I, Akshata Muddebihal, hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:
Area Efficient Multi-Ported Memories with Write Conflict Resolution

Student's name: Akshata Muddebihal

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.
Committee member: Wen Ben Jone, Ph.D.
Committee member: George Purdy, Ph.D.
Area Efficient Multi-Ported Memories with Write Conflict Resolution

A thesis submitted to the
Graduate School of University of Cincinnati
in partial fulfillment of the requirements for the degree of
Master of Science
in the Department of Electrical Engineering
of the College of Engineering and Applied Sciences

April 25, 2014

by

Akshata Muddebihal

B.E, Visvesvaraya Technological University, India, 2009

Thesis Advisor and Committee Chair: Dr. Carla Purdy
Abstract

Computing systems these days are being designed to take advantage of parallelism in order to increase computational speed. This gives rise to the need for both hardware and software to be designed to achieve as much parallelism as possible. Issue width in the processors is being increased to support Instruction Level Parallelism. To gain performance increase from this, corresponding changes have to be made in the processor architecture as well. Wide issue machines like superscalar processors require an increase in the number of ports in the data path memory. In this thesis we focus on enhancing parallelism in memory by making it multi-ported.

FPGAs offer an attractive platform to build multi-ported memories. They have dual ported embedded memory blocks of fixed number and size which can be used as building blocks for memories with more ports. Various authors previously have constructed multi-ported memory with an arbitrary number of ports using the Live Value Table (LVT) technique. In this work, we build better memories in terms of area by modifying the LVT technique. As an example, we build a 32-bit wide, 256-bit deep 4W/8R memory which consumes 43% fewer logic elements and 75% fewer block RAMs as compared to the original LVT technique.

The effect of different write port-to-read port ratios is studied apart from the common case where read ports are double the number of write ports.

We also add an extra feature to memory. A write conflict detection and resolution circuit is built entirely in hardware to relieve the software programmer of the task of scheduling the writes using software. Two designs for this circuit are implemented and compared.
Acknowledgements

Firstly, I want to thank Dr. Carla Purdy for the support and guidance she gave me throughout the course of my thesis. The projects on FGPAs in the Embedded Systems course, taught by her, interested me in FPGAs and led me to choose a research topic on FPGAs. She has been very communicative despite the distance after my relocation to another city. I would also sincerely like to thank Dr. Wen Ben Jone and Dr. George Purdy for being a part of my thesis defense committee and reviewing my thesis.

I would like to thank Charles Eric Laforest for having had so much patience in writing those detailed e-mails answering my queries regarding his work on multi-ported memories. My understanding on his work would not have been to this level without his guidance. Thank you so much Eric.

Thanks to my parents for supporting me in every walk of my life. Thanks to all my friends for having made Cincinnati feel like home.

Finally, this would not have been possible without the constant support and encouragement of my husband, Nitin Puranik. He has made me a better person.
## Contents

1. Introduction ........................................................................................................................................... 1  
   1.1 Previous methods to implement multi-ported memory ................................................................. 2  
   1.2 Goals of thesis ................................................................................................................................. 7  
   1.3 Organization of thesis ..................................................................................................................... 8  
2. Background ........................................................................................................................................... 9  
   2.1 Cyclone IV architecture .................................................................................................................. 16  
       2.1.1 Logic Elements (LE)............................................................................................................. 16  
       2.1.2 Block RAM (BRAM) ............................................................................................................. 16  
       2.1.3 Delay locked loop and phase locked loop ............................................................................. 17  
3. Experimental setup ............................................................................................................................. 18  
   3.1 Memory design ............................................................................................................................... 19  
   3.2 Experimental setup ......................................................................................................................... 20  
   3.3 Area measurement .......................................................................................................................... 23  
4. Write-to-read port ratio ....................................................................................................................... 24  
   4.1 Implementation ............................................................................................................................... 24  
   4.2 Results ............................................................................................................................................ 25  
5. Multipumping ....................................................................................................................................... 27  
   5.1 Pure multipumping .......................................................................................................................... 28  
   5.2 Implementation ............................................................................................................................... 32  
   5.3 Operation ......................................................................................................................................... 34  
   5.4 Area consumption ........................................................................................................................... 36  
6. Write conflict resolution ....................................................................................................................... 41  
   6.1 CMOS v/s FPGA .............................................................................................................................. 42  
   6.2 Nature of conflict ............................................................................................................................. 43  
   6.3 Address comparator ......................................................................................................................... 44  
   6.4 Priority method to handle write conflicts ....................................................................................... 47  
       6.4.1 Operation ............................................................................................................................... 47  
       6.4.2 Performance ........................................................................................................................... 49  
   6.5 Write conflict resolution by adding phantom bank ........................................................................ 54
6.5.1 Operation ........................................................................................................57
6.5.2 Performance ....................................................................................................58
7. Conclusions and future work .............................................................................59
   7.1 Conclusions .......................................................................................................59
   7.2 Future work ......................................................................................................61
8. References ............................................................................................................64
List of figures

1.1 Traditional techniques of implementing multi-ported memories in FPGAs..........................4
1.2 Generalised mW/nR memory constructed using LVT technique......................................5

2.1 Timing diagram of modified clocking scheme for register file access...............................11
2.2 Waveform of BUSY arbitration cycle..............................................................................14
2.3: Key architectural features of Cyclone IV FPGA..............................................................17

3.2.1 Proper and improper placement of a 4W/8R memory..................................................22

4.2a Plot of Fmax versus the area for different write-to-read port ratios...............................25
4.2b BRAM utilization for different write-to-read port ratios..................................................26

5.1.1 Read and write operations in a 2W/4R pure multipumped memory ................................28
5.1.2 Speed and area for M9K-based 2W/4R multipumped memory implemented on
    StratixIII....................................................................................................................29
5.1.3 Speed (Unrestricted Fmax) and area for M9K-based 2W/4R multipumped
    memory implemented on CycloneIV.........................................................................30
5.1.4 Speed (Restricted Fmax) and area for M9K-based 2W/4R multipumped
    memory implemented on StratixIII FPGA..................................................................31
5.2 Generalized mW/nR LVT-based multi-ported memory, built from pure-multipumped
    banks............................................................................................................................33
5.3.1 Write and read operation of a 4W/8R LVT_PMP memory.............................................34
5.3.2: Waveforms showing external clock and internal clock used for a
    4W/8R LVT_PMP memory..........................................................................................35
5.4.1: Plot of speed and area of 4W/8R LVT_PMP memory..................................................37
5.4.2: Plot of speed and area of M9K based 8W/16R LVT_PMP memory.........................38
5.4.3: Comparison of BRAM utilization between the old LVT and LVT_PMP...............39

6.3.1: Address comparator for a memory with 2 write ports......................................44
6.3.2: Address comparator for a memory with 3 write ports......................................45
6.3.3: Address comparator for a memory with 4 write ports......................................46
6.4 An mW/nR LVT based memory with write conflict resolution – Priority Method........48
6.4.2.1 Plot of Fmax versus number of logic elements used for a 2W/4R memory
with and without conflict resolution circuitry..........................................................50
6.4.2.2 BRAM consumption of 2W/4R memory with and without conflict
resolution circuitry....................................................................................................50
6.4.2.3 Plot of Fmax versus versus number of logic elements used for a 3W/6R memory
with and without conflict resolution circuitry..........................................................52
6.4.2.4: Plot of number of BRAM’s used for a 3W/6R memory with and without
conflict resolution circuitry.......................................................................................52
6.4.2.5 Plot of Fmax versus number of logic elements used for a 4W/8R memory
with and without conflict resolution circuitry..........................................................53
6.4.2.6 Plot of number of BRAM’s used for a 4W/8R memory with and without conflict......54
6.5 mW/nR LVT based memory with write conflict resolution circuitry –
Phantom bank method................................................................................................56

7.2.1 mW/nR memory with banks made up of both 1W/nR and 2W/4R pure multipumped memory........................................................................................................62
Chapter 1

Introduction

The need for processing speed today is compelling researchers to improve the processor architecture. One of the components that has direct impact on the computational speed is the memory. Different types of memories (shared and distributed), different levels of memory (L0, L1, L2 etc.) have been explored in detail. One more aspect of the memory that has a lot of scope for modification is adding multiple ports to the memory. Unlike a single ported memory which can either read or write through one port at a given time, a multi-ported memory has many ports through which simultaneous reads and writes can be performed. The goal of making the memory multi-ported is to avoid memory serialization and to reduce the wait states.

