An Intelligent Multi-Port Memory

Zuo Wang
School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China
wuchenjian.wang@gmail.com

Abstract*—It has become clear that on-chip storage is critical in large FPGAs. Scholars have done some researches on implementing user logic memory models with single-port and dual-port physical arrays. Their work is based on the assumption that user memory models are either single-port or dual-port. However, their works may not be helpful to multi-processor applications since single or dual-port arrays may not satisfy the simultaneous accesses from different processors. This paper proposes a novel multi-port memory design. In our design, distributed memory resources of LUT are mapped as 1-port memory banks. Data in different memory banks can be accessed simultaneously. With the help of port-priority and r/w-priority, our multi-port memory can resolve both write-write conflict and read-write conflict. When write-write conflict occurs, the port with the highest priority can execute its write operation. When read-write conflict occurs, either read-then-write or write-then-read type is selected according to r/w-priority. Besides, data-switch paths between ports are implemented by utilizing the read-write conflict. Extend Port Importance Hierarchy (EPIH) algorithm is proposed for basic conflict handling, while Block Access Control (BAC) algorithm is proposed for reducing conflict when processors carry block read/write. Experiment results on Xilinx Virtex-II show that: compared to implementation of N ports in each cell, our design saves 88% LUT resources. Experiment results on Xilinx Virtex-II also show that, as port number N increases, the cell cost increases significantly which restricts the reasonable port number to rather small values in practice. Experiment results on simulation show that BAC algorithm performs more efficiently according to increasing block read/write length.

***Index Terms*—FPGA, N-port memory, mapping, hierarchy, embedded arrays**

I. INTRODUCTION

The efficiency of shared memory is very important in applications like digital signal processing, multi media and communication systems [1] [2]. Multi-port memory is widely used as the shared memory in multi-processor systems since it can provide parallel access to the shared memory. There are many commercial ASIC multi-port products, such as IDT6116LA [3], CY7C130 [4], IDT70V5388 [5] and DT70V9279 [6]. Most of these produces do not consider the conflicts of accessing to the same data, while a few products only provide simple conflict mechanisms, such as busy control scheme [4]. In order to improve the efficiency of the shared memory, we need both effective and flexible mechanisms to manage conflicts. Thus, we propose an intelligent multi-port memory in this paper.

II. BACKGROUND

A. Conflict Control for Multi-port Memory

Single-port memory has only one data bus. If there are simultaneous accesses from different processors, the bus conflict may occur since each access requires occupying the data bus. There are two different solutions to resolve the bus conflict. One is time sharing, in which different processors compete for occupying the data bus in short time slices. The other option is to provide several data buses so that each port can have its own data bus to access the different addresses independently. However, if different ports access the same address at the same time, conflict problems still exist. For example, when two or more ports write the same address simultaneously, write-write conflict will cause cell data integrity problem [7]. When one port performs a write access while the other ports issue read accesses to the same address at the same time, read-write conflict will cause read data integrity problem. It is worth mentioning that two or more ports read the same address will not result in any conflict.

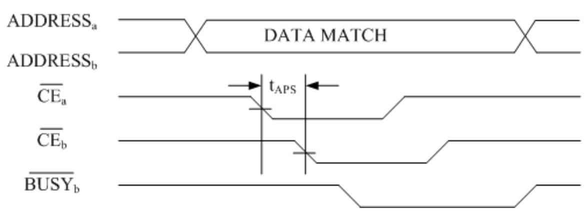

Figure 1. Note how the caption is centered in the column.

Busy control scheme, which servers the quest coming first, is always adopted to resolve conflict. Fig. 1 gives how busy control scheme works. When processors access the shared memory independently, there will be an access request queue with various intervals between consecutive requests. As shown in Fig. 1, if the chip enable signals ($\overline{CE}$) are spaced out by a time slice longer than “t_{APS}”, the request coming first will be served while the request coming later is marked as “busy”. However, when the intervals between consecutive requests are shorter than “t_{APS}”, the busy control scheme fails and the data integrity cannot be assured. Besides, busy control scheme is not flexible in some situations. For example, when processor A is doing an important job, its accesses to the shared memory should not be interrupted by other processors. Obviously, busy control scheme can not guarantee to provide the un-interrupted memory accesses in such situation. Un-interrupted memory accesses can also be implemented by software, such as semaphore, but we think hardware implementation will be more efficient.

