2011 21st International Conference on Field Programmable Logic and Applications

Scalable arbiters and multiplexers for on-FGPA interconnection networks

Giorgos Dimitrakopoulos
Informatics & Communications Engineering
University of West Macedonia, Greece

Christoforos Kachris
Networks Lab
Athens Information Technology (AIT), Greece

Emmanouil Kalligeros
Information & Comm. Systems Engineering
University of the Aegean, Greece

***Abstract*—Soft on-FGPA interconnection networks are gaining increasing importance since they simplify the integration of heterogeneous components and offer, at the same time, a modular solution to the complex system-wide communication issues. The switches are the basic building blocks of such interconnection networks and their design critically affects the performance of the whole network. The way data traverse each switch is governed by the operation of the arbiter and the crossbar's multiplexers that need to be efficiently mapped on the FPGA fabric under tight area and delay constraints. This paper explores the design space of an arbiter and a multiplexer as a unified entity and proposes two new circuit alternatives that allow the design of scalable soft FPGA switches.**

I. Introduction

Interconnection networks lie at the kernel of any complex system-on-chip. They offer a modular communication infrastructure, while they parallelize the communication between system modules by utilizing a network of switches connected with multiple point-to-point links. Such on-chip interconnection networks are already a mainstream technology for ASICs [1], while they gain significant importance in FPGA-based SOCs [2]. 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 tradeoffs shift and they should appropriately adopted to the characteristics of the implementation fabric [3], [4].

There are several forms of interconnection networks [5]. Besides traditional packet switched architectures, the reconfigurable nature of the FPGA logic allows the design of other customized alternatives that try to exploit in the best way the logic of the FPGA fabric [6], [7], [8], [9] and the already existent configuration network [10]. Nevertheless, in this paper, we take a generic approach and focus on the basic components of a soft dynamically switched interconnection network, such as wormhole or virtual-cut-through, with single or multiple-word packets. In such networks, a critical factor to overall system's efficiency is the selection of the appropriate network topology that would reduce the communication distance and utilize efficiently the on-chip wiring resources. However, reducing the communication distance between any two nodes comes with a penalty, since the smaller the number of hops, the larger the radix (number of input and output ports) of the switches at each node of the network [5]. Increasing the radix of the switches increases significantly their area and creates hard to solve timing problems.

The basic components of the switch that are immediately affected by the increased radix is the arbiter and the multiplexer. The arbiter decides which inputs are allowed to send their data and the multiplexer performs the data switching. The LUT mapping of wide multiplexers has been well explored, either by enhancing the features of the corresponding mapping algorithms, or by fully exploiting the structure and the additional features of the FPGA logic blocks [11], [12]. However, the combined mapping of an arbiter and a multiplexer as a whole to the programmable logic and interconnect of an FPGA has not been sufficiently explored.

Our study aims in bridging this gap (a) by revealing the area/delay characteristics of traditional arbiter and multiplexer structures borrowed from the ASIC domain, and (b) by proposing two new circuit architectures that offer significant benefits over previous state-of-the-art and allow the construction of scalable soft switches. In every case, we present the most efficient alternative as it is derived by following an automated design flow without any manual intervention to the mapping process. In the following, we discuss the baseline arbiter and multiplexer structures and comment on the logic-design-based implications that guide their design (Section 2). The proposed architectures are presented in Section 3, while the experimental results that prove the benefits of the new circuits, are reported in Section 4. Finally, conclusions are drawn in Section 5.

II. Baseline arbiters and multiplexers

The design of arbiters and multiplexers is closely interrelated. Arbiters receive the requests from all inputs and decide to grant only one of them. The grant signals are sent to the multiplexers' select lines, and in many cases, they are also sent back to the inputs informing them about the acceptance of their requests. The serial connection between the arbiters and the multiplexer affects their structure and their LUT mapping, as well as their combined area and delay characteristics.

