Protocol for a Simplified Processor-Memory Interface Using High-Speed Serial Link

HyukJe Kwon(✉) and Yongseok Choi

Electronics and Telecommunications Research Institute,
218 Gajeongdong, Yoseong-Gu, Daejeon, South Korea
{heavenwing,shine24}@etri.re.kr

Abstract. In our work, the interface protocol between a processor and memory built using an optical connection is described. We designed a serial interface protocol to simplify a parallel interface between the processor and memory, and implemented the protocol engine to be executed on the interface. There are three main roles of the protocol engines. The first is the data collection and sorting. The second is the data error detection and retransmission request. The third is packetizing the memory command and data.

Keywords: Memory · DDR · Protocol · Serial connection · Optical connection

1 Introduction

Owing to the development of the Internet and personal communication, a large amount of data and information has been recently produced and circulated. Because a large amount of data requires faster networks and a more effective processing, the requirements of more computing and more memory capacity have emerged. However, there is no memory structure satisfying the features of such applications as "small computing, big memory" [1]. In addition, the issues of more required memory bandwidth and capacity are emerging from the recently evolving "multi-core and many-core" idea [2, 3]. If the memory capacity and bandwidth per core are evenly assigned, as the number of cores increases, the capacity and bandwidth are reduced. To maintain a constant performance of the core, only increasing the capacity of the local memory is not appropriate.

A processor's memory access is made through a memory channel. More memory capacity and bandwidth are required at server-class data centers. And to ensure the stability of signals in channels, the memory should be closely attached to the server processor [2]. But the process's region to connect to memory is limited. Also, to stably transfer a normal data signal using on-board parallel channels, meticulous P&R is required.

A number of studies on improving the performance per core are still ongoing. There are several requirements to improve the performance of memory systems: the growth of the memory capacity, decreased memory latency, an expansion of the memory bandwidth, and an increased speed of the memory I/O operation, among others [4–6].

To increase the memory capacity, a stacked structure has been exploited. In recent years, other researchers have been conducting research and development for stacking the memory cells in the memory package. Such approaches include a stacked structure hybrid

Y. Zhang et al. (Eds.): CloudComp 2015, LNICST 167, pp. 301–310, 2016.
DOI: 10.1007/978-3-319-38904-2_31

memory cube (HMC) [7, 8], high bandwidth memory (HBM), and Wide IO [2, 9]. Because memory cells with the same capacity may be stacked vertically in a limited area, they can increase the memory capacity. However, because a stacked structure is built up vertically, the die breakage is caused by an increased weight of the middle layers [10, 11].

Studies on reducing the memory latency time are carried out mainly on SDRAM, which has a relatively long latency time. Because SDRAM has a 1T1C structure, it requires a sufficient amount of time to be charged and discharged to access data stored in the capacitor. Until now, there has been no structure capable of reducing the charging or discharging time of the capacitor. Several studies on methods for improving the performance of the SDRAM cell's internal I/O by connecting to an optical fiber are ongoing [12, 13].

To increase the memory bandwidth, while the I/O operation speed has a limit of 200 Mbps, the I/O count increased by 512 and 1024, such as Wide IO(2) or HBM. These can be connected directly using external IO connections, or the connection between the memory and processor can be configured on SiP (system in package).

The focus of this paper is on memory I/O. An I/O associated with existing memory is in the form of a parallel bus interfacing with a processor. In this case, if the other memories are expanded onto the main board, the signals of those will become difficult to connect to a processor. Because there are many signals to interface with a processor. To resolve and manage these points, using high-speed serial communication and applying a packet protocol engine design, we designed protocol engines that are able to communicate a processor with memories [1, 14].

We describe the conventional structure between the CPU and memory in Sects. 2.1 and 2.2. In Sect. 3, the proposed protocol engines are described through a read and write memory operation. Section 4 then provides a description of the implementation and test of the protocol engines. Finally, in Sect. 5, some concluding remarks are provided.

2 Memory Systems and Protocol Engines

A central processing unit mainly performing the operation of each core has a separate cache for each storage device. It has from one to eight configurable cores. The memory cache for each core and the shared cache memory are interfaced. The shared cache memory, being interfaced with the memory controller and the peripheral controller, can exchange the data with those.

For the treatment of a large-capacity and high-speed main memory unit, a DDR memory is mainly used, and the interface of the data and memory controller in the processor is generally 32-bit or 64-bit. In addition to the 40-bit address and control information, more than 100 signal lines are required for a memory system to normally operate. In the case of the latest processors, the memory channel (dual) bus occupies about 50% of all package pins [16].

