The Design and Implementation of AMBA Interfaced High-Performance SDRAM Controller for HDTV SoC

Pan Guangrong[1], Feng Da[2], Wang Qin[1], Qi Yue[1], Yu Meiqiang[1]
[1]*University of Science and Technology Beijing, Beijing, 100083, China*
pennypan.libra@gmail.com, wangqin@ies.ustb.edu.cn, qiy@m165.com, myairare@yahoo.com.cn
[2]*China Light Industrial Corp. for Foreign Economic and Technical Cooperation, Beijing, 100011, China*
fengda1627@hotmail.com

Abstract

For cost reasons, the usage of SDRAM is preferred in HDTV SoC. However, accessing SDRAM is a complex task, especially when the same SDRAM is shared by various functional modules with different bandwidth requirements and requirements for responding speeds. For two-way cable networked HDTV SoC especially when the network data throughput is very high, the performance of SDRAM controller is very important. This paper describes a high-performance AMBA interfaced SDRAM controller design that exploits SDRAM features and uses popular IC designing techniques such as the buffering technology, ping-ponging between buffers and adopts bank-closing control as well. Simulation results under a realistic application demonstrate a significant decrease of total execution time of the program used in our experiments. The SDRAM controller IP is suitable for FPGA implementation and is flexible enough to be used in the application of two-way cable networked HDTV SoC.

1. Related work and introduction

Dynamic RAM memories are important components in embedded systems. They have been used in embedded processors such as Blackfin [1] and some specific applications including fingerprint recognition system [2], HDTV SoC and so on.

In order to enhance overall performance, SDRAMs offer features including multiple internal banks, burst mode access, and pipelining of operation executions [3]. Accessing one bank while precharging or refreshing other banks is enabled by the feature of multiple internal banks. By using burst mode access in a memory row, current SDRAM architectures can reduce the overhead due to access latency. The pipelining feature permits the controller to send commands to other banks while data is delivered to or from the currently active bank, so that idle time during access latency can be eliminated. This technique is also called interleaving.

Researches on SDRAM controllers have been trying to deploy the features that SDRAMs can offer. The interleaving technique and pipelining feature have been respectively exploited in a memory controller of a commercial HDTV mixer application [4] and in a SDRAM controller of a HDTV video decoder [5]. Paper [6] added arbitration mechanism to the SDRAM controller and used full page mode to finish the access requirements. However, due to the complexity and the cost of IC implementation, SDRAM features are hard to be applied to a system all together. The most popular groups researching on SDRAM controllers should be the numerous IP (Intellectual Property) suppliers such as XILINX, ALTERA, Lattice Semiconductor Corporation etc. Using IP cores can significantly shorten the development time. However, due to cost issues, it is not always the best way to buy an IP core from a supplier.

In our project of two-way cable networked HDTV SoC, due to cost issue of the IP warrant, we decided to design our own SDRAM controller IP core at the same time taking advantages of SDRAM features and some IC design techniques. In Section 2, we describe our design and its implementation based on a FPGA platform. The SDRAM controller architecture is given and some techniques we used are described in details. In section 3, the testing results from the application

978-0-7695-3507-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CSIE.2009.47

program running in the final two-way cable networked HDTV SoC are shown, and the typical timing diagrams are given as well. Section 4 concludes the paper.

2. Design and implementation of SDRAM controller

In HDTV SoC, there are usually multiple modules which need to access memories off chip [7]. In our two-way cable networked SoC, they are EuroDOCSIS protocol processor, Godson CPU, DMA and Ethernet MAC connected by AMBA (Advanced Microcontroller Bus Architecture) as shown in Figure 1. Each of them has their own bandwidth requirements and responding speed requirements for SDRAM. By analyzing the multiple accesses from the 4 modules and the SDRAM specifications such as its accessing delay, we take both side 1 and side 2 (shown in Figure 1) into consideration respectively. On side 1, we use bank closing control. On side 2, the controller employs two data write buffers to reduce the data access awaiting time, and uses 2 read buffers to decrease the CAS delay time when reading data from SDRAM. Due to the complexity of implementing the interleaving technique, we haven't introduced that technique to our design yet. However our design is proved to be functionally correct and high-performance in section 3.

