2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines

A Multi-Ported Memory Compiler Utilizing True Dual-port BRAMs

Ameer M.S. Abdelhadi and Guy G.F. Lemieux
Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
{ameer,lemieux}@ece.ubc.ca

***Abstract*—Recent work has shown how multi-ported RAMs can be built out of dual-ported RAMs. Such techniques combine two structures: a set of "data banks" to hold the data, and a method for selecting the bank containing the last-written data, often called a live-value table (LVT). Most previous work has focused on the design of the LVT to reduce area and improve performance. In this paper, we instead reduce area by optimizing the design of the "data banks" portion. The optimization is embedded into a memory compiler that solves a set cover problem. When the set cover problem is solved optimally, the data banks use minimum area. Our technique applies to multi-ported RAMs that have a structural pattern we describe as "switched ports". Switched ports are a generalization of true ports, where a certain number of write ports can be dynamically switched into a possibly different number of read ports using one common read/write control signal. Furthermore, a given application may have multiple sets, each set with a different read/write control. While previous work generates multi-port RAM solutions that contain only true ports, or only simple ports, we contend that using only these two models is too limiting and prevents optimizations from being applied. Experimental results on 10 random instances of multi-port RAMs show 17% BRAM reduction on average compared to the best of other approaches. The compiler and a fully parameterized Verilog implementation is released as an open source library. The library has been extensively tested using Altera's EDA tools.**

Keywords—embedded memory; block RAM; multi-ported memory; shared memory; register-file; parallel memory access

I. INTRODUCTION

Multi-ported memories are the cornerstone of all high-performance CPU designs. They are often used in register files, but also in other shared-memory structures such as caches and coherence tags. Hence, high-bandwidth memories with multiple parallel reading and writing ports are required. In particular, multi-ported RAMs are used by wide superscalar processors [1], VLIW processors [1][2], multi-core processors [3][4], vector processors, and coarse-grain reconfigurable arrays (CGRAs). For example, the second generation of the Itanium processor architecture employs a 20-port register file constructed from SRAM bit cells with 12 read ports and 8 write ports [3]. The key requirement for all of these designs is fast, single-cycle, concurrent access from multiple requesters.

In FPGAs, one way of synthesizing a multi-ported RAM is to build it from registers and logic. However, this is only feasible for very small memories. Another way is to alter the basic SRAM bit cell to provide extra access ports, but this requires a custom design for each unique set of parameters (*e.g.* number of ports). Since FPGAs must fix their RAM block design for the most common usage case, it is too costly to provide highly specialized RAMs with a large number of ports in a device.

Recently, a few different multi-ported RAM designs for FPGAs have been proposed. All of these techniques use two structures: a set of "data banks" to hold the data itself, and a method for selecting the bank containing the last-written data, often called a live-value table (LVT). Implementation of the LVT varies, from a register-based LVT [5], to an invalidation-based LVT [6] which uses block RAMs only. All of these techniques optimize the LVT itself, but leave the data banks portion untouched.

It is worth noting that, since these approaches focus on the LVT, they all use block RAMs only in simple dual-port mode for the data banks, where one port is dedicated to reads, and the other is dedicated to writes. However, most FPGA block RAMs can operate as true dual ports, where each port can dynamically switch (independently) between read and write. This paper exploits this true dual-port capability to save area in the data banks.

Two recent works take advantage of the true dual-port capability to reduce the number of block RAMs used for the data banks (as opposed to the LVT). The first work designs a multi-port RAM where every port is a true dual-port [7]; it requires half as many block RAMs for data banks as when the original LVT method is used with fixed ports [5]. However, this approach makes every port a true dual port; we show that even fewer block RAMs can be used if multiple ports can be grouped into switched ports. The second work allows simple read and simple write ports as before, but adds the notion of a single "switched port" [8]. While a true port is based on the pairing of a single read and single write port, switched ports are best described as a set of read ports and set of write ports switched under a common read/write control signal. This saves block RAMs by placing some ports into true dual-port mode. However, this second work is limited to implementing just a single switched port.

In this paper, we demonstrate how to build multi-port RAMs that contain any number of swiched ports. As well, the multi-port RAM may contain any number of simple read

978-1-5090-2356-1/16 $31.00 © 2016 IEEE
DOI 10.1109/FCCM.2016.45

ports, simple write ports, and any number of true dual ports (each of the latter will be treated as a switched port). Since a true port is a special case of a switched port, we are effectively generalizing the techniques used in [7] and [8]. Under this new model, we show an optimization method that can minimize the use of block RAMs in the data banks. We also note that for any given problem, there may be multiple solutions. Our solution uses a minimum set cover algorithm to map required read/write dependences into true dual-port block RAMs. In all of our test cases, the set cover problem was solved optimally, leading to the use of minimal block RAMs in the data banks.

This paper constructs novel, generic, modular and parametric switched BRAM-based multi-switched-ports RAMs. Compared to previous simple-port and true-port methods, the proposed technique significantly reduces BRAM consumption. The data banks are optimized to support mixed-port configuration by utilizing true-dual-ported BRAMs. The data bank connectivity is converted into a Data-Flow-Graph (DFG), where vertices represent ports and edges represent data banks, respectively. Each block RAM deployed in the solution will cover one or more DFG edges. An optimal set covering all of the DFG edges results in deploying a minimal number of block RAMs.

Our architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II. A large variety of different architectures and parameters, bypassing, depth and width are simulated in batch, each with over a million random memory cycles. Stratix V, Altera's high-end performance-oriented FPGA, is used to implement and compare the proposed approach with previous techniques. A run-in-batch simulation and synthesis flow manager is also provided. The Verilog modules and the flow manager are available online [9][10].

The rest of this paper is organized as follows. In Section II, RAM port classification is provided. BRAM-based RAM multi-porting techniques are reviewed in Section III. The proposed multi-switched-ports synthesis method is described in detail in Section IV. The experimental framework and results, are discussed in Section V. Conclusions are drawn in Section VI.

II. RAM Port Classification

As described in Fig. 1 (top), RAM ports can be classified into two categories: fixed ports and switched ports.