The encoding selected for the grant signals directly affects the design of the multiplexers. Depending on the encoding of the grant decisions we have two options regarding the

978-0-7695-4529-5/11 $26.00 © 2011 IEEE
DOI 10.1109/FPL.2011.26

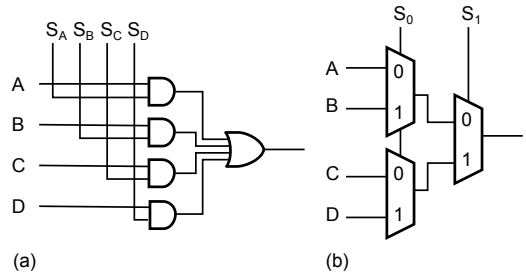

Figure 1. The multiplexer logic level implementation alternatives: (a) The AND-OR implementation and (b) the tree of smaller multiplexers.

design of the corresponding multiplexers. In the first case, shown in Fig. 1(a), the grant decision is encoded in onehot form, where only a single bit is set, and the multiplexer is implemented using an AND-OR structure. On the contrary, in Fig. 1(b), 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 a weighted binary representation. [1]

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 [11], or the engagement of the multiplexers participating in the dedicated carry logic [12], [13], as well as the mapping of the multiplexers on the hard multipliers of the FPGA [14].

Even if the design choices for the multiplexer are practically limited to the alternatives shown in Fig. 1, the design space for the arbiter is larger. This characteristic stems from the fact that 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. For example, the most widely used round-robin policy dictates that the request served in the current cycle gets the lowest priority for the next arbitration cycle.

The block diagram of a general Variable Priority Arbiter is shown in Fig. 2. It consists of two parts; the arbitration logic that decides which request to grant based on the current state of the priority vector, and the pointer update logic that decides according to the current grant vector, which input to promote. Round-robin arbitration logic scans the input requests in a cyclic manner beginning from the position that has the highest priority. If after scanning all inputs the arbiter

[1]There is a third choice for design of a multiplexer that utilizes the weighted binary representation for the select signals and still it is built with an AND-OR structure. Such a structure requires also a decoder that translates the weighted binary select signals to their onehot equivalents.

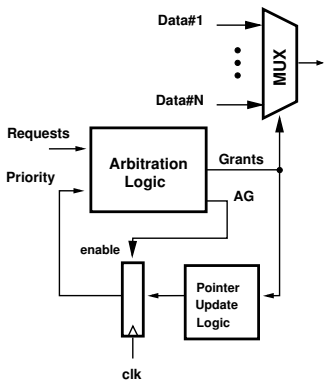

Figure 2. The block diagram of a VPA.

does not find an active request it de-asserts the AG (Any Grant) signal that declares that no input was granted.

A. Priority-encoding based arbiters

The simplest VPA is based on priority encoders [15]. A priority encoder (PE) is a fixed priority arbiter that always gives to position 0 the highest priority. It grants the ith request, i.e., $G_i = 1$, when $R_i = 1$ and no other request exists to a position with an index smaller than i. This condition can be computed by the well-known priority encoding relation as follows: $G_i = R_i \cdot \overline{R}_{i-1} \cdot \ldots \cdot \overline{R}_1 \cdot \overline{R}_0$, where $\cdot$ represents the boolean-AND operation and $\overline{R}_i$ denotes the complement of R_i.

The design of a PE-based VPA 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. 3. In order to declare which input has the highest priority, the priority vector P of the VPA is thermometer coded. For example, when P=11111000 for an 8-input arbiter, position 3 has the highest priority. The upper PE of Fig. 3 is used to search for a winning request from the position denoted by the priority vector P, i.e., $Ppos$, 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 $N-1 \ldots Ppos$, the requests that 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 $Ppos \ldots N-1$, the correct output is the same as the output of a lower PE. If there is a request in the range $Ppos \ldots N-1$, 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 FPA. In the following, we will refer to this architecture as the dual-path PE arbiter.

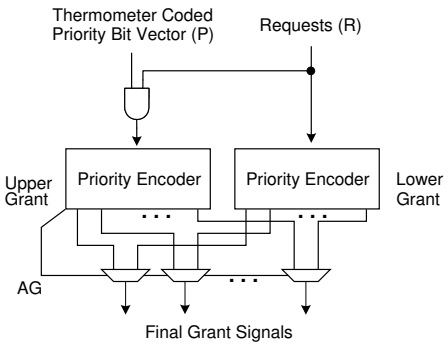

Figure 3. The dual-path priority-encoder-based VPA.

