electronics

Article

Study on the Implementation of a Simple and Effective Memory System for an AI Chip

Taepyeong Kim [1], Sangun Park [2] and Yongbeom Cho [2,*]

[1] SYNOPSYS, Seoul 13494, Korea; pilgrim82@gmail.com
[2] Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, Korea; kos8108@konkuk.ac.kr
* Correspondence: ybcho@konkuk.ac.kr

Abstract: In this study, a simple and effective memory system required for the implementation of an AI chip is proposed. To implement an AI chip, the use of internal or external memory is an essential factor, because the reading and writing of data in memory occurs a lot. Those memory systems that are currently used are large in design size and complex to implement in order to handle a high speed and a wide bandwidth. Therefore, depending on the AI application, there are cases where the circuit size of the memory system is larger than that of the AI core. In this study, SDRAM, which has a lower performance than the currently used memory system but does not have a problem in operating AI, was used and all circuits were implemented digitally for simple and efficient implementation. In particular, a delay controller was designed to reduce the error due to data skew inside the memory bus to ensure stability in reading and writing data. First of all, it verified the memory system based on the You Only Look Once (YOLO) algorithm in FPGA to confirm that the memory system proposed in AI works efficiently. Based on the proven memory system, we implemented a chip using Samsung Electronics' 65 nm process and tested it. As a result, we designed a simple and efficient memory system for AI chip implementation and verified it with hardware.

Keywords: memory controller; memory system; AI; YOLO

Citation: Kim, T.; Park, S.; Cho, Y. Study on the Implementation of a Simple and Effective Memory System for an AI Chip. *Electronics* **2021**, *10*, 1399. https://doi.org/10.3390/electronics10121399

Academic Editor: Costas Psychalinos

Received: 11 April 2021
Accepted: 1 June 2021
Published: 10 June 2021

1. Introduction

Artificial intelligence (AI) technology has been developed for a long time. Initially, it was approached in a mathematical way based on theory, and it was developed and implemented as software [1]. Since the AI technology implemented in software uses the existing CPU, GPU, and memory system, hardware implementation for the AI structure itself was not required [2]. However, as the utilization of AI gradually increased, not only did the implementation of AI through CPU and GPU become necessary, but so did an AI-dedicated process represented by a neural processing unit (NPU) [3]. Table 1 shows the reason for the need of AI-dedicated hardware [4]. As shown in Table 1, the AI-only processor compared to the CPU can be implemented in less than half the size. Of course, it can also be implemented in a smaller size compared to the GPU. Moreover, even if you look at the power, the speed can be implemented in less than half that of the GPU. Finally, it shows superior performance even in TOPS/s, which shows its potential in AI applications.

AI makes inferences and judgments about new data based on the results of learning a lot of data. To learn data, data must be read, and for this, a system that reads and writes memory is required.

As mentioned earlier, when using AI as an existing CPU or GPU, it can use the memory system connected to the CPU and GPU. Figure 1 is a picture of the CPU, GPU, and AI using the memory system. As shown in Figure 1, even if a dedicated AI core is developed without using a CPU or GPU, the use of the memory system remains unchanged [5,6]. The most popular memory system at present is the DDR4 method specified by the Joint Electron Device Engineering Council (JEDEC) as standard. The DDR4 operates at a maximum speed of 1600 MHz and 3200 MT/s [7]. In addition, data transmission can be performed simultaneously from 8- to 32-bit. However, memory systems such as DDR4 require special physical layers, as mentioned above, and the use of these physical layers leads to certain problems, such as: First, for high speed, a large number of delay cells are required to match the skew between PLL, high-speed IO, and data bits. Second, operating systems, including the CPU and GPU, must implement data processing methods for their memory systems. Although the size and speed of the AI are different depending on the implementation method, the DDR4 mentioned earlier is very likely to have a larger area of the entire chip than the AI implemented by the total size of the memory system. In addition, the level of implementation if the memory system is also difficult, so it may be hard to implement the AI chip. To solve the above problems, a simple and effective memory system is required for implementation [8–10]. In this study, the following method was followed to satisfy the above conditions. First, a simple memory controller was implemented to read and write the data in the external memory. The memory controller was designed to be flexible so that the operation could be changed according to the external memory through I2C. Second, the external memory and the chip were connected through GPIO. Since this is a common IO, no separate design is required, but an algorithm that automatically prevents skew between data even when the internal data are distorted with respect to the external environment (e.g., package, PCB design, and actual temperature and humidity) is useful. This algorithm ensures stable reading and writing of memory. In addition, through compensation and monitoring circuit, the algorithm was implemented to use the system stably in the external environment where AI is used. Before making the chip, the AI environment was verified using an FPGA. The proposed memory interface system was implemented using the Samsung foundry 65 nm process and verified by connecting the test board and the external memory to the chip.

Table 1. CPU, NPU, and AI size performance power comparison.

