Fast Arbiters for On-Chip Network Switches

Giorgos Dimitrakopoulos and Nikos Chrysos
Institute of Computer Science (ICS)
Foundation for Research and Technology - Hellas (FORTH),
FORTH-ICS, 100 Plastira Ave, Heraklion, Crete GR-70-013 Greece
- Member of HiPEAC -

Kostas Galanopoulos
Computer Engineering and Informatics Dept.
University of Patras, Patras, GR 26500, Greece

***Abstract*— The need for efficient implementation of simple crossbar schedulers has increased in the recent years due to the advent of on-chip interconnection networks that require low latency message delivery. The core function of any crossbar scheduler is arbitration that resolves conflicting requests for the same output. Since, the delay of the arbiters directly determine the operation speed of the scheduler, the design of faster arbiters is of paramount importance. In this paper, we present a new bit-level algorithm and new circuit techniques for the design of programmable priority arbiters that offer significantly more efficient implementations compared to already-known solutions. From the experimental results it is derived that the proposed circuits are more than 15% faster than the most efficient previous implementations, which under equal delay comparisons, translates to 40% less energy.**

I. Introduction

Arbitration is needed in any case that multiple contenders request access to a shared resource. This scenario appears in many forms in almost every computer system. The most common cases, where conflicting requests are resolved by arbiters, are the widely used bus-based systems where multiple masters and slave modules compete for gaining exclusive access to the bus, and the memory systems, where the small number of supported memory ports do not suffice to serve the read or write requests. Additionally, arbitration is the core function of network switching fabrics where packet flows arriving from different inputs need to be directed to the appropriate output.

On-chip networks are widely used in many complex systems such as general-purpose chip multiprocessors and large systems-on-chip in the embedded systems domain. When the number of cores used is small, a centralized switching fabric is preferred [1], [2]. Due to scalability and technology issues in systems with a large number of cores direct network topologies, such a 2D mesh, are preferred [3], [4], [5]. In these cases, the switches at each node of the network support a smaller number of I/O ports, while a packet needs to travel along many hops before reaching its final destination.

In both cases, either using a centralized switch or a direct network, the switches employed follow almost the same architecture as the one shown in Fig. 1. Incoming packets

This work was supported by the European Commission in the context of the SARC integrated project 27648 (FP6), and the HIPEAC network of excellence (project 004408 and 217068) under the Interconnects research cluster. E-mail: dimitrak@ics.forth.gr

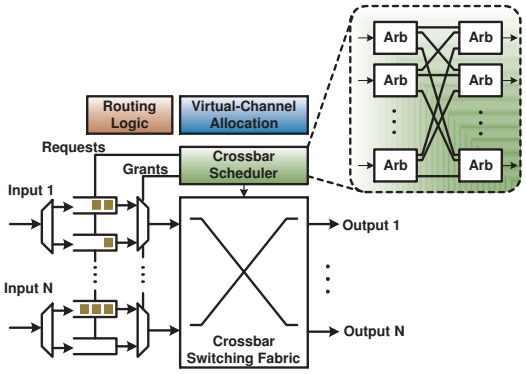

Fig. 1. The abstract architecture of a packet switching fabric.

are stored in input buffers and possibly in output buffers (not show in Fig. 1) after crossing the crossbar. Which inputs are allowed to send their data over the crossbar are determined by the crossbar scheduler (or switch allocator) that resolves all conflicting requests for the same outputs. In many cases, in order to allow the sharing of the network's channels, to differentiate between separate traffic classes, i.e., request/reply packets, and to offer deadlock-free adaptive routing, virtual-channels (VCs - or virtual lanes) are used [6]. The VC allocator similar to the crossbar scheduler is responsible for distributing the outputs' VCs to the requesting inputs. The complexity of the VC allocator increases with the number of available VCs per output link and the versatility of the routing logic [7]. The time needed to complete, either switch or VC allocation is critical to the performance of the switch and it determines the critical path of the design [7], [5], [8].

A. Crossbar scheduler

In the following we will briefly describe the organization of the crossbar scheduler. Practically, the VC allocator follows exactly the same structure. The only difference is that the number of input and outputs of the circuit are multiplied by the number of supported VCs.

The crossbar scheduler is responsible for determining in each cycle the connections between the input and the output ports of the switch. The configuration must meet certain constraints: an input can be connected to at most one output for unicast traffic (or several outputs for multicast traffic), and an output can be connected to at most one input for either unicast or multicast traffic.

The scheduler accepts the requests from each input and

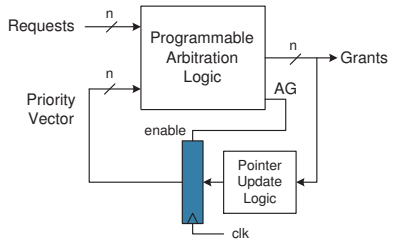

Fig. 2. The block diagram of a programmable-priority arbiter.

decides which one to grant so as to produce a valid connection pattern for the crossbar. Each input holds packets (or flits in case of a wormhole switching) for different outputs. Depending on the organization of the input queues and the packets' destinations, each input can request one or more outputs every cycle. If just a single FIFO queues exists per input, only one request per input is eligible. In that case the crossbar scheduler is constructed using a single arbiter per output of the switch, which decides independently which input to serve.

