

Bachelorarbeit

**Extendable Hardware-Based Main Memory
Access Snooping for Non-volatile Memory
Simulations and Analysis**

Junior Delrich Kamtchogom Namtchueng
04.09.2020

Supervisors:

Prof. Dr. Jian-Jia Chen

M.Sc. Christian Hakert

Technische Universität Dortmund

Fakultät für Informatik

Lehrstuhl Informatik 12 (Eingebettete Systeme)

<http://ls12-www.cs.tu-dortmund.de>

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Objectives and Methodology	1
2	Inside a programmable logic hardware: FPGA	3
2.1	Definition and applications	4
2.2	Structure of an FPGA	4
2.2.1	Configurable Logic Block (CLB)	4
2.2.2	Routing logic	6
2.2.3	Input/Output Interface	7
2.3	Programming of an FPGA	7
3	Basics of VHDL	9
3.1	History and Concept	9
3.2	Hello World Example	9
4	Introduction to AXI Protocol	11
4.1	Basic concepts and keywords used in the AXI protocol	12
4.2	AXI4-Stream protocol	13
4.2.1	Handshake process	14
4.3	Memory Map Protocols and AXI4-Lite	14
4.3.1	Handshake process	14
5	Memory Access Snooping	19
5.1	Description of Equipment	19
5.1.1	FPGA	19
5.1.2	Others tools	19
5.2	Register access snooping	20
5.3	Block Random Access Memory (BRAM)	21
5.4	BRAM access snooping	22
5.5	DRAM access snooping	23
6	Evaluation	25
6.1	Evaluation of access to registers	25

6.2 Evaluation of access to BRAM	25
7 Conclusion and Outlook	27
List of figures	29
Source code	31
Bibliography	34
Eidesstattliche Versicherung	35
Eidesstattliche Versicherung	35

Abstract

In order to determine the internal functioning of a computer memory, a simulation is necessary. Simulate a memory means, make read/write requests to this memory and observe the path that the requests are going through. Knowledges about the internal functioning is for example important to be able to give some guarantees about its latency or its reponse time. Make this simulation implicitly means the need of a simulator.

This thesis is about designing the simulator by using an extension hardware interface of the processor and then analysing the output values of the simulator.

The methodology followed here is, first the presentation of the hardware interface, then the presentation of the dataflow during the different requests, then the presentation of the design used for the simulation and finally the analysis of the simulation result.

The analysis of the simulation result showed that both part of the memory (registers and BRAM) simulated here work perfectly. In addition, during the access to the BRAM, it results a delay between the moment when it receives the request and the moment when it returns a response. Beyond that, the registers and the BRAM have in common that they have a small addressable space. To have an access to a larger space another means (DRAM access) is slightly described.

1 Introduction

1.1 Motivation

Simulating a memory means making read and write requests and analyzing what happens throughout these requests, from the moment they are sent to the returned value. This simulation can for example be made in the context of determining the internal functioning of the memory, i.e. for example determining its latency or its response time. The read/write requests to memory are effectively not possible without an adequate simulator.

1.2 Objectives and Methodology

This thesis is about designing the simulator by using an extension hardware interface of the processor and then analysing the output values of the simulator. The extension hardware used here is a programmable circuit (FPGA).

The methodology followed throughout the thesis is, first of all, a general presentation of the hardware interface(FPGA), i.e. its functioning, the programming language used to program it and the dataflows within it. Subsequently the description of the design with the descriptions of different operations on the memory and finally an analysis of the results obtained during the simulation.

2 Inside a programmable logic hardware: FPGA

This chapter is about details on the functioning of the hardware platform used throughout the thesis.



Figure 2.1: Xilinx ZC706 FPGA [24]

A typical system design consists of several chips: a CPU (Central processing unit), Input/Output interfaces, flash SDRAM (Synchronous Dynamic Random Access Memory) and a DSP (Digital Signal processor) chips. For this system a quite large board is required in order to contain all this chip. This configuration increases the design cost and the complexity of the board. In order to decrease these parameters, it would be better to have all-in-one chip; that means the CPU, the IO interfaces and the DSP in one chip. We could achieve this objective by using a programmable logic interface like a FPGA. In this chapter, it is about exploring the functionalities of programmable logic devices and in particular the details of the architecture of the FPGA. The FPGA will be used as a hardware interface within the framework of this thesis in order to carry out all the experiments.

2.1 Definition and applications

FPGA stands for Field Programmable Gate Array. It's a sophisticated and flexible piece of silicon, that implements an arbitrary digital design. An advantage of using the FPGA is that it can be programmed at logic level. Hence it allows to efficiently accelerate algorithms and achieving performance gains otherwise not possible. The primary difference between an FPGA and other traditional logic circuits is that the FPGA is completely fabricated before it has been customized with a design. This makes the FPGA flexible and programmable at any time on the software level.

The use of the FPGA as a hardware interface also has some disadvantages: the programming of FPGA requires knowledge of VHDL/Verilog programming languages as well as digital system fundamentals. Moreover engineers need to learn the use of simulation tools. In addition, the power consumption is more than power consumption of traditional circuits and programmers do not have any control on power optimization in FPGA.

Digital logic in the form of programmable logic devices can be found everywhere. A well-known example of his application is the use to make real-time applications. Real-time applications are visible in several areas of daily life like air traffic control, network communication, command control in the aviation engineering etc. . . The correct functioning of this applications depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. This second parameter is decisive for the choice of using an FPGA because it makes it possible to give good time guarantees for the operations to be executed [13].