A **fixed port** is either a **simple read port** or a **simple write port**; the activity of any write is not switched with any other read. Hence, for a set of fixed ports, all writes and reads in the set can be concurrently active.

A **true dual-port**, or simply a **true port**, is a single port that can perform either read or write under the control of a single read/write line. A true port is often drawn as two distinct data ports (one for read, one for write), but they share address lines in common. It is possible (but uncommon) to do a read at the same time as a write to the same address, but the resulting data are implementation-specific.

A **switched port** is a collection or set of read ports and write ports. The number of read and write ports may be different. The

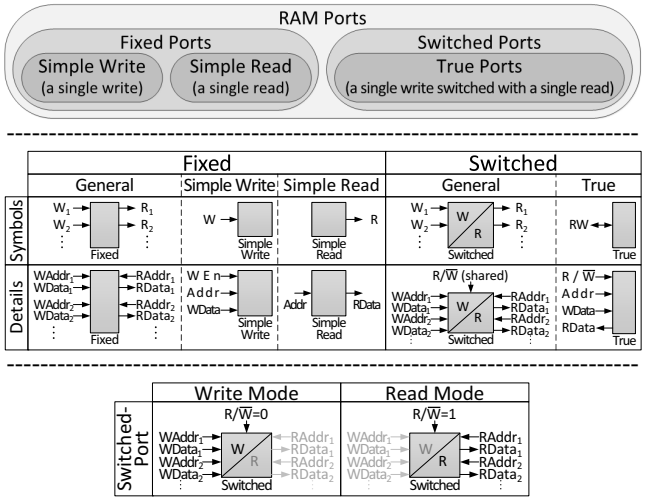


Fig. 1. RAM port classification: (top) Venn diagram. (middle) Symbols. (bottom) Switched-port modes; faint ports are inactive.

read-address lines are usually distinct from the write-address lines. However, the entire set of ports share a single read/write line that controls whether the write ports are active, or the read ports are active. Reads and writes cannot be simultaneously active. Note that a true port is a special case where a switched port consisting of a single read and a single write, and the address lines are shared. A given application may have multiple switched ports, each with an independent read/write line.

Fig. 1 (middle) shows symbols and black box connectivity for these different types of RAM ports. The switched port read/write activity is controlled by a shared $R/\overline{W}$ control signal. Fig. 1 (bottom) shows the two modes of a switched port. The first is the write mode where $R/\overline{W} = 0$, writes are active and reads are inactive. The second is the read mode, where $R/\overline{W} = 1$, reads are active and writes are inactive.

III. Previous Work

In this section, we review two approaches to reduce the size of the data banks. The first turns all ports into true dual-ports, while the second achieves savings with a single switched port.

A. Multi-Ported RAM with True Dual-ports

Choi *et al.* [7] introduced a modification to the data banks to build a multi-ported RAM where all ports are true dual ports. A reduction in area is achieved by utilizing the bidirectional functionality of true dual-port BRAMs.

Fig. 2 (left) illustrates a generalization of this method. Each port is a true dual port that either writes to or reads from a set of data banks. Each pair of these ports has one data bank in common. Hence, when a given port is reading, it can access data written by any other bank.

A register-based LVT determines which bank was last written at each address. An example of a 3-port RAM is shown in Fig. 2 (right). Data written from one port must be accessible to all other ports via a shared true dual-port RAM. Hence, a total of $\frac{1}{2} \cdot n_t \cdot (n_t - 1)$ data copies are required, where n_t is the number of ports. In contrast, the original LVT approach [5] requires ${n_t}^2$ data copies.

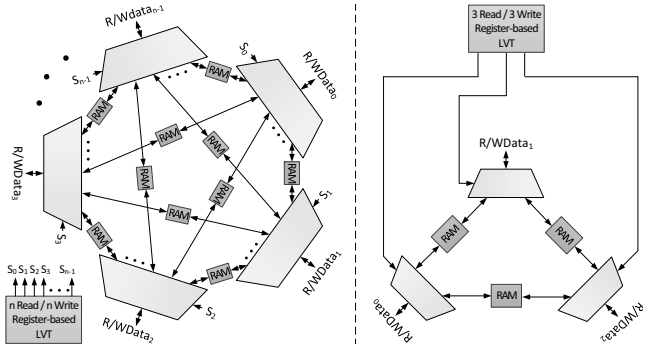


Fig. 2. True-port banks. Each port share a single BRAM with any other port. (left) Generalized approach. (right) A 3 ports example.

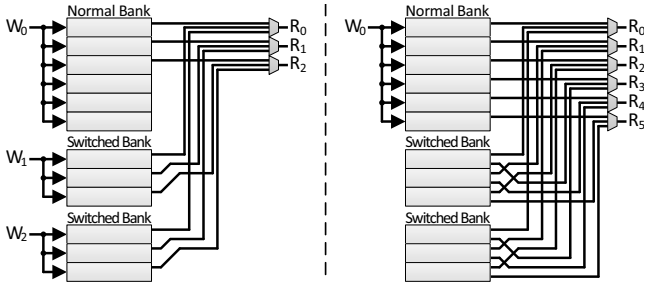


Fig. 3. A switched-port example with $\boldsymbol{n_{W,f} = 1; n_{R,f} = 3; n_{W,s} = 2; n_{R,s} = 3}$. (left) write configuration (right) read configuration.

B. Multi-Ported RAM with a Single Switched Port

The notion of a switched port was introduced in [8], where true dual-ported BRAMs are utilized to construct switched ports. This architecture contains two sets of ports: (1) a set of fixed ports supporting $n_{R,f}$ simultaneous reads and $n_{W,f}$ simultaneous writes, and (2) a single switched port with $n_{R,s}$ reads alternating dynamically in time with $n_{W,s}$ writes. Thus, the ports on a given multi-ported RAM design instance can be characterized by a set $\{ (n_{W,f}, n_{R,f}), (n_{W,s}, n_{R,s}) \}$, where the first pair of values are the fixed port quantities and the second pair are the switched port quantities.