In [17], a bottleneck in the system performance between the processor and memory was reported. To solve this problem, the memory architecture of a 24 point-to-point method based on the memory module daisy-chain network is investigated. In [4], the SRAM memory was used in the research on memory- processor communication. In [1],

it was formed in terms of a shared memory system. Specifically, rather than handling the memory I/O to improve the performance of the memory system, the memory formed by the silicon photonics to increase the operation speed in the inner memory has been studied. [3, 12].

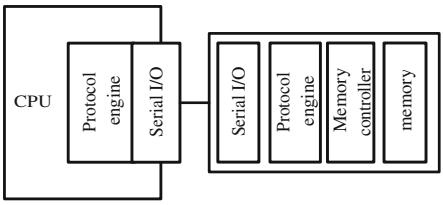

Fig. 1. Proposed memory system including the protocol engine.

In this paper, we propose a memory system to deal with communication problems between a processor and memory controller. Earlier studies dealt with a memory controller and processor, or a memory controller and memory. This paper is similar to [15] in which the memory controller is installed in the in/out side of the processor, and overrides these functional systems. We do not use the parallel communication of a conventional method between the memory controller and the memory bus, and the memory chip is built inside the memory controller. If the memory controller is located inside a memory chip, as shown in Fig. 1, the function of the memory controller can be separated. The master protocol engine of the main control processor is in charge of the memory controller and acts as a packetizer of the memory address and data generated by the processor. The memory controller has a memory part dependent control (slave) protocol engine, and the memory system can be configured together.

2.1 Proposed Memory Structure

The Double Data Rate (DDR) has a concept in which the data bandwidth is widened, that is, the memory chip latches the data on the 8 DQ (in/out) at the two edges (rising and falling) of the clock. To widen the bandwidth of the data, we have to increase the I/O clock speed, and we must also be able to afford a higher operating clock in which pre-fetched data are prepared in advance, which is a necessary task. However, the capacitor with SDRAM has a charge/discharge feature. Because of these features, the operating speed of the internal SDRAM does not increase as much as the IO operating speed, and the SDRAM internal speed is maintained without a significant change.

In this paper, we propose the reduction of the memory IO counts and the remove of the timing protocol of the DDR. The proposed scheme does not need different complex timing information to be controlled by the memory controller. The interface of the PCB or external IO can be simplified because the timings are dealt with by the inner memory because the interface is serial.

The leveling is a latching type in which the DQS signal can be able to latch the same value DQ in memory, and the use of a DQ and probe (strobe) is a way to control the DQS controlled by the memory controller. Two leveling types are used, reading and writing, and

these processes are certain to be executed so that we obtain stable data in DDR3/4. Though the leveling is the skew of a method to correct a plurality (DQ count is 64) of data lines, when we are introducing a packet method, this feature may not be used.

2.2 Proposed Protocol Engines

There are three main roles of the protocol engines. The first is the data collection and sorting. The second is the data error detection and retransmission request. The third is packetizing the memory command and data. The protocol engine configuration can be divided into parts of the processor and the memory chip.

The master protocol engine is responsible for packetizing of the generated memory-related commands, data, and address by the processor. Another main role of the master protocol engine is to manage the detection and correction of the errors in the data collected, and if the re-transmission request is generated, it has to retransmit the packet that had been transmitted to the memory chip. The slave protocol engine is made up of the memory inside the chip.

The slave protocol engine re-analyzes the packet into the command, data, and address, which are transmitted by the master engine control protocol. These are then passed to the internal logic of the memory, and after the slave protocol engine generated the corresponding response packet, the packet is transmitted to the master control protocol engine. In addition, if the slave engine receives the re- transmission request, the engine has to retransmit the packet that had been transmitted to the master protocol engine.