In applications such as media processing, DSP processors, high performance processors, etc., where large blocks of data need to be transferred, having multi-ported memories speeds up memory access. Superscalar architectures and vector processors by default need multi-ported memories in their data paths to support large amounts of instruction level parallelism. Multi-ported FIFOs in communication processors are needed to store and move the data faster.

The use of multi-ported memories in ASIC devices is uncommon because of the large area cost and the increase in access time to the memory [1]. But in FPGAs there is a fixed number of embedded memory blocks available. These on chip memory blocks are generally dual-ported and can be efficiently used to build a larger memory which has many ports.
1.1 Previous methods to implement multi-ported memory

FPGAs have embedded memory called Block RAMs (BRAMs) of fixed size and number. These are divided into blocks and each BRAM is dual ported. Researchers have implemented multi-ported memories using these two ported BRAMs.

Previously three methods were commonly used to construct multi-ported memories [2]:

1) Replication: the RAM blocks are replicated so that we have n read ports. All the write ports of the RAM blocks are connected to maintain data consistency. All the block RAMs hold the same data. This method is good if we want multiple read ports but we have the disadvantage of having only 1 write port. This is shown in Figure 1.1a. For example, consider a 1W/1R memory. To construct a 1W/4R memory we must replicate the 1W/1R memory block 4 times. All the four blocks have the same data.

2) Banking: from Figure 1.1b we can see that the memory is divided into banks and that each bank has a fixed number of ports. Data in each bank is mutually exclusive. Hence when we read or write, each port is limited to read or write to the address range of that particular bank. This configuration also doesn’t represent a true multi-ported memory. For example, consider a 1Kb memory split into four banks: B0 [R0 - R255], B1 [R256 - R511], B2 [R512 - R757], B3 [R758 - R1023], where each bank has 1W/1R port. The memory now has 4W/4R ports but each individual port can access only its bank. Write port 0 and read port 0 can access only B0, write port 1 and read port 1 can access only B1 etc.
3) Multipumping: this uses a modified clocking scheme where the system has two clocks: a base clock and an internal clock which is an integer multiple number of times faster than the base clock. The memory operates at the frequency of the internal clock. Hence it appears to us that the memory ports increase by a factor equal to the multipumping factor. For example, for memory having 1W/2R ports, if the internal clock is twice as fast as the external clock, i.e., if the multipumping factor is 2, it appears to us that it is a 2W/4R memory. This is because, in the first clock cycle of the internal clock we can perform one write and two reads and in the second clock cycle, we can perform one more write and two more reads, thus amounting to two writes and four reads in one external clock cycle.

There is a lot of overhead to this method because it needs multiplexers and registers. We can see this in Figure 1.1c. There is a drop in external operating frequency too.
Figure 1.1: Three traditional techniques of implementing multi-ported memories in FPGAs –

(a) Replication, (b) Banking, (c) Multipumping [2].
Using these traditional techniques, LaForest and Steffan have constructed ‘soft multi-ported memories’ which employ the dual ported BRAMs of an FPGA [2]. They have constructed true multi-ported memories having m write ports and n read ports, for m ranging from 1 to 8 and n ranging from 2 to 16, which scale very well in terms of area and performance as compared to the older techniques. The Live Value Table (LVT) technique introduced in [2] uses replicated block RAMs, a lookup table called Live Value Table and multiplexers. The LVT is constructed using
logic elements and it itself is a true multi-ported memory with mW/nR ports. The LVT holds the bank number of the most recently written bank for each individual address. It also directs the reads to the bank which has the most recently written value for that address. This technique gives good area and frequency measures. To further reduce the area, LaForest and Steffan employ multipumping to BRAMs in true dual ported mode. This technique is called pure-multipumping. Pure-multipumping does not require the LVT technique to increase the number of ports but uses just the multipumping to create an illusion that the memory has more ports.

They also show an interesting case where a 2W/4R memory was constructed using pure multipumping. The frequency of operation was almost a constant (279MHz on Stratix III series) and the area usage was also very small, and these values remain almost constant over different memory widths and depths.

If the FPGA fabric had embedded memory blocks with slightly more ports, bigger multi-ported memories could have been constructed with less area cost. But since most of the FPGAs have at most dual ported BRAMs, we can use them in true-dual ported mode, multipump them and use these pure-multipumped memories to construct bigger memories which would save a significant amount of area.

In this thesis we build soft multi-ported memory. We use banks made up of the above mentioned pure-multipumped 2W/4R memory to construct the banks of a LVT-based memory to achieve smaller memories as compared to the previous techniques. The width of the LVT is reduced by one since each bank now inherently has two write ports. The area utilised by the memory is almost halved.
In a multi-ported memory, two or more ports can attempt to write to the same memory address at the same time. Building a write conflict circuitry out of hardware elements relieves the programmer of doing the same job in software. We propose two methods to build write conflict handling circuitry around the memory completely in hardware. One is a conventional method of assigning priority to the ports and another is an unconventional method that uses a “phantom bank” which the conflicting ports are redirected to. By doing so, we are sure about which ports are used when data gets written into the memory.

1.2 Goals of thesis

This thesis aims to improve the existing techniques to build multi-ported memories in FPGAs and adds a new feature to the multi-ported memories. Thus the main goals are:

1) To build smaller and more efficient multi-ported memories by combining the LVT and pure-multipumping technique.

2) To study the effect of different write port to read port ratios.

3) To provide an additional feature to the memory: Write conflict detection and resolution circuit, which is completely implemented in hardware.
1.3 Organization of thesis

A quick description of rest of the chapters in the thesis is given below.

Chapter 2 tells us why FPGAs are preferred over ASICs to build multi-ported memories. Also background on previous methods to build multi-ported memories is discussed. Examples of commercially available multi-ported chips and the mechanisms they employ to detect write conflicts are also discussed.

In chapter 3, the experimental setup is explained. The memory design, FPGA targeted for experiments, tools used and design constraints are explained.

In chapter 4 we study the effect of different write to read port ratios (for LVT based multi-ported memory) other than the commonly used ratio of 1:2. Results in terms of frequency of operation and area occupied are presented.

Chapter 5 gives an introduction to pure multipumping and shows how 2W/4R pure multipumped memories built by LaForrest and Steffan [2] can be used to build area efficient non-multipumped LVT based multi-ported memories. Performance of such memory in terms of area and frequency is reported.

