MULTI-PORTED MEMORIES FOR FPGAS WITH OPTIMUM USAGE OF BRAM

A PROJECT REPORT

Submitted by

DEEPAN SARATHY S

Register No: 15MAE002

Inpartial fulfillment for the requirement of award of the degree

of

MASTER OF ENGINEERING

in

APPLIED ELECTRONICS

Department of Electronics and Communication Engineering

KUMARAGURU COLLEGEOF TECHNOLOGY

(An autonomous institution affiliated to Anna University, Chennai)

COIMBATORE-641049

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2017

BONAFIDE CERTIFICATE

Certified that this project report titled **"Multi-Ported Memories for FPGAs with Optimum Usage of BRAM"** is the bonafide work of **DEEPAN SARATHY S [Reg. No. 15MAE002]** who carried out the research under my supervision. Certified further that, to the best of my knowledge the work reported here in does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Dr K.PARAMASIVAM Ph.D
PROJECT SUPERVISOR
Department of ECE
Kumaraguru College of Technology
Coimbatore-641 049

SIGNATURE
Prof. K. RAM PRAKASH
HEAD OF THE DEPARTMENT
Department of ECE
Kumaraguru College of Technology
Coimbatore-641 049

The candidate with **Register No.15MAE002** is examined by us in the project viva-voce examination held on.....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First I would like to express my praise and gratitude to the Lord, who has showered his grace and blessing enabling us to complete this project in an excellent manner. He has made all things in beautiful in his time.

I express my sincere thanks to my beloved Joint Correspondent, **Shri. Shankar Vanavarayar** for his kind support and for providing necessary facilities to carry out the project work.

I would like to express my sincere thanks to our beloved Principal **Dr.R.S.Kumar M.E., Ph.D.,** who encouraged me with his valuable thoughts.

I would like to express my sincere thanks and deep sense of gratitude to my HOD, **Dr.K.Malarvizhi M.E.,Ph.D.,** and **Prof K. RamPrakash M.E.,** for their valuable suggestions and encouragement which paved way for the successful completion of the project.

In particular, I wish to thank with everlasting gratitude to the Project Coordinator **Ms.S.Nagarathinam M.E., (Ph.D),** Assistant Professor-III, Department of Electronics and Communication Engineering, for her continuous encouragement and motivation throughout the course of this project work.

I greatly privileged to express my deep sense of gratitude to the Project Supervisor **Dr K.Paramasivam Ph.D.,** Professor, Department of Electronics and Communication Engineering,for his continuous support throughout the course. In particular, I wish to thank and express my everlasting gratitude to him for the expert counselling in each and every steps of project work and I wish to convey my deep sense of gratitude to all teaching and non-teaching staff members of ECE Department for their help and cooperation.

Finally, I thank my parents and my family members for giving me the moral support in all of my activities and my dear friends who helped me to endure my difficult times with their unfailing support and warm wishes.

ABSTRACT

Now a days most of FPGA based applications require multi-ported memories to improve the processing speed and allow concurrent read and write data. Most of the FPGAs families are limited with BRAM memory and it support dual port RAM which includes simple and true dual ports only. In order to increase the numbers of ports BRAMs are reconfigured to multi-port memories with available memory resources when application needs more than two ports. In this paper, the various approaches for implementing multi-ported memories are analyzed by using Spartan 6 FPGA of Xilinx. This paper includes a new design and a more efficient way of using a conventional read/write operations and it exploits the different approaches for implementing 2W4R architecture which is used as the building block for overall design architecture of nRmW this paper introduces a efficient design of 4R2W memory that requires less BRAMs memories. The proposed design reduces BRAM compared to previous works. For complex multiported designs, the proposed BRAM-efficient approaches can achieve higher clock frequencies with minimal increase in area. Experimental simulation results explore various parameters that could initiate to improve performance in area, speed and increase the memory depth compared with the previous works. The area and logical elements utilized is also explored and analyzed in detail to help the designer to decide which blocks should be optimized for better efficiency with better area reduction. The new modified LVT design has been proposed with clock gating which optimizes the clock power consumption. The modified LVT design architecture as reduced clock power consumption up to 48 % in 2W4R design, 46% in 2W6R design and 51% in 2W8R design.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	iv
	LIST OF FIGURES	v
	LIST OF ABBREVIATION	vi
1	**INTRODUCTION**	1
	1.1 Replication	2
	1.2 Banking	2
	1.3 Multipumping	3
2	**REVIEW OF LITERATURE**	4
	2.1 Multi-ported memory design approaches	4
	2.1.1 Adaptive Logic Modules-ALM based approach	4
	2.1.2 Replicated approach	5
	2.1.3 Banking approach	5
	2.1.4 Multipumping approach	6
	2.1.5 Live Value Table-LVT approach	6
3	**PROPOSED METHODOLOGY**	8
	3.1.Techniques to increase read ports	8
	3.2.Techniques to increase write ports	9
	3.3.Integrating the read/write techniques	11
4	**BRAM REDUCTION TECHNIQUES**	14

4.1 Techniques to Improve Read Ports 14

4.1.1 Bank Division With XOR Design Scheme 14

4.1.2 2R1W/4R(An Efficient Two-Mode Memory) 16

4.1.3 HBDX Designs With 2R1W/4R Module 17

4.2 Techniques to Improve Write Ports 17

4.3 Integrating the Read/Write Techniques 19

4.3.1 Example of 2R2W memory 19

4.3.2 Extend to 4RnW Memory With 2R1W/4R Module 20

5 SOFTWARE USED 21

5.1 Xilinx 21

5.1.1 Xilinx design tools 21

5.1.2 Xilinx hardware tools 21

6 SIMULATION RESULTS 22

6.1 Design Implementation Using Multipumping 22

6.2 Design Implementation Using Live Value Table 24

7 MODIFIED MULTIPUMPING ARCHITECTURE 28

7.1 Pure Multipumping 28

7.2 Implementation 30

7.3 Operation 32

7.4 Area consumption 33

7.5 Speed 34

8	**MODIFIED MULTIPUMPING ARCHITECTURE WITH POWER OPTIMIZATION**	36
	8.1 Design Implementation Using Modified	38
	8.1.1 2w8r Logical Module Test Bench Waveform	
	8.2 Results With Clock Gating	40
	8.3 Power Reduction Analysis	41
	8.4 Overall Result Analysis	42
9	**CONCULSION**	43
	REFERENCES	44
	LIST OF PUBLICATION	

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
1.1 (a)	Replication	3
(b)	Banking	3
(c)	Multipumping	3
2.1	Multi-ported memory using ALM	5
2.2	Multi-ported memory using multipumping approach	6
2.3	Multi-ported memory using LVT approach	6
3.1 (a)	Replication technique that replicates BRAMs to support m read ports.	10
(b)	memory design that can support two concurrent reads with two BRAMs	10
3.2 (a)	LVT technique that supports *n* write ports	13
(b)	Memory design that can support two concurrent writes with LVT technique.	13
4.1 (a)	2R1W mode	16
(b)	4R mode	16
4.3	Architecture that integrates HBDX and BDRT to achieve a mRnW memory	18
6.1	Multipumping mechanism	21
6.2	Synthesis window for 2W4R architecture using multipumping	23

6.3	Live value table mechanism	25
6.4	Synthesis window for 2W4R architecture using live value table	27
7.1	Simplified figure showing read and write operations in a 2W/4R pure multipumped memory (With a multipumping factor of 2)	29
7.2	Shows a generalized design for LVT-based modified LVT architecture with mW/nR ports.	30
7.3	Shows write and read operation of a 4W/8R LVT based memory whose banks are built using 2W/4R modified LVT design.	31
7.4	Waveforms showing external clock and internal clock used for a 4W/8R LVT_PMP memory.	32
8.1	Clock Gated Low Power RAM	36
8.2	Generalized design for Modified LVT architecture with mW/nR ports,	38
8.3	2W8R Logical Module Test Bench Waveform	39

LIST OF ABBREVIATIONS

ACRONYMS	ABBREVIATION
BRAM	Block RAM
LVT	Live Value Table
BDX	Bank Division With XOR Design Schemes
HBDX	Hierarchical Bank Division With XOR
BDRT	Bank Division With remap table
CG	Clock Gating

CHAPTER 1

INTRODUCTION

Field-Programmable gate arrays (FPGAs) have been broadly used in fast prototyping of complex digital systems. FPGAs contain programmable logic arrays, usually referred to as slices [4]. Slices can be configured into different logic functions. The flexible routing channels can support data transferring between logic slices. In addition to implementing logic operations, if needed, the slices can also be used as storage elements, such as flip-flops, register files, or other memory modules. Due to the increasing complexity of digital systems, there is a growing demand for in- system memory modules. Synthesizing a large number of memory modules would consume a significant amount of slices, and would therefore result in an inefficient design. The excessive usage of slice could also pose a limiting factor to the maximum size of a system that can be prototyped on an FPGA .

To more efficiently support the in-system memory, modern FPGAs deploy Block RAMs (BRAMs) that are hard core memory blocks integrated within an FPGA to support efficient memory usage in a design. Compared with the storage module synthesized by slices, BRAMs are more area and power efficient while at the same time achieving higher operating frequencies.

An FPGA usually deploys multiple BRAMs with the same specification. Each BRAM can be configured as two port mode or dual-port mode [4]. Designers can utilize these memory blocks to implement the in-system storage module of a design. Multiported memories, which allow multiple concurrent reads and writes, are frequently used in various digital designs on FPGAs to achieve high memory bandwidth. For example, the register file of an FPGA-based scalar MIPS-like soft processor [3] requires one write port and two read ports. severe concerns to designers when implementing multiported memories on an FPGA. Using an excessive amount of BRAMs multiported memory could seriously restrict the usage of BRAMs for other parts of a design. Furthermore, the in sufficient BRAMs would force designers to synthesize the storage modules with slices, and further consume a vast amount of

slices as well as limiting the maximum operating frequency of the design. The technique's that are used by the designers are as follows

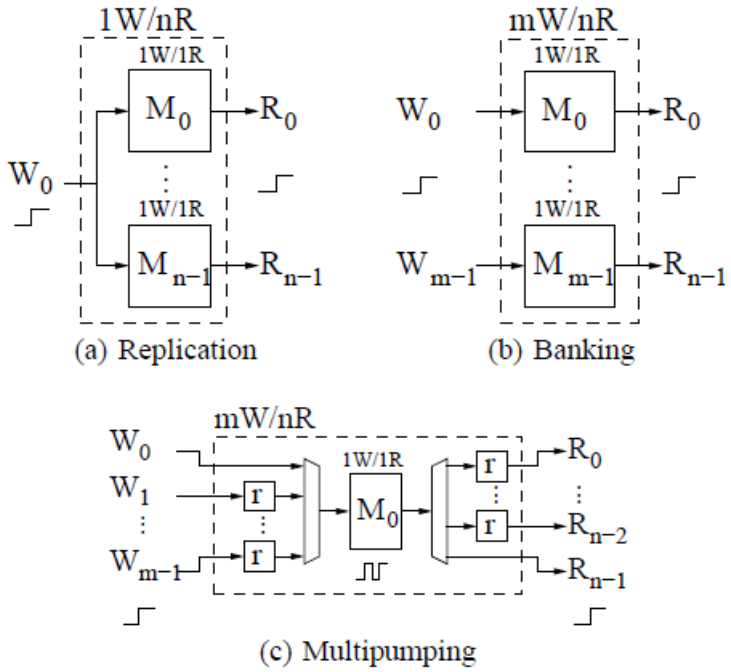

Fig 1.1 (a) replication (b) Banking (c) Multipumping

1.1 REPLICATION

Replication Fig 1.1 (a) is an easy way to increase the number of read ports of a simple memory (i.e., to 1W/nR):simply provide as many copies of the memory as you require read ports, and route the write port to all copies to keep them up-to-date We evaluate replication for bothM9Ks (Repl-M9K) and MLABs (Repl-MLAB). All of the Repl-M9K designs fit into twoM9K BRAMs, such that those points are all co-located in the figure. Replication requires no additional control logic, hence these designs are very efficient. For 1W/2R memories with a depth greater than 256 elements, another pair of M9Ks would be added at every depth increment of 256 elements—resulting in a relatively slow increase in area as memory depth increases. We also consider replicated designs composed of MLABs (Repl-MLAB). Unfortunately, Quartus could not place and route any MLAB-based memory with more than 64 elements. Since each MLAB stores the equivalent of 160 ALMs, the Repl-MLAB implementation requires much less interconnect than the Pure-ALM implementation but considerably more than the Repl-M9K implementation. For example, the 32-entry Repl-MLAB 1W/2R memory requires only 198 equivalent ALMs, but still suffers a lower

operating speed of 376 MHz.