2.2 Structure of an FPGA

In order to use an FPGA for complex tasks, it is very important to know how itself and its main components are designed. FPGA logic is made up of number of logic blocks, a routing logic and a I/O interface.

2.2.1 Configurable Logic Block (CLB)

The logic blocks are the fundamental elements of FPGAs. They are in particular responsible for the reconfiguration property of FPGAs. Each logic block usually contains look up tables, registers and other configurable features. The routing logic inside the FPGA is separated from logic blocks elements. The following part gives further informations about the three main part of a logic block which are : Look Up Table (LUT), registers and carry logic. [6] [25]

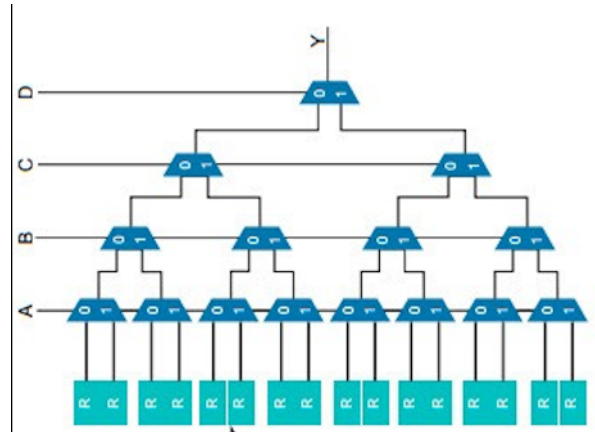


Figure 2.2: How a LUT might look for an AND gate with 4 inputs

Look Up Table (LUT)

The LUT is the key component that allows to realize the functions of the combinatorial logic. It is made up of the series of cascaded multiplexers where the LUT inputs are used as the select lines. The inputs to the multiplexers are programmed as high- or low-level logic levels. The logic is called a lookup table because the output is selected by looking up to the correct program level and routing it correctly through the multiplexers based on the LUT input signals. The program level selected are based on the truth table for the function.

As an example of the operation of the LUT fig. 2.3 the minterm is routed through the logic block where A and B are high and C and D are low. This particular minterm corresponds to this product term in the desired logic function. [5]

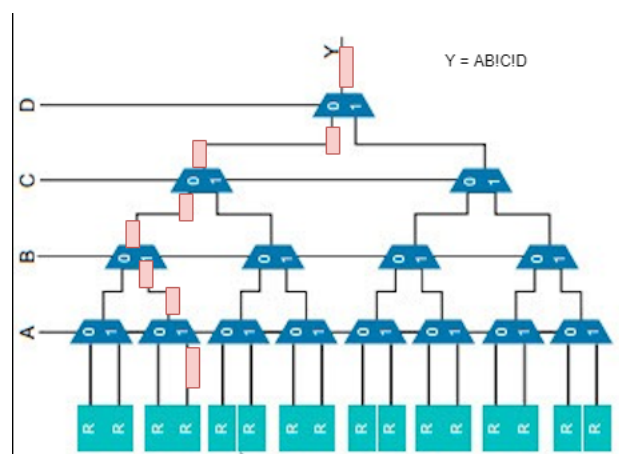


Figure 2.3: Example of LUT for an AND gate with 4 inputs

Registers

A register is a group of flip-flops that stores a bit pattern. The register controls the synchronisation inside a logic block. It has a clock, input data, output data and an enable signal port. Every clock cycle, the input data is latched, stored internally, and the output data is updated to match the internally stored data. If the register isn't needed, it can be bypassed to produce a strictly combinatorial logic function from the LUT. Again, similar the LUT can be bypassed and the register would behave like a storage or synchronisation function. [3]

Carry chain logic

In order to achieve high performance hardware, the circuit critical path has to be optimized. For most datapath circuits this critical path goes through the carry chain used in arithmetic and logic operations. The logic blocks contain specific carry chain logic to provide shortcuts for some signals. The carry chain is the feature allowing FPGAs to be efficient at arithmetic operations (counters, adders...). For this logical arithmetic operation, the chain communicates the cumulative information needed to perform these computations. For those FPGA which implement the carry logic, bypassing LUT and the carry logic will turn the logic block to a shift register, that could be used for example to compute DSP (Digital Signal Processing) type operations. [7]

2.2.2 Routing logic

The routing channels and FPGA devices seem simple, but they actually provide lots of functionality and connectivity. FPGA routing channels allow all device resources to communicate with all other resources anywhere on the chip. The interconnection delays are often greater than the logic delays of the designed circuit. Therefore, an efficient routing algorithm tries to reduce the total wiring area and the lengths of critical-path nets to improve the performance of the circuit; and for this, the router needs the interconnect information of the target FPGA architecture. Throughout this work, the board used comes from the company Xilinx, that's why we are interested in the architectures used by Xilinx.