Chapter 6 deals with write conflicts in LVT based multi-ported memory. The nature of write conflicts is studied and two methods: priority method and phantom bank method, to resolve them are introduced.

Finally conclusions and future work are presented in chapter 7.
Chapter 2

Background

FPGAs offer us a lot of flexibility in implementing any logic because of their reconfigurable logic. Because of their facilitation of design reuse, they also provide a smaller production cost for small volumes to develop prototypes etc., FPGAs turn out to be an attractive platform as compared to ASICs. As a result, FPGAs have continued to evolve from having only programmable logic blocks, interconnects and I/O pins to having specialized hardware like embedded memory blocks, DSP blocks, transceivers, multipliers etc., on the FPGA fabric. These days most FPGAs have fixed size low latency on-chip memory that can be used as RAM memory. These are dual-ported. Memories like register files and FIFO buffers need to be multi-ported. Previous work has been done on building memory with more ports using these embedded dual ported RAMs. In this chapter we review the previous work that has been done on building multi-ported memories and describe how a combination of these methods has been used by some researchers to construct better memories.

The data paths of superscalar processors and Very Long Instruction Word (VLIW) processors invariably need register files with more ports to support instruction level parallelism (ILP). ILP is a form of parallelism where non-dependent instructions are overlapped in the pipeline and executed. The instructions can be executed out-of-order which speeds up the execution time. Significant work has been done on building multi-ported register files for soft core superscalar and VLIW processors. Multi-ported register files are needed to support wide instruction issues where the number of instructions processed per cycle is greater than 1. Giving the memory more
ports is one of the ways of solving structural hazards in pipelines. More ports for the memory can be used effectively to access the operands and results. Saghir et al. [3] have proposed a register file with more ports by organizing the BRAMs into banks. There are many such banks and data is duplicated within each bank. Each of the banks has an independent set of read and write ports. They use this register file in the data path of a configurable soft VLIW processor. Though the access latency of the memory and area occupied in the data path are reduced, we cannot perform more than one write to the same block at the same time, nor can the functional units that are connected to different banks issue reads to the same data at the same time.

Jones et al. [4] have designed a FPGA based VLIW processor in which some of the execution units are application specific. To increase the scalability of the processor they suggest a design with more ports to the shared register file. This scalable multi-ported register file can be used to interconnect VLIW processors and is also available to the application specific hardware functions. This custom hardware is accessible just like other software instruction that can execute in parallel, thus eliminating interfacing overhead with rest of the processor. But the disadvantage of using their multi-ported register file is that, as the number of ports scale, the logic resources that are required increase and there is a significant drop in the performance of the processor.

Authors of [5] have proposed a memory structure with two write ports to aid data transfers in high level synthesis (HLS) in FPGAs. HLS is an automated design process that takes in complex algorithmic descriptions of a system and synthesizes that into a digital hardware that then implements that behaviour of the system. The memory has two decoders to facilitate two writes and a write control logic to prevent the ports from writing to the same cell. However, this technique is not scalable in terms of more write ports.
To save area occupied by the memory, the author of [6] has come up with a register file design for a pipelined superscalar processor. [6] proposes a modified clocking scheme to increase the number of ports for the register file. The design uses two consecutive clock cycles as one processor clock. Reads happen on one edge of the clock cycle and all the writes happen on the other edge of the clock cycle. This can be seen in figure 2.1 [6]. Thus we get an illusion that there are more number of physical ports than there actually are. But the drawback of this method is that this clocking scheme must be applied to the entire data path and not just the memory.

![Figure 2.1: Timing diagram of modified clocking scheme with four phases (labelled $\Phi_0$ - $\Phi_3$) for register file access. All the reads happen on the rising edge and writes happen on the falling edge of the clock [6].](image)

Xilinx devices like Virtex, Spartan [7] and Actel’s ProASIC devices [9] have been designed with quad-port memories. These use a modified clock method similar to that in [6]. The ports operate on a clock frequency that is double the processor clock frequency. This doubled clock is phase aligned with the system clock. The disadvantage is that there is overhead of including extra logic and the operating frequency drops to 50% of the original frequency.
More recently LaForest et al. [2] have explored a number of efficient ways of constructing multi-ported memories. They have concluded that building multi-ported memories using only FPGA logic scales poorly in terms of area and operating frequency when the depth of the memory is increased. They have proposed a soft multi-ported memory structure that is built out of dual ported on-chip memory blocks. It is a true multi-ported memory with mW/nR ports. Their method uses replication and banking of BRAMs to increase the number of ports. Replicated BRAMs are divided into banks. Each bank has 1 write port and n read ports. An m write memory will have m banks. This can be seen in Figure 1.2 above. The key to their design is a Live Value Table (LVT) which is also a multi-ported memory built using pure logic elements. The LVT redirects the reads to the appropriate bank based on which bank stores the most recently written value or the live value. This approach scales very well. The authors have also traded frequency with area by adding multipumping, explained in section 1.1, to the LVT technique. It is demonstrated further in this thesis how the LVT method is modified to construct smaller and faster memories.

The LVT technique scales very well for shallow designs. Shallow memories in this context are memories with depth ranging from 32 – 256 bits. However, the amount of FPGA resources used increases and the frequency of operation drops as the memory grows wider. To avoid the overhead of LVT and multiplexing, LaForest et al. [8] have proposed an alternative design, building on the interesting property of XOR, A \( \oplus \) B \( \oplus \) B = A. The XOR method consumes far fewer logic elements than LVT. It again uses the dual ported embedded BRAMs as the basic building elements. Every time we write the contents of a given memory location to a particular bank, we XOR it with the old contents of the same memory location in the other banks. When we read a location, we XOR it with the values in the same location in the other banks to get the
latest written value. This technique works well for deeper memories. However the XOR technique is slower than the LVT technique and consumes a significant on-chip memory.

There are quite a few commercially available dual ported memory chips.[10,11,12,13,14,15] These have simple mechanisms to handle port conflicts. There are three types of port conflicts that can occur. Read-write conflicts occur when one port tries to read and the other tries to write simultaneously. Write-write conflicts arise when both ports try to write to the same location simultaneously. Read-read conflicts arise when both ports try to read from the same location. This last conflict, however, is not really a conflict. All of these chips contain busy logic which signals the hardware that the two ports are accessing a given memory location simultaneously. The chip enable signals and addresses on the two ports are spaced by a time period of $t_{APS}$, where APS stands for arbitration priority set-up time. This is depicted in Figure 2.2. If the requests on the ports are spaced by a time more than $t_{APS}$, the arbiter will serve the request that comes in first. In case the $t_{APS}$ is not satisfied, i.e., if the time period between the requests is less than $t_{APS}$, the BUSY signal will be asserted on one of the two ports, but there is no guarantee of which port it will be asserted on. Thus the BUSY scheme fails under this condition because it does not guarantee data integrity. Some researchers also have added a feature of semaphore logic.[13,14,15] This is useful in a multiprocessor environment where a processor can block other processors from accessing a portion of the dual-ported RAM.
But again if the time period between the requests to the semaphore is less than the semaphore flag contention window, we cannot be sure which request the semaphore will enable.

