Karlsruhe Institute of Technology

Institut für Technische Informatik
Chair for Embedded Systems

An RISCV Emulation Board for Non-Volatile Memory

Bachelorarbeit
von

Ruoheng Ma

am Karlsruher Institut für Technologie (KIT)
Fakultät für Informatik
Institut für Technische Informatik (ITEC)
Chair for Embedded Systems (CES)

Erstgutachter:	Prof. Dr. Jörg Henkel
Zweitgutachter:	Prof. Dr. Wolfgang Karl
Betreuer:	Dipl.-Inf. Paul R. Genssler

Tag der Anmeldung:	30.12.2019
Tag der Abgabe:	23.04.2020

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die verwendeten Quellen und Hilfsmittel sind im Literaturverzeichnis vollständig aufgeführt.

Karlsruhe, den 23.04.2020

_____

Ruoheng Ma

Zusammenfassung

Dynamic Random-Access Memory (DRAM) ist seit vielen Jahren als Arbeitsspeicher des Computers eingesetzt. Wegen seines einfachen One-Transistor One-Capacitor Aufbau, niedrigen Kosten und kurzen Zugriffszeit dient DRAM als eine Brücke zwischen dem teureren und schnelleren Static Random-Access Memory und dem günstigeren und langsameren Hard Disk Drive oder Solid-State Drive. Heutzutage treten jedoch intrinsische Probleme des DRAM langsam auf, und zwar seiner großen Energieverbrauch und eingeschränkten Skalierbarkeit. Deswegen ziehen die entstehenden nichtflüchtigen Datenspeicher-Technologien mehr und mehr Blicke der Forscher auf sich.

Aber momentan sind Datenspeicher von meisten diesen Technologien noch nicht verfügbar auf dem Markt. Um die verschiedenen Eigenschaften dieser Technologien zu ermitteln, ist ein Emulationsboard in dieser Bachelorarbeit gebaut. Das Emulationsboard basiert auf dem SoC FU500, der Prozessoren mit einer Open-Source Befehlssatzarchitektur Reduced Instruction Set Computer Five (RISCV) nutzt. Außerdem ist ein Ein-Gigabyte DDR3 Arbeitsspeicher verfügbar auf dem Emulationsboard. Weil die Geschwindigkeit der entstehenden nichtflüchtigen Datenspeicher-Technologien langsamer als die von DRAM ist, lassen sich diese neuen Technologien emulieren durch DRAM. Das Emulationsboard nutzt ein Verzögerungsmodul, das aus einem Schieberegister mit Erweiterung besteht, um den Zugriff auf die Arbeitsspeicher zu verlangsamen und die Datenübertragunsrate des DDR3-Arbeitsspeicher zu verringern. Die Anzahl von Flipflops des Schieberegisters lässt sich von dem Benutzer beim Kompilieren einstellen und somit verschiedene Verzögerungszyklen erreichbar sind.

Außerhalb dem Aufbau des Emulationsboard ist ein Benchmark-Paket aufgebaut. Das Benchmark-Paket umfasst zwei Open-Source Benchmarks, nämlich MiBench und Tinymembench, und ein selbst geschriebenes Benchmark für Schreib-Latenz-Test. Das Emulationsboard ist mit diesen zwei Benchmarks evaluiert.

Abstract

As DRAM is facing some problems such as limited scalability and huge power consumption, researchers are turning their focus on the emerging Non-Volatile Memory (NVM) technologies, like Phase-Change RAM (PRAM), Spin-Transfer Torque RAM (STT-RAM) and Resistive RAM (ReRAM), to get rid of these problems. Compared to DRAM, NVM is non-volatile and has higher density. Moreover, in contrast to flash memory, NVM are byte-addressable. Therefore, they can be considered as a candidate for replacing traditional DRAM. In order to explore the characteristics of different NVM technologies, in this thesis, an emulation board with parameterizable memory access latency is built upon a platform called FU500, which utilizing the open-source instruction set architecture RISCV, to ensure flexibility and extensibility for further research. Moreover, a benchmark suite comprising MiBench and Tinymembench is also built for benchmarking this board.

Contents

List of Acronyms

1T1C one-transistor one-capacitor. 1

ADPCM Adaptive Differential Pulse Code Modulation. 29

AR Read Address channel. VII, XIII, XIV, 7, 10, 12, 14, 15, 22–25, 31, 35

AW Write Address channel. VII, XIII, XIV, 7, 12, 14, 15, 22–25

AXI Advanced eXtensible Interface. VII, 5, 8, 10, 12, 18, 19, 21, 26, 31

AXI4 Advanced eXtensible Interface 4. 5, 7, 8, 10, 18

B write response channel. VII, XIII, XIV, 7, 10, 22–25, 31, 35

DDR Double Data Rate. VII, 17, 18

DRAM Dynamic Random-Access Memory. V, 1, 10–13, 18, 37

DRC Design Rule Checking. 18

ETH Zürich Eidgenössische Technische Hochschule Zürich. 17

FeRAM Ferroelectric RAM. 2

GST $Ge_2Sb_2Te_5$. 1, 2

ISA Instruction Set Architecture. 2, 5, 17, 37

LLC Last Level Cache. 11, 12, 16, 18, 29

MIG Memory Interface Generator. 13, 16, 21

MTJ Magnetic Tunnel Junction. 2

NVM Non-Volatile Memory. V, 1, 2, 5, 11–13, 17, 37

PCM Pulse Code Modulation. 29

PL Programable Logic. 12, 13

PRAM Phase-Change RAM. V, 1, 2

PS Processing System. 12

PULP Parallel Ultra Low Power. 17

R Read data channel. 7, 22

ReRAM Resistive RAM. V, 2

RISCV Reduced Instruction Set Computer Five. V, 2, 5, 17, 37

SoC System on Chip. 2

SRAM Static Random-Access Memory. 1

STT-RAM Spin-Transfer Torque RAM. V, 1, 2

UCB University of California, Berkeley. 5

W Write data channel. VII, XIII, XIV, 7, 22–25

List of Tables

List of Figures

Chapter 1

Introduction

Traditional Dynamic Random-Access Memory (DRAM) is playing an important role as main memory in computer systems. Due to its low cost, simple construction, high capacity and byte-addressability, it serves as a bridge between expensive but fast Static Random-Access Memory (SRAM) and cheap but slow flash memory or hard disk drive. But in recent years, its intrinsic shortcomings are hindering it from further development.

First, DRAM will reach its scaling limit by 2024[1]. As DRAM keeps being scaled down, the cell capacitance will also be scaled down, which is a crucial parameter determing the storing-ability of a cell. Since DRAM is in a one-transistor one-capacitor (1T1C) structure and utilizes the capacitance of a cell to store data, if the cell capacitance is too low, many problems will occur. Data retention time will become too short to meet JEDEC specification of refresh period, as well as sensing margin will be insufficient for accessing a DRAM cell[2].

Moreover, interference between neighboring cell transistors will increase and lead to unexpected degradation. Furthermore, requirement for high performance peripheral transistor is also preventing DRAM to be further scaled down[2]. Though scaling down DRAM continually is difficult, at the same time, demand on main memory with larger capacity and higher density is always increasing.

Second, due to the 1T1C structure of DRAM cell, it utilizes the charge state of the capacitor to store information. As the electric charge of the capacitor will slowly leak off, in order to preserve the information, a DRAM cell has to be charged periodically. This procedure is called self-refresh and consumes a great amount of eletrical power. In the meanwhile, main memory with lower power-consumption is needed both in general computers and embedded systems, what is becoming difficult to be fulfilled by DRAM.

Therefore, more and more researchers are turning their focus on some emerging Non-Volatile Memory (NVM) technologies, such as Phase-Change RAM (PRAM)[3] and Spin-Transfer Torque RAM (STT-RAM)[4], to find a way to satisfy the increasing demand on main memory. These NVM technologies are byte-addressability. In contrast to DRAM memory, which needs to be refreshed after each short time interval, these memory technologies can provide non-volatile storage so that refresh procedure is no longer needed and power consumption for refreshing is saved. Moreover, they offer greater density than DRAM and faster access time than flash memory or hard disk drive. With such advantages, the emerging NVM technologies are attracting attention of researchers to themselves.

Different NVM is derived from different physical principles. PRAM utilizes a chalcogenide glass, whose state can be changed to store information. $Ge_2Sb_2Te_5$ (GST) is the

most commonly used material for chalcogenide glass. It is a material which can switch between amorphous and crystalline state by heating it in specific ways. When GST is in amorphous state, it produces high resistance, while in crystalline state, it is in low-resistance state. Hence it can represent binary 0 and 1 with its two states. Furthermore, it can even be in a number of distinct intermediary state, such that being able to hold multiple bits in just one memory cell. Currently, PRAM is already available[5] on the market.

STT-RAM utilizes Magnetic Tunnel Junction (MTJ) as its memory cell to store data. MTJ consists of a free ferromagnetic layer, a reference ferromagnetic layer and a tunnel barrier layer. The magnetic orientation of the reference layer is fixed while the one of the free layer can be changed to have a magnetic field being parallel or anti-parallel to the one of reference layer. With the magnetic field of the free layer being parallel to the one of fixed layer, the MTJ will produce low eletrical resistance and represent binary 0, while it will has high eletrical resistance and represent binary 1 if it is anti-parallel to the reference layer. With these two states of the magnetic field, data can thereby be stored.

Except for the two memory types listed above, there still exist some other NVM technologies, such as Ferroelectric RAM (FeRAM)[6], ReRAM[7] and so on. As these NVM technologies are derived from different physical principles, they have different characteristics. Since most of them are currently prototypical NVM technologies and are not available on the market (except for PRAM), in order to make further research on them possible as well as fully explore their strengths and deficiencies, a hardware emulation platform is needed.

Compared to simulation platform, an emulation platform can provide faster experiments, tangible results and often reproducible debug function[8]. Moreover, though there already exist some prototypical emulator designs, such as Quartz[9], Write-back Aware Emulator[10], TUNA 1/2[11][8] and NVMM emulator[12] with coarse- and fine-grain delay injection, but all of them are utilizing processors based on close-source Instruction Set Architecture (ISA), which may affect the flexibility and extensibility of further research. Thus, in this thesis, a new prototypical NVM emulation board was built.

The emulation board is based on the Reduced Instruction Set Computer Five (RISCV) System on Chip (SoC) FU500, which is implemented on Xilinx Virtex-7 FPGA VC707 Evaluation Kit. Additional delay modules with parameterizable delay cycles and asymmetric read/write latency adjustment are inserted to this emulation board to delay memory accesses so that system with NVM of different bandwidth and latency can be emulated. Users can tune this emulation platform to achieve different latency models, emulate and analyse the behaviours of different NVM.

Besides building the emulation board, a benchmark suite comprising MiBench and another open-source benchmark called Tinymembench was also packaged to benchmark and evaluate the emulation board. Important behaviour parameters like memory access latency, memory bandwidth and application run time can be reported by this benchmark-suite.

The contribution of this thesis is summarized as follows.

- An emulation board was built upon FU500 platform, which utilizes the open-source ISA RISCV. With parameterized and adjustable main memory read/write latency, it can be used to provide a hardware environment for further research on NVM technologies, for evaluating performance of operating systems and applications aiming at running upon the emerging NVM.

- A benchmark suite was built to benchmark and evaluate the emulation board.

- The memory access latency, memory bandwidth and application run time reported by the benchmark suite was analysed and the relation between system performance and additional delay cycles was figured out.

Chapter 2

Background

2.1 SiFive FU500 Platform

The emulation board is based on SiFive FU500 platform, whose SoC is generated by Rocket Chip SoC Generator developed by University of California, Berkeley (UCB) in 2016. As seen in Figure 2.1, the FU500 platform has a U54-MC coreplex as its central processing unit, which is based on RISCV ISA and consists of one E51 core and four U54 cores. According to the manual[14], each U54 core owns 32KB 8-way L1 instruction cache and 32KB 8-way L1 data cache. E51 core owns 32KB 2-way L1 instruction cache and 64 KB L1 data tightly integrated memory. All the cores share a 2MB 16-way L2 cache. The U54-MC coreplex supports physical memory protection and virtual memory management, therefore Linux can boot on it. Components of the platform such as cores, memory controller and peripherals are connected via TileLink, a protocol designed by UCB to be a substrate for cache coherence transactions[15].

2.2 Xilinx Virtex-7 FPGA VC707 Evaluation Kit

The Xilinx Virtex-7 FPGA VC707 Evaluation Kit is a full-featured, highly-flexible, high-speed serial base platform utilizing the Virtex-7 XC7VX485T-2FFG1761C FPGA, which contains 485,760 logic cells and 700 pins. As shown in Figure 2.2, the evaluation board consists of a 1GB DDR3 memory, SD card slot, FMC connectors and other common peripherals. As the RTL code of FU500 platform is open-source and designed to be mapped onto a VC707 evaluation board, thus it is set as the base board of the NVM emulation board.