Model	mm^2	nm	Measured		TOPS/s	
			Idle	Busy	8b	FP
Haswell E5-2699 v3	662	22	41 W	145 W	2.6	1.3
NVIDIA K80 (2 dies/card)	561	28	25 W	98 W	–	2.8
TPU	<331	28	28 W	40 W	92	–

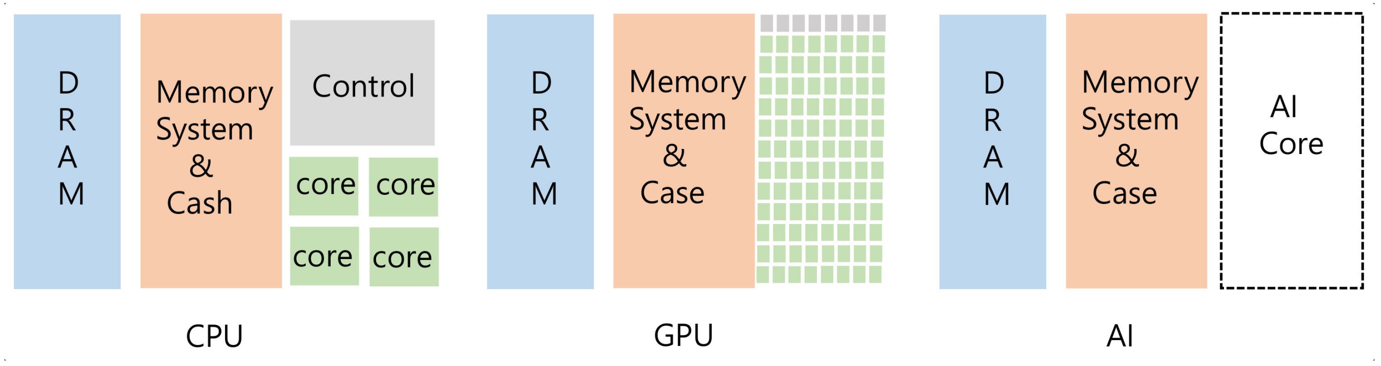


Figure 1. Comparison of CPU, GPU, and AI.

2. Design

The proposed method consists of the blocks shown in Figure 2. First, a delay cell is connected to an external memory through the general purpose input output (GPIO). Each delay cell is connected to a delay control block to adjust the amount of delay. The output of the delay is connected to the memory controller, whose role is to deliver addresses, reads, and writes in a timely manner with respect to memory operations. There is a compensation block under the memory controller that not only adjusts skew between pins connected to the memory, but also corrects data to be reliably exchanged from the external environment at the beginning of the system. The core, which is a processor of AI, uses the necessary data by passing instructions to the memory controller to read and write the data from external memory.

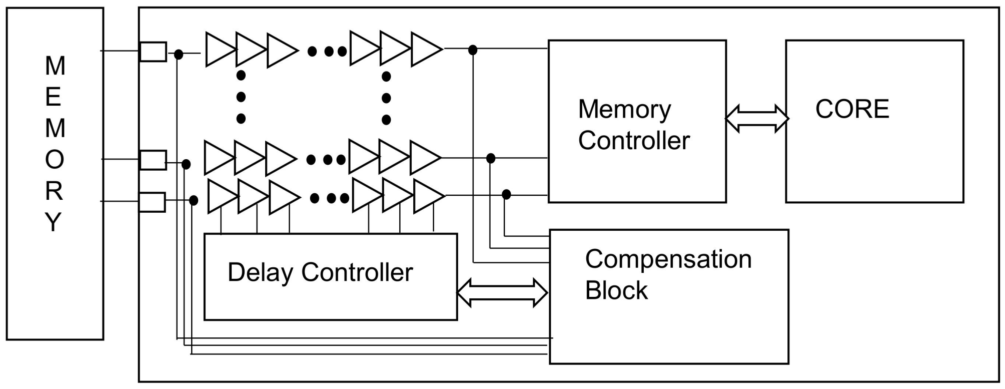


Figure 2. Top block diagram.

2.1. Delay Logic

The delay controller is a block that controls the amount of delay by turning the delay cell on and off. Delay controllers can be connected to each delay chain to control the delay individually. A duty cycle check (DCC) is connected to the compensation block to ensure that the data are at the center of the clock, so that the data can be sent and received reliably. The delay cell used in this study was a primitive cell provided in the process. Therefore, no analog design or layout was required, and the place and routing (P&R) could be carried out quickly after designing using the register transfer level (RTL).

Figure 3 shows a block diagram of one delay chain. The inverter is connected, and the output of each inverter is connected to the multiplexer so that the desired delay output can be selected by the control pin. To design an RTL using a delay cell, a model must be created by calculating the delay corresponding to the inverter cell provided by the process. Figure 4 shows the simulation logic code for the inverter logic.

The inverter cell among the primitive cells provided by the process is selected, and the delay value of the inverter cell is calculated and put it in the # delay to see the change in the delay in the RTL simulation and to check the overall operation. Figure 5 shows the simulation results, indicating that the delay increases as the delay step is increased using the simulation model. In Figure 5, each step can increase the delay with the delay set in #delay in Figure 4: (1) is the result when the delay is made into one step and it pushes about 200 ps, while (2) shows that 6600 ps is pushed because of running the simulation by pushing 33 steps. Among the primitive cells provided by the process, it is possible to use the primitive cells suitable for the desired amount of delay for the testing.