The configuration of the master and slave control protocol engine is shown in Fig. 2. Commonly, the engine is made up of parallelizer and serializer modules. The packet generator takes on the role of reconfiguring the data obtained by the memory controller, or is configured by the processor to access the memory according to the two communication protocols. The packet sequence and CRC are used for calculating the CRC, and the CRC attached to the end of the signal packet is used for error checking. If there are more data than a fixed packet size, the data must be split into multiple packets. The packet sequence module generates the sequence number to indicate where an individual packet was generated in the entire packets. The packet checker module determines the packet processing based on the results of the packet sequence and CRC checking, i.e., whether to use or discard the received packet. At the packet checker, whether there is a

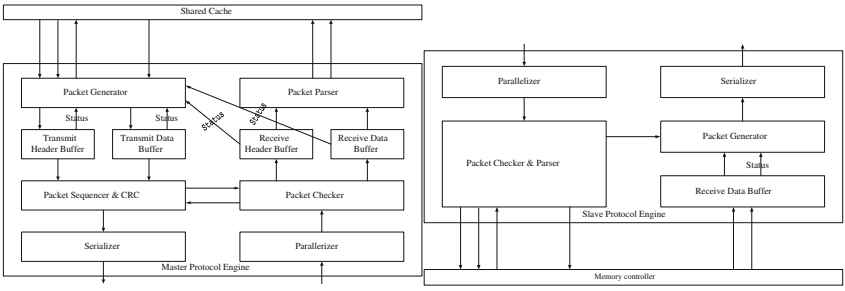

Fig. 2. Configuration of the master/slave protocol engine of connecting optical.

CRC error in a packet or the presence of duplicate packets is determined. If an error has occurred, the packet is discarded. Otherwise, the packet is stored in the receive buffer. In the stored packet, the packet parser analyzes the header and extracts the command from the memory access or collects information on the packet processing result of the other side of the protocol engine. The results collected are transmitted to the processor or memory controller.

The packet retransmission means that the transfer destination device is requested to resend the packet when an error occurs in the packet communication. For this purpose, both protocol engines should keep the transmitted packets in the buffer. If the normal memory access process is completed without the retransmission request, the data in that buffer is to be deleted or classified as not being used, and the data are then replaced with new data.

3 Operation of Protocol Engine

3.1 Write Operation

The processor holds the input data for the engine control protocols using the memory address, memory control information, burst to determine the data length, and mask information. The memory address is the entire memory address that can be accessed by the processor (Fig. 3).

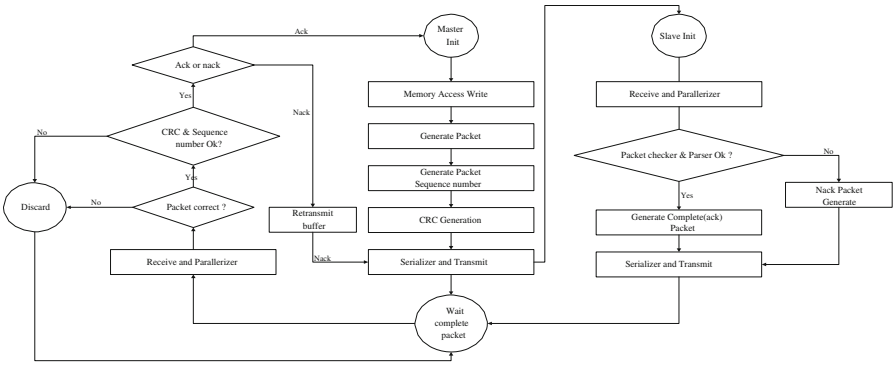

Fig. 3. Write access flow charts.

The packet generator generates a packet and is responsible for storage and access control. The configuration factors are the transmission packet header, transmission data, and packet sequence. The generated packets are immediately transmitted depending on the circumstances of the memory access, or may be temporarily stored in a transmission header buffer and a transmission data buffer to wait for the previous packet processing to be complete. A sequence number is generated by the packet sequence, and it is argued that the master protocol engine sends this number to the slave protocol engine to confirm the order and delivery of the packet header and the data necessary for reconstruction of the factor.

The packet checker and parser in the slave control protocol are passed to the memory controller to execute a write access if a packet is received with a correct CRC. Otherwise,

the packet is discarded. If that access is a completed operation, the slave protocol engine transmits the information indicating that the operation is complete to the slave control protocol. The master protocol engine proceeds to the next access after receiving a complete write access operation.

If the slave protocol engine received the packet with errors, it generates the packet based on indicating that there are problems with the write operations, and then transmits. As a result, the master protocol engine retransmits the write access packet until it receives a normal response and information for some time. Even if the received information of a normal response packet arrives later than expected, the master protocol engine judges that the write access packet sent to the main memory has not been communicated well and will retransmit that packet.

