and over time and temperature capacitors leak out current and they need to be periodically refreshed. This means that the capacitors need to be read and then written back the value they contained to preserve the data. The usual refresh period of DRAM cell is 64 milliseconds which means that it gets refreshed about 16 times a second. It should be noted that reading the DDR is a destructive operation and if data needs to be preserved then it needs to be written back to the memory after reading it. (26.)

3 DESIGN

The subject of this thesis was to create a logic between the vendor IP and company logic for controlling the DDR3. This block is a subblock of a bigger entity that is controlling DDR3 memory related tasks. The work was split into four parts. The first part was to ensure that the vendor IP was working and DDR3 pins were correctly routed in PCB. The last three parts were to create specification, produce implementation according to the specification and finally verify that implementation fulfills the specification.

3.1 Vendor IP and DDR3 I/O Validation

The first iteration of validation work started by doing simple data writer and reader block using VHDL. The block starts by writing to DDR3 one full row from the first bank and after the first bank is completed it reads the data back from the same bank and row and compares that both data are the same. Then it writes the next bank of the same row and compares data again. It proceeds this writing and reading pattern through whole DDR3 memory address space. Data used to write to the memory was alternating data pattern between even and odd bank numbers described on figure below.

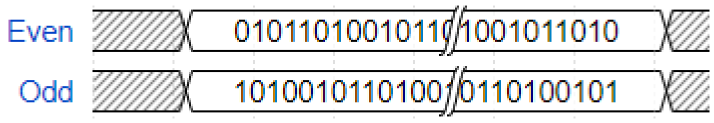


FIGURE 14. Data pattern on even and odd rows

There were also implemented different modes for testing that the data mask is working. Data mask was tested by using data mask for alternating data nibbles between first and last nibble. Even banks were written with first nibble and odd banks with last nibble.

Results that the write and read data were correct in DDR3 were verified by using debug cores which were synthesized to design and then reading them with vendor provided software and using JTAG-debugger which was also provided by the vendor.

3.2 Specification

Specification work was started before the actual implementation of the EMIF block. First it was defined what requirements EMIF block should fulfill. After consideration two (major) higher level requirements emerged, which were handling initialization of the DDR3, and handling data reads and writes to the DDR3. EMIF block is part of the bigger Memory Interface (figure 15). EMIF gets data with the address and stores it to the DDR3 outside of the FPGA. It also handles retrieving the data from the DDR3 memory when it gets read address.

