Switch Allocator for Bufferless Network-on-Chip Routers

Giorgos Dimitrakopoulos
Informatics and Communications Engineering
University of West Macedonia
Karamanli & Ligeris, Kozani, Greece
gdimitrak@uowm.gr

Kostas Galanopoulos
Electrical and Computer Engineering
National Technical University of Athens
Zografou campus, Athens, Greece
galanopu@mail.ntua.gr

ABSTRACT

Bufferless switches can be an attractive and energy-efficient design option for on-chip networks when network utilization is low and low-latency operation matters the most. However, this promising design option is limited by the complexity of the control logic required to operate a bufferless switch that imposes large delays and limits the clock frequency. Pipelining is not an option in this low-latency environment. In this paper, we propose a new switch allocator for bufferless switches that parallelizes the steps required for achieving a match between requesting inputs and available outputs and offers significantly faster implementations.

Keywords

bufferless router, switch allocation, deflection routing, logic design

1. INTRODUCTION

Bufferless switches have been proposed as a low-cost (low power) solution for on-chip networks. In this case, when two or more incoming flits compete for the same output only one of them is granted to move to a productive output closer to its final destination. The remaining flits can be either misrouted (deflected) to other outputs [1], [2], or dropped and not-acknowledged to the upstream switch [3], [4]. In this way, buffering the non-granted flits at the inputs of the switch is avoided. Similarly, circuit switching [5] can also remove buffering requirements at the cost of connection setup overhead.

Such bufferless switches target mainly low-latency operation at low network loads. Several researchers have shown that this region of operation matches well the network traffic imposed by several applications [1], [6]. On the other hand, bufferless switches offer limited maximum throughput when compared to high-end buffered switches [7]. The cause for this effect is that the deflection of flits to unproductive outputs spreads the congestion throughout the network. Nevertheless, the savings in buffer and clocking power may alleviate this negative effect under certain circumstances. Also, the power saved from buffers can be re-spent to the network by making the links and the crossbars wider thus gaining back some of the lost bandwidth.

INA-OCMC '11, January 23, 2011, Heraklion, Greece

The ultimate goal for low-latency operation is to allow switch traversal to happen in a single cycle even if high clock frequencies are required. The clock frequency of the switch can be increased by pipelining its operations. However, in the case of bufferless switches, this is not a good option mainly for two reasons: First, pipelining increases the latency seen by a flit when it traverses an almost empty network at low loads and second it increases the negative effect of a possible deflection, since more cycles are needed for a deflected flit to get back on track.

As a first thought bufferless switches are expected to lead to simpler switch designs since they allow the movement of flits even to unproductive directions. However, this goal has not been succeeded yet. One critical parameter that limits the applicability of bufferless switches and restricts their scalability, is the delay complexity of the corresponding switch allocator that decides which flits will leave from their requested output and which ones will be misrouted to the rest outputs. The delay of commonly used switch allocators in the case of bufferless switches is significantly larger than that of buffered switches. In this way, bufferless switches cannot a high clock frequency and the goal for a true single-cycle operation is limited only to slow clock frequencies [7], [8].

In the case of the simpler form of buffered switches the switch allocator is composed of a single arbiter per output. This organization is not enough in the case of bufferless switches since somehow the output port of the deflected flits needs to be determined. At each cycle, every input port may have a new flit for transmission. According to the operation of FLIT-BLESS switch allocator for bufferless switches [1], the priority of a flit to move to a productive output is determined by its weight relative to the weights of the flits from the remaining inputs that want to leave from the same output. The weight of each flit reflects the amount of time it spent in the network. Thus, the requests coming from the oldest flits receive the highest priority, and they are considered first. These flits always gain access to a productive output port that brings them closer to their final destination. By not allowing the oldest flits to be deflected livelock conditions are also prevented.

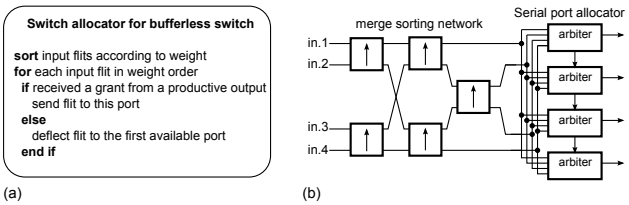

Figure 1: (a) FLIT-BLESS switch allocation for deflection-based bufferless switches and (b) the fastest previous implementation.