However, in case that the input buffers are organized in multiple-independent queues forming virtual channels, the inputs can send multiple request per clock cycle. In that case, the scheduling operation is organized in two phases where output arbitration is followed by an input arbitration phase, in order to resolve the grants that came from different outputs to the same input. In many cases, for locality reasons input arbitration is performed first where each input independently decides which of its request to nominate to the output arbiters. In either case, the delay of the scheduling operation equals the delay of 2 arbitrations plus any overhead imposed by the wires connecting the input and output arbiters. A rough diagram of the organization of the $2n$ arbiters of the crossbar scheduler is shown also in Fig. 1.

The core function of the scheduling operation is performed by the arbiter which grants only one of the incoming requests, serving first the request with the highest priority. To allow a fair allocation of resources and to achieve high performance switch operation, we should be able to change the priority of the arbiters. The way the priority changes is part of the policy employed by the crossbar scheduling algorithm. For example, the round-robin policy which is one of the most widely used priority update schemes, dictates that the request served in the current cycle gets the lowest priority for the next arbitration cycle.

Hence, efficient schedulers are built using programmable priority arbiters (PPAs). The block diagram of a general PPA is shown in Fig. 2. It consists of two parts; the programmable arbitration logic (PAL) 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. PAL scans the input requests in a cyclic manner beginning from the position that has the highest priority. For example, if the ith request has the highest priority then the priority is diminishing in a cyclic manner to positions $i+1, i+2, \ldots, n-1, 0, 1, \ldots i-1$, giving to the request $i-1$ the lowest priority to win a grant. The focus of this paper is the design of PAL units that clearly determine the energy-delay behavior of the PPA and the whole scheduler. In the most common cases, like the round-robin arbiter, the design of the pointer update logic is trivial and it consists of just re-arranging the grant signal wires.

Another form of centralized scheduler is the wavefront arbiter [9][1] which receives all input requests and decides which one to grant using a single circuit instead of the distributed input and output arbiters. The main drawback of the wavefront arbiter is that its delay grows linearly with the number of requests. Also the cyclic combinational paths that are inherent to its structure, cannot be handled by commercial static timing analysis tools.

B. Paper Organization

In the following, we briefly review the most efficient approaches used today for the design of PALs and comment on their characteristics. Next, in Section III, we present the proposed arbitration circuits that are based on a new circuit design methodology. The proposed solutions are highly regular, borrowed from the parallel prefix structures employed for their design and offer significant delay savings that are more than 15% when compared to the most efficient previous implementation. Under equal delay comparisons, the proposed circuits can be designed either to support $1.6\times$ the input requests compared to previous implementations, or to save more than 40% of energy consumption for the same number of inputs. To substantiate these arguments detailed experimental results are given in Section IV, while conclusions are drawn in the last section.

II. Programmable Arbitration Logic: State-of-the-art

The simplest form of arbitration can be achieved by the fixed priority arbiter (FPA) (also called priority encoder). The n-bit FPA takes n requests R_i, where the least significant line (position 0) has the highest priority and the most significant one (position $n-1$) the lowest. The FPA produces n grant signals G_i and an additional flag AG (Any Grant) that denotes if at least one input request was granted. A grant is given to the ith bit position when $R_i = 1$ and no other requests exist to any of the rest less significant bit positions.

The FPA can be implemented in many ways. We are interested only in high-speed implementations, where all grant signals are computed in parallel for each bit position. In this case, the grant signal G_i is computed via the well-known priority encoding relation [10] $G_i = R_i \cdot \overline{R}_{i-1} \cdot \ldots \overline{R}_1 \cdot \overline{R}_0$, where $\cdot$ represents the boolean-AND operation and $\overline{R}_i$ denotes the complement of R_i. Assuming 2-input gates, the output of the FPA is computed in the best case in $\log_2 n + 1$ logic stages, using regular parallel prefix structures. Custom solutions also exist that provide high-speed impelementation [11]. When using AND gates with more inputs it is possible to get faster circuits depending on the ratio of the available input capacitance and output loading capacitance [10].

[1]Please, note that the wavefront arbiter, although called an arbiter, is a complete scheduler. In this paper, the term arbiter refers to a circuit that accepts many requests and grants only one them.

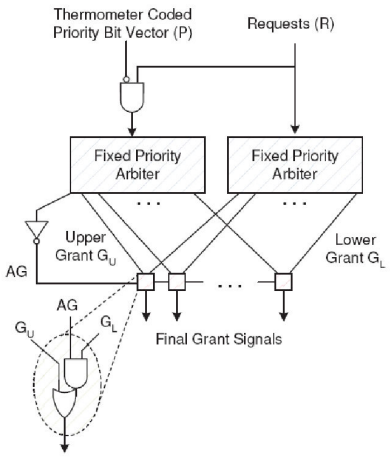

Fig. 3. The dual-path arbiter proposed in [12].