The key idea behind the SRAM savings is reconfiguring unused writing ports into reading ports. Fig. 3 describes an example of a switched multi-ported RAM with a fixed port of $n_{W,f} = 1$ writes and $n_{R,f} = 3$ reads and a switched port of $n_{W,s} = 2$ writes and $n_{R,s} = 3$ reads. Fig. 3 (left) shows the write mode configuration, while Fig. 3 (right) shows the read mode configuration. In this example, the upper multi-read bank keeps a single write operation, while other banks sacrifice write ports to provide additional read ports. Only data banks whose writing ports are unused in read mode are altered, namely $n_{W,s}$ banks. The writing ports of each of these banks are redirected to serve as reading ports in read mode. The other $n_{W,f}$ banks that keeps writing ports in read mode must increase the number of reading ports to $n_{R,f} + n_{R,s}$ to match read port requirements in read mode. Hence, a total of $n_{W,f} \cdot (n_{R,f}+n_{R,s}) + n_{W,s} \cdot n_{R,f}$ data copies are required.

IV. Multi-ported RAM with Multiple Switched Ports

In this section, we describe how multiple switched ports may arise, and how to design multi-ported RAMs with them.

A. Motivation and Key Idea

Consider the design of a processing element (PE) that has three distinct (non-overlapping) states of operation:

1. Write to shared RAM using 4 write ports
2. Compute value in shared RAM with ALU using 2 read ports, 1 write port
3. Read from shared RAM using 4 read ports

In this example, it is necessary to build a shared RAM structure that has multiple read and write ports. There are several approaches to building such a multi-port RAM, including:

A. 4 fixed read ports and 4 fixed write ports

B. 4 true dual-ports

C. two switched ports, with the first having 1 read or 1 write port (ie, a true port), and the second having 3 read or 3 write ports

D. 2 fixed read ports, 1 fixed write port, and one switched port containing 2 read or 3 write ports

In addition, there are many other possible "port assignment" solutions, where each port assignment may yield a solution requiring a different number of block RAMs. In this paper, we do not consider the problem of computing an optimal port assignment; that is left for future work. Instead, we must first be able to compute the block RAM requirements for a given port assignment; that is the purpose of this paper.

The example above requires port assignment because there are multiple states. Consider simpler problems, such as those in Fig. 4, where each "user" of the shared memory is under the control of a single read/write line. In such cases, there exists only one possible port assignment with switched ports.

The one valid port assignment is illustrated on the right-hand side of Fig. 4 and can broken down as follows. The ALU consists of two mutually-exclusive functional blocks, f and g. First, the control signal $f/\bar{g}$ enables either f or g functional blocks; in either case the R_0 operand must always be read and can be assigned to a fixed port $R_{0,0}$. When f is selected, the R_1 operand can be read from switched read port $R_{2,0}$, but when g is selected we do not need $R_{2,0}$ but instead need switched write ports $W_{2,0}$ and $W_{2,1}$. Thus, $f/\bar{g}$ is also the read/write control signal for the P_2 group of switched ports. Similarly, the bus has a read/write control signal that directly controls the P_1 group of switched ports with $R_{1,0}$ and $W_{1,0}$.

Thus, Fig. 4 shows two possible implementations of the shared memory: one with all fixed ports on the left, and one with switched ports on the right. This leads to two possible

ways to build the shared memory; we will show that the latter way is more efficient in terms of block RAM usage.

The key idea of our methodology is based on constructing a Data-Flow-Graph (DFG) to describe RAM port dependencies, where vertices represent ports and edges represent data banks. Two types of edges exist: regular (solid) edges for fixed ports, and dashed edges for switchable ports.

The goal is to "cover" all of these edges, and this cover directly describes an implementation using BRAMs. For example, a single regular edge can be covered by a simple dual-port BRAM with 1 fixed write port and 1 fixed read port (1W/1R). However, true dual-port BRAMs have two terminals on each end (2W/2R), and up to 4 edges between them. Thus, up to 4 edges in the graph can be covered by a true dual-port BRAM. The objective is to cover all edges with minimal BRAMs. This forms a set cover problem (SCP) which can be solved using special subgraph patterns to indicate possible covers.

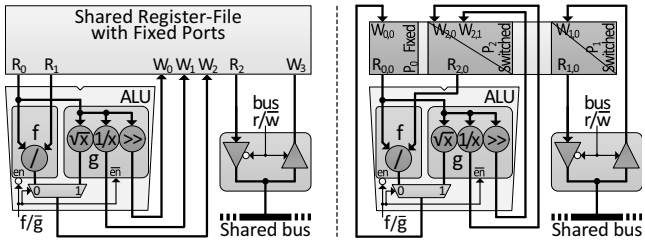


Fig. 4. Simplified parallel system with shared memory. (left) Multi-ported RAM with fixed ports connection. (right) Multi-switched-ports connection.

B. Port Assignment and Problem Definition

The problem input is a list of port requirements. Unlike switched ports, fixed ports are unrelated and operate individually; hence, they can be aggregated into a single port group (as in Fig. 4) named P_0. The superset of all port groups, P, includes n_P port groups, were the first port P_0 is a fixed-port and each of the remaining $P_{1\dots n_P-1}$ are switched port groups. Each P_i represents an ordered pair with the number of writes $n_{W,i}$ and the number of reads $n_{R,i}$ for this specific port, namely

$$P = \{P_0, P_1, \dots, P_{n_P-1} | P_i = (n_{W,i}, n_{R,i})\}. \quad (1)$$

For instance, port requirements for the example in Fig. 4 is

$$P = \{P_0 = (1,1), P_1 = (1,1), P_2 = (2,1)\}. \quad (2)$$

Writes and reads in this RAM are indexed by two indices, the first index is the port group index ranging $0 \dots n_P - 1$, while the second index is the write index within a specific port i ranging $0 \dots n_{W,i} - 1$, or the read index within a specific port i ranging $0 \dots n_{R,i} - 1$. The writes group W and the reads group R are defined as

$$\begin{aligned} W &= \{W_{i,j} | 0 \le i < n_P, 0 \le j < n_{W,i}\} \\ R &= \{R_{i,j} | 0 \le i < n_P, 0 \le j < n_{R,i}\} \end{aligned}. \quad (3)$$

For instance, write and read sets for the example in Fig. 4 are $W = \{W_{0,0}, W_{1,0}, W_{2,0}, W_{2,1}\}$ and $R = \{R_{0,0}, R_{1,0}, R_{2,0}\}$, respectively.

