STM JOURNALS

Journal of
VLSI Design Tools & Technology

ISSN: 2249-474X (Online)
ISSN: 2321-6492 (Print)
Volume 14, Issue 1, 2024
January–April
DOI (Journal): 10.37591/JoVDTT

http://engineeringjournals.stmjournals.in/index.php/JoVDTT/index

Research JoVDTT

Study and analysis of the Double Data Rate SDRAM Controller for High-speed Interfacing with Processing Device

Kuldeep Jaiswal[1,] *, Sunil Kumar Shah[2]

Abstract

A real-time embedded system must now manage many programs running concurrently. Increased Data Rate Because of its burst access, speed, and pipeline features, synchronous DRAM is a typical memory-building material. DDR transfers are performed using synchronous dynamic access memory. The memory controller must be set with a pipelined design for various applications and systems to perform effectively. The purpose of this study is to design a DRAM controller that will allow for quick data interfacing between the main CPU and main memory. This is accomplished using an innovative Super Harvard parallel data, program, and instruction interface. This is a common digital design approach in which an instruction or data is processed in four phases. Coarse-grain FPGAs are tuned for certain tasks. The use of a vertex coarse-grain FPGA indicates that the design has been tuned for certain vertex processing workloads, which may result in a size reduction. This suggests that your design employs a range of modelling approaches or architectures, maybe for flexibility or optimization in dealing with various processing components. The design and construction of a DDR SDRAM controller optimized for design area consumption. Our major goal is to communicate with processing devices at fast speeds while using as few resources as possible. To improve performance, the suggested controller employs complicated techniques such as pipelining and optimization for a specific processing device interface. The architecture makes full use of the DDR SDRAM interface, making use of double data rate capabilities to boost data transmission speeds. The controller features a specific pipelining structure to enhance resource use and assure effective data processing at all levels. The Integrated Software Environment is used to model the DDR SDRAM controller architecture. Verilog Hardware Description Language (HDL) is a well-known digital design language. Xilinx EDA (Electronic Design Automation) tools like the Integrated Software Environment (ISE) are widely used for FPGA design and verification. When you use ISE for verification, you are verifying the correctness of your design before FPGA implementation via simulation and testing.

Keywords: FPGA, DDR, DRAM, High-Speed Interfacing, Pipelining, Area Optimization, Verilog HDL, Xilinx EDA, Integrated Software Environment (ISE).

***Author for Correspondence**
Kuldeep Jaiswal
E-mail: kourav530@gmail.com

[1]M. Tech. Scholar, Department of Electronics and Communication Engineering, Gyan Ganga College of Technology, Jabalpur, India
[2]Assistant Professor Departments of Electronics and Communication Engineering, Gyan Ganga College of Technology, Jabalpur, India

Received Date: May 16, 2024
Accepted Date: May 23, 2024
Published Date: May 30, 2024

Citation: Kuldeep Jaiswal, Sunil Kumar Shah. Study and analysis of the Double Data Rate SDRAM Controller for High-speed Interfacing with Processing Device. Journal of VLSI Design Tools & Technology. 2024; 14(1): 8–13p.

INTRODUCTION

SDRAM, or Synchronous Dynamic Random Access Memory, has advanced through several phases in recent years, beginning with the initial SDRAM DDR and concluding with the most current SDRAM SDR. SDRAM has been employed in a broad variety of applications due to its great performance and inexpensive cost, including general computers, visual technologies, and embedded devices. Its performance has continually increased in three dimensions: capacity, frequency, and bandwidth. When operating as an SDRAM master, the processor's frequency is