According to the data sheet of general SDRAMs, a SDRAM must be initialized before starting to access it. In the last part of this section we also give a universal and configurable initialization scheme considering that the initializing process might have tiny differences between different SDRAMs from different corporations.

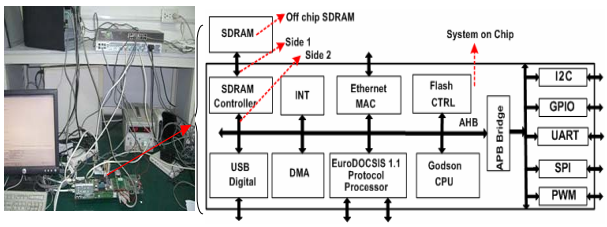

Figure 1. **Diagram of two-way cable networked HDTV SoC and the implemented platform in the lab**

2.1. Bank closing control

Only one open row in an active bank can be accessed. An ACTIVE (refer to [8] for details about SDRAM commands) command can open a row and make active the bank which the open row is in. A PRECHARGE command issued to a bank can set the bank to idle state, i.e. closing the open row in this bank. If every time after accessing a row AUTO-PRECHARGE command is performed, and the next access is actually involving the same row of the same bank, an ACTIVE command needs to be applied again in order to access last open row. As we all know, tRCD (time needed between ACTIVE and READ/WRITE commands) has to be fulfilled. According to the local principles of programs, in a successive period of time a program probably only accesses a small part of continuous address space. So that it is not necessary to issue an AUTO-PRECHARGE command after every read/write access without judging if there's any need. We use Bank Status (refer to Figure 2) to record bank state (active or idle) and pre-accessing row address of each bank. If the current access is for an open row in an active bank, the READ or WRITE command is directly issued to off chip SDRAM, and if it is for an idle bank, ACTIVE and READ or WRITE commands in order. When the auto-refresh (controlled by the Auto-refresh Control in Figure 2) is required, all the active banks will be inactivated by applying PRECHARGE ALL command as auto-refresh can only be issued when the whole SDRAM is in idle state. In this way, the overhead caused by frequently opening and closing the SDRAM banks can be decreased.

Figure 2 is the whole architecture for our SDRAM controller. Only the data path relating to read and write buffers, AMBA interface and off chip SDRAM is shown.

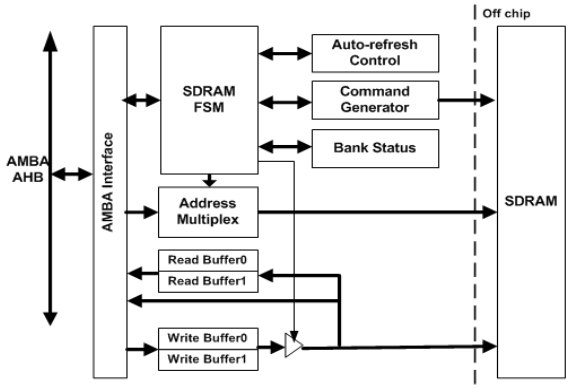

Figure 2. **Big picture of the SDRAM controller**

2.2. Read buffering

An AHB (Advanced High-performance Bus) transfer consists of two distinct sections, the address phase and the data phase [9]. The address phase lasts only a single cycle. The address and some control signals are transferred from a master to a slave during this phase. During the data phase, data and responding signals are delivered. If a slave can't finish the data phase in one cycle, it can use HREADY to make the

master hold the address and control signals. During the data phase, slave will send the corresponding response signals to the master to notify if the transfer is successfully finished or if it needs to retry the transfer. As we all know, SDRAM can not finish the data access in a single cycle. So we need some strategies to decrease the responding time.

We analyzed our application and found that if the SDRAM is accessed in the single mode (each time only a beat can be accessed), the needing data is available at least after 2 cycles (CAS latency). Plus time consumed by the latching in the bus interface part and the PRECHARGE command, in fact, it is more than 2 cycles. To support the fast response time, burst mode access and read buffering techniques need to be used whenever possible.