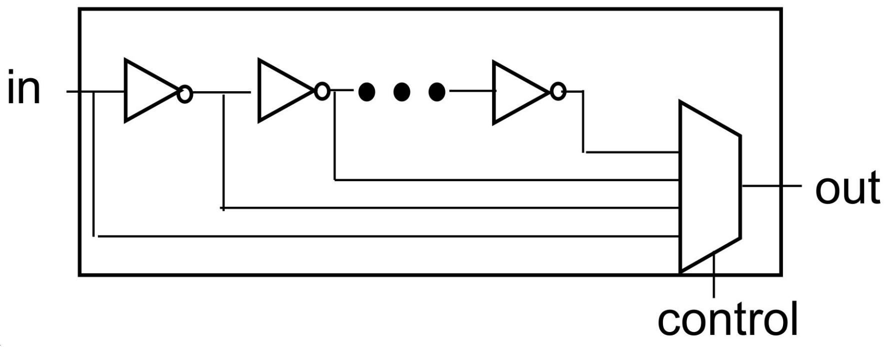
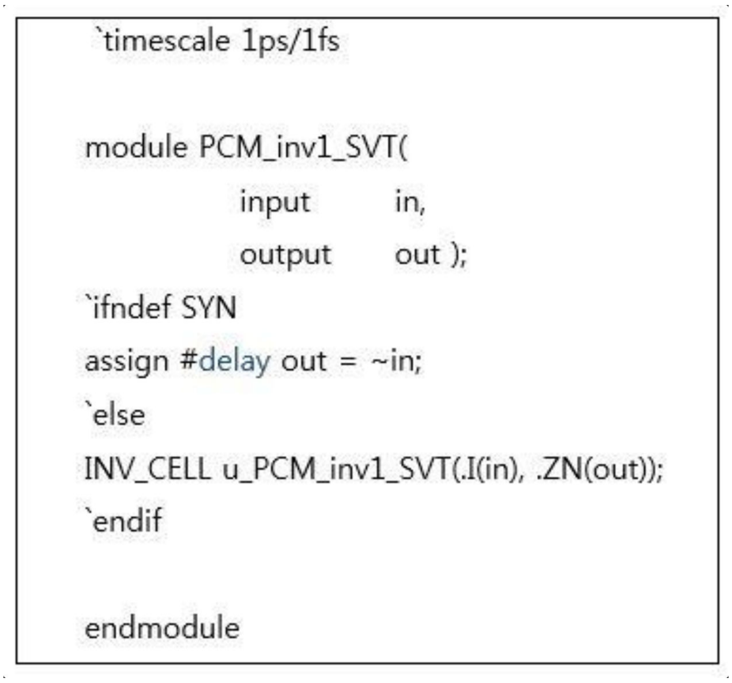


Figure 3. Delay chain unit.



```
 `timescale 1ps/1fs

module PCM_inv1_SVT(
            input       in,
            output      out );
`ifndef SYN
assign #delay out = ~in;
`else
INV_CELL u_PCM_inv1_SVT(.I(in), .ZN(out));
`endif

endmodule
```

Figure 4. Simulation model of inverter logic.

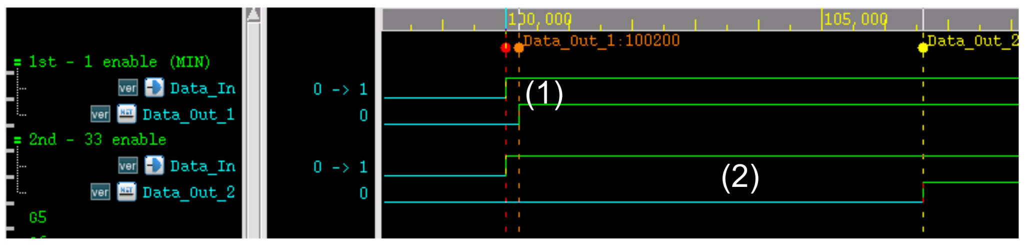


Figure 5. Simulation result using the inverter model.

The delay controller is a combination of mux, as shown in Figure 3. It is directly connected to the compensation block and indirectly to the memory controller. Its main role is to increase or decrease the amount of delay in the memory controller or compensation block, and it outputs the desired delay by changing the mux output. The number of delays should be adjusted to an even number, as odd number of adjustments will cause the frequency or data to be inverted, resulting in unwanted results.

2.2. Memory Controller

The memory controller is responsible for reading and writing data in accordance with the specifications of the memory [11]. In this study, a memory that can be used in the AI chip and can be purchased in the market was selected rather than a memory with very good performance. In this study, AS6C6416-55TIN SDRAM developed by Alliance Memory was used, which has a storage capacity of 64 M bits.

Figure 6 shows the timing diagram for reading and writing SDRAM.