B. N-port Mapping Technique for FPGAs

As FPGAs grow in logic capacity, they are being used to implement entire systems which require large amount of storage. Most commercial devices implement on-chip storage by providing several large arrays embedded into the FPGA, such as Embedded Array Blocks for FLEX 10K devices [8], Embedded System Blocks for Altera APEX 20K devices [9] and Block SelectRAM for Xilinx Virtex FPGAs [10]. Scholars have done some research on implementing user logic memory models with either single-port or dual-port physical arrays. Implementing user memories with physical embedded arrays is called logical-to-physical mapping [12]. With many variations in properties of user memories, the logical-to-physical mapping is a non-trivial task, especially for data intensive applications [13]. In [12] [14] [15], logical-to-physical mapping for FPGAs with single-port embedded arrays is discussed. In [16], optimizations are achieved by intelligently packing the single-port user memories into the dual-port physical arrays. In [17], a complete ILP formulation is proposed to achieve optimal design performance of a hierarchy of memories, constructed by on-chip memory and off-chip ram in reconfigurable computer. In [18], an ILP formulation of the global mapping process is described, which is simpler and faster than the complete ILP formulation introduced in [17]. All of their work is based on the assumption that user memory models are either single-port or dual-port. However, in some applications, user memory models are not limited to single-port and dual-port. So, we need an efficient way to implement N-port memory on FPGA considering most FPGA vendors do not provide N-port memory arrays (N≥3).

III. Multi-port Memory Design

A. FPGA Architecture

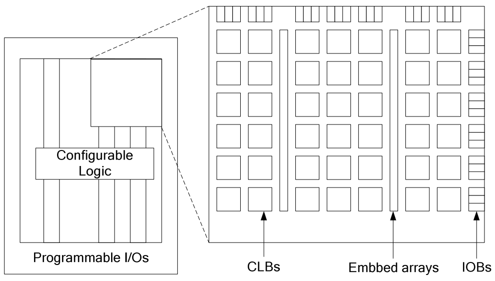

Figure 2. FPGA architecture overview.

FPGA is comprised of input/output blocks (IOBs) and internal configurable blocks (CLBs). Programmable I/O blocks provide the interface between package pins and the internal configurable logic. Fig. 2 shows the basic architecture of FPGA [11].

The internal configurable logic, associated with storage, includes two major elements organized in a regular array [11]. Configurable logic blocks (CLB) provide functional elements for combinatorial and synchronous logic while embedded arrays are used to implement large embedded storage blocks, in various depth and width configurations.

B. Hierarchical Architecture Based on FPGA

We propose port-priority and r/w-priority to solve both read-write and write-write conflict for multi-port memory. For write-write conflict, the port with the highest priority can execute its write operation while other requests fail. For read-write conflict, we provide two options to choose. One is to execute the read operation first and then write; the other option is to transfer data from the write port to the read ports directly, and then carry out write operation.

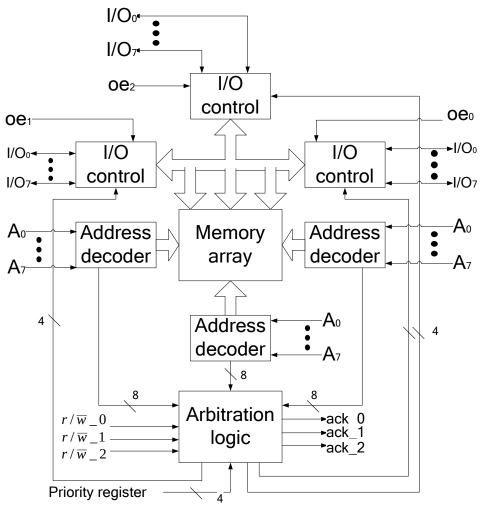

Figure 3. Memory architecture at top level.

Fig. 3 demonstrates the logical architecture of tri-port memory at top level. *Memory array* is the collection of memory cells. *I/O control* is the control logic used to manage the data transfer between the *memory array* and ports. *Arbitration logic* is the control logic used to detect and manage the conflict caused by accessing the same address simultaneously. *Priority registers* are used to configure both port-priority and r/w-priority, while the *ack* signals are used to indicate write failures.

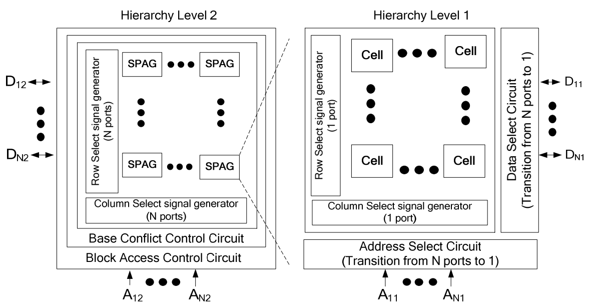
Hierarchy Level 2
Hierarchy Level 1
D12
DN2
SPAG
Row Select signal generator (N ports)
Column Select signal generator (N ports)
Base Conflict Control Circuit
Block Access Control Circuit
A12
AN2
Cell
Row Select signal generator (1 port)
Column Select signal generator (1 port)
Data Select Circuit (Transition from N ports to 1)
D11
DN1
Address Select Circuit (Transition from N ports to 1)
A11
AN1

Figure 4. Hierarchical architecture for achieving N-port memory with 1-port cells.

