LETTER

Implementation of the Complete Predictor for DDR3 SDRAM

Vladimir V. STANKOVIC[†a)], ***Member***, **Nebojsa Z. MILENKOVIC**[†], ***and*** **Oliver M. VOJINOVIC**[†], ***Nonmembers***

SUMMARY In the arsenal of resources for improving computer memory system performance, predictors have gained an increasing role in the past few years. They can suppress the latencies when accessing cache or main memory. In our previous work we proposed predictors that not only close the opened DRAM row but also predict the next row to be opened, hence the name 'Complete Predictor'. It requires less than 10 kB of SRAM for a 2 GB SDRAM system. In this paper we evaluate how much additional hardware is needed and whether the activations of the predictors will slow down the DRAM controller.

key words: *DDR3 SDRAM, latency, predictor*

1. Introduction

The Complete Predictor [1] consists of two predictors: a close page predictor and an open page predictor. First the close page predictor predicts whether to close the currently opened DRAM row or to keep it open. In the case of a row closing, the open page predictor predicts the next row to be opened. The mentioned close page predictor also consists of two predictors: a zero live time predictor and a dead time predictor. The first one is always used when a new row is opened, and it predicts whether the row's live time will be a zero live time or not. If yes, that row is closed immediately after completing the DRAM access. If not, the row is kept open and after that access and during further accesses the dead time predictor is used to predict whether that row has entered its dead time. If it has, the row is closed. In case of a prediction that closes the row (either by the zero live time or by the dead time predictor) the open page predictor is activated. It predicts the next row to be opened, so the DRAM controller may open this row in advance.

The goal when using predictors is to achieve a hybrid policy that yields lower DRAM latency than both the Open Row Policy and the Close Row Autoprecharge Policy. Since we have already achieved this [1], we wanted to estimate the quantity of hardware needed to implement our predictors. We were mainly interested in seeing what percentage would be spent on SRAM memory and what on all the other hardware elements. Namely, we know that the amount of the needed SRAM is small. But what about the rest of the components? With a variety of additional hardware elements, they may occupy a significant part of the total hardware requirements. Therefore, we gave ourselves the task of estimating the amount of hardware needed to implement the rest of the components, compared to the needed amount of SRAM. We decided to implement the Complete Predictor on an FPGA chip and see the number of equivalent gates we will gain. That way we will be able to directly compare the SRAM part and all the other components, since the number of equivalent gates for the SRAM memory is practically known up front, it depends only on the SRAM amount. The number of equivalent gates (or equivalent gate count) is a measure of logic capacity, where a gate is ostensibly a 2-input NAND [5]. Another important thing we wanted to see is whether the activations of the predictors will slow down the DRAM controller and by how much. If they slow it down too much, then using the predictors is pointless. Knowing these two things will help us judge if similar predictors may be implemented as parts of modern DRAM controllers, i.e. modern processors (in contemporary processors the DRAM controller is part of the processor). Using the software packet Xilinx ISE WebPack v.8.2, we have implemented the best variants of the zero live time, dead time and open page predictor from [1] on FPGA chip, Xilinx SpartanII family, model xc2v500-6fg256, and we show these results in Sects. 2, 3 and 4, respectively. Section 5 contains results of the implementation of the sync algorithm of the predictors and Sect. 6 contains the complete results. Section 7 is the conclusion.

2. Zero Live Time Predictor

2.1 Behavior

It is easy to implement the zero live time predictor in a form of SRAM memory. Namely, the best variant of the zero live time predictor from [1], signed as *2b16*, has two bits for every 16 adjacent DRAM rows in the system, used as a saturated counter, with values from 0 to 3. For the adopted 2 GB DDR3 SDRAM structure from [1] with total of 16 banks in the system and 8 k rows per bank, the total number of rows in the system is 128 k. For that number of rows we need 16 kb or 2 kB of SRAM memory.