The replicated M9K designs (Repl-M9K) are evidently far superior to the alternatives, with an area of 90 equivalent ALMs and maximum operating frequency of 564 MHz. However, the drawback to this approach is that there is no way to provide additional write ports with replication alone—we must pursue other techniques to provide more write ports.

1.2 BANKING

Banking fig 1.1(b) is similar to replication, except that the memory copies are not kept coherent; each additional memory now supports an additional read and write port, providing an easy way to increase ports arbitrarily (mW/mR). The conventional way to use banking is to divide memory locations evenly among the banks, such that each read and write port are tied to a certain memory division. However, a memory with only banking is not truly multi-ported, since only one read from a certain division is possible in a given cycle. For this reason we do not evaluate banked-only memories, although a close estimate of the Fmax/area of amW/mR banked memory is the corresponding 1W/mR replicated design.

1.3 MULTIPUMPING

Multipumping fig 1.1(c) internally uses an integer multiple of the external system clock to multiplex a multiported memory with fewer ports, giving the external appearance of a larger number of ports (mW/nR). This requires the addition of multiplexers and registers to hold temporary states, as well as the generation of an internal clock, and careful management of the timing of read-write operations. for further describe the details of implementing a multi pumped design in the next section.

CHAPTER 2

REVIEW OF LITERATURE

This chapter deals with review of literature about feature selection algorithms, various design techniques that are implemented to improve speed and reduce Bram usage.

2. MULTI-PORTED MEMORY DESIGN APPROACHES

In this section, conventional approaches of designing multi-ported memory are described briefly. Each one will be introduced in the structure of circuits, operation, advantages, disadvantages and corresponding applications. At the end of this section, proposed approach is presented to reduce most of the conventional approaches' disadvantages, to give effective performance that could satisfy the demand of modern designs.

2.1. ADAPTIVE LOGIC MODULES-ALM BASED APPROACH

The first method is to implement Multi-Ported Memory by ALMs (Adaptive Logic Modules) which are the basic building block of logic in the Stratix III architecture [2] and provide advanced features with efficient logic utilization. This design approach uses D locations memory (equivalent to log2D address width) with D **(m-to-1) multiplexers** (m is the number of write ports) in front of memory to select the suitable memory locations to write data and n (n is the number of read ports) **(D-to-1) multiplexers** behind memory to select suitable data to read from memory. The problem is that a very large area of ALMs will be used to implement 2 groups of multiplexers and memory; they will reduce the operation frequency and make the CADs tool take a long time to implement place & route in FPGAs. So,

This method is not effective to implement multi ported memory when the number of ports as well as the data depth is large [6], [7]. Figure 2.1 illustrates the operation of multi ported memory using ALMs with m Write Ports and n Read Ports. In this figure, only the data signals are shown (the address, clock, enable signals are ignored).Implementing multi-ported memory using ALMs usually takes more resource than using specific RAM blocks, because ALMs are usually configured to implement logic, arithmetic and register functions, thus not good in both area and

speed performance to implement memory. RAM blocks would be concentrated to implement multi-ported.

2.2. REPLICATED APPROACH

The replicated method is used to expand the number of read ports of memory by adding some extra copies of RAM blocks with the number of copies based on the number of read ports. This method is only used when the memory have only one write port.

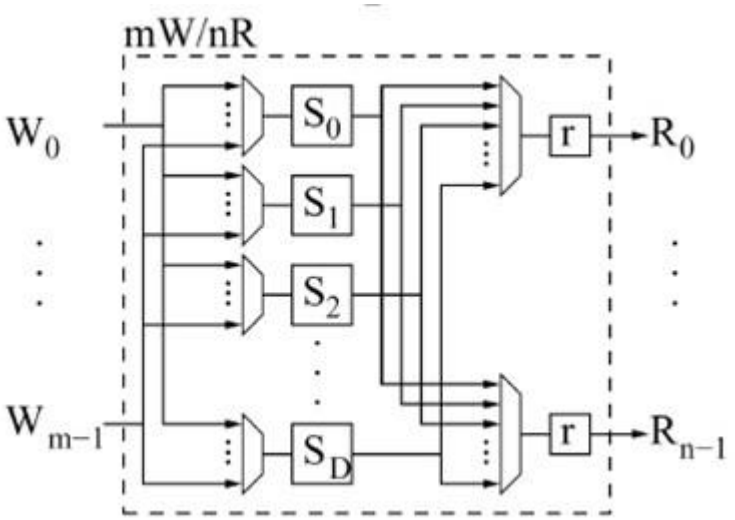

Fig 2.1 Multi-ported memory using ALM

2.3. BANKING APPROACH

Compared with the replicated method, the banking method as in Figure 2.2 is different and can support an arbitrary number of read ports and write ports. Memory will be divided into m sections (corresponding to m write ports). So that, each write port and read port can only access to only one corresponding section. Therefore, this method does not support fully multiported memory because a write port cannot access to any location and similarly with the read port.

2.4. MULTIPUMPING APPROACH

This approach uses an external clock that is multiple speed of system clock (that called multipump factor) to control the memory location access process as in

The temporary registers and multiplexers will be added to the circuit before and after the memory to multiplex the read/ write operations. For instance, a 1W/1R can support to 2 write ports and 2 two read ports with the multipump factor is 2.

Therefore, multipumping approach helps to reduce the area of the design and is suitable for some applications that don't need high speed. On the contrary, the main disadvantage of this design is the reduction of the operation speed to multipump factor time, for example, multipumping with multipumping factor 2 will reduce the operation speed by two times.

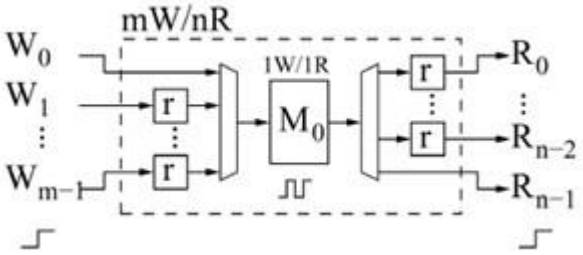

Fig 2.2 Multipumping

2.5. LIVE VALUE TABLE - LVT APPROACH

The LVT approach [3] was proposed from a form of indirection through a structure called the Live Value Table (LVT), which is itself a small multi-ported memory implemented in reconfigurable logic similar to Figure 2.1. This approach allows a banked design to behave like a true multi-ported design by directing reads to appropriate banks based on which bank holds the most recent or "live" write value as use be combined with any other methods to create the memory

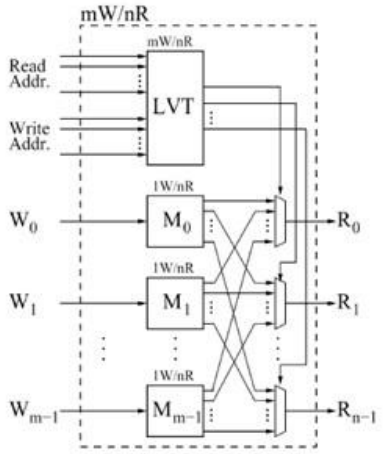

Fig 2.3 Multiported memory using Live Value Table

DRAWBACKS

Every approach as some drawbacks in it as mentioned below so in order to rectify those problem this project is under taken

- The replication can support Only one write port
- The banked BRAM approach as difficulties in Fragmenting data
- The live value table approach can be used only for a small memory depth
- The xor based approach requires more BRAM compare to LVT Approach

CHAPTER 3

PROPOSED METHODOLOGY

3.1 MUTIPORTING MEMORY TECHNIQUES

To implement a multiported memory on an FPGA, two types of design techniques are required, namely increasing read ports and increasing write ports. The approach of replication [1], [3] enables multiple read ports by replicating the data on multiple BRAMs. This technique uses low complexity of control logic, but requires excessive usage of BRAMs. LVT, which is implemented by synthesizing slices on FPGA, enables multiple write ports by duplicating BRAMs and tracking which BRAM stores the latest value of an address. The other approach to increase write ports is referred to as XOR-based [2]. Different from LVT, which uses a table to track the location of the latest value, the XOR-based design duplicates BRAMs and encodes the stored data with XOR operations. The target data can be retrieved by applying the XOR again. In general, the XOR-based approach can achieve a higher operating frequency, but requires more BRAMs than the LVT approach. Note that this paper focuses on architectural solutions to achieve multiple accesses for a general memory that takes requests at the current cycle and returns results in the next cycle.

The Users of the multiported memory can be completely ignorant of the details of memory designs. There are other works focusing on enabling multiple accesses for specific types of storage elements, such as register files [6]–[8]. They enable concurrent reads with an approach similar to replication, but avoid write conflicts by renaming the registers with software approaches, such as compiler or assembler. Approaches which tackle specific storage functions and involve effort of users, are not in the scope of this work. The following sections will provide more in-depth discussions about implementations and design concerns of these techniques. To facilitate a more general discussion, the following paragraphs use a memory bank to refer to a standalone memory module used as a building block to implement a memory system. A memory system usually consists of multiple banks. The memory space, also referred to as memory depth, is distributed across the banks. When designing a memory system on FPGAs, a BRAM can be used to support the complete memory space.

BRAMs can also be deployed as banks to enable larger memory space or higher access bandwidth.

3.1.1 TECHNIQUES TO INCREASE READ PORTS

Replication is a widely adopted technique to increase read ports [1], [3]. This technique enables multiple read ports. This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination. EFFICIENT DESIGNS OF MULTIPORTED MEMORY ON FPGA 3

Fig 3.1(a) Replication technique that replicates BRAMs to support m read ports. Fig 3.1(b) Example of a memory design that can support two concurrent reads with two BRAMs M0 and M1. The read R1 is accessing M0 at address 2, while the read R2 is accessing M1 at address 3. by replicating the data to multiple BRAMs. When there are multiple reads, each of these reads will be directed to a distinct BRAM to access the target data without conflicting with other reads. The data between these BRAMs should be updated simultaneously. When there is a write to an address, this write needs to be routed to every BRAM and updates the data to the corresponding address in each BRAM. Fig. 3.1(a) illustrates how multiple read ports are enabled by the replication technique. Assume a multiported memory design supporting m concurrent reads. Each Mi, where i is from 0 to m−1, is a BRAM module. The data in M0 are replicated to other BRAMs (M1 to Mm−1) in order to support multiple concurrent reads R0 to Rm−1. When there is a write W0, the write request is routed to the write ports of all the BRAMs and updates the values at the corresponding address of each BRAM simultaneously. Fig. 3.1(b) shows an example of a memory that can support two concurrent reads. In this example, there are two data stored in the BRAMs, where data A are at address 2, and data B are at address 3. When the memory receives two read requests R0 and R1 accessing addresses 2 and 3, respectively, each read request can access one of the BRAMs and avoid conflicting with each other. In this way, the memory with two BRAMs M0 and M1 can support two simultaneous reads. The main advantage of replication is its simplicity without requiring complex control logic; however, it needs to replicate m times the number of memory modules, where m is the number of read ports supported by the target multiported memory design.

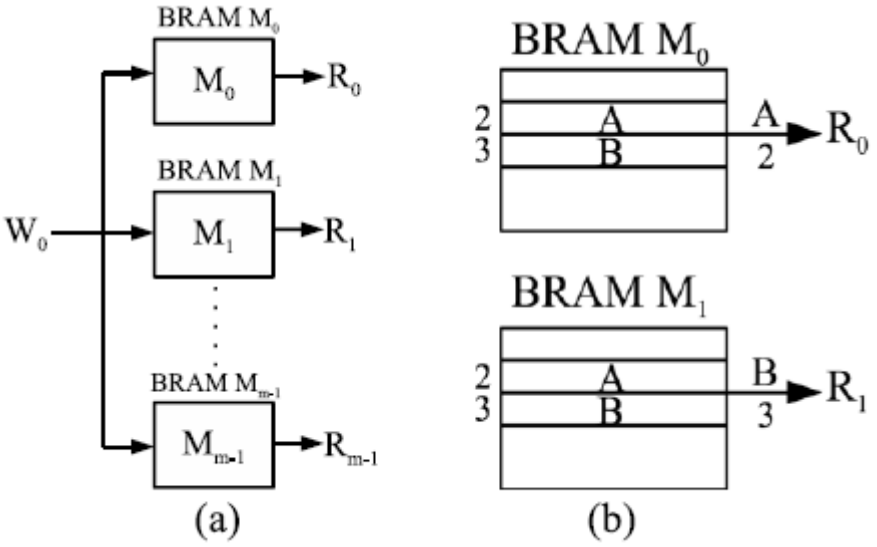

Fig.3.1. (a) Replication technique that replicates BRAMs to support *m* read ports. (b) Example of a memory design that can support two concurrent reads with two BRAMs *M*0 and *M*1.