2.3 AXI Protocol

In order to make a reliable and effective connection between the memory bus and memory controller on the FU500 platform, the AXI4, which is a part of the ARM Advanced Microcontroller Bus Architecture specification, is introduced into the design. It is one of the most widely used protocol in today's SoC design because of its simplicity, royalty-free of use and reliablity. This section will provide an overview of how Advanced eXtensible Interface 4 (AXI4) works.

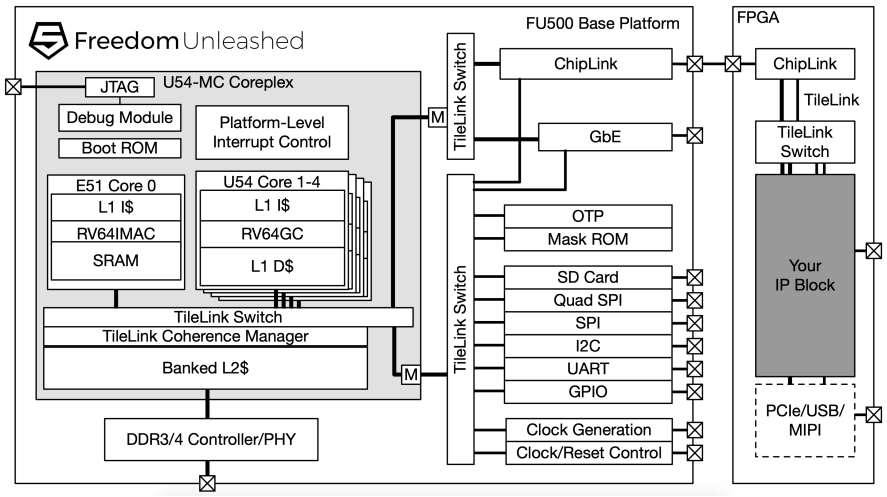


Figure 2.1: SiFive FU500 architecture[13]

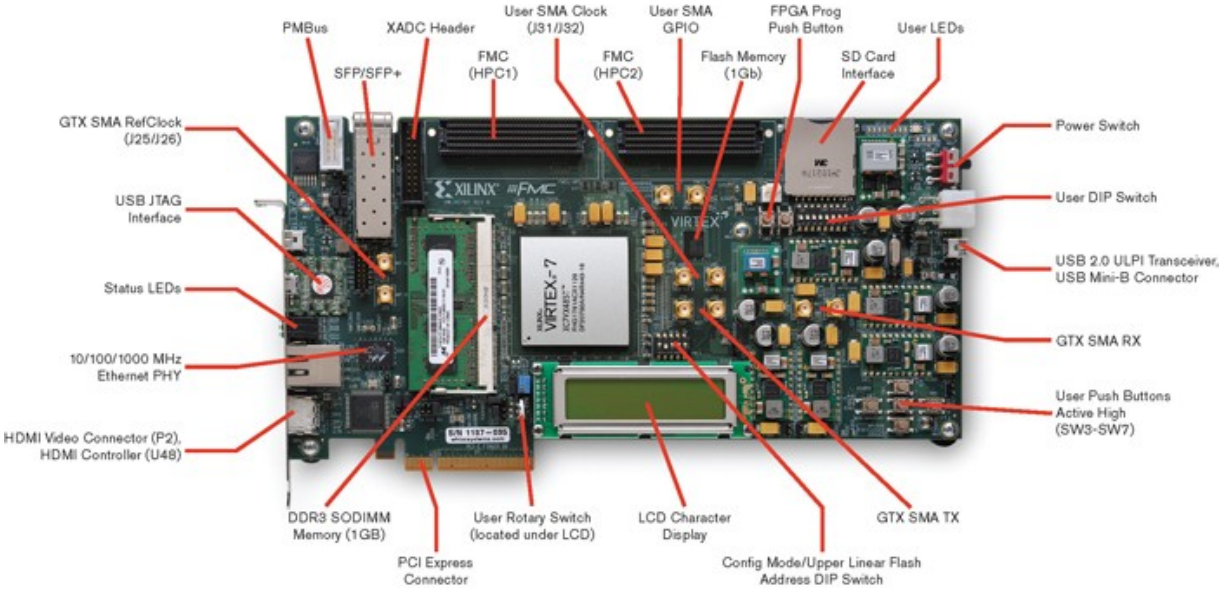


Figure 2.2: VC707 Evaluation Board[16]

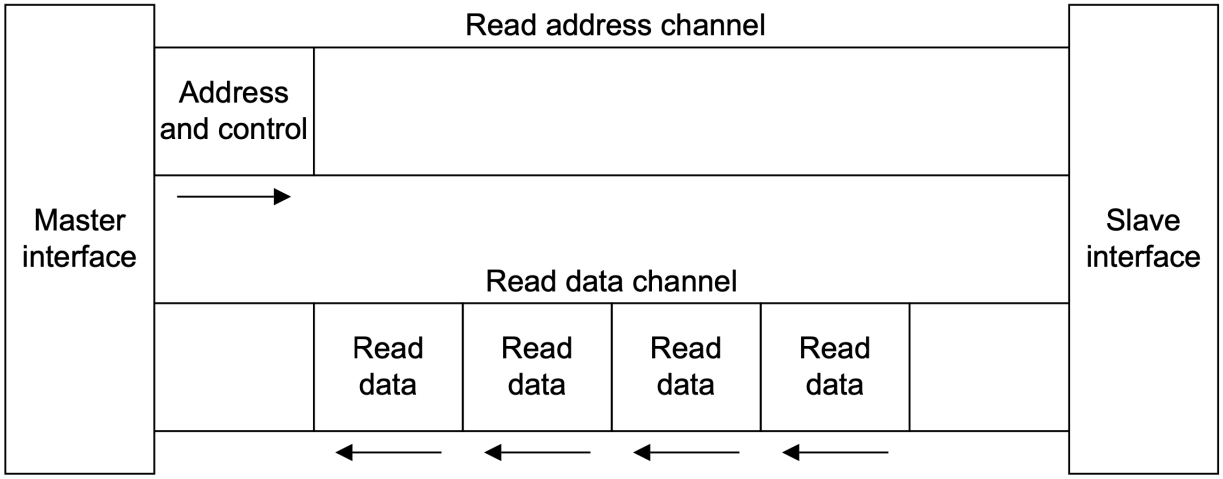


Figure 2.3: AXI4 read transaction[17]

2.3.1 The Five Channels of AXI4

According to the AXI4 standard, communication is established between two chip components, one among which has a master interface and the other has a slave interface. Five channels are set up for data transfer, which are:

- Read Address channel (AR)
- Read data channel (R)
- Write Address channel (AW)
- Write data channel (W)
- write response channel (B)

These five channels can be grouped into two groups in terms of their purposes: AR and R are for read transaction; AW, W and B are for write transaction. AR and AW are for read/write address and control signals transfer, R and W are for read/write data transfer, and B is for write reponse transfer, which indicates the completion of a write transaction. Each channel has a number of control signals, data signals and xValid/xReady signals. As Figure 2.3, 2.4 shows, in read transaction, master interface will send address and control signals to slave interface through AR, then slave interface response with data from the specified memory address via R. In write transaction, address and data will be placed in AW and W respectively by master interface and slave will response with the completion state of the write transaction through B.

Therefore, data will be transfered in R channel only after the address data is transfered successfully in AR. But there is no such restriction between AW and W, because both write address and write data come from the master interface. Only the response signal of B will be transfered from slave interface to master interface after the transactions in AW and W are completed. These dependencies are achieved by the xValid/xReady signals, which also contribute to the two-way handshake mechanism below.

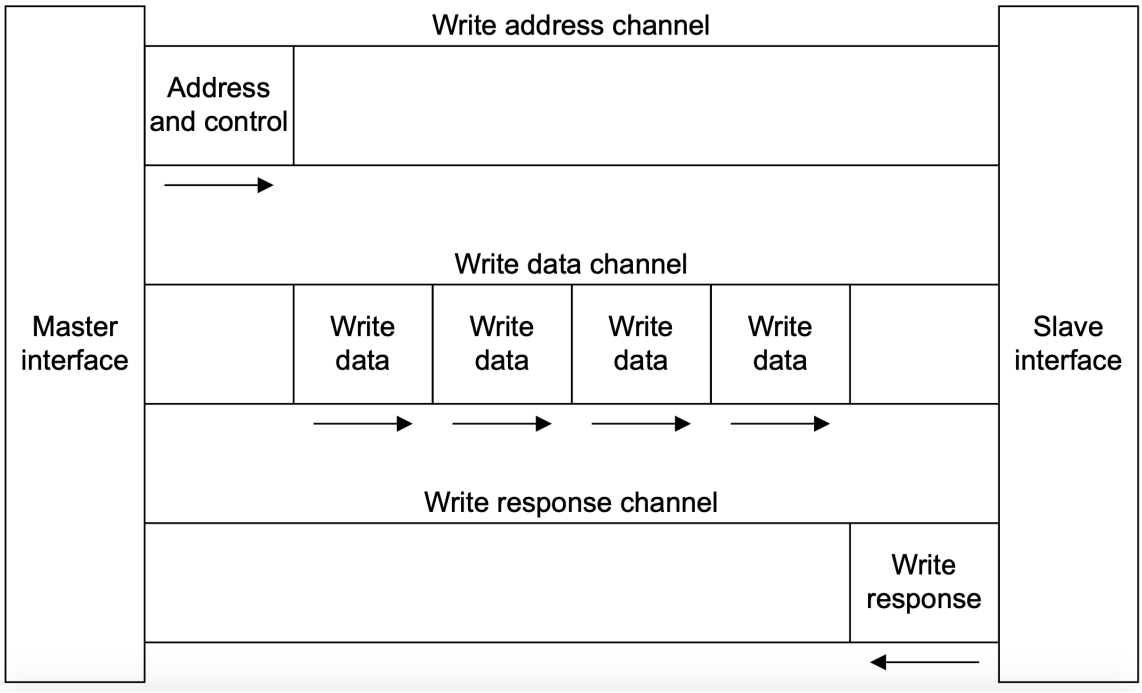


Figure 2.4: AXI4 write transaction[17]

2.3.2 Two-Way Handshake Mechanism

AXI4 has adopted a two-way handshake mechanism to enable flow control. Each channel of AXI4 utilizes its own xValid/xReady signal to ensure every transfer will only start after the two-way handshake is finished, which indicates that both sides of the connection are ready to send or receive data. The xValid signal is set by the side that trigger a transaction and the xReady signal is set by the receiving side. As Figure 2.5 depicts, data transfer starts immediately when the xValid/xReady signals are both asserted. The control signals and data must be retain until the next rising edge of clock after the assertion of the xValid/xReady signals. In order to prevent deadlock, the xValid signal must be asserted once the data to be transfered is ready and can not deassert after that, while the xReady signal doesn't have this restriction and can be asserted or deasserted any time.

Furthermore, there are dependencies between handshake signals of different channels. As shown in Figure 2.6 and 2.7 (single-headed arrows point to signals that can be asserted before or after the signal at the start of the arrow while double-headed arrows point to signals that must be asserted only after assertion of the signal at the start of the arrow), in read transaction, the RValid signal depends on the assertion of ARValid and ARReady signal; in write transaction, the BValid signal can only be asserted when AWValid/AWReady and WValid/WReady are asserted.

2.3.3 Asynchronous Bridge

As it is illustrated in Figure 2.8, when a master interface needs to establish a connection with a slave interface in different clock domain, they need an asynchronous AXI-to-AXI bridge to solve this problem. The master interface and the slave interface will connect to

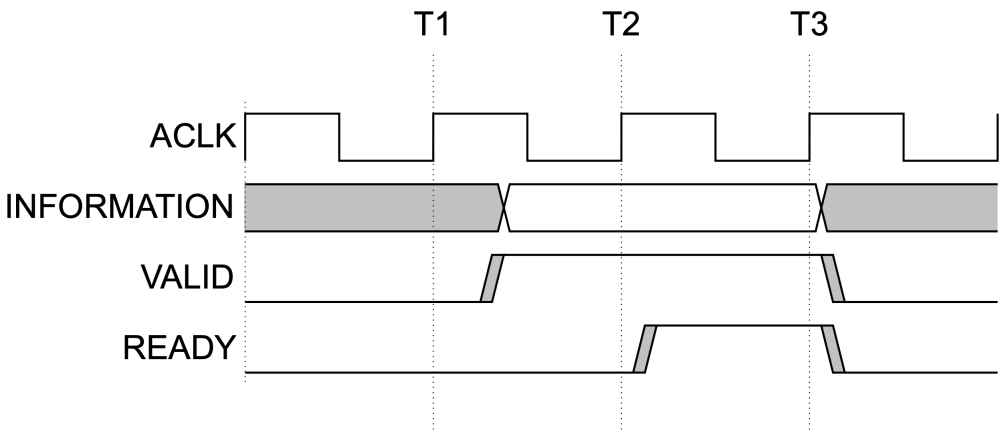


Figure 2.5: Two-way handshake mechanism[17]