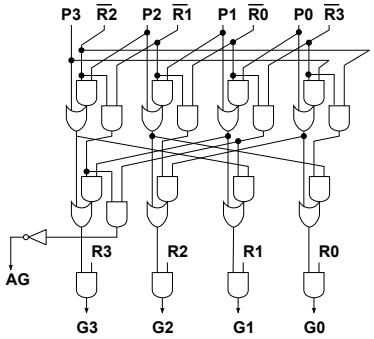

Figure 4. The logic-level structure of a CLA-based VPA.

In the dual-path PE arbiter, the grant vectors follow the onehot 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 VPA and it does not impose any delay overhead since it runs in parallel to the multiplexer.

B. Carry-lookahead based arbiters

Alternatively, a VPA can be built using carry-lookahead (CLA) structures. In this case, the highest priority is declared using a priority vector P that is encoded in onehot form. The main characteristic of the CLA arbiters is that they do not require multiple copies of the same circuit and they inherently handle the circular nature of priority propagation [16].

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: $X_i = P_i + \overline{R}_{i-1} \cdot X_{i-1}$. 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: $G_i = R_i \cdot X_i$.

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, the signal X_{N-1}, out of the most-significant position should be fed back as a carry input to position 0, $X_{N-1} = X_{-1}$. Of course we cannot connect X_{N-1} directly to position 0 since this creates a combinational loop. In [16] 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 4. 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:

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

where, $|x|_N = x \mod N$. Finally, since both the grant vector and the priority vector are encoded in onehot form, no extra translation circuit is required in the pointer-update unit of the VPA.

In both cases of the baseline VPAs, the grant outputs are given in onehot form. This form restricts the designer to use only the AND-OR multiplexers of Fig. 1(a). The combination of the baseline arbiters with a tree multiplexer similar to the one shown in Fig. 1(b) does not show any delay benefits since it requires the addition of an encoder that translates the onehot grant signals to their equivalent weighted binary representation.

III. Proposed alternatives

In this section, we present two alternatives for the design of efficient arbiters and multiplexers that offer significant benefits, both in terms of delay and area, relative to the baseline configurations. The first one introduces the implementation of a PE-based arbiter via a leading-zero counter, and the second one transforms the round-robin operation of a VPA to an equivalent operation that allows the merging of arbitration and multiplexing into a new unified operation.

A. Leading zero counting based 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, 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

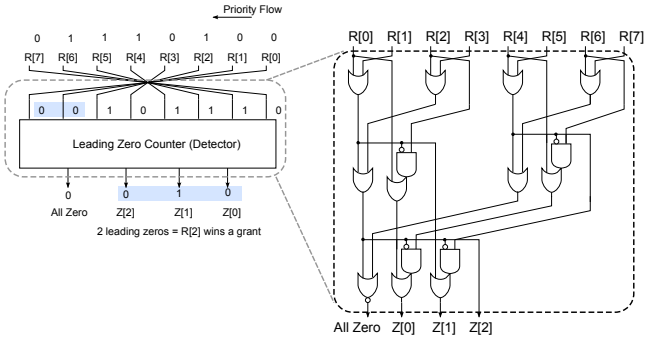

Figure 5. The structure of an LZC used as a priority encoder.

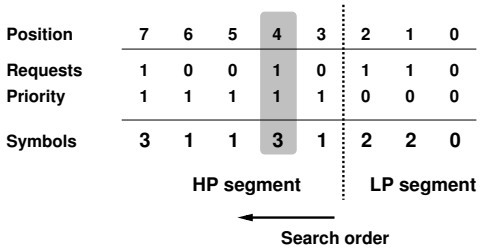

Figure 6. The transformation of the priority and the request vector to arithmetic symbols.

case of priority encoding the grant vector is in onehot form, while in the case of leading-zero counting the output vector follows the weighted binary representation.

A VPA that is based on leading-zero counters can be designed by following again the dual-path approach presented in Fig. 3. The priority encoders are replaced by the corresponding leading-zero counters 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.