Conflict detection and handling is necessary to avoid write-after-write and write-after-read hazards. But usually most of the multi-ported memories do not have conflict resolution [17,18,19,20] or have a simple conflict handling scheme as discussed above. Zuo has proposed two algorithms to address the issue of conflicts in multi-ported memories[16]. The multi-ported memory is constructed using a two level hierarchical architecture enabled by FPGA logic resources. Interleaving of 1-port memory cells is used to form an N-port memory. The memory cells are divided into banks. Memory cells in different banks can be accessed simultaneously, while accessing cells from the same bank simultaneously will result in a conflict. The extend port import hierarchy algorithm described in [16] assigns a static priority to the ports according to the value in the port priority register. The priority registers set the port priority as well as the read/write priority. So the port with the highest priority wins during a conflict. During a write-write conflict, the port that loses sends an acknowledgement signal to indicate write failure. The block access control algorithm is used to reduce conflicts when the processors perform a block

Figure 2.2: Waveform of BUSY arbitration cycle controlled by address match timing [14, 15].
read-write. This memory structure works well for a very small number of ports. However, the cell cost increases significantly as the number of ports is increased.

We will see how the LVT design comes with the ease of including conflict resolution circuitry built into it. In the case of a read-write conflict where one port tries to read from a particular memory location as the other port tries to write to it in the same clock cycle, we can have the option of reading either the old contents of the memory from the previous write or reading the new contents being written. The older LVT technique in [2] follows the former approach. However, LaForest et al. in [8] have proposed improvements to the old LVT design wherein they add a forwarding circuitry which bypasses the BRAMs such that if a memory address being written into is read in the same clock cycle, it reads the new write value.

We propose two methods for handling write conflicts for the multi-ported memory proposed in [2]. For a multi-ported memory based on LVT technique, if two or more ports try writing to the same address in the same clock cycle, the respective data gets written in the corresponding banks, since each write port has its own set of banks. But the live value table which is a true multi-ported memory constructed out of logic elements will store an undefined bank number. By adding conflict detection and resolution circuitry, we can be sure which bank number gets stored in the LVT in case of a conflict and hence there is no ambiguity over which bank the data gets read from during subsequent reads to that memory location.
2.1 Cyclone IV architecture

In this thesis we use Altera’s Cyclone IV family of FPGAs. However, the same work can be translated onto FPGAs from other manufacturers as well. Figure 2.3 shows the architectural features of the Cyclone IV FPGA. The features that are relevant to our study are discussed below.

2.1.1 Logic Elements (LE)

LEs are the smallest units of logic in Cyclone IV. These have a four input lookup table, carry chain connection, registers and some additional logic. Though LVT based multi-ported memories with any configuration and any number of ports can be built using LEs, they are slow and occupy a lot of area [2].

2.1.2 Block RAM (BRAM)

Cyclone IV has 6.5 Mbits of on-chip memory arranged in blocks of 9K bits each. These are called M9k block RAMs. BRAMs are dual ported and they have variable port configurations. M9k’s can be configured to be either in simple dual port mode where one port acts as a read port and the other as a write port, or in true dual port mode where both the ports can be used as two read ports or two write ports or one read and one write port. The M9k block can have a configuration (depth x width) of 256 x 32 bits or 512 x 8 bits etc. In the fastest speed grade, C6, the M9k operates at a maximum frequency of 315 MHz. Both the memory ports are synchronous, i.e., both the ports operate at the same clock frequency.
2.1.3 Delay locked loop and phase locked loop

Phase locked loops (PLLs) and delay locked loops (DLLs) are used to multiply the clock frequency with a given phase shift, which helps in synchronization. PLLs are used to generate a clock which is a multiple of the system clock in order to implement multipumping. In this thesis, however, we will not be using these, since we assume all clocks are externally generated.
Chapter 3

Experimental setup

In this thesis we work on improving the LVT technique proposed in [2] to build better multi-ported memories. The structure of LVT based multi-ported memory is as follows. As seen in Figure 1.2, a mW/nR memory has \( m \) banks (\( M_0 - M_{m-1} \)). Each bank is composed of \( n \) replicated BRAMs to facilitate \( n \) reads (\( R_0 - R_{n-1} \)). Thus each bank is a 1W/nR memory. Each write port writes to its own bank so that we have \( m \) concurrent writes. Each read port can read from any of the banks via a multiplexer. Since there are \( m \) write ports and each of the \( m \) ports writes to its set of \( n \) BRAMs, the total number of BRAMs needed for LVT-based design is \( m \times n \). The live value table (LVT) is a true multi-ported memory with \( m \) write and \( n \) read ports implemented only using LEs. The depth of the LVT is the same as the depth of the memory and each LVT location corresponds to a memory location. But the catch is that the LVT, instead of storing the write data into its memory locations, stores the bank number into which the data is written. The width of the LVT is \( \log_2(m) \). The output of the LVT is given as the control input for the mux which selects the correct read data.

Working of LVT based memory: During a write operation the write port updates the new value at the given address in its respective bank. The LVT simultaneously updates the bank number in its corresponding memory location. The bank number that is stored is the same as the port number which wrote into that particular address. During a read operation, the read address is simultaneously sent to all the banks and to the LVT. All the banks return the value at that address.
and the LVT returns the bank number which has the most recently written value for that address, driving the multiplexer of the read port to select the output of the correct bank.

### 3.1 Memory design

All the memories we consider in this thesis are 32 bits wide, since this is the common case in many processors even today and simple to implement. Our building blocks are two port M9k memory blocks. We consider shallow memories with a depth ranging from 32 to 512 bits wide since LaForest *et al.* [8] have inferred that LVT technique works best for shallow memories. Too much of FPGA resource is consumed if the memory is made deeper. We vary the number of write ports from two to eight and number of read ports from four to sixteen. We also evaluate memories with different write to read port ratios: 1:1, 1:2, 1:3, 1:4. Pure multipumped memory blocks are used to construct non-multipumped LVT memories to save area. To handle access conflicts where two or more ports try to write to the same memory location in the same clock cycle, a conflict resolution circuit is added to the LVT memory. Even though the internal implementation of memory might differ, all of these memories remain black box equivalent from an outside point of view. We do not consider memories that stall, i.e, take multiple cycles to read or write because of unavailability of a resource. This is because if we consider stalling memories in case of a write/read miss, to hide the latency due to miss penalty we need to consider scheduling and re-ordering reads and writes (out-of-order execution). That is a separate problem in itself. The assumption has been made for simplicity. The original LVT based memory in [2] also does not consider stalling memories.

The memory is not initialized at the beginning.
3.2 Experimental setup

We use Altera’s Quartus II version 12.1 (Service Pack 1) to target a Cyclone IV GX EP4CGX110CF23C7 device. This is a low power, high functionality, low cost device. It has 6.5Mbits of M9K blocks. For simulation purposes we use Verilog 2001 to implement the BRAMs. None of the Verilog modules are Altera specific, hence the design can be ported onto FPGAs manufactured by other vendors. The operation of the memory is verified by simulating it in Modelsim. The synthesis is balanced and not biased towards optimizing speed or area.

The memory is wrapped in a test harness so that the I/O pin count is reduced and synthesis of memories with a higher number of ports is possible. The design otherwise would not have fit on the FPGA. The test harness is used for synthesis purposes only and not during simulation. The test harness consists of two modules: single input pins are connected to serial-in parallel-out shift registers which are in turn connected to the word wide inputs of the memory. The word wide outputs of the memory are again registered and ANDed so that they are reduced to a single set of input output pins.