The total number of writes and reads is denoted by n_W and n_R (without indices), respectively, namely,

$$n_W = |W| = \sum_{i=0}^{n_P-1} n_{W,i}; \; n_R = |R| = \sum_{i=0}^{n_P-1} n_{R,i}. \quad (4)$$

For instance, the example in Fig. 4 consists of 4 writing ports and 3 reading ports in total, hence $n_W = 4$ and $n_R = 3$.

Fig. 5 generalizes the port assignment for the multi-switched-ports RAM. Given these port requirements, and using true-dual-ported BRAMs, the objective of our work is to construct the data banks to satisfy the given fixed and switched ports with the fewest BRAMs.

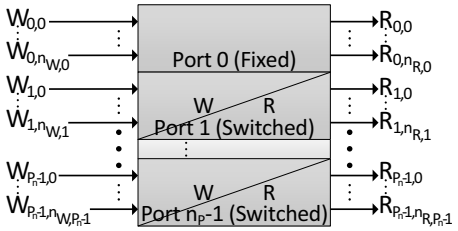


Fig. 5. Multi-switched-ports RAM port assignment.

C. Modeling Data Banks with Data-Flow-Graph (DFG)

An LVT-based multi-ported RAM built using only fixed ports requires every write port to write to a dedicated data bank, allowing concurrent writes. Furthermore, every write-specific bank should be accessible by every read port, allowing all read ports to read data written by any write port. This requirement can be modeled as a *complete bipartite graph* (*complete bigraph*).

A bigraph is a graph G consisting of two disjoint sets of vertices, say U and V. Each edge in a bigraph connects a vertex from U to another vertex in V. A complete bigraph is a special case where every vertex in U is connected to every vertex in V, in other words

$$G = (U, V, E | U \cap V = \emptyset, E = \{\{u, v\} | u \in U, v \in V\}). \quad (5)$$

To model data bank connectivity, a bigraph DFG is constructed where source vertices are writing ports $U = W$ and sink vertices are reading ports $V = R$. Graph edges connect writes to reads, hence they represent 1W/1R simple-dual-port BRAMs. Fig. 6 (left) shows the bigraph of the fixed-port system in Fig. 4 (left).

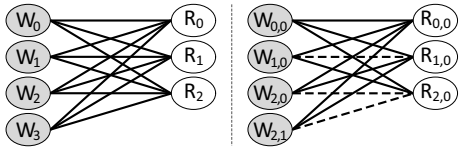


Fig. 6. Bigraph DFG representing data banks connectivity of the shared memory in Fig. 4. (left) fixed-ports data banks (right) multi-switched-ports.

Similarly, Fig. 6 (right) shows a bigraph of the switched-port system in Fig. 4 (right). However, the bigraph is slightly different from before; some edges $E_s \subseteq E$ are labeled as switched edges using dashed lines in the figure. Except for

port group P_0, which has only fixed ports, the other port groups give rise to switched edges that connect the write vertices to read vertices within the same port group. Formally, the switched edge set is described as follows

$$E_s = \left\{\{W_{p,i}, R_{p,j}\} \middle| 1 \le p < n_P, 0 \le i < n_{W,p}, 0 \le j < n_{R,p}\right\}. \quad (6)$$

For instance, in Fig. 6 (right), the switched edges are $E_s = \left\{\{W_{1,0}, R_{1,0}\}, \{W_{2,0}, R_{2,0}\}, \{W_{2,0}, R_{2,1}\}\right\}$.

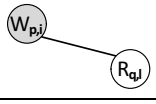
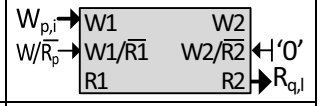
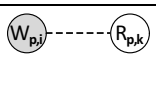
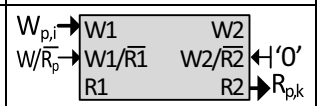
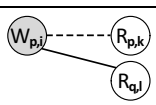
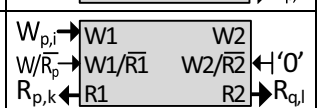
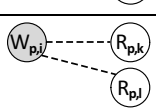
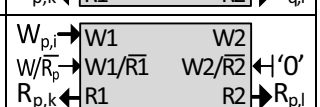
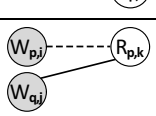
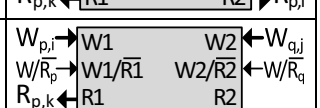
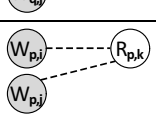
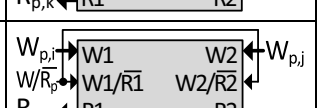
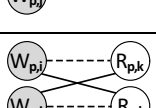
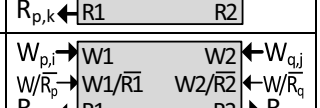
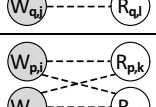
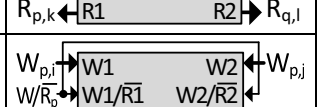
D. Multi-switched-ports DFG Optimization

Using a true dual-ported BRAM gives us the ability to cover several possible subgraphs that appear in a bigraph DFG. Since these subgraphs are also complete bigraphs, we call them *biclique patterns*, or BPs. All different biclique patterns are described in Table I.

For each BP in Table I, we can identify several specific instances that it can cover within the biclique DFG from Fig. 6 (right). Fig. 7 enumerates all possible BP instances that occur within the original biclique DFG.