The most efficient LZC is presented in [17], 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 that are equal to the bits required for the weighted binary representation of the winning position. In 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 OR relation the non-consecutive pairs of bits of the previous request vector. This OR reduction is equivalent to a binary tree of OR gates.

B. Merged arbitration and multiplexing

The second proposed circuit does not resemble in any part either the baseline or the LZC-based arbiter. It is based on the transformation of the cyclic search among the requests imposed by round-robin policy, to an equivalent cyclic-free operation that enables the merging of arbitration and multiplexing. We will present the proposed method via an example, assuming again that the priority vector P of the pointer-update logic is thermometer coded.

As shown in the example of Fig. 6, the priority vector splits 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. 6. From the 4 possible arithmetic symbols, i.e., 3, 2, 1, 0, the symbols that represent an active request are either 2 (from the LP segment) or 3 (from the HP 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. 6, 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 contains 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 map 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

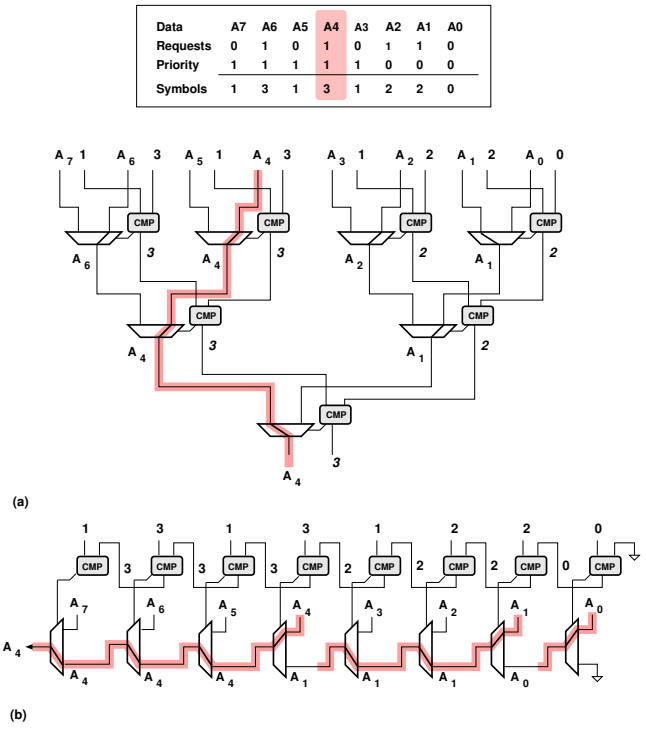

Figure 7. The merged arbiter multiplexer: (a) Tree and (b) linear comparison structure.

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 (MARX) with 8 inputs, along with a running example of their operation is shown in Fig. 7. Following the example we observe that the first, in a round-robin order, data word A_4 is correctly routed at the output without requiring any cyclic priority transfer. Although, the tree-structured MARX has smaller delay than the linear-structured one, the latter can take advantage of the dedicated mux-carry logic of the FPGA.

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.

IV. Experimental results

In this section, we compare the proposed designs against the baseline solutions for the arbiter and multiplexer pairs. The comparisons aim 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 FPGA chip XC5VLX330. For the synthesis, mapping, and placement & routing of the designs we used the ISE 12.2 toolset 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 re-produced by the general designer by just following the same automated design flow.

At first, we compare the proposed designs and the baseline configurations 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* delay 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. 8. From the presented results that were measured after place & route, we can draw several conclusions. The proposed designs, and especially the merged arbiter multiplexer (we refer to the tree-structured implementation) is, in all cases, the fastest, and the delay saved is 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 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. 9. For example, Fig. 9(a) reports the area occupied by the compared designs assuming 8 bits per port and varying the number