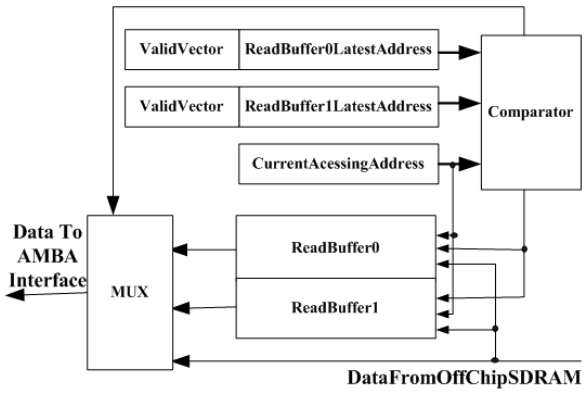

Figure 3. **Read buffering**

Figure 3 shows the read buffering scheme. When AHB bus needs to read data, see if data is already in read buffer by checking if the current accessing address is in the range of either of the read buffers and if the corresponding bit in ValidVector is valid. ValidVector is used to mark if the data stored in data buffer is valid. ValidVector works like tags for a cache. Every read buffer is only as big as the capacity of a SDRAM burst, so that ValidVector is only a several-bit vector. If the needed data is in read buffers, data can be directly read from them, otherwise, a READ command is needed issuing to off chip SDRAM. After a preset number of clock cycles, the data is available on the output latches of the SDRAM for reading, and data is delivered to AHB bus and written to one of the read buffers at the same time. The whole burst access data will be loaded in read buffer. In this way, we implemented prefetching. According to the local principles of programs, the next read access is possibly a successive address to the current one. So the next data can be read from read buffers, which can fasten the responding speed. In order to reduce the complexity of implementing read buffering, data from SDRAM are stored in read buffer0 and read buffer 1 in turn.

2.3. Write buffering and ping-ponging

As it is described in section2.2 about the AHB, in order to reduce the responding time to write access, write buffers are used to pack and align the data when the AHB bus data transfer size does not match the SDRAM data bus width. Based on our previous research on FPGA designs, we didn't use only one write buffer, instead, we use 2 buffers. See Figure 2. It is well known that ping-ponging can reduce or eliminate the mismatch effects between 2 different modules which are operating at different speeds. By utilizing ping-ponging between the two write buffers, part of the time writing one buffer and part of the time moving data from another buffer to off chip SDRAM can be overlapped. If write buffer0 is not empty and write buffer1 is being written by AHB bus, move buffer0's data to off chip SDRAM and this buffer will be in the progress of moving until all data is moved to SDRAM, vice versa. This is the ping-ponging technique we used between write buffer0 and write buffer1.

The algorithm of moving data from buffers to off chip SDRAM is as follows:

If write buffer0 is being written by AHB bus and write buffer1 is not empty, move buffer1's data to off chip SDRAM;

Else if write buffer1 is in the progress of AHB writing and write buffer0 is not empty, move buffer0's data to off chip SDRAM.

This algorithm explains how the ping-ponging can make the time writing buffer and writing SDRAM overlap. The description of operations about AHB bus writing buffers is as follows:

If write buffer0 is empty, data will be written to write buffer0;

Else if write buffer0 is not empty or in the progress of moving its data to off chip SDRAM and write buffer1 is empty, data will be written to write buffer1;

Else make the AHB bus hold the bus signals until one of the buffers is empty.

In order to keep the data consistency among the read buffers, write buffers and the off chip SDRAM, whenever the data is written to write buffers, match the write address with the addresses of data in the read buffers. If the address matches, data will be written to read buffers at the same time it is written to write buffers. When reading SDRAM, match the read address with the read buffer address and the write buffer address first to see if the data can be read from those buffers.

