Switch Design for Soft Interconnection Networks

Giorgos Dimitrakopoulos, Christoforos Kachris, and Emmanouil Kalligeros

1 Introduction

Many-core chip multiprocessors integrate many processing cores that need a modular communication infrastructure in order to show their full potential. Scalable interconnection networks that use a network of switches connected with point-to-point links can parallelize the communication between these modules and improve performance significantly [7]. Such on-chip interconnection networks are already a mainstream technology for ASICs, while they gain significant importance in FPGA-based systems-on-chip (SOCs) [1]. The first on-chip interconnection networks mimicked the designs that were architected for large, high-performance multiprocessors. However, as interconnects migrate to the on-chip environment, constraints and trade-offs shift, and they should be appropriately adapted to the characteristics of the implementation fabric [2].

An FPGA can host two forms of interconnection networks: the soft (or overlay) interconnection networks that are statically mapped on the configurable logic of the FPGA using LUTs, registers, and RAMs, as any other ordinary circuit [5], and the dynamically reconfigurable interconnection networks that exploit the reconfigurable nature of the FPGA and allow the design of customized alternatives

G. Dimitrakopoulos (✉)
Electrical and Computer Engineering Department, Democritus University of Thrace (DUTH), Kimmeria Campus B(1.11), Xanthi, GR 67100, Greece
e-mail: dimitrak@ee.duth.gr

C. Kachris
Athens Information Technology (AIT), Athens, Greece
e-mail: kachris@ait.edu.gr

E. Kalligeros
Information and Communication Systems Engineering Department, University of the Aegean, Samos, Greece
e-mail: kalliger@aegean.gr

P. Athanas et al. (eds.), *Embedded Systems Design with FPGAs*,
DOI 10.1007/978-1-4614-1362-2_6, © Springer Science+Business Media, LLC 2013

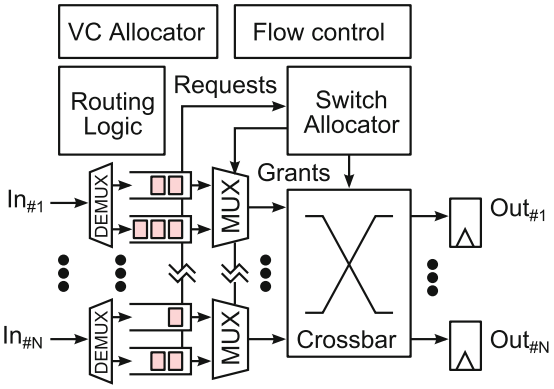


Fig. 1 Basic building blocks of a switch

that can be adapted at runtime using both the available logic blocks and even the reconfiguration network itself [25]. In the first case, the FPGA fabric acts as an ASIC replacement that hosts a complex multiprocessor system on chip. The system consists of general-purpose soft processors, application-specific accelerators, as well as memory and I/O controllers communicating via the soft interconnection network that transfers the software-generated messages. In the second case, the system is customized for a specific set of tasks, and any application change requires its dynamic reconfiguration at the logic level [27]. The performance of the system depends on how often a reconfiguration is required and how much gain can be earned by the customization of both the processing elements and the interconnection network.

Soft interconnection networks are more generic and can support various switching technologies ranging from statically scheduled circuit-switched operation, with predetermined and prescheduled routes that avoid contention, to fully dynamic packet switching that favors statistical multiplexing on the network's links [15]. In this chapter, we focus on the dynamic approach, assuming wormhole or virtual-cut-through networks with single- or multiple-word packets. When such networks are mapped on the FPGA, a critical factor to overall system's efficiency is (a) the selection of the appropriate network topology that would reduce the communication distance and utilize efficiently the on-chip wiring resources and (b) the selection of the appropriate switch architecture that fits better to the LUT-based nature of the FPGA and offers area-delay efficient designs [16, 24]. Both factors are closely interrelated, since the reduction of the communication distance between any two nodes increases the radix (number of input and output ports) of the switches and makes their design more difficult. Concentration or the addition of express channels further increases the radix of the corresponding switches [8].

The switches of the network follow roughly the architecture depicted in Fig. 1. Incoming packets are stored in input buffers and possibly in output buffers after crossing the crossbar. Routing logic unwraps incoming packets' header and determines their output destination. The inputs that are allowed to send their data over the crossbar are determined by the switch allocator. The switch allocator accepts

the requests from each input and decides which one to grant in order to produce a valid connection pattern for the crossbar. In cases that we want to differentiate between separate traffic classes, i.e., request/reply packets, and to offer deadlock-free adaptive routing, we can allow the sharing of network's channels by virtual channels (VCs) [6]. The assignment of a valid VC to a packet, before it leaves the switch, is a responsibility of the VC allocator.

While routing computation can be performed in parallel to the rest operations by employing lookahead routing [11], and VC allocation can operate in parallel to switch allocation using speculation [22], switch allocation and traversal remain closely interrelated, with switch allocation always preceding and guiding switch traversal. In fact, several designs already proved that switch allocation and traversal determine the critical path of the switch and limit any potential speed improvements [17]. So far, any innovation regarding the removal of this speed bottleneck relied mostly to architecture-level solutions that took for granted the characteristics of the allocators and the crossbar and, without any further modifications, tried to reorganize them in a more efficient way. Examples of this approach are the pipelined switch allocation and traversal units that increased the latency of the switches or prediction-based switches [18, 21]. Although such pure high-level design has produced highly efficient switches, the question on how better the switch would be if better building blocks were available remains to be investigated.

In this chapter, we try to answer this question for the case of the switch allocators and the crossbar that constitute a large part of the switch and determine the delay of the implementation. Our study will first present and compare the traditional implementations that are based on separate allocator and crossbar modules, and then will expand the design space by presenting new soft macros that can handle allocation and multiplexing concurrently. With the new macros, switch allocation and switch traversal can be performed simultaneously in the same cycle, while still offering energy-delay efficient implementations.

The rest of the chapter is organized as follows: Introductory material regarding the switch allocator alternatives at the architectural and the logic level is given in Sect. 2. Then, a review of the state-of-the-art separate arbiters and multiplexers that are used to build the switch allocator and the crossbar is presented in Sect. 3. Section 4 introduces the new merged arbiter and multiplexer module, while its efficiency relative to state-of-the-art is investigated experimentally in Sect. 5. Finally, conclusions are drawn in the last section.

2 Switch Allocation and Traversal

Each input of the switch can hold packets (or flits for wormhole switching) for multiple outputs. Therefore, it can request one or more outputs every cycle. For a VC-less switch that has a single FIFO queue per input, only one request per input is possible. In that case, as shown in Fig. 2, the switch allocator is constructed using a

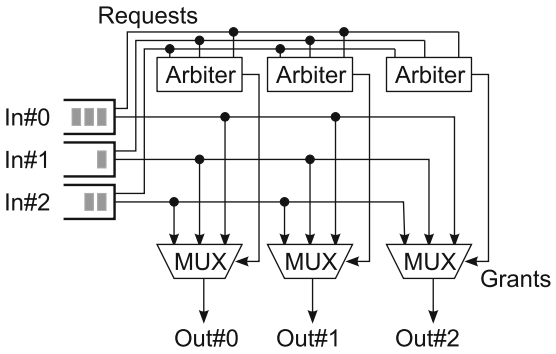


Fig. 2 Single arbiter per output for switches with one FIFO per input

single arbiter per output of the switch, which decides independently which input to serve. The grant signals of each arbiter drive the corresponding output multiplexer, and they are given back to the inputs to acknowledge the achieved connection.