Once this full enumeration has occurred, all that is necessary is to select a subset of these BP instances such that all edges in the original biclique DFG are covered. Each BP instance requires a BRAM, so minimizing the number of BP instances in the cover will minimize BRAM usage.

TABLE I. BICLIQUE PATTERNS AND THEIR ATTRIBUTES

Pattern		Name	Biclique	BRAM Connectivity
1W/1R	Fixed	F1W1R		
	Switched	S1W1R		
1W/2R	Fixed	F1W2R		
	Switched	S1W2R		
2W/1R	Fixed	F2W1R		
	Switched	S2W1R		
2W/2R	Fixed	F2W2R		
	Switched	S2W2R		

Biclique patterns must cover all the edges in the switched bigraph DFG. However, different BPs may have shared edges. *Efficiently* covering the bigraph DFG edges is not trivial; simply choosing all largest 2W/2R-BPs may result in inefficient results. Covering the edges of the bigraph DFG is actually equivalent to the set cover problem. The set cover problem is an NP-complete problem [12].

E. Solving the Cover Problem

The set cover problem takes a universe set U and another set S of subsets of U whose union covers U ($\bigcup_{s\in S} s = U$), namely,

$$SCP = (U, S | \forall s \in S: s \subseteq U, \bigcup_{s\in S} s = U). \quad (7)$$

The objective of the set cover problem is to find a cover of U with the fewest sets from S. Let $T \subseteq S$ be such a subset of S, therefore the set cover problem objective is,

$$\min_{T\subseteq S} \left(|T| \middle| \bigcup_{t\in T} t = U\right) \quad (8)$$

The bigraph DFG optimization solves a set cover problem where all DFG edges are the universe and all biclique patterns are the covering subsets, namely,

$$SCP = \left(U = E, S = \begin{matrix} F1W1R \cup S1W1R \cup F1W2R \cup S1W2R \cup \\ F2W1R \cup S2W1R \cup F2W2R \cup S2W2R \end{matrix}\right). \quad (9)$$

The set cover problem can be formulated and solved as the following binary linear programming (BLP) problem

$$\begin{aligned} &\text{minimize } \textstyle\sum_{s\in S} x_s \\ &\text{subject to } \textstyle\sum_{s:e\in s} x_s \ge 1 \ \ \forall e \in U, \\ &\qquad x_s \in \{0,1\} \ \ \forall s \in S \end{aligned} \quad (10)$$

Where x_s is a binary decision variable, indicating whether s is part of the solution or not.

Fig. 7 provides a synthesis example for the bigraph DFG from Fig. 6 (right). A set cover solution uses the highlighted biclique patterns in Fig. 7 (left), producing the final synthesized data bank shown in Fig. 7 (right).

As a comparison, Table II compares solutions for the purely fixed-ports and multi-switched-ports implementations of Fig. 4. For this specific example, the fixed-ports method consumes 12 BRAMs to construct the data banks while the multi-switched-ports method consumes only 8 BRAMs, yielding a 25% BRAM reduction.

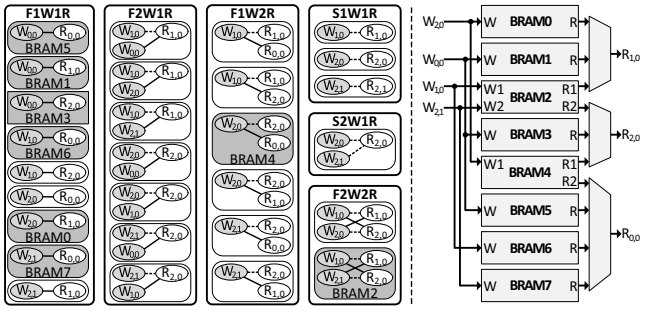
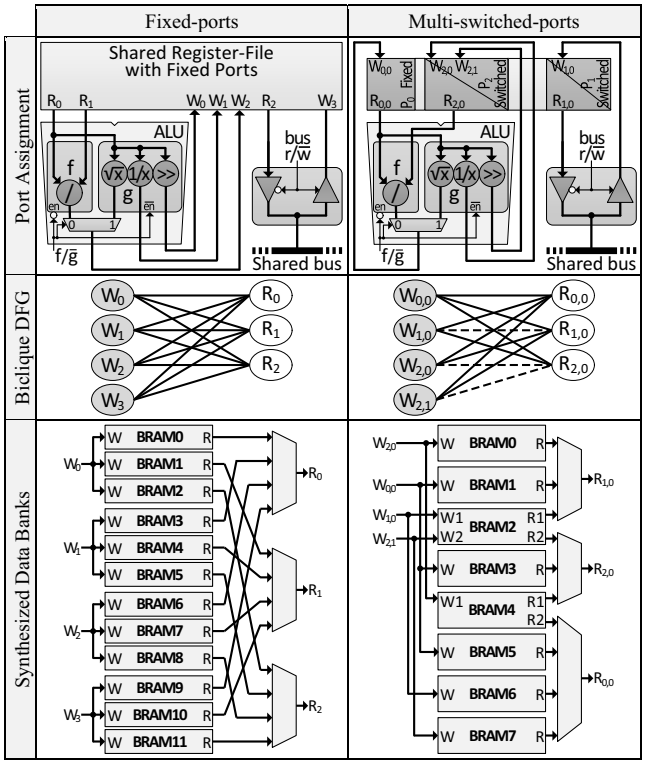


Fig. 7. Synthesis example of the DFG from Fig. 4. (left) All possible biclique patterns; optimal BP's are highlighted. (right) Synthesized data banks.

TABLE II. BICLIQUE PATTERNS AND THEIR ATTRIBUTES



F. Data Dependencies and Bypassing

Due to the pipelined nature of building a full multi-switched-port RAM, data dependencies due to internal latencies arise naturally. This requires internal forwarding and bypassing to solve these hazards. The full multi-switched-port RAM design consists of the data banks and the I-LVT. The I-LVT itself consists of feedback banks and output extraction banks. For each of these three structures, Table III summarizes the type of bypassing required to produce a correct design that can tolerate certain hazards. Further detail is provided below.