2.4. The initialization scheme

SDRAMs must be powered up and initialized in a predefined manner. The general initializing process includes maintaining a period of time before issuing any commands to the SDRAM after the clock is stable, applying a PRECHARGE ALL command afterwards, and then performing several times of AUTO-REFRESH commands, at last issuing a LOAD MODE REGISTER command. Different SDRAMs from different corporations need different periods of maintaining time and different auto-refresh times. For example, a kind of SDRAM from MICRON needs 100us [8] after powering up, while a SDRAM from HYNIX requires 200us [10]. After precharging all banks in the progress of initialization, auto-refresh command needs to be applied 2 times for a MICRON SDRAM MT48LC8M16A2 [8] and 8 times for HY57V561620C(L)T(P)[10].

We apply a configurable startup register and a configurable auto-refreshing counting register to our design. The startup register is used to count for the maintaining time after powering up. The auto-refreshing counting register is used to store and count the auto-refresh times needed during initialization. It is also used to record the auto-refresh requirement times generated by the sub-module auto-refresh control which takes charge of maintaining the accuracy of the data stored in the off chip SDRAM.

3. Simulation results and analysis

The SDRAM controller was implemented in Verilog at the RT-Level. The typical read and write timing diagrams are given in Figure 4 and 5 which were picked from the simulation results by ModelSim.

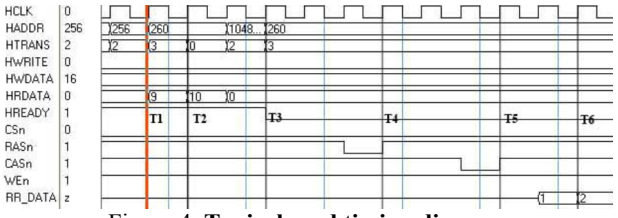

Figure 4. **Typical read timing diagram**

Figure 4 is the typical timing diagram for reading. At T1 SDRAM controller can see the read request from AHB bus, and the responding data can be available between T1 and T2 (i.e. the data phase of AHB specifications) because the required data is in read buffer, which is the highest speed at which a SDRAM controller can respond to a read command from AHB bus. At T3, controller sees the reading demand and judges that the requiring data is not in read buffer, so the signal HREADY is pulled down to make the master hold the bus signals. At T4, the off chip SDRAM sees the ACT command. Then after tRCD, SDRAM can see the read command at T5. After a delay of CAS latency, the SDRAM places the first data on the RR_DATA bus between T5 and T6.

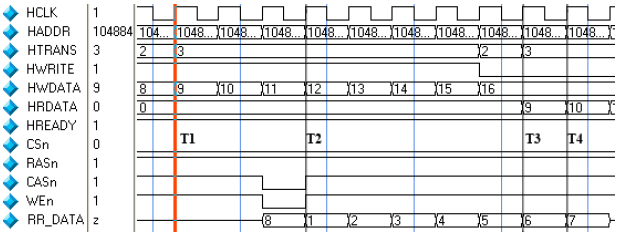

Figure 5. **Typical write timing diagram**

Figure 5 is the timing diagram for writing a burst of eight words to SDRAM. Because one of the write buffers is empty, the controller can directly write the data into the write buffer successively without extending the data phase by using HREADY. It is also can be seen that, at T2, off chip SDRAM can see the write command and the first data from the other write buffer. That's how the time responding AHB write transfer and the time moving data from the other buffer to SDRAM overlaps, which illustrates the benefits produced by deploying ping-pong technique. At T3, AHB wants to read a data from the address that has just been written to SDRAM. So we can see that data becomes available at the data phase between T3 and T4 without any delay, because the data is directly read from write buffer.

After implementation of the whole SoC for a Xilinx Virtex2P FPGA (refer to Fig.1), the SDRAM software test programs are executed to verify the accuracy of the SDRAM controller. We run a program that fully writes the off-chip SDRAM, and then reads all the data out from it. The column Bandwidth in Figure 6 shows the comparison of bandwidth which is defined to be all data bits divided by the total time that is used to transfer those data bits. We can see that our SDRAM controller provides a large bandwidth. After the verification, we compared the execution time of running a HDTV application program using our SDRAM controller to that using the traditional one which is functionally similar to the descriptions of XAPP134, an IP application note for an IP from Xilinx. Note that all the results in Figure 6 are normalized according to those from the SDRAM controller described in this paper.