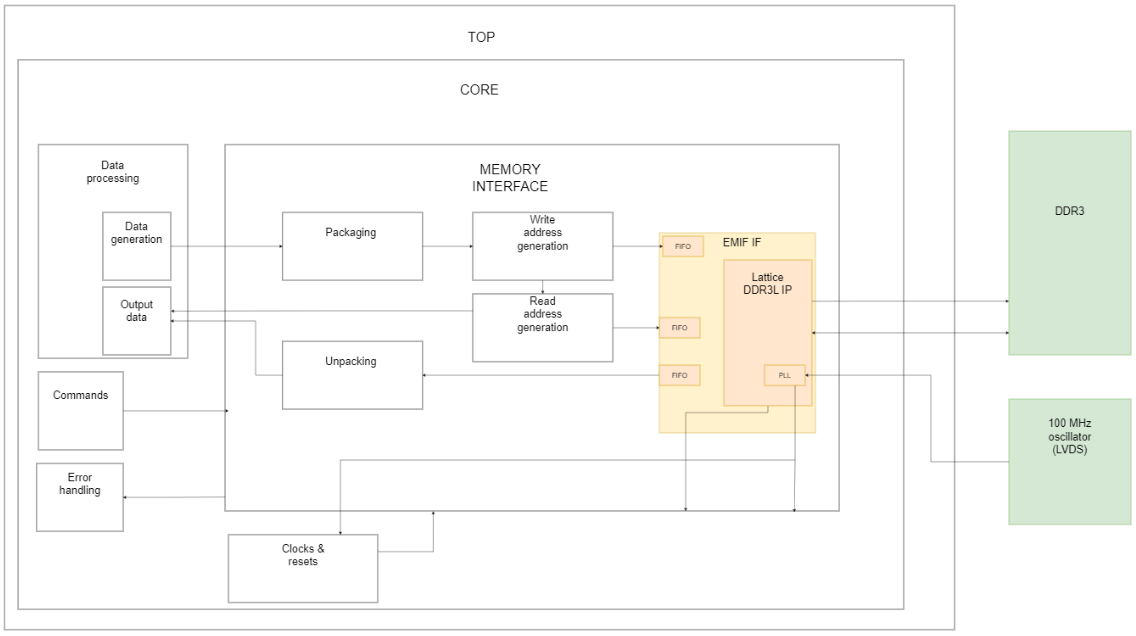


FIGURE 15. Top level diagram of Memory Interface

The design also needed to be AXI4-stream compatible because the rest of the design used the same data streaming protocol.

3.2.1 Specifying the interface for the EMIF block

First it was defined what interfaces EMIF block should have. EMIF interface was quite large since there were many pins on it and some of the pins were routed all the way to the top level of design because some of the pins were interfacing with the physical DDR3 pins on the PCB. On the right side (figure 16), it can be seen which I/O pins are facing DDR3 pins.