The I-LVT-based structure [6] is used to steer the read data out of the multi-switched-ports data banks, however the I-LVT incurs data dependencies due to the feedback functions and the latency of reading the I-LVT to decide about the last written bank [6]. Data dependencies can be handled by employing bypassing, also known as forwarding. Bypassing is necessary since dual-port BRAMs cannot internally forward new data when one port reads and the other port writes the same address on the same clock edge, constituting a read-during-write (RDW) hazard.

Table IV shows two types of bypassing based on write data and address pipelining. Both bypassing techniques are functionally equivalent, allowing reading of the data that is being written on the same clock edge, similar to single register functionality. However, the fully-pipelined two-stage bypassing shown in Table IV (bottom) can overcome an additional cycle latency, namely an additional pipe stage on writing data and address (not shown in the figures). This capability is required if a BRAM has pipelined inputs (*e.g.*, cascaded from another BRAM) that need to be bypassed.

The proposed multi-switched-ports RAM utilizes true-dual-port BRAMs to provide switched port functionality. However, since writing and reading operations in true-dual-ported RAMs are exchangeable, the bypassing circuitry requires special handling. As described in Table IV (right), the bypass circuit of the true/true RAM configuration is mirrored compared to the true/simple RAM configuration. Thus, it can bypass written data from any direction. However, the control logic that drives the bypassing mux selectors need to be altered to detect the direction of writing.

The most severe data dependency that I-LVT design [6] suffers from is write-after-write (WAW), namely, writing to the same address that has been written in the previous cycle. This dependency occurs because of the feedback reading and writing latency. A single-stage bypassing for the feedback banks solves this dependency.

Two types of reading hazards are also introduced by the I-LVT design, read-after-write (RAW) and read-during-write (RDW). RAW occurs when the same data that was written in the previous clock edge are read in the current clock edge. RDW occurs when the same data are written and read on the same clock edge.

Due to the latency of the I-LVT, reading from the same address on the next clock edge after writing (RAW) will provide the old data. To read the new data instead, the output extraction banks of the I-LVT should be bypassed by a single-stage bypass to overcome the I-LVT latency.

The deepest bypassing stage is reading new data on the same writing clock edge (RDW), which is similar to a single register stage latency. This can be achieved by 2-stage bypass on the output extraction banks of the I-LVT to allow reading on the same clock edge. The data banks, which are working in parallel with the I-LVT should be bypassed by a single-stage to provide new data.

V. EXPERIMENTAL RESULTS

A. Experimental Framework

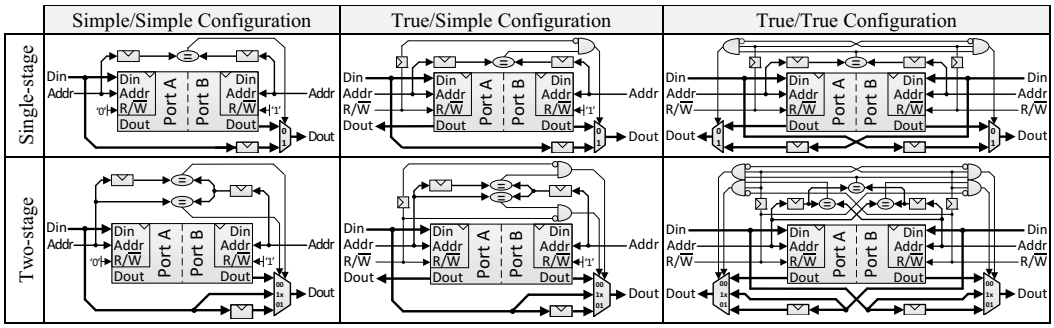
The proposed multi-switched-port RAM approach, complete with bypassing ,has been fully implemented in parameterized Verilog. For a given design instance, we developed a memory compiler to convert the RAM port assignment into a biclique DFG, enumerate all of the biclique pattern instances in the DFG, and use these to describe a set cover problem instance. The set cover problem is formulated as a Binary Linear Programming (BLP) problem using AMPL (A Mathematical Programming Language) [13], which is an algebraic modeling language used to describe large-scale mathematical optimization problems. The BLP optimization problem is solved using GLPK (GNU Linear Programming Kit) [14], an open source large-scale linear programming solver. Finally, the selected biclique patterns (covers) are used to automatically construct the data banks as described in Table I and shown for example in Fig. 7.

A run-in-batch flow manager has also been developed to simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II. The Verilog modules, the algorithmic scripts and the flow manager are available online as an open source contribution [9][10].

TABLE III. BYPASSING OF MULTI-SWITCHED-PORTS

	Data Banks	I-LVT Banks	
		Feedback	Output Extract
Allow WAW	None	1-stage	None
New Data RAW	None	1-stage	1-stage
New Data RDW	1-stage	1-stage	2-stage

TABLE IV. SINGLE-STAGE AND TWO-STAGE BRAM BYPASSING



To verify correctness, each design instance is simulated using Altera's ModelSim. A large variety instances with of different RAM port assignments and design parameters, e.g. bypassing, RAM depth and data width, are swept and simulated in batch, each with over a million random cycles. These multi-switched-port RAM design modules were then compiled with Altera's Quartus II into Altera's Stratix V 5SGXEA7N1F45C1 device [11]. This is a speed grade 1 device with 234k ALMs and 2560 M20K blocks.

B. Methodology

The proposed multi-switched-port RAM design process is generic and can support any number of fixed and heterogeneous switched ports. This means all previous multiport-RAM methods are actually special cases of this new proposed method. For benchmarking purposes, we take a number of design instances, and for each one we find a multi-switched port RAM using our new method. We then need to generate comparative results using multiport RAM designed with fixed ports [6], with true ports [7], and with a single switched port [8].

Then, we can compare results against the older fixed-port method [6] using the same tooling. By treating switched ports as fixed ports, we thereby place all read and write ports into the first port group P_0 (the fixed port). This generates a fixed-port solution that satisfies the same design instance requirements.