2.2 Hardware Cost Estimation

The implementation of the zero live time predictor demands 65,595 equivalent gates. Only 59 gates of those are spent on implementation of the control block while the rest 65,536 gates are spent on the 2 kB SRAM. This means that 4 gates

Manuscript received September 24, 2013.
Manuscript revised November 19, 2013.

[†]The authors are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia.
a) E-mail: vladimir.stankovic@elfak.ni.ac.rs

DOI: 10.1587/transinf.E97.D.589

are spent for each bit of the SRAM, so we could even calculate this number in advance. Only 4 4-input LUTs were used for this predictor.

Another important thing considering the implementation are the time intervals needed to execute the two basic operations of the predictors: Lookup and Update. It is important that the time parameters are such that the execution of these operations does not slow down the DRAM controller, or at least not too much. Obviously, Lookup is performed when the controller calls a predictor for its prediction, and Update is performed when important cases occur, that demand the predictor to update the data upon which the predictions are performed. In case of the zero live time predictor, the needed times for executing these two operations are 1 clock cycle for Lookup and 2 clock cycles for Update.

3. Dead Time Predictor

3.1 Behavior

The dead time predictor is based on access interval time values. Simulation results show that the average dead time is much larger than the average access interval time, hence when a value that is the last access interval time, multiplied by 2 or 4, elapses, it is predicted that the row has entered its dead time. To implement the dead time predictor the DRAM controller needs several things:

1. There has to be one counter for each bank to take care of the elapsed time since last access. In order to minimize the counters' length, they may be triggered with a signal derived by dividing the DRAM's clock.
2. A register for each bank is needed for storing the last access interval value. Every time there is an open page hit in any of the banks that means occurrence of a new access interval time. In that case the counter's value for that bank should be stored into the proper register and the counter should be reset. The operation of multiplying by 2 or 4 could be implemented by fixing the least significant bit to zero (multiply by 2) or the two least significant bits to zeros (multiply by 4) and then storing into this register from the next position(s).
3. The DRAM controller also needs one comparator for each bank. In case the counter's value is greater or equal to the register's value the row is to be closed, so the DRAM controller should issue a precharge command.
4. We also need two queues: *QueueA* and *QueueB*. The first is a queue with orders from the processor or the cache memory (this queue already resides within standard DRAM controllers) and the second is a queue with orders for bank precharges from the dead time predictor. When serving the queues *QueueA* is always of higher priority than *QueueB*.

3.2 Hardware Cost Estimation

Although this predictor seems rather complex, with lots of different hardware elements, the results show that the implementation of the best dead time predictor from [1], signed as *sep2*, demands only 17,718 equivalent gates. The number of 4-input LUTs used for this predictor is 1,404. As for the time parameters of this predictor, they are not important since it works in parallel with the DRAM controller, meaning that the DRAM controller does not have to wait for the predictor's response. The Lookup command does not exist as a classic command, since it is not the DRAM controller's job to check if there is a prediction - the predictor itself, if it predicts that some row has entered its dead time, executes the write of the proper bank id into *QueueB*. The DRAM controller's job is to check if there is something written into *QueueB* every time it is idle. If there is, the controller performs a read operation and issues a row precharge of the proper bank. The Update command does exist as a command that the DRAM controller dictates to the dead time predictor. Its execution time depends on the moment it appears and may last from 1 to 16 clock cycles. However, this is not important since the DRAM controller does not have to wait for this command to finish. It issues the Update command whenever it is needed (at each new access to a bank) and continues its job while the dead time predictor executes the command (writes the content of the proper counter into the proper register and resets the counter).

4. Open Page Predictor

4.1 Behavior

The open page predictor consists of two tables: Row History Table (RHT) and Pattern History Table (PHT). RHT stores the last k rows that were activated in each of the banks, so there are k fields in an item for each of the banks. PHT contains the predictions. It has $m \leq n$ items, where n is number of bank rows. Each item contains j two-part fields: row and next predicted row. PHT access index is obtained as t least significant bits of the truncated addition of the last k row indices from the proper item for that bank in RHT, so $m = 2^t$. If $j = 2$ and $k = 4$, for the adopted DDR3 SDRAM size and structure from [1] we need 832 bits for RHT and 6.5 kB for PHT, for the variant *m1k*.