3.2 Read Operation

The slave protocol engine periodically transmits the packet to the master to be informed the state of the reception data buffer. The packet generator in the master protocol engine will consider the status of the transmission and reception header buffer and the status of reception data buffer, and also will consider the state of the reception data buffer in the slave protocol engine. Considering the above condition, a packet that includes content such as the memory address, reading field, mask information, and length of the data, is stored in the buffer (Fig. 4).

After inspecting the sequence number and CRC of the read access packet received, the packet checker and parser in the slave protocol engine transmit the memory address and the control information to the memory controller. Moreover, the slave protocol engine generates and transmits simultaneously the read access reception information to the master protocol. The packet checker and parser are redundant packet inspections and if duplicated, the packets are discarded. As the data accessed from memory by the memory controller is stored in a buffer, that data will be transmitted to the master protocol engine.

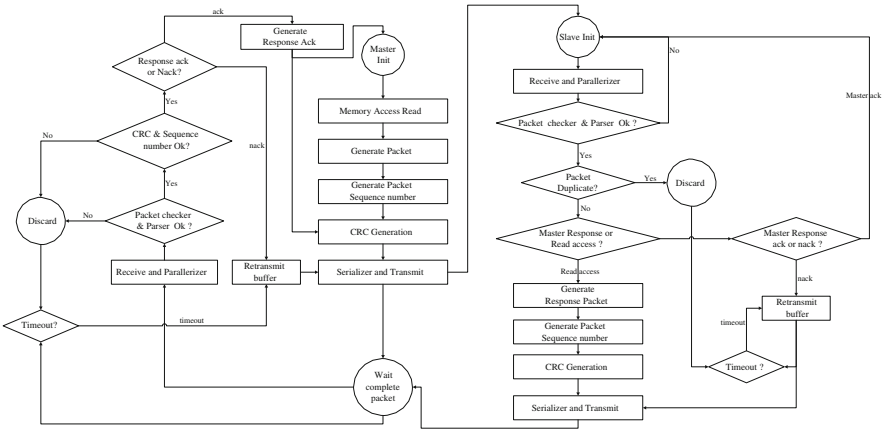

Fig. 4. Read access flow charts

The packet checker in the master protocol engine will check the sequence number and the CRC after receiving the read response packet. If there is an overlapping, it is discarded. Otherwise, it will be stored at the data buffer. Memory read data are transmitted through the processor interface, and the ACK packet is transmitted by the master protocol engine that has received a read response packet, and the read access is then completed. If the master protocol engine reads the response packet to inform it that the read access packet has not been passed to the slave protocol engine, then the packet is retransmitted and the engine waits until it receives a response packet. If the received information of a normal response packet arrives later than expected, the master protocol engine will retransmit the read access packet.

4 Test for Read and Write Access

4.1 Test Setup

To test the feasibility of the protocol engine between the memory and processor, the processor was emulated on an Altera Stratix IV FPGA. The maximum operating speed of the emulated processor is 200 MHz, the width of the system bus is 32-bit, and the processor system may be configured with a DMA, LED, LCD, inner program memory, general I/O, JTAG UART, and the master controller (protocol engine); the memory system is made up of a memory controller, LCD, memory, slave controller (protocol engine), and so on.

The configuration of the test system is as shown in Fig. 5. We constructed a single FPGA system for easier testing on a single FPGA. That is, a processor system and memory system are on a single FPGA. They are connected by an optical line in 10 Gbps, but without an internal connection.

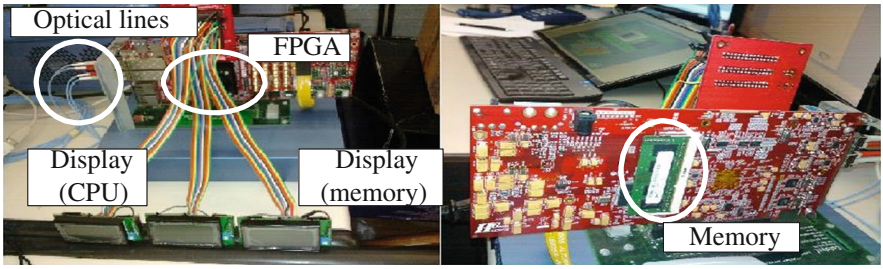

Fig. 5. Experimental setups.

We coded a C-code that drives the process, which is used to validate the communication between the processor and memory coupled to an optical system.

The C-code executes a basic operation of the reading and writing of memory, or represents an operation between the processor and memory using three character LCD, or includes an LED control code. Two LCD shows a basic command and the data transmitted by the memory. One LCD show the reading memory data before transmitting to the processor over an optical connection.