To compare against using the true-ports method [7], we must avoid using the fixed port group P_0. Instead, as described in Table V, each fixed port must be mapped into a true port (with a fixed $R/\overline{W}$ control), hence $n_{W,0}+n_{R,0}$ true ports are required to implement the fixed ports. In addition, every read and write pair in a switched port can be mapped into a single true port, hence, $\max(n_{W,i}, n_{R,i})$ true ports are required to implement a switched port group P_i. In total, the number of the required true ports is

$$n_t = n_{W,0}+n_{R,0} + \sum_{i=1}^{n_P-1} \max(n_{W,i}, n_{R,i}). \quad (11)$$

For example, the system in Fig. 4 can be implemented using $n_t = 5$ true ports.

To compare against the single-switched-port method [8], the largest switched port (say P_m) is chosen to be implemented as the single-switched port, and all the other ports are implemented using fixed ports. This becomes

$$P = \{P_0 = (n_W - n_{W,m}, n_R - n_{R,m}), P_1 = (n_{W,m}, n_{R,m})\}, \quad (12)$$

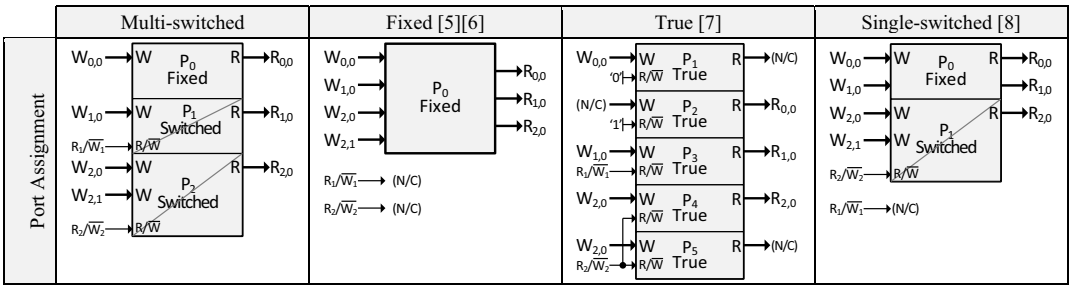
where m is the index of the switched-port with the maximum writes and reads,

$$1 \le m < n_P \left| n_{W,m}+n_{R,m} = \max_{1\le i<n_P} (n_{W,i}+n_{R,i}). \right. \quad (13)$$

C. Test Cases

A number of random multi-switched-ports test-cases are listed in Table VI. Ten random test cases, TC1 to TC10, have been generated for illustrative purposes; real cases are difficult to extract from applications without a precise understanding of their use of multi-port RAMs and how their FSMs specify (possibly mutually exclusive cases of) read/write behaviour. While our method can synthesize any number ports, the test-cases are limited to 8 switched ports to avoid accidentally exceeding device resources. These random test-cases use 4 to 8 switched ports, where each switched port has 1 to 4 writes or reads. For each test case, we specify a multi-port RAM that the multi-switched ports approach proposed in this paper. In addition, we show other multi-port RAM designs with fixed ports [6], true ports [7], and a single switched-port [8] that address the same requirements.

TABLE V. MULTI-SWITCHED-PORTS CONVERSION (EXAMPLE FROM FIG. 4)



D. Results

Table VII lists the experimental results of the ten random test-cases defined in Table VI, implemented using the four design styles. All synthesized test-cases have one byte of data width and 8k-lines in depth, new-data-RAW bypassing and use a binary-coded I-LVT [6]. BRAM consumption, ALMs and Fmax are given directly in the table. Table VIII lists the change percentage in these parameters compared to our proposed method. Compared to the single switched-port [8] and the fixed-port [6] methods, ALM consumption and Fmax are similar while BRAM consumption reduced by 18% on average. On the other hand, comparing our proposed method to the true-ports method shows a 42% BRAM reduction, 53% fewer ALMs, and 15% higher Fmax.

TABLE VI. HETEROGENEOUS MULTI-PORTED RAM TESTCASES

Test-case #	Writes and reads for each port; $(n_{W,i}, n_{R,i})$ pairs from (1)													
	Multi-switched								Fixed[6]	True[7]			Single-switched[8]	
	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_0	P_0	$P_1 \dots P_{n_t}$	n_t	P_0	P_1
TC1	(1,1)	(1,1)	(2,1)	(2,1)	(1,2)	(1,2)	(2,3)	(2,3)	(11,13)	(0,0)	(1,1)	15	(9,10)	(2,3)
TC2	(1,1)	(1,1)	(2,1)	(1,2)	(2,2)	(2,3)	(3,2)	(3,3)	(13,14)	(0,0)	(1,1)	15	(10,11)	(3,3)
TC3	(1,2)	(1,1)	(1,2)	(2,1)	(2,2)	(2,3)	(3,2)	-	(12,12)	(0,0)	(1,1)	15	(9,10)	(3,2)
TC4	(2,1)	(1,1)	(1,1)	(1,2)	(1,3)	(2,3)	(3,3)	-	(9,15)	(0,0)	(1,1)	15	(6,12)	(3,3)
TC5	(1,1)	(1,3)	(2,1)	(2,1)	(2,2)	(2,3)	-	-	(10,10)	(0,0)	(1,1)	13	(8,7)	(2,3)
TC6	(1,3)	(1,1)	(1,1)	(1,3)	(2,2)	(3,3)	-	-	(8,13)	(0,0)	(1,1)	13	(5,10)	(3,3)
TC7	(2,2)	(1,1)	(2,4)	(2,1)	(1,3)	-	-	-	(8,11)	(0,0)	(1,1)	14	(6,7)	(2,4)
TC8	(2,1)	(1,1)	(1,4)	(2,1)	(3,1)	-	-	-	(8,10)	(0,0)	(1,1)	14	(7,6)	(1,4)
TC9	(2,3)	(2,1)	(1,4)	(2,4)	-	-	-	-	(7,12)	(0,0)	(1,1)	15	(5,8)	(2,4)
TC10	(3,1)	(1,2)	(2,4)	(3,4)	-	-	-	-	(9,11)	(0,0)	(1,1)	14	(6,7)	(3,4)