Based on FPGA, two levels of hierarchical architecture are exploited in Fig. 4. In general case there are: a) N ports; b) M_2 Smallest Parallel Access Granularity (SPAG) in level 2; c) M_1 memory cells in level 1; d) M capability, $M=M_1*M_2$, $M_1 = 2^{m1}$, $M_2 = 2^{m2}$; e) m-bit address signal, m=m1+m2. Hierarchy level 2 contains M_2 SPAGs, which appear to have N-port capability. Memory cells in different SPAGs can be accessed simultaneously, while accessing cells in the same SPAG at the same time will cause conflict. Base Conflict Control Circuit detects such conflicts by comparing address signals Ax2 and resolves them by an algorithm for port priority. SPAGs can be implemented by CLBs, which are distributing on FPGA chip symmetrically. The horizontal and vertical routing resources between CLBs can be used to implement address and data signal routing for N ports. Thus, implementing large number of SPAGs is practical. In [19], similar two-level architecture is discussed but is not interacted with FPGA yet.

In order to analyze the conflict probability, we use PL to describe the size of SPAG. PL=1 means implementing N ports in each memory cell. We use PR and PW to denote the read conflict probability and write conflict probability. They can be calculated as:

$$PR=PW=1-(P^{N}_{(M/PL)})/(M/PL)^{N} \tag{1}$$

The symbol P in (1) indicates the permutation in combinatorial mathematics. We should note that (1) is based on the assumption that each port continues to carry access action stochastically and each access action reads/writes a cell once. Fig. 5 gives the theoretical conflict probability calculated by (1). We can achieve very low conflict probability by keeping high M/PL value.

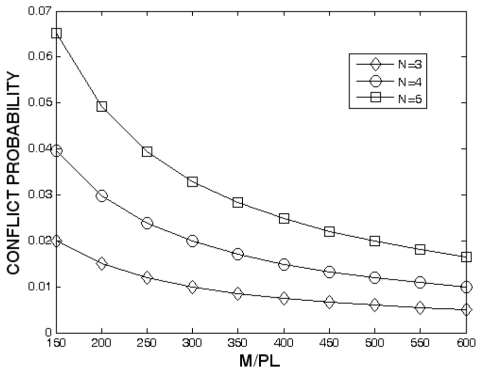

Figure 5. Conflict probability according to different M/PL.

C. Implementation

The logic design of cells is described in Fig. 6. Each cell consists of a group of bit registers which are used as data storage and two groups of tri-state buffers which are used to control the data transfer between bit registers and internal data bus. The *AS* signal, generated by *address decoder*, is used to select the memory cell we want to access. The $r/\overline{w}$ signal, generated by *arbitration logic*, is used to indicate the access type. It is worth mentioning that the $r/\overline{w}$ signal in Fig. 6 is different from that in Fig 3. In our design, the $r/\overline{w}$ signal only has "0" and "1" states to indicate read or write. However, when write-write conflict occurs, we need extra signals to indicate write failure state. In order to avoid using extra signals, *arbitration logic* automatically changes the failed write request into a read request. The data integrity problem caused by read-write conflict is not a real problem since the data fetched from the cell is not used by the write port.

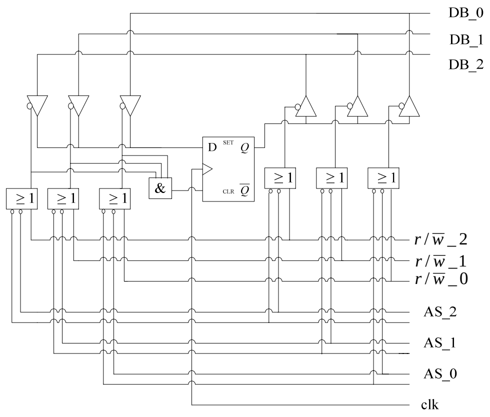

Figure 6. Memory cell architecture.

Fig. 7 gives the logical design of *I/O control*. We adopt two registers to lock the read data and write data for each port. In order to drive data flow to the external data bus in read access, we enable the tri-state buffers of read registers and disable the tri-state buffers of write registers. However, all the tri-state buffers' status is reversed to drive data flow from the external data bus in write access. Besides, *RWB* is used to transfer the data between read port and write port in read-write conflict. *r_en_x* and *w_en_x* control the data transfer between *RWB* and ports. If we choose write first in read-write conflict, the write port disables the *r_en_x* signal and enables the *w_en_x* signal to drive data flow to *RWB*; the read port disables the *w_en_x* signal and enables the *r_en_x* signal to drive data flow from *RWB* to the read port. We should note that, if write-write and read-write conflict occur at the same time, only the write port with the highest priority can transfer its data to *RWB*.