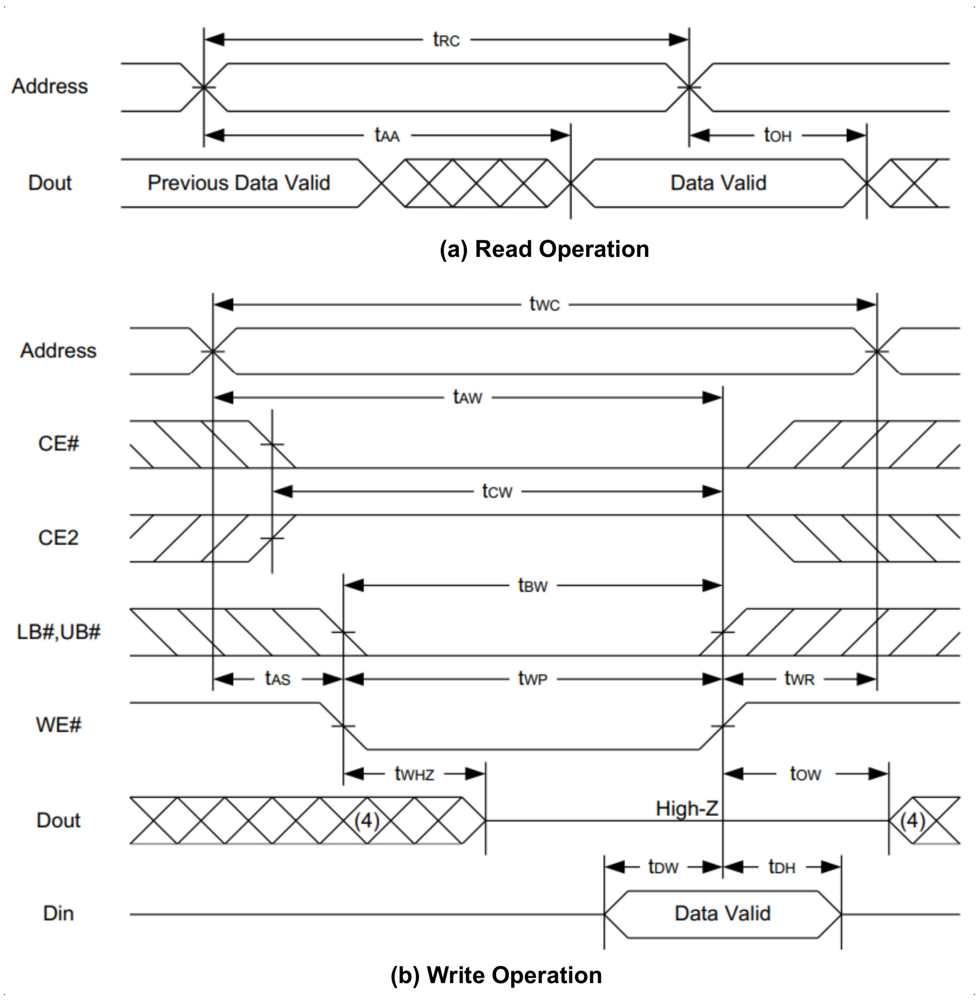


Figure 6. SDRAM read/write operation.

The read operation proceeds in the following order:

a. The corresponding address is written first.
b. The wait is as long as tAA.
c. After the tAA time, the data value of the corresponding address is output at DQ0~7 pin.

The write operation is as follows.

a. The desired address is written.
b. The CE # (chip enable #) is set to 0 and CE2 to 1.
c. The LB # (low byte) and UB # (up byte) are both set to 0.
d. The WE # (write enable) is set to 0.
e. The desired value is written from DQ0~7.

2.3. Compensation Circuit

The compensation block consists of two blocks. Block 1 confirms that the location of the data detected by the delay is in the desired position, while block 2 compensates for the error so that the current system can operate stably from the external environment. The block that finds the location uses the method of delay lock loop (DLL) [12], which is a circuit that checks whether the data exist at the desired position by the delay chain. Figure 7 displays a block diagram of this circuit.

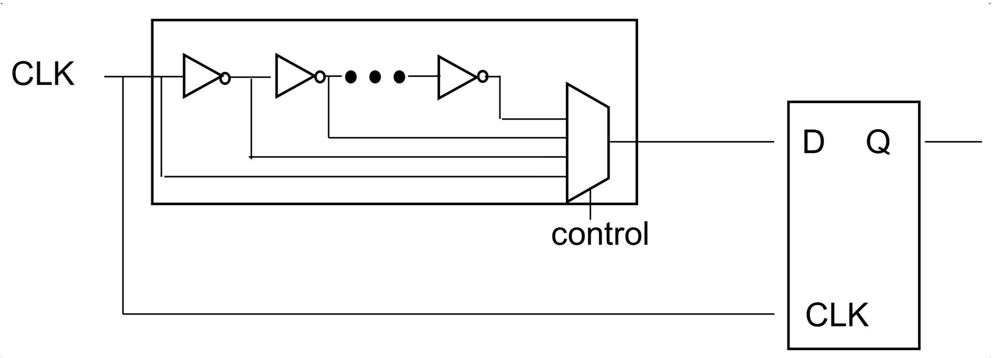


Figure 7. Delay check circuit.

Compensation Algorithm

Figure 8 shows how the output Q of the flip-flop changes with changing delays. The Q value changes when the clock cycles over. This lets you know where the current delay is.

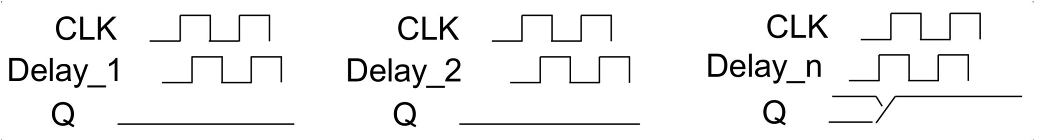


Figure 8. Delay checker out value.

We used primitive cells to provide D Flip-Flop in the process. Similarly to the model for the delay cell, the flip-flop should be simulated by applying the amount of delay provided in the process.

The compensation block is a Build-In-Self-Test (BIST) method [13]. Compensation operates in the following order:

a. The register set is changed to compensation enable mode with I2C.
b. The memory controller in Figure 2 writes the 0xaa data to address 0.
c. After writing the operation, the value of address 0 is read.
d. If the read value is not 0xaa, the delay value of DQ is increased to unmatched bits.
e. If the data match, it passes.
f. A value of 0x55 written to the address.
g. The same procedure as above is followed.

Since 0xaa is 10101010 and 0x55 is 01010101, we can examine both 0 and 1 of the data. Usually, CMOS circuits change their operating speed due to external conditions such as temperature and humidity. Therefore, for accurate data transmission, the distortion of the data due to the surrounding situation can be corrected through the delay cell using the above-described compensation algorithm.

3. Implementation

3.1. FPGA

For system-level verification, we implemented YOLO-V2, the most used algorithm when performing object recognition in an embedded environment in recent years, in the FPGA, and checked the result and operation. First, the YOLO-V2 algorithm was verified using the C simulation function of Xilinx's Vivado high-level synthesis (HLS). Next, the block to be implemented with the actual FPGA was converted into IP using HLS, and the entire block was designed using the Vivado tool and the IP synthesized through HLS. Using Vivado SDK, the board was set up to measure communication with the control host PC, the FPGA control signal, and the processing time, and the design was finalized using Petalinux to enable on FPGA board.