The FPA is the basic block for the design of the most efficient up-to-now programmable arbitration logic, which is the dual-path arbiter [12] and employs two FPAs in parallel. In order to work properly the priority signal P is thermometer coded; in the case that the ith position has the highest priority then the $i+1$ less significant bits of P are equal to 1 while the rest $n-i-1$ most significant bits are equal to 0. The block diagram of the dual-path arbiter is shown in Fig. 3. The upper FPA is used to search for a winning request from the position denoted by the priority vector P, i.e., P_{pos}, up to position $n-1$. It does not cycle back to input 0, even if it could not find a request among the inputs $P_{pos} \ldots n-1$. In order not to allow the upper FPA to search in positions $0 \ldots P_{pos}-1$ its requests are masked with the thermometer coded priority vector P. The lower FPA is driven by the original request lines and searches for a winning request among all positions assuming that position 0 has the highest priority.

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 $P_{pos} \ldots n-1$, the correct output is the same as the output of a lower FPA. If there is a request in the range $P_{pos} \ldots n-1$, then the correct output is given by the output of the upper FPA. Differentiating between the two cases is performed by using the AG signal of the upper FPA. No multiplexers are needed to select between the two computed grant vectors and the output is derived by using a simpler AND-OR gate, as shown in Fig. 3. The dual-path arbiter requires 2 more stages of logic compared to the FPA, i.e., $\log_2 n+3$ in total, while the AG line of the upper FPA needs to drive n logic gates in order to perform the selection procedure.

Several other methods have appeared for the design of PAL units. All proposals are specific cases of the designs analyzed in [12] with only minor modifications. In [13], a hierarchical structure is described that uses multiple smaller FPAs to build a larger arbiter. Also, another class of designs rely on carry-lookahead like structures that try to transfer the priority to the lower-priority positions when high-priority positions do not have an active request [1], [12], [14], [15]. The main drawback of these approaches is that when the priority needs to be transferred from position $n-1$ back to position 0 a cyclic combinational path appears to the circuit that increases the delay of the design and causes testability problems. One simple, but inefficient approach, to break the loop is to add two carry-lookahead structures where the carry out of the first adder feeds the carry-in of the second [15]. Practically, all these techniques do not offer any delay benefits compared to the dual-path arbiter.

The proposed method, presented in the following section, is also derived from the carry-lookahead formulation of the programmable arbitration logic. Nevertheless, the introduced methodology leads to completely new circuits that do not have the aforementioned limitations and offer significantly faster implementations compared to the dual-path arbiter.

III. New Programmable Arbitration Logic

In our case, we eliminate the need for multiple copies of the FPA and we design a new circuit that handles in a unified manner the cyclic nature of the diminishing priority transfer signals. In our case, the position with the highest priority is declared using priority vector P which is encoded in one-hot form. For example, if position 1 has the highest priority in the case of an 8-input PPA the P vector equals 00000010.

As in the case of FPA a request is granted when no other higher priority input has an active request. Therefore, at each bit position a new signal is required that indicates whether a grant was given to a highest priority request. This priority transfer signal is denoted as C_i and it is computed at each bit position based on the value of the local request signal R_i, the priority signal P_i , and the corresponding priority transfer signal of the higher priority position C_{i-1}. When $C_i = 1$ it means that the next position $i+1$ can produce a winning grant assuming that R_{i+1} is asserted, since all previous request lines are equal to zero. In the opposite case, when $C_i = 0$, no other winning grant should be produced in the positions with index larger than i (taken in a cyclic manner). Assuming that $R_i = 1$ the ith input can give a grant when either P_i or the incoming priority transfer signal C_{i-1} is asserted. In that case, the grant signal G_i is set equal to 1 and the priority transfer signal out of the ith bit position C_i is set equal to zero. The above conditions for the assertion of the grant and the priority transfer signals have been described in [12] and [15] and can be written as follows (+ denotes the boolean-OR operation):

$$G_i = R_i \cdot (P_i + C_{i-1}) \quad (1)$$

$$C_i = \overline{R}_i \cdot (P_i + C_{i-1}) \quad (2)$$

The priority transfer signal C_{n-1} should be fed back to position 0, i.e., $C_{n-1} = C_{-1}$ in order to guarantee the cyclic transfer of the diminishing priority. This property is the cause of the cyclic combinational path, which was discussed at the end of Section II.