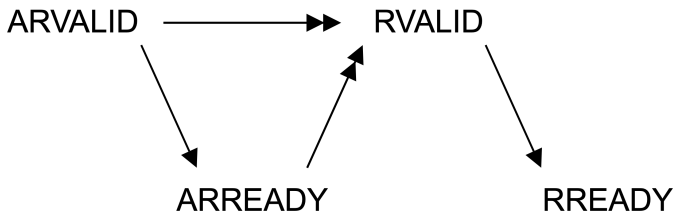


Figure 2.6: Read dependencies of handshake signals from different channels[17]

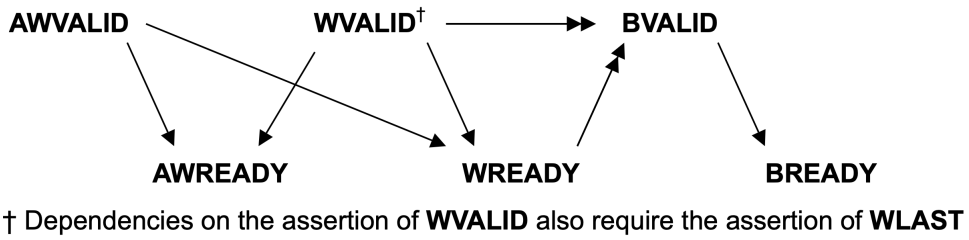


Figure 2.7: Write dependencies of handshake signals from different channels[17]

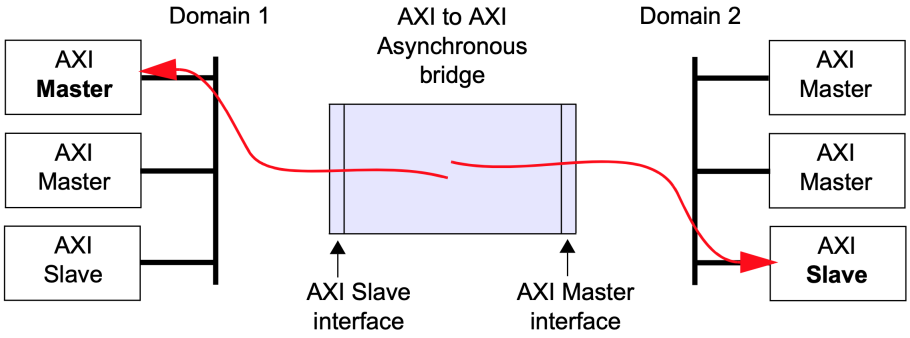


Figure 2.8: Connection of AXI Subsystems through a Asynchronous AXI to AXI Bridge[18]

the corresponding interface of the asynchronous bridge via AXI bus and send/receive data to/from each other, while the bridge will solve the different clock domain problem.

On the FU500 platform, the memory bus is implemented with TileLink bus, which is a chip-scale interconnect standard designed for use in SoCs. Then the memory bus connects itself to a wrapper of the memory controller, which provides buffer and TileLink-to-AXI4 infrastructure to link both interconnect standards. Due to the incompatibility of the clock frequency of the memory controller and DRAM, an asynchronous bridge is employed in this wrapper to deal with request sent to the memory controller and response sent back to the memory bus. In the emulation board, the delay module is applied on the AXI4 bus, to be specific, on the AR and B, between the memory controller and the sink node of the asynchronous bridge.

Chapter 3

Related Work

There already exist some prototypical NVM emulators, such as Quartz[9], Write-back Aware Emulator[10], TUNA 1/2[11][8] and NVMM emulator[12] with coarse- and fine-grain delay injection. A short introduction to their respective basic design and precision will be given below.

3.1 Quartz

Quartz is a lightweight performance emulator for NVM based on Intel processors. It is developed in 2015 and its basic idea is to dynamically inject software created delays after a specified time interval, which is called epoch. As seen in Figure 3.1, applications will run without any delay in an epoch. When Last Level Cache (LLC) miss occurs, which indicates that a memory access should be generated, a counter will record it until the end of the epoch. When the epoch elapsed, the emulator will figure out the extra latency Δ_i should be injected to the system based on the following formula:

$$\Delta_i = M_i \cdot (NVM_{lat} - DRAM_{lat}) \tag{3.1}$$

where i represents the i_{th} epoch, M_i represents the recorded memory accesses, NVM_{lat} represents the average NVM access latency and $DRAM_{lat}$ represents the average DRAM access latency. The system will stall for the additional delay cycles and start a new epoch after that.

As a result, this emulator delivers an average overhead of 4000 cycles for each creation of epochs. Such calculation overhead will be compensated by decreasing the injected delay.

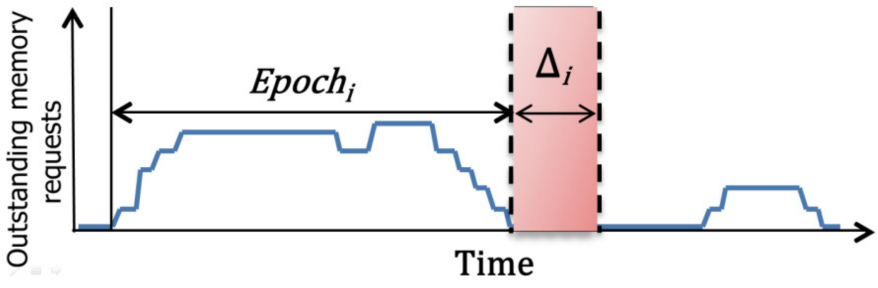


Figure 3.1: Delay injection model of Quartz[9]

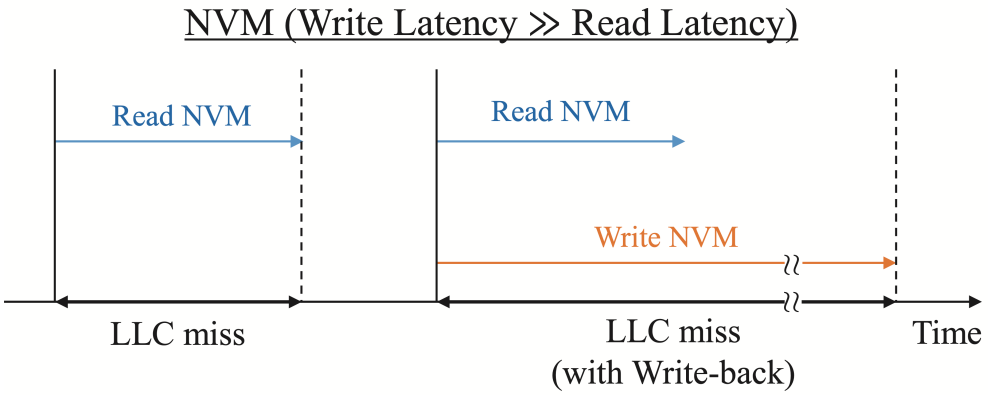


Figure 3.2: Different read and write latency of write-back aware emulator[10]

If the injected delay is less than this overhead, this overhead will be carried over to the next epoch. This emulator can deliver an emulation result with error rate between target latency and measured one of less than 9% on Intel Sandy Bridge processors, 2% on Ivy Bridge processors and 6% on Haswell processors respectively.

3.2 Write-back Aware Emulator

Write-back Aware Emulator is another emulator which is developed in 2017 and also injects software created delays to the system. It is also based on Intel processors. Its latency-injection model is based on the epoch model of Quartz, but has add an extension to it. The emulator divides the LLC misses into two groups: one group for reading data and one group for writing data. As seen in Figure 3.2, this emulator defines different NVM latency for the two groups because the write latency in a NVM environment is much greater than the read latency. The additional delay is calculated with the following formula:

$$\Delta_i' = MA_i^{WB} \cdot (NVM_{lat}^{Write} - DRAM_{lat}) + MA_i^{RO} \cdot (NVM_{lat}^{Read} - DRAM_{lat}) \quad (3.2)$$

With this extension the write-back aware emulator can perform a more precise NVM emulation.

As shown in Figure 3.3, this emulator can produce measured write latency of 112.1ns, 244.6ns, 376.4ns, 510.0ns and 642.5ns for target write latency of 100ns, 200ns, 300ns, 400ns und 500ns, leading to error rate of 12.1%, 22.3%, 25.5%, 27.5% and 28.5% respectively.

3.3 TUNA 1/2

TUNA 1/2 are ARM-FPGA-based emulation boards developed respectively in 2014 and 2017. As shown in Figure 3.4 and 3.5, TUNA 1 is an emulation board based on Xilinx Zynq XC7Z020, which consists of a Processing System (PS) based on an ARM processor and a FPGA Programable Logic (PL). Both the PS and the PL have their own DRAM. The emulation model is implemented on the PL-side DRAM. As the DRAM controller utilizes AXI bus to communicate with the PL-side DRAM, a delay module is placed inbetween to delay handshake signals on AW and AR of the AXI bus so that

Emulated latency	Measured latency	error
100 ns	112.1 ns	12.1 ns (12.1 %)
200 ns	244.6 ns	44.6 ns (22.3 %)
300 ns	376.4 ns	76.4 ns (25.5 %)
400 ns	510.0 ns	110.0 ns (27.5 %)
500 ns	642.5 ns	142.5 ns (28.5 %)

Figure 3.3: Error rate between target write latency and measured write latency[10]

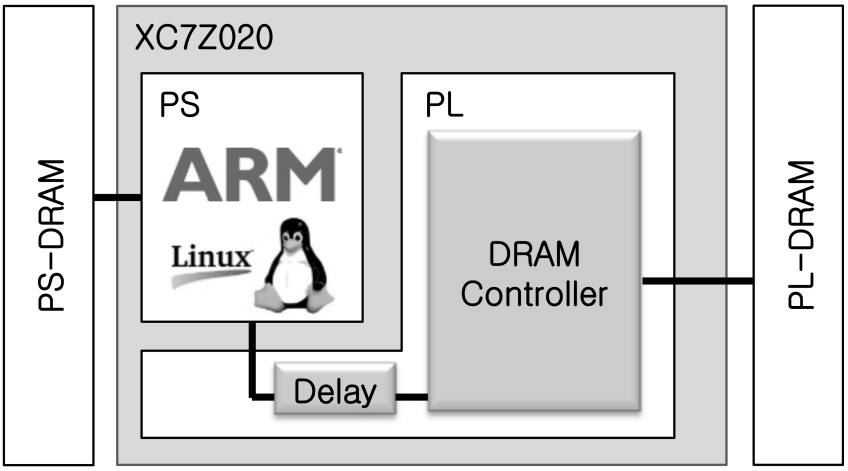


Figure 3.4: TUNA 1 architecture[11]

address information tranfer of read/write transactions are delayed independently. The impact of additional delay cycles on memory read/write latency and bandwidth can be seen in Figure 3.6, 3.7 and 3.8.

Furthermore, the researchers of TUNA 1 have implemented a separate NVM power switch for PL memory so that if main power is down, data stored in PL memory will not get lost and thus non-volatility of NVM is also emulated.

But this emulator board can only perform a rough latency emulation due to its coarse implementation. As DRAM has rank/bank parallelism and memory controller has buffers and schedulers to achieve better performance, in order to get a more accurate latency emulation, the researchers of TUNA 1 have developed the TUNA 2 later. They have augmented the RTL code of the Memory Interface Generator (MIG) and adjusted the memory timing parameters accordingly to construct a better emulation model. In this way, the emulator board can inject delay to ACT and PRE commands that are issued by the memory controller when triggering a read or write transaction. In addition, user can adjust the memory timing parameters tRCD and tRP in run time via U-boot or mmap system call to change delay cycles on ACT and PRE commands. The average response time of random memory read/write access in terms of different tRCD and tRP is shown in Figure 3.9, 3.10.