Such a test harness can be partitioned from the RAM anytime. Analysing the RAM without the test harness will give a pessimistic analysis. But including it will give us realistic values of timing which we would get if the RAM were to be in an actual pipeline.

The inputs and outputs of the memory are registered for correct timing analysis.
For proper placement and routing the following measures have been taken:

i) Clock constraints - The maximum frequency of operation of the M9K BRAM in Cyclone IV is 315 MHz in the highest speed grade. We constrain the memory design to have clock frequency less than this value. Setting a higher target frequency will not improve the speed of the design but instead may worsen it [8]. This is because over-constraining the device will negatively impact the performance because of competition for device resources.

For each memory design, we vary the memory depth from 32 to 512 bits. For each memory depth, the clock constraints are changed iteratively until the timing constraints for the design are satisfied. TimeQuest Timing Analyser in Quartus II is used to report the unrestricted and the restricted\(^1\) maximum frequencies (Fmax) of all the designs. Fmax is averaged over five different place and route runs with random initial seeds.

We assume that all clocks are externally generated. Multiplied clocks which are used in multipumped designs are generated externally too and we assume that the base clock does not generate the multiplied clock. PLLs can be used to generate such multiplied clocks which will be synchronous with the system clock.

\(^1\) The restriction is due to the minimum pulse width of the CycloneIV device which restricts the frequency of M9K to 250 MHz.
ii) Placement of I/O pin registers into the device pin registers is avoided. This is because it would otherwise create artificially long combinational paths which would wreck the timing analysis of the RAM.

Figure 3.2.1a shows an improper placement of a 4W/8R, 256-bits deep memory, where all the elements except for BRAMs are placed near I/O pins. Figure 3.2.1b shows the same memory which is properly placed.

The FPGA is operated under its normal temperature range of 0 - 85°C and a voltage of 1.2V.

Figure 3.2.1: (a) shows improper placement of 4W/8R, 256-bits deep memory on a Cyclone IV device. (b) A 4W/8R, 256-bits deep memory which is properly placed on a Cyclone IV device.
3.3 Area measurement

The area occupied by the multi-ported memory is measured in two ways:

i) number of BRAMs consumed – Quartus II infers BRAMs from the Verilog code and maps these onto the M9K blocks of the FPGA;

ii) number of LEs consumed - the logic other than the BRAMs is implemented in LEs. Hence the area consumed will be equal to total number of BRAMs and total number of LEs used.

Note: Some of the experiments done in [2] on Stratix III were repeated on the Cyclone IV device for comparison purposes with the other experiments in this thesis that target Cyclone IV.
Chapter 4

Write-to-read port ratio

In a multi-ported memory, it is necessary to know the effect of the number of read and write ports and different ratios of read and write ports. In this thesis, since we concentrate on LVT-based multi-ported memory, we study different ratios of read and write ports.

From [2],[3],[4] it is evident that the as number of ports increases there is an increase in area and a decrease in frequency. All of the memories discussed in these studies have twice as many read ports as write ports. Although this is the common case in multi-ported memories like register files, it is interesting to know what the effect of other write-to-read port ratios is. In this chapter we study LVT-based multi-ported memories with write-to-read port ratios other than most general case of 1:2.

4.1 Implementation

Although the LVT-based memory proposed in [2] supports an arbitrary number of ports with \(m\)-write and \(n\)-read ports, studying the impact of different ratios of write-to-read ports helps us choose the right ratio according to the parameter we are trying to optimize, area occupied or frequency. We study the effect for LVT-based memory with write-to-read port ratios of 1:1, 1:3, 1:4 apart from the common case of 1:2. For this we build 2W/2R, 2W/6R and 2W/8R memories using the LVT technique.
4.2 Results

Figures 4.2a and 4.2b compare memory designs with different write-to-read port ratios. The memory depths ranging from 32-512 are shown as data labels in the graph.

**Performance in terms of frequency:** It is clearly evident from Figure 4.2a that a ratio of 1:2 has the highest frequency measure for all memory depths. It is then followed by 1:1, 1:3 and then 1:4. At a depth of 256 a 2W/4R memory has Fmax of 226 MHz, a 2W/2R memory has Fmax of 191 MHz, 2W/6R operates at 168 MHz and 2W/8R at 161 MHz.

![Graph showing Fmax vs. No. of LE s used for different memory depths and port ratios](image)

**Figure 4.2a:** Plot of Fmax versus the area utilized in terms of logic elements for memories with write-to-read port ratios of 1:1, 1:2, 1:3, 1:4.
**Performance in terms of area:** It is intuitive that the number of LEs used increases as the number of ports increases. This can be seen in Figure 4.2a. The number of BRAMs also increases proportionally. For an m-write, n-read memory the number of BRAMs consumed is equal to m x n. This is shown in Figure 4.2b.

![BRAM utilisation chart](chart.png)

**Figure 4.2b:** BRAM utilization for memories with write-to-read port ratios of 1:1, 1:2, 1:3, 1:4.
Chapter 5

Multipumping

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.
5.1 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 5.1.1.

![Read operation](read_operation.png) ![Write operation](write_operation.png)

Figure 5.1.1 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 $\lceil m/2 + n/2x \rceil$, where x is the total number of BRAMs used and where $\lceil p \rceil$ 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 StratixIII 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. This can be seen in Figure 5.1.2.

![Graph](image)

(a) Fmax vs Area

Figure 5.1.2: Speed and area for M9K-based 2W/4R multipumped memory implemented on StratixIII FPGA. The circled part is the pure-multipumped memory [2].
We implement 2W/4R pure multipumped memory (PMP_2W4R) on our target Cyclone IV FPGA. As seen from Figures 5.1.3 and 5.1.4 we get similar results on the Cyclone IV. We plot both the restricted and unrestricted frequencies but consider only the unrestricted frequency to quote the results*. Area in terms of BRAMs is shown in Figure 5.4.3.

* The restriction is specific to this device of Cyclone IV FPGA. Reporting the unrestricted frequency will give more generalized values.
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 result, only a quarter of the BRAMs are used to construct the new LVT based memory. This 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 $b = (\log_2 m) - 1$, 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.

5.2 Implementation

The LVT based design would have a significant area improvement if the FPGAs were provided with even moderately multi-ported BRAMs. Since most FPGAs have dual ported BRAMs, we can use these small pure-multipumped memories to build bigger memories.

In contrast to the old LVT-based memory, our LVT-based memory with pure-multipumped banks has banks of 2W/nR memory and not 1W/nR memory. The 2W/nR 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 \((\log_2 m) - 1\). 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.

Here the port number doesn’t correspond to the bank number as in LVT-based memory. Otherwise the way the read and write operations happen remains the same as in the old LVT-based memory explained in section 3.

Figure 5.2 shows a generalized design for LVT-based multi-ported memory with mW/nR ports, built from pure-multipumped banks.
5.3 Operation

We illustrate the operation of a 4W/8R memory in Figure 5.3.1.

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 5.3.1: Shows write and read operation of a 4W/8R LVT based memory whose banks are built using 2W/4R pure multi-pumped memory.](image)

Figure 5.3.1: Shows write and read operation of a 4W/8R LVT based memory whose banks are built using 2W/4R pure multi-pumped memory.
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 5.3.2.

Figure 5.3.2: Waveforms showing external clock and internal clock (internal clock frequency = 2 x external clock frequency) 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 5.3.1 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 5.3.1 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.