Departing from (1) and (2), we express the priority transfer operation differently. The priority transfer signal out of the ith position is represented by X_i that is equal to

$$X_i = P_i + C_{i-1} \quad (3)$$

Using X_i we are able to combine in one signal the two sources of priority transfer. The priority either comes from the previous bit positions via C_{i-1} or is set by signal P_i. For a systematic design, we need to derive a recursive formula that connects X_i and X_{i-1}. To achieve this goal we express at first the bit C_i as a function of X_i. Following (2) and (3) we can write C_i as $\overline{R}_i \cdot X_i$. Replacing in (3) the new definition derived for C_{i-1} we get that

$$X_i = P_i + \overline{R}_{i-1} \cdot X_{i-1} \tag{4}$$

Hence, the C terms have disappeared and the new value of X_i depends only on the priority P_i, the request R_{i-1} and the unified incoming priority transfer signal X_{i-1}. Using X_i the grant signal at the ith bit position can be computed as $G_i = R_i \cdot X_i$.

The recursive definition of X_i in has exactly the same form as the well known carry lookahead equation $c_i = g_i + p_i \cdot c_{i-1}$ where in place of the carry generate bit g_i we have the priority signal P_i (called priority generate) and instead of the carry propagate bit p_i we use the inverted request signal $\overline{R}_{i-1}$ (called priority propagate) For example, by unrolling Eq. (4) the priority transfer X_2 in the case of a 4-input PAL can be computed using the priority generate and propagate bits as follows:

$$X_2 = P_2 + \overline{R}_1 \cdot P_1 + \overline{R}_1 \cdot \overline{R}_0 \cdot P_0 + \overline{R}_1 \cdot \overline{R}_0 \cdot \overline{R}_{-1} \cdot X_{in}$$

$X_{in} = X_{-1}$ denotes the incoming priority transfer similar to the carry-in signal of an adder and due to the cyclic transfer of the priority of the arbiter $R_{-1} = R_{n-1}$ and $X_{-1} = X_{n-1}$.

Following adder design principles, we set at first $g_i = P_i$ and $p_i = \overline{R}_{i-1}$ while $p_0 = \overline{R}_{n-1}$. Then, we can define groups of priority generate and priority propagate. In the general case each priority generate group starting at position j and ending at position i, with $i \geq j$ is defined as

$$GP_{i:j} = g_i + \sum_{k=j}^{i-1} (GR_{i:k+1} \cdot g_k)$$

where $GR_{i:j}$ denotes the group propagate term in the range $i \ldots j$ of positions that is defined as follows:

$$GR_{i:j} = \prod_{k=j}^{i} p_k$$

For the degenerated case $GP_{i:i} = g_i$ and $GR_{i:i} = p_i$. Please, note that the AG signal is equal to $\overline{GR}_{n-1:0}$ indicating that at least one request is active.

Using groups of priority propagate and generate signals we can express the priority transfer signal X_i as follows:

$$X_i = GP_{i:0} + GR_{i:0} \cdot X_{in} \tag{5}$$

The group priority generate term $GP_{i:0}$ covers the case that the priority is generated for the ith position after having searched in all less significant positions down to position 0. For the case of the most significant bit position $n-1$ the corresponding group generate term $GP_{n-1:0}$ searches all the input requests from input 0 to input $n-1$. Therefore, in all cases $GP_{n-1:0}$ is equal to the desired priority transfer signal

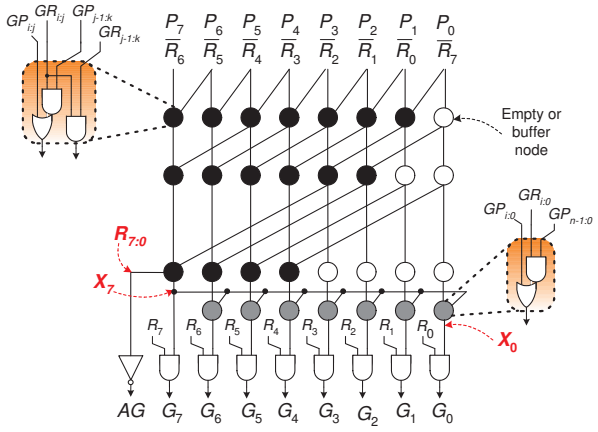

Fig. 4. The Proposed-I arbiter following the end-around-carry approach.

for position $n-1$ that is X_{n-1}. Following this observation and the fact that $X_{n-1} = X_{in}$ the equation describing the computation of the bit X_i can be transformed as follows:

$$X_i = GP_{i:0} + GR_{i:0} \cdot GP_{n-1:0} \tag{6}$$

Thus the ith position has the highest priority because either the priority was generated in the range $0 \ldots i$ or it is coming from a more significant position as declared by $GP_{n-1:0}$ and has been propagated to i via the propagate term $GR_{i:0}$. In fact in the case that the priority was transferred in a circular manner to the ith position only the range $n-1 \ldots i+1$ needs to be examined.