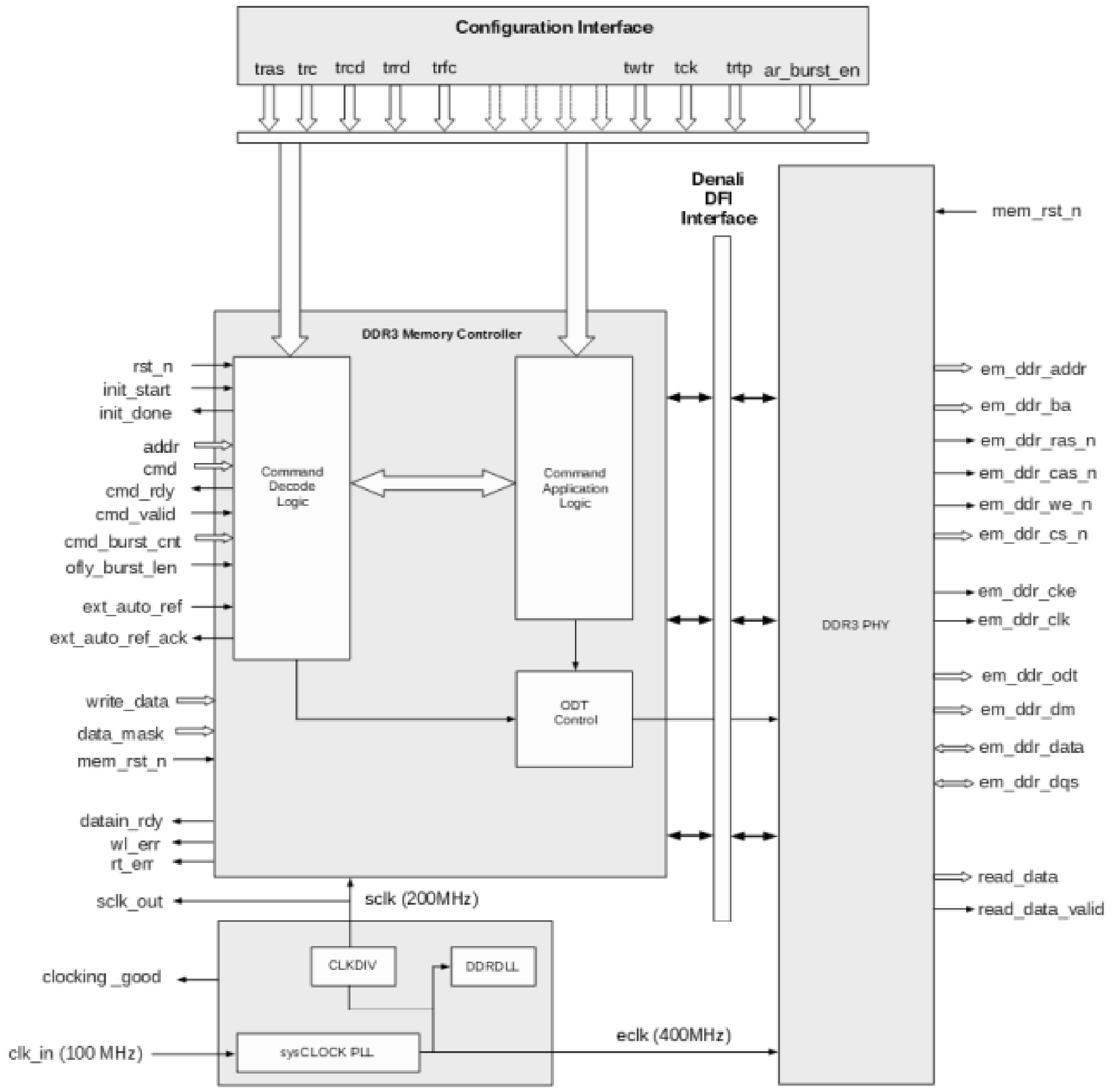


FIGURE 116. DDR3 SDRAM controller block diagram (9.)

EMIF block needs to interface with DDR3 memory controller side to write and read data out of DDR3. Those signals are internal FPGA signals that are not physical I/Os. One exception is signal “clk_in” which is coming from an external oscillator outside of the FPGA.

The main purpose of EMIF block is to interface with DDR3 IP but there are also other signals in its interface. The previous block outputs pixel data stream to EMIF which is written to DDR3 and the next block which sends data stream out of the device via ethernet. Data stream moves in and out of the EMIF block as AXI4-stream so there are “tready”, “tvalida” and “tlast” signals alongside the data bus. There are also a few error- and status signals inside the block.

3.2.2 Data flow protocol

It is important that EMIF block can process a great deal of data and keep the data integrity from start to finish. In EMIF block it was decided that AXI4-stream is used. It was a straightforward decision because it was basically the only option since data is streamed with the same protocol elsewhere in the design. To be noted that there needed to be some extra logic because the DDR3 IP, provided but a third party was not AXI4 compliant. The AXI4 interface was violated because the IP only sends acknowledgement when it is ready to write or read valid data, but the other part of the handshake signal was missing. There is half of the usual handshake signals missing and "TLAST" signal is missing from the interface. The before mentioned signals all are inside the EMIF block, so inter-block communication still works with AXI4 standard.

3.2.3 Handling clock domain crossing

Handling clock domain crossing correctly is a vital part of the design. There are three different CDCs in the EMIF design. The first one is when data is coming inside the EMIF block from the previous block. This CDC is handled with dual clock FIFO where the write side of the FIFO is driven by previous blocks slower clock and read side of the FIFO is driven by faster clock utilized in the EMIF block. The second CDC is the opposite of the first one where EMIF sends data to the slower clock domain, and it is also handled with dual clock FIFO.

Third CDC is a little bit tricky because it does not show itself to the designer that easily. It is between the DDR controller and DDR PHY part of the DDR3 IP. This CDC is hidden from the designer because it is inside the obfuscated/crypted IP block which handles said CDC. Even though the designer does not have to deal with this at VHDL-level, it is good to understand that this might still affect the timing constraints of the design.

3.3 Implementation

The task for implementing EMIF block was relatively simple since most of the code is just instantiating different IP components from the vendor and interconnecting them together. This was possible because of the extensive design work. Also, there was a similar component designed in the previous project which acted as a starting point for the design.

The first part of the design was to handle the CDCs in the design because incoming pixel data comes from the slower time domain than what the DDR3 is running. The simplest solution was to use dual clock FIFOs to take data in and buffer it to make the design more robust. Because of the FIFOs which synchronize the data there are needed two different clocks in the design. For the FIFO implementation it was decided to use the readymade vendor IP. This was decided because those IP blocks are already tested and verified by the vendor, so it saves time for the designer and more likely the outcome is better than designing the FIFOs from scratch.