TABLE VII. EXPERIMENTAL RESULTS

Test-case#	Multi-switched			Single-switched[8]			True[7]			Fixed[6]		
	BRAMs	ALMs	f_{max}(MHz)	BRAMs	ALMs	f_{max}(MHz)	BRAMs	ALMs	f_{max}(MHz)	BRAMs	ALMs	f_{max}(MHz)
TC1	826	3417	247.4	1054	3349	242.25	1290	6222	215.84	1034	3271	235.29
TC2	1044	4781	224.16	1316	4828	228.1	1290	6222	215.84	1352	4840	225.33
TC3	904	3793	232.56	1104	3717	237.3	1290	6222	215.84	1080	3634	241.9
TC4	726	2732	250.75	882	2653	253.87	1290	6222	215.84	918	2617	245.04
TC5	628	2434	248.32	756	2441	244.38	962	4503	235.68	780	2414	253.94
TC6	568	2031	258	700	1948	260.42	962	4503	235.68	704	1916	246.37
TC7	540	1801	257.67	608	1746	265.32	1120	5483	214.13	608	1703	260.96
TC8	524	1675	265.82	576	1629	270.42	1120	5483	214.13	592	1614	266.03
TC9	504	1499	279.17	556	1502	270.27	1290	6222	215.84	560	1444	275.71
TC10	586	2321	252.33	690	2205	257.33	1120	5483	214.13	702	2154	247.46
Avg.	685	2648.4	251.62	824.2	2601.8	253	1173.4	5656.5	219.3	833	2560.7	249.8

TABLE VIII. RESULTS COMPARISON

Test-case#	BRAM Reduction compared to:			ALM Reduction Compared to:			f_{max} Increase Compared to:		
	Single-switched[8]	True[7]	Fixed[6]	Single-switched[8]	True[7]	Fixed[6]	Single-switched[8]	True[7]	Fixed[6]
TC1	22%	36%	20%	-2%	45%	-4%	2%	15%	5%
TC2	21%	19%	23%	1%	23%	1%	-2%	4%	-1%
TC3	18%	30%	16%	-2%	39%	-4%	-2%	8%	-4%
TC4	18%	44%	21%	-3%	56%	-4%	-1%	16%	2%
TC5	17%	35%	19%	0%	46%	-1%	2%	5%	-2%
TC6	19%	41%	19%	-4%	55%	-6%	-1%	9%	5%
TC7	11%	52%	11%	-3%	67%	-6%	-3%	20%	-1%
TC8	9%	53%	11%	-3%	69%	-4%	-2%	24%	0%
TC9	9%	61%	10%	0%	76%	-4%	3%	29%	1%
TC10	15%	48%	17%	-5%	58%	-8%	-2%	18%	2%
Avg.	17%	42%	18%	-2%	53%	-3%	-1%	15%	1%

VI. CONCLUSIONS

In this paper, we propose a new idea of having multiple switched ports in multi-ported RAM design. This method requires a memory compiler to create a specific design instance, and solving a set cover problem to optimize its implementation. Our CAD approach always finds a minimal implementation for all of our test cases, but there is opportunity for further CAD research to improve run-time while still being optimal. On average out of 10 random test-cases, the suggested multi-switched-ports method reduces BRAM use by 18% compared to the best of previous methods, while maintaining ALM count and Fmax. Future research may address the RAM port assignment problem to more complex cases where there are more than two states governing memory port usage.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions. Especially the suggestion to replace the linear-programming approach with an exact solution.

REFERENCES

[1] J.H. Tseng and K. Asanovic, "Banked multiported register files for high-frequency superscalar microprocessors," *Int'l Symp. on Computer Architecture (ISCA)*, May 2003, pp. 62–71.

[2] J.A. Fisher, "Very Long Instruction Word architectures and the ELI-512," *Int'l Sym. on Comp. Arch. (ISCA)*, 11(3), June 1983.

[3] E.S. Fetzer and J.T Orton, "A fully-bypassed 6-issue integer datapath and register file on an Itanium microprocessor," *IEEE Int'l Solid-State Circuits Conf.*, vol. 1, Feb. 2002, pp. 420–478.

[4] H. Bajwa and X. Chen, "Low-Power High-Performance and Dynamically Configured Multi-Port Cache Memory Architecture," *Int'l Conf. on Elec. Eng.*, Apr. 2007, pp. 1–6.

[5] C.E. LaForest and J.G. Steffan, "Efficient Multi-ported Memories for FPGAs," *ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA '10)*, Feb. 2010, pp. 41–50.

[6] A. M.S. Abdelhadi and G. G.F. Lemieux, "Modular multi-ported SRAM-based memories," *ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays (FPGA '14)*, Feb. 2014, pp. 35–44.

[7] J. Choi, K. Nam, A. Canis, J. Anderson, S. Brown, and T. Czajkowski, "Impact of Cache Architecture and Interface on Performance and Area of FPGA-Based Processor/Parallel-Accelerator Systems," *Int'l Symp. on Field-Programmable Custom Computing Machines (FCCM '12)*, Apr. 2012, pp. 17–24.

[8] A. M.S. Abdelhadi and G. G.F. Lemieux, "Modular Switched Multi-ported SRAM-based Memories," *ACM Transactions on Reconfigurable Technology and Systems (TRETS) Special Issue on Reconfigurable Components with Source Code*, in press, accepted in Jul. 2015. 27 pages.

[9] G. G.F. Lemieux. (2016). Software *Downloads page* [online]. Available: http://www.ece.ubc.ca/~lemieux/downloads/

[10] A. M.S. Abdelhadi. (2016). *GitHub repository* [online]. Available: https://github.com/AmeerAbdelhadi/

[11] Altera Corporation, Stratix V Device Handbook, June 2011.

[12] R.M. Karp, "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations, pp. 85–103, Plenum Press, NY, 1972.

[13] R. Fourer, D.M. Gay, B.W. Kernighan, AMPL: A Modeling Language for Mathematical Programming. Duxbury Press, 2002.

[14] https://www.gnu.org/software/glpk/