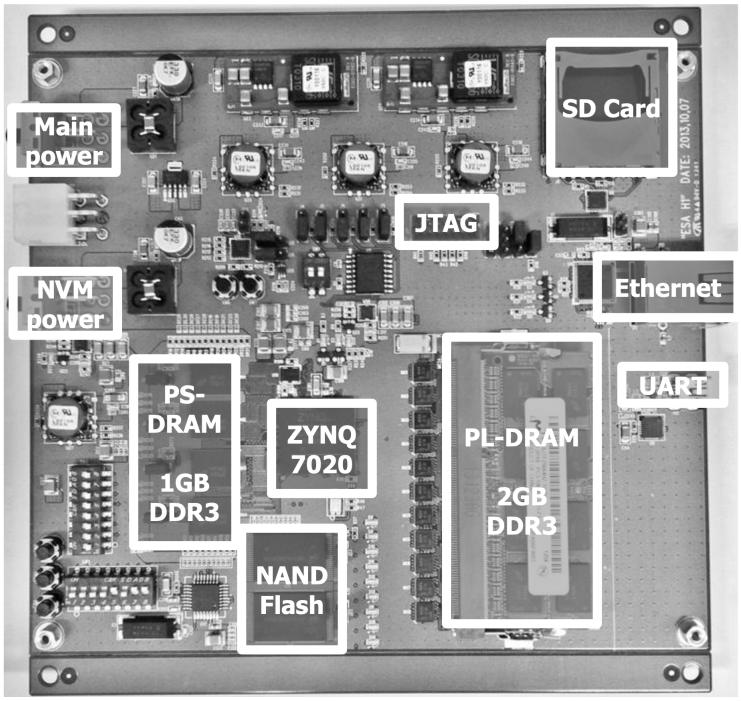


Figure 3.5: TUNA 1 board[11]

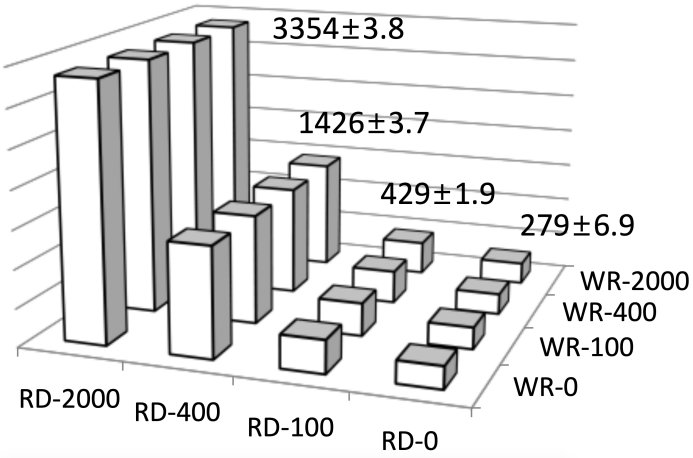


Figure 3.6: TUNA 1 read latency with different additional AR and AW delay cycles[11]

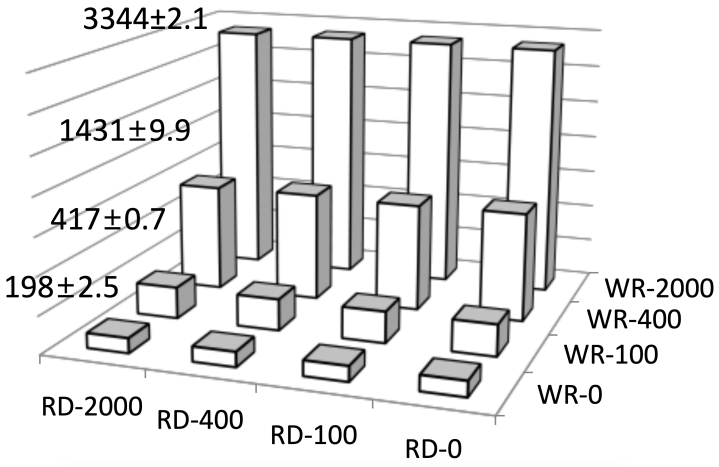


Figure 3.7: TUNA 1 write latency with different additional AR and AW delay cycles[11]

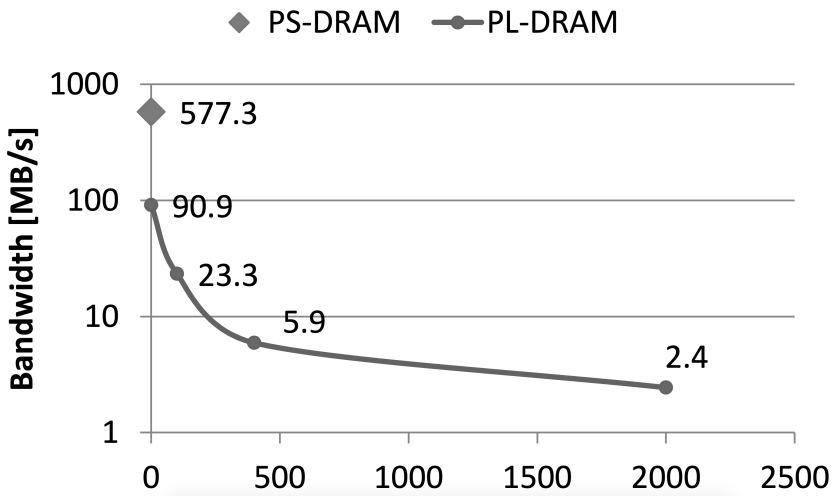


Figure 3.8: TUNA 1 bandwidth of 4kB memset with different delay cycles[11]

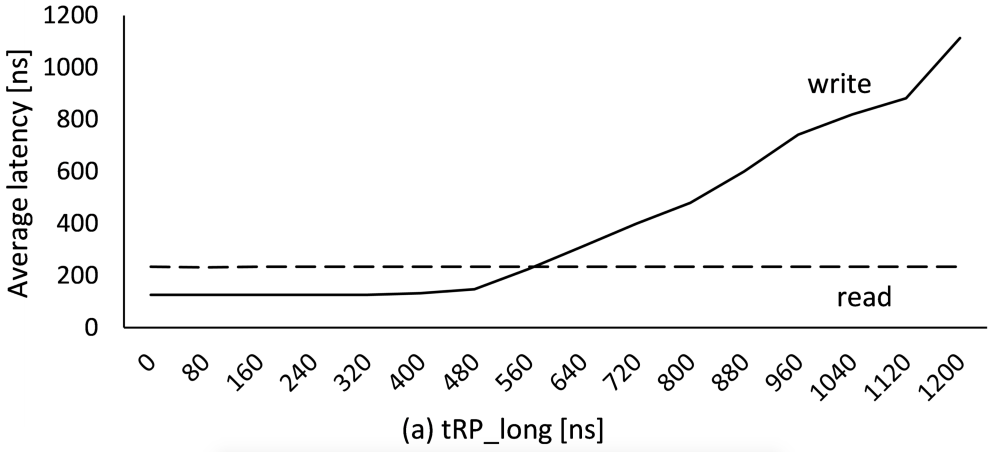


Figure 3.9: TUNA 2 average response time of random memory access in terms of different tRP_long[8]

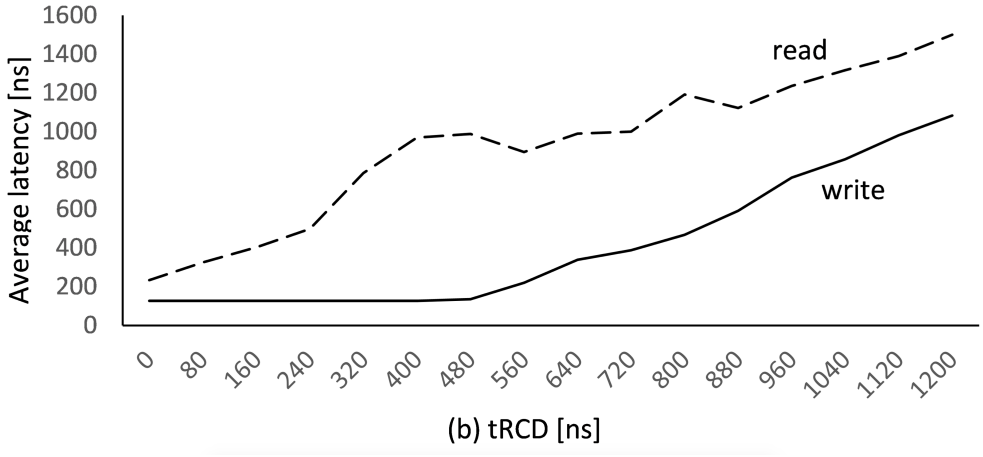


Figure 3.10: TUNA 2 average response time of random memory access in terms of different tRCD[8]

NBANK	Coarse-Grain [ns]	Fine-Grain [ns]
1	1071.0	634.0
2	1068.9	405.0
3	1061.6	399.5
4	1061.6	423.0

Figure 3.11: NVMM average read latency in terms of N memory banks[12]

3.4 NVMM emulator with fine-grain delay injection

The NVMM emulator with coarse- and fine-grain delay injection is also an ARM-FPGA-based emulation board, which is developed in 2019. It inherits mainly the idea of TUNA 1/2, but extends it by providing custom memory allocation and cache eviction functions for the emulation environment.

The coarse-grain delay injection of this emulator board is implemented by inserting a delay module between the LLC and the MIG, while the fine-grain delay injection is implemented by modifying the RTL code of MIG and the memory timing parameters tRCD and tRP. With this emulator, the read/write latency of memory with different number of banks is also benchmarked. The result is shown in Figure 3.11, 3.12.

NBANK	Coarse-Grain [ns]	Fine-Grain [ns]
1	1072.6	3468.2
2	1067.3	1827.3
3	1060.9	1337.6
4	1061.8	1357.7

Figure 3.12: NVMM average write latency in terms of N memory banks[12]

Chapter 4

Design

As discussed in the previous chapter, an emulation board with an open-source ISA based SoC should be of benefit to further research, therefore the emerging RISCV based SoC platform is chosen to be the base platform for implementing the emulation board. To achieve the emulation, delay modules are inserted to the platform so that DDR memory can emulate the slower emerging NVM technologies. In this chapter, the selection of an appropriate RISCV based platform and a discussion about the two options of placements of the delay modules are presented.

4.1 RISCV Platform

There are already some RISCV projects under development. At the beginning of the design, the lowRISC project[19], the Parallel Ultra Low Power (PULP) Ariane project[20] and the SiFive FU500 project[21] were under consideration. lowRISC is a not-for-profit company in Cambridge, UK. After investigation, it is found that the lowRISC project only maintains a type of RISCV core developed in the PULP project and does not provide any SoC platform currently. Thus implementation upon the lowRISC project is not feasible.

PULP is a RISCV joint project started and developed by Eidgenössische Technische Hochschule Zürich (ETH Zürich) and University of Bologna. In its Ariane project, a Linux-capable core called Ariane is under development, but as shown in its Github website, building of this core is still problematic. Furthermore, PULP does not provide a platform that runs on Ariane. Therefore, the PULP Ariane project is not suitable to the implementation of the emulation board.

SiFive is a company founded by Krste Asanović, who is currently the chairman of the board of the RISCV foundation. SiFive FU500 SoC can boot Linux and shares the same design of FU540 SoC, the first Linux-capable RISCV SoC in the world, and can be evaluated on Xilinx VC707 evaluation board. In addition, the RTL code of FU500 is easily available on Github with active community support. Most of all, the FU540 is already taped out and on sale, which means that FU500 also has a robust, reliable and stable design. Hence this platform is chosen as the base platform for the emulation board.

4.2 Placement of Delay Module

In order to achieve additional latency for read/write memory access, inserting a hardware delay module with parameterizable delay cycles is a feasible solution, not only because it adds less overhead to the system, but also because it outputs a more precise emulation. Under consideration of not bringing too many unpredictable factors to the system and not making the modification of the platform too complicated, the delay module should be placed somewhere outside the CPU. So the interconnect that connects the memory controller with the other components of the system would be a good choice for placement of the delay module.

For this emulation board design, there are two placement options for the delay modules. One of them is placing the delay modules on the DDR3 bus that connects the memory controller and DRAM, the other one is inserting them on the AXI4 bus, which is a part of the interconnect that connects the memory controller with the LLC.

4.2.1 Delay Module on DDR Bus

Theoretically placing the delay modules on the DDR3 bus can deliver a more accurate emualtion result, because on this bus there are no extra system components or even a single electronic component. If a delay module is placed there, the factors that influence the relationship between the delay cycles and the resulting latency can be minimized and the impact of each additional delay cycle is more predictable, In addition, the delay module would be adjusted by delay command signals sent from the memory controller to DRAM, instead of by the delayed two-way handshake signals of the AXI4 bus. In this way, the behaviour of the DRAM could be totally under control, so that fine-grain delay injection can be realized.

4.2.2 Delay Module on AXI Bus

Besides inserting the delay module on the DDR3 bus, placing it on the AXI bus is a feasible alternative. As AXI4 bus is a technically mature bus widely used in on-chip communication and its functionalities are well documented, it eases the difficulty of the implementation. Besides, the AXI4 bus is not connected with any physical pins of the FPGA, hence it does not need to make modification on the FU500 route and placement design. Therefore, placing delay modules on AXI4 bus and exploiting its strengths is also a good solution.

4.2.3 Final Decision of Placement

Even though a delay module placement on DDR3 bus can achieve a theoretically better result, such a placement is difficult to be realized on FU500 platform. First, for such a design, the floorplan of the platform should also be adjusted accordingly. Otherwise, as shown in a try of this implemention, Vivado would throw Design Rule Checking (DRC) errors regarding the ports that are connecting with the delay module. Second, when a memory controller command is delayed, the timing parameters of the memory controller should also be adjusted accordingly to synchronize the behaviour of the memory controller and the memory cell. Both changing the floorplan and adjusting the memory timing parameters would lead to unpredictable consequences and are time-consuming. In contrast,

placing the delay module on AXI bus does not need much effort to modify the platform, so that the main focus can be concentrated on implementing the delay module and development effort and time can be saved. Thus, implementing a delay module on AXI bus is the final decision of the placement. In the next chapter, details of delay module implementation will be further presented.

Chapter 5

Implementation

In this chapter, the exact location of the delay module, its implementation in detail and the way it works will be presented.

5.1 Location of the Delay Module

On the FU500 platform, an AXI asynchronous bridge is employed to connect the MIG island module, which is a wrapper of the blackbox module containing the actual memory control unit, and the other neccesary components of the memory controller. As shown in the Figure 5.1, the delay module is placed between the sink node of the asynchronous bridge and the blackbox module because such a way does not require great change in the Chisel source code of the FU500 platform.