3.2 TECHNIQUES TO INCREASE WRITE PORTS

There are two types of techniques that have been used to increase write ports in a multiported memory design on FPGAs. The first technique applies LVT. This technique utilizes multiple replicated BRAMs to enable multiple concurrent writes. Different write ports are connected to different BRAMs. A table, called LVT, is used to track the correct BRAM that stores the most up-to-date data of an address. The second technique uses an XOR-based scheme to enable multiple write ports. The XOR-based technique encodes the stored data by using XOR operations, and retrieves the correct data by applying the XOR again. The following paragraphs will elaborate the details of these techniques. LVT is a technique proposed in [1] and [3] to support multiple write ports on FPGAs. illustrates the data access flow of the LVT design that enables n concurrent write requests. The memory is replicated n times with BRAMs M0 to Mn−1. each BRAM Mi, where i is from 0 to n − 1, can support one write port. Since multiple writes would update different memory addresses, an additional block LVT is implemented to keep track of the location of the latest value of a memory address. illustrates an example when there are two write requests W0 and W1. W0 writes data A to address 5, while W1 writes data B to address 7. In this example, W0 stores data A to BRAM M0, and W1 stores data B to BRAM M1.

Each BRAM is associated with an identification number defined by designers. This paper simply uses the serial number of a BRAM as its identification number. Then the writes W0 and W1 will, respectively, update the LVT with identification numbers of the BRAMs that keep the latest data. The identification numbers for M0 and M1 (values 0 and 1) are updated to the entries 5 and 7, respectively, in the LVT.

The XOR-based approach proposed in [2] is a way to increase write ports without a table. illustrates an XOR-based memory design that can support two simultaneous writes W_0 and W_1 and one read R_0. The design contains a total of four BRAMs where each BRAM is assumed to support one read and one write. The XOR-based design encodes the stored data by using XOR operations. For example, when W_0 stores a new value A_{new}, the A_{new} will be XOR-ed with the stale value A_{stale} at the same memory address from the two bottom BRAMs. The XOR-ed value of the two instances ($A_{new} \oplus A_{stale}$) will be stored to the two top BRAMs. Similarly, the write W_1 will XOR the values B_{new} and B_{stale} and store

The encoded value to the two bottom BRAMs. With the encoded values, the read R_0 can recover the most recent value A_{new} by XOR-ing the two values from the two BRAMs. The value-recovering operation. The XOR-based approach can achieve multiple writes without requiring a table to track the position of the most recent value. However, this approach needs n^2 memory modules to enable a design with n writes and one read

$$R_0 = (A_{new} \oplus A_{stale}) \oplus A_{stale} = A_{new}.$$

3.3 Integrating The Read/Write Module For Improving Multiporting memory Techniques

The write $W0$ updates the value C to address 2 while $W1$ stores the value D to address 3. The reads $R0$ and $R1$ are retrieving the data from address 0 (value A) and address 1 (value *B)*, respectively. These four accesses ($W0$, $W1$, $R0$, and $R1$) are sent to the memory simultaneously. The reads try to identify the correct BRAM2 modules that have the most recent values of the target data. After querying the LVT, the most recent values for $R0$ (address 0) and $R1$ (address 1) are both located at M20. Recall that each BRAM2 is a 2R1W module. The two reads ($R0$ and $R1$), in this example, will both access M20. Meanwhile, the two writes $W0$ and $W1$ will, respectively, store the data to M20 and

M21, and update the LVT. Fig. 3.2(b) shows the memory state after completing the requests of all the reads and writes. Fig. 3.2 shows the approach adopted in [2] that combines replication and XOR-based to support two reads (*R*0 and *R*1) and two writes (*W*0 and *W*1). Note that each building block is also a BRAM2 module that can support 2R1W.The data flow of two reads and two writes. *R*0 reads both the values *A*stale and *A*⊕*A*stale from address 2 at two BRAM2 modules, and recovers A by XOR-ing these values. *R*1 can recover B with a similar flow. The operations of reads are shown in the following equations:

$$R0 = (A \oplus A\text{stale}) \oplus A\text{stale} = A \quad (2)$$

$$R1 = (B \oplus B\text{stale}) \oplus B\text{stale} = B. \quad (3)$$

At the same time, *W*0 reads value *C*stale from three BRAMs at the bottom of Fig. 3.2(b), and updates the encoded value *C*new ⊕ *C*stale to address 0 of the three BRAMs on the top of Fig. 3.2(b). *W*0 can update the values at address 1 with a similar flow. Recall that each BRAM2 is a 2R1W module. In this 2R2W design, each write needs to first read all of the stale values from BRAM2 modules and update the XOR-ed values. Therefore, the design in needs a total of six BRAM2 modules to provide sufficient internal read ports and support all the data accesses. Each read port of the multiported memory needs to occupy two internal read ports of a BRAM2 module.

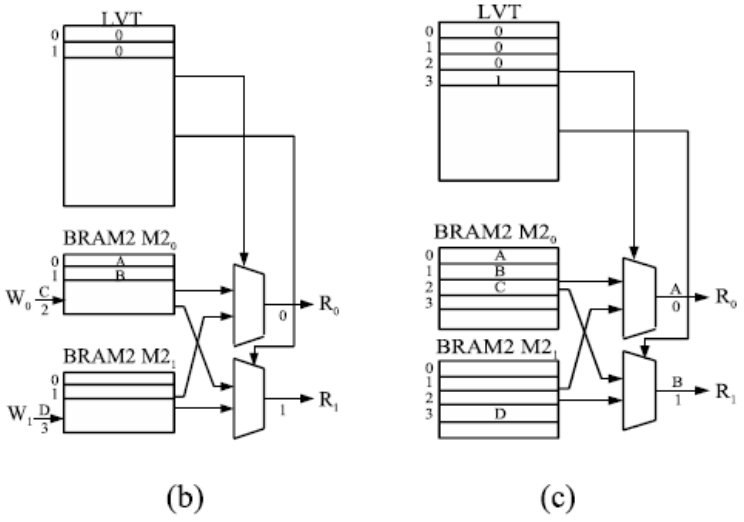

Fig.3.2. (a) LVT technique that supports *n* write port. (b) Example of a memory design that can support two concurrent writes with LVT technique.

CHAPTER 4
BRAM REDUCTION TECHNIQUES

This section proposes efficient solutions to implement multiported memories on FPGAs. Unlike the replication method in [1] and [3], the approach proposed in this paper supports multiple reads with XOR operations, while multiple writes can be enabled using additional BRAMs. A remap table is added to track the location of the correct data. On top of the main architecture, this paper introduces a brand new perspective of using a 2R1W module as either a 2R1W or a 4R module, denoted as a 2R1W/4R memory. By applying the 2R1W/4R, this paper exploits the versatile usage mode and proposes a hierarchical XOR-based design of 4R1W memory that requires fewer BRAMs than previous designs. Memories with more read/write ports can be supported by extending the proposed 2R1W/4R memory and the hierarchical 4R1W memory.

4.1. TECHNIQUES TO INCREASE READ PORTS

4.1.1 BANK DIVISION WITH XOR DESIGN SCHEME*:*

Bank Division With XOR (BDX) is an approach to increase read ports proposed in [9]. Unlike the method used in [1] and [3], BDX avoids replicating the storage elements of the whole memory space. With BDX, multiple reads can be supported by using the XOR operations. Note that BDX is different from the XOR-based design in [2]. The XOR-based approach in [2] uses XOR operations to increase write ports by storing the encoded data to maintain the data coherence between memory modules. BDX uses XOR operations to increase read ports by retrieving the target data from the encoded value. The memory space is distributed to four data banks (banks 0–3). One XOR-bank is added to keep the XOR values of the databanks. Each of these memory banks is assumed to be a 2RW module that can support two reads, or two writes, or one read and one write. This is the dual-port mode supported by the BRAMs in Virtex-7 FPGA. Compared with the dual port mode of BRAMs, the BDX scheme enables one more read port. Therefore, the BDX design become

XOR-bank stores the XOR-ed value of all the data at the same offset in every data bank. The XOR operation for this example is shown in (4).

there are four data banks the value at the same offset (offset n) of each bank (banks 0–3) will be XOR-ed and stored to the same offset (offset n) at the XOR-bank

$$Pn = D(0,n) \oplus D(1,n) \oplus D(2,n) \oplus D(3,n). \quad (4)$$

Two reads $R0$ and $R1$ are both going to bank 1. In this example, one read ($R0$) will access the data directly from bank 1 ($R0 = D(1,0)$). Due to the bank conflict, the other read $R1$ cannot access bank 1. Instead, the target data of $R1$ can be recovered by XOR-ing the values at the same offset of the other data banks as well as the XOR-bank.

As shown in the following equation, the target data at offset n of data bank 1 can be recovered by XOR-ing the values at the same offset (offset n) from banks 0 to 3, as well as the XOR-bank:

$$D(1,n) = D(0,n) \oplus D(2,n) \oplus D(3,n) \oplus Pn. \quad (5)$$

The write request in this example is implemented in the two-cycle pipeline architecture. At the first cycle W0 writes the data D(1,n) directly to bank 1. At the same cycle, the data at the same offset of the other banks [D(0,n), D(2,n), and D(3,n)] together with D(1,n) are read and XOR-ed. At the second cycle, the XOR-ed value will be written back to the XOR bank at offset n. With the pipeline architecture, this design can process one write request every cycle.

The 2R1W memory introduced previously only needs an additional XOR-bank that requires the same storage size of each data bank. In this case, assume all the data banks are of equal size. The number of memory entries in the XOR-bank is MemoryDepth/#DataBanks, where MemoryDepth represents the total number of memory entries in the memory spaced notes the number of data banks in the multiported memory. To support more than two read ports, BDX needs to deploy multiple 2R1W memories, shows an example of mR1W memory. Each 2R1W module supports two out of m reads. In this way, all the m reads can be serviced concurrently by multiple 2R1W modules. Note that when implementing a multiported memory on FPGAs, To address this issue, this paper proposes a new architecture to increase read ports, referred to as hierarchical bank division with XOR (HBDX). HBDX applies a new perspective of using a 2R1W module as a 2R1W/4Rmodule. Section III-A2 will

discuss the 2R1W/4R module while Section III-A3 will introduce the design of HBDX

4.1.2 2R1W/4R (AN EFFICIENT TWO-MODE MEMORY):

To implement HBDX in an efficient way, this paper introduces a brand new perspective of using a 2R1W module as either a 2R1W or a 4R module. This new way of using the 2R1W module is denoted as 2R1W/4R. This hybrid module can support either2R and 1Wor 4R. Note that the 2R1W/4R module uses exactly the same design as the 2R1W module introduced. illustrates how the two modes work. Fig. 4.1(a) shows the 2R1W mode. When there is a write request W0, this design can support up to two conflicting reads. The write request W0 stores the data directly to the target data bank, and reads all the data at the same offset from the other data banks (Rupdate) to update the XOR-bank. Fig. 4.1(b) shows the 4R mode. This mode only works when there is no write request. In this case, the design can support up to four conflicting reads. Consider one of the worst cases when all the four reads (R0 to R3) are going to bank 0. As illustrated in Fig. 4.1(b), R0 and R2 access bank 0 directly. At the same time, R1 and R3 can retrieve the target data by XOR-ing the values at the same offset of the other data banks as well as the XOR-bank.

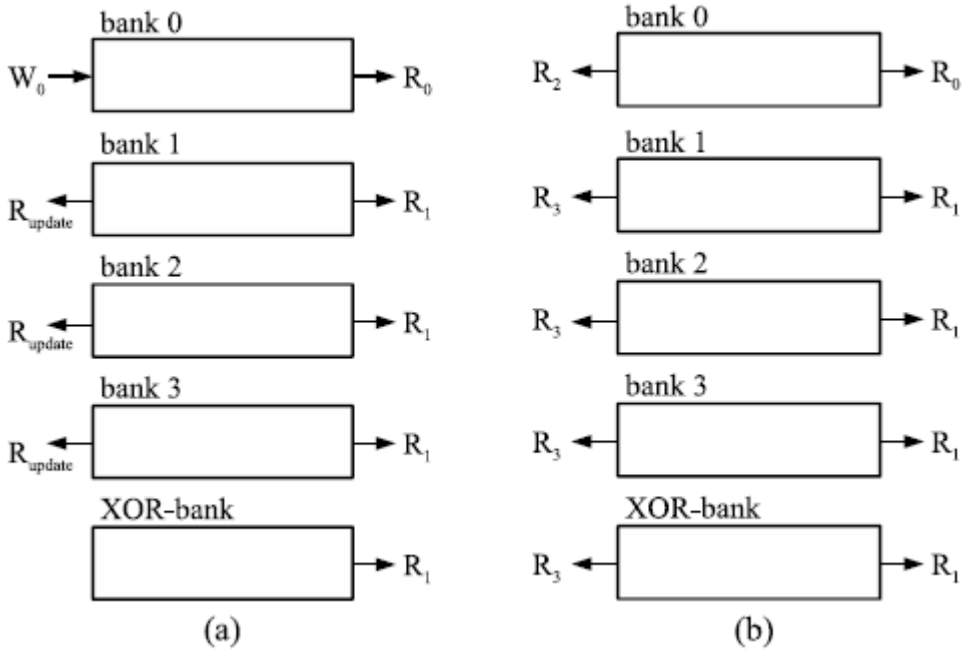

Fig. 4.1.Two modes of 2R1W/4R module. The module is implemented with four data banks and one XOR-bank. (a) 2R1W mode. (b) 4R mode.