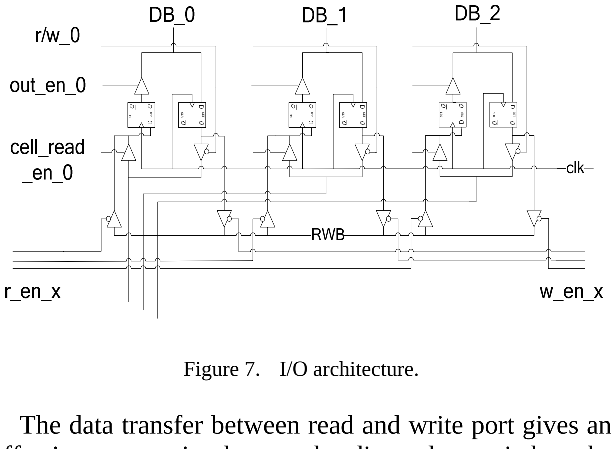

Figure 7. I/O architecture.

The data transfer between read and write port gives an effective way to implement the direct data-switch paths between processors. In common situations, if processor A wants to transfer data to processor B, processor A need first write data to the shared memory, and then tell processor B to fetch data no matter whether processor B is waiting for processor A's write operation. In our design, if we make processor A write the same address which processor B reads in write-then-read configuration, processor A will transfer data to processor B directly. Moreover, we can create complex conflicts to implement more data-switch paths. As shown in Fig 8, redundant data mirror is realized by configuring 1-write-3-read conflict.

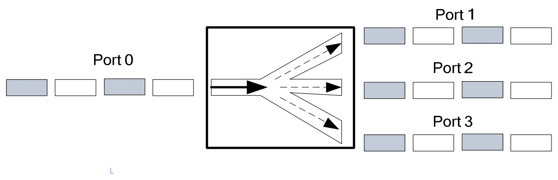

Figure 8. Redundant data mirror.

In our design, we adopt *port-priority register* to record the port-priority. The static port-priority is defined as P_0->P_1->…->P_N. For example, if port 0 and port 1 have write-write conflict, port 0 can finish its access while port 1 fails. Besides, we can change port-priority by updating *port-priority register*. Fig. 9 gives conflict control circuit in the case of 4 ports.

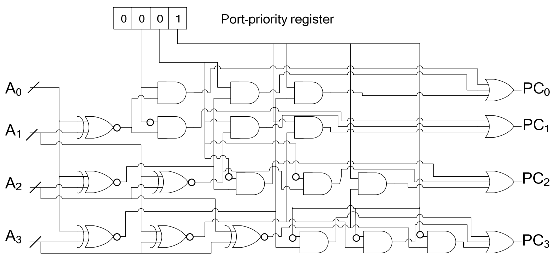

Figure 9. Conflict control circuit in the case of 4-port.

The circuit compares the address signals from each port to generate PC_i: "0" indicates no conflict while "1" indicates write failure. For example, the register value "0001" changes port-priority into P_3->P_0->P_1->P_2. As referred before, busy control scheme is not flexible in some situations. However, our design will not have such problems. For example, if processor A is ready to start an important job, we can change *port-priority register* to assign processor A the highest port priority. When processor A finishes its job and processor B is ready to start another important job, we can also change *port-priority register* to provide un-interrupted memory accesses for processor B.

Block Access Control (BAC) algorithm is proposed to reduce conflict when processors carry block read/write. We assume that all ports trend to access the same blocks. In the first cycle, Port 1 and port 2 access to $bank_j$ while others ports just wait. In the second cycle, while port 1 and port 2 move to access $bank_{j+1}$, port 3 and port 4 start to access to $bank_j$(other ports continue to wait). Port N needs to wait (n/2)-1 cycles before accessing to $bank_j$.

IV. Working Timing Graph

In write access, we lock the $r/\overline{w}$ signal, address and data at the first rising edge, and then write data to the memory cell at the next rising edge. In read access, we lock the $r/\overline{w}$ signal and address at the first rising edge, and then output data at the second rising edge. As shown in Fig. 10, the switching from read to write needs inserting a NOP cycle, while switching from write to read does not need insert any cycle.

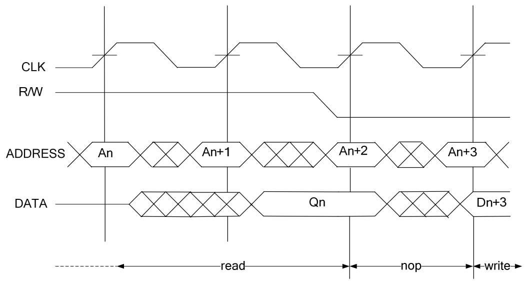

Figure 10. Switching from read to write.

Fig. 11 describes the read-write conflict timing graph with write-then-read configuration. When port A writes data D_x to the address which port B reads, the data D_x is transferred from port A to port B at the next rising edge. Continuous read-write conflicts between the two ports implement the synchronous data transfer from port A to port B.

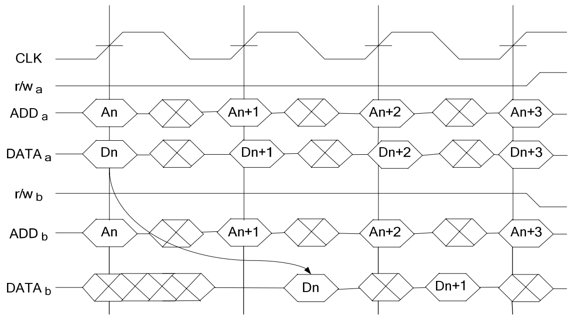

Figure 11. Read-write conflict timing graph with write-then-read configuration.