increased. And the number of processing cores, which might number in the thousands, grows with time [1]. The performance of systems equipped with SDRAM processors, on the other hand, frequently degrades quickly. Memory controllers are critical in embedded systems and high-performance computing for ensuring appropriate data flow between processor units and memory modules. To meet the needs of real-time processing, multimedia applications, and complex algorithms, current applications necessitate progressively higher data transfer rates between processing devices and memory. DDR SDRAM has developed as a standard memory technology due to its better data transmission capabilities, making it critical for memory controllers to fully use its potential [2]. The need to address the rising demand for high-speed data transport in memory-intensive applications motivated this notion. Given the resource limits that are often present in embedded systems and specialized processing devices, it is vital to optimize the design area. The principal functionality of current DRAM systems can result in quite significant access latencies [3]. Real-time systems (RTS) that adopt modern conventional designs suffer from significant resource over-allocation to compensate for uncertainties induced by speculative, average case optimizations, which need lengthy study. An important research area is the creation of time deterministic RTS optimized architectures that enable easy timing analysis and precise timing guarantees. A task in a real-time system (RTS) has a time restriction. The delayed procedure may have devastating implications for hard RTS and should be avoided at all costs. To ensure that such systems always operate promptly, they must be thoroughly tested and proven. To meet the needs of current processing equipment, create a DDR SDRAM controller capable of high-speed data transmission rates. Use methods to make the most of the design space while preserving efficiency and resource conservation. Customize the controller for increased compatibility and performance to provide simple integration with a certain processing device. The design and implementation of a DDR SDRAM controller, with a strong emphasis on optimizing the design area Figure 1 depicts the memory for the Real-Time System [4]. To achieve the anticipated high-speed interfacing, the scope includes studying advanced techniques like pipelining and device-specific optimizations.

The success of the RTS (Real-Time Systems) memory controller design is intrinsically linked to a deep understanding of the setup represented in Figure 1 This configuration comprises three essential elements that, when combined, determine the memory controller's efficiency and performance. The memory subsystem is based on DRAM (Dynamic Random Access Memory) technology [5]. A detailed grasp of DRAM complexity is necessary since it specifies the fundamental constraints and characteristics of the memory controller architecture. The unique nature of DRAM, with its refresh cycles and access patterns, raises difficulties that have a direct impact on memory controller performance. The controller must navigate these challenges in order to optimize data transmission speeds and decrease latency.

SDRAM Technology

The term “dynamic RAM” refers to the fact that each bit's value is represented as a charge on a tiny capacitor, which gradually depletes owing to leakage. The images above show an SDRAM array with addressable rows and columns [6]. Each SDRAM device is composed of four banks of two ranks each, and each module is composed of four devices. Each device has its own data bus, which might be 16-bit or 64-bit. This fundamental understanding of SDRAM structure and operation serves as the foundation for further research into the complexity of this memory technology [7]. The sections that follow will go into further detail on the evolution of SDRAM standards, its performance characteristics, and the unique issues and optimizations involved in developing controllers for the Synchronous DRAM interface in real-time systems. DRAM is referred to as “dynamic” due to the nature of its storing method. The charge on a small capacitor indicates the value of each bit. As an electrical charge, this capacitor serves as a data-storing device. Because of leakage, these capacitors deplete over time, necessitating periodic refresh cycles to guarantee data integrity. The SDRAM array is made up of addressable rows and columns that form the fundamental structure for data storage and retrieval. The arrangement is represented in the diagram above, with rows and columns available for efficient addressing. Each SDRAM chip is partitioned into four banks [8]. A module contains two levels. A rank is a grouping of

memory banks that may be accessed independently. This modular architecture allows for more efficient parallel processing and higher data throughput. Each module is composed of four SDRAM devices, each with its own 16-bit or 64-bit data bus. This configuration enhances overall data transfer capability and memory system efficiency. Understanding the complexity of SDRAM's structural components and operational principles is essential for designing effective memory controllers [8]. The dynamic nature of the memory cells, together with the ordered architecture of the SDRAM array, provide a foundation for resolving performance issues and improving the Synchronous DRAM interface with processing devices. This two-step approach ensures that the row's critical sections are ready for reading [9]. The regular refresh of capacitors is crucial for preventing charge leakage and protecting the integrity of data held in dynamic memory cells. The complicated coordination of these activities is important to the proper operation of dynamic random-access memory (DRAM) [10]. DRAM Refresh-Dynamic Memory Cells DRAM stores data as electrical charges on capacitors in memory cells. Because of the inherent characteristics of capacitors, these charges leak away over time. To prevent data loss, DRAM requires periodic refreshing to refill the charge in its memory cells. Restart the operation. During a refresh operation, the memory controller reads and immediately rewrites the data stored in each memory cell. This technique ensures that the capacitors' charge is replenished. The Refresh Cycle The refresh cycle is a basic element of DRAM functioning and is essential for the memory to function correctly [4]. All memory rows are gradually updated. The memory controller cycles across each row, activating and changing the contents of each row. Some SDRAM models, especially DDR SDRAM and later, include a hidden refresh mechanism. This allows the memory controller to do background refresh operations while maintaining regular read and write operations. The auto-refresh feature streamlines the refresh process. In the absence of explicit CPU directions, the memory controller manages refresh operations [11]. For devices and low-power applications, there is a self-refresh mode. The memory enters this mode during periods of inactivity, allowing it to perform refresh operations independently of the memory controller as mentioned in Table 1.