In Xilinx routing, connections are made from logic blocks into the channel through a connection block. Because of the use of SRAM (Static random-access memory) to load LUTs, the connection sites become very large. A logic block is bounded by connection blocks on all four sides fig. 2.4 and they connect logic block pins to wire segments. The logic block pins connecting to connection blocks can then be connected to any number of wire segments through switching blocks. [11]

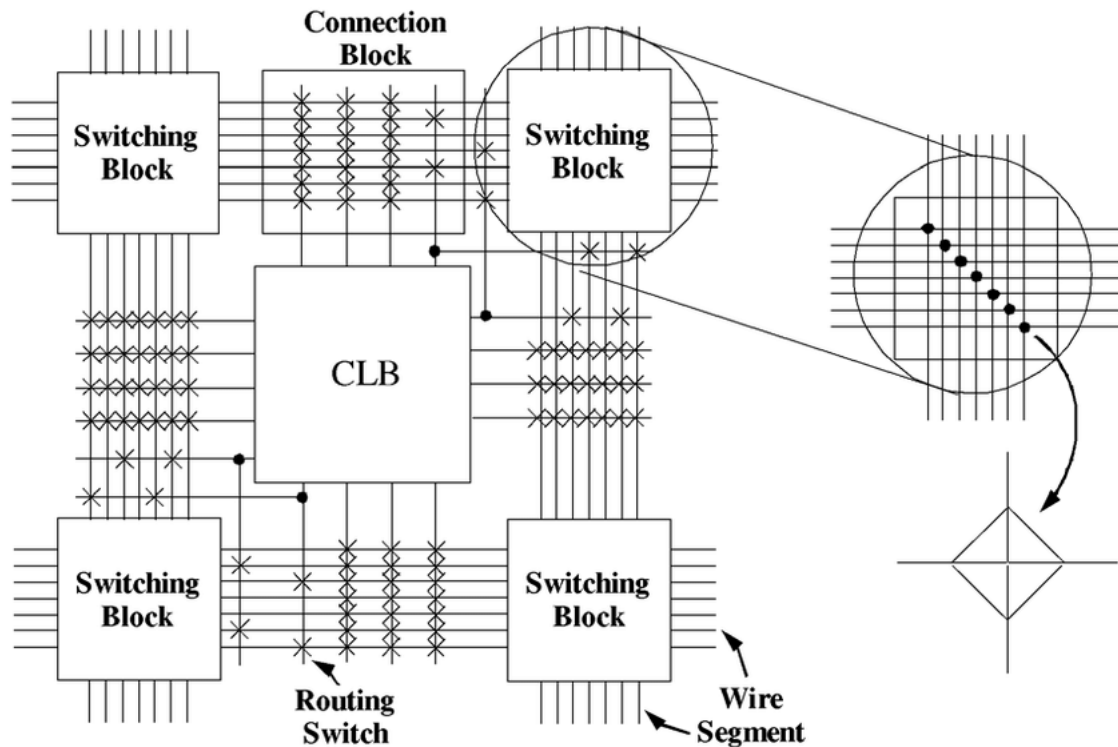


Figure 2.4: FPGA routing example [1]

2.2.3 Input/Output Interface

FPGA I/O control is containing blocks found around the outer edge of the array and made available to all device resources through the FPGA routing channels. The I/O interface supports a wide variety of standards. Pairs of I/O pins can also be combined to support differential signaling I/O standards such as Low-voltage differential signaling (LVDS). Other features include variable current drive strength and slow rate control to help improve board level signal integrity. The I/O Interface supports both digital and analog Input/Output. [4]

2.3 Programming of an FPGA

To support possible connections between all components of the FPGA architecture, while still including a large amount available user logic, simpler and smaller programming structures are needed. To meet these goals, FPGAs use SRAM cells to program logic levels and make routing connections. To program an FPGA, the procedure is as follows:

1. Design the circuit to be implemented using tools that, most of the time for performance and time saving reasons, are made by the company that manufactures the FPGA model. But generic opensource tools are also available. The design itself

consists of one or more modules commonly called IP (Intellectual Property) core in Xilinx jargon. The modules are described using a programming language; in this case it is VHDL.

2. Always by using Xilinx tools, load the design on the FPGA memory cells. At this level it can be done using a memory card, a JTAG connection (a common hardware interface that provides the computer with a way to communicate directly with the chips on a board) or a USB connection directly on the board.

During the loading of the design into the memory cells, the values of the SRAM bits are determined and then the configuration of each logic block is determined by the SRAM. The configuration of each logic block is determined by SRAM bits, which are connected to the circuit elements. The SRAM acts basically as a latch. When programming is enabled, the values of the SRAM bits are determined by the configuration data is loaded into the FPGA internal memory cells. This configuration data is loaded into the FPGA from some external device such as a Read-Only Memory (ROM). Therefore, the FPGA can be reconfigured an unlimited number of times just by reloading a new set of values from the external device.

3 Basics of VHDL

In order to perform the memory accesses, the logic described by the design must be implemented on the FPGA and to do so, it must be described with a programming language that it can understand and interpret. The programming language chosen here is VHDL. This chapter is about some basics concepts of the VHDL.

3.1 History and Concept

VHDL (VHSIC-HDL, Very High Speed Integrated Circuit Hardware Description Language) is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits. [2]

It was originally developed at the behest of the U.S. Department of Defense in 1983 to document the behaviour of the ASICs (Application-Specific Integrated Circuit), then developed with the idea of being able to interpret the information contained in the documentation and so logical simulators were developed to read the VHDL files. VHDL inherited for budgetary reasons the syntax and concepts established for the ADA programming language. [10]

The main concept of VHDL used in this work is the parallel programming concept. Indeed in the case of the FPGA the operations are parallelized with clocks which are used as tact to trigger the operations.

3.2 Hello World Example

In this example it is a question of implementing the basic example well known enough in VHDL and then to explain the different keywords to allow its comprehension. This implementation serves as a basis for all components of the design described in this work with VHDL.

```
1 entity HelloWorld_entity is
2 end entity;
3 architecture HelloWorld_entity_arch of HelloWorld_entity is
4 begin
5     process is
6     begin
```

```
7     report "Hello World!";
8     wait;
9     end process;
10 end architecture;
```

Listing 3.1: Hello World in VHDL

On the first two lines we declared the entity. The entity of a module declares its inputs and outputs. For it to be possible to run a module in a simulator, it cannot have any inputs or outputs. Therefore, the module doesn't have anything other than an empty entity declaration.