The FPGA implementation environment was Xilinx ZCU102 and Zedboard, and detailed specifications are shown in Table 2. For implementing YOLO-V2, it was applied to ZCU102, which has sufficient resources, and Zedboard, which has relatively fewer resources.

Table 2. Xilinx ZCU102 FPGA and Zedboard specifications.

	ZCU102	**Zedboard**
FPGA	zynq ultra + MPSOC	zynq-7020
BRAM 18 kb	1824	280
DSP	2520	220
FF	548,160	106,400
LUT	274,080	53,200

Table 3 outlines the software environment required for the experiment, and to compare it in the same environment as the lightweight YOLO-V2 design, it was implemented with YOLO-V2 of 224 $\times$ 224 in size and 416 $\times$ 416 (original size). The 224 $\times$ 224 YOLO-V2's total computational load is 11.7 BFlops, which is a smaller design than the 416 $\times$ 416 YOLO-V2 with 26.464 BFlops.

Table 3. Environment software.

	Software	
Algorithm	YOLO-V2	YOLO-V2
Input size	224 $\times$ 224	416 $\times$ 416
Flops (BFlops)	11.7	26.464
Batch size	1	1
Precision	Fixed 16	Fixed 16

The verified algorithm was synthesized into real FPGA IP through Vivado HLS. At this time, the use of the internal #pragma instruction varied according to the number of resources of the target FPGA. Since ZCU102 has a relatively good number of resources compared to Zedboard. On the contrary, Zedboard cannot unroll many loops because of its lower number of resources. Therefore, the parameters were designed according to the resources of the target FPGA. Table 4 indicates a large amount of digital signal processors (DSPs) as the number of resources when synthesized with ZCU102 and Zedboard.

Table 4. ZCU102 and Zedboard DSP synthesis results.

	ZCU102	**ZCU102**	**Zedboard**
Input size	224 $\times$ 224	416 $\times$ 416	416 $\times$ 416
BRAM 18 kb	366	528	101
DSP	288	384	142
FF	127,901	169,345	32,086
LUT	138,739	185,031	34,192
Power (W)	3.82	4.48	2.54

Figure 9 shows a block diagram of the entire FPGA system. The IP synthesized through HLS was connected to the PS for control, and AXI was used to access the DRAM and divided data.

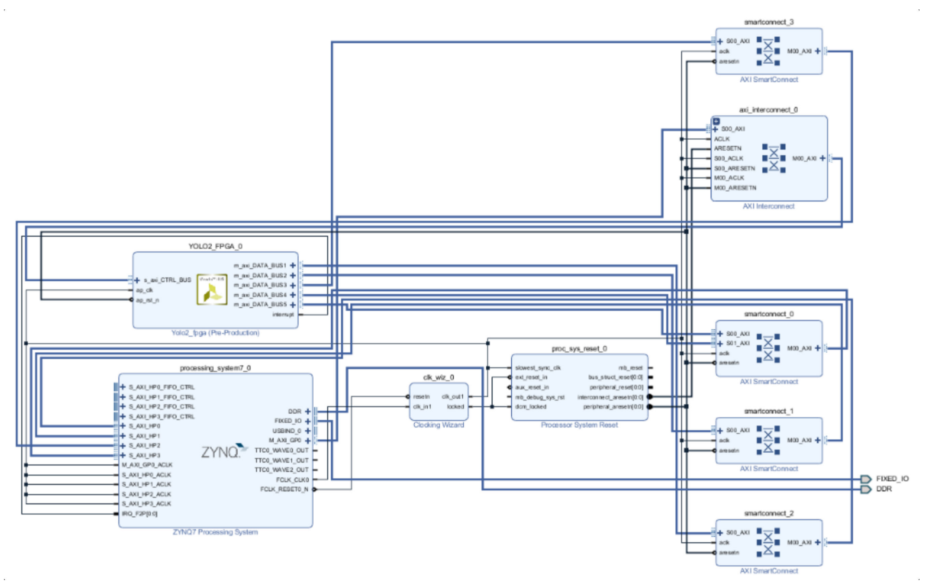


Figure 9. FPGA block diagram.

3.2. ASIC Chip

Figure 10 shows the overall chip design. Two memory controllers are connected, and the I2C slave and register are placed inside to control the register chip. The core is a circuit related to image compression. Table 5 shows the composite size of the delay cell and the P&R size of the actual chip. The unit delay cell is the size of the cell used for the two inverters and mux, while the memory controller is the sum of the sizes of some flip-flops and debugging operations inside. The delay control includes logic to add and subtract blocks and other delays that determine the edge.