4.1.3 HBDX DESIGNS WITH 2R1W/4R MODULE:

Design scheme that can support more read ports by replicating the 2R1W module. However, this design scheme could significantly increase the usage of the limited BRAMs on an FPGA. To achieve a more BRAM-efficient design, this paper proposes HBDX, which adopts a hierarchical structure that organizes the 2R1W to achieve 4R1W without replicating the 2R1W module. To further enhance the design, HBDX in this section leverages the 2R1W/4R scheme introduced in the previous section as the basic building module to implement a 4R1W module. In this 4R1W design, each basic building block is a 2R1W module of the BDX scheme. According to the 2R1W/4R memory proposed in the previous section, this 2R1W module can be used as either 2R1W or a 4R module. The HBDX will utilize this versatile usage mode to achieve a more efficient 4R1W design.

Consider one of the worst cases when all the 4R and 1W are going to bank 0. Two read requests, *R*0 and *R*1 in this example, would read directly from bank 0. The other two read requests, *R*2 and *R*3, would read the same offset from the other data banks (banks 1–3) and the XOR-bank to recover the target data. And the write request *W*0 can be supported with a pipeline architecture similar. In this case, *W*0 stores the data directly to bank 0. The XOR-ed value will be stored back to the XOR bank in the next cycle. In this example, bank 0 is used in the 2R1W mode, while other banks are used in the 4R mode. The HBDX-based 4R1W memory has demonstrated to be more BRAM efficient than the previous approach of simply duplicating the 2R1W module [9]. Section IV will more comprehensively discuss the design concerns between the number of BRAMs and operating frequency on an FPGA.

4.2 TECHNIQUES TO INCREASE WRITE PORTS

Bank division with remap table (BDRT) is an approach to increase write ports proposed in [7]. Unlike the LVT design used in [1] and [3], BDRT avoids replicating the whole memory space and supports multiple writes using additional BRAMs and a remap table to track the location of the latest data.

This example consists of two data banks (banks 0 and 1), one bank buffer, and a remap table. The Null entries in a memory bank are the entries that do not store and valid data. When receiving *W*0 and *W*1, these requests will first look up the remap table to identify the correct BRAM that stores the latest data. According to the remap table, *W*0 and *W*1 are, respectively, going to address 0 and address 1 in bank 0. In this case, *W*0 will store the value directly to address 0 in bank 0. *W*1 will be directed to the bank buffer whose offset 1 is a Null entry. At the same time, the remap table needs to be updated to reflect the modified location of *W*1, as shown in Fig. 4.2.

The address 1 of the remap table stores the value 2 which is the identification number of the bank buffer. This state of remap table shows that the data of address 1 is now stored in the bank buffer. Compared with the LVT approach used in previous works, the BDRT requires smaller storage space. The extra cost of BDRT is the additional registers to implement the remap table. The required number of these extra registers is shown in the following equation, where *MemoryDepth* is the total number of memory entries in the memory space and *#DataBanks*denotes the number of data banks in the multiported memory

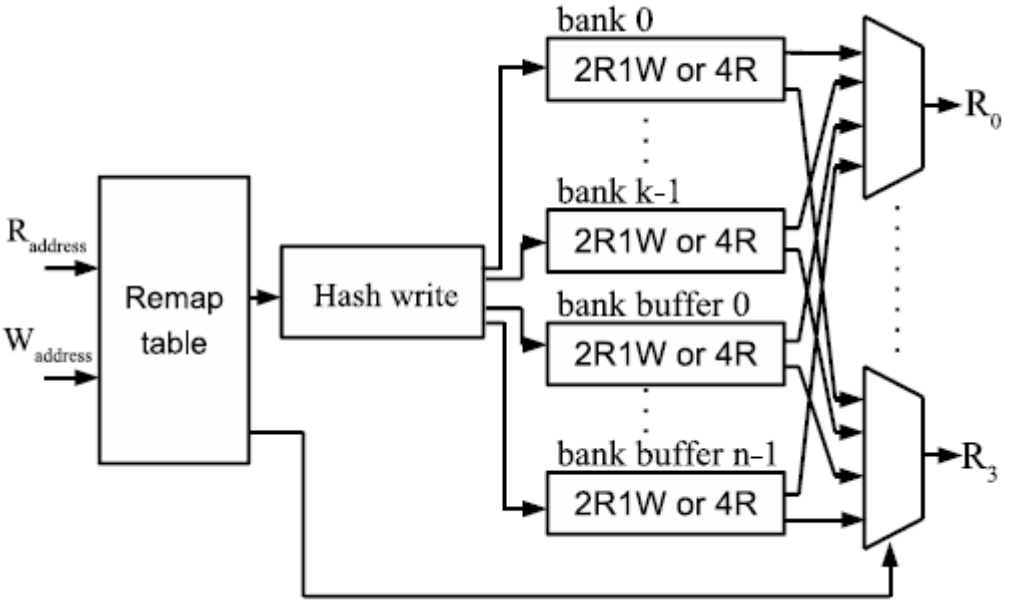

Fig.4.2. Architecture that integrates HBDX and BDRT to achieve a*mRnW* memory.

4.3 INTEGRATING THE READ/WRITE TECHNIQUES

This section shows the design of a multiported memory that integrates HBDX and BDRT. An architecture that uses HBDX and BDRT to implement an *m*R*n*W memory. This memory architecture is divided into *k* data banks. Based on BDRT, to support all the writes, there require total $n - 1$ bank buffers. A hash mechanism is added to distribute the writes to banks.

4.3.1 EXAMPLE OF 2R2W MEMORY:

Example of 2R2W memory that applies this architecture. Assume there are two write requests and two read requests. The writes *W*0 and *W*1 are going to addresses 0 and 1, respectively. The reads *R*0 and *R*1 are retrieving the data from addresses 2 and 3, respectively. Each bank is a 2R1W module that can support two read ports and one write port. The 2R1W module can be implemented with BDX. A read request will access all the banks and choose the data returned by the bank that stores the latest value of the target data based on the record in the remap table. The two write requests, *W*0 and *W*1, will look up the remap table for the correct banks to store the values. In this example, the addresses of the two writes, addresses 0 and 1, both belong to bank 0. In this case, *W*0 will store the value directly to bank 0. *W*1 will be directed to a bank buffer that has a Null entry at the same offset of *W*1 on bank 0. At the same time, the remap table needs to be updated to reflect the modified location of *W*1,

4.3.2 EXTEND TO 4RNW MEMORY WITH 2R1W/4R MODULE:

This section further extends the design to a 4R*n*Wmemory. According to the architecture presented, a 4R*n*W memory requires 4R1W modules as building blocks. A 4R1W module can be implemented simply by the replication technique introduced in [1], [3], and [9].

This paper proposes a more efficient way of implementing a 4R*n*W memory by using the 2R1W/4R module introduced in this paper. Although supporting two different usage modes, a 2R1W/4R module is essentially a 2R1W memory block in terms of design cost and performance will be improved. As will be shown in Section IV, compared with the 4R1Wmodule from replication, the 2R1W/4R module on an FPGA requires fewer BRAMs and attains higher operating frequencies. shows a 4R*n*W memory implemented with *k* data banks and using 2R1W/4R as the building

block. The four reads can be supported by exploiting the 4R mode of the2R1W/4R module. Note that a 2R1W/4R module cannot support any write requests when it is servicing more than two reads by using BRAM memory. To support *n* write requests, this design needs *n*ext a bank buffers to solve conflicting write requests.

CHAPTER 5

SOFTWARE USED

5.1 XILINX

Xilinx ISE (Integrated Software Environment) is a software tool produced by Xilinx for synthesis and analysis of HDL designs, enabling the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

It provides synthesis and programming for a limited number of Xilinx devices. In particular, devices with a large number of I/O pins and large gate matrices are disabled. The low-cost Spartan family of FPGAs is fully supported by this edition, as well as the family of CPLDs. The 14.4 version released in 2012-12-18 and the GNU/Linux version has a size of 5.8 GB (update size needed)

5.1.1 XILINX DESIGN TOOLS

- The BRAM Block design is generated by the EDK tools.
- XST is the synthesis tool used for synthesizing the BRAM Block. The EDIF netlist output from XST is then input to the Xilinx Alliance tool suite for actual device implementation.

5.2 MODELSIM

ModelSim® PE, our entry-level simulator, offers VHDL, Verilog, or mixed-language simulation. Coupled with the most popular HDL debugging capabilities in the industry, ModelSim PE is known for delivering high performance, ease of use, and outstanding product support.

ModelSim PE fully supports the VHDL and Verilog language standards. You can simulate behavioral, RTL, and gate-level code separately or simultaneously. ModelSim PE also supports all ASIC and FPGA libraries, ensuring accurate timing simulations. ModelSim PE provides partial support for VHDL 2008.

CHAPTER 6

SIMULATION RESULTS

6.1 DESIGN IMPLEMENTATION USING MULTIPUMPING

- MULTI-PUMPING technique allows multiple operations to be performed by a single functional unit in one clock cycle.
- This approach operates functional units at a higher frequency than the surrounding system logic, typically 2×, allowing multiple computations to complete in a single system cycle.
- Multi-pumping approach is useful for achieving area reductions of resource sharing, with considerably less negative impact to circuit performance.

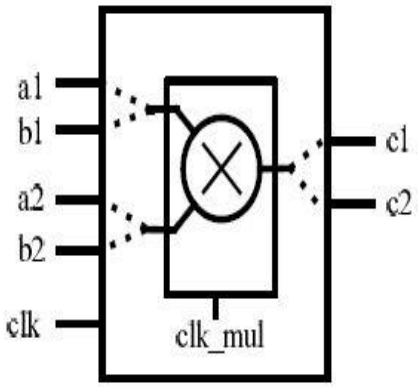

Fig 6.1 Multipumping mechanism

This implementation is done with FPGA device and parameters as specified below:

TARGET DEVICE: xc6slx45t-3fgg484 (Spartan 6)

BRAM MEMORY: 256k

MEMORY DEPTH: 32k

INPUT : Two write address [7:0]
Two write data[31:0]
Four read address[7:0]

OUTPUT : Four read data [31:0]

The Multipumping memory 2W4R architecture design mainly implemented for improving the speed performance internally through running the clock at a multiple of

external clock to provide the illusion of a multiple of the actual number of ports and allowing a designer to trade speed for area reduction.

6.1.1 2W4R LOGICAL MODULE TEST BENCH WAVEFORM

This implementation is done with FPGA device and parameters are as specified below:

The target device is of type xc6slx45t-3fgg484 (Xilinx Spartan 6) and BRAM memory is 256K with memory depth of 32K. The input signals are two write address[7:0], two write data[31:0], four read address[7:0]and output is four read data [31:0].The input data is taken in 32 bit data and input address is a 8 bit data which is used to represent the specific location to which input data as to be saved**.** The output data is taken in 32 bit data and output address is a 8 bit data which is used to represent the specific location to which data as to be accessed.

As shown in the Fig. 6.2 base_clock represent the system clock and clock represent internal clock which runs twice that of system clock. Write_address_0 and write_address_1 are two write address ports. Write_data_0 and write_data_1 are two data write ports. Four address ports are used to read data from memory through read data ports of size 32 bits.