Let us show the details of the implementation of the open page predictor for a DRAM system organized as 2 ranks with 8 DDR3 SDRAM 1Gb chips per rank, 8 banks per chip and 8 k rows per bank. Each row has 2 k columns. Figure 1 shows the implementation of the open page predictor. The RHT table with 2 x 8 = 16 items is implemented as *MRHT* memory, organized as 4 parallel memories *MRHT(0)* to *MRHT(3)* of the type 16 x 13 bits. Here a single RHT item with addresses of the 4 last open rows in a proper bank is stored into 4 locations in these 4 parallel memories with same relative addresses, which enables simultaneous read of all 4 rows from the given item of the RHT table. In that way the addition of the 4 rows can be executed with 2 of them happening simultaneously, thus shortening the total addition time. Since this addition is used to form an address for accessing the PHT table, which has 1 k items, the adders are 10 bits wide (the 10 least significant bits from the 13 bits of the

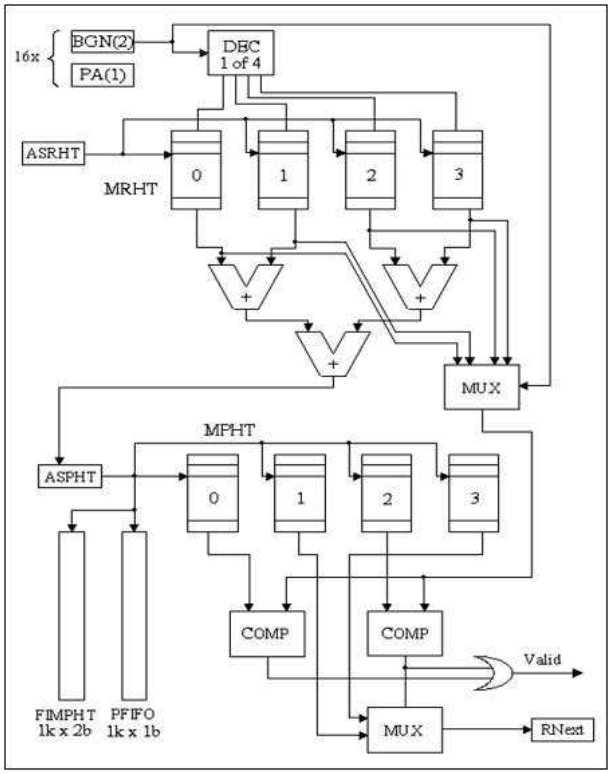

Fig. 1 Implementation of the open page predictor.

RHT items are used), with ignoring the carry. *ASRHT* is a 4b register which contains the address for accessing the items in *MRHT(0)* to *MRHT(3)* with same relative addresses. *BGN* is a 2b circle queue used for updating the RHT table. There are 16 of them, each for a proper item in RHT. The starting value for *BGN* is zero (00) and each time a new row is to be written into RHT *BGN* is incremented. If the value of *BGN* is 3 (11) its next value after the increment will be zero again. That way we can remember the last accessed row in a proper bank without having to shift the values from each *MRHT* to the next. *PA* is a 1b indicator which shows whether the prediction for the proper bank is allowed. At the beginning the prediction is not allowed. The moment all 4 rows are filled in a proper RHT item, its *PA* indicator is set and it remains set.