Next, we declared the architecture of the module. While the entity is a modules interface to the outside world, the architecture is its internal implementation. A module may have several architectures which may be used with the same entity. That are advanced VHDL features that could be found in advanced VHDL guides.

Inside of the architecture we declared a process. For now, we can think of a process as a thread in our program, where things happen sequentially.

Inside the process we print "Hello World!" by using the keyword "report". On the next line there is a single wait; When the simulator hits this line, nothing more is going to happen. The process will wait there forever.

4 Introduction to AXI Protocol

In order to create a design that will be loaded on the FPGA, it is necessary to understand how the designs are made. The FPGA provided by the company Xilinx uses an interfacing protocol that facilitates the implementation and the dataflow between the different components of the design. The definition of a design for the FPGA is done at a higher level of abstraction than the direct manipulation of the logic blocks, i.e. with the tools provided by the company that manufactures the FPGAs. It is quite possible to describe the design with a programming language such as C or C++ while using all the abstractions related to these programming languages and then transcribe the resulting code into VHDL code that can be read and loaded into the hardware to define the configuration of the logic blocks. It is notably possible to directly describe the components of the design in VHDL, but regularly this makes the task a little more complex depending on the objective to be achieved. The design components are called IP (Intellectual Property) core and the interface protocol they use in the case of FPGAs produced by Xilinx is the AXI protocol. AXI (Advanced eXtensible Interface) is an interface protocol defined by ARM as part of the AMBA (Advanced Microcontroller Bus Architecture) standard. An interface protocol is a specification that defines how data is delivered and interpreted [9]. The version of the protocol used in this work is version 4 included in version 4.0 of AMBA released in 2010. The essence of the AXI protocol is that it provides a framework for how different blocks inside each chip communicate with each other. It offers a procedure before anything is transmitted, so that the communication is clear and uninterrupted.

The AXI Protocol has several advantages:

- **Productivity:** Because the protocol has been standardized and made available for free, IP cores have been made easier to design since only one protocol needs to be mastered.
- **Flexibility:** The AXI protocol is subdivided into several parts that can be used according to the objective to be achieved. The details of these different parts are discussed below.
- **Availability:** The AXI protocol is not only used by Xilinx for the creation of core IPs for FPGAs designed by Xilinx, but also by many members and partners of the ARM community. In short, there is a considerable amount of IP core designed by

the large community available that can be imported and even used for Xilinx FPGAs and many others.

- **Parallelism:** Multiple bits, typically 32, are sent in parallel, what can have other sizes.
- **High-performance:** The AXI protocol contains features such as streaming, to move data more quickly than word by word.

4.1 Basic concepts and keywords used in the AXI protocol

Before detailing the AXI protocol, it is important to clarify the meaning of a few terms and concepts that recur fairly regular throughout this thesis.

- AXI protocol is the set of rules that define a data exchange between the AXI Master and the AXI Slave.
- AXI Master/Slave model is an asymmetric communication protocol where an object called master controls another object called slave. The definition given to the master and slave varies according to the context of use. In the case of the AXI protocol, the master is the one that initiates an AXI transaction and the slave is the one that responds to this transaction.
- AXI transaction is the flow that defines the transfer of data from one point to another using the elements of the AXI protocol.
- Data burst is the broadcast of a relatively high-bandwidth transmission of data over one transaction
- Intellectual Property (IP) core is a reusable unit block of logic made in order to simplify the design of the FPGA.
- Signal is a sequence of bits that carries information from one point to another. Within the framework of this thesis, signals have types that define their length. The main types used here are `std_logic` which defines a signal of size 1 whose value can be 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'; `std_vector` which defines a signal of variable size and value. Further details about the types can be found here [18]
- Memory address is a value that refers to a memory area of defined size. The size of this memory area is 32 bits in this thesis.

4.2 AXI4-Stream protocol

The AXI4-Stream protocol is used for applications that typically focus on a data-centric and data-flow paradigm where the concept of an address is not present or not required. Each AXI4-Stream acts as a single unidirectional channel for a handshake dataflow.

Data centric refers to an architecture where data is the primary and permanent asset, and applications come and go. In the data centric architecture, the data model precedes the implementation of any given application and will be around and valid long after it is gone. [8]

Data flow is a concept aiming to decentralize the operations carried out during a program in the form of a pipeline in order to execute them in parallel.

To transfer data between the AXI Master and the AXI Slave, the AXI4-Stream protocol uses different signals to communicate.[26]

Signal	Status	Notes
TVALID	Required	
TREADY	Optional	TREADY is optional, but highly recommended.
TDATA	Optional	
TSTRB	Optional	Not typically used by endpoint IP; available for sparse stream signalling. Note: For marking packet remainders, TKEEP use rather than TSTRB.
TKEEP	Absent	Null bytes are only used for signaling packet remainders. Leading or intermediate Null bytes are generally not supported.
TLAST	Optional	
TID	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TDEST	Optional	Not typically used by endpoint IP; available for use by infrastructure IP.
TUSER	Optional	

Figure 4.1: AXI4-Stream signal [14]

The most interesting signals are:

- TVALID indicates when data is available.
- TREADY indicates that the AXI slave can accept data
- TDATA is the primary payload of the AXI4-Stream interface and is used to transport data from a source to a destination.