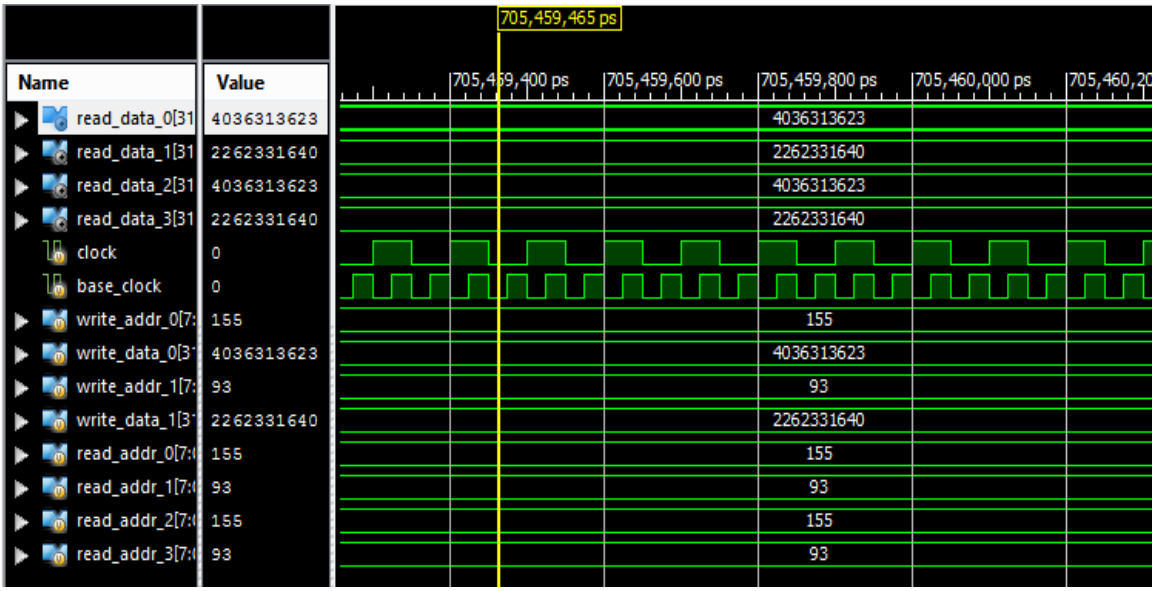

Fig 6.2 2W4R logical module test bench waveform

During read/write operation, writing a data to the address 155 and 93 through write ports require two clock cycles but by using Multipumping approach it can be done with a single clock cycle which can increases the speed of data accessing process. It is applicable for all other concurrent read and write operations also.

The main idea behind multipumping is to *time multiplex* the ports to perform read or write operations.an system clock and an internal clock, which is an integral multiple of the system clock. Both the clocks have to be synchronous and in phase with each other so for this on-chip PLLs used to generate such a relationship.

The Multipumping approach also uses only less device utilization which could be useful for BRAM optimization.

Table 6.1. Synthesis Report

Device utilization summary(estimated values)			
Logic utilization	**used**	**Available**	**Utilization %**
Number of slice registers	**85**	**54576**	**0.155**
Number of slice LUTs	**35**	**27288**	**0.128**
Number of fully used LUT-FF pairs	**20**	**100**	**20**
Number of bonded IOBs	**242**	**296**	**81.7**
Number of block RAM/FIFO	**2**	**116**	**1.72**
Number of BUFG/BUFGCTRLS	**1**	**16**	**6.25**

The Synthesis report shows the memory and device utility as shown in Table 6.1 which specifies number of slice LUTs used null and number of Block RAM are used 3% number of fully used LUT-FFT pairs are used 20% which is quite less compared to other architecture.

6.2 DESIGN IMPLEMENTATION USING LIVE VALUE TABLE

- The live value table (LVT) multi-ported memory design allows the implementation of a memory with more than one write port to be based on BRAMs
- The basic idea of an LVT design is to augment a banked design with the ability to connect each read port to the most-recently written bank for a given memory location.
- Live Value Table is implemented to keep track of where the latest value of a data is stored.

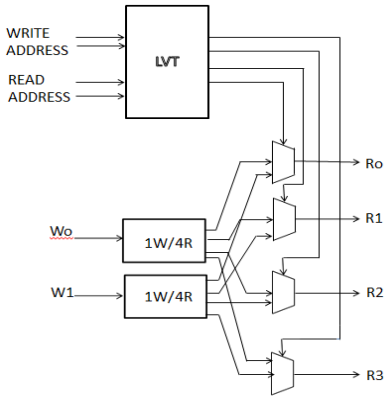

Fig 6.3 LVT mechanism

This implementation is done with FPGA device and parameters as specified below

TARGET DEVICE : xc6slx45t-3fgg484 (spartan 6)

BRAM MEMORY : 256k

MEMORY DEPTH: 32k

INPUT : Two write address [7:0]

Two write data[31:0]

Four read address[7:0]

OUTPUT : Four read data [31:0]

6.2.1 2W4R LOGICAL MODULE TEST BENCH WAVEFORM

The Live Value Table approach improves the speed performance of memory access. This implementation is done as shown in the Fig, 6.3. The design implementation for 2W/4R ports in which write_address_0 [7:0],write_address_1 [7:0] are used to specify the memory address to which the data as to be stored. Write_data_0 [0:31],write_data_1[0:31] are used to denote the data that is written through to the specific location ports. Four read address ports are used to retrieve data from specified address through the help of LVT tracks.Read_0[0:0], read_1[0:0], read_2[0:0],read_3[0:0] in the fig specify the bank identification number to identify the correct BRAM that stores the latest data. which data is written or accessed. As this approach follows replication and bank division process for data write. Data can be accessed through any of the four ports. if data as to be fetched from a memory address 170 it initially access the data through first port which will be mentioned in

live value table by 0,so when same data is accessed by other memory ports at the same time it look at the LVT and access the data for other port which remains idle.Read_port_0_0 [0:31], read_port_0_1 [0:31] ,read_port_0_2 [0:31],read_port_0_3 [0:31] in used to specify the sub-block in BRAM to which data is written. This approach increases the data accessing speed but it requires large device utilization. The BRAM usage is increased compared to Multipumping approach.

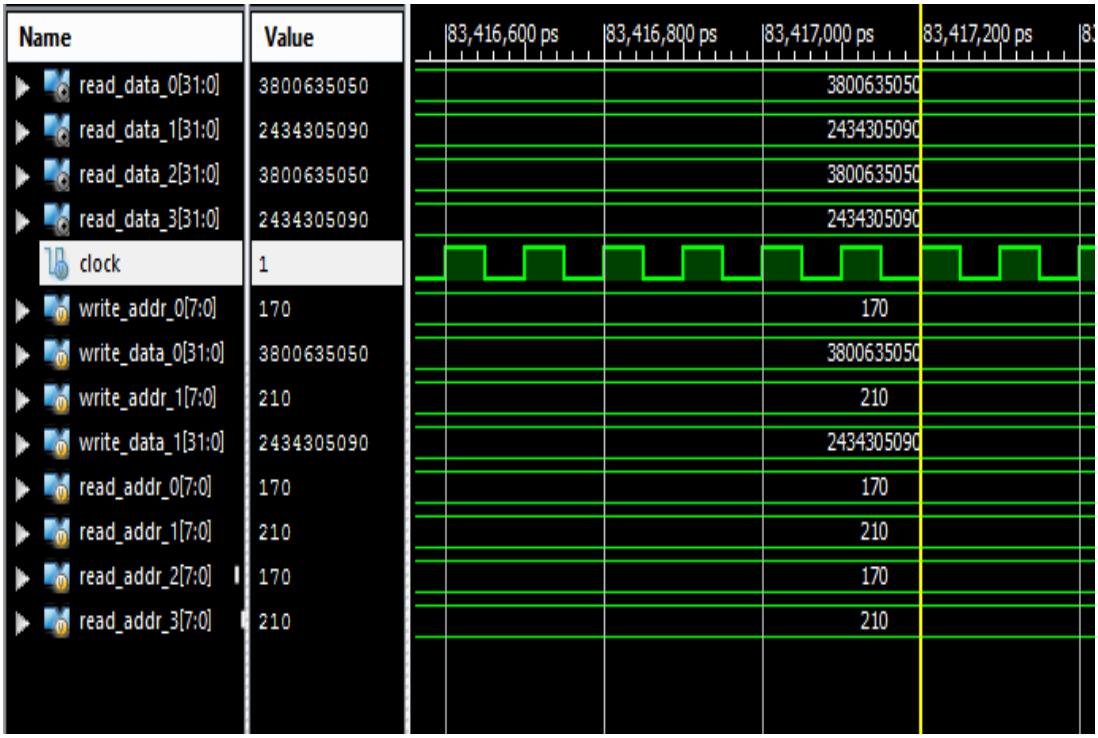

Fig 6.4 2W4R logical module test bench waveform

Table 6.2 Synthesis Report

Device utilization summary (estimated values)			% Utilization
Logic utilization	**Used**	**Available**	
Number of slice registers	**260**	**54576**	**0.47**
Number of slice LUTs	**992**	**27288**	**3.63**
Number of fully used LUT-FF pairs	**256**	**996**	**25.70**
Number of bonded IOBs	**241**	**296**	**81.41**
Number of block RAM/FIFO	**4**	**116**	**3.44**
Number of BUFG/BUFGCTRLS	**1**	**16**	**6.25**

The Synthesis report shows the memory and device utility as shown in table 6.2

Comparing PMP Approach LVT Requires Large Area And BRAM Usage.

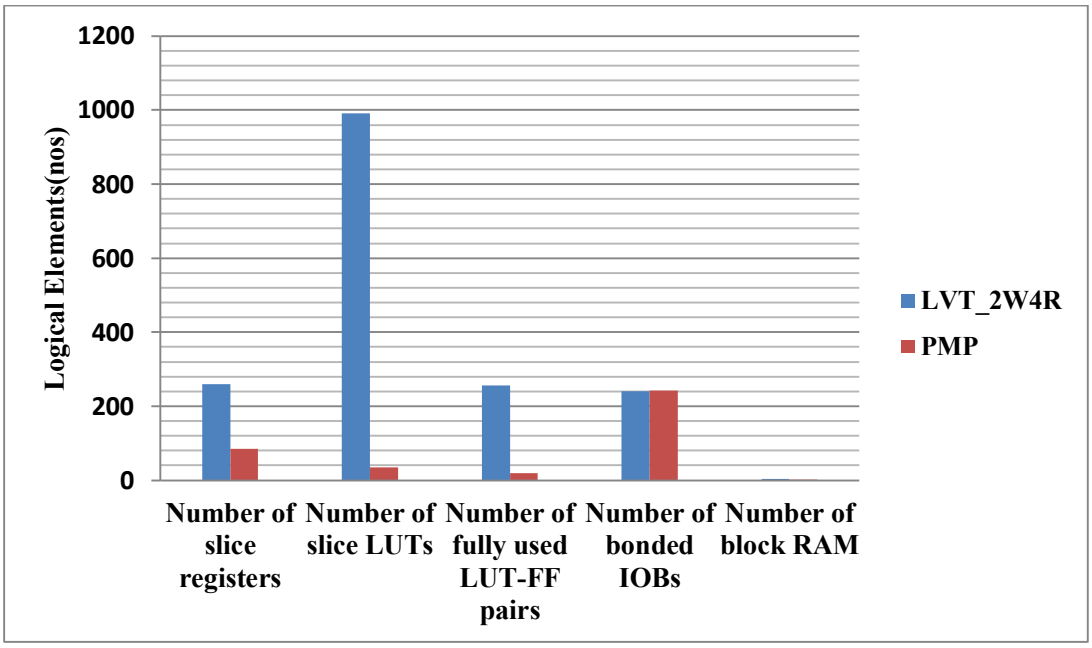

Fig. 6.5. Utilization comparison of PMP_2W4R and LVT_2W4R

The comparison chart shown in fig 6.5 implies that PMP approach utilizes the logical element efficiently compared to LVT approach.

Table 6.3. Number of frequency used for different Memory depth.

Design Arch.	Frequency (Mhz)				
	32 bits	64 bits	128 bits	256 Bits	512 bits
LVT_2W4R	290	285	255	225	175
PMP_2W4R	158	158	158	158	158

Table 6.4. Maximum logical elements for different Memory depth

Design Arch.	No of logical elements				
	32 bits	64 bits	128 bits	256 bits	512 Bits
LVT_2W4R	400	500	750	1350	2550
PMP_2W4R	60	60	60	60	60

The comparison result for various memory depth from 32 bits to 512 bits in Table 6.3 and Table 6.4 shows that the frequency remains almost a constant at about 158 MHz and the area averages out to 60 LEs and 2 BRAMs usage for PMP approach.

CHAPTER 7

MODIFIED LIVE VALUE TABLE ARCHITECTURE

From the previous chapters, we have seen that multipumping is one of the ways to increase the number of ports for memory by saving on area. In this chapter we look into the details of Multipumping, give a brief background about the pure-multipumped memory implemented by LaForest et al. in [2] and describe how we have used this pure-multipumped memory to build smaller and faster non-multipumped LVT memories.

The main idea behind multipumping is to time multiplex the ports to perform read or write operations. In order to do this, multipumping uses two clocks: an external clock and an internal clock, which is an integral multiple of the external clock. Both the clocks have to be synchronous and in phase with each other. Any one of the on-chip PLLs can be used to generate such a relationship. Since memory ports are multiplexed over time, we need to ensure that the order of reads and writes is correct.

In order to avoid write-after-read (WAR) hazard, where the result of the (i+1)st operation is written into a memory location before the ith instruction can read from that memory location, the writes are performed after reads. The read and write operations can be sequenced in such a manner by a Multipumping Controller (MPC), which forcibly synchronizes the system clock and the multiple of the system clock. The MPC generates phase and write- enable signals alongside the internal clock to force the operations controlled by the internal clock to occur with certain phase relationships relative to the operations controlled by the external clock.