In the case of switches with VCs, the input buffers are organized in multiple-independent queues, one for each VC. Each input can send multiple requests per clock cycle. This feature complicates significantly the design of the switch allocator relative to a VC-less switch. In that case, switch allocation is organized in two phases since both per-input and per-output arbitrations are needed.[1] Even though the per-input and per-output arbiters operate independently, their eventual outcomes in switch allocation are very much dependent, each one affecting the aggregate matching quality of the switch [3, 20].

The two possible switch allocators for an N-input switch with V virtual channels are shown in Fig. 3. In this figure, the output port requested by each VC is denoted by an N-bit wide one-hot coded bit vector. In the first case (Fig. 3a), each input is allowed to send to the outputs only one request. To decide which request to send, each input arbitrates locally among the requests of each VC. On the contrary, in the case of output-first allocation, all VCs are free to forward their requests to the output arbiters (Fig. 3b). In this way, it is possible that two or more VCs of the same input will receive a grant from different outputs. However, only one of them is allowed to pass its data to the crossbar. Therefore, a local arbitration needs to take place again that will resolve the conflict.

The grant signals produced by the input arbiters of an input-first switch allocator can drive the input local multiplexers in parallel to output arbitration. Therefore, when switch allocation and crossbar traversal are performed in the same cycle, this feature of input-first allocation allows some overlap in time between arbitration and

[1] An alternative to separable allocation is a centralized allocator like the wavefront allocator [26]. The main drawback of this design is the delay that grows linearly with the number of requests, while the cyclic combinational paths that are inherent to its structure cannot be handled by static timing analysis tools. The latter constraint can be removed by doubling the already aggravated delay [13].

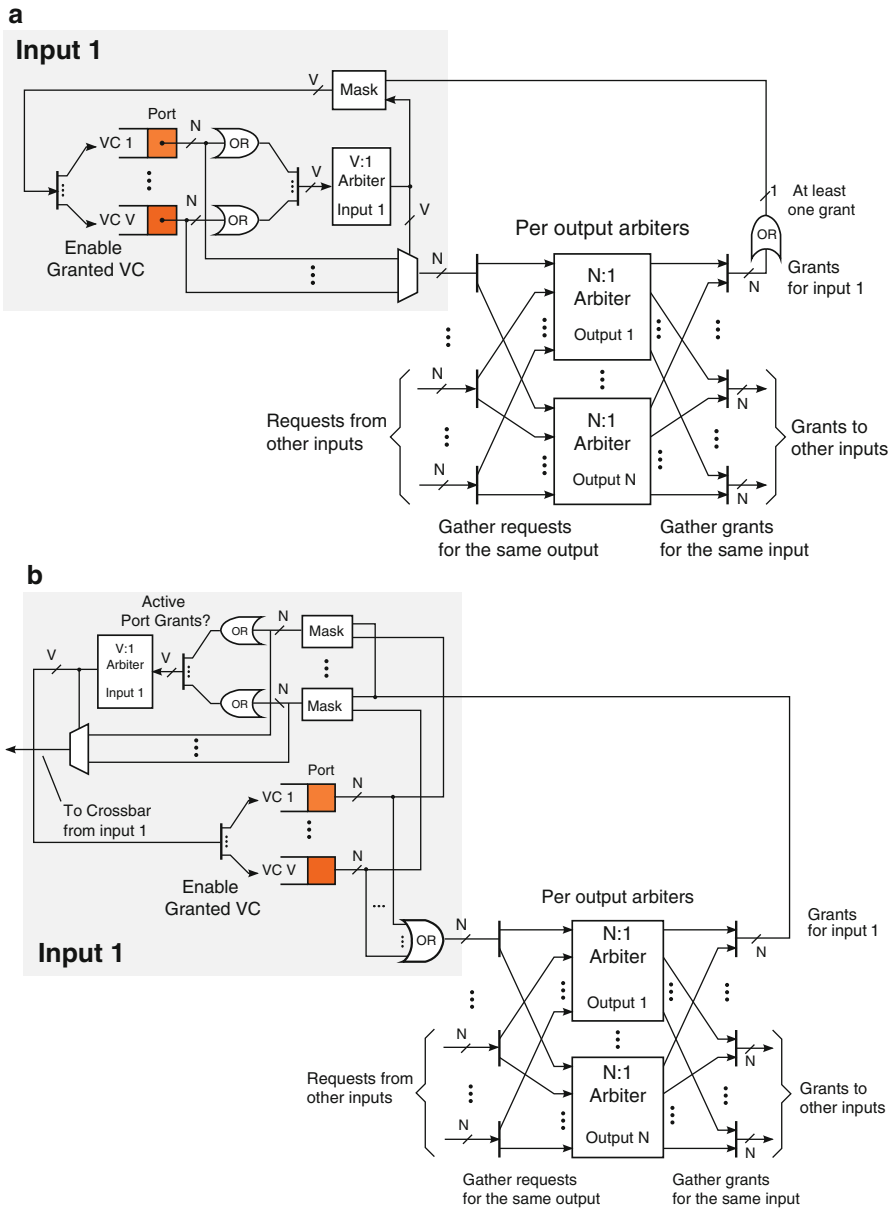


Fig. 3 Separable switch allocation for VC-based switch: (**a**) input-first allocation, (**b**) output-first allocation

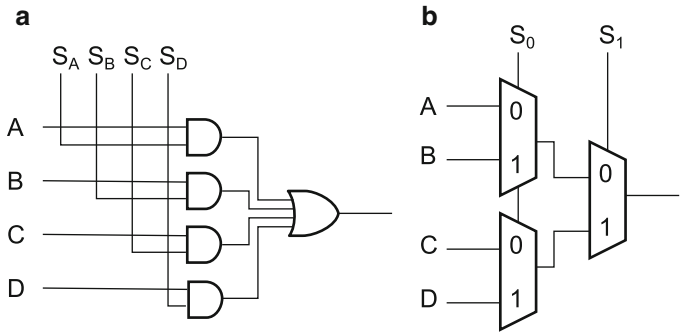


Fig. 4 Multiplexer implementations: **(a)** AND-OR structure and **(b)** tree of smaller multiplexers

multiplexing that reduces the delay of the combined operation. Such an overlap is not possible to output-first allocation, where both stages of arbitration should be first completed before driving the multiplexers.

In every case, the kernel of switch allocation and traversal involves arbiter and multiplexer pairs that need to be carefully co-optimized in order to achieve an overall efficient implementation. For example, the encoding selected for the grant signals directly affects the design of the multiplexers. In the first case, shown in Fig. 4a, the grant decision is encoded in one-hot form, where only a single bit is set, and the multiplexer is implemented using an AND-OR structure. On the contrary, in Fig. 4b, the multiplexer is implemented as a tree of smaller multiplexers. In this case, the select lines that configure the paths of each level of the tree are encoded using weighted binary representation.

The LUT mapping of either form of multiplexers is well explored, and several optimizations have been presented in open literature. Briefly, the optimizations presented so far for the implementation of wide multiplexers on an FPGA fabric involve either the best possible packing of the multiplexer inputs and select signals in LUTs [4, 19] or the engagement of the multiplexers participating in the dedicated carry logic [28], as well as the mapping of the multiplexers on the embedded multipliers of the FPGA [14].

Even if the design choices for the multiplexer are practically limited to the alternatives shown in Fig. 4, the design space for the arbiter is larger. The arbiter, apart from resolving any conflicting requests for the same resource, it should guarantee that this resource is allocated fairly to the contenders, granting first the input with the highest priority. Therefore, for a fair allocation of the resources, we should be able to change dynamically the priority of the inputs [12]. A generic dynamic priority arbiter (DPA), as shown in Fig. 5, consists of two parts: the arbitration logic that decides which request to grant based on the current state of the priorities and the priority update logic that decides, according to the current grant vector, which inputs to promote. The priority state associated with each input may be one or more bits, depending on the complexity of the priority selection policy.