- TLAST marks the end of data transmission.

4.2.1 Handshake process

All data transfer procedures in a chip are clocked by at least one clock. The data transfer takes place only when TVALID and TREADY are asserted at the same time. When the data is present at the AXI Master, it asserts the TVALID signal; when the AXI Slave is available for data reception, it asserts the TREADY signal. The assertion of the TVALID and TREADY signals is independent of each other and the data transmission only takes place on a rising clock. In fig. 4.2 the arrow shows when the transfer occurs.[14]

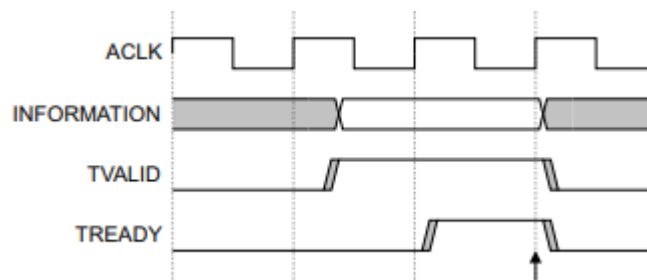


Figure 4.2: AXI4-Stream handshake process

4.3 Memory Map Protocols and AXI4-Lite

This protocol emphasizes the use of addresses for data transfer. It has the advantage of giving a homogeneous view of the memory space because the IP core evolves in a predefined memory space. The AXI4 Lite protocol is a subset of the Memory Map Protocols with only basic features.

- It simplifies its use by removing many of the data transfer signals but follows the AXI4 specification for the rest.
- It allows only one data transfer in one time, i.e. data-burst mode is not possible.

Because of its relative simplicity, this protocol is used for data transfer in the practical part of this work.[26]

4.3.1 Handshake process

In the methodology for using the memory map protocol, the concept of Write Address/-Data/Response channel and Read Address/Data channel fig. 4.3 is used. The communication over each channel is done by using a handshake protocol just like in AXI4-Stream protocol.

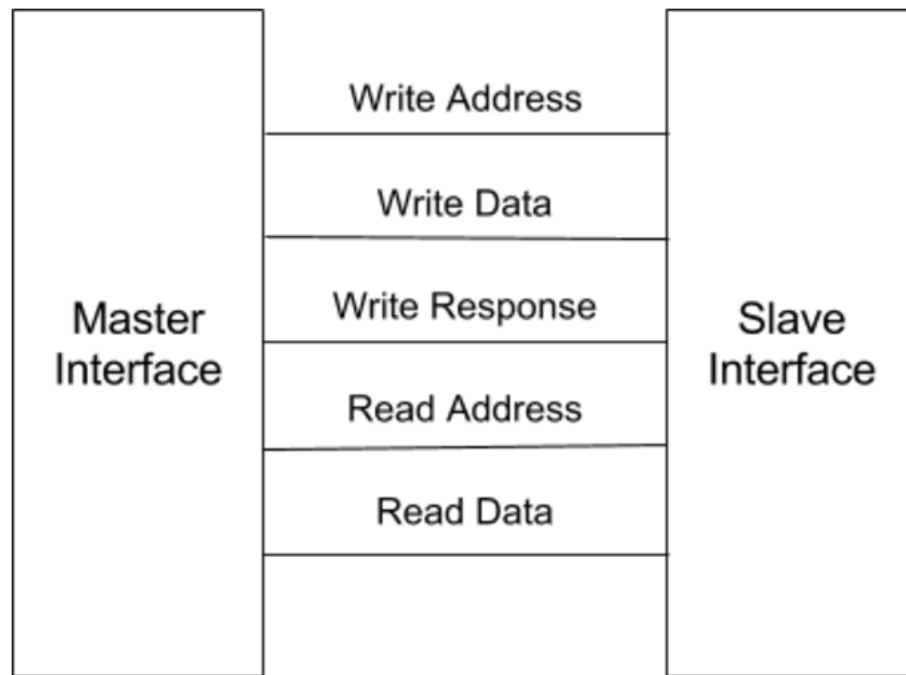


Figure 4.3: Channel connection between the AXI Master and the AXI Slave interface [15]

- Write Address channel provides address by the AXI Master at which data should be written.
- Write Data channel is the channel used to send data from an AXI Master to an AXI Slave.
- Write Response channel specifies the burst status but it's not needed in AXI4-Lite protocol because it doesn't provide this functionality.
- Read Address channel provides address by the AXI Master from which to read data.
- Read Data channel is channel used by the AXI Slave to send data back.

The handshake protocol also uses signals to coordinate the transfer of data and these signals are relative to the channel being used. Here are the signals used in the AXI4-Lite protocol

- AWVALID is generated by the AXI Master when Write Address is valid
- AWREADY is generated by the AXI Slave when it can accept Write Address.
- AWADDR holds the Write Address
- WVALID is generated by the AXI Master when Write Data is valid
- WREADY is generated by the AXI Slave when it can accept the Write Data.