By definition $GP_{n-1:0}$ can be derived by any smaller group generate and propagate terms. Thus we can analyze $GP_{n-1:0}$ as $GP_{n-1:i+1} + GR_{n-1:i+1} \cdot GP_{i:0}$. Substituting the new expression to (6) we get that

$$X_i = GP_{i:0} + GR_{i:0} \cdot (GP_{n-1:i+1} + GR_{n-1:i+1} \cdot GP_{i:0})$$

Eliminating the redundant term $GR_{i:0} \cdot GR_{n-1:i+1} \cdot GP_{i:0}$, the new relation can be written as:

$$X_i = GP_{i:0} + GR_{i:0} \cdot GP_{n-1:i+1} \tag{7}$$

In this way, the redundant examination of the requests in the range $i \ldots 0$ has been removed, and the circular operation of the priority transfer has been embedded inside each relation. In order to better understand the derived relation we will write the new equations describing the bits of X in the case of a 4-input arbiter.

$$X_3 = P_3 + \overline{R}_2 \cdot P_2 + \overline{R}_2 \cdot \overline{R}_1 \cdot P_1 + \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0 \cdot P_0$$
$$X_2 = P_2 + \overline{R}_1 \cdot P_1 + \overline{R}_1 \cdot \overline{R}_0 \cdot P_0 + \overline{R}_1 \cdot \overline{R}_0 \cdot \overline{R}_3 \cdot P_3$$
$$X_1 = P_1 + \overline{R}_0 \cdot P_0 + \overline{R}_0 \cdot \overline{R}_3 \cdot P_3 + \overline{R}_0 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot P_2$$
$$X_0 = P_0 + \overline{R}_3 \cdot P_3 + \overline{R}_3 \cdot \overline{R}_2 \cdot P_2 + \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot P_1$$

A. Circuit Organization

Equations (6) and (7) allow the derivation of two new circuits for the design programmable priority arbitration logic that are described in the following paragraphs.

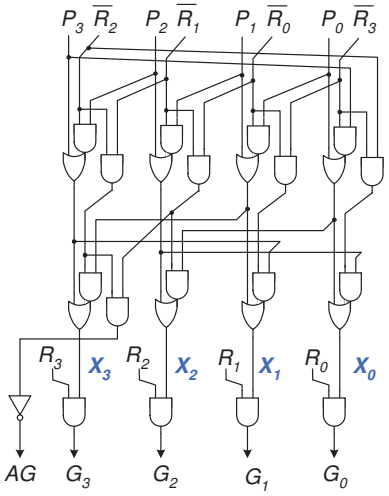

Fig. 5. Fast 4-bit arbiter.

1) Proposed I: Eq. (6) can be computed in a very elegant way using only a single circuit, instead of the two FPAs used in the case of the dual-path arbiter. Using traditional CLA techniques we can compute in parallel the terms $GP_{i:0}$ and $GR_{i:0}$ for each position from 0 up to $n-1$ and compute the final value of the priority transfer signal X_i using an additional AND-OR gate, as shown in Fig. 4. In this way there is no cyclic operation and the most significant output $GP_{n-1:0}$ is just given to n AND-OR gates in order to derive the final bits of X. This technique resembles the carry-increment stage used in end-around-carry adders [16]. At the final stage, the bits of X are ANDed with the request lines in order to produce the final grant signals.

Any form of carry-computation unit can be used for the computation of $GP_{i:0}$. Instead, for fast implementations we prefer the powerful parallel prefix carry computation units. Parallel prefix circuits are designed using the associative operator $\circ$ that combines groups of bits and was defined in [17] as $(t,z)\circ(t',z') = (t+z\cdot t', z\cdot z')$. Thus to compute the priority transfer for ith position we must associate via the $\circ$ operator all the terms $(P_i,\overline{R}_{i-1})$ from position 0 up to position i. For example $(GP_{2:0}, GR_{2:0})$ is computed as follows:

$$(GP_{2:0}, GR_{2:0}) = (P_2,\overline{R}_1)\circ(P_1,\overline{R}_0)\circ(P_0,\overline{R}_3)$$

The $\circ$ operator accepts two pairs of inputs and produces two outputs, i.e., a new group generate term and a new group propagate term. The logic-level implementation of the $\circ$ operator that is graphically represented as a 2-input-1-output $\bullet$ cell is also shown in Fig. 4. The grey circles in Fig. 4 represent a simplified form of the operator where no new group propagate term is produced.

The Proposed-I arbiter although solving the arbitration problem in a unified and concise manner, it stills has some serious drawbacks. It requires an additional logic level compared to the FPA and has a large fanout line that negatively affects the delay of the circuit. These shortcomings are alleviated by the Proposed II solution that is derived using the simplified equation (7).