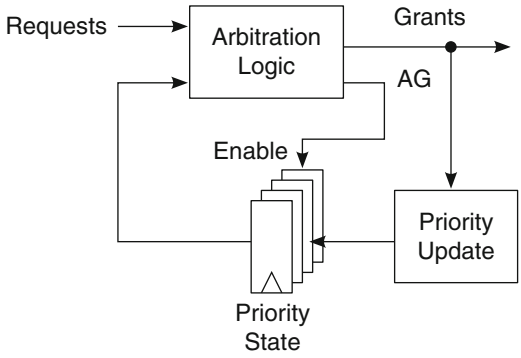


Fig. 5 Dynamic priority arbiter

For example, a single priority bit per input suffices for round-robin policy, while for more complex weight-based policies such as first come first served (FCFS), multibit priority quantities are needed [23].

Therefore, the combined mapping of the arbiter–multiplexer pair to the programmable logic and the interconnect of the FPGA needs further exploration for unveiling the area-delay characteristics of traditional arbiter and multiplexer structures borrowed from the ASIC domain and for quantifying the potential benefits of new proposals.

3 Separate Arbiter and Multiplexer Design Choices

The simplest form of arbitration, called fixed-priority arbitration or priority encoding, assumes that the priorities of the inputs are statically allocated and only the relative order of the inputs' connections determines the outcome of the arbiter. In this case, the request of position 0 (rightmost) has the highest priority and the request of position $N-1$ the lowest. For example, when an 8-port fixed-priority arbiter receives the request vector $(R_7 \ldots R_0) = 01100100$, it would grant input 2 since it has the rightmost active request. Two versions of an 8-port priority encoder driving a multiplexer are shown in Fig. 6. The first one involves a slow ripple-carry alternative, while the second is based on a fast parallel prefix structure.

In the case of fixed priorities, the combined operation of arbitration and multiplexing can be performed using only multiplexers. Such a structure is shown in Fig. 7. The fixed-priority order of assignment is implicitly implemented by the linear connection of the multiplexers, and thus the use of an arbiter is avoided. Despite its simplicity, the structure of Fig. 7 is only rarely used, mostly due to its increased delay.

Fixed priority is not an efficient policy, and hence it is not used in practice. On the contrary, round-robin arbitration is the most commonly used technique. Round-robin arbitration logic scans the input requests in a cyclic manner, beginning from the position that has the highest priority, and grants the first active request.

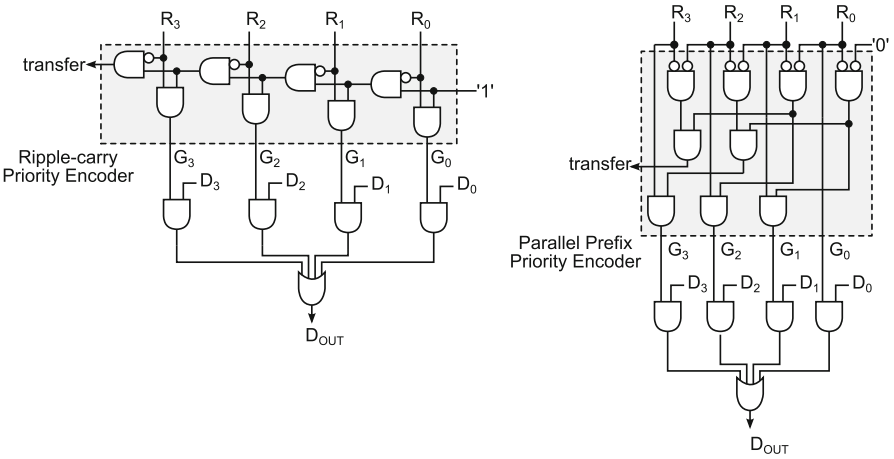
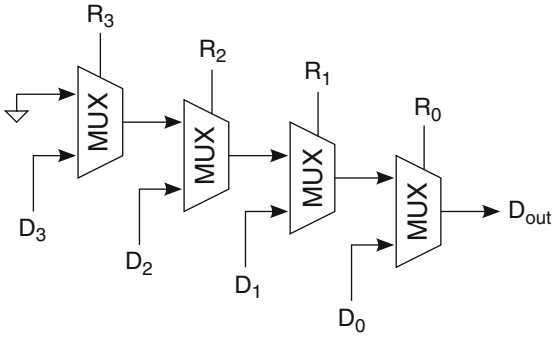


Fig. 6 Fixed-priority arbiter driving an AND-OR multiplexer

Fig. 7 A linear multiplexer implicitly implementing fixed priority



For the next arbitration cycle, the priority vector points to the position next to the granted input. In this way, the granted input receives the lowest priority in the next arbitration cycle. An example of the operation of a round-robin arbiter for four consecutive cycles is shown in Fig. 8 (the boxes labeled with a letter correspond to the active requests).

In the following, we will present three alternatives for the design of round-robin arbiters that are based on multiple priority encoders and on a customized carry-lookahead (CLA) structure. We focus on the implementation of the arbitration logic that scans the input requests in a cyclic manner. The design of the update logic is only briefly described since it consists of very simple modules that do not cause any timing violations. Besides, in a single cycle switch allocation and traversal, the delay of the pointer update logic is hidden, since it operates in parallel to the crossbar multiplexers.

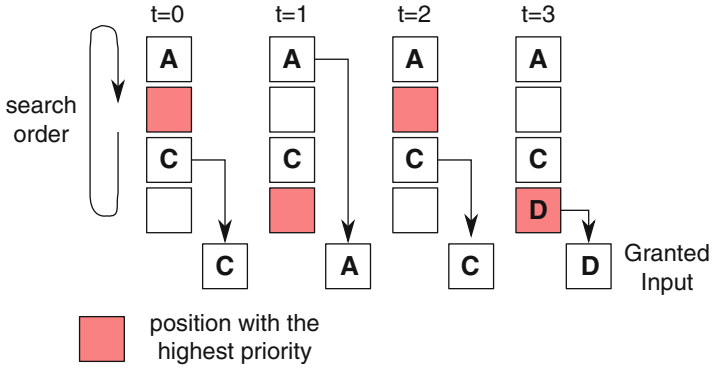
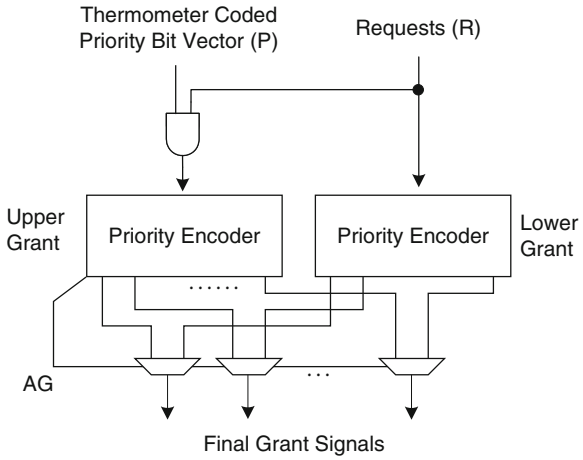


Fig. 8 Steps of round-robin arbitration on consecutive cycles

Fig. 9 PE-based round-robin arbiter



3.1 Priority Encoding-Based Round-Robin Arbiters

The design of a priority encoding (PE)-based round-robin arbiter [12] that searches for the winning request in a circular manner beginning from the position with the highest priority involves two priority encoders, as shown in Fig. 9. In order to declare which input has the highest priority, the priority vector P of the arbiter is thermometer-coded. For example, when $P = 11111000$ for an 8-input arbiter, position 3 has the highest priority. The upper PE of Fig. 9 is used to search for a winning request from the highest-priority position indexed by vector P (hereafter, we will refer to this position as Ppos), up to position $N-1$. It does not cycle back to input 0, even if it could not find a request among the inputs Ppos$\ldots N-1$. In order to restrain the upper PE to search only in positions Ppos$\ldots N-1$, the requests it receives are masked with the thermometer-coded priority vector P. On the contrary, the lower PE is driven by the original request lines and searches for a winning request among all positions.