5.4 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 (LVT_PMP). Figure 5.4.1 shows the area comparison in terms of number of LEs utilized by LVT_4W8R and LVT_PMP_4W8R. There is an average 43% decrease in the number of LEs used and 75% decrease in BRAM consumption. For example, for a 256 deep memory LVT_4W8R uses 4300 LEs and 32 BRAMs whereas LVT_PMP_4W8R uses 2400 LEs and only 8 BRAMs.
Figure 5.4.1: Plot of speed and area of M9K based 4W/8R multi-ported memory constructed using 2W/4R pure multipumped memory.
Similar results can be seen for an 8W/16R memory in Figure 5.4.2. There is a significant decrease in the number of LEs used. But there is a drop in frequency as well. For a memory 256 bits deep, the number of LEs used by the old LVT method, LVT_8W16R, is 11873 and for our method, LVT_PMP_8W16R, the number is 7812. The BRAM utilization drops from 128 to 32 which is a reduction of 75%. This is seen in Figure 5.4.3.

![Comparison of LVT_8W16R and LVT_PMP_8W16R](image)

Figure 5.4.2: Plot of speed and area of M9K based 8W/16R multi-ported memory constructed using 2W/4R pure multipumped memory.
Figure 5.4.3: Comparison of BRAM utilization between the old LVT method and the new LVT method (LVT_PMP). We see that in all cases LVT_PMP consumes 75% fewer BRAMS.

5.5 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 memory results in drastic reduction in the area (logic elements and BRAM consumption) as seen from the results in the previous section, it has a harsh impact on the frequency of operation of memory. Since the LVT_PMP is inherently made up of PMP_2W4R memory which has a maximum frequency of operation of 151.8 MHz, the operating frequency of LVT_PMP memories is even lower. For a LVT_PMP_4W8R memory, as we can see from Figure 5.4.1, the frequency of operation ranges
from 125 MHz – 75 MHz for memory depth varying from 32 to 512 bits. Hence this type of memory is best used for more shallow memories because of the inverse relationship between depth and frequency.
Chapter 6

Write conflict resolution

In a multi-ported memory there is every possibility that a race condition, where two or more ports attempt to write to the same address in the same clock cycle, might occur. Write conflict is said to occur in such cases. Unknown data will be written into an address when a write conflict occurs. To avoid data ambiguity, such conflicts need to be detected and resolved. This chapter deals with designing write conflict detection and resolution circuitry for a LVT-based multi-ported memory.

Although the system enclosing the multi-ported memory, for example, a superscalar processor, VLIW processor, etc., can use software to handle conflicting writes, handling them in hardware makes the software more efficient. Also, if the memory is made as a separate chip, it gives a sense of completeness to the memory because it comes along with a conflict handling circuit. We can consider two general cases of handling write conflicts: let any one of the conflicting ports win or let none of them win. For a LVT-based multi-ported memory, we implement the first case. We do not choose to implement the last case of letting none of the ports win. This is because LVT-based multi-ported memory is a banked design. So the conflicting ports would have written their respective write values into the same write address but in their respective banks. Discarding these values will result in additional clock cycles to re-write the correct data into the conflicting address. For the former method, the conflict resolution happens in one clock cycle and the data from that address can be read in the next clock cycle.
6.1 CMOS v/s FPGA

In this section we see what happens during a write conflict situation in a CMOS circuit and in an FPGA.

In a CMOS RAM, the value to be written is applied to the bit lines. In the case where two or more bit lines want to write to the same memory location, the data can be written if the bit lines are writing the same data value. But if the data values conflict, then the data value stored depends on the relative driver strengths and metastability. Hence the data stored in the conflicted memory location will be truly undefined.

In an FPGA, using an LVT based multi-ported memory, since each bank has its own write port, the data value is correctly written into each of the banks in the case of a conflicting write. But the live value table, being a true multi-ported memory built using logic elements, will store an undefined bank number. So the conflicting writes in an LVT on an FPGA won’t interfere directly as in CMOS memory, since no such connections can exist; the conflicting address and the data will go through LUTs before being stored in a register. The logic function implemented by the LUTs is unknown to us ahead of time since it depends on the implementation of the LUT and the CAD tools that program it. Hence output of these LUTs is not known to us, thus the data stored in the registers will be termed undefined as in CMOS memories.

Understanding the exact nature of these conflicts helps us to handle the conflicts better. The next section explains the nature of conflict in an LVT-based multi-ported memory.
6.2 Nature of conflict

During a write operation, under normal conditions, each write port writes into a *unique* address location in its corresponding bank. The live value table simultaneously stores these bank numbers in its address locations. During a read operation, the live value table directs the read to the bank number that was stored in that address location. Thus there is no ambiguity in the LVT while storing a bank number during a write operation and in addition we are sure which bank the data will be read from.

But in the case of a write-write conflict, when two or more write ports try to write to the same memory location, the data value does get written into the individual banks. But the LVT will not know which bank number to store and an undefined bank number gets stored in that conflicted address location. Hence when that particular location is read in subsequent cycles, the LVT may steer the read to any of those banks into which the data were written or to a completely different bank. Hence we are unsure where the data value is being read from.

Experimental results show that the LVT stores one of the bank numbers into which the conflicted values were written but we are unsure which one it would be. For example, in a 4W/8R memory, let’s say port 1, port 2 and port 3 write the values 10, 20 and 30 to the same address location 45 in their respective banks, i.e., bank 1, bank 2, and bank 3, respectively, in the same clock cycle. The data will be written correctly into the individual banks. The bank number that the LVT stores in its 45th location will be one among these banks, but it is unsure of what exactly it will be, 1 or 2 or 3.
To resolve this ambiguity, we modify the LVT-based multi-ported design by adding extra hardware circuitry to detect and resolve write conflicts. The sections below explain the address comparator, the two methods implemented to avoid conflicts and the performance impact on the memory after adding the extra hardware.

### 6.3 Address comparator

Since the LVT is where the problem occurs during a conflicting write situation, we add an address comparator circuit inside the LVT which checks all the write addresses at the write ports of LVT and outputs values according to whether the addresses match or not. The circuit is implemented entirely using hardware. We implement address comparators for a 2-write, 3-write and 4-write memory.

A single XOR gate forms a comparator for a 2-write memory. This is shown in Figure 6.3.1. Accordingly, if the output of the XOR gate is equal to 1, it means that both the write addresses are unique and if it is a 0, it means that the write addresses are the same.

![Figure 6.3.1: XOR gate used as an address comparator in a memory with 2 write ports, to compare if write addresses are equal.](attachment:image.png)
For a 3-write memory, Figure 6.3.2 shows the address comparator. The outputs of each of the XNOR gates, if equal to 1, indicate that two corresponding two addresses are the same else, if equal to 0, indicate that both the addresses are different. The digits on the wires at the output of each gate in Figure 6.3.2 indicate which addresses are being XNORed or ANDed. The comparison of addresses is done in two stages. In the first stage, all the input combinations are checked two at a time. This is done by using 2-input XNOR gates. Now, in the second stage, we check if all the address are equal. This is done simply by ANDing the outputs of the XNOR gates. If the output is equal to 1, it means that all the input addresses are equal. This covers all possible combinations of input addresses to check if any of them are the same.