- WDATA holds the Write Data
- ARVALID is generated by the AXI Master when Read Address is valid
- ARREADY is generated by the AXI Slave when it can accept Read Address and control signals
- ARADDR holds the Read Address
- RVALID is generated by the AXI Slave when Read Data is valid
- RREADY is generated by the AXI Master when it can accept the Read Data and response.
- RRESP indicates the status of data transfer.
- RDATA holds the Read Data

Write address channel	Write data channel
AWVALID	WVALID
AWREADY	WREADY
AWADDR	WDATA
AWPROT	WSTRB

Figure 4.4: Write signals sorted by channel

Read address channel	Read data channel
ARVALID	RVALID
ARREADY	RREADY
ARADDR	RDATA
ARPROT	RRESP

Figure 4.5: Read signals sorted by channel

As mentioned above, there are indeed two types of transactions possible : read and write transactions. [26] [14]

AXI4 Lite Read Transaction

The steps for a read transaction are quite simple [12]

1. The AXI Master puts an address on the Read Address channel as well as asserting ARVALID, indicating the address is valid, and RREADY, indicating the AXI Master is ready to receive data from the AXI Slave.
2. The AXI Slave asserts ARREADY, indicating that it is ready to receive the address.

3. Since both ARVALID and ARREADY are asserted, on the next rising clock edge the handshake occurs, after that the AXI Master and AXI Slave deassert ARVALID and the ARREADY, respectively. (At this point, the AXI Slave has received the requested address).
4. The AXI Slave puts the requested data on the Read Data channel and asserts RVALID, indicating the data in the channel is valid. The AXI Slave can also put a response on RRESP.
5. Since both RREADY and RVALID are asserted, the next rising clock edge completes the transaction. RREADY and RVALID can now be deasserted.

AXI4 Lite Write Transaction

The steps for a write transaction are quite simple [12]

1. The AXI Master puts an address on the Write Address channel and data on the Write Data channel. At the same time it asserts AWVALID and WVALID indicating the address and data on the respective channels is valid.
2. The AXI Slave asserts AWREADY and WREADY on the Write Address and Write Data channels, respectively.
3. Since Valid and Ready signals are present on both the Write Address and Write Data channels, the handshakes on those channels occur and the associated Valid and Ready signals can be deasserted. (After both handshakes occur, the AXI Slave has the Write Address and Data)
4. The AXI Slave asserts BVALID, indicating there is a valid response on the Write Response channel.
5. The next rising clock edge completes the transaction, with both the Ready and Valid signals on the write response channel high.

It's important to notice that the handshakes on the Write Address and Write Data channel do not necessarily occur simultaneously but both must occur before the AXI Slave can send a Write Response. Also Write Address and Write Data handshakes can occur independently or simultaneously and no order is enforced, only that both must occur to complete the transaction.

5 Memory Access Snooping

This chapter is about describing the design carried out in order to solve the problem raised in the introduction i.e. perform the simulation on the memory. The chapter is divided into two parts. The first part consists of describing the material used and the second part consists of describing the accesses to the different types of memories.

5.1 Description of Equipment

The materials used to achieve the objective are as follows

5.1.1 FPGA

The FPGA used is a Zynq-7000 series FPGA from the company Xilinx, more precisely the evaluation board ZC706.



Figure 5.1: FPGA Evaluation Board ZC706 [24]

The interesting memory characteristics for this work are the following:

- The board has 350K logic cells
- The board has a 19 Mb Block RAM.
- The board has a 2 Gb SDRAM (Synchronous Dynamic Random Access Memory).

Further details on the board can be found here [23].

5.1.2 Others tools

The two other tools mainly used are the following:

- The vivado IDE which is mainly used for the design and simulation of IP cores before their synthesis. More generally it is used for PL (Programmable Logic) modeling.

- The Eclipse IDE is adapted by the company Xilinx for PS (Processing System) programming.

5.2 Register access snooping

The FPGA has registers and as mentioned above section 2.2.1, those registers can be used as memory for operations that do not require too much memory space. The design needed to access this memory is described here. The objective is to count the number of write/read requests on this memory.

The design essentially has the following components:

- An IP core ZYNQ7 Processing system which is the intermediate component between PS and PL. It is also the element that plays the role of the AXI Master for all the others AXI Slaves components (In this design there is only one AXI Slave) and its behavior is programmable in C/C++ from the IDE ECLIPSE.
- An IP core Processing system reset: It provides customized resets for an entire processor system, including the processor, the interconnect and peripherals. [22]
- An IP core AXI Interconnect: It is used to establish the connection and control data transmission between AXI Masters and AXI Slaves. For example, it controls the protocol version used, the size of the addresses and the size of the transmitted data. [16]
- An IP core IP_MEMORY_BLOCK whose functionalities and behaviour will be described below.

It is important to note that the first three components mentioned above are components designed by the company Xilinx. The last component has been designed throughout the thesis in order to achieve access to registers.

Functionalities and Behaviour of IP_MEMORY_BLOCK

This component allows access to a memory space made up of 40 addresses of 32 bits each one which follow one another, i.e. a total memory space of $32 \times 40 = 1280$ bits = 160 bytes. It has an AXI Slave port to receive data and address from the AXI Master and a array of size 40 where the received data is stored.

Address 0 of the array is a counter that stores the number of write accesses, address 1 stores the number of read accesses and address 2 has the value by default 00000000 but takes the value FFFFFFFF if there is a write request to address 0, 1 or 2 or if there is a read/write request to an address greater than 40 and ignore the request.

How the request happen is quite simple. To perform a write request, the AXI Master sends the data to be written and the address, then the AXI transaction is carried out according to the AXI4-Lite protocol. The AXI Slave then receives the data and saves it in the array. To perform a read request, the AXI Master sends the address, then the AXI transaction is performed and the AXI Slave returns the value at the requested address from the array.