5.2 Selection of Control Signals

Data transmission on the AXI bus is controlled by various control signals defined in the AXI protocol. Thus the goal of delaying data transmission is achieved by delaying specific control signals of the five channels of the AXI bus. There are three optional groups of control signals to be delayed.

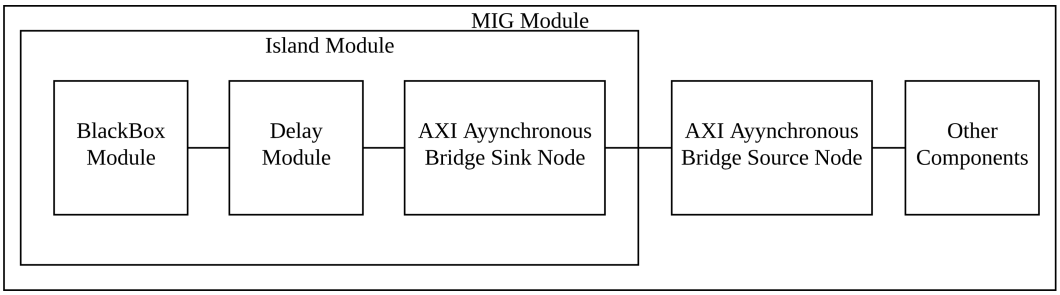


Figure 5.1: Delay module placement on FU500

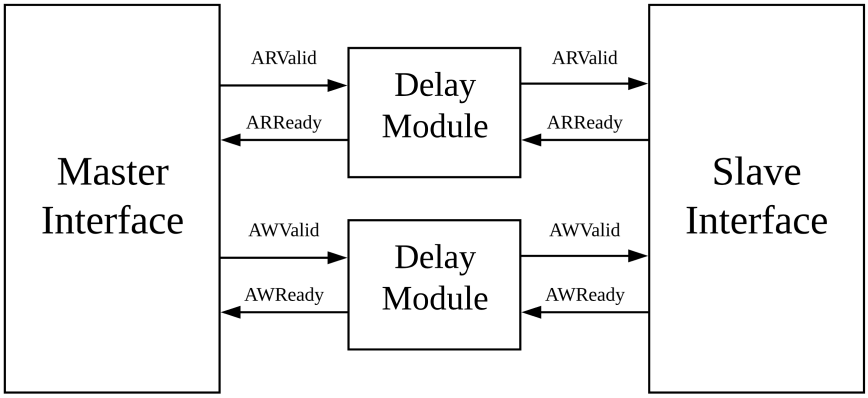


Figure 5.2: Delay module on AR and AW

5.2.1 First Option: AR and AW

The first option is delaying the xValid/xReady signals of the AR and AW channels, inspired by [11]. As mentioned in chapter 2, in read transaction, data in R will only be transfered when the address in AR are transfered successfully, while write transaction can only be completed when both address/control information in AW and data in W are tranfered to memory controller. In addition, the transfer of read address and write address obeys the two-way handshake mechanism and will only succeed after the both interfaces have found that the ARValid/ARReady and AWValid/AWReady are asserted. Therefore, as shown in Figure 5.2, by the first option delay modules that adds extra delay cycles to xValid/xReady signals are inserted both on AR and AW.

5.2.2 Second Option: AR, AW and W

The second option can be seen in Figure 5.3. Similar to the first option, in this option delay modules are inserted on AR and AW channels. Additionally, the xValid/xReady signals of W channel are also delayed, so that the whole transfer issued by the master interface is delayed. This option can be considered as an alternative of the first option.

5.2.3 Third Option: AR and B

The last option is delaying the xValid/xReady signals of the AR and B channels. By delaying AR channel, the read transaction will be delayed, while by delaying the B the response of a write transaction will be delayed, so that the write transaction is also delayed.

5.2.4 Final Decision

Although the first option is used in [11] and can already deliver an emulation without problems, there is a drawback that can impact its emulation accuracy greatly on this platform and therefore the emulation board can not deliver the desired latency and bandwidth by write transaction. Assume that the master interface would like to start a write transaction. As in chapter 2 described, unlike in a read transaction, in a write transaction the address/control information and data can be tranfer to slave interface independently. Hence, when the master interface completes its preparation of address/control information

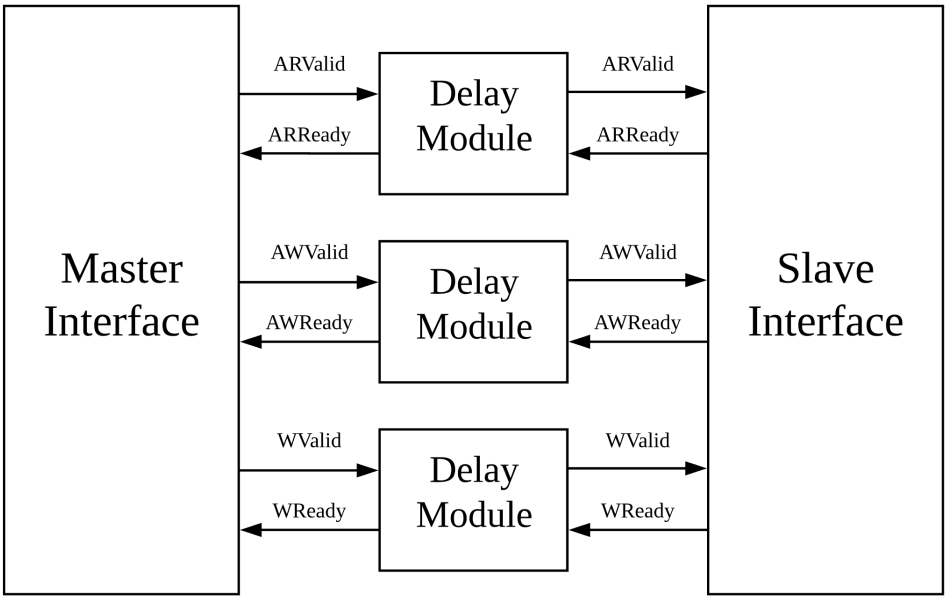


Figure 5.3: Delay module on AR, AW and W

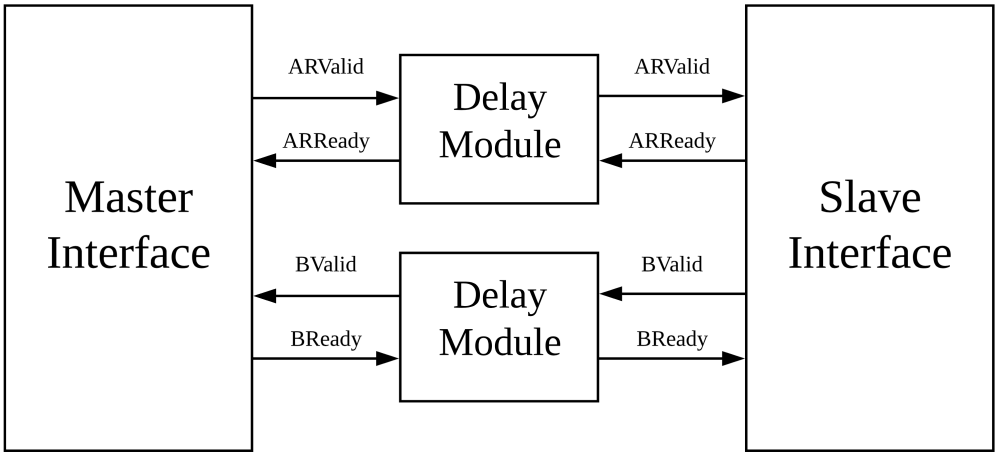


Figure 5.4: Delay module on AR and B

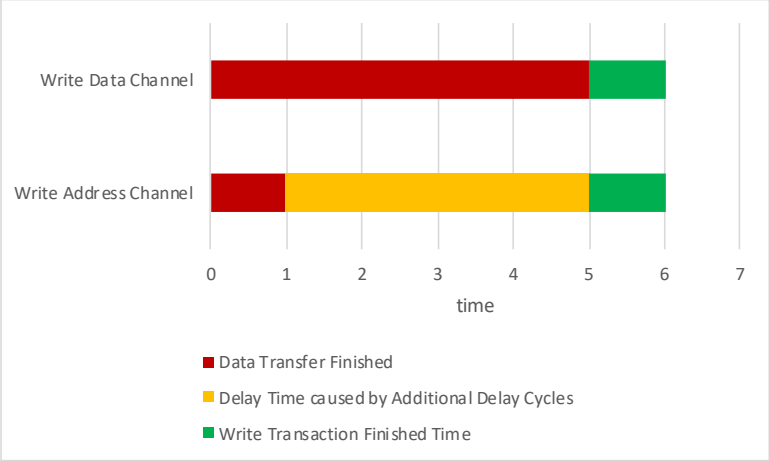


Figure 5.5: Wasted delay cycles of AW. Delay cycles have no influence because the write address channel has to wait for the write data channel

earlier, and the completion of data preparation follows a few cycles later, some extra delay cycles will make no sense. And in an extreme situation, as seen in 5.5, assume that the master interface would like to start a write transaction, and the additional delay cycles on the write address channel are four cycles. In addition, assume that the master interface always finishes its data transfer on write data channel four cycles after its address transfer on write address channel. When the write address channel can originally finish its transfer after the first cycle, then the additional delay cycles will be added to it so that it actually finishes the transfer after the fifth cycle. But as the write data channel can only finish its transfer after fifth cycle, so the additional delay cycles will have no influence to the bus. Thus, this option is not good enough.

The second option deliver an even worse emulation result. It results in great changes in latency and bandwidth even only a small amount of delay cycles are added. It is due to the delayed write access. By write transaction, data is grouped into an amount of transfers, each transfer consists of 64 bits data, conforming to the width of data lane. To trigger one transfer, the master interface and slave interface have to make an agreement through WValid/WReady each time. Therefore, assume that the master interface has 4KB data to be transfered (which is also the maximum transfer size of a write transaction burst), this data will be transfered in 64 transfers. For each 64 bits data transfer the same delay cycles will be injected, which results in great impact on the system even though the delay module is configured only with a few cycles delay. Figure 5.6 and 5.7 show the bandwidth and latency benchmark result between design without delay module, design with delay modules of 200 delay cycles on AR and B, which will be presented later, and design with delay modules of five delay cycles on AR, AW and W. As seen in these figures, a delay module with only five delay cycles already leads to very small bandwidth and high memory access latency. Hence this design is coarse-granular and not chosen for further implementation.

The third option is a feasible solution. Since a write transasction can only be completed after the master interface has received a write reponse coming from B, and this channel also obeys the two-way handshake mechanism, two delay modules are inserted in AR and B separatedly to delay read and write transaction, as shown in Figure 5.4. In this way, if the delay module adds N delay cycles to B, the master interface must wait for exact N additional cycles until it receives response from slave interface in a write transaction.

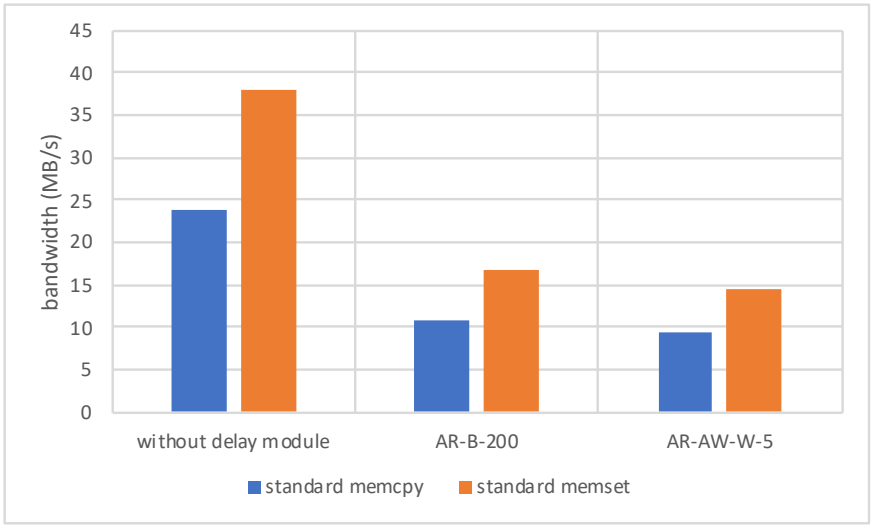


Figure 5.6: Bandwidth comparison of three groups of delayed control signals: design without delay module, design with 200 delay cycles on AR and B, design with 5 delay cycles on AR and AW and W(benchmarked with Tinymembench). A delay module delaying AR, AW and W with a small amount of delay cycles causes much more bandwidth decrease than a delay module delaying AR and B with a great amount of delay cycles