The two arbitration phases work in parallel, and only one of them has computed the correct grant vector. The selection between the two outputs is performed by employing a simple rule. If there are no requests in the range $P\text{pos} \ldots N-1$, the correct output is the same as the output of a lower PE. If there is a request in the range $P\text{pos} \ldots N-1$, then the correct output is given by the output of the upper PE. Differentiating between the two cases is performed by using the AG signal of the upper PE (AG is asserted if any input has been granted). In the following, we will refer to this architecture as the dual-path PE arbiter.

In the dual-path PE arbiter, the grant vectors follow the one-hot encoding, while the priority vector is thermometer-coded. Therefore, in order to implement correctly the round-robin pointer update policy, the grant signals should be transformed to their equivalent thermometer code. This transformation is performed inside the pointer update logic of the arbiter.

3.2 LZC-Based Round-Robin Arbiters

Priority encoding identifies the position of the rightmost 1 on the request vector and keeps it alive at the output. At the same time, the remaining requests are killed, and the grant vector contains a single 1. Similarly, the process of leading-zero counting (or detection) counts the number of zeros that appear before the leftmost 1 of a word. If a transposed request vector is given to the leading-zero counter (LZC), then priority encoding and leading-zero counting are equivalent, since they both try to encode the position of the rightmost 1 in a digital word. The difference between the two methods is the encoding used to denote the selected position. In the case of priority encoding the grant vector is in one-hot form, while in the case of leading-zero counting, the output vector follows the weighted binary representation.

A round-robin arbiter that is based on LZCs can be designed by following again the dual-path approach presented in Fig. 9. The priority encoders are replaced by the corresponding LZCs that receive the requests transposed. In this case, the grant vector is composed of $\log_2 N$ bits that encode the position of the winning request, and it is connected directly to a tree of multiplexers that switch to the output the winning data, as shown in Fig. 10 (note that AZ is the All Zero signal of an LZC and is essentially the complement of the AG signal of a typical arbiter).

The most efficient LZC is presented in [10], where, for the first time, compact closed-form relations have been presented for the bits of the LZC. The iterative leading-zero counting equations can be fed directly to a logic synthesis tool and derive efficient LUT mappings. The employed LZC works in $\log_2 N$ stages, equal to the bits required for the weighted binary representation of the winning position. At each stage, the LZC computes one bit of the output by deciding, via the same operator, if the number of the leading zeros of the requests is odd or even. The first stage involves all the requests, while the following stages assume a reduced request vector. At each stage, the reduced request vector is produced by combining with an

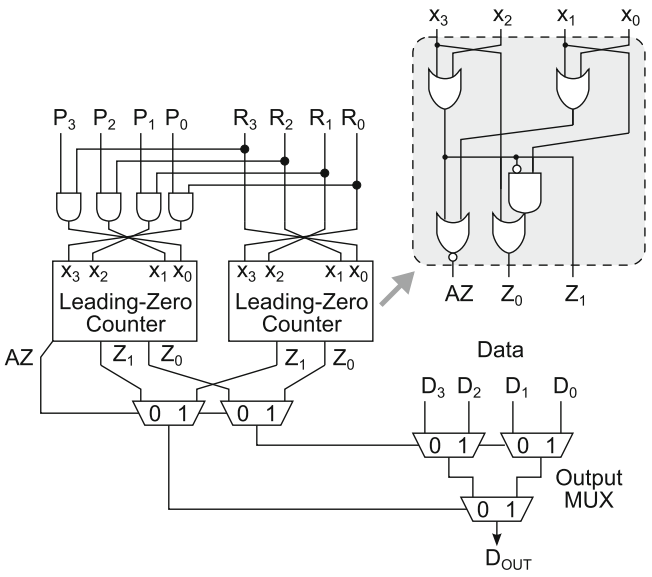


Fig. 10 An LZC-based round-robin arbiter driving a multiplexer

OR relation the nonconsecutive pairs of bits of the previous request vector. This OR reduction is equivalent to a binary tree of OR gates.

3.3 Carry-Lookahead-Based Round-Robin Arbiters

Alternatively, a round-robin arbiter can be built using CLA structures. In this case, the highest priority is declared using a priority vector P that is encoded in one-hot form. The main characteristic of the CLA-based arbiters is that they do not require multiple copies of the same circuit, since they inherently handle the circular nature of priority propagation [9]. In this case, the priority transfer to the ith position is modeled recursively via a priority transfer signal named X_i. The ith request gets the highest priority, i.e., $X_i = 1$, when either bit P_i of the priority vector is set or when the previous position $i-1$ does not have an active request ($R_{i-1} = 0$). Transforming this rule to a boolean relation we get that

$$X_i = P_i + \overline{R}_{i-1} \cdot X_{i-1} \tag{1}$$

When $X_i = 1$ it means that the ith request has the highest priority to win a grant. Therefore, the grant signal G_i is asserted when both X_i and R_i are equal to 1:

$$G_i = R_i \cdot X_i \tag{2}$$

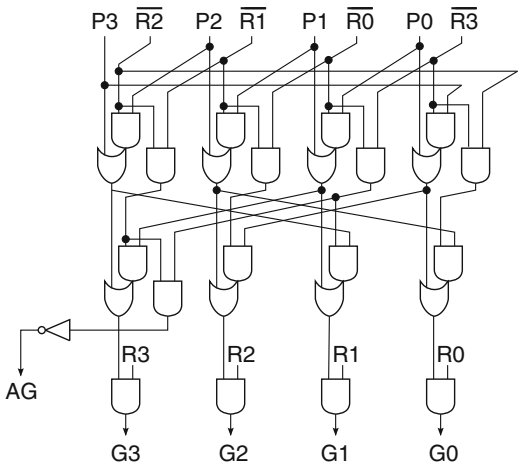


Fig. 11 The logic-level structure of a CLA-based round-robin arbiter

The search for the winning position should be performed in a circular manner after all positions are examined. Therefore, in order to guarantee the cyclic transfer of the priority, signal X_{N-1} out of the most significant position should be fed back as a carry input to position 0, i.e., $X_{-1} = X_{N-1}$. Of course, we cannot connect X_{N-1} directly to position 0 since this creates a combinational loop. In [9], an alternative fast circuit has been proposed that avoids the combinational loop and computes all X bits in parallel using the butterfly-like CLA structure shown in Fig. 11. Similar circuits can be derived after mapping on the FPGA the simplified and fully unrolled equations that describe the computation of the priority transfer signal X_i:

$$X_i = P_i + \sum_{j=0}^{n-2} \left(\prod_{k=j+1}^{n-1} \overline{R}_{|k+1|_N} \right) P_{|i+j+1|_N}, \tag{3}$$

where $|y|_N = y \mod N$. Finally, since both the grant vector and the priority vector are encoded in one-hot form, no extra translation circuit is required in the pointer update unit of this round-robin arbiter.