2) Proposed II: Following (7), and by properly grouping the input signals $(P_i,\overline{R}_{i-1})$, according to the guidelines presented in [18] for the case of 1's complement adders, we can derive very fast arbiters. We will present the Proposed-II solution via an example. Using the new formulation all priority transfer bits of a 4-input arbiter (shown in Fig. 5) can be computed using the $\circ$ operator as follows:

$$X_3 \leftrightarrow (P_3,\overline{R}_2)\circ(P_2,\overline{R}_1)\circ(P_1,\overline{R}_0)\circ(P_0,\overline{R}_3)$$
$$X_2 \leftrightarrow (P_2,\overline{R}_1)\circ(P_1,\overline{R}_0)\circ(P_0,\overline{R}_3)\circ(P_3,\overline{R}_2)$$
$$X_1 \leftrightarrow (P_1,\overline{R}_0)\circ(P_0,\overline{R}_3)\circ(P_3,\overline{R}_2)\circ(P_2,\overline{R}_1)$$
$$X_0 \leftrightarrow (P_0,\overline{R}_3)\circ(P_3,\overline{R}_2)\circ(P_2,\overline{R}_1)\circ(P_1,\overline{R}_0)$$

In the first level of the parallel prefix tree all terms of the form $(GP_{i:i-1}, GR_{i:i-1}) = (P_i,\overline{R}_{i-1})\circ(P_{i-1},\overline{R}_{i-2})$ are computed. For the case, of $(GP_{0:-1}, GR_{0:-1})$ the pairs $(P_0,\overline{R}_3)$ and $(P_3,\overline{R}_2)$ are used in order to satisfy the cyclic nature of the priority transfer. In the next level these terms are combined to produce double size terms. For example in the case of X_1 we need to combine the terms $(GP_{1:0}, GR_{1:0})$ and $(GP_{3:2}, GR_{3:2})$ produced in the first level of the tree in order to produce the final relation. This combination involves lines that run from the more significant positions back to less significant positions. This property is better shown in the 8-input arbiter depicted in Fig. 6.

In the general case, for the construction of a n-bit arbiter we need $n\log_2 n$ $\bullet$ operators placed on the $\log_2 n$ levels of the parallel prefix tree. The last level is composed of simpler grey cells. Every operator placed on the ith row (prefix level) and the jth column (bit position) is connected to the 2 operators of row $i-1$ that are placed on columns j and $(j-2^i) \mod n$, respectively. In the first level, the operators are driven directly from the priority signals and the inverted request lines. When n is not a power-of-two simpler structures an be derived following the methodology presented in [19].

Using this technique the grant vector is computed in exactly $\log_2 n+1$ logic levels, as in ordinary FPA. Also, no large fanout line is required, since the cyclic nature of the priority transfer is performed inside the prefix tree. The only drawback of the proposed circuits are the long lines inside the priority transfer computation unit that increase its layout area. Although the extra capacitance added by these lines degrades by a small percentage the delay of the circuit, the overall circuit is faster than the most efficient previous implementation, i.e., the dual-path arbiter.

IV. Results

The proposed circuits have been evaluated using static CMOS implementations in the UMC 130nm standard performance CMOS technology [20]. The delay measurements for all examined designs are reported in fanout-of-4 inverter delays (FO4). The FO4 delay metric equals to the delay of an inverter that drives four equally-sized inverters, and it is used since it provides in some sense a technology independent way to express the delay of a circuit [10]. In our case 1FO4 $\sim$ 62ps, under typical process conditions, nominal V_t and $V_{dd} = 1.2V$.

In order to explore the energy-delay space for each design, we performed gate sizing for several delay targets, beginning

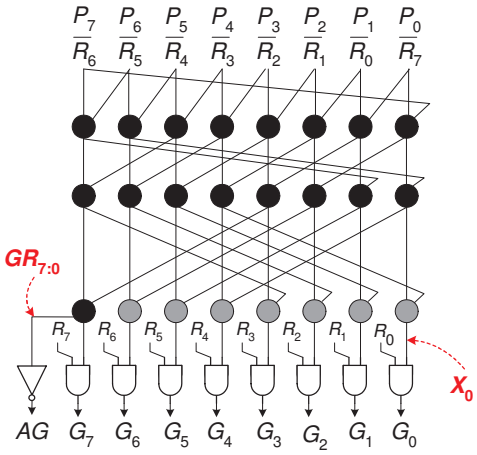

Fig. 6. The 8-bit arbiter designed according to Proposed-II methodology.

from the circuit's minimum achievable delay. Circuit sizing is performed using geometric-programming-based optimization following the guidelines presented in [21]. For the formulation of the geometric program, the delay and the energy of each gate are expressed in mathematical form as a function of the gate's size (transistor widths) and the slope of the input signals. The delay and energy models have been calibrated using HSpice simulations and appropriate data fitting. For multi-input gates the input-to-output delay is modeled separately for each input. Geometric programming leads to convex optimization problems that were solved using the solver of [22]. For the derived gate sizes, the energy and the delay of each circuit have been measured back in HSpice. For the energy measurements, we assumed random inputs.