As an example, let’s say the write addresses at port 0 and port 1 are the same, 21, and port 2 writes to address 10. The address comparison happens as follows. First all the address combinations taken 2 addresses at a time are checked. Address_0 xnor address_1, Address_0 xnor address_2, Address_1 xnor address_2. Then the output of XNOR gates, wire_01 and wire_02 (or wire_12) are ANDed to check if all the three addresses are the same. Wire_01 will be equal to 1 since address 0 and address 1 are same. Output of AND gate will be 0 which
indicates the three addresses are not the same.

Figure 6.3.3 shows the comparator for a 4-write memory. As before, the numbers on the wires indicate that the corresponding addresses are equal if the output of the gates is equal to 1.

![Address comparator circuit for a memory with 4 write ports.](image)

In general, for an m-write memory, we have \( \sum_{i=2}^{m} C_i \) comparisons to cover all the combinations of addresses. The term \( mC_m \) checks whether all the addresses are equal, terms \( mC_{m-1}, mC_{m-2}, \ldots \) check if the other combinations are equal. \( mC_2 \) signifies the minimum number of addresses that are compared, i.e., 2.

It is evident from this equation that, as the number of write ports increases, the number of combinations of comparisons increases exponentially. Hence this kind of a comparator though
very easy to implement is suitable for only a small number of write ports.

In the next section, we will introduce changes made to the basic LVT-based memory structure proposed in [2] to handle write port conflicts.

### 6.4 Priority method to handle write conflicts

In this method the write conflict problem is solved by giving priority to the ports, either by giving a high priority or a low priority. The only structural change in the memory design is in the LVT. The LVT now has an address comparator which selects a write port according to priority in case of a write conflict. Otherwise the memory functions like the old LVT based multi-ported memory.

Figure 6.4 shows a mW/nR LVT-based memory with a conflict resolution circuit included.

#### 6.4.1 Operation

Every time a write operation is performed, all the write addresses are compared with each other in the LVT by the address comparator. If there is no conflict, the operation is the same as in the LVT memory described above. But if the address comparator detects a write conflict, then the conflicting address at one of the ports is selected according to the priority of the ports and that port number gets stored in the corresponding LVT location. Writes to individual banks happen as before. When the conflicted address is read in subsequent cycles, the LVT steers the read to a known bank. Thus there is no data ambiguity because we know which bank the conflicted address is being read from.
Figure 6.4: An mW/nR LVT based memory with an additional feature of write conflict resolution added to it. Port priority is used to solve write conflicts.

Consider an example of a 4W/8R memory with a conflict resolution circuit which gives ports with lower number a higher priority. During a write operation, let’s say write port 0 and write port 2 attempt to write data values 80 and 88 respectively to the same memory location 25 in the
same clock cycle. The data will be written in the individual banks, i.e., bank 0 stores 80 in its address location 25 and bank 2 stores 88 in its address location 25. This conflict is detected by the address comparator in the LVT and it stores the bank number 0 in its memory location 25 since the lower numbered port gets higher priority. During a read cycle, when address 25 is read, the LVT directs the read to bank 0 and the value 80 is read. Thus there is no uncertainty about which bank the data is read from.

6.4.2. Performance

Solving write conflicts in LVT-based multi-ported memory using priority method performs very well both in terms of frequency and area consumption. Since the address comparator is a simple circuit consisting of XNOR and AND gates, it does not contribute much to the area consumption in terms of logic elements. Figures 6.4.2.1, 6.4.2.3, and 6.4.2.5 show the plot of frequency vs area consumed in terms of LEs by 2W/4R, 3W/6R and 4W/8R memories respectively. Figure 6.4.2.2, 6.4.2.4, and 6.4.2.6 show their BRAM consumption.
Figure 6.4.2.1: Graph comparing frequency of operation versus number of logic elements used for a 2W/4R memory with and without conflict resolution circuitry. Conflict handling methods include priority method and phantom bank method.

Figure 6.4.2.2: Plot of number of M9K’s used for a 2W/4R memory with and without conflict resolution circuitry.
As we can see from Figures 6.4.2.1 and 6.4.2.2, a 2W/4R memory with write conflict resolution using the priority technique occupies almost the same amount of area as 2W/4R memory without write conflict resolution circuitry. The frequency drops by an average of 50 MHz but the memory is still in the operating range of frequency. A 256 bit deep 2W/4R memory without conflict resolution operates at 225 MHz and occupies 1315 LE’s and 8 BRAMs, whereas the same memory with conflict resolution operates at 182 MHz and occupies an area of 1385 LE’s and 8 BRAMs.

Similar results can be seen for a 3W/6R memory in Figures 6.4.2.3 and 6.4.2.4. Area consumption remains almost the same with about a 2% increase in LE consumption. Operating frequency drops by an average of just 20 MHz.

Thus the results show that addition of this extra feature to the memory does not significantly impact its performance.
Figure 6.4.2.3: Graph comparing frequency of operation versus number of logic elements used for a 3W/6R memory with and without conflict resolution circuitry. Conflict handling methods include priority method and phantom bank method.

Figure 6.4.2.4: Plot of number of BRAMs used for a 3W/6R memory with and without conflict resolution circuitry.
For our 4W/8R memory, the performance and area consumption is similar to the original LVT memory for memory depths below 256 bits. Beyond that the number of logic elements consumed increases drastically. BRAM consumption however, remains the same. Results are shown in Figures 6.4.2.5 and 6.4.2.6.

Figure 6.4.2.5: Graph comparing frequency of operation versus number of logic elements used for a 4W/8R memory with and without conflict resolution circuitry. Conflict handling methods include priority method and phantom bank method.
Figure 6.4.2.6: Plot of number of BRAM’s used for a 4W/8R memory with and without conflict resolution circuitry.

6.5 Write conflict resolution by adding phantom bank

The ambiguity of which bank number the LVT would store in cases where two or more write ports attempt to write to the same address in the same clock cycle can also be solved by allocating a separate bank which all such conflicting writes are redirected to. We call this bank a ‘phantom bank’. All the conflicting writes are written into phantom bank and read from this bank in subsequent cycles until that address gets written into uniquely in a later clock cycle. Introduction of this bank brings about changes in the old LVT-based memory design. Since the phantom bank needs to be allocated a bank number which will be stored in the LVT, width of the LVT has to be increased by one bit. The width of LVT now becomes \( \log_2 (m + 1) \), where \( m \) is the number of write ports. LVT also has an address comparator, explained in section 6.3, which
compares all the write addresses to check if any of them are conflicting. If there is no conflict, the design operates like the original LVT based multi-ported memory. However, if a conflict is detected between ports, LVT stores the phantom bank number in that address location. The value of data that has the greatest value among data values at the conflicted address location is written into the phantom bank. Thus we know for sure which data value is written into memory. To facilitate read from the phantom bank, the read mux in the original LVT design has to be given one more input. The rest of the memory design remains the same as the original LVT based multi-ported memory.

Since LVT based memory is a banked design, the advantage of introducing the phantom bank is that we can be sure where data is being read from in case a write conflict arises, since the LVT knows which bank number to store in such a case. This surety wouldn’t be there if memory did not have a write conflict resolution circuitry, in which case the LVT would randomly store any bank number out of all banks involved in conflicting writes. Also, one among the original data values is retained. Choice to retain data value at the conflicted location: largest, smallest, etc., can be made depending on the application. If write data values are the same, then that single data value is stored. Figure 6.5 shows a generalised mW/nR LVT based memory with conflict detection and resolution circuitry added which includes a phantom bank and a comparator.
Figure 6.5: An mW/nR LVT based memory with write conflict resolution circuitry included. An additional bank called ‘Phantom bank’ is used to store conflicted write values.
6.5.1 Operation

During every write cycle, all the write addresses are compared by the comparator in LVT. If all addresses are unique, the memory operates like the original LVT based memory. All write ports write into their individual banks and LVT stores the bank number those addresses were written into. Reads to those addresses are directed to appropriate banks by LVT.