4 Merged Arbiter and Multiplexer

In this section, we present new soft macros that can handle concurrently arbitration and multiplexing. In this way, switch allocation and traversal can be performed efficiently in the same cycle, while still offering energy-delay efficient implementations. The design of these new efficient macros is based on an algorithmic transformation of round-robin arbitration to an equivalent sorting-like problem.

Similar to the PE-based round-robin arbiter of Sect. 3.1, the merged round robin arbiter and multiplexer utilizes an N-bit priority vector P that follows the thermometer code. As shown in the example of Fig. 12, the priority vector splits

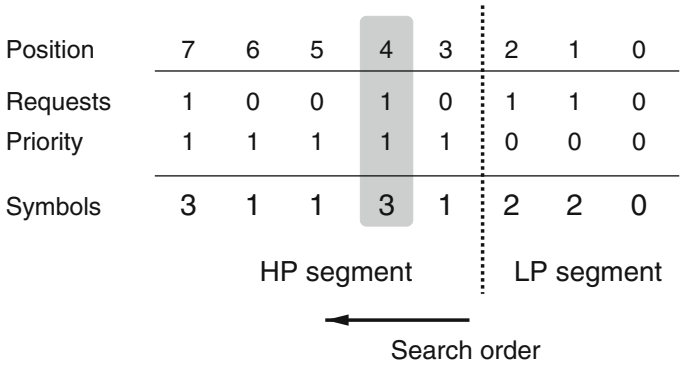


Fig. 12 The separation of input requests to HP and LP segments according to the contents of the priority vector and the transformation of the priority and the request vectors to arithmetic symbols

the input requests in two segments. The high-priority (HP) segment consists of the requests that belong to high-priority positions where $P_i = 1$, while the requests, which are placed in positions with $P_i = 0$, belong to the low-priority (LP) segment. The operation of the arbiter is to give a grant to the first (rightmost) active request of the HP segment and, if not finding any, to give a grant to the first (rightmost) active request of the LP segment. According to the already known solutions, this operation involves, either implicitly or explicitly, a cyclic search of the requests, starting from the HP segment and continuing to the LP segment.

Either at the HP or the LP segment, the pairs of bits (R_i, P_i) can assume any value. We are interested in giving an arithmetic meaning to these pairs. Therefore, we treat the bits R_iP_i as a 2-bit unsigned quantity with a value equal to $2R_i + P_i$. For example, in the case of an 8-input arbiter, the arithmetic symbols we get for a randomly selected request and priority vector are shown in Fig. 12. From the four possible arithmetic symbols, i.e., 3, 2, 1, 0, the symbols that represent an active request are either 3 (from the HP segment) or 2 (from the LP segment). On the contrary, the symbols 1 and 0 denote an inactive request that belongs to the HP and the LP segment, respectively.

According to the described arbitration policy and the example priority vector of Fig. 12, the arbiter should start looking for an active request from position 3, moving upwards to positions 4, 5, 6, 7, and then to 0, 1, 2 until it finds the first active request. The request that should be granted lies in position 4, which is the first (rightmost) request of the HP segment. Since this request belongs to the HP segment, its corresponding arithmetic symbol is equal to 3. Therefore, granting the first (rightmost) request of the HP segment is equivalent to giving a grant to the first maximum symbol that we find when searching from right to left. This general principle also holds for the case that the HP segment does not contain any active request. Then, all arithmetic symbols of the HP segment would be equal to 1, and any active request of the LP segment would be mapped to a larger number (arithmetic symbol 2).

Therefore, by treating the request and the priority bits as arithmetic symbols, we can transform the round-robin cyclic search to the equivalent operation of selecting the maximum arithmetic symbol that lies in the rightmost position. Searching for the maximum symbol and reporting at the output only its first (rightmost) appearance, implicitly implements the cyclic transfer of the priority from the HP to the LP segment, without requiring any true cycle in the circuit. In principle, any maximum selector does not contain any cycle paths and is built using a tree or a linear comparison structure. The proposed arbiter is built using a set of small comparison nodes. Each node receives two arithmetic symbols, one coming from the left and one from the right side. The maximum of the two symbols under comparison appears at the output of each node. Also, each node generates one additional control flag that denotes if the left or the right symbol has won, i.e., it was the largest. In case of a tie, when equal symbols are compared, this flag always points to the right. In this way, the first (rightmost) symbol is propagated to the output as dictated by the operation of the arbiter.

In every case, the winning path is clearly defined by the direction flags produced by the comparison nodes. Thus, if we use these flags to switch the data words that are associated with the corresponding arithmetic symbols, we can route at the output the data word that is associated with the winning request. This combined operation can be implemented by adding a 2-to-1 multiplexer next to each comparison node and connecting the direction flag to the select line of the multiplexer. The structure of both a binary tree and a linear merged arbiter multiplexer with 8 inputs, along with a running example of their operation, is shown in Fig. 13. Following the example, we observe that the first, in a round-robin order, data word A_4 is correctly routed at the output.

Although the tree-structured merged arbiter multiplexer has smaller delay than the linear-structured one, the latter can take advantage of the dedicated mux-carry logic of the FPGA.

4.1 Computation of the Grant Signals

The merged arbiter multiplexer, besides transferring at the output the data word of the granted input, should also return in a useful format the position of the winning request (or equivalently the grant index). The proposed maximum-selection tree, shown in Fig. 13a, that replaces the traditional round-robin arbiter can be enhanced to facilitate the simultaneous generation of the corresponding grant signals via the flag bits of the CMP nodes.

At first, we deal with the case that the grants are encoded in weighted binary representation. In this case, we can observe that, by construction, the weighted binary encoding of the winning request is formed by putting together the flag bits of the CMP nodes that lie in the path from the winning input to the root of the tree (see Fig. 14a). Consequently, the generation of the grant signals in weighted binary representation is done by combining at each level of tree the winning flag bits of

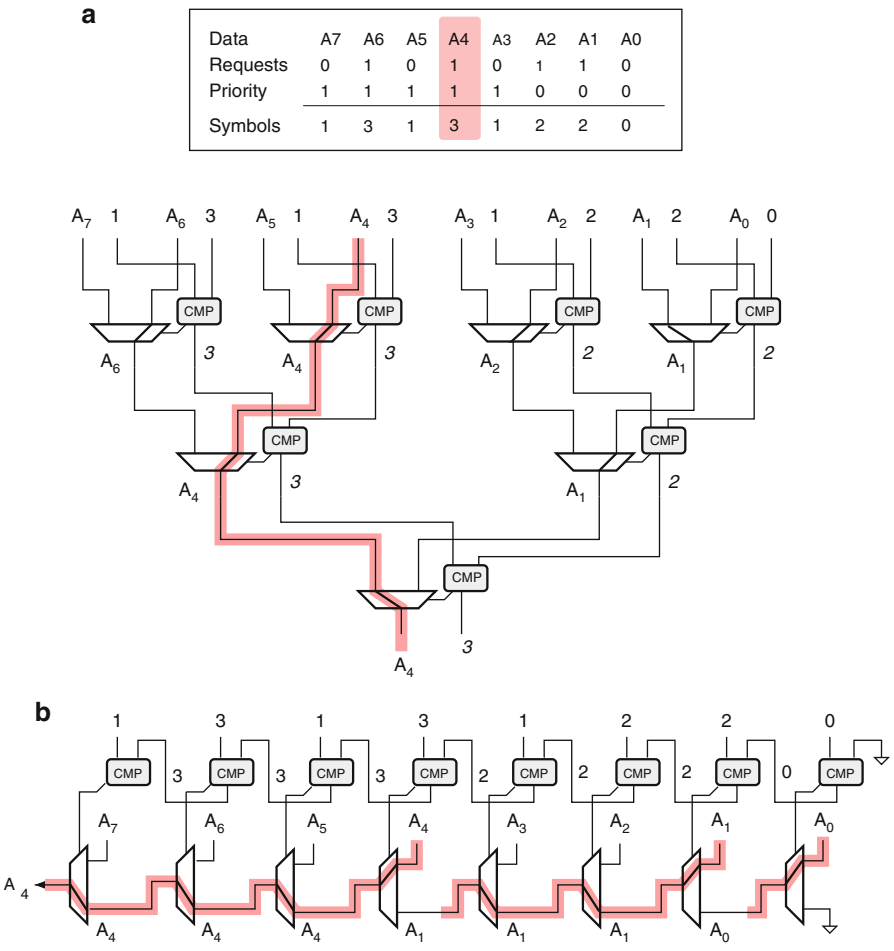


Fig. 13 The merged arbiter multiplexer: (**a**) Tree and (**b**) linear comparison structure