The frequency of the whole SoC got from ISE is 190MHZ for the SoC used the traditional SDRAM controller and 182MHZ for that used the SDRAM

controller we described in this paper. By translating into time the cycles that running the application program used, we compared the execution time of the application program used those 2 different SDRAM controllers. Referring to Figure 6, we can see that our SDRAM controller has made a significant decrease of the total execution time of the application program. The speedup rate is 2.6.

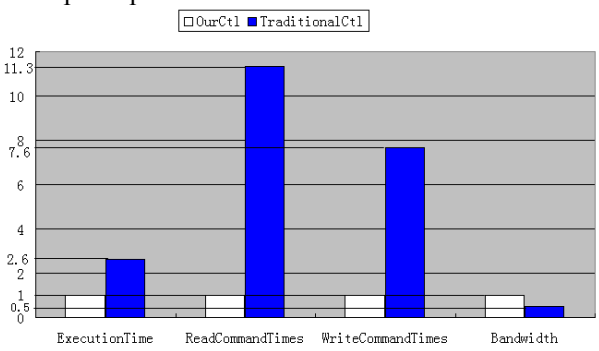

Figure 6. **Testing results between our controller and the traditional controller**

Moreover, for the 2 SDRAM controllers, we compared the times of issuing READ and WRITE commands to off chip SDRAM while running the application program. It is shown in Figure 6 that, running the same application program, our SDRAM controller can extraordinarily reduce the times of accessing off chip SDRAM, which has a heuristic meaning for low power designs as well.

4. Conclusion

An AMBA interfaced high-performance SDRAM Controller has been proposed, implemented, verified and evaluated in two-way cable networked HDTV SoC implemented in FPGA. The SoC environment consists of all the modules that listed in Figure1. With reasonable and detailed evaluation, we find this SDRAM controller has high performance by taking good use of the features of SDRAM architecture and utilizing the well-known techniques such as data buffering and ping-ponging. When running a typical HDTV application program, the testing result shows a speedup up to 2.6. The IP can easily and simply changed to be adapted to other systems.

5. References

[1] AD. Blackfin, "Embedded Processor Data Sheet", http://www.analog.com/static/imported-files/data-sheets/.

[2] Xin Zhang, Jian Tang, and Dawei Zhang, "Application of SDRAM in Fingerprint Recognition System", *Journal of Shenyang Institute of Technology*, Shenyang, China, Vol. 23, 2004, PP. 9-11.

[3] Betty Prince, *High Performance Memories: New Architecture DRAMs and SRAMs Evolution and Function,* John Wiley & Sons, New York, NY, USA, 1999.

[4] Kersten Henriss, Rolf Ernst, and Peter Rüffer, "Arrangement for Processing Digital Video Signals in Realtime", International Patent #WO 01/80549 A1, Oct. 2001.

[5] Qiang Zhao, Rong Luo, Hui Wang, and Yang Hua-zhong, "High Performance SDRAM Controller Design for HDTV Video Decoder", *Journal of Electronics & Information Technology*, Beijing, Jun. 2007, PP. 1332-1337.

[6] Shen Dong, Wang Feng, and Yu Song-yu, "SDRAM Controller Design in HDTV SOC Project", *Microcomputer Information*, Beijing, Vol. 22, No. 5-2, 2006, PP. 110-112.

[7] Yamauchi H., Okada S., Taketa K., Mihara Y., and Harada Y., "Single Chip Video Processor for Digital HDTV", *IEEE Trans. on Consumer Electronics*, IEEE-Inst Electrical Electronics Engineers INC, NY, USA, Aug. 2001, PP. 394-404.

[8] Micron Technology Inc., Synchronous DRAM (MT48LC128Mb) Data Sheet, 2001.

[9] ARM, AMBA Specification Rev.2.0, 1999.

[10] Hynix Semiconductor Inc., SDRAM Device operation Rev.1.1, Sep. 2003.