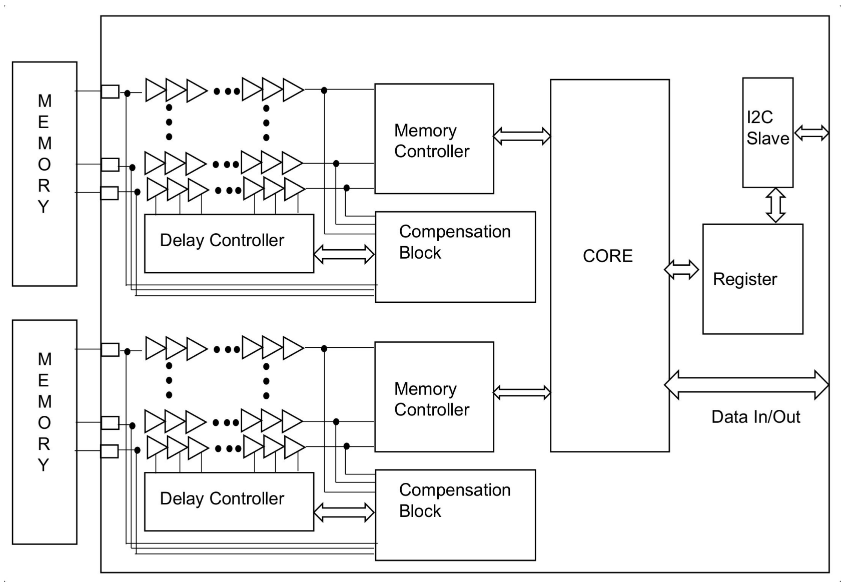


Figure 10. ASIC chip block diagram.

Table 5. Delay cell size.

Unit	Synthesis	Place and Route
Unit delay cell	10.24 μm^2	13.312 μm^2
Memory control	276.48 μm^2	365.23 μm^2
Delay control	792.32 μm^2	1171.712 μm^2

Table 6 is a comparison between a previous study and this study. The memory system is largely composed of PHY, including IO and a memory controller. In this study, PHY and memory controller were implemented together. The table also shows the results of comparing the gate count of this study, designed with a 65 nm process as one and relatively size. Because processes differ from process company to process company, general Moore's Law was applied and compared. In [13], the PHY of LPDDR4 was implemented by the Samsung 10 nm process. Compared to this study, the size was small, but considering the process, the size was approximately three times larger. Moreover, if the implementation of the controller is added, the size is expected to be approximately seven times. [14] provided a thesis implemented on the controller, excluding PHY, in the memory system. The memory type was not known exactly, but when compared to this paper, a size of approximately four times larger can be determined. [15] implemented PHY for DDR3. Again, only the PHY was designed, but it was approximately three times larger than that in this study. Although it is difficult to decipher from Table 6, this study, implemented purely digitally, has two advantages in contrast to [13,15]. The first is the time to reach implementation. It is difficult to quantitatively judge the time spent in purely digital design compared to analog circuit design, but considering the layout or simulation time, purely digital design takes much less time to implement. The second is the cost aspect. As mentioned earlier, the cost is reduced because the implementation time is shortened, but there is an effect of reducing the die size because the layout is optimized on the chip.

Table 6. Size comparison with a conventional memory system.

	[13]	**[14]**	**[15]**	**This Work**
Size	0.57 mm^2	4.71 mm^2	3.1 mm^2	1.171 mm^2
Process	10 nm	65 nm	40 nm	65 nm
Design	PHY	Controller	PHY	PHY + Controller
Memory Type	LPDDR4	n/a-	DDR3	SDRAM1
Relative size	3.42	4.02	3.97	**1**

Table 7 shows the power measurement results. The operation voltage was 1.2 V. "Busy" measured here is the power consumed for reading and writing the data. In particular, a lot of power was consumed in the operation of the IO and the delay cell. In the future, if the delay cell is optimized, less power consumption can be expected.

Table 7. Operation power.

Unit	**Power**
Busy	4.0738 mW
Idle	0.6719 mW

As mentioned in the introduction, the existing AI system does not have its own memory system, but instead uses the memory system used by the GPU or CPU. Table 8 shows a comparison of the memory used in the existing AI environment and the memory used in this study.

Table 8. Comparison of the existing memory system and the proposed memory [16].

Feature	**DDR4**	**GDDR5**	**HBM2**	**This Work**
Monolithic die density	16 Gb, 8 Gb	8 Gb	8 Gb	0.064 Gb
IO Configuration	x4, x8	x16, x32	x1024	x16
Clock Speed	3.2 Gb/s	10 Gb/s	2.0 Gb/s	0.02 Gb/s
Band width	25.6 Gb/s	80 Gb/s	256 Gb/s	0.32 Gb/s

As shown in the results of Table 3, when YOLO-V2 was implemented in Zedboard, 101 18 kb BRAMs inside the FPGA were used. It is difficult to map the RAM used in the FPGA and the external RAM 1:1, but 18.18 Mb of RAM is used numerically. This means that in a simple AI application such as YOLO-2, a limited memory system is available.

As mentioned in Section 2, since the first stage of the delay cell is 200 ps, the first stage of the delay cell cannot be accurately measured using an oscilloscope. However, when the delay cell is increased in 10-step units, it is possible to confirm delayed data by the delay cell. In this study, much smaller delay steps could be obtained by using synthesized delay cells rather than analog designs. Table 9 displays the delay step of our work compared to other studies. The smaller the delay step, the more precisely the skew of the data bus can be adjusted, so the smaller the value that can be adjusted in one step, the better.

Table 9. Comparison of delay step.

This Work	This Work	[17]
Process	65 nm	130 nm
Period	200 ps	340 ps

Figure 11 is the result of measuring the data pin using an oscilloscope. A total of three data points were measured at the same time, and a phase difference was deliberately made using a delay controller for the measurement. Looking at Figure 11a, we can see that the skews of DQ0, DQ1, and DQ2 are different. Figure 11b shows the result of adjusting the skew by adjusting the delay controller corresponding to DQ1 through I2C.