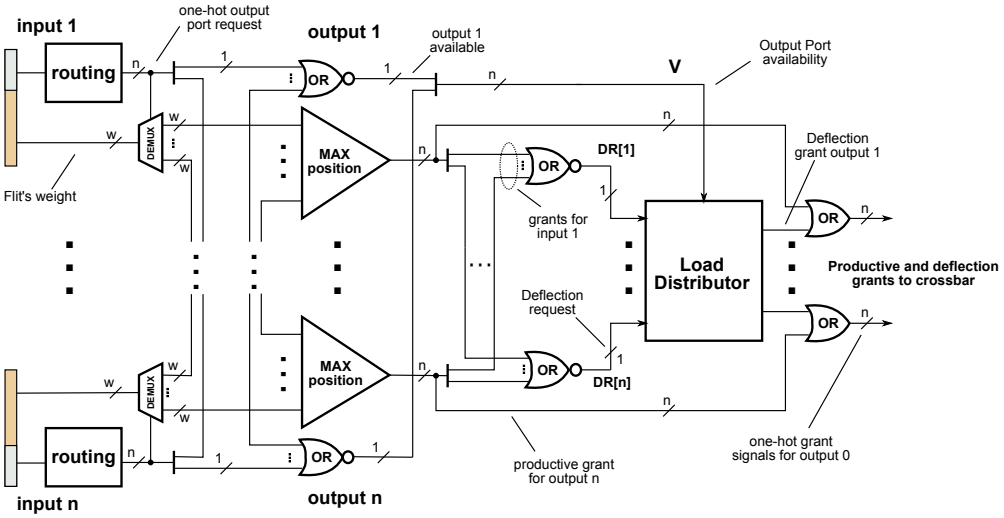

Figure 2: The organization of the proposed switch allocator.

Fig. 1 summarizes the operation of the switch allocator for bufferless switches and also gives a rough block diagram of its implementations according to [1], [7]. The switch allocator is built around serially connected arbiters that inevitably yield a slow critical path. Each arbiter receives one request and tries to match it with the corresponding output if it is available. When an output is matched to an input request, it is considered taken and it cannot be used by the following arbiters. In the case that the received request corresponds to an unavailable output, the arbiter matches the requesting input to the first available output even if this decision sends the flit to an unproductive path. The flits are sorted according to their priority and the ones with the highest priority are served first. In this way, the the number of productive outputs given at each cycle is maximized. Sorting is best implemented in hardware by a fast sorting network [9]. Each node of the sorting network contains a comparator and puts the largest element in the output pointed by the arrow inside each node.

In this paper, we offer a new organization for the switch allocator of bufferless switches. The proposed design implements also the algorithm of Fig. 1(a) but it completely removes the serial dependency among output arbiters that appear in the best so far implementation of Fig. 1(b). Also, at the same time the weight comparison is transformed to a simpler form and the need for a complete sorting network is canceled. In this way, new scalable circuits can be designed that reduce significantly the delay of the switch allocator.

2. THE PROPOSED SWITCH ALLOCATOR

The organization of the proposed switch allocator is shown in Fig. 2. At first, the destination address of each flit is passed to the routing logic that determines which output port leads to a productive path. According to the one-hot index of the output port the weights of the flits are gathered per output port after a simple masking with the output requests and appropriate wire rearrangment. At each output port we may have one or more active requests along with their corresponding weights. The selection of the highest priority flit is performed by selecting the flit that is associated with the largest weight. The MAX operation shown in Fig. 2 is implemented as a binary tree of comparators and is faster than a general sorting network. The speed of determining the maximum weight can be also improved by following the techniques presented in [10].

The flits of the inputs that did not receive any grant after this step should be deflected to any available output. Therefore, we need to identify appropriately which outputs are available and which flits should be deflected. An output is available if it did not receive any active requests from any input. This is known in advance and in parallel to the max-weight selection by checking the requests produced by the routing logic. The flits that should be deflected are the ones that did not receive a grant by the MAX logic at each output. Thus, at each input, the grant signals received from each output are ORed together to identify if there was at least one active grant. If this is not the case, the flit of the corresponding input is eligible for deflection.