Self-refresh is an independent refresh mode done by the SDRAM chip after lengthy periods of inactivity. This power-saving option must be enabled manually on the chip. In other cases, self-refresh might not be appropriate for continuous communication since returning to regular functioning takes a long time. To initiate the refresh, a distinct refresh command is necessary [12]. Instead of specifying the row's address, the instruction is sent to the row by leveraging an internal chip counter. At the same time, the same row is updated in all banks, and the counter is increased for the next refresh operation. Refresh Distributed is the accepted procedure. The refresh activity is dispersed consistently across time. For example, sending Auto Refresh every 7810 ns ensures that each row fulfils the refresh time requirement of 64 Ms (7810 × 8192 × 63.98 × 106 ns).

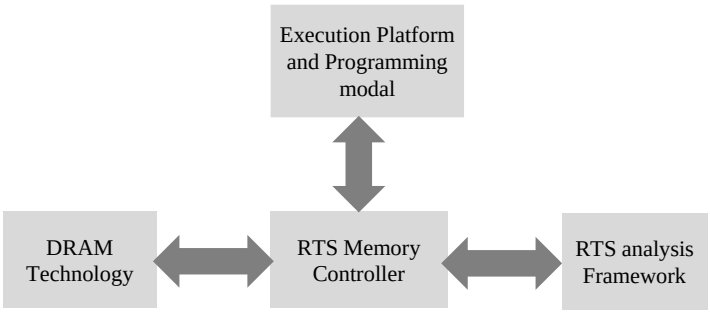

Figure 1. Memory for the Real-Time System.

Table 1. DPRAM Refresh Parameters.

Parameters	Trace	64-Mb	128-Mb	256-Mb	512-Mb	1-Gb	2-Gb	4-Gb	8-Gb
DDR	55–70	70	80	120	130				
DDR2	55–65		75	105	127.5	195	327.5		
DDR3	43.3–52.5				90	110	160	260	350

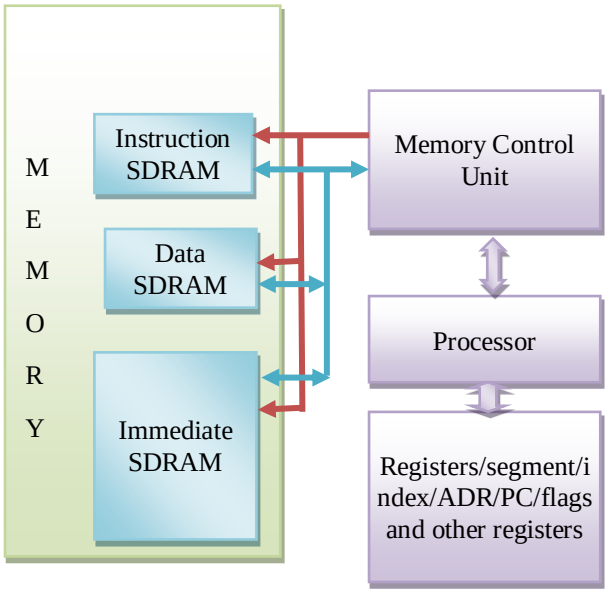

Figure 2. Block Diagram of Proposed Controller Architecture.

Proposed Design

Hybridization of FPGAs in reality, it is an FPGA with a Hybrid Interconnect Structure. In other words, a Hybrid FPGA fabric is a one-of-a-kind combination of logic modules and connections. The logic modules' functionality can be fixed or field-programmable, and the link that joins the modules can also be fixed or field-programmable [2]. This field-programmable wiring is made up of metal wire tracks and field-programmable switches, the most common of which is a transistor switch controlled by a RAM cell. The purpose of this project is to create a DRAM controller that connects the main CPU to the main memory. The use of a revolutionary Super Harvard type of parallel interfacing for data, program data, and instructions, as well as four-step pipelining to offer high throughput and high-speed interfacing, allows rapid data interfacing. The work was created using Vertex Corse grain FPGA, which allowed a smaller footprint [4]. In addition, a modelling architecture combination was applied. The architecture was created using Xilinx EDA and tested using ISE. In terms of speed and area, the results are excellent. Because various FPGAs are available, we may do hands-on research in the field of ASIC design. Furthermore, if our application is specialized, a hard-core ASIC will outperform a software-based library routine in terms of throughput. We have nothing to lose if our application is mentioned; for increased performance, FPGA-based Intellectual Property is the best solution. The steps of the ISE design flow are as follows: design entry, design synthesis, design execution, and Xilinx device programming. The instruction SDRAM holds the OPCODE of the instructions, whereas the data SDRAM saves temporary data generated during program execution. If necessary, direct data from the application can be saved in the immediate SDRAM memory.