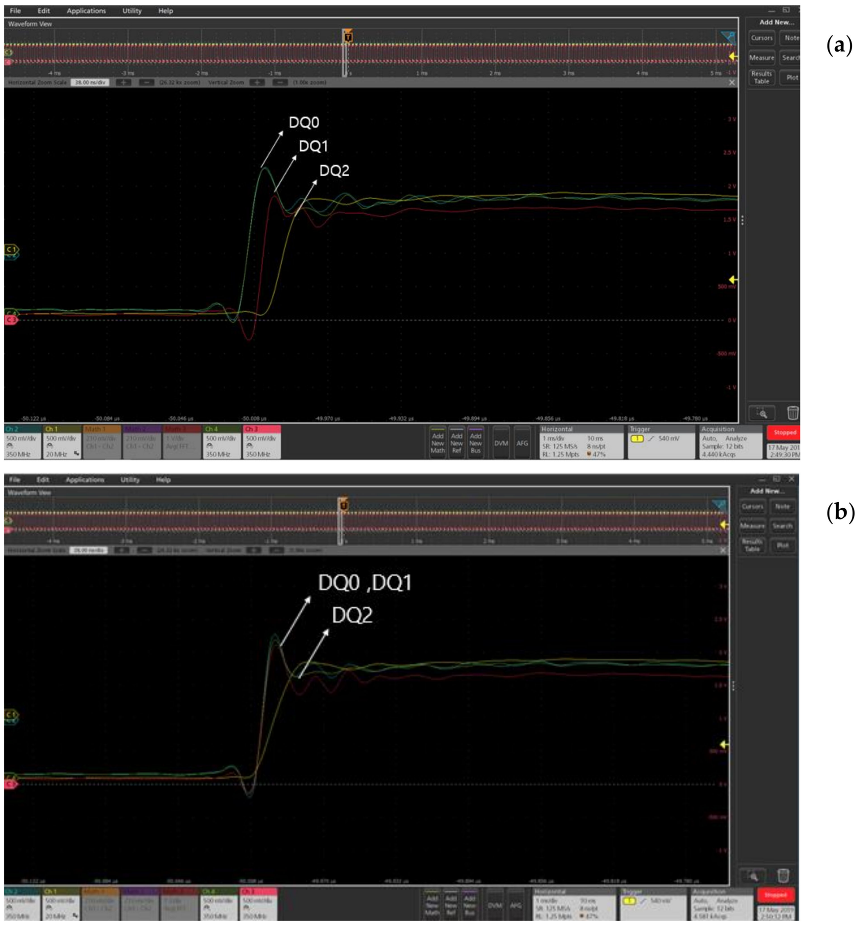


Figure 11. (**a**) Data signals before alignment using the delay controller. (**b**) Data signals after alignment using the delay controller.

The read and write of the memory can be confirmed indirectly using I2C. If we write the address and data, we want it to register inside the chip through I2C and to write the corresponding pin in the order of Figure 11, where the data in SDRAM is saved in the register inside the chip. The value was read through the external I2C master and was found to be identical to the value written in the SDRAM.

Table 10 describes the time it takes to read and write data from the external memory. In this study, it was designed to operate from 1 to 100 MHz, but this is the result when measured based on 20 MHz.

Table 10. Read/write time.

Unit	clk	One Time	16-bit/16 MB	16-bit/32 MB
Read	4	200 ns	2 s	4 s
Write	5	250 ns	1.6 s	3.2 s

The memory used in this study can be used in both 8- and 16-bit modes. Table 10 describes the time required to read and write 16 and 32 MB.

Figure 12 is the environment for measuring the chip. The chip used a socket to enable change, and the FPGA was used to set the environment of the I2C master and measurement board. For measurement, all of the IO of the chip was pulled out with pins.

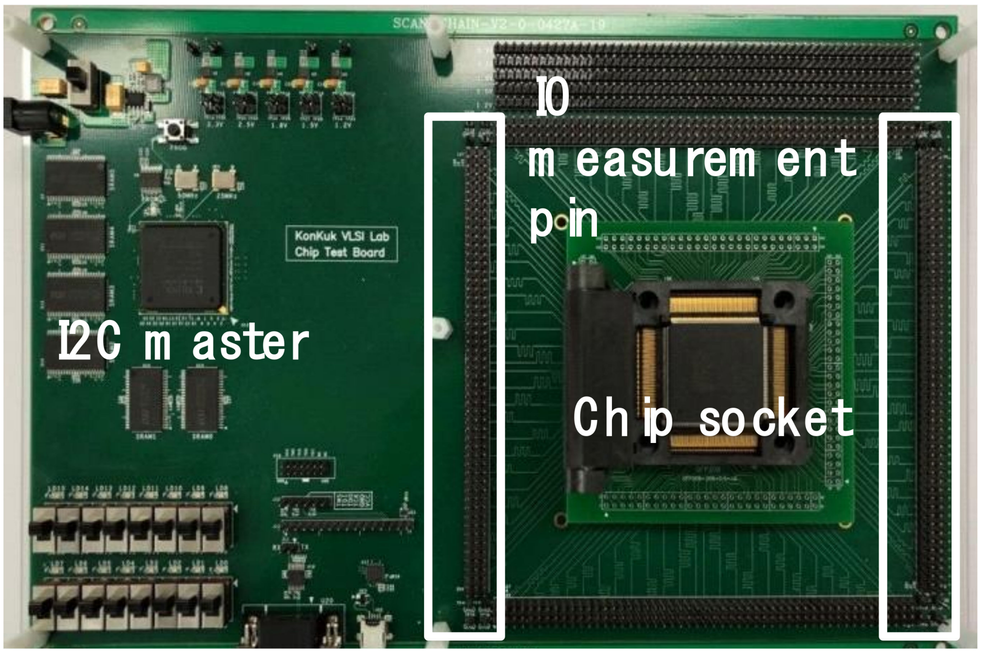


Figure 12. Test environment.