7.1 PURE MULTIPUMPING

When block RAMs are used in true dual port mode to build multipumped memories, the technique is called pure multipumping. The fact that the two ports of the BRAM in true dual port mode can be used to perform different combinations of operations--two writes, two reads, or one write and one read--is used to multiplex ports over time. First, the ports are used to read data from all the banks, and then the ports write into the BRAMs simultaneously. Thus the block RAMs are read as a banked memory and written as a replicated memory. This is depicted in Figure 7.1.

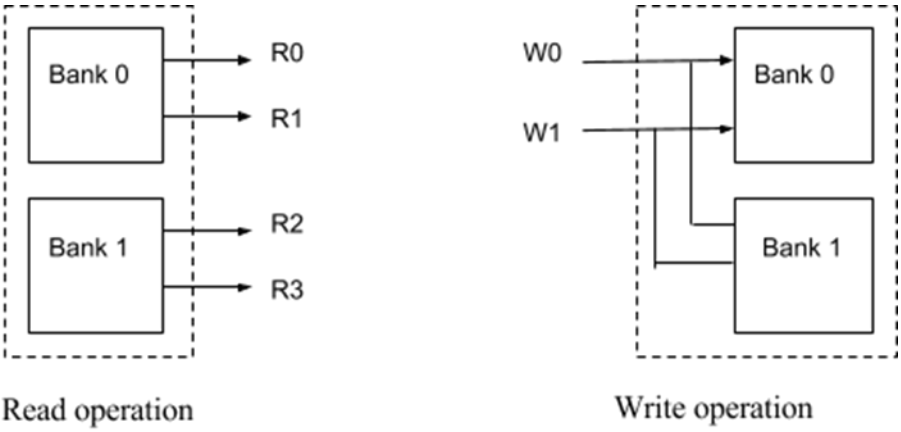

Figure 7.1 Simplified figure showing read and write operations in a 2W/4R pure multipumped memory (With a multipumping factor of 2)

From [2] we find that the number of cycles required for a mW/nR memory to perform all the m writes and n reads is [m/2 + n/2x], where x is the total number of BRAMs used and where p represents the integer ceiling of p. The term m/2 comes from the fact that each write is replicated to all the other BRAMs to maintain data consistency. The term n/2x signifies that each BRAM can support two reads in each cycle because of its true dual port nature and because the write ports write the same data into all the BRAMs. The ceiling function handles cases where the number of internal ports is more than the number of external ports or when the internal ports do not evenly divide into the number of external ports. When the authors of [2] implemented a 2W/4R pure multipumped memory on a Spartan 6 FPGA, with the internal clock frequency twice the external clock frequency (denoted as MP2X in the graph below) they found an interesting fact: for all memory depths there was an almost constant frequency of 279 MHz and an area equivalent to 255 adaptive logic modules We implement 2W/4R pure multipumped memory (PMP_2W4R) on our target Spartan 6 FPGA. As seen previous chapters we get similar results on the Spartan 6.

We vary the memory depth from 32 bits to 512 bits and see that the frequency remains almost a constant at about 151.8 MHz and the area averages out to 60 LEs and 2 BRAMs. We can use this interesting result to build LVT based multi-ported memories that have their banks made up of these 2W/4R pure multipumped (PMP_2W4R) memories. The advantage of this method is that, since the write ports of

each bank are doubled, we need only half the number of BRAMs to support write ports of LVT based memory.

Each bank will now have half the number of BRAMs to support the read ports. As a results in significant area saving. Also, a common MPC is used for all the banks. Since each bank has two write ports and the number of banks is halved, the number of bits stored by the LVT reduces, i.e, the width of the LVT is b = (log2m) -1, where m is the number of write ports.

In the next section we discuss how we implement LVT memories based on 2W4R pure-multipumped memory.

7.2 IMPLEMENTATION

In contrast to the old LVT-based memory, our LVT-based memory with pure-multipumped banks has banks of bank memory and not 1W/nR memory. The banks are made up of 2W/4R memory but the number of read ports for each bank can be extended just by replicating the 2W/4R memory. In the old LVT-based memory, each write port had its own set of banks. In our memory, since each bank has 2 write ports, the number of banks is halved and thus the number of bits needed to be stored by the LVT reduces by (log2m)-1.

This is significant since typically LVT is 3 bits wide or less. Reduction in the number of banks also implies smaller multiplexers at the read ports. Although the individual banks are pure-multipumped using a common MPC, the memory as a whole is not multipumped. Thus there is no need for additional registers to hold the temporary states. Altogether, this results in significant area reduction. Since the banks need to be pure-multipumped, the memory still operates on two clocks, one a system clock and another a multiple of the system clock, with the internal operations controlled by the internal clock as explained in the beginning of this chapter.

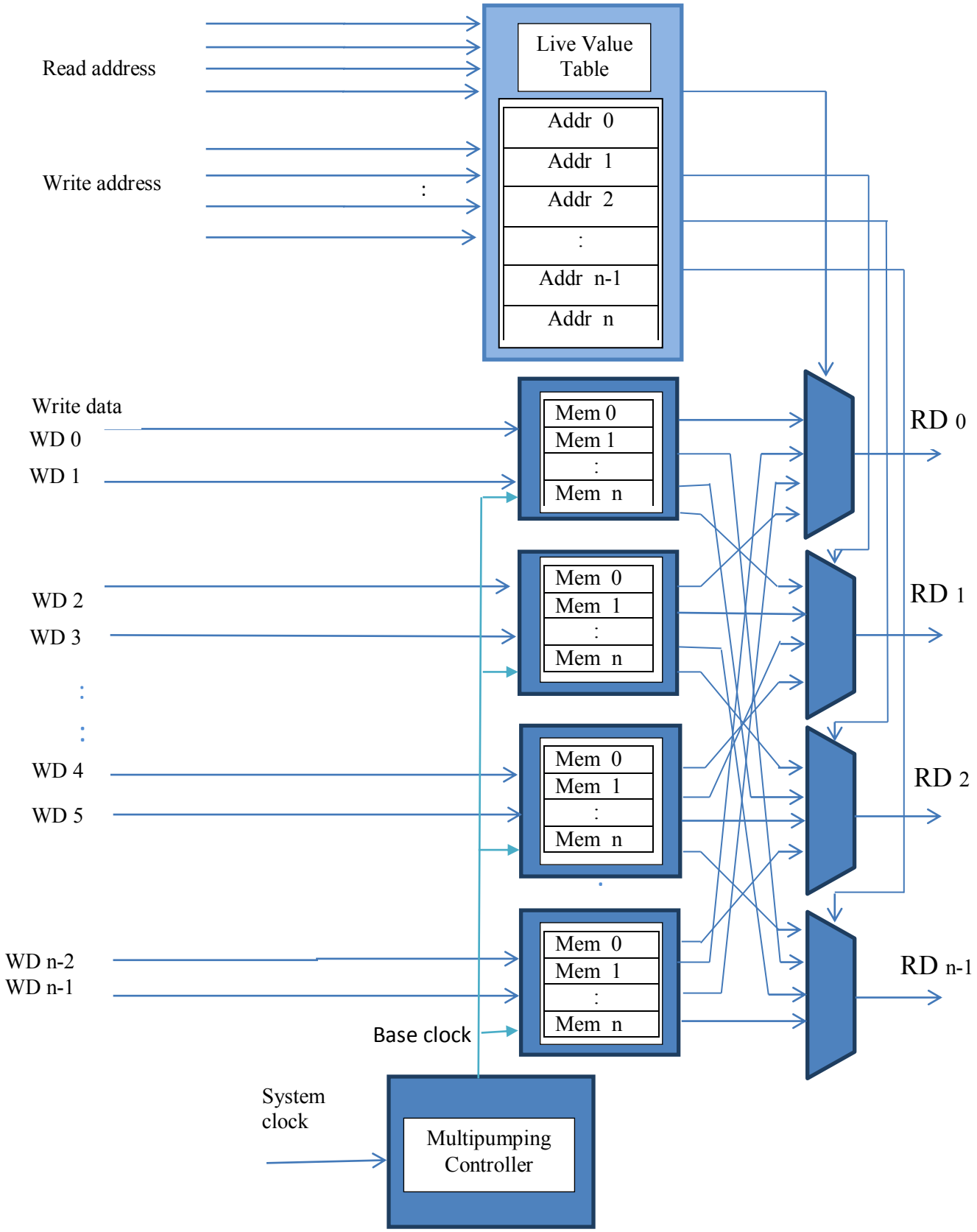

Fig 7.2 Generalized design for LVT-based Modified LVT Architecture with mW/nR ports.

7.3 OPERATION

We illustrate the operation of a 4W/8R memory in Figure 7.3.

In this example, Bank0 and Bank1 each have 2 write ports W0, W1 and W2, W3 respectively. Each bank has 8 read ports. There are also 8 multiplexers which choose the read data from the bank which has the most recently written value for that address. The LVT has 4 write ports and 8 read ports. Depth of the LVT is the same as the depth of each bank and its width is equal to 1 bit since it has to represent only 2 banks.

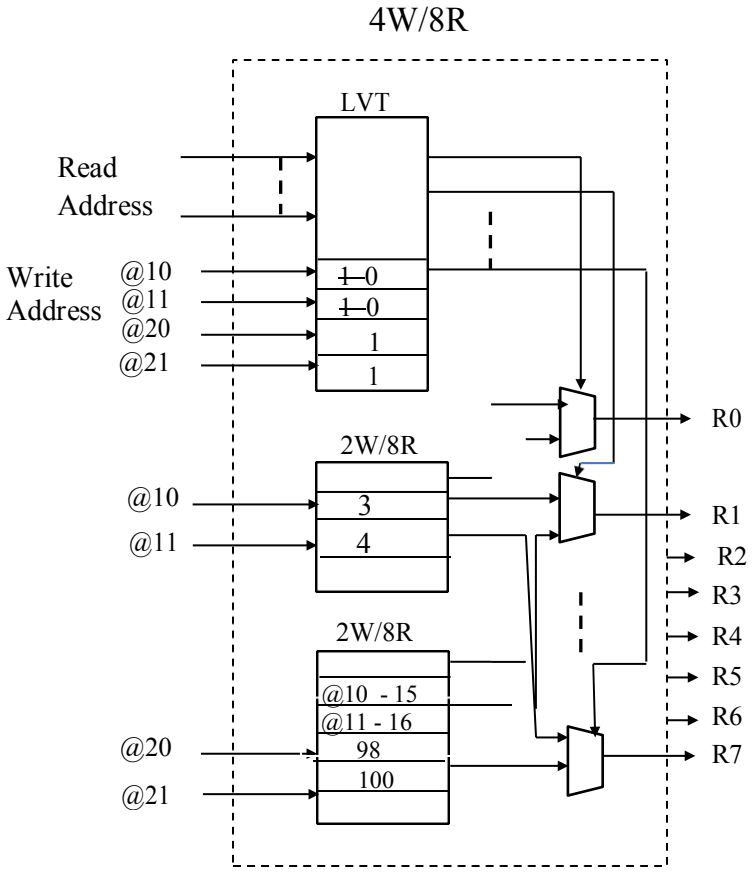

Figure 7.3 write and read operation of a 4W/8R LVT based memory whose banks are built using 2W/4R modified LVT design.

It stores a 0 if Bank0 writes the data and stores a 1 if Bank1 writes the data. Assuming that we do four writes and four reads, the rest of the four read ports are free. Since we have enough ports to service the read and write requests, all the operations happen in one external clock cycle. Since internal clock frequency is twice the external clock frequency, all the reads happen in the first internal clock cycle (first half of external cycle) and the writes are performed in the next internal clock cycle (next half of external

cycle). This can be seen in Figure 7.4

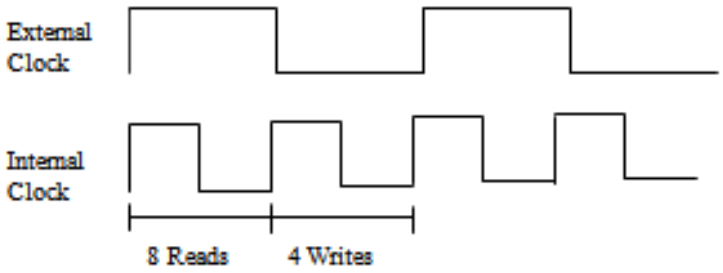

Figure 7.4 Waveforms showing external clock and internal clock used for a 4W/8R LVT_PMP memory.

All the 8 reads happen in the first clock cycle of the internal clock followed by 4 writes which happen in the second clock cycle of the external clock. This entire operation of 4 writes and 8 reads happens in one external clock cycle.

An example write and read operation for memory in Figure 7.3 is discussed below. Initial condition: Assume that there was a previous write to address 10, 11 in Bank1. Location 10 has data 15 and location 11 has value 16.