4.2 Experimental Results

As we can see in Fig. 6, the master protocol engine generates a request packet for the read access (1). The generated packet is transmitted to the slave protocol engine. The slave protocol engine that receives the read access request (2) delivers its data to the memory controller after analyzing the packet and reconfiguring the needed data for the memory controller (3). The memory controller executes a memory access using that data over a DFI interface. The data obtained by executing a memory read access is transmitted to the master protocol engine (4). The master protocol engine receives the read-ACK in the read access response packet, and the response-ACK corresponding for the read-ACK is then transmitted to the slave protocol engine (5). The memory data are passed to the processor through the processor system bus, and displayed through the LCD (6). If the slave protocol engine receives a response-ACK generated by the master protocol engine, the protocol engine completes the read access and then waits for the next access (7).

As we can see in Fig. 7, the master protocol engine generates a request packet for the write access (1). The packet generated is transmitted to the slave protocol engine. After analyzing the packet, the slave protocol engine, which receives the write access request, delivers a write-ACK packet to the master protocol engine to confirm whether a normal packet has been received (2). The slave protocol engine delivers its data to the memory controller after analyzing the packet and reconfiguring the needed data for the memory controller (3). In addition, the memory controller executes the memory access using the data over a DFI interface (4). If the master protocol engine receives a write-ACK generated by the slave protocol engine, the protocol engine completes the write access and waits for the next access (5).

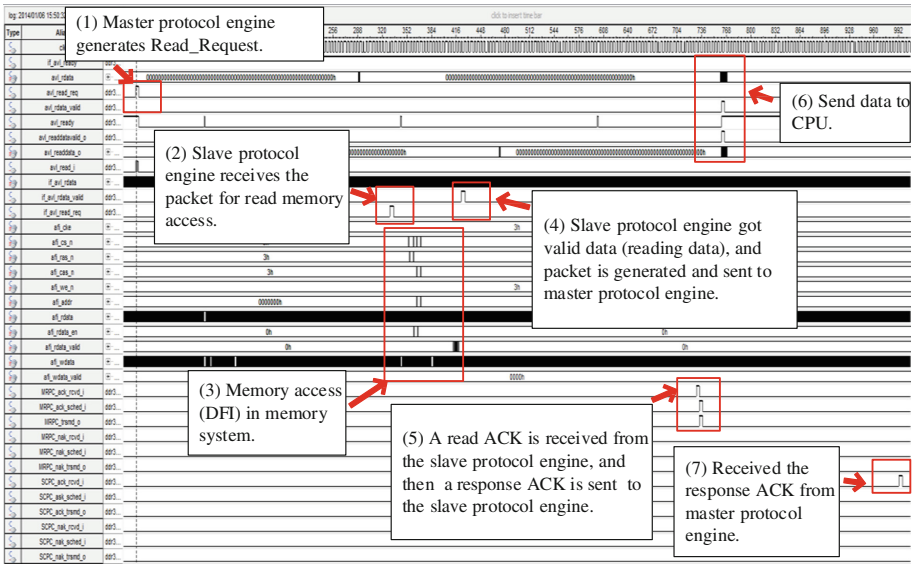

Fig. 6. Waveform of reading access captured by using SignalTap [18].

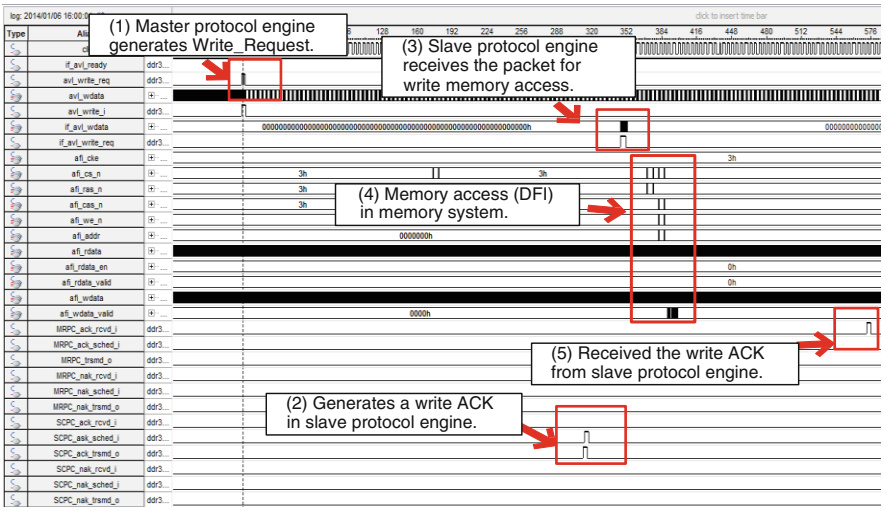

Fig. 7. Waveform of writing access captured by using SignalTap [18].