the previous levels with the flags of the current level. This is achieved by means of some additional multiplexers, as shown in Fig. 14a, for the case of a tree-based merged arbiter multiplexer.

For the one-hot encoding, we need a different implementation. Initially, i.e., at the inputs of the one-hot grant generation circuit, we assume that every position has a grant signal equal to 1. At the following levels, some of these grant signals are transformed to 0s if their associated symbols are not the winning ones at the corresponding CMP nodes. Thus, at the output, only a single 1 will remain and the rest would be nullified. The circuit that generates the corresponding grant signals in one-hot form, for 4 input symbols, is shown in Fig. 14b. Keeping and nullifying the grant signals is performed by the AND gates that mask, at each level of the

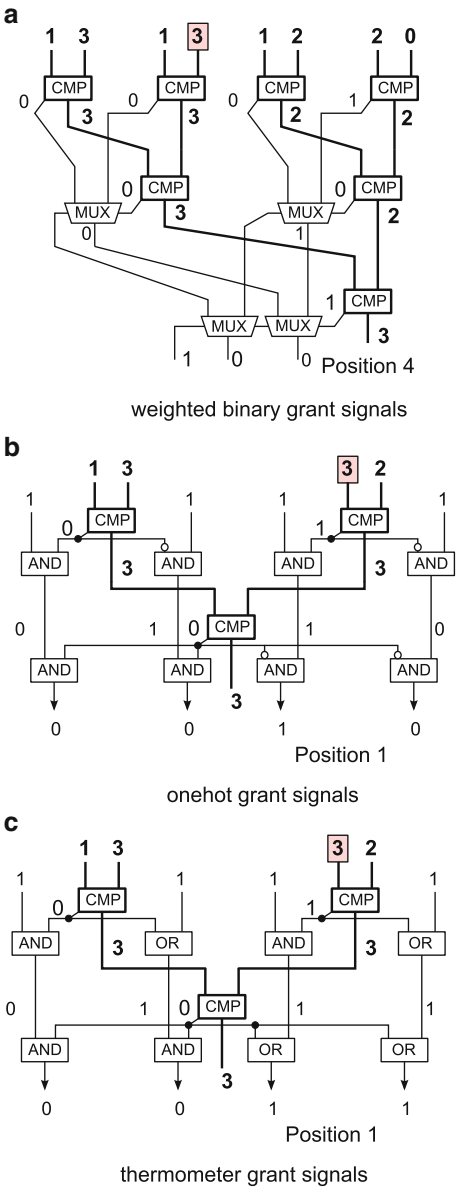


Fig. 14 The grant generation circuits that run in parallel to the CMP nodes for a tree organization of the merged arbiter multiplexer

tree, the intermediate grant vector of the previous level with the associated direction flags. The inversions are needed to keep alive the grant signals that correspond to a winning symbol of the right subtrees. Observe that, if we replace the invert-AND gates of Fig. 14b with OR gates, the outcome would be a thermometer-coded grant vector instead of the one-hot code. The resulting circuit is shown in Fig. 14c.

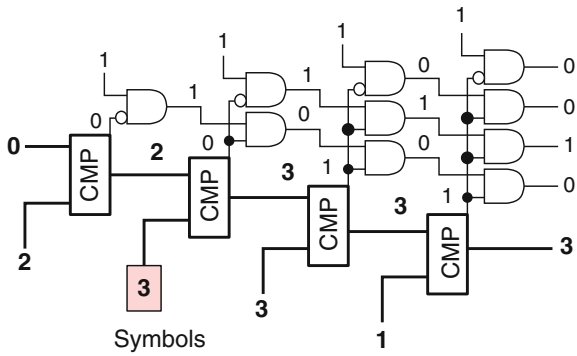


Fig. 15 The one-hot grant generation circuit for the linear organization of the merged arbiter multiplexer

In this way, with minimum cost, we are able to fully cover all possible useful grant encodings, thus alleviating the need for additional translation circuits.

When the linear comparison structure is selected for the organization of the merged arbiter multiplexer, we can design the grant generation circuits following a similar procedure. A one-hot grant generation circuit for the case of a 4-input merged arbiter multiplexer is shown in Fig. 15. The AND gates at each comparison stage are driven by the direction flags of the CMP nodes and a constant 1 that allows us to simplify the invert-AND gates to inverters. Again, if the invert-AND gates are replaced by OR gates, a thermometer code word can be derived for the grant signals.

As for the weighted binary grant generation circuit, we can use the same linear structure of Fig. 13b, replacing the data words that drive the multiplexers with the various position indices in weighted binary format (i.e., in Fig. 13b, A_0 is replaced by 000, A_1 by 001, etc.). This means that, in this case, two multiplexers are needed at each stage of the linear structure, one for switching the data and another for switching the weighted binary indices.

When there is no active request, the arbiter should deassert the AG signal. This case is identified by observing the symbol at the output of the comparison structure. When it is equal to either 0 or 1, it means that no active request exists in either priority segment.

4.2 Switches with Merged Arbiter-Multiplexer Structures

The design of switches that use the proposed round-robin merged arbiter multiplexers (MARX) is straightforward. Figure 16a depicts the design of a VC-less switch using the proposed macros. This is the simplest case, where the arbiter-multiplexer pairs that existed at each output are directly replaced by the proposed units. As in any switch, the data placed on the input registers or the head of the input queues should not be changed or dequeued until the corresponding input is granted access to the requested output port.

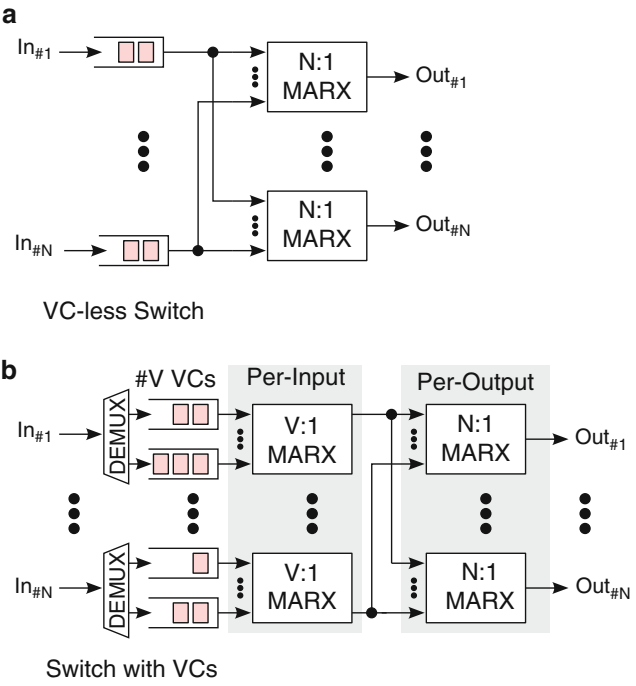


Fig. 16 Switches built with MARX units: **(a)** VC-less wormhole switch and **(b)** VC-based switch

In the case of switches with VCs, the design is more complicated due to the per-input and per-output stages of arbitration and multiplexing. The proposed macros fit better in the case of input-first allocation. This organization is shown in Fig. 16b. The per-input MARX units select locally an eligible VC among the V available and carry along its corresponding flit. The VCs selected from each input compete for gaining access to their requested outputs via the per-output MARX units that simultaneously resolve any conflicts and give at the output the flit of the winning VC.