4. Conclusions

In this study, we proposed and implemented a memory system that is effective in an AI environment. To determine how much memory is needed in an AI environment, the required memory capacity was calculated by implementing the YOLO application in an FPGA. Based on this, an easy-to-implement and efficient memory system was designed and manufactured using Samsung Electronics' 65 nm process. As a result of this, it was possible to design all digitally with a size more than three times smaller than that of the system using the existing LPDDR. In the future, we will implement the YOLO application in an FPGA and verify it, and this will be implemented in a chip.

Author Contributions: Conceptualization, T.K.; methodology, T.K.; software, S.P.; validation, T.K. and S.P.; writing—original draft preparation, T.K.; writing—review and editing, Y.C.; visualization, T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yoon, Y.H.; Hwang, D.H.; Yang, J.H.; Lee, S.E. Intellino: Processor for Embedded Artificial Intelligence. *Electronics* **2020**, *9*, 1169. [CrossRef]
2. Kang, M.; Lee, Y.; Park, M. Energy Efficiency of Machine Learning in Embedded Systems Using Neuromorphic Hardware. *Electronics* **2020**, *9*, 1069. [CrossRef]
3. Park, S.S.; Chung, K.S. CENNA: Cost-Effective Neural Network Accelerator. *Electronics* **2020**, *9*, 134. [CrossRef]
4. Jouppi, N.P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Yoon, D.H. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, Toronto, ON, Canada, 24–28 June 2017; pp. 1–12.
5. Madhuri, R.A.; Hampali, M.M.; Umesh, N.; Pooja, K.S.; Shirur, Y.J.M.; Chakravarthi, V.S. Design and Implementation of EDMA Controller for AI based DSP SoCs for Real-Time Multimedia Processing. In Proceedings of the 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 7–9 October 2020.
6. Shiah, C.; Chang, C.N.; Crisp, R.; Lin, C.P.; Pan, C.N.; Chuang, C.P.; Lu, N. A 4.8GB/s 256Mb(x16) Reduced-Pin-Count DRAM and Controller Architecture (RPCA) to Reduce Form-Factor & Cost for IOT/Wearable/TCON/Video/AI-Edge Systems. In Proceedings of the 2019 Symposium on VLSI Circuits, Kyoto, Japan, 9–14 June 2019.
7. DDR4 SDRAM STANDARD. Available online: https://www.jedec.org/standards-documents/docs/jesd79-4a (accessed on 31 May 2021).
8. Suri, M.; Gupta, A.; Parmar, V.; Lee, K.H. Performance Enhancement of Edge-AI-Inference Using Commodity MRAM: IoT Case Study. In Proceedings of the 2019 IEEE 11th International Memory Workshop (IMW), Monterey, CA, USA, 12–15 May 2019.
9. Lai, T.Y.; Chen, K.H. On-Chip Memory Optimization of High Efficiency Accelerator for Deep Convolutional Neural Networks. In Proceedings of the 2018 International SoC Design Conference (ISOCC), Daegu, Korea, 12–15 November 2018.
10. Lin, C.H.; Cheng, C.C.; Tsai, Y.M.; Hung, S.J.; Kuo, Y.T.; Wang, P.H.; Chen, C.C. 7.1 A 3.4-to-13.3TOPS/W 3.6TOPS Dual-Core Deep-Learning Accelerator for Versatile AI Applications in 7nm 5G Smartphone SoC. In Proceedings of the 2020 IEEE International Solid-State Circuits Conference-(ISSCC), San Francisco, CA, USA, 16–20 February 2020.
11. Sreehari, S.; Jacob, J. AHB DDR SDRAM enhanced memory controller. In Proceedings of the 2013 International Conference on Advanced Computing and Communication Systems, Coimbatore, India, 19–21 December 2013; pp. 1–8.
12. Waris, M.; Mehta, U.; Kumaran, R.; Mehta, S.; Chowdhury, A.R. An all digital delay lock loop architecture for high precision timing generator. In Proceedings of the 2015 19th International Symposium on VLSI Design and Test, Ahmedabad, India, 26–29 June 2015; pp. 1–6.
13. Chae, K.; Choi, J.; Yi, S.; Lee, W.; Joo, S.; Kim, H.; Lee, S. A 690 mV 4.4Gbps/pin all-digital LPDDR4 PHY in 10 nm FinFET technology. In Proceedings of the ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference, Lausanne, Switzerland, 12–15 September 2016.
14. Sudarshan, C.; Lappas, J.; Weis, C.; Mathew, D.M.; Jung, M.; Wehn, N. A Lean, Low Power, Low Latency DRAM Memory Controller for Transprecision Computing. In Proceedings of the Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2019), Samos, Greece, 18–20 July 2019.
15. Kaviani, K.; Wu, T.; Wei, J.; Amirkhany, A.; Shen, J.; Chin, T.J.; Yuan, X. A Tri-Modal 20-Gbps/Link Differential/DDR3/ GDDR5 Memory Interface. *IEEE J. Solid-State Circuits* **2012**, *47*, 926–937. [CrossRef]
16. Godse, R.; McPadden, A.; Patel, V.; Yoon, J. Technology enabling the next Artificial Intelligence revolution. In Proceedings of the 2018 IEEE Nanotechnology Symposium (ANTS), Albany, NY, USA, 14–15 November 2018.
17. Yuan, H.; Long, T.; Yue, Y. Build-in-self-test of a real time digital signal processing system. In Proceedings of the 2001 CIE International Conference on Radar Proceedings (Cat No. 01TH8559), Beijing, China, 15–18 October 2001; pp. 983–986.