The second part was to design all the control logic that EMIF block needs. In the control logic both requirements needed to be fulfilled. They were implemented in two different processes within the VHDL code. Memory initialization is a simple task from the controller logics point of view. Memory initialization is started when “init_start” signal is asserted high by the control logic. Once it is asserted, it needs to be held high until the initialization process is finished and “init_done” is asserted high for one clock cycle. It is important to deassert “init_start” on same clock cycle than “init_done” is deasserted or otherwise new initialization cycle begins. Below there is a timing diagram of the successful initialization sequence.

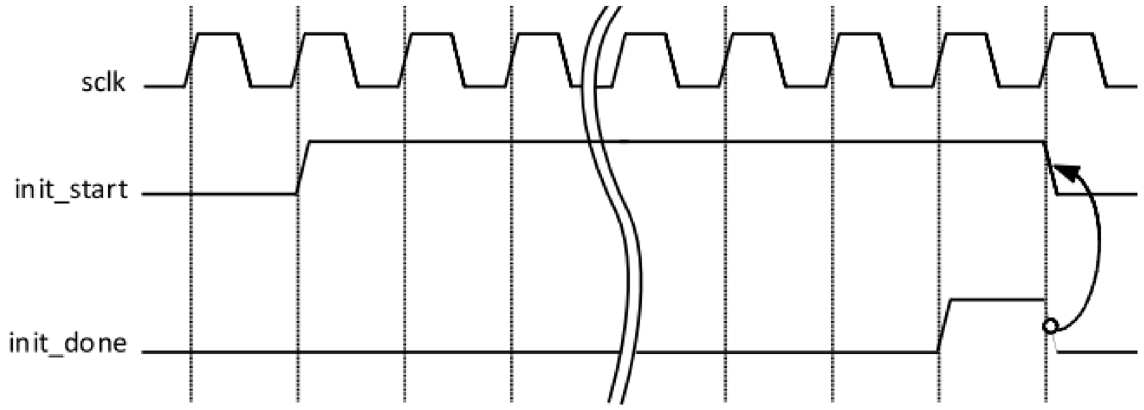


FIGURE 17. Timing of the Memory Initialization Control (9.)

Writing and reading from DDR3 is done by sending adequate control signals to the vendor provided IP block. It was decided to start with basic write and read commands even though the vendor IP offers more elaborate commands that can increase the throughput of the DDR3. This is not necessary since there are bottlenecks elsewhere in the system and DDR3 is not the critical part when assessing the throughput on system level. There is still a possibility to speed up both writing and reading in burst mode. The DDR3 IP supports burst writing or reading up to 32 data words per command.

There is a state machine in logic handling the writes and reads to the DDR3. There are small input FIFOs for both commands and data words. These FIFOs are single clock versions of the vendor IP, and their main purpose is to compensate if some parts of the data pipe are stalling so no data is getting lost.

3.4 Verification

After the initial I/O validation test, some testbenches for memory interface were written. There is no block level testbench for EMIF block, but it might be done in the future. The higher-level tests with the memory interface have provided enough coverage to find critical bugs affecting the day-to-day use of the device. All tests were made self-checking so they could be run in the continuous integration and quality assurance pipeline.

Simulation time of the EMIF block was quite slow because there was a behavioral model of the DDR3 designed by the vendor in addition to the DDR3 controller model made by the FPGA vendor. These two models needed ten picosecond, high resolution, in the time domain which slowed simulation down drastically. Due to this there were also created a SystemVerilog simulation model for EMIF block to speed up simulations.

3.4.1 Scripting of simulations

Simulation flow of the project was twofold. Simulations were started in a way that higher level Make script calls for the lower-level Tool Command Language TCL scripts. The reason for using two different scripting languages was that the Make based scripting is needed for the CI and QA pipelines. On the other hand, the same simulations are also used by the designers, and they want to use TCL based scripts because that is a more convenient way to control simulators.