During optimization and measurements, the maximum allowable input capacitance equals 25fF, while the circuits under comparison drive a load of 100fF. Also interstage wiring loads, both capacitance and resistance, have also been taken into account. The RC contribution of each wire has been estimated according to its length. In datapath functions, as the proposed ones, every two stages of logic are connected both with short wires running vertical (from inputs to outputs direction) and with longer wires that are placed orthogonal to the rest data signals. The length of these wires is then a function of how many bit positions they have to cross in order to reach the connecting node. Hence, the length of each wire inside the datapath is the product of the bit pitch and the number of bit positions it has to cross. For the circuits under evaluation, we assume a bit pitch of 12 metal-1 tracks, as the one used in state-of-the-art microprocessors [23].

We compared the proposed arbiters with the most efficient arbiter that is the dual-path architecture. In order to get a high-speed implementation for the dual-path arbiter we implemented the FPAs using a Kogge-Stone parallel prefix topology [24] of AND gates. Other topologies have also been tested but lead to slower circuits.

At first, we present our results for the case of small arbiter circuits with input length of 8 bits. The corresponding energy-delay curves are shown in Fig. 7. It can be observed that the proposed methodology (Proposed II) leads to faster circuits and their minimum achievable delay is smaller than

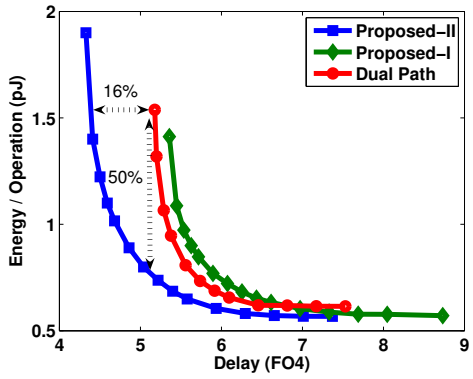

Fig. 7. The energy-delay curves of the 8-input arbiters under comparison.

the corresponding delay of the dual-path PAL by more than 16%. From the delay results reported and extra experiments performed, we conclude that a complete 8-input scheduler can be built with a delay well under 10 FO4 (2 arbiter in series plus wiring overhead).

For equal delay measurements the proposed circuits offer more than 50% of energy savings[2]. The reduced energy per arbitration of the proposed structures partially comes from the one-hot encoding of the priority vector. When the priority changes, and assuming a constant request vector, only 2 positions switch their values from 1→0 and from 0→1, respectively. On the contrary in the case of the dual-path arbiter when the priority moves from a more significant position (close to $n-1$) back to a less significant position many gates change their state due to the thermometer encoding of the priority. [3] This feature also positively affects the energy of the register holding the priority vector (see Fig. 2).

Please notice that the energy dissipation of the crossbar scheduler is only a small part of the total power consumption of the switch, so reducing the arbiter's energy does not improve much the overall power efficiency. However, this extra energy benefit offered by the proposed designs can be used for increasing the available input capacitance of the proposed circuits in order to further decrease the delay of the critical path. Allowing such an optimization leads to circuits that require the same energy as their dual path counterparts and are faster by more than 20%. Increasing the available input capacitance increases also the delay of the driving gates of the proposed circuit by a small percent. This overhead has been taken into account into the delay savings reported.

In Fig. 7, we included also the energy-delay behavior of the Proposed I solution. This structure does not offer any benefit compared to the dual-path arbiter, except from slightly reduced energy for delays larger than 7 FO4. This behavior is expected since Proposed I has almost the same characteristics as Dual-Path arbiter, i.e., high fanout line and extra logic stages.

For even smaller arbiters servicing between 4 to 6 requests, as those required in on-chip network switches implementing

[2]Please notice than the energy reported refers to the energy of the programmable arbitration logic of Fig. 2 and does not include the overhead of registers and the pointer update logic.

[3]The dual-path arbiter needs the thermometer encoded priority vector in order to function correctly and it cannot be changed to an one-hot code.

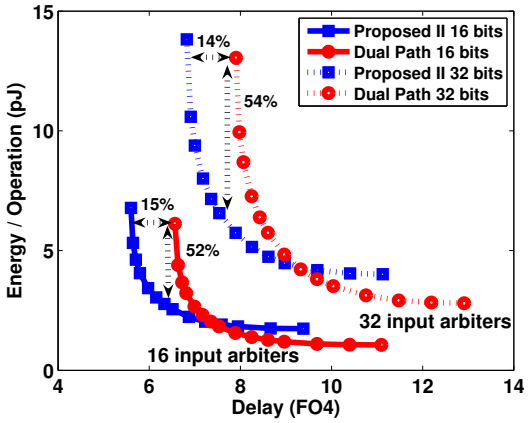

Fig. 8. The energy-delay curves for the most efficient 16 and 32-input arbiters.

low-dimension networks, the proposed arbiters offer significantly faster implementations. Under the same optimization conditions, we report that the minimum achievable delay of the proposed 4-input arbiter shown in Fig. 5, which follows the Proposed II design method, is roughly equal to 3.2 FO4 which is 20% less than the 4 FO4 of the dual-path circuit. In such small designs, the main cause of the delay reduction is the the reduced number of logic levels of the new circuits, while the reduced fanout plays a supplementary role.