5.3 Block Random Access Memory (BRAM)

The FPGA has a type of memory called BRAM located in the PL. The BRAM (sometimes called embedded memory, or Embedded Block RAM (EBR)) larger than the registers. In a generic way how BRAM works, is quite simple. It uses the following signals [17]:

- Enable (EN) : it determines whether a read or write operation can happen. It has a length of 1 bit and is of type `std_logic`.
- Write enable (WE) determines whether a write operation can happen. It has a length of 4 bits and is of type `std_vector`. When its value is "0000", only read operations can be performed, when its value is "1111", only write operations can be performed.
- Address (ADDR) holds the address whose content must be read during a read operation or whose content must be modified during a write operation.
- Data in (DIN) : it's a signal that holds the data to be written during a write operation.
- Data out (DOUT): it's a signal that holds the data that was read during a read operation.

BRAM uses the concept of a port. A port is a logical group of signals that do not necessarily have an interdependence between them. Each BRAM port groups at least these signals together. Unlike the AXI-4 Lite data transfer protocol transactions, the one used for writing and reading BRAM is quite simple.

- If Enable has the value '0' no operation is possible. During the simulation on the memory Enable is initialized with the value '1' and does not change until the end of the execution.
- If the Write enable has the value "0000", the memory is read at ADDR and the read value is written to the DOUT.
- If the Write enable signal has the value "1111" then DIN is written to ADDR in the memory. Once this is done the same address is read and its content is written to DOUT. There is a delay between the moment when an address is indicated and when the value written to this address can be read from DOUT.

At the logic level, there are two types of BRAM configuration:

- **Single Port BRAM:** As the name implies, this is a BRAM with only one communication port. This port is used for both reading and writing data, i.e. it is not possible to read and write simultaneously.
- **Dual port BRAM.** This configuration uses two ports and allows simultaneous reading and writing.

5.4 BRAM access snooping

The goal of this design is the same as the one performed on the registers, i.e. to count the number of write and read requests made on the memory. For this, the design previously used for the registers is reusable here. The presence of the IP core `IP_MEMORY_BLOCK` is not necessary here but it does not influence the result either. To achieve this goal two main components are added to the design:

- **Block Memory Generator (BMG)** is a component designed by Xilinx that represents the BRAM. The written data has a 32-bit size and the depth (number of addresses) of the BRAM is 2048, i.e. the total accessible memory is $32 * 2048 = 65536$ bits. For this design the single port BRAM configuration is used.
- **SLAVEBRAM** is the component designed to communicate with the BMG. It has one AXI Slave port to communicate with the AXI Master and one BRAM port to communicate with the BMG.

Functionalities and Behaviour of SLAVEBRAM

This IP core works by using the AXI4-Lite protocol to communicate with the AXI Master and the communication protocol described above for BRAM. The data transaction steps combine these two protocols. The handshake for writing happens as follows:

- The AXI Master puts an address on the Write Address channel and data on the Write Data channel. At the same time it asserts `AWVALID` and `WVALID` indicating the address and data on the respective channels is valid.
- The AXI Slave asserts `AWREADY` and `WREADY` on the Write Address and Write Data channels, respectively.
- Once `AWREADY`, `WREADY`, `AWVALID`, `WVALID` are asserted at the same time, Write enable is set to "1111" and `AWADDR` respectively `WDATA` is copied on the `ADDR` respectively on `DIN`.

- 2 Clock Cycles later Write Enable is reset to "0000" and the transaction is completed.

The handshake for reading happens as follow :

- The AXI Master puts an address on the Read Address channel as well as asserting ARVALID, indicating the address is valid, and RREADY, indicating the AXI Master is ready to receive data from the AXI Slave.
- The AXI Slave asserts ARREADY, indicating that it is ready to receive the address on the bus.
- Once ARVALID, ARREADY are asserted at the same time and RVALID is not, Write enable is set to "0000" and AWADDR is copied into ADDR. The content of the address is read and is written on DOUT with a delay of 2 clock cycles.
- DOUT is copied to RDATA and RVALID is asserted.
- The rest of the transaction conforms to the AXI4-Lite protocol after assertion of RVALID.

For the case where the address sent by the AXI Master is 0 1 or 2 the processing is the same as described for the register memory. For access attempts with an address greater than 2047, the requested address modulo 2047 will be considered as the address sent by the AXI Master. Also the number of clock cycles indicated above is not mentioned in any documentation concerning the BRAM, it was obtained after observation of the simulations.

5.5 DRAM access snooping

On the PS there is a DRAM (Dynamic Random Access Memory) whose size varies according to the board used. In our case a 2 Gb memory is available. This is the most important advantage of using DRAM. There are particularly two types of DRAM present on the board dedicated to this experience:

- Double Data Rate 3 Synchronous Dynamic Random-Access Memory (DDR3 SDRAM)
- Double Data Rate 2 Synchronous Dynamic Random-Access Memory (DDR2 SDRAM)

Access to DRAM is done by adding the MIG (Memory Interface Generator) component to the previously used design. No read/write experiments are performed here. For a possible research the documentation of the MIG is available [21] as the first step to achieve this purpose.

6 Evaluation

This chapter is about the evaluation of the result obtained during the simulations made on the design described above. For the evaluation a few tests have been written. These tests are indeed programs written in C in the Eclipse IDE modified by the company Xilinx. The written tests will define the data and addresses that will trigger the AXI transactions in the design.

6.1 Evaluation of access to registers

The tests on the registers worked perfectly and the counters were justly updated. The disadvantage of using the registers is its small size. On the board dedicated to this experiment, the registers have a total size of 5088 kilobits [20]. However, the registers offer a better read/write access time than all other memories.