3.4.2 Simulation model and actual implementation mismatch

During the testing phase when the first simulations were done, it was noticed that there was mis-match between simulation model and real-world implementation in the resetting scheme of the device. It was because at the first iteration of simulation model all reset signals were released simultaneously which caused issues in other parts of the design. This was done to speed up the simulation, but it went too far. After consideration to reach better parameters, 40 nano second delays were added between the release of resets in different blocks.

4 RESULTS

The original task of developing an external memory interface or so-called glue logic for the vendor provided memory controller IP was successful. EMIF block was able to fulfill all its main tasks which were initialization of the DDR3 and performing both read and write operations.

4.1 Simulation

After the RTL part of the EMIF block was finished, implementation of the testbenches was started. There were some simple simulations composed parallel with the implementation phase to ease up the design work. After the implementation phase, serious development of the tests started, thinking about corner cases and the coverage of the whole test set. Test sets were incremented iteratively by increasing the number corner cases and adding different stimulus variations. EMIF block was tested as part of the bigger entity (figure 15) to cover the whole pipeline from data input to the DDR3 and towards the next block.

A whole memory interface was also added to the top-level simulations where the whole system is under the test. Soon it was realized that this was slowing down simulations too much. This was because the simulation model of the vendor provided IP and simulation model of the DDR3 memory both needed high, ten picosecond resolution to be simulated correctly. On the top-level this resulted in too long simulation times. This was resolved by a colleague who wrote a SystemVerilog model of the EMIF block which could be simulated with the lower, one nanosecond resolution. Integrating this change into the top-level simulations it had a substantial increase in the simulation speed.

4.2 Discussion

During the implementation of the tests there were two major aspects that were focused on, which were the data interface and functionality of the block. The interface is important to be tested thoroughly to ensure data integrity throughout the pipeline. This was achieved by toggling ready signals with randomized pattern and using interface validators to ensure that the last and valid signals are handled properly. Functionality of the EMIF block was validated by writing known data to the DDR3 and comparing it to the data which was read out from the same address. From writing and reading could also be concluded that the DDR3 has been initialized properly since the data was correct.

The final version of the EMIF block took some iterations before an adequate design was achieved. In the first version of the state machine there was only one FIFO, but it was quickly seen that the one FIFO implementation would not work due to the unfortunate fact that the DDR3 IP block takes data few clock cycles after the command is written to the control logic. This was resolved by splitting the FIFO into two smaller ones where one handled the data and the other handled the address of the corresponding data. There was also a small mischief i.e., the first version did not flush FIFOs between the data acquisition and there was risk of storing unwanted data in the FIFOs. There was a trivial fix for this by just resetting the FIFOs between the acquisitions.

There are possible improvements within the EMIF block. When it was first designed all the write and read commands are written one command at a time for the vendor provided IP. This was done to keep the design as simple as possible and there was no downside because the throughput was enough for the incoming data stream. If higher throughput is needed in the future, it is trivial to change to burst commands to the vendor IP. Vendor provided IP supports up to 32 consecutive write or read addresses burst with one command. (9.)

There were also observed situations where simulation model of the EMIF block failed to find or reproduce bugs. This was because it was just a model for speeding up the simulations. It was not a cycle-accurate one-to-one model of the EMIF block. It would defeat its original purpose if it were this accurate and would slow down simulations again. To get around this problem, there is a smaller subsystem unit test with real DDR and vendor IP models.

5 CONCLUSION

The subject of the thesis was interesting and challenging. There were many tasks involved in the design process where both soft and hard engineering skills were needed. There was a vast number of different tasks needed to perform the work in the thesis. Doing RTL design with VHDL, debugging simulation code in Verilog and SystemVerilog, planning and documenting project work, setting up simulation environments with different scripting languages, and collaborating with teammates to name a few. During the thesis, I was able to learn a lot about all the subjects and I would like to thank my current and former teammates Jarmo Heino, Mark Hawkins, Puneet Sinha, Kimmo Ranta, Kim Berg, Oualid Saidi and Teemu Pitkänen.