Figure 7.3 shows the state of the memory after four writes are performed. Ports W0 and W1 write data 3, 4 to address 10, 11 in Bank0. Ports W2 and W3 write data 98, 100 to address 20, 21 in Bank1. The LVT simultaneously updates its address locations with the bank numbers. Address 10 and 11 store a 0 whereas address 20 and 21 store a 1 Output of the read operation: port0, port1, port5, port6 read from addresses 10, 11, 20, 21 respectively. Though both the banks have data stored in address locations 10 and 11, LVT would have stored the bank number of the bank that most recently stored that value, in this case Bank1. The output of the read operation will be 3, 4, 5, 6.

7.4 AREA CONSUMPTION

When compared to the old LVT-based design our modified design clearly performs better in terms of area. We can see this in the graphs shown below. We compare old LVT-based memory (LVT) and LVT memory based on pure-multipumped banks.

Comparison of 2W4R for various design architecture

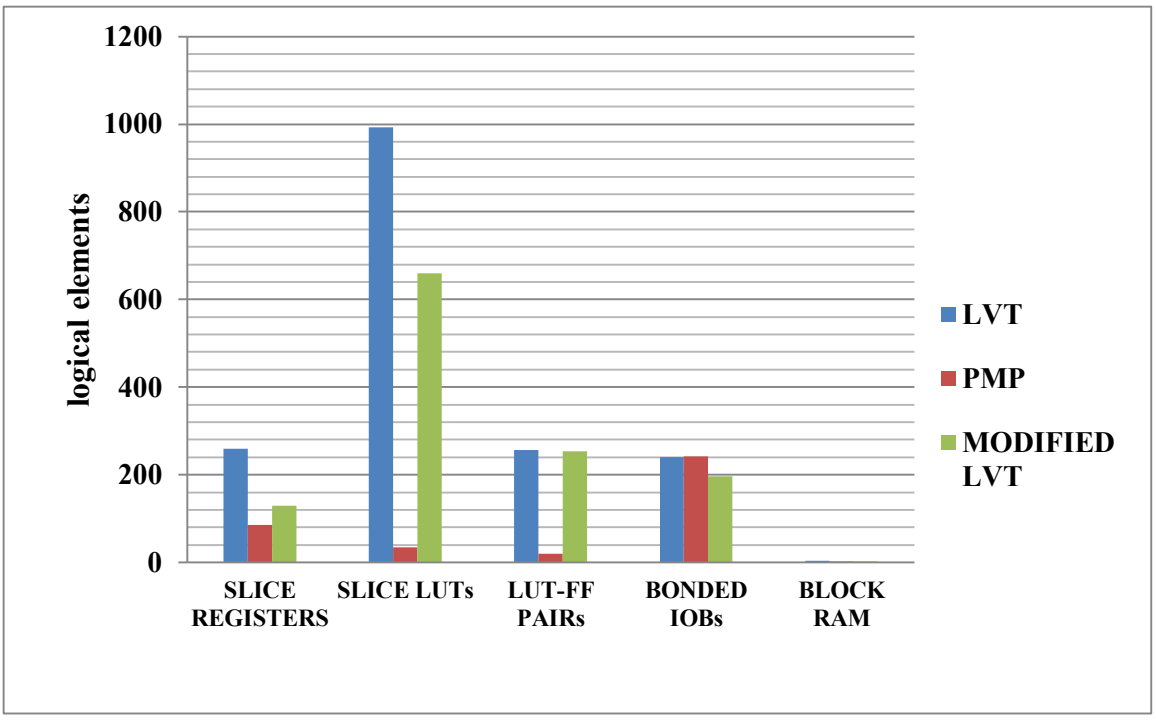

Fig 7.5. Area comparison in terms of number of LEs utilized by LVT_2W4R and modified_LVT 2W4R..

There is an average 67% decrease in the number of LEs used. For example, for a 256 deep memory LVT_4W8R uses 960 LEs and 2 BRAMs whereas LVT_PMP_4W8R uses 620 LEs and only 2 BRAMs.

7.5 SPEED

From [2] we can see that multi-pumping impacts the operating frequency negatively. Though using pure multi-pumped banks to construct a LVT based multi-Ported results in drastic reduction in the area (logic elements and BRAM)as seen from the results in the previous section, it has a harsh impact on the frequency of operation of memory. Since the MOD_LVT is inherently made up of PMP_2W4R memory which has a maximum frequency of operation of 151.8 MHz, the operating frequency of MOD_LVT memories is even lower. For a MOD_LVT 4W8R memory,the frequency of operation ranges from 125 MHz – 75 MHz for memory depth varying from 32 to 512bits. Hence this type of memory is best used for more shallow memories because of the inverse relationship between depth and frequency

CHAPTER 8

MODIFIED LIVE VALUE TABLE ARCHITECTURE WITH POWER OPTIMIZATION

From the previous chapters, we have seen that modified Multipumping design is one of the ways to increase the number of ports for memory by saving on area with increased frequency. In this chapter we look into the architectural design that decrease the power consumption and describe how we have used this clock gating in the modified design to build smaller, faster and less power consuming design.

In order to design, synthesize and implement multi-porting memory with optimized power on FPGA, we are taking spartan 6 Fpga. The example schematic diagram of random access memory that store 16-bit data on 16-bit address is shown fig. That perform read and write depending on value of write enable (we). The enable (en) signal is used to implement clock gating in RAM.

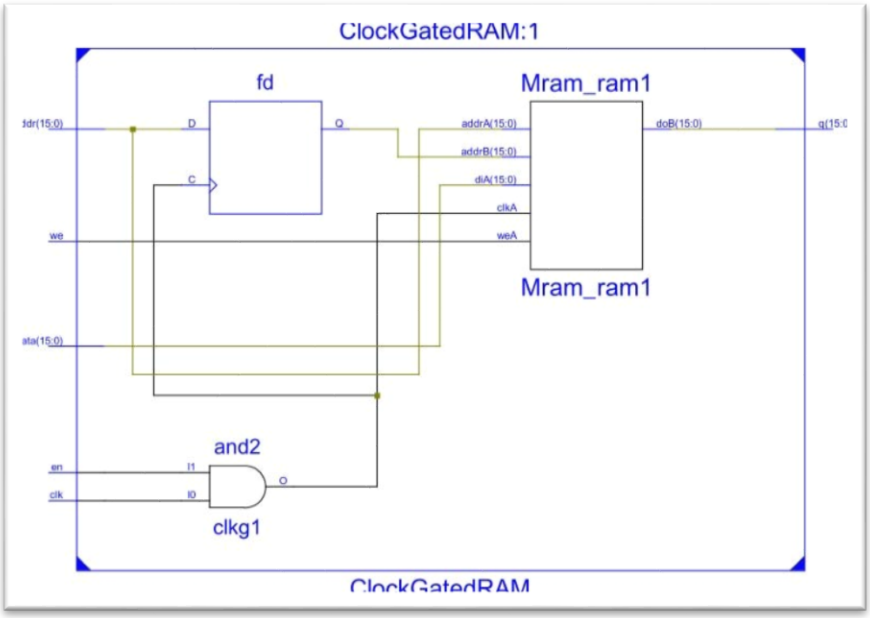

Fig 8.1 Clock Gated Low Power RAM

We know that, Clock gating is a low power technique, which is effective in reduction of dynamic power. Dynamic power is a power consumed by device when device is in on- state. Dynamic power consists of Clock Power, Leakage Power, Signal Power, BRAM Power and IOs Power.

Clock Gating is a technique which decreases clock power but increases Logic Power due to added Logic in Design. Irrespective of increase in number of

Signal and IO buffer due to Clock Gating, there is significant decrease in IO Power and Signal Power due to decrease in number of frequency of device operating. The increase in Logic and Signal power is relatively small in magnitude than decrease in clock power that translates to decrease in overall dynamic power.

Reference [10] describes clock gating and similar techniques of clock gating to turn off inactive module of a system and switch between active and inactive modules. The main ideas discussed in [10] are: clock gating, clock enable, and blocking inputs. After implementation of these techniques on different type of multiplier in [10], Clock Enable is Power hungry techniques and Clock Gating is energy efficient technique. According to Reference [11], clock gating refers to disable the clocks inactive logic module. The work in [11] investigates the various clock gating techniques that is used to optimize power consumption in circuits at register transfer level and deals with different factor involved while applying this power optimization techniques at RTL level.

In our low memory design, we are using the efficient one which is discussed in [11]. Result of [12] indicates that increase in leakage power is proportional to the extra logic used for CG. In [13], a set of energy efficient logical-to-physical RAM mapping algorithms is discussed, to convert user memory specifications to on chip BRAM and LUT RAM like FPGA memory block resources. Power saving algorithms minimizes dynamic power consumption of RAM by taking the most power-efficient choice. In this design, we are using techniques of [14] along with clock gating for memory block mappings and reduce power consumption of our design. According to reference [15], memory merging saves up to 44.32% of block RAM (BRAM). In this design of low power memory, we use intelligent memory merging which is automatically implemented by Xilinx synthesis Technology.

In [15], when any one of module of 16 modules of ALU execute because of clock gating rest 15 modules switched off and reduce (15/16)*100=93.75% power. In this design we are using clock gating for same purpose of low power RAM design and achieving 38.83% and 41.3% power

reduction when our design operate respectively on 1GHz.

8.1 DESIGN IMPLEMENTATION USING MODIFIED LIVE VALUE TABLE WITH CLOCK GATING

In FPGAs it is highly important to optimize power consumption. In order to decrease the power consumption clock gating is introduced into the multiporting memory architectural design. The modified lvt design with clock gating requires some extra logical elements. The comparator circuit is attached to circuit which takes the incoming data as input and compares the new data with the pervious data. if the input data is same then it disable clock for the particular bank and it remains in the same state till the new data enters the circuit this could reduce the dynamic power consumption to a extent

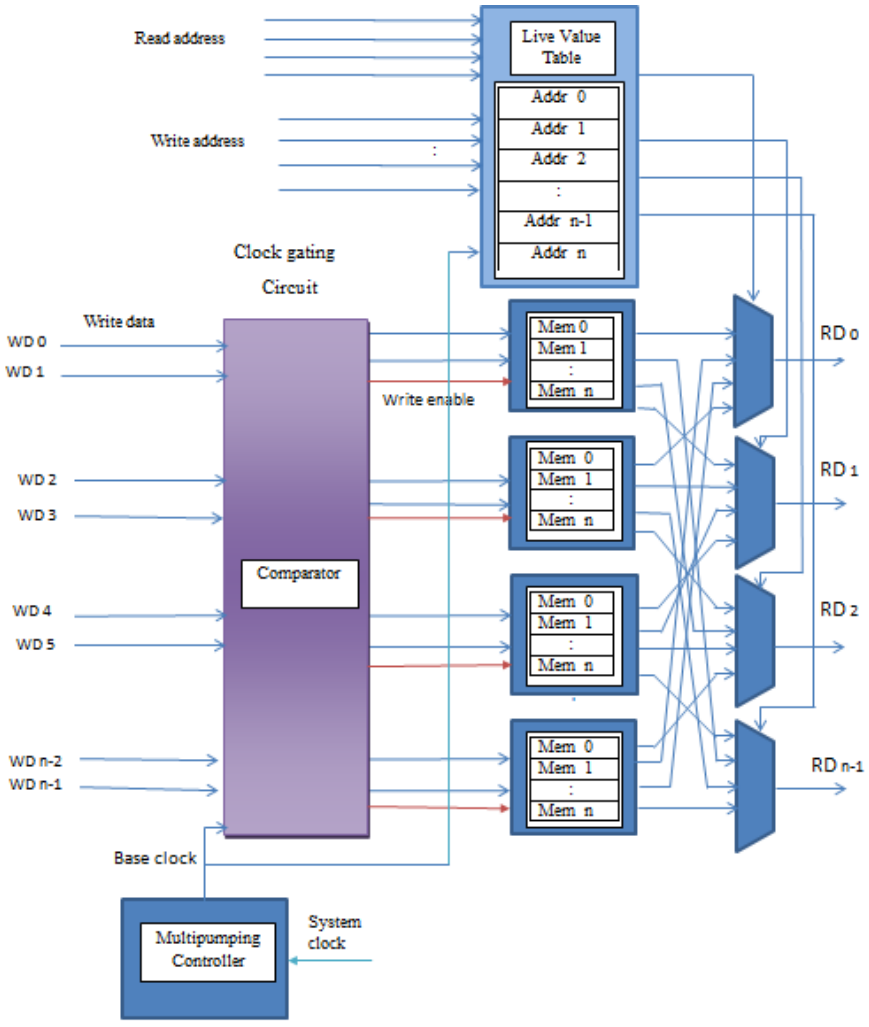

Figure 8.2 Generalized design for Modified LVT architecture with mW/nR ports,

This implementation is done with FPGA device and parameters as specified below

TARGET DEVICE : xc6slx45t-3fgg484 (spartan 6)