This design provides a segregated memory for immediate data in Super Harvard architecture with three levels of SDRAM memory. Using the suggested Controller architecture, it was able to execute the CISC instruction set with the RISC feature, that is, each instruction in one clock cycle [3]. It increases pipeline efficiency Figure 2 displays the Proposed Processor Module's Developed Architecture. It was possible due to the professional design of the Decoder and the implementation module of the required work. Because the suggested concept fetch module does not need to wait for the decoder module, it continues to obtain the OPCODE at each clock without regard for the decoder signal; the decoder signal is only required by the execution unit [4].

RESULTS

There are two testing methodologies in use. VHDL test bench with FPGA controller simulation scripts. The controller's behavior was isolated using the test bench (TB) for RTL-level simulation. The

method has the advantage of authenticating the controller at the processor's interface, but it adds complexity to the TB. A TB built from the ground up that works at the controller's interface may be a better option. To perform writes and reads, the TB employs unique addresses, ensuring that the data retrieved corresponds to the original data placed at that location. The non-matching entries are reported. Other flags impact the verbosity of the controller and processor interfaces. Use the reporting tool in batch mode to quickly verify accuracy without having to scrutinize signal waveforms. Because the controller contains so many choices, the TB does not try to test them all. Configuration variables, on the other hand, govern which configuration is tested. To avoid the refresh logic interfering with routine transactions, the refresh period was lowered. The SDRAM simulation model ensures that the controller adheres to the SDRAM time constraints. The concept's first use resulted in major simulation mistakes. The problem was the model's internal clock gating, which generated a delta cycle discrepancy between the controller's clock and the SDRAM model's internal clock. The problem was fixed by introducing a very brief propagation delay to signals on the SDRAM interface. As a result, the data sampled by both clocks corresponded to the same logical clock cycle, and the behavior was similar to that of the synthesized version. This section digs into the outcomes of controller synthesis. The first section compares our controller's architecture to that of existing general-purpose SDR SDRAM controllers. We will compare the FPGA synthesis performance of our controller to three rivals. We begin by briefly introducing each design before presenting the findings after the section. Because the entire observable of the controller's state was available during the simulation, the TB was also useful for pinpointing the source of problems. Patmos' link to the TU memory controller provided the initial TB source. The identical synthesis tool configuration was utilized, but we only looked at Xilinx synthesis results for the Vertex-4 target because it is the FPGA on which the controller is built. The Xilinx reference design was an exception since it featured Xilinx-specific primitives such as LUT shift registers. The Vertex4FPGA was chosen for comparison since its architecture is close to the one for which the design was created. We replicated our design with the same purpose in mind to aid in comparison. The regular tool settings were utilized, with the exception that the register packing into IOB was disregarded for Xilinx synthesis to maintain the flip-flops in slices.

Table 2 shows a comparison of the synthesis outcomes for the SDR controllers that were tested.

Table 2. Synthesis outcomes for SDR controllers.

Design	Fax (MHz)	Slices	LUT
Xilinx	251 Mhz	97	72
[3]	202 Mhz	101	
[2]		104	
[1]	349 Mhz		127
Proposed	374 Mhz	93	74

CONCLUSIONS AND FUTURE WORK

Conclusion

The SDR SDRAM controller is now available as open source. It was first developed and tested on two RTS systems. There has been research on memory access scheduling solutions for the FPGA platform**.** For hard-RTS, round robin has no advantage over time division. TDM, on the other hand, can reduce WCET. During the research, critical considerations and successes were discovered, leading to several remarkable outcomes. Because the requester with the lowest priority has a delay proportionate to the total bandwidth available to successive requesters, static priority arbiters such as CCSP and PBS are not scalable for WCET analysis. Memory access time analysis at the WCET level is limited by fundamental constraints in terms of reducing memory bandwidth over-allocation. The creation of an area-optimized DDR SDRAM controller for high-speed interaction with a processor device is a major undertaking with the potential to greatly affect system performance. As a consequence of the investigation, estimates of their RTS efficiency and hardware implementation feasibility were created.