V. Experiments

We use Xilinx's Virtex-II chips as platform. Each CLB of Virtex-II consists of 4 Slices and each Slice consists of two LUTs. One LUT can be mapped as 16×1 single-port memory, so one CLB can be mapped as a 16×8 single-port memory bank with address decoder circuit integrated. In order to optimize utilization of CLB and reduce the complexity of SPAG, we set the memory width = 8 in our experiment. We call the logic circuit around memory bank as Parallel Access Circuit. In order to reduce routing complexity, memory bank should locate close to its Parallel Access Circuit. Thus, we use a cluster of CLBs, which locates close to each other physically to implement one SPAG. Experiments focus on three aspects: conflict control test, resource cost and the performance of BAC algorithm.

Conflict control test has been carried out in two scenarios according to whether conflicts exist or not [21]. In our design, conflicts have three kinds: read-write conflict, write-write conflict and the combination of the former two. Each of the three kinds can be divided into more small species according to the number of conflict ports, the access type of each port, port-priority and r/w-priority configuration. Fig. 12 shows the time graph of the combination of read-write conflict and write-write conflict. In the first phase, Port 0 and port 1 have write-write conflict, while port 2 has both read-write conflict with port 0 and port 1. In the second phase, port 0 and port 2 have write-write conflict while port 1 has both read-write conflict with port 0 and port 2. In the two phases, simulation works at static port-priority and read-then-write configuration.

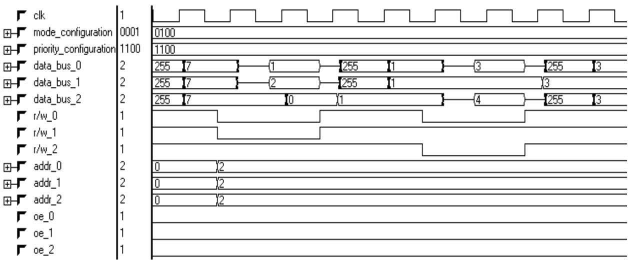

Figure 12. Timing graph of the read-write conflict and write-write conflict.

Resource cost experiments focus on two aspects. First, we measure the LUT resources cost consumed by our design and that consumed by implementing N ports in each memory cell. Second, we discuss the disadvantage of hierarchical architecture for reducing route complexity.

The average cell cost, described by the ration between total LUTs and the capacity M, is indicated by ε. The Parallel Access Circuit's complexity, described by the ratio between LUTs consumed by Parallel Access Circuit and LUTs consumed by memory bank, is indicated by λ.

TABLE I.
LUTs COST WHEN N = 3 AND PL = 1

M/PL	Total LUTs	ε
64	2360	36.875
128	4696	36.688
256	9365	36.582
512	18898	36.910

TABLE II.
LUTs COST WHEN N = 4 AND PL = 1

M/PL	Total LUTs	ε
64	3296	51.500
128	6550	51.172
256	13084	51.109
512	26392	51.547

TABLE III.
LUTs COST WHEN N = 5 AND PL = 1

M/PL	Total LUTs	ε
64	4296	67.125
128	8543	66.742
256	17055	66.621
512	34398	67.184

Table I, II and III describe the different LUTs cost according to different capacity M when PL=1 and N=3, 4 and 5. Implementing N ports in each memory cell can be regarded as the special case, where PL = 1.

TABLE IV.
LUTs COST WHEN N = 3 AND PL = 16

M/PL	LUTs for Memory	LUTs for Logic	Total LUTs	λ	ε
64	512	3688	4200	7.203	4.102
128	1024	7304	8328	7.133	4.066
256	2048	14577	16625	7.118	4.059
512	4096	29294	33390	7.152	4.076

TABLE V.
LUTs COST WHEN N = 4 AND PL = 16

M/PL	LUTs for Memory	LUTs for Logic	Total LUTs	λ	ε
64	512	5896	6408	11.516	6.258
128	1024	11704	12728	11.429	6.215
256	2048	23968	26016	11.703	6.352
512	4096	48518	52614	11.845	6.423

TABLE VI.
LUTs COST WHEN N = 5 AND PL = 16

M/PL	LUTs for Memory	LUTs for Logic	Total LUTs	λ	ε
64	512	7704	8216	15.047	8.023
128	1024	15288	16312	14.930	7.965
256	2048	30527	32575	14.906	7.953
512	4096	61282	65378	14.961	7.981

Table IV, V and VI describe the LUTs cost according to different capacity M when PL =16 and N=3, 4 and 5. It is also can be seen from tables that, as the port number N increases, the Parallel Access Circuit's complexity (λ) increases significantly.