6.2 Evaluation of access to BRAM

The tests on the BRAM worked perfectly and the counters were justly updated. On the board dedicated to this experiment the BRAM has a total size of 19.2 Megabits [20] [23]. The BRAM size is sufficient for a large number of applications designed for FPGA. Also the read/write access times are acceptable. To optimize the latency, the BRAM is subdivided into blocks whose sizes vary depending on the board used. On this board, each blocks can be 18 or 36 kilobits.[19]

7 Conclusion and Outlook

This thesis is about simulating and analyzing memories using an extension hardware platform to the Processor, the FPGA. In this framework two ways to achieve that purpose has been presented. The first one is the access to registers and the second one is the access to Block Random Access Memory (BRAM). The simulation consists of performing write and read requests on the memory and the analysis consists of following the request from the moment where they are sent to the moment when the returned value show up by the sender. After the simulation, some tests has been written in order to make sure that the returned values are the expected values. The result is conclusive since all the returned values corresponded to the expected values. In order to perform the simulation, a design was made and implemented on the FPGA. The special feature of this design regarding BRAM is the synchronization of the data between the moment when the requests are made and the time it takes for BRAM to respond to them. This synchronization has been achieved here by the use of delay section 5.4.

As far as the size of the simulated and analyzed memory is concerned here, an access to the registers and BRAM gives restricted access to the memory. A way to get an access to more space is the Dynamic Random Access Memory (DRAM). Indicative elements such as the MIG to start the simulation and the analysis on it are mentioned here section 5.5. The next step could be , by using the MIG , find out which characteristics, constraints, advantages and disadvantages the DRAM has and then compare them to characteristics of the registers and the BRAM.

List of Figures

2.1	Xilinx ZC706 FPGA	3
2.2	How a LUT might look for an AND gate with 4 inputs	5
2.3	Example of LUT for an AND gate with 4 inputs	5
2.4	FPGA routing example	7
4.1	AXI4-Stream signal	13
4.2	AXI4-Stream handshake process	14
4.3	Channel connection between the AXI Master and AXI Slave interface . . .	15
4.4	Write signals sorted by channel	16
4.5	Read signals sorted by channel	16
5.1	FPGA Evaluation Board ZC706	19

Source code

3.1 Hello World in VHDL	9
-----------------------------------	---

Bibliography

- [1] F. A. Aloul. A comparative study of two boolean formulations of fpga detailed routing constraints.
- [2] D. R. Coelho. *The VHDL Handbook*. June 1989.
- [3] N. I. CORP. Introduction to fpga hardware concepts (fpga module). 2018.
- [4] N. I. CORP. Introduction to fpga hardware concepts (fpga module). 2018.
- [5] A. Corporation. Fpga architecture white paper. July 2006.
- [6] A. Corporation. *Cyclone II Device Handbook*, volume 1. 2007.
- [7] FPGA4fun.com. Counters 4 - the carry chain.
- [8] D. McComb. The data-centric revolution: Data-centric vs. data-driven. September 2016.
- [9] newwavedv.com. Interface protocols.
- [10] D. of Defense. Military standard, standard general requirements for electronic equipment. 1992.
- [11] D. F. G. Prado. Tutorial on fpga routing. August 2006.
- [12] readigital.org. Introduction to axi4-lite (advanced extensible interface).
- [13] D. B. T. Shane T Fleming. Fpga based control for real time systems. Number 13873926. IEEE, October 2013.
- [14] J. Szeber. Axi4 stream signals. 2019.
- [15] C. Toole. Introduction to axi protocol: Understanding the axi interface. October 2016.
- [16] Xilinx. Axi interconnect v2.1.
- [17] Xilinx. Block memory generator v8.3.
- [18] Xilinx. Data types in vhdl.

- [19] Xilinx. Kintex-7 fpga feature summary.
- [20] Xilinx. Kintex-7 fpgas.
- [21] Xilinx. Memory interface solutions.
- [22] Xilinx. Processor system reset module v5.0.
- [23] Xilinx. Product description.
- [24] Xilinx. Zc706 evaluation board for the zynq-7000 xc7z045 soc user guide.
- [25] Xilinx. *Xilinx UG070 Virtex-4 FPGA User Guide*. December 2008.
- [26] Xilinx. Axi reference guide. March 2011.

Eidesstattliche Versicherung (Affidavit)

Kamtchogom Namtchueng Junior Delrich

197819

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed present Bachelor's/Master's* thesis with the follow title independently and without any unauthori assistance. I have not used any other sources or ; than the ones listed and have documented quotati and paraphrases as such. The thesis in its curren similar version has not been submitted to an audi institution.


Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Extendable Hardware-Based Main Memory Access Snooping for Non-volatile Memory Simulations and Analysis

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

03.09.2020

Ort, Datum
(Place, date)


Unterschrift
(Signature)

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:

Any person who intentionally breaches any regula of university examination regulations relating deception in examination performance is ac improperly. This offense can be punished with a fin up to €50,000.00. The competent administra authority for the pursuit and prosecution of offense this type is the chancellor of TU Dortmund Univer. In the case of multiple or other serious attempt deception, the examinee can also be unenrol section 63, subsection 5 of the North Rh Westphalia Higher Education Act (*Hochschulgesetz*

The submission of a false affidavit will be punis with a prison sentence of up to three years or a fine

As may be necessary, TU Dortmund will make us electronic plagiarism-prevention tools (e.g. "turnitin" service) in order to monitor violations du the examination procedures.

I have taken note of the above official notification:**

03.09.2020

Ort, Datum
(Place, date)


Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.