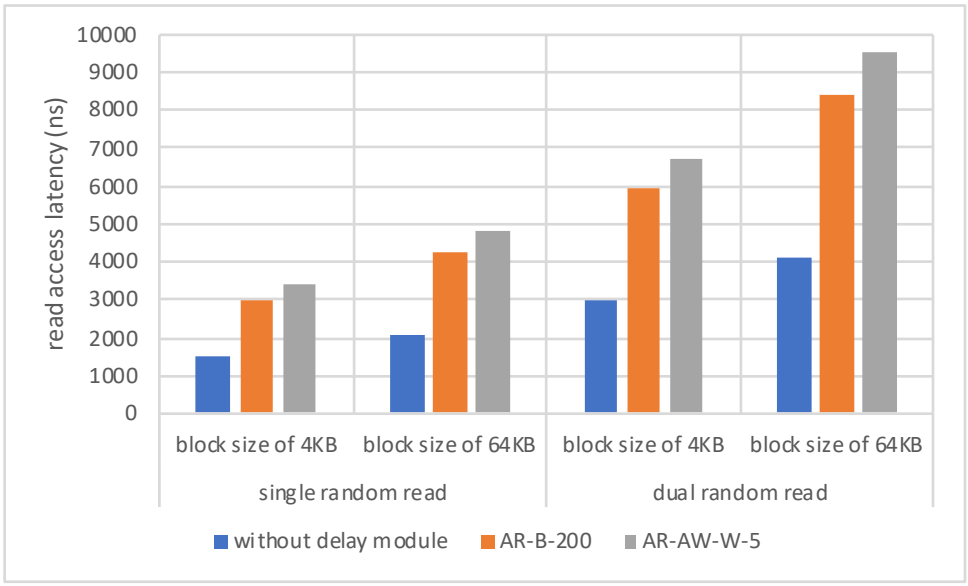


Figure 5.7: Read latency comparison of three groups of delayed control signals: design without delay module, design with 200 delay cycles on AR and B, design with 5 delay cycles on AR and AW and W(benchmarked with Tinymembench). A delay module delaying AR, AW and W with a small amount of delay cycles causes more additional read latency than a delay module delaying AR and B with a great amount of delay cycles

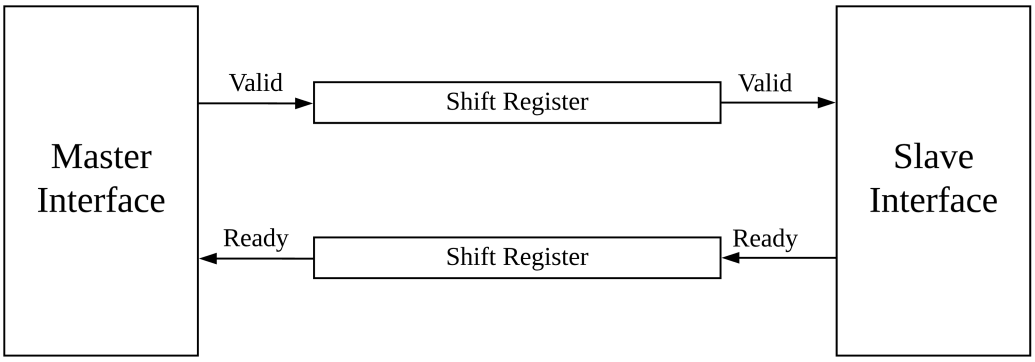


Figure 5.8: Independent shift registers on Valid/Ready signal lanes

Therefore, this option is selected to be implemented.

5.3 Implementation of Delay Module in details

Shift register is employed as the central component of the delay module to delay specific signals of the bus. Shift register is actually a chain of flip flops that are synchronous with the same clock, each flip flop takes the output of previous flip flop as input, and propagating its value to the next flip flop in each clock cycle. With such a characteristic the number of flip flops of a shift register can be parameterizable in source code and adjusted accordingly to the user's demand. The implementation of the delay module with shift register is presented below.

The delay module has two inputs and two outputs. Valid and Ready signals flow into the inputs, get processed by the delay module, and then come out from the delay module and reach the other side. When implementing such a delay module, attention for coordination between the xValid and the xReady signals is needed, as simple implementation shown in Figure 5.8 can not achieve additional delay cycles to the system and can cause the system down. As mentioned in chapter 2, each AXI channel implement the two-way handshake mechanism. The xValid signal of a channel will not change after it is asserted, while the xReady can still be deasserted before both side are aware of the assertion of xValid signal. After xValid signal is asserted, xReady signal can not be changed anymore if it is asserted, and then transaction begins.

Therefore, in the implementation of the delay module, the xValid signal is delayed by the shift register, while xReady signal remains undelayed but blocked, as shown in Figure 5.9. The barrier is implemented with an AND gate. When the xValid signal has gone through all flip flops and has reached the output of shift register, it will not only go directly to the slave interface but also release the barrier that blocks the xReady lane. That means, when xValid signal has reached the other side, the xReady signal can also be detected by the other side if it is asserted. And when the xValid is deasserted, the barrier will be activated again until next removal by the xValid signal.

In addition, besides the system reset that is attached to the shift register, the xValid signal together with a NOT gate will be added as another reset for the shift register, as seen in Figure 5.10. When the xValid signal is active, flip flops of the shift register can propagate the signal one cycle after another. And when the signal is deasserted, the new reset will be active and all the flip flops are reset to their default value immediately, producing a logical low to the slave interface. If a transaction is completed, the deassertion

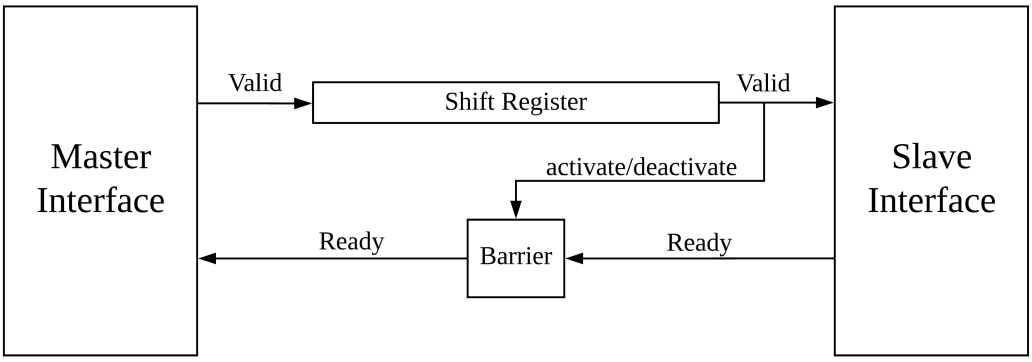


Figure 5.9: Delay module implementation

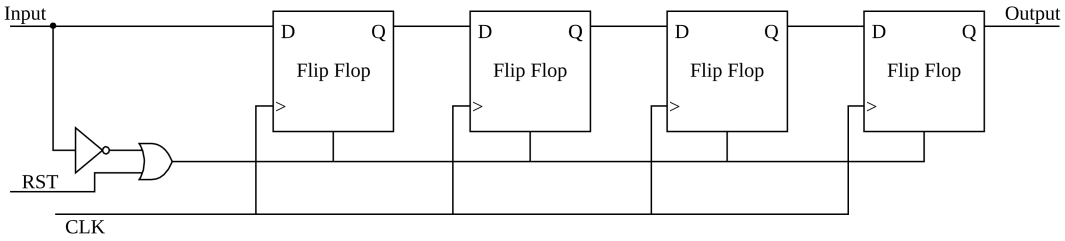


Figure 5.10: Shift register with input signal as one of its reset

of both xValid and xReady signals can be triggered immediately. The simulation waveform of such a shift register with 24 delay cycles is shown in Figure 5.11. In this simulation, the clock period was 5ns. The Valid signal occured on 50ns and was reset on 200ns, while the Ready signal was set to 1 for the whole time. The Ready signal can not reach its output until the Valid signal, which was delayed for 120ns, has reached its output on 170ns. When the Valid signal was reset to 0 on 200ns, its corresponding output and the output of Ready were also reset to 0 immediately, although the Ready signal still remained set.

Moreover, the shift register of the delay module is coded with an adjustable parameter so that the delay cycles can be adjust. It helps the users to tune this emulation board conveniently. And as different channels are delayed by different delay modules, different read and write latency can be achieved independently.

As this implementation uses a shift register along with a barrier, it helps to coordinate the xValid and xReady signal. It also saves the logic cells of another shift register to delay the xReady signal. In addition, using the xValid signal itself as reset signal helps to make the shift register to react immediately to the deassertion of the xValid signal. Without such a design, the deassertion of xValid signal will only be aware by the other side after some delay cycles which equals the amount of flip flops. The xValid signal from previous transaction would be regarded as from current transaction and xReady sigal would be

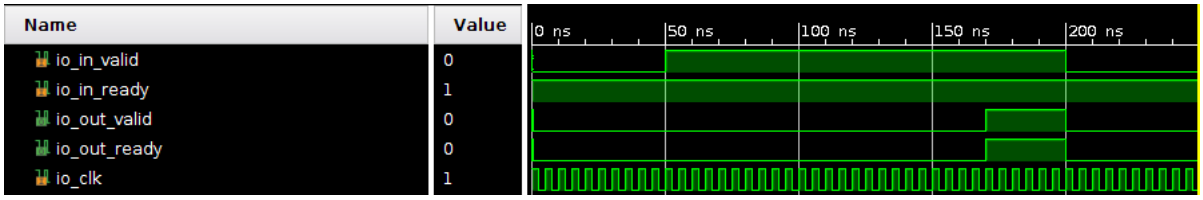


Figure 5.11: Simulation waveform of the shift register

transfered freely in this time interval, which lead to read/write errors.

Chapter 6

Result

6.1 Benchmark Suite

MiBench[22], Tinymembench[23] and a self-written write latency benchmark are the three benchmarks for evaluating the impact of delay module with different delay cycles. In this thesis, they are packaged together, along with a shell script, so that benchmarking-convenience is ensured. This benchmark-suite can be transmitted to the emulation board via SCP program. Instead of running every programs in the benchmark-suite manually, the shell script can run all the programs one after another and write their results to a well-organized directory so that the user can obtain the results by only copying this directory to the local host.

6.1.1 MiBench

MiBench is an open-source embedded benchmark developed in 2001 and is available from this website[24]. MiBench consists of six categories: Automotive and Industrial Control, Network, Security, Consumer Devices, Office Automation, and Telecommunications. For each benchmark application, small and large data sets are provided for benchmarking. The small data set represents light-weight application and the large data set represents stressful, real-world application. For benchmarking the emulation board, 5 of 25 applications (Figure 6.1) are chosen which can be built successfully and run on the board. They are namely basicmath, bitcount, FFT, Adaptive Differential Pulse Code Modulation (ADPCM) and stringsearch. Basicmath performs basic mathmatical calculation, bitcount tests the bit manipulation abilities of a processor, FFT performs a Fast Fourier Transform and its inverse transform on an array of data, ADPCM is a variation of the well-known standard Pulse Code Modulation (PCM) and stringsearch searches for given words in phrases using a case insensitive comparison algorithm. The real run time of these applications will be recorded to evaluate the system performance.

6.1.2 Tinymembench

Tinymembench, developed in 2018, is another open-source benchmark for benchmarking memory bandwidth and latency. It performs the memset and memcpy function of C library to test the memory bandwidth and random read access on a block with specified size to test the memory read latency. For a larger block, more LLC misses will be issued by read

Table 6.1: Build state of MiBench's applications

Build and Run State	Application Name
Build Success	adpcma, basicmath, bitcount, FFT, stringsearch
Failed, segmentation fault (run state)	qsort
Failed, read input file error (run state)	rijndael, susan, tiff2bw, tiff2rgba, tiffdither, tiffmedian
Build Failed, can not be built for RISCV architecture	blowfish, CRC32, ghostscript, gsm, ispell, jpeg, lame, mad, pgp, rsynth, sha, sphinx, typeset

Table 6.2: Lookup-Table usage of each build

Delay Cycles	Total LUT Usage	Logic LUT Usage
0	149490	142326
25	149556	142392
50	149637	142473
100	149670	142506
150	149764	142600
200	149777	142613
300	150090	142926
500	150225	143061

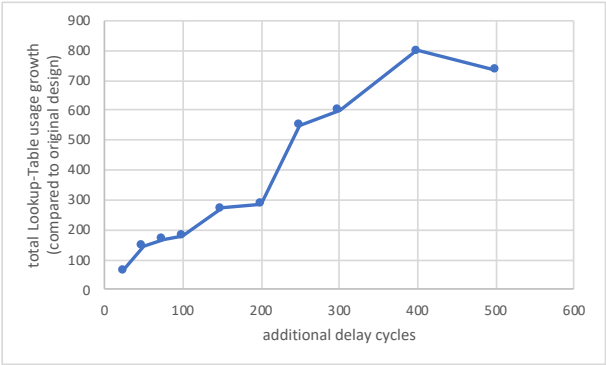
access and thus main memory will be accessed more often. Read access consists of two types, namely single random read and dual random read. Dual random read means that two independent memory accesses with large gap are performed simultaneously on the same block of data while single random read means only one memory access is performed at a time.