MEMORY DEPTH : 32k

INPUT : Two write address [7:0]

Two write data[31:0]

Four read address[7:0]

OUTPUT : Four read data [31:0]

8.1.1 2W8R LOGICAL MODULE TEST BENCH WAVEFORM

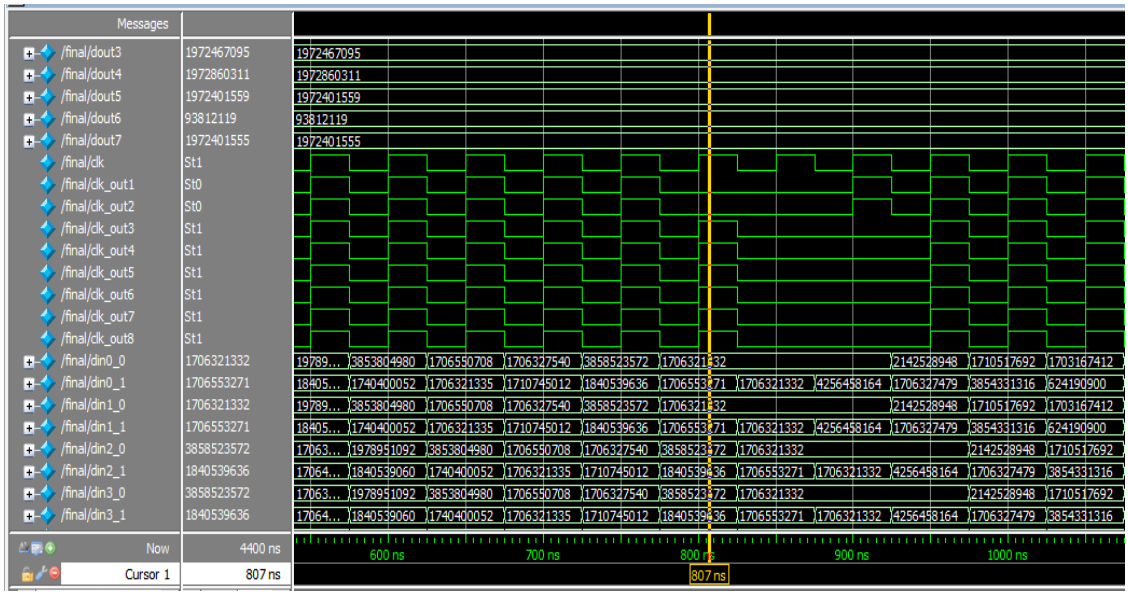

Fig 8.3 2w8r Logical Module Test Bench Waveform

The Live Value Table approach improves the speed performance of memory access. This implementation is done as shown in the Fig, 6.3. The design implementation for 2W/4R ports in which din 0_1 are used to specify the memory address to which the data as to be stored. din 0_0 are used to denote the data that is written through to the specific location ports. Eight read address ports are used to retrieve data from specified address through the help of LVT tracks.

As this approach follows replication and bank division process for data write. Data can be accessed through any of the eight ports.in this when the data 1706321332 enters the write data port it is analyzed with previous input data and compared when the previous data is same as the new data then it disables the clock for the particular bank and in this first cycle data is written normally and in the second clock cycle data is compared and the previous written value is same as the new value

so it disables the clock for that clock cycle on words. The clock remains disabled till the new data enters the write port

8.1.2 RESULTS WITH CLOCK GATING

Technology schematic and RTL schematic of our design RAM is almost same with or without clock gating techniques. Power Consumption and Current Flow change with or without clock gating techniques.

Table 8.1 Power Consumption on 1 GHz Device Operating Frequency with clock gating

Device Operate On 1 GHz Clock Frequency					
ARCHITECTURAL DESIGN	**CLOCK**	**SIGNAL**	**BRAM**	**IOs**	**LEAKAGE**
2W4R	**6.07mW**	**0.57 mW**	**9.6mW**	**1.95mW**	**2.31 mW**
2W6R	**7.08mW**	**1.94 mW**	**9.6mW**	**2.01mW**	**2.64 mW**
2W8R	**7.28mW**	**1.38 mW**	**9.6mW**	**2.3 mW**	**2.84 mW**

Various design architecture which uses clock gating is implemented and analyzed on operating clock frequency of 1GHZ . The various power consuming factors are analyzed like clock, signal ,BRAM, IOs and Leakage power consumption as shown in Table:3.the 2W4R design consumes power of about clock 6.07mW,signal 0.7mW,bram 9.6mW,IOs 1.95mW and leakage current of about 2.31

8.1.3 RESULTS WITHOUT CLOCK GATING

Table 8.2 Power Consumption on 1 GHz Device Operating Frequency without clock gating

Device Operate On 1 GHz Clock Frequency					
ARCHITECTURAL DESIGN	**CLOCK**	**SIGNAL**	**BRAM**	**IOS**	**LEAKAGE**
2W4R	**12.64 mW**	**2.28 mW**	**9.6 mW**	**1.84 mW**	**2.13 mW**
2W6R	**15.2 mW**	**1.98 mW**	**9.6 mW**	**1.91 mW**	**2.53 mW**
2W8R	**14.14m W**	**2.43 mW**	**9.6 mW**	**1.96 mW**	**2.76 mW**

Various design architecture which uses clock gating is implemented and analyzed on operating clock frequency of 1GHZ . The various power consuming factors are analyzed like clock, signal ,BRAM, IOs and Leakage power consumption as shown in Table8.2.the 2W4R design consumes power of about clock 12.64mW,signal 2.28mW,bram 9.6mW,IOs 1.84mW and leakage current of about 2.13mW Comparison chart of Power reduction through power gating

After analyzing the power reduction with and without using clock gating it is found that various architecture like 2W4R , 2W6R and 2W8R as clock power reduction of about 2.62mw,5.06mw and 6.77mw

8.3 POWER REDUCTION ANALYSIS

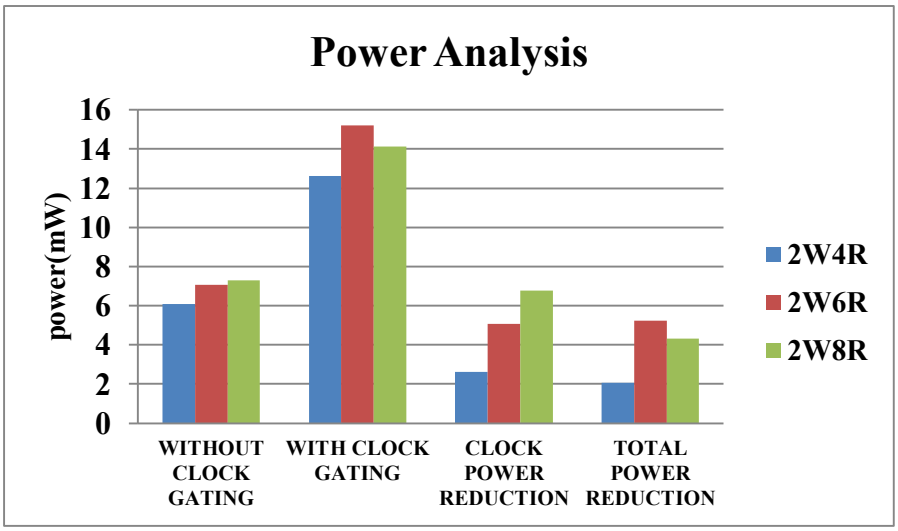

Fig 8.4 power reduction comparison

8.4 OVERALL RESULT ANALYSIS

- After analysis two Multiporting techniques it is concluded that Multipumping technique is suitable for the applications that requires large area but less speed
- The live value table technique is suitable for applications that requires high speed data access but with less area utility
- After Simulation and Analysis of modified live value table architecture with clock gating in various memory depth it is noted that, this architectural design improved the efficiency by increases the frequency with dynamic power reduction

CHAPTER 9
CONCULSION

This project proposed efficient BRAM-based multiported memory designs on FPGAs. The existing design methods require significant amounts of BRAMs to implement a memory module that supports multiple read and write ports. Occupying too many BRAMs for multiported memory could seriously restrict the usage of BRAMs for other parts of a design. This project proposes techniques that can attain efficient multiported memory designs. This project introduces a novel 2W4R memory. By exploiting the 2W4R as the building block, a hierarchical design of nRmW memory is proposed that requires less number of BRAMs than the previous designs based on replication approach. Result conclude that usage of logical elements a by live value table is higher than that of PMP approach. Synthesis report reveals that number of slice registers used is 2.2 % , slice LUTs utilizes about 28.4 % and number of block ram used is twice higher than that of Multipumping approach number of LUT-FF pair usage is 12 % greater for Multipumping compared to live value table. For complex multi-ported designs, the proposed BRAM-efficient approaches can achieve higher clock frequencies by alleviating the complex routing in an FPGA. The new design can be formulated to save more area of BRAMs while at the same time enhance the operating frequency by the combination of both LVT and PMP and also reduce the bonded IOBs used. This project also explains the need of applying an appropriate bank organization in a memory design. It is shown that a multiported design with proper bank organization could achieve a BRAM reduction, a higher frequency, and a lower slice utilization. The results analysed as great potential of introducing a design refinement that could be achieved by optimizing the bank organizations. The new mod_LVT approach is proposed with Clock gating in which clock power consumption is reduced by 48% in 2W4R 46% in 2W6R 51% in 2W8R this could reduce the overall power consumption is 2.05% in 2W4R, 5.05% in 2W6R and 4.319% in 2W8R.

REFERENCES

[1] C. E. LaForest and J. G. Steffan, "**Efficient multi-ported memories for FPGAs**," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2010, pp. 41–50.

[2] C. E. LaForest, M. G. Liu, E. Rapati, and J. G. Steffan, "**Multi-ported memories for FPGAs via XOR**," in *Proc. 20th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2012, pp. 209–218.

[3] C. E. Laforest, Z. Li, T. O'Rourke, M. G. Liu, and J. G. Steffan, "**Composing multi-ported memories on FPGAs**," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 3, Aug. 2014, Art. no. 16.

[4] Xilinx. *7 Series FPGAs Configurable Logic Block User Guide*, accessed on May 30,2016.Available:http://www.xilinx.com/support/documentation/user_guides/ug474_ 7Series_CLB.pdf

[5] Xilinx. *Zynq-7000 All Programmable SoC Overview*, released on May 30, 2016. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq- 7000-Overview.pdf

[6] G. A. Malazgirt, H. E. Yantir, A. Yurdakul, and S. Niar, "**Application specific multi-port memory customization in FPGAs**," in *Proc. IEEE Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2014, pp. 1–4.

[7] H. E. Yantir and A. Yurdakul, "**An efficient heterogeneous register file implementation for FPGAs**," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2014, pp. 293–298.

[8] H. E. Yantir, S. Bayar, and A. Yurdakul, "**Efficient implementations of multi-pumped multi-port register files in FPGAs**," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Sep. 2013, pp. 185–192.

[9] J.-L. Lin and B.-C. C. Lai, "**BRAM efficient multi-ported memory on FPGA**,"
in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2015, pp. 1–4.

[10] J. P. Oliver, J. Curto, D. Bouvier, M. Ramos, and E. Boemo, "**Clock gating and clock enable for FPGA power reduction**", *8th Southern Conference on*

Programmable Logic (SPL), pp. 1-5, 2012.

[11] J. Shinde, and S. S. Salankar, "**Clock gating-A power optimizing technique for VLSI circuits**", *Annual IEEE India Conference (INDICON)*, pp. 1-4, 2011.

[12] J. Castro, P. Parra, and A. J. Acosta, "**Optimization of clock-gating structures for low-leakage high-performance applications**", *Proceedings of IEEE International Symposium on Efficient Embedded Computing*, pp. 3220-3223, 2010.

[13] B. Pandey; M. Pattanaik, "**Clock Gating Aware Low Power ALU Design and Implementation on FPGA**", International Journal of Future Computer and Communication (IJFCC), Vol.3, ISSN: 2010-3751, 2013. (In Press)

[14] Ortega-Cisneros; J.J. Raygoza-Panduro; J. Suardiaz Muro; E. Boemo, "**Rapid prototyping of a self-timed ALU with FPGAs**" International Conference on Reconfigurable Computing and FPGAs,pp. 26-33, 2012

[15] Bishwajeet Pandey, Deepa Singh, Manisha Pattanaik, "**Low Power Design of Random Access Memory and Implementation on FPGA**", Journal of Automation and Control Engineering, ISSN:2301-3702

LIST OF PUBLICATION

- Presented a paper titled **"**Multi-Ported Memories for FPGA with Optimum Usage of BRAM**"**at International Conference On Science, Technology, Engineering And Management (ICSTEM'17) at Kalaignar Karunanidhi Institute Of Technology.