Although port number N can be set to any positive integer theoretically, increased port number N will lead to more LUT resources consumed by implementing Parallel Access Circuit and Basic Conflict Control Circuit. These restrict the reasonable port number to rather small values in practice.

TABLE VII.
AVERAGE CELL COST COMPARISON

Port N	Cell cost of implementing N ports in each cell (ε)	Cell cost of hierarchy architecture (ε)	Reduction
3	36.764	4.076	88.91%
4	51.332	6.312	87.70%
5	66.918	7.981	88.07%

In Table VII, we compare the average cell cost of hierarchical architecture to that of implementing N ports in each memory cell. We can see from Table VII that: hierarchical architecture save almost 88% LUT resources. We can also get good results on other Xilinx Virtex chips, however, such excellent results may be hard to achieve on other FPGA vendors' chips due to different CLB architecture.

TABLE VIII.
MEMORY ACCESS TIME WHEN N = 3, 4, 5 AND PL = 1

M/PL	Access Time (ns) N = 3	Access Time (ns) N = 4	Access Time (ns) N = 5
64	13.453	13.503	13.548
128	14.369	14.405	14.359
256	16.819	17.301	17.924
512	18.346	18.492	18.728

TABLE IX.
MEMORY ACCESS TIME WHEN N = 3, 4, 5 AND PL = 16

M/PL	Access Time (ns) N = 3	Access Time (ns) N = 4	Access Time (ns) N = 5
64	17.161	17.671	18.273
128	18.273	18.765	19.237
256	18.947	19.508	20.319
512	20.684	20.898	21.179

TABLE X.
MEMORY ACCESS TIME COMPARISONS

M/PL	Increment N = 3	Increment N = 4	Increment N = 5
64	27.56%	30.87%	34.88%
128	27.16%	30.27%	33.97%
256	12.53%	12.76%	13.36%
512	12.74%	13.01%	13.09%

As it is well known to us, large capacity M will increase route complexity and increased route complexity will cause longer memory access time. Table VIII shows different memory access time according to different capacity M when PL = 1 and N = 3, 4, 5. Table IX shows different memory access time under the same condition except PL = 16. Table X lists memory access time comparison of the two tables above. It can be seen from Table X that hierarchical architecture reduces route complexity significantly, which leads to achieving 16 times capacity while memory access time only increases almost 30% to 12%. Besides, the efficiency of hierarchical architecture for reducing route complexity will become more significant as capacity M increases.

Access conflict is determined by software behavior. In order to verify the efficiency of BAC algorithm for reducing conflict probability we need high conflict frequency. But, most parallel benchmarks are always unlike to tolerate so high conflict frequency as to impacts program's performance. In some real time applications, multi-processors systems always behave stochastically and independently. So, we pay attention to the efficiency of BAC algorithm for reducing conflict probability under stochastic block read/write. Note that the time proportion between port-access and port-idle varies with different codes. Here, we choose the worst condition that each port carries memory access all the time.

Our work has some comparability with conflict research under the conventional architecture [20], which is built on ASIC memory chips and a separate switch network for 1- to N-port transition. For example, a SPAG is equal to a 1-port ASIC memory chip. In order to achieve low conflict probability conventional architecture needs to integrate enough 1-port ASIC memory chips. However, as the number of 1-port ASIC memory chips increases, memory access time increases sharply. This restricts practical ASIC memory chip number to rather small value. In hierarchical architecture, memory access time increases more slowly according to increasing number of SPAGs. So, large number of SPAGs is practical, which makes our work distinguished from conflict research before.

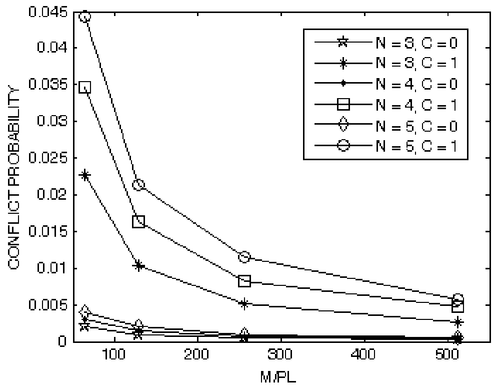

Figure 13. Conflict probability comparison when block read/write length is set to 16.

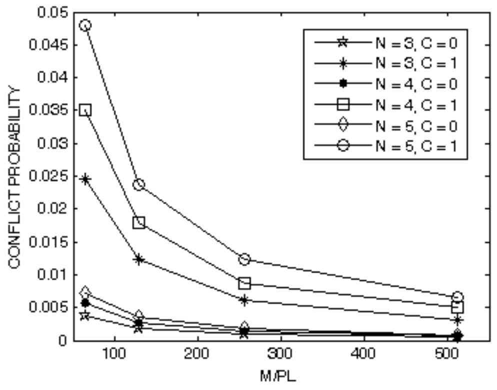

Figure 14. Conflict probability comparison when block read/write length is various from 1 to 16.