Future Work

Study programming methods for inter-core communication and on-chip memory. Explore ways to improve external memory access arbitration for varied programming styles. Develop programming model-specific changes to increase external memory consumption. Examine task memory demand modelling's accuracy and applicability. Improve scheduling-level memory access patterns to boost system performance. Future research on optimizing a DDR SDRAM controller for a high-speed processor interface is intriguing and profitable. Develop techniques to enhance memory access and bandwidth consumption. Explore new scheduling and command optimization strategies to boost data transmission speeds. To reduce memory access latency, use techniques such as pipelining and prefetching. Implement read and write latencies-reducing solutions to boost system performance. To optimize energy use, create DDR SDRAM controller power-saving strategies. Check power-saving settings and dynamic frequency scaling based on processor unit workload.

Training and Adaptive Timing: Utilize adaptive timing technologies for dynamic timing adjustments. Memory system operating conditions and variations determine adaptive timing and training.

REFERENCES

1. S. Wijeratne, S. Pattnaik, Z. Chen, R. Kannan and V. Prasanna, “Programmable FPGA-based Memory Controller,” 2021 IEEE Symposium on High-Performance Interconnects (HOTI), 2021, pp. 43-51, doi: 10.1109/HOTI52880.2021.00020.
2. Sateesh Kourav, Sunil Shah “Design and Implementation of 64-Bit Arithmetic Logic Unit on FPGA Using VHDL” International Journal of Algorithms Design and Analysis Vol. 6: Issue 2
3. H. Jain, M. Edwards, E. R. Elenberg, A. S. Rawat and S. Vishwanath, “Achieving Multi-port Memory Performance on Single-Port Memory with Coding Techniques,” 2020 3rd International Conference on Information and Computer Technologies (ICICT), 2020, pp. 366-375, doi: 10.1109/ICICT50521.2020.00065.
4. Sateesh Kourav1, Sunil Shah2 “Design and Analysis of Low Power, High Speed 64-Bit Alu Using Efficient Technique” International Journal of Research Publication and Reviews, Vol 3, no 4, pp 1197-1200, April 2022.
5. G. Harling, “A DRAM compiler for fully optimized memory instances,” Proceedings 2001 IEEE International Workshop on Memory Technology, Design and Testing, 2001, pp. 3-8, doi: 10.1109/MTDT.2001.945221.
6. Sateesh Kourav, Sunil Shah “Analysis of High Performance 6-Stage 64-Bit MIPS RISC Pipelined Processor Using FPGA VHDL” Research & Reviews: A Journal of Embedded System & Applications ISSN: 2395-6712 (Online) ISSN: 2321-8533 (Print) Volume 9, Issue 2, 2021 DOI (Journal): 10.37591/JoESA.
7. M. Ma, X. Chen and J. Liu, “Characterize the DRAM with FPGA,” 2020 IEEE 8th International Conference on Computer Science and Network Technology (ICCSNT), 2020, pp. 142-145, doi: 10.1109/ICCSNT50940.2020.9304999.
8. Sateesh Kourav, Sunil “Area and Speed Efficient Floating Point Unit Implementation on Hybrid FPGAs” International Journal of Research Publication and Reviews, Vol 3, no 4, pp 1461-1466, April 2022.
9. Yadav and V. Bendre, “Design and Verification of 16 bit RISC Processor Using Vedic Mathematics,” 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), 2021, pp. 759-764, doi: 10.1109/ESCI50559.2021.9396965.
10. S. M. Bhagat and S. U. Bhandari, “Design and Analysis of 16-bit RISC Processor,” 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-4, doi: 10.1109/ICCUBEA.2018.8697859.
11. G. N. Chiranjeevi and S. Kulkarni, “Pipeline Architecture for N==K 2L Bit Modular ALU: Case Study between Current Generation Computing and Vedic Computing,” 2021 6th International Conference for Convergence in Technology (I2CT), 2021, pp. 1-4, doi: 10.1109/I2CT51068.2021.9417917.