In case of conflict, apart from the conflicting write ports writing data into their individual banks, the conflicted address also gets written into the phantom bank. It is the phantom bank number that LVT stores at this location so that there is no ambiguity as to which bank that conflicted address will be read from, in subsequent read cycles. Data at conflicting write ports are compared and the chosen data value gets stored at conflicted address location in phantom bank.

Consider a 4W/8R memory. Assume that we choose to write the largest value at conflicting ports. In some cycle assume that port 0 and port 2 attempt to write data values 80 and 88 respectively to the same memory location 25 in the same clock cycle. Port 1 writes data 5 to address 10 and port 3 writes data 6 to address 15. So in the write cycle data value 5 is stored in address 10 of bank 1 and value 6 is stored in address 15 in bank 3. LVT stores 1 and 3 in its address locations 10 and 15 respectively. This is the same as the original LVT method. But the conflict between port 0 and port 2 is resolved as follows: though writes to bank 0 and bank 2 happen as before, data 88 (because the greater of 80 and 88 is chosen) is written into address 25 of the phantom bank. LVT stores the phantom bank number in its 25th location. Subsequent reads to address 25 happen from the phantom bank until another write port writes to this address uniquely.
6.5.2 Performance

Figures 6.4.2.1 through 6.4.2.6 show operating frequency and area consumption (in terms of LEs and BRAMs) of different multi-ported memories with write conflict resolution circuit (phantom bank). We can see that operating frequency is not affected when compared with memory without conflict detection. For all memories, 2W/4R, 3W/6R and 4W/8R, we can see that operating frequency remains approximately in the same range as that of an old LVT based memory.

However, due to the presence of comparators (to detect conflict of write address and to choose write data that has a greater value, in case of write conflicts) and the phantom bank, there is some increase in area consumption. From Figures 6.4.2.4 and 6.4.2.6 we see that for larger memories, 3W/6R and 4W/8R, for memory depths greater than 128, 6 more BRAMs are consumed. LE consumption also doubles. Hence this method is suited for shallow memories. For example, a 128-bit deep 3W/6R memory without conflict resolution circuitry operates at 187.3 MHz, consumes 1670 LEs and 18 BRAMs, whereas the same memory with conflict resolution circuitry operates at 181 MHz and consumes 1774 LEs and 8 BRAMs. Thus we see that our design is very efficient for memories that have small depth.
Chapter 7

Conclusions and future work

7.1 Conclusions

Multi-ported memories are needed to avoid structural hazards and to increase processing speed in a processor. FPGAs provide dual ported block RAMs on their fabric. One of the main reasons multi-ported memory is avoided is because of its large area requirement. Previous techniques to build area efficient multi-ported memories have not been very successful. LVT based multi-ported memory proposed in [2], though fast, occupies reasonable amount of area. We have modified the old LVT based memory to optimize area occupied by the memory. We use 2W/4R pure multi-pumped memory that has the same operating frequency and area across different memory depths, to build banks of LVT-based memory. Using our approach instead of just replicated dual ported BRAMs to build banks of LVT-based memory results in a 75% decrease in the number of BRAM’s consumed and an approximately 43% decrease in the number of logic elements used. For example, on Altera’s Cyclone IV FPGA, a 256-bit deep, 4W/8R LVT memory consumed 32 BRAMs and 4300 logic elements whereas our 4W/8R LVT_PMP memory consumes only 8 BRAMs and 2874 logic elements. That is a significant decrease in area consumption.

We further explore LVT based multi-ported memory to study the effects of different write to read port ratios. The common case is write to read port ratio of 1:2. We studied ratios of 1:1, 1:3 and 1:4. We find that for this design 1:2 ratio performs best in terms of speed, followed by
memories with a ratio of 1:1 > 1:3 > 1:4. A ratio of 1:2 between write and read ports gives us a good trade-off between speed and area consumed.

We also handle write conflicts in LVT based multi-ported memory. Two methods of resolving write conflicts are implemented in hardware. Both methods have address comparators inside LVT. The first method is the priority method, where ports are given a high / low priority. So, in case of write conflict, the port having the higher (lower) priority wins. When compared to memory without conflict resolution circuitry, performance of our memory with this added feature shows that operating frequency remains approximately in the same range, with very less area penalty. It is very simple to implement.

In our second method, we add another bank called ‘phantom bank’ which the conflicting address is redirected to. We choose to write the greatest / least data (modified in a way to suit the application the memory is used for) compared to other data at the conflicting ports. If ports are trying to write the same data, it is written without issue. In subsequent read cycles, all reads to conflicted addresses are redirected to the phantom bank by the LVT. Frequency of operation for this method stays in the range of that of a memory without conflict resolution. But in terms of area consumption, it is suited for shallow memories. For deeper memories, area overhead due to introduction of another bank, address and data comparators becomes too much.
7.2 Future work

1) We modified the structure of banks old LVT based multi-ported memory. Instead of the banks being made up of replicated dual ported memory, we used 2W/4R pure-multipumped memory to construct the banks. Though this reduced the memory area greatly, it had a negative impact on frequency of operation. So instead of building all banks of LVT based memory using only replicated dual ported BRAM’s or using only 2W/4R pure-multipumped memory, we could use a combination of the two to achieve a good trade-off between area and operating frequency. Frequency advantage from traditional LVT technique and area advantage from our LVT_PMP technique. For example, a 4W/8R memory could have two 1W/8R banks and one bank built from pure-multipumped 2W/4R memory. This would consume 20 BRAM’s as against 32 BRAM’s needed for old LVT based memory and 8 BRAM’s needed for our LVT_PMP memory. LE consumption too would be in between that of these two methods. Similarly a 8W/4R memory could be made up of alternating banks of just replicated BRAM’s and 2W/4R pure-multipumped memory. This would use 56 BRAM’s as against 128 BRAM’s needed for old LVT based memory and 32 BRAM’s needed for our LVT_PMP memory.

A generalized diagram for this kind of a memory is shown in Figure 7.2.1.

This method might have a frequency better than that of LVT_PMP memories since a mix of banks that are not pure-multipumped and banks that are pure-multipumped would not degrade frequency of operation of memory as much as when only pure-multipumped banks are used. Thus we could achieve both area reduction and a good frequency of operation using this method.
2) To handle conflicting writes at write ports, one might also choose not to write any of the conflicting data into memory. In this case, an acknowledgement signal can be sent to writers that conflicted, to perform write operation once more. LVT could retain whatever bank number was stored in the conflicted address until another port writes into this address uniquely. Once an acknowledgement signal is sent to writers, those ports could be freed to carry out writes at other
addresses. However, instructions that needed to perform write would have to wait till that write could be performed without any conflicts.
References


7. Nick Sawyer and Marc Defossez, "Quad-port memories in Virtex devices". Xilinx application note, XAPP228 (v1. 0), September 24, 2002.


18. Spartan-6 FPGA Block RAM Resources User Guide, Synchronous Dual-Port and Simple Dual-Port RAMs, Conflict avoidance, UG383 (v1.5) July 8, 2011.