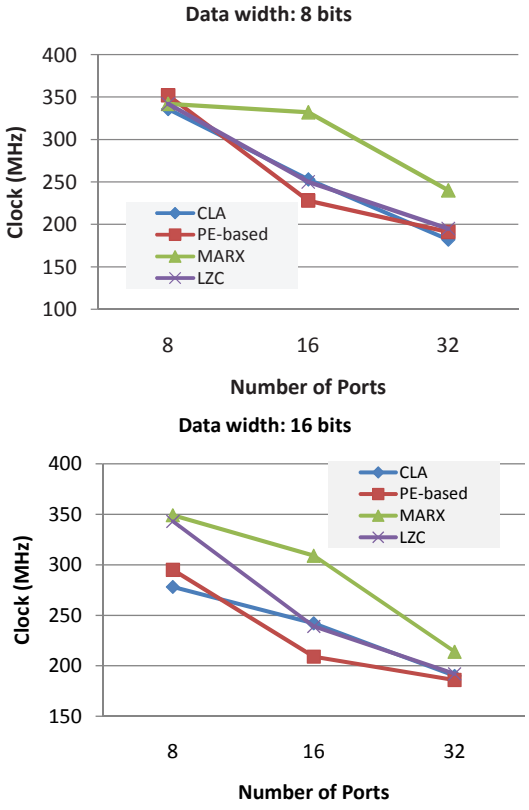

Figure 8. The delay of arbiters and multiplexers varying the number of ports for 8 and 16 bits port width.

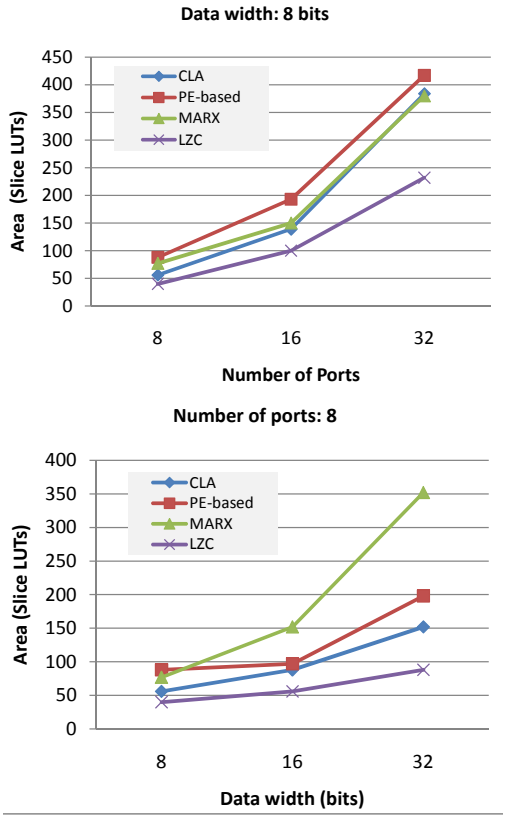

Figure 9. The area of the arbiters and multiplexers when varying (a) the number of ports at a constant port width of 8 bits and (b) the width of each port of an 8-input circuit.

of ports. On the other hand, Fig. 9(b) 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 proposed LZC-based arbiter and multiplexer is the most area efficient solution requiring roughly 30% less area on average. On the contrary, the fastest design, that is the proposed merged arbiter and multiplexer, although it behaves similarly to the baseline designs for small port widths, it requires significantly more area when the bits per port are increased to 16 and 32 bits. This behavior of the proposed solutions 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 (comparable or smaller than those of the baseline solutions).

A. *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. 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 F between the two extremes: $F = 1$ that corresponds to no slicing and $F = 32$ that corresponds to full slicing, where each bit has each own arbiter. In the general case, bit slicing by a factor F means that we implement F multiplexer and arbiter pairs with each multiplexer carrying $32/F$ bits. The obtained results that clearly depict an optimum slicing factor for each design are shown in Fig. 10. Observe that again, the fastest design lies between the two proposed ones.