Fig. 13 compares the conflict probability results under BAC algorithm to the results which are obtained without such algorithm when block read/write length is set to 16 (N indicate port number; C=0 indicates adopting BAC algorithm). Fig. 14 compares conflict probability results under the same condition except block access length is various from 1 to 16. It can be seen from Fig. 13 and 14 that: as the M/PL doubles, conflict probability reduce almost 50% no matter whether adopt BAC algorithm and whether adopt constant-length block read/write or not. Here we only give part of the experiment results, whose block read/write length is 16 or various from 1 to 16, but this conclusion is the same with other block read/write length.

Note that conflict probability obtained under BAC algorithm is much lower than the results without such algorithm. For example, in Fig. 14 the conflict probability under BAC algorithm is 0.001718 when N=5 and M/PL = 256, while the conflict probability without such algorithm is 0.012406 under the same condition. This indicates that: other than increasing the M/PL value, block read/write is an efficient way to reduce conflict. The following experiment results will show the connection between block read/write length and conflict probability.

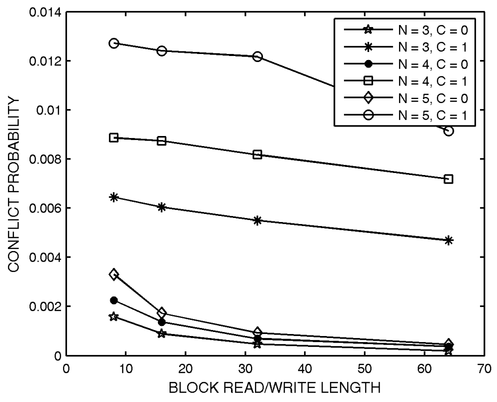

Figure 15. Conflict probability comparison when M/PL is set to 256.

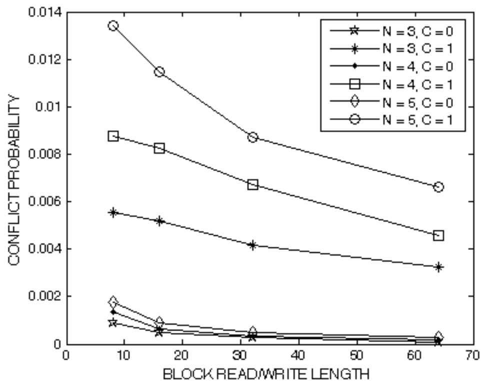

Figure 16. Conflict probability comparison when M/PL is set to 256.

Fig. 15 compares conflict probability results under BAC algorithm to results which are obtained without such algorithm when M/PL is set to 256 and block read/write length is set to 8, 16, 32 and 64. Fig. 16 compares conflict probability results under the same condition except block read/write length is various from 1 to 8, 16, 32 and 64. It can be seen from Fig. 15 and 16 that: as the block read/write length doubles, BAC algorithm can reduce almost 50% conflict compared to almost only 10% (obtained under constant-length block read/write) and 20% (obtained under various block read/write length) reduced conflict without such algorithm. This indicates that: as block read/write length doubles, the conflict probability results under BAC algorithm will decrease more quickly than the results without such algorithm. Although we only show part of the experiment results whose M/PL value is set to 256, this conclusion is the same with other M/PL. We can see this trend from Fig. 17 and 18 that the rate between the conflict probability results under BAC algorithm and the results without such algorithm keeps on decreasing according to the increasing block read/write length.

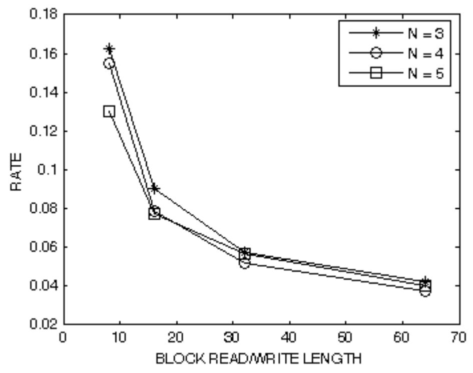

Figure 17. Rate between the conflict probability results under BAC algorithm and the results without such algorithm.

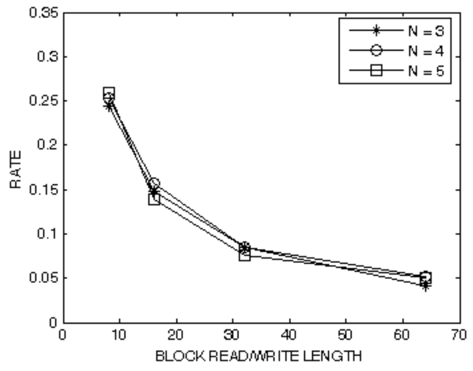

Figure 18. Rate between the conflict probability results under BAC algorithm and the results without such algorithm.