The PHT table is organized as 4 parallel memories *MPHT(0)* to *MPHT(3)*, each organized as 1 k x 13 bits. Each PHT item is stored in 4 locations in *MPHT* with same relative addresses, which enables simultaneous read operations. These 4 locations contain the two pairs *row - next row*. *ASPHT* is a 10 bit register used for addressing the four locations in *MPHT(0)* to *MPHT(3)* with same relative addresses. When accessing *MPHT*, 2 additional bits are needed to determine which of the 4 locations with same relative addresses in *MPHT* are to be accessed. These 2 bits are determined with the help of *FIMPHT* and *PFIFO*. *FIMPHT*, organized as 1 k x 2 b has 2 bits for each PHT item. These 2 bits show whether the two pairs in an item are filled (the 1st bit points to the 1st pair and the 2nd bit points to the 2nd pair). In case both pairs are filled and a new pair is to be written, *PFIFO*, organized as 1 k x 1 b, defines which of the two pairs will be removed. When this happens the proper *PFIFO* bit is complemented.

4.2 Hardware Cost Estimation

Implementation of the open page predictor demands 402,536 equivalent gates. This value is more than 40% greater than the realistic value. Namely, the used FPGA chip has block RAM elements with fixed sizes of 16 kb, so these blocks were used for FIMPHT and PFIFO, meaning there was unused space of 14 kb for FIMPHT and 15 kb for PFIFO. That gives 29 kb of excess memory space which corresponds to 118,784 equivalent gates. We may conclude that if there were block RAM elements with proper sizes of 1 kb, the open page predictor would probably demand 283,752 gates. It is important here to emphasize that the MPHT memory itself demands 262,144 gates, while 140,392 gates were spent on all the other components and with optimal FIMPHT and PFIFO implementation only 21,608 gates would be spent on all the other components. If we subtract the number of gates needed for RHT, FIMPHT and PFIFO from the last two values, we can get that only 5,992 gates are needed for all the components other than SRAM. The implementation of the MPHT memory of the type 4 k x 13 b demands 212,992 gates and a greater value (262,144) was gained since it was actually implemented as 4 k x 16 b, i.e. using 4 block RAM elements. This predictor used 307 4-input LUTs. As for the time parameters, Lookup executes for 3 clock cycles, while the execution of Update may vary, and its maximal value is 7 clock cycles.

5. Synchronization of the Predictors

In order for the predictors to work as a team a sync algorithm is needed, to activate the proper predictor when needed. We defined and implemented this algorithm as a finite state machine and its implementation demands 11,456 equivalent gates, with 904 used 4-input LUTs. These values include all the needed registers, indicators, additional logic etc. for controlling the execution of the predictors.

6. Implementation of the Complete Predictor

Table 1 contains data about the Complete Predictor. This data, as well as the data shown in the previous sections, was gained using the software packet Xilinx ISE WebPack v.8.2, on FPGA chip, Xilinx SpartanII family, model xc2v500-6fg256. The total number of equivalent gates used for implementation of all the predictors as well as the sync algorithm amounts 497,305 gates. From those almost 500,000 gates, 462,080 gates are needed for the SRAM (for the zero live time predictor, the RHT and PHT tables of the open page predictor, as well as the FIMPHT and PFIFO memories), and 35,225 gates (or 7.1%) are spent for all the rest.

Table 1 Equivalent number of gates for the Complete Predictor.

Total	SRAM	All the rest
497,305 (329,369)	462,080 (294,144)	35,225

With better organization of the FIMPHT and PFIFO memories, and implementation of the PHT table as 4 k x 13 b, only 329,369 gates would be needed for the Complete Predictor - 294,144 gates for the SRAM and 35,225 gates (or 10.7%) for all the rest. This means that, even though it might seem that the predictors are rather complex, the majority of the hardware, almost 90%, would be spent on the needed SRAM memory, whose total capacity is about 9 kB. Since contemporary processors contain MBs of SRAM (spent on cache memories) in a single chip [2], we may conclude that implementing the Complete Predictor into contemporary processors would not increase their complexity and price. This is true even for SDRAM memory systems with much larger capacities than the one we have dealt with in this paper. A few years ago, when we started to study predictors for SDRAMs, we decided to consider a 2 GB DDR3 SDRAM memory system, which was, should we say, a top choice at that time. In a few years from now some newer type of SDRAM memories (like DDR4 or DDR5) would probably become a standard for contemporary main memory systems, with capacities of probably 10 or maybe even 15 or 20 GB. What would change considering the implementation of our predictors? It is obvious that the main change would apply to the capacity of the needed SRAM memory, which, as we saw, occupies majority of the hardware. Let us be liberal in our evaluations and assume that the total capacity of the needed SRAM increased 20 times (although a 10 times increase for a 10 times larger SDRAM capacity of 20 GB would be logical). Even then we would still need less than 200 kB of SRAM memory, and there are already MBs of SRAM in a single processor chip. So not only there are no technological problems for implementation of our predictors in the future, but similar predictors might even become one of the better solutions considering the cost/performance pay off.