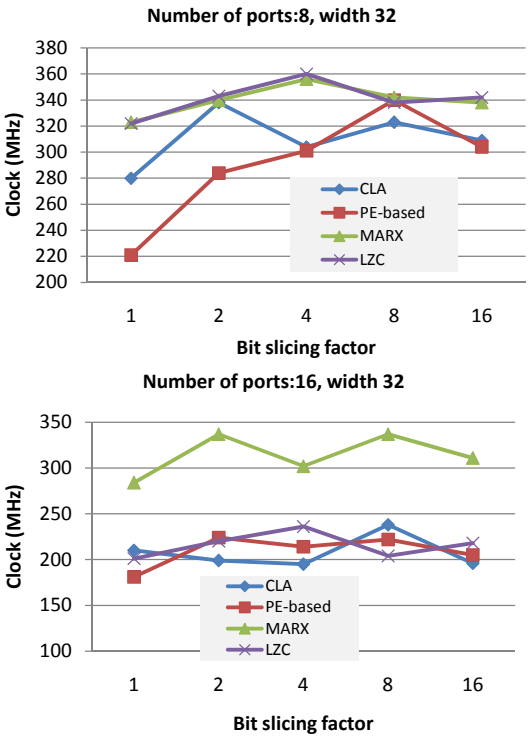

Figure 10. The delay of all 8-input and 16-input circuits for all power-of-two bit slicing factors.

V. Conclusions

In this paper we presented and compared two new solutions regarding the design of an arbiter and a multiplexer when mapped to FPGA logic. The presented designs are faster than the most efficient previous solutions and provide truly scalable solutions for the implementation of wide multiplexers. Although the mapping of multiplexers in LUT logic has received a lot of attention in the 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.

Acknowledgements

The authors would like to thank Xilinx University Program (XUP) for the kind donation of the Xilinx EDA tools.

References

[1] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, pp. 15–31, Sep./Oct. 2007.

[2] Altera, "Applying the benefits of Network on a Chip architecture to FPGA system design," Tech. Rep., January 2011.

[3] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. D. Micheli, and L. Benini, "Bringing NoCs to 65 nm," *IEEE Micro*, vol. 27, no. 5, pp. 75–85, 2007.

[4] T. Mak, P. Sedcole, P. Cheung, and W.Luk, "On-FPGA communication architectures and design factors," in *Proc. of Field Programmable Logic and Applications*, Aug. 2006, pp. 1–8.

[5] J. Flich and D. Bertozzi, *Networks-on-Chip in the Nanoscale Era*. CRC press, 2010.

[6] G. Brebner and D. Levi, "Networking on chip with platform FPGAs," in *Proc. of the IEEE International Conference on Field-Programmable Technology*, Dec. 2003, pp. 13–20.

[7] S. Young, P. Alfke, C. Fewer, S. McMillan, B. Blodget, and D. Levi, "A high i/o reconfigurable crossbar switch," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2003, pp. 3–10.

[8] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, "Packet-switched vs. time-multiplexed FPGA overlay networks," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2006, pp. 205–216.

[9] S. Vassiliadis and I. Sourdis, "FLUX interconnection networks on demand," *Journal of Systems Architecture*, vol. 53, no. 10, pp. 777–793, October 2007.

[10] M. Shelburne, C. Patterson, P. Athanas, M. Jones, B. Martin, and R. Fong, "MetaWire: Using FPGA configuration circuitry to emulate a Network-on-Chip," in *Proceedings of the 2008 International Conference on Field Programmable Logic and Applications*, September 2008.

[11] P. Metzgen and D. Nancekievill, "Multiplexer restructuring for FPGA implementation cost reduction," in *Proceedings of the 42nd Design Automation Conference*, 2005, pp. 421–426.

[12] N. K. Bhatti and N. Shingal, "LUT based multiplexers," *US Patent*, no. 7486110, Feb. 2009.

[13] K. Chapman, "Multiplexer selection," in *White Paper 274: Xilinx FPGA Families*, February 2008.

[14] P. Jamieson and J. Rose, "Mapping multiplexers onto hard multipliers in FPGAs," in *Proceedings of IEEE NewCAS conference*, June 2005, pp. 323–326.

[15] P. Gupta and N. McKeown, "Design and implementation of a fast crossbar scheduler," *IEEE Micro*, pp. 20–28, Jan. 1999.

[16] G. Dimitrakopoulos, N. Chrysos, and C. Galanopoulos, "Fast arbiters for on-chip network switches," in *IEEE International Conference on Computer Design (ICCD)*, 2008, pp. 664–670.

[17] G. Dimitrakopoulos, K. Galanopoulos, C. Mavrokefalidis, and D. Nikolos, "Low-power leading-zero counting and anticipation logic for high-speed floating point units," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 7, pp. 837–850, July 2008.