At the start of the thesis, I did not have much experience working with DDR3, but with the meticulous reading of datasheets, simulating the design, and interviewing of my colleagues I was able to grasp at least basic principles of how DDR3 works. The subject is so complex and there is still lot to learn, but understanding is in the level that basic RTL design can be done and configuring the DDR3 is possible. One would assume that knowing the basics of DDR3 gives a solid foundation if one needs to work with different DDR standards in the future.

It also opened a comprehension to the author how much “soft” skills are needed during the project work. Developing a complicated design is much more than just writing the RTL code and ending the day´s work. Specifying and documenting all the different components and functionalities in an unambiguous way is important. Also, one should demand that the specifications given to the designer are accurate enough. This is to guarantee that a properly functioning device, which is fulfilling the needs of the stakeholders adequately, can be designed.

When considering the whole project during the thesis it went well when looking at the big picture. It was helpful to prepare for the task thoroughly, even when at the time it felt that there was no progress made, when reading the data sheets over and over again. After studying, simulating a reference design helped to assimilate how DDR3 was used in practice. Even though the overall progress of the thesis went well, there was room for improvement. After the specification and implementation phase, the author was quite busy with other work and the writing part of the thesis was on hold for several months partly due to the poor time management and planning by the author. With more accurate and determined planning the thesis could have been finished earlier.

In conclusion, this thesis fulfilled its objectives which were set at the beginning of the project by producing a functioning solution. The knowledge acquired in this thesis provides a solid foundation for further development and refinement of DDR3 related tasks. This thesis has not only been a means to an end, but also a significant milestone in the never-ending learning in a fast-paced field like information technology.

REFERENCES

1. Department of Defense 1990. Very High Speed Integrated Circuits (VHSIC) Final Program Report. Accessed 4.9. 2023. https://apps.dtic.mil/sti/pdfs/ADA230012.pdf
2. Ashenden Peter J. 2008. The Designer's Guide to VHDL. Third edition. Croydon: Morgan Kaufmann publications.
3. Chaney Thomas, Molnar Charles 1973. Anomalous Behavior of Synchronizer and Arbiter Circuits. Accessed 19.9. 2023 https://ibm-1401.info/AnomalousSynchronizer_Chaney-Molnar_IEEE1973.pdf
4. Wellheuser Chris 1996. Metastability Performance of Clocked FIFOs. Accessed 19.9. 2023 https://www.ti.com/lit/an/scza004a/scza004a.pdf
5. Dally William 2005. Lecture notes for EE108A, Metastability and Synchronization Failure (When Good Flip-Flops go bad). Accessed 19.9. 2023 https://view.officeapps.live.com/op/view.aspx?src=http%3A%2F%2Fcva.stanford.edu%2Fpeople%2Fdavidbbs%2Fclasses%2Fee108a%2Fwinter0607%2520labs%2Flect.9.Metastability-blackschaffer.ppt&wdOrigin=BROWSELINK
6. Altera/Intel 2009. White Paper: Understanding Metastability in FPGAs ver. 1.2. Accessed 20.9. 2023 https://www.intel.com/content/www/us/en/content-details/650346/understanding-metastability-in-fpgas.html
7. ARM limited 2003. AMBA AXI Protocol Specification v. 1.0. Accessed 22.9. 2023 https://documentation-service.arm.com/static/5f915920f86e16515cdc3342?token=
8. ARM limited 2010. AMBA 4 AXI4-Stream Protocol Specification v. 1.0. Accessed 22.9. 2023 https://developer.arm.com/documentation/ihi0051/a
9. Lattice Semiconductor 2020. Double Data Rate (DDR3) SDRAM Controller IP Core User Guide. Accessed 22.9. 2023 https://www.latticesemi.com/view_document?document_id=36115
10. Carloni Luca 2006. The Role of Back-Pressure in Implementing Latency-Insensitive Systems. Accessed 24.9. 2023. https://www.sciencedirect.com/science/article/pii/S1571066106000247.
11. Rushton Andrew 2010. VHDL for Logic Synthesis. Third edition. Chichester John Wiley & Sons Ltd.
12. 1076-1987 1987. IEEE Standard VHDL Language Reference Manual. Accessed 24.9. 2023. https://doi.org/10.1109/IEEESTD.1988.122645. Requires license.