5 Conclusion

In this paper, we designed a protocol to test our proposed structure, and configured the optically connected processor and memory system. The processor system includes the master protocol engine and general processor system, and the memory system is composed of a slave protocol engine and a memory controller. We conducted an operation test of the protocol engine on each processor and memory system created within the FPGA for emulation. The operation of each system is independent of each other, and each system is supported by a 10 Gbps FPGA high-speed serial I/O connected optically.

The DIMM is configured with a memory channel in a conventional processor, which accounts for about 50 % of the number of pins in the processor. Through these proposals, if the protocol engine and memory controller can be mounted on an individual memory chip, the channel interface can be simplified. A built-in memory controller in a separate memory chip can access and control various memory, without the need to control a variety of complex timing information to the processor-memory interface, and the memory chip can control the timing internally. The change in the processor-memory interface has some advantages, i.e., a reflected signal and an inter-signal delay problem between the DIMM and a conventional memory controller may not occur.

This paper mainly described the functional aspects of the designed protocol, more experimental work is needed in the future. For this purpose, a system using a multiple memory configuration rather than a single memory configuration is necessary. In addition, we will build and test high-capacity services such as video-enabled protocols as a system for verification of the system performance.

Acknowledgement. This work was supported by the ICT R&D program of MSIP/IITP, Korea. [10038764, Silicon Nano Photonics Based Next Generation Computer Interface Platform Technology]

References

1. Li, L., et al.: Grid memory service architecture for high performance computing. In: Seventh International Conference on Grid and Cooperative Computing, pp. 24–26 (2008)
2. Okazaki, A., Katayama, Y.: Optical interconnect opportunities for future server memory systems. In: High Performance Computer Architecture, pp. 46–50 (2007)
3. Batten, C., et al.: Building many-core processor-to-DRAM networks with monolithic CMOS silicon photonics. In: HOTI 2008, pp. 21–30 (2009)
4. Yin, Y., et al.: Experimental demonstration of optical processor-memory interconnection. In: Advanced Intelligence and Awarenss Internet, pp. 23–25 (2010)
5. Hadke, A., et al.: Design and evaluation of an optical CPU-DRAM interconnect. In: ICCD, pp. 492–497 (2008)
6. Brunina, D., et al.: Building data centers with optically connected memory. J. Opt. Soc. Am. **3**(8), A40–A48 (2011)
7. H. M. C. Consortium: Hybrid Memory Cube Specification 1.0 (2013)
8. Jeddeloh, J., Keeth, B.: Hybrid memory cube new DRAM architecture increases density and performance. In: Symposium on VLSI Technology Digest of Technical Papers, pp. 87–88 (2012)
9. JEDEC: Wide I/O Single Data Rate (2011)
10. Kang, U.: 8 Gb 3D DDR3 DRAM using through-silicon-via technology. In: ISSCC 2009, pp. 130–131, 131a (2009)
11. Lee, W.S.: A Study on the effectiveness of underfill in the high bandwidth memory with TSV. In: IMAPS (2013)
12. Beamer, S., et al.: Re-architecturing DRAM memory systems with monolothically integrated silicon photonics. In: Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA 2010, pp. 129–140 (2010)
13. Udipi, A.N., et al.: Combining memory and a controller with photonics through 3D- stacking to enable scalable and energy-efficient systems. In: ISCA 2011 (2011)
14. JEDEC: DDR4 SDRAM JESD79-4, September 2012
15. Jun, H.T., et al.: Disintegrated control for energy-efficient and heterogeneous memory systems. In: HSPA2013, pp. 424–435 (2013)
16. http://www.bit-tech.net/hardware/cpus/2010/04/21/intel-sandy-bridge-details-of-the-next-gen/1
17. Brunina, D.: Building data centers with optically connected memory. IEEE/OSA J. Opt. Commun. Networking **3**(8), A40–A48 (2011)
18. http://www.altera.com/literature/hb/qts/qts_qii53009.pdf