5 Experimental Results

In this section, we explore the implementation characteristics of the presented designs. The analysis that follows aims to identify the fastest and/or the most area-efficient alternative by varying the number of ports of the arbiter and multiplexer and the data width of each port. For attaining our comparison data, we first generated the equivalent VHDL descriptions of all designs under comparison. After extensive simulations that verified the correctness of each description, each design was synthesized and mapped to a Virtex-5 XC5VLX330 FPGA chip. For the synthesis, mapping, and placement and routing of the designs, we used the ISE 12.2 toolset

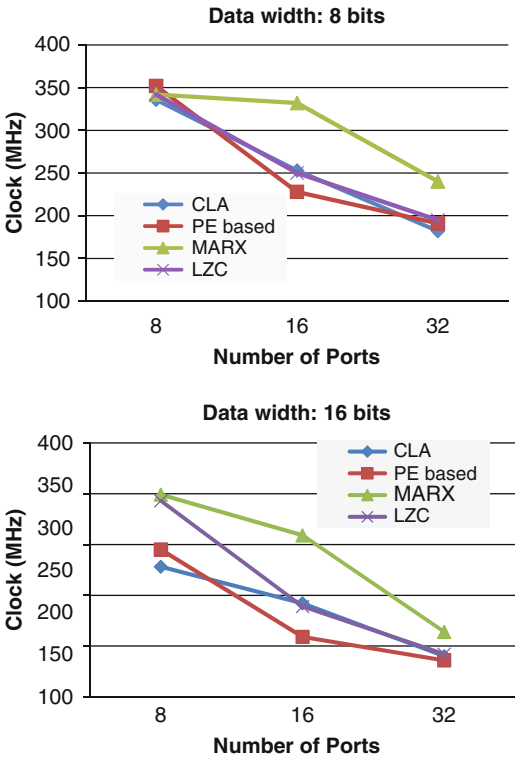


Fig. 17 The delay of arbiters and multiplexers varying the number of ports for 8 and 16 bits port width

of Xilinx. Please note that the reported results involve only the optimizations performed by the CAD tools alone, without any manual intervention that would further optimize the circuits under comparison. In this way, the presented results can be reproduced by every designer by just following the same automated design flow.

At first, we compare the presented design alternatives in terms of delay. Delay is critically affected by the number of ports that the circuit is designed to serve, as well as the width of the corresponding data words that increases the loading of the multiplexers' control signals. The best delays achieved for each circuit after varying the number of inputs and keeping constant the data width to 8 and 16 bits are shown in Fig. 17. From the presented results that were measured after place and route, we can draw several conclusions. The merged arbiter multiplexer (we refer to the tree-structured implementation) is, in all cases, the fastest, and the delay savings are more than 20 % on average. This trend is followed irrespective of the number of bits used per multiplexer port. The observed delay convergence, when the number of ports increases, is attributed to the aggravated effect of routing interconnect delay that constitutes more than 80 % of the total path delay. This is a sign that such wide multiplexers of single-stage switching systems should be avoided, and

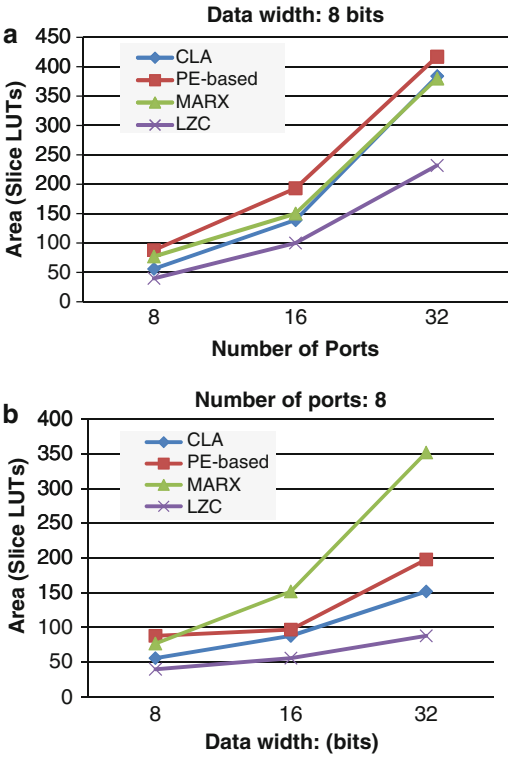


Fig. 18 The area of the arbiters and multiplexers when varying (**a**) the number of ports at a constant port width of 8 bits and (**b**) the width of each port of an 8-input circuit

the communication among multiple modules should be organized as a network of switches. Observe though that, even for such extreme multiplexer widths, the delay advantage of MARX is considerable.

The area of the examined designs is reported in Fig. 18. Specifically, Fig. 18a reports the area occupied by the compared designs assuming 8 bits per port and varying the number of ports. On the other hand, Fig. 18b shows the area of all the designs for an 8-input arbiter and multiplexer when varying the width of each port. The most clear conclusion derived from both figures is that the LZC-based arbiter and multiplexer is the most area-efficient solution requiring roughly 30 % less area on average. On the contrary, the fastest design, i.e., the merged arbiter and multiplexer, although it behaves similarly to the other designs for small port widths, requires significantly more area when the bits per port are increased to 16 and 32 bits. This behavior though enables the designer to explore the area-delay trade-off; the MARX allows for very fast implementations with the overhead of extra area for increased data widths, whereas the LZC-based design offers low implementation cost with fairly small delays.

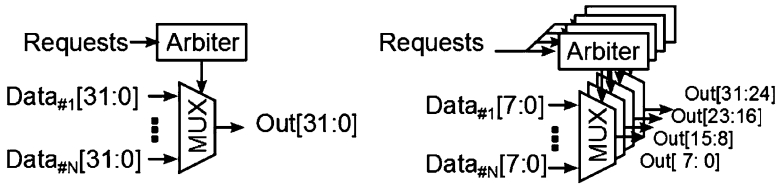


Fig. 19 The application of bit slicing to arbiter and multiplexer pairs

5.1 Bit Slicing

The multiplexer can be sliced to smaller multiplexers with the same number of input ports but of smaller data width. This operation is equivalent to spreading parts of input words to multiple smaller multiplexers, where each multiplexer is driven by a dedicated arbiter. As shown in Fig. 19, the control logic of the independent multiplexers remains unified, and each submultiplexer receives the same grant decisions. This happens since all arbiters work in sync, receiving the same requests and updating, in the same way, the priority of each position. Bit slicing partially alleviates the high-fanout problem of the grant signals and may offer higher-speed solutions. In reality, the fanout taken from the grant signals is given to the request lines that now need to be broadcasted to more arbiters.