6.1.3 Self-Written Write Latency Benchmark

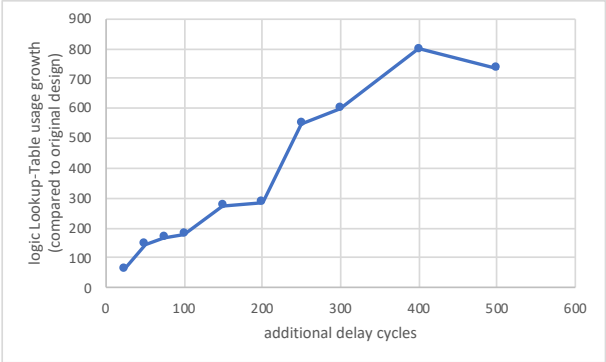
Because the above two benchmarks do not provide tests for write latency, in this thesis, a write latency benchmark utilizing the memset function to test the emulation board with a 4MB block size is written.

6.2 Resource Utilization

In the following Table 6.2, the Lookup-Table usage of each build is presented. The Lookup-Table usage growth compared to original design is shown in Figure 6.1.



(a) Total Lookup-Table growth



(b) Logic Lookup-Table growth

Figure 6.1: Lookup-Table usage growth compared to original design

6.3 Result Analyse

While benchmarking, the same amount of delay cycles is both inserted into AR and B of the AXI bus. In the following subsections, the impact of the additional delay cycles is presented. The result of bandwidth and read access latency impact is measured by Tinymembench, while MiBench shows the impact on real world applications.

6.3.1 Bandwidth

Tinymembench has two ways to measured the bandwidth of a system, to be specific, it executes the two standard C functions memcpy and memset to benchmark the target. Since by memcpy function both memory read access to a memory address and memory write access to the other memory address are triggered, and by memset function only one memory write access is issued, therefore memset always produces a higher bandwidth than memcpy.

By this design, as seen in Figure 6.2, when additional delay cycles are increasing from 0 to 500, the bandwidth of memory is decreasing from 23.9 MB/s to 5.8 MB/s tested by memcpy, and from 38.1 MB/s to 8.9 MB/s tested by memset. As shown by Figure 6.3, as total delay cycles are increasing, although the memory bandwidth is decreasing, the bandwidth loss caused by each delay cycle is also decreasing.

Especially by delay modules with not greater than 100 delay cycles, the impact of each additional delay cycle is weakened more significantly than that by delay modules with more than 100 delay cycles. For example, by memset, bandwidth loss per delay cycle is

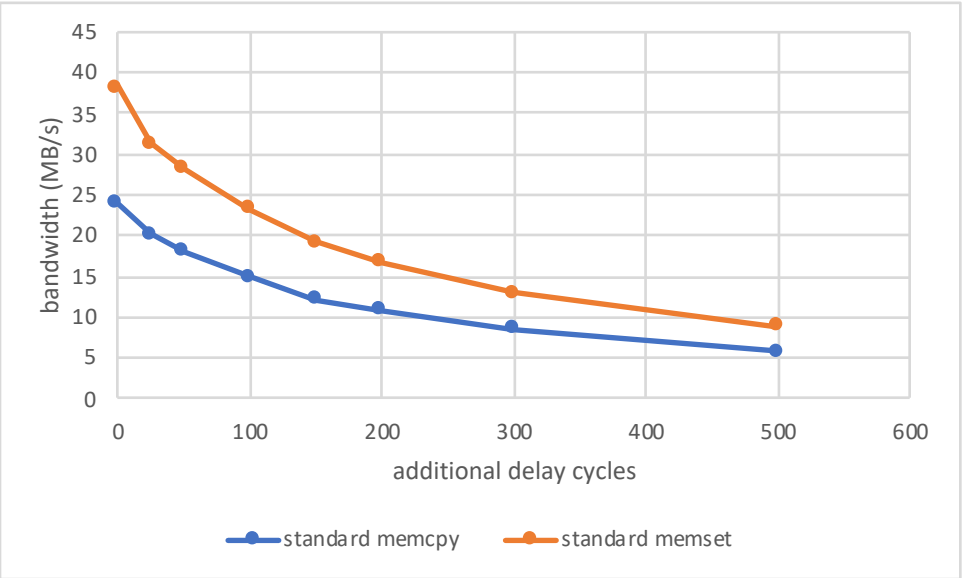


Figure 6.2: Memory bandwidth benchmarked with function memcpy and memset

282.624 KB/s by delay modules with 25 delay cycles and 151.552 KB/s by delay modules with 100 delay cycles, resulting in a 131.072 KB/s reduction, while the reduction between that of 100 and 200 delay cycles is only 42.496 KB/s. The impact of delay cycles on bandwidth becomes gradually weaker.

6.3.2 Read Access Latency

The same as memory bandwidth, random read access latency is also increasing as the delay cycles are rising. As delay cycles are increasing from 0 to 500, latency of single random read access with a block of 4KB is rising from 1515.1ns to 5412.3ns, and from 2105.2ns to 7636.2ns when the block size is 64KB. For dual random read access, latency of read access with a block size of 4KB is increasing from 2977.4ns to 10736.4ns, and from 4144.3ns to 15185.3ns for a 64KB block.

But as seen in Figure 6.5, the latency impact of each delay cycle on read access with the same block size is rather stable compared to impact of a single delay cycle on bandwidth. For single random read access, every additional delay cycle leads to around 7.670ns and 10.913ns delay time with the small and large block respectively. For dual random read access, these values are about 15.269ns and 21.764ns. Therefore, the additional latency caused by each delay cycle is on account of block size and read access mode (single or dual read access).

Because the dual read latency is always about two times of the single read latency with the same block size, according to the specification of Tinymembench, it can be concluded that the memory subsystem can not handle two memory accesses simultaneously. Besides, since the emulation board run at frequency of 50MHz, every additional delay cycle should cause extra latency of 20ns for each memory access. The error between target latency and measured latency can be seen in the following Table 6.3 and 6.4. As seen in these two tables, larger block can cause less error because it can trigger more cache misses and thus more memory accesses are issued.

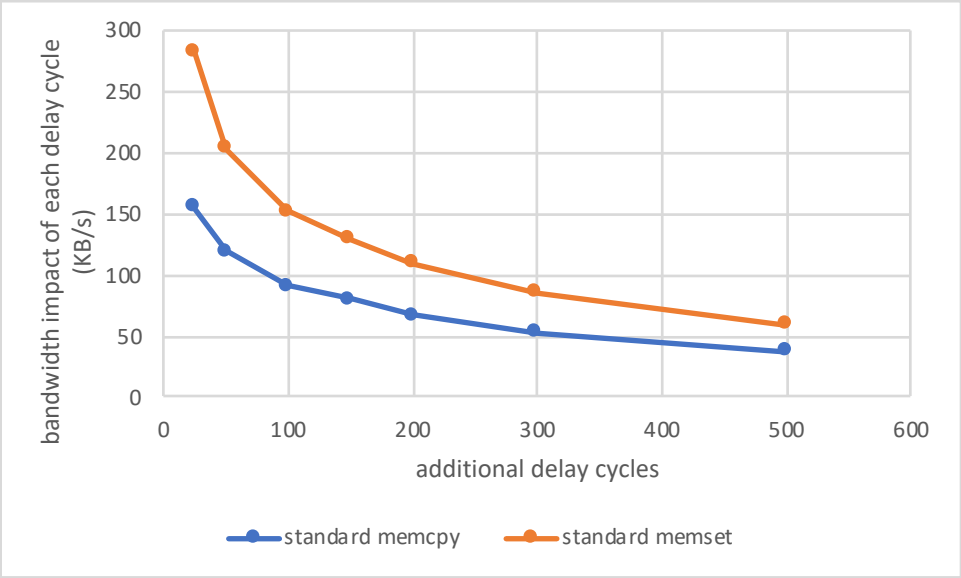


Figure 6.3: Memory bandwidth impact of each delay cycle with delay modules of different additional delay cycles

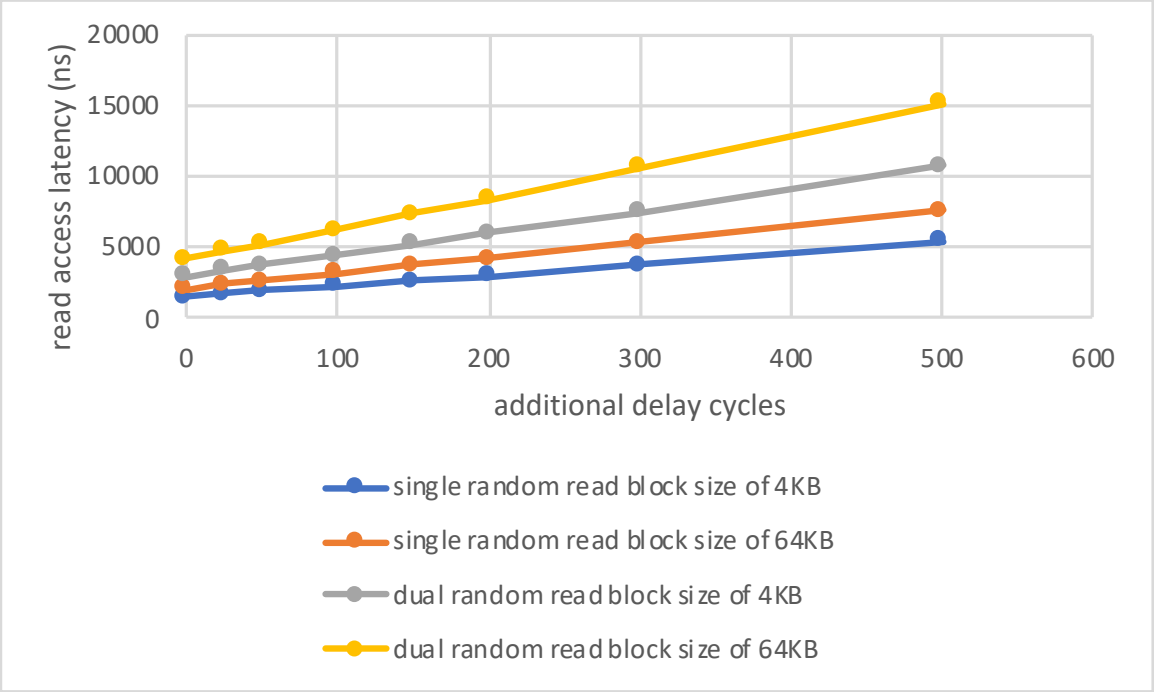


Figure 6.4: Memory read access latency with different block size and delay modules of different additional delay cycles

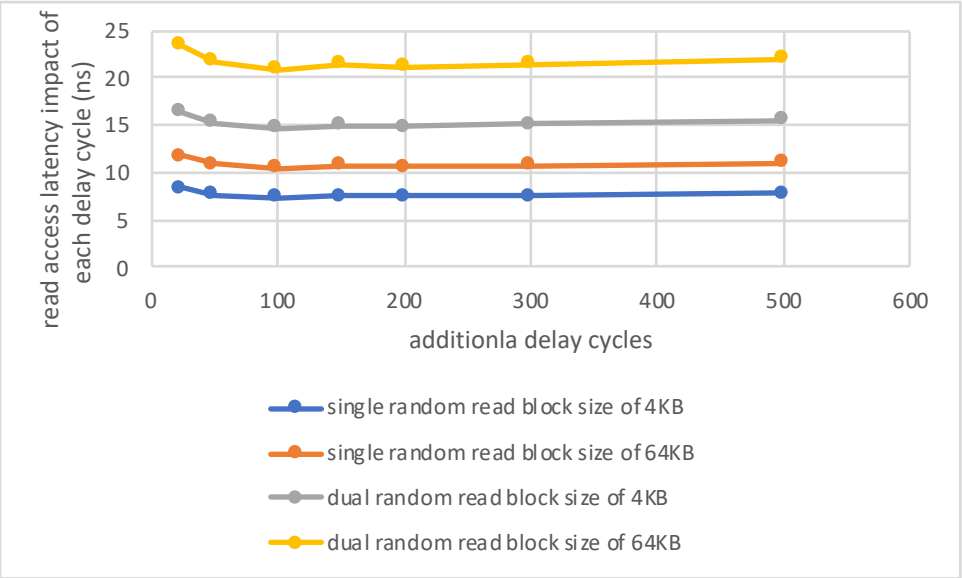


Figure 6.5: Memory read access latency impact of each additional delay cycle with different block size and delay modules of different additional delay cycles

Table 6.3: Error between target latency and measured latency with block size of 4KB of single read access

Delay Cycles	Target Latency (ns)	Measured Latency (ns)	Error (ns)
25	500	207.1	-292.9
50	1000	383.4	-616.6
100	2000	741.3	-1258.7
150	3000	1131.2	-1868.8
200	4000	1490.3	-2509.7
300	6000	2270.5	-3729.5
500	10000	3897.2	-6102.8

Table 6.4: Error between of target latency and measured latency with block size of 64KB of single read access

Delay Cycles	Target Latency (ns)	Measured Latency (ns)	Error (ns)
25	500	294.2	-205.8
50	1000	546.7	-453.3
100	2000	1051.3	-948.7
150	3000	1608.9	-1391.1
200	4000	2128.7	-1871.3
300	6000	3236.4	-2763.6
500	10000	5531	-4469