13. AMD Inc. 2023. Versal Adaptive SoC Hardware, IP and Platform Development Methodology Guide (UG1387). Accessed 28.9. 2023. https://docs.xilinx.com/r/en-US/ug1387-acap-hardware-ip-platform-dev-methodology/Synchronous-Reset-vs.-Asynchronous-Reset
14. JEDEC Solid State Technology Association 2012. JEDEC Standard. DDR3 SDRAM Specification JESD79-3F. Accessed 29.9. 2023. https://www.jedec.org/sites/default/files/docs/JESD79-3F.pdf Requires license.
15. Detection Technology. Accessed 2.10. 2023. https://www.deetee.com/
16. Zwolinski Mark 2004. Digital System Design with VHDL. Second edition. Pearson Education Limited.
17. Kafig William 2011. VHDL 101 Everything you need to know to get started. Elsevier Inc. Accessed 10.11.2023 https://learning.oreilly.com/library/view/vhdl-101/9781856177047/?ar%2F%3Forpq=&email=%5Eu
18. McMurchie Larry and Ebeling Carl 1995. Pathfinder: A Negotiation-Based Performance-Driven Router for FPGAs. Accessed 12.11. 2023. https://www.cecs.uci.edu/~papers/compendium94-03/papers/1995/fpga95/pdffiles/6a.pdf Accessed
19. Lattice Semiconductor 2023. ECP5 and ECP5-5G Family datasheet. Accessed 14.11. 2023. https://www.latticesemi.com/view_document?document_id=50461
20. Floyd Thomas 2006. Digital Fundamentals Ninth edition. Pearson Education Limited.
21. National Instruments 2020. FPGA Fundamentals. Accessed 19.11.2023. https://www.ni.com/en/shop/electronic-test-instrumentation/add-ons-for-electronic-test-and-instrumentation/what-is-labview-fpga-module/fpga-fundamentals.html
22. Lattice Semiconductor 2023. ECP5 and ECP-5G Memory User Guide. Technical Note. Accessed 19.11. 2023 http://www.latticesemi.com/view_document?document_id=50466
23. Stauffer, Mechler, Sorna, Dramstad, Ogilvie, Mohammad & Rockohr 2008. High Speed Serdes Devices and Applications. Springer Science + Business Media LLC. Accessed 20.11. 2023. https://books.google.fi/books?hl=fi&lr=&id=Cx3r0H-4AhEC&oi=fnd&pg=PR5&dq=serdes&ots=vqOwBuch1F&sig=JUpA8uigabNdme7VjYG0bPLH0Vg&redir_esc=y#v=onepage&q&f=false
24. Xilinx 2021. UltraScale Arcitecture DSP Slice. User Guide. Accessed 23.11. 2023. https://www.xilinx.com/content/dam/xilinx/support/documents/user_guides/ug579-ultrascale-dsp.pdf
25. Maxfield Clive 2008. FPGAs Instant Access. Elsevier Limited. Accessed 23.11. 2023. https://learning.oreilly.com/library/view/fpgas-instant-access/9780750689748/content/kindle_split_0.html

26. Micron Technology Inc 2018. 8Gb: x4, x8, x16 DDR3L SDRAM data sheet. Accessed 24.11. 2023. https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/ddr3/8gb_ddr3l.pdf?rev=9889999683d446c8bfb3e79f4b255d33
27. Wakerly John 2006. Digital Design Principles and Practices. Fourth Edition. Pearson Education, Inc.