It is also important to emphasize that we proved that using the predictors would not increase at all the DRAM latency for executing the Lookup and Update operations. Namely, each of these two operations runs simultaneously with one of the 3 basic DRAM operations: row precharge, row activation or column access, and each of these 3 operations lasts more than 10 ns [3], [4]. This value is not smaller even for DDR4 SDRAM [4]. These 10 ns correspond to 10 clock cycles for SDRAMs that operate on 1 GHz. For higher frequency SDRAMs this value is even greater, and we saw that the longest Lookup/Update operation for whose execution the DRAM controller has to wait to finish lasts maximum 7 cycles. The value of 10 ns will probably not change much in the following years even for newer types of memories. Namely, DDR, DDR2, DDR3, etc. all have practically the same DRAM core, with very little difference in the latency parameters. What changes is the bandwidth - DDR2 has twice the bandwidth of DDR, DDR3 has twice the bandwidth of DDR2 [3] etc. On the other hand the latency changes much slower, the statistics show that the SDRAM latency improves only about 5% per year [2]. And even if, hypothetically speaking, the latency parameters became several times smaller in the future (which is very unlikely), the use of the predictors would still be justified and payable. This stands both from the fact that the other predictors' Lookup/Update operations last only 1-3 cycles, as well as from the significant latency decrease gained in [1] when using predictors.

7. Conclusion

In this paper we have shown the results of FPGA implementation of the Complete Predictor. The results show that about 90% of the number of equivalent gates would be spent on the needed SRAM memory for implementing the RHT and PHT tables and FIMPHT and PFIFO memories of the open page predictor as well as the memory of the zero live time predictor. All the rest is spent on the needed additional logic for the zero live time and open page predictors, the counters, registers and comparators plus additional logic for the dead time predictor, as well as the needed registers, indicators and additional logic for implementing the sync algorithm. If we wanted to roughly express the total hardware requirements only in amount of SRAM memory we could say that approximately the total hardware requirements for the Complete Predictor is only about 10 kB of SRAM memory for a 2 GB SDRAM memory system. This small amount of needed SRAM memory will remain relatively small even for much higher capacities of SDRAM in the memory system.

We have also shown that using the predictors will not increase the SDRAM latency for executing the predictors' basic operations Lookup and Update. Considering the trend of rather slow change in the SDRAM latency, it is expected that this will stand true even for newer types of SDRAM memories. The main conclusion we may derive is that the predictors like the Complete Predictor could be easily implemented in the near future and might become a successful weapon in the battle of decreasing the processor-memory speed gap.

References

[1] V. Stankovic and N. Milenkovic, "DDR3 SDRAM with a complete predictor," IEICE Trans. Inf. & Syst., vol.E93-D, no.9, pp.2635–2638, Sept. 2010.
[2] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 5th ed., Morgan Kaufman, 2011.
[3] B. Jacob, S. Ng, and D. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufman, 2007.
[4] Micron, "DDR4 SDRAM UDIMM," www.micron.com, Document name: atf16c1gx64az.pdf
[5] S. Brown, R. Francis, J. Rose, and Z. Vranesic, Field-Programmable Gate Arrays, Springer/Kluwer Academic Publishers, 1992.