In the following, we investigate the delay benefits of bit slicing and try to identify which slicing factor is the best for the designs under comparison. We applied bit slicing on an 8-input and a 16-input arbiter and multiplexer carrying data of 32 bits. We used all power-of-two slicing factors SF between the two extremes: SF = 1 that corresponds to no slicing and SF = 32 that corresponds to full slicing, where each bit has its own arbiter. In the general case, bit slicing by a factor SF means that we implement SF multiplexer and arbiter pairs, with each multiplexer carrying 32/SF bits. The obtained results that clearly depict an optimum slicing factor for each design are shown in Fig. 20.

6 Summary and Conclusions

In this chapter we presented and compared various alternatives for the design of an arbiter and a multiplexer in an FPGA. The design space includes traditional separate arbiter and multiplexer pairs, as well as recently introduced merged arbiter and multiplexer macros that handle arbitration and multiplexing concurrently. Although the mapping of multiplexers in LUT logic has received a lot of attention in the

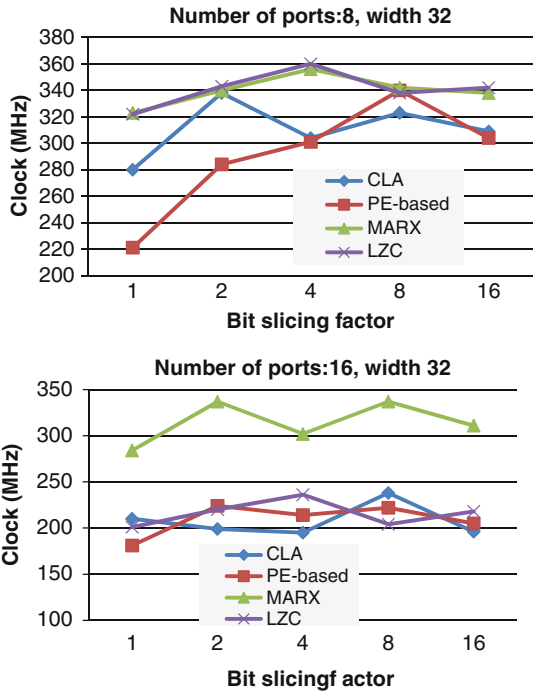


Fig. 20 The delay of all 8-input and 16-input circuits for all power-of-two bit slicing factors

previous years, the combined implementation of an arbiter and a multiplexer was partially unexplored. This work covers this gap and extends, at the same time, the design space with new efficient solutions that simplify the design of high-radix soft switches.

References

1. Altera (2011) Applying the benefits of network on a chip architecture to FPGA system design. Tech. Rep., Jan 2011
2. Azimi M, Dai D, Mejia A, Park D, Saharoy R, Vaidya AS (2009) Flexible and adaptive on-chip interconnect for terascale architectures. Intel Tech J 13(4):62–77
3. Becker DU, Dally WJ (2009) Allocator implementations for network-on-chip routers. In: Proceedings of the ACM/IEEEE international supercomputing conference, 2009
4. Bhatti NK, Shingal N (2009) LUT based multiplexers. US Patent 7,486,110, Feb 2009
5. Brebner G, Levi D (2003) Networking on chip with platform FPGAs. In: Proceedings of the IEEE international conference on field-programmable technology, Dec 2003, pp 13–20
6. Dally WJ (1990) Virtual-channel flow control. In: Proceedings of the 17th annual international symposium computer architecture (ISCA), May 1990, pp 60–68
7. Dally WJ, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: Proceedings of the 38th design automation conference (DAC), June 2001, pp 684–689
8. Dally WJ, Towles B (2004) Principles and practices of interconnection networks. Morgan Kauffman, San Francisco

9. Dimitrakopoulos G, Chrysos N, Galanopoulos C (2008) Fast arbiters for on-chip network switches. In: IEEE international conference on computer design (ICCD), 2008, pp 664–670
10. Dimitrakopoulos G, Galanopoulos K, Mavrokefalidis C, Nikolos D (2008) Low-power leading-zero counting and anticipation logic for high-speed floating point units. IEEE Transactions on very large scale integration (VLSI) Systems (16)7:837–850
11. Galles M (1997) Spider: a high-speed network interconnect. IEEE Micro 17(1):34–39
12. Gupta P, McKeown N (1999) Designing and implementing a fast crossbar scheduler. IEEE Micro 19(1):20–28
13. Hurt J, May A, Zhu X, Lin B (1999) Design and implementation of high-speed symmetric crossbar schedulers. In: IEEE international conference on communications (ICC), June 1999, pp 253–258
14. Jamieson P, Rose J (2005) Mapping multiplexers onto hard multipliers in FPGAs. In: Proceedings of IEEE NewCAS conference, June 2005, pp 323–326
15. Kapre N, Mehta N, Delorimier M, Rubin R, Barnor H, Wilson MJ, Wrighton M, Dehon A (2006) Packet-switched vs. time-multiplexed FPGA overlay networks. In: Proceedings of the IEEE symposium on field-programmable custom computing machines, 2006, pp 205–215
16. Lee J, Shannon L (2010) Predicting the performance of application specific NoCs implemented on FPGAs. In: Proceedings of the 18th annual ACM/SIGDA international symposium on field programmable gate arrays - FPGA 10. ACM Press, New York, 2010, pp 23–32
17. Lu Y, McCanny J, Sezer S (2011) Generic low-latency NoC router architecture for FPGA computing systems, fpl, In: 21st international conference on field programmable logic and applications, 2011, pp 82–89
18. Matsutani H, Koibuchi M, Amano H, Yoshinaga T (2009) Prediction router: yet another low latency on-chip router architecture. In: Proceedings of the 15th IEEE international symposium on high-performance computer architecture (HPCA), Feb. 2009, pp 367–378
19. Metzgen P, Nancekievill D (2005) Multiplexer restructuring for FPGA implementation cost reduction. In: Proceedings of the 42nd design automation conference, 2005, pp 421–426
20. Mukherjee SS, Silla F, Bannon P, Emer JS, Lang S, Webb D (2002) A comparative study of arbitration algorithms for the Alpha 21364 pipelined router. In: Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X), 2002, pp 223–234
21. Mullins RD, West AF, Moore SW (2004) Low-latency virtual-channel routers for on-chip networks. In: Proceedings of the 31st annual international symposium on computer architecture (ISCA), 2004, pp 188–197
22. Peh L.-S, Dally WJ (2001) A delay model and speculative architecture for pipelined routers. In: Proceedings of the 7th international symposium on high-performance computer architecture (HPCA-7), 2001
23. Pirvu M, Bhuyan L, Ni N (1999) The impact of link arbitration on switch performance. In: Proceedings of the 5th high-performance computer architecture (HPCA), 1999, pp 228–235
24. Saldana M, Shannon L, Craig J, Chow P (2007) Routability of network topologies in FPGAs. IEEE Transactions on very large scale integration (VLSI) Systems 15(8):948–951
25. Shelburne M, Patterson C, Athanas P, Jones M, Martin B, Fong R (2008) MetaWire: using FPGA configuration circuitry to emulate a network-on-chip. In: Proceedings of the 2008 international conference on field programmable logic and applications, Sept 2008, pp 257–262
26. Tamir Y, Chi H.-C (1993) Symmetric crossbar arbiters for VLSI communication switches. IEEE Trans Parallel Distr Syst 4(1):13–27
27. Vassiliadis S, Sourdis I (2007) FLUX interconnection networks on demand. J Syst Architect 53(10):777–793
28. Wittig RD, Mohan S (2003) Method for implementing large multiplexers with FPGA lookup tables. US Patent 6,505,337 B1, Jan 2003