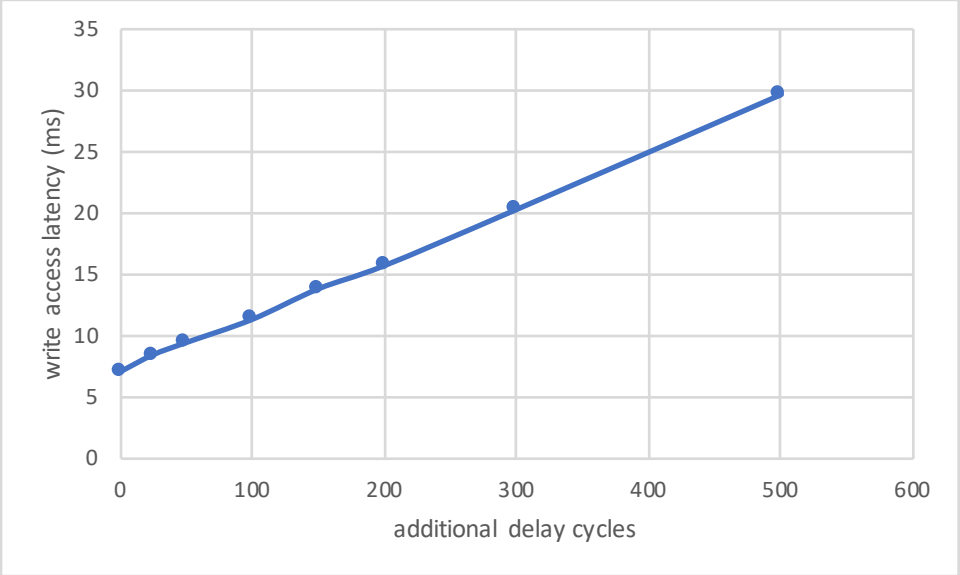


Figure 6.6: Memory write access latency with 4 MB block size and delay modules of different additional delay cycles

6.3.3 Write Access Latency

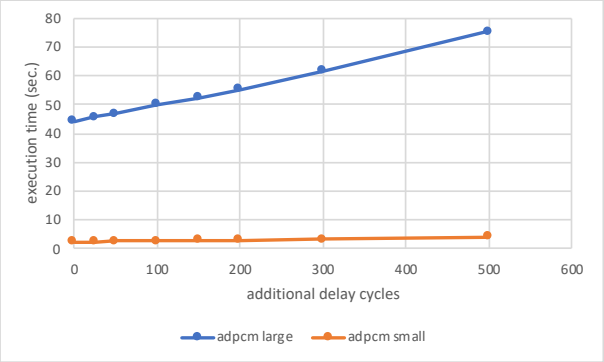
As for write access latency, the benchmark result can be seen in the following Figure 6.6. As additional latency is increasing from 0 to 500 cycles, the write access latency is also increasing almost linearly from 6.9 to 29.61 millisecond.

6.3.4 Real World Applications

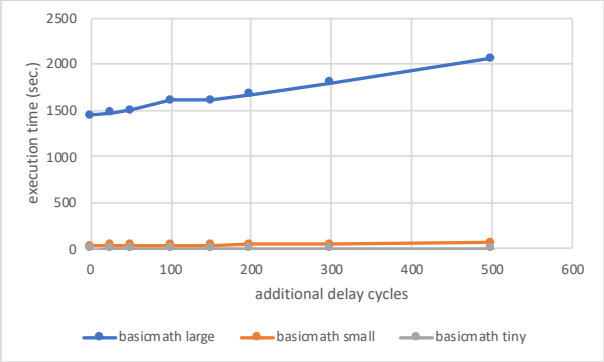
Benchmark results of MiBench applications are shown in the following Figures 6.7a, 6.7b, 6.7c, 6.7d and 6.7e respectively.

As shown in these graphs, the additional delay cycles on memory traffic have slowed down the applications, even though they do not perform memory-consuming operations. In addition, applications running large data set will be affected more significantly by the increasing delay cycles than running small data set, because they will perform more memory read/write accesses. Especially, the execution time of basicmath with large data set has increased from 1449.11 seconds by zero delay cycles to 2066.82 seconds by 500 delay cycles on both AR and B.

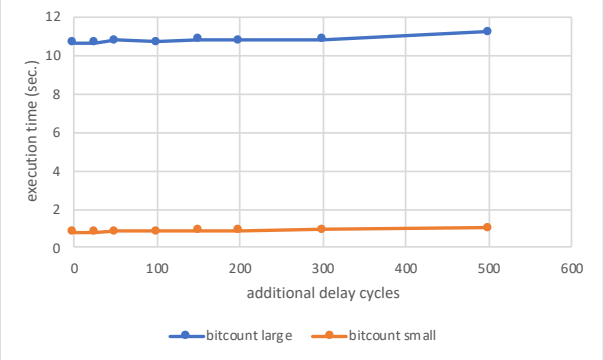
Figure 6.7: Benchmark result of MiBench



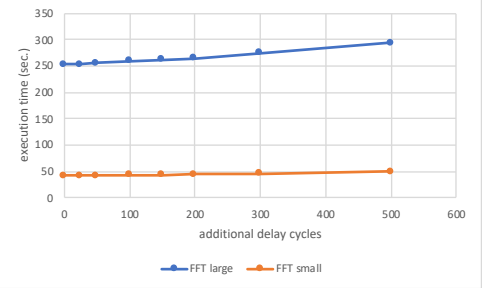
(a) Adpcm benchmark result



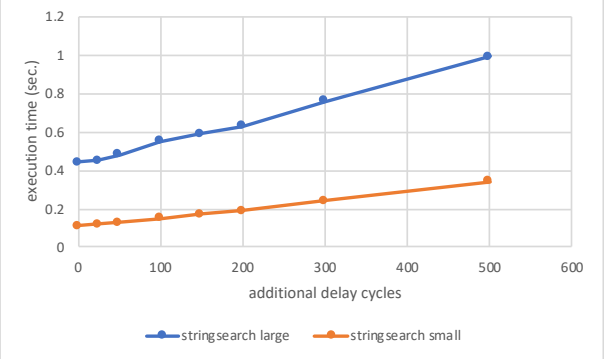
(b) Basicmath benchmark result



(c) Bitcount benchmark result



(d) FFT benchmark result



(e) Stringsearch benchmark result

Chapter 7

Conclusion and Future Work

To emulate NVM, an emulation board based on FU500, which implements the open-source ISA RISCV, is built. Delay module utilizing parameterizable shift register is inserted into the system to inject delay cycles to memory access, so that the slower NVM can be emulate upon faster DRAM. In addition, compared to the existing NVM emulators, this emulation board has more flexibility for further development because it is based on the open-source RISCV ISA. A benchmark suite comprising open-source benchmarks MiBench and Tinymembench is also packaged and used to benchmark the emulation board. It is shown that this emulation board can emulate computer system with main memory of different bandwidth and latency.

Future work about this emulation board is listed here: First, except for slower access speed compared to DRAM, non-volatility is also an important characteristic of NVM, which is not implemented in this thesis. Second, NVM would be used along with DRAM in the future memory hierarchy[25], thus an emulation board with cooperating separated DRAM and emulated NVM will be closer to real-world usage. Third, an emulation board with runtime delay cycles adjustment will be helpful to speed up the emulation process.

Bibliography

[1] *International Technology Roadmap for Semiconductors (ITRS)*. URL: http://www.itrs2.net (Retrieved 12/04/2019).

[2] Sung-Kye Park. "Technology Scaling Challenge and Future Prospects of DRAM and NAND Flash Memory." In: *2014 25nd IEEE International Symposium on Rapid System Prototyping* (2015). DOI: 10.1109/IMW.2015.7150307. URL: https://ieeexplore.ieee.org/document/7150307.

[3] Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger. "Phase-Change Technology and the Future of Main Memory." In: *IEEE Micro* (2010). DOI: 10.1109/MM.2010.24. URL: https://ieeexplore.ieee.org/document/5430747.

[4] Emre Kültürsay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu. "Evaluating STT-RAM as an energy-efficient main memory alternative." In: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2013). DOI: 10.1109/ISPASS.2013.6557176. URL: https://ieeexplore.ieee.org/document/6557176.

[5] Hoeju Chung, Byung Hoon Jeong, and et al. "A 58nm 1.8V 1Gb PRAM with 6.4MB/s program BW." In: *2011 IEEE International Solid-State Circuits Conference* (2011). DOI: 10.1109/ISSCC.2011.5746415. URL: https://ieeexplore.ieee.org/document/5746415.

[6] T. Eshita, W. Wang, and et al. "Development of ferroelectric RAM (FRAM) for mass production." In: *2014 Joint IEEE International Symposium on the Applications of Ferroelectric, International Workshop on Acoustic Transduction Materials and Devices & Workshop on Piezoresponse Force Microscopy* (2014). DOI: 10.1109/ISAF.2014.6922970. URL: https://ieeexplore.ieee.org/document/6922970.

[7] S. B. Tekin, S. Kalem, Z. E. Kaya, and E. Jalaguier. "Electrical characterization of HfO2 based resistive RAM devices having different bottom electrode metallizations." In: *2018 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon (EUROSOI-ULIS)* (2018). DOI: 10.1109/ULIS.2018.8354734. URL: https://ieeexplore.ieee.org/document/8354734.

[8] Taemin Lee and Sungjoo Yoo. "An FPGA-based Platform for Non Volatile Memory Emulation." In: *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)* (2017). DOI: 10.1109/NVMSA.2017.8064466. URL: https://ieeexplore.ieee.org/document/8064466.

[9] Haris Volos1, Guilherme Magalhaes2, Ludmila Cherkasova1, and Jun Li. "Quartz: A Lightweight Performance Emulator for Persistent Memory Software." In: *the 16th Annual Middleware Conference* (2015). DOI: 10.1145/2814576.2814806. URL: https://www.researchgate.net/publication/301453562_Quartz_A_Lightweight_Performance_Emulator_for_Persistent_Memory_Software.

[10] Atsushi Koshiba, Takahiro Hirofuchi, Soramichi Akiyama, Ryousei Takano, and Mitaro Namiki. "Towards write-back aware software emulator for non-volatile memory." In: *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)* (2017). DOI: 10.1109/NVMSA.2017.8064479. URL: https://ieeexplore.ieee.org/document/8064479.

[11] Taemin Lee, Dongki Kim, Hyunsun Park, Sungjoo Yoo, and Sunggu Lee. "FPGA-based prototyping systems for emerging memory technologies." In: *2014 25nd IEEE International Symposium on Rapid System Prototyping* (2014). DOI: 10.1109/RSP.2014.6966901. URL: https://ieeexplore.ieee.org/document/6966901.

[12] Yu Omori and Keiji Kimura. "Performance Evaluation on NVMM Emulator Employing Fine-Grain Delay Injection." In: *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)* (2019). DOI: 10.1109/NVMSA.2019.8863522. URL: https://ieeexplore.ieee.org/document/8863522.

[13] *U54-MC Architecture*. URL: https://content.riscv.org/wp-content/uploads/2017/12/Wed-1154-TileLink-Wesley-Terpstra.pdf (Retrieved 12/04/2019).

[14] *SiFive U54-MC Manual v19.08p0*. URL: https://sifive.cdn.prismic.io/sifive%5C%2Fdc4980ff-17db-448b-b521-4c7ab26b7488_sifive+u54-mc+manual+v19.08.pdf (Retrieved 12/04/2019).

[15] *TileLink*. URL: https://bar.eecs.berkeley.edu/projects/tilelink.html (Retrieved 12/04/2019).

[16] *VC707 Emulation Board overview*. URL: https://www.xilinx.com/products/boards-and-kits/ek-v7-vc707-g.html#hardware (Retrieved 12/05/2019).

[17] *AMBA® AXITM and ACETM Protocol Specification*. URL: http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf (Retrieved 12/04/2019).

[18] *AXI to AXI Asynchronous Bridge Cycle Model*. URL: http://infocenter.arm.com/help/topic/com.arm.doc.dui1058a/cycle_models_AMBA_AXI_AXI_ASync_Bridge_User_DUI1058A_en.pdf (Retrieved 12/04/2019).

[19] *lowRISC official site*. URL: https://www.lowrisc.org (Retrieved 12/04/2019).

[20] *PULP official site*. URL: https://www.pulp-platform.org (Retrieved 12/04/2019).

[21] *SiFive official site*. URL: https://www.sifive.com (Retrieved 12/04/2019).

[22] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin anbd T. Mudge, and R.B. Brown. "MiBench: A free, commercially representative embedded benchmark suite." In: *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)* (2001). DOI: 10.1109/WWC.2001.990739. URL: https://ieeexplore.ieee.org/document/990739.

[23] *Tinymembench github site*. URL: https://github.com/ssvb/tinymembench (Retrieved 01/26/2020).

[24] *MiBench*. URL: http://vhosts.eecs.umich.edu/mibench// (Retrieved 12/04/2019).

[25] Shuichi Oikawa and Satoshi Miki. "Future Non-volatile Memory Storage Architecture and File System Interface." In: *2013 First International Symposium on Computing and Networking* (2013). DOI: 10.1109/CANDAR.2013.69. URL: https://ieeexplore.ieee.org/document/6726931.