VI. Conclusions

This paper proposes an intelligent N-port memory design based on FPGA. Distributed memory sources on FPGA are mapped as 1-port memory banks and we interleaved these memory banks to form N-port memory. Data in different banks can be accessed simultaneously while accessing the same bank will cause conflicts. In order to manage access conflict, Extend Port Importance Hierarchy (EPIH) algorithm is proposed for basic conflict handling, while Block Access Control (BAC) algorithm is proposed for reducing conflict when processors carry block read/write.

We implement hierarchical architecture on Xilinx's Virtex- II chips. Experiment results shows that, compared to implementation of N ports in each memory cell, hierarchical architecture saves 88% LUT resources. We can also achieve good results on other Xilinx Virtex chips. However, such excellent results may be hard to achieve on other FPGA vendors' chips due to the different CLB architectures. Besides, our design can achieve 16 times capacity than implementation of N ports in each memory cell, while memory access time only increases 30% to 12%. Besides, the efficiency of our design for reducing route complexity will become more significant as capacity M increases. Experiment results on software simulation also show that: M/PL value plays an import role in reducing conflict probability no matter whether under BAC algorithm or not. However, as the block read/write length increases, the conflict probability results under BAC algorithm will decrease more quickly than the results without such algorithm.

ACKNOWLEDGMENT

This paper is partially supported by Beijing Key Discipline Program.

REFERENCES

[1] D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood, "Analytic Evaluation of Shared-Memory Parallel Systems with ILP Processors," *Computer Architecture*, pp. 380-391, June 1998.

[2] Daniel J., Jonathan L. Lemon, Derek L. Eager, Mary K. Vernon, "Analytic Evaluation of Shared-Memory Architectures," *IEEE transactions on Parallel and Distributed Systems*, pp 166-179, February 2003.

[3] IDT corporation "CMOS Static RAM 16K (2K x 8-Bit) IDT6116SA IDT6116LA," datasheet.

[4] CYPRESS corporation "1K x 8 Dual-Port Static Ram CY7C130/CY7C140," datasheet.

[5] IDT corporation "32K x 18 Synchronous Fourport Static Ram IDT70V5388/78," datasheet.

[6] IDT corporation "32/16K x 16 Synchronous Dual-port Static Ram DT70V9279/69S/L," datasheet.

[7] Wen-Tsong Shiue, "Low Power Multiport Memories Exploration and Design," *Canadian Conference on Electrical and Computer Engineering*, pp. 1285–1290, May 2001.

[8] Altera Corporation. "FLEX 10K Embedded Programmable Logic Family Data Sheet", May 2000.

[9] Altera Corporation. "APEX 20K Programmable Logic Device Family Data Sheet", March 2000.

[10] Xilinx, Inc. "Virtex 2.5V Field Programmable Gate Arrays", September 2000.

[11] Xilinx, Inc. "Virtex- II Platform FPGAs Complete Data Sheet," March 2000.

[12] Jha, P.K.; Dutt, N.D. "Library mapping for memories," *European Design and Test Conference*, 1997.

[13] S. J. E. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory," *PhD thesis, University of Toronto*, January 1997.

[14] J. Cong and S.Xu, "Technology mapping for FPGAs with embedded memory blocks," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 179–187, February 1998.

[15] J.Cong and K.Yan. "Synthesis for FPGAs with Embedded Memory Blocks," *International Symposium on Field Programmable Gate Arrays*, pp. 75-81, February 2000.

[16] William K.C. Ho and Steven J.E. Wilton "Logical-to-Physical Memory Mapping for FPGAs with Dual-Port Embedded Arrays," *Lecture Notes in Computer Science*, August 1999.

[17] Iyad Ouaiss; Ranga Vemuri. "Hierarchical memory mapping during synthesis in FPGA-based reconfigurable computers Design," *Automation, and Test in Europe*, pp. 650-657, March 2001.

[18] Iyad Ouaiss; Ranga Vemuri. "Global memory mapping for FPGA-based reconfigurable systems," *15th International Parallel & Distributed Processing*, April 2001

[19] Mattausch, H.J. "Hierarchical N-port memory architecture based on 1-port memory cells," *Solid-State Circuits Conference*, pp. 348–351, September 1997.

[20] Mattausch,H.J; Yamada,K. "Application of port-access-rejection probability theory for integrated N-port memory architecture optimization," *Electronics Letters*, pp. 861–862, April 1998.

[21] A.J. van de Goor and S. Homdioui, "Thorough Testing of any Multiport Memory with Linear Tests," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 217-231, February 2002.

Zuo Wang was born in China in 1982. He received Bachelor Degree of Engineering from Beijing Institute of Technology in 2004. Now, he is a PHD Candidate at Beijing Institute of Technology. Published articles include: Group-caching for NoC Based Multicore Cache Coherent Systems, Nice at France, Design, Automation and Test in Europe 2009; Traffic Analysis for Triplet-based Network Based on Full-system Simulation, Shanghai China, International Symposium on Information Science and Engineering 2008. His researches mainly focus on on-chip network and cache systems.