Finally, for completeness, we repeated the same experiments for larger arbiter designs that are used in high-end switches and routers with a large number of input ports. In the case of high-radix switches, again the design of efficient arbiter circuits is critical for the overall switch performance [25]. Fig. 8 shows the energy-delay curves obtained for the case of 16 and 32-input arbiters. In both cases, the proposed arbiters are faster in average by 14% while they require significantly less energy per arbitration. For larger delays, where the dual-path arbiter appears more energy-efficient, we can instead use the Proposed-I designs and further reduce energy. Also, note that the delay of the Proposed-II solution in case of 16 inputs is very close to the delay of an 8 input dual-path arbiter. After performing additional experiments, we found that the proposed circuits can support roughly 1.6x the number of input requests of the dual-path arbiter without incurring any delay overhead.

V. Conclusions

A new bit-level algorithm for the design of programmable-priority arbiters has been presented in this paper, along with an efficient circuit implementation. The new parallel-prefix arbiters compute the winning grants using only $\log_2 n + 1$ logic stages and follow a regular structure that makes them amenable to efficient VLSI implementations. From, the experimental results it is derived that the proposed circuits offer significant delay reductions compared to state-of-the-art arbiter architectures. Taking into account that crossbar scheduling and/or VC allocation determine the operation speed of the whole switch we conclude that switch designers can truly benefit by the adoption of the proposed arbiters.

References

[1] V. Yalala, D. Brasili, D. Carlson, A. Hughes, A. Jain, T. Kiszely, K. Kodandapani, A. Varadharajan, and T. Xanthopoulos, "A 16-Core RISC Microprocessor with Network Extensions," in *Proceedings of the International Solid-State Circuits Conference*, Feb. 2006.

[2] U. G. Nawathe, M. Hassan, K. C. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, "Implementation of an 8-Core, 64-Thread, Power-Efficient SPARC Server on a Chip," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 6–20, Jan. 2008.

[3] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, pp. 6–20, Jan. 2008.

[4] 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.

[5] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, pp. 41–50, Sep./Oct. 2007.

[6] D. W. J., "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 2, pp. 194–205, March 1992.

[7] L.-S. Peh and W. J. Dally, "A Delay Model for Router Micro-architectures," *IEEE Micro*, pp. 26–34, Jan./Feb. 2001.

[8] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *31st Annual International Symposium on Computer Architecture (ISCA'04)*, 2004, pp. 188–198.

[9] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 1, pp. 13–27, 1993.

[10] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2005.

[11] C.-H. Huang, J.-S. Wang, and Y.-C. Huang, "Design of high-performance cmos priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *IEEE Journal of Solid-State Circuits*, vol. 37, pp. 63–76, Jan. 2002.

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

[13] E. S. Shin, V. J. M. III, and G. F. Riley, "Round-robin Arbiter Design and Generation," in *Proceedings of the International Symposium on System Synthesis*, Oct. 2002.

[14] T. B. Preuer, M. Zabel, and R. G. Spallek, "About Carries and Tokens: Re-Using Adder Circuits for Arbitration," in *Proceedings of the Signal and Image Processing Symposium*, 2005, pp. 59–64.

[15] B. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan-Kaufman, 2004.

[16] R. Zimmerman, "Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication," in *Proc. of 14th IEEE Symposium Computer Arithmetic*, April 1999, pp. 158–167.

[17] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. on Comp.*, vol. 31, no. 3, pp. 260–264, Mar. 1982.

[18] L. Kalampoukas, D. Nikolos, C. Efstathiou, H. T. Vergos, and J. Kalamatianos, "High-Speed Parallel-Prefix Modulo $2^n - 1$ Adders," *IEEE Trans. on Computers*, vol. 49, no. 7, pp. 673–680, Jul. 2000.

[19] G. Dimitrakopoulos, H. T. Vergos, D. Nikolos, and C. Efstathiou, "A family of parallel prefix modulo $2^n - 1$ adders," in *IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, 2003, pp. 326–336.

[20] *UMC 130nm CMOS FSG process*. Europractice IC Service.

[21] S. P. Boyd, S. J. Kim, D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, no. 6, pp. 899–932, Nov.-Dec. 2005.

[22] A. Mutapcic, K. K., S. Kim, and S. Boyd, "GGPLAB: A MatLab toolbox for geometric programming," 2006.

[23] H.-J. O. et al., "A Fully-Pipelined Single-Precision Floating Point Unit in the Synergistic Processor Element of a CELL Processor," in *Symp. VLSI Circuits Dig. of Technical Papers*, June 2005, pp. 24–27.

[24] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, pp. 786–793, Aug. 1973.

[25] J. Kim and B. Dally, "The microarchitecture of a high-radix router," in *31st Annual International Symposium on Computer Architecture (ISCA'04)*, 2004.