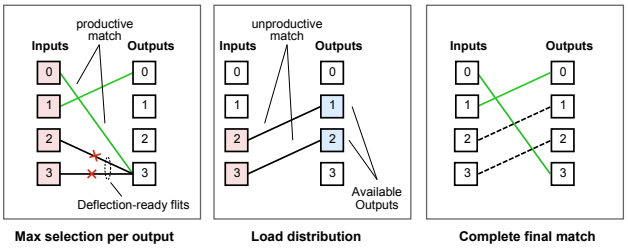

Figure 3: An example of the operation of the proposed switch allocator implementing the algorithm of Fig. 1.

The input that has a flit ready for deflection asserts a deflection request DR. DR signals along with the output-port availability flags V are passed to a new circuit called *load distributor* that matches between the deflection-ready inputs and the available outputs so that no available output remains underutilized. An example of the assignment of 4 inputs to 4 outputs is shown in Fig. 3. In this example, 2 inputs are ready to transmit a deflection cell and 2 outputs are available to receive one. This case may arise from the original set of requests where only input 0 and 1 have won a productive output either because they had the highest priority (the case for input 0) or because they requested an output without any contention (the case for input 1).

By the construction of the proposed switch allocator it is evident that the allocation procedure always maximizes the number of

productive outputs and it does not wrongly deflects any flit that requested an output with no contention. What remains to be clarified is the design of the fast load distributor that spreads deflection flits to the available outputs.

3. LOAD DISTRIBUTOR

At first, we begin with a topological organization of the load distributor, assuming that it is implemented as a box where input deflection requests DR are aligned vertically in the left (west) side of the box, while the availability flags V that denote which output is free are aligned horizontally at the upper (north) side of the box. The deflection request of the first input and the availability flag of the first output are placed on the upper-left (north-west) corner of the distributor. The assignment of requests to available outputs begins from input 0 and output 0, and moves gradually to the remaining inputs and outputs placed to positions with a larger index.

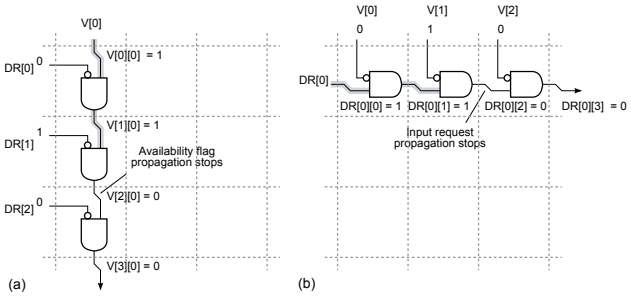

Figure 4: The propagation of (a) the availability flags and (b) the input requests.

At the beginning, as depicted in Fig. 4(a), we assume the availability flag of the first output. According to the operation of the distributor the availability flag is propagated downwards until it finds an active deflection request. When a match is achieved, the availability flag seen by the remaining inputs is set to zero since this output is no longer available and has been selected by another input. Therefore, the availability flag $V[0]$ moving in the NORTH-SOUTH direction is transferred unchanged from row i to $i+1$, *i.e.*, $V[i+1][0] = V[i][0]$, when $DR[i] = 0$. On the contrary, when $DR[i] = 1$, the availability of this output is consumed by input i and the availability seen by the remaining inputs with index larger that i is set equal to 0. Hence, when $DR[i] = 1$, flag $V[i+1][0]$ is set to 0. Summarizing both conditions that connect $V[i][0]$ and $DR[i]$ with $V[i+1][0]$, we get that $V[i+1][0] = V[i][0] \cdot \overline{DR[i]}$. This operation is exactly the same with the operation of the priority encoder. The only difference is that instead propagating the priority for an input, we propagate the output availability flags.

In a similar manner, when we isolate the deflection request of input 0, we need to propagate it to all outputs until we find the first available one. In other words, $DR[0]$ is kept alive and keeps moving in the WEST-EAST direction, until it meets an available output, as shown in Fig. 4(b). When an available output is found the request is set to zero since the corresponding input is not allowed to send more than one flits at each cycle. Therefore, the request that propagates from input i to all outputs (columns) is equal to $DR[0][i+1] = DR[0][i] \cdot \overline{V[i]}$. Again the problem is solved via a priority encoder that now acts on the availability flags and decides on the propagation of the input request.

The solution to the load distribution problem needs at the same time both the per-row and the per-column arbitration steps. The combined operation leads to a regular 2D arbitration array based on the unified arbitration cell as shown in Fig. 5 and is just a combination of the per-row and per-column cells presented earlier. In this way, while the availability flags are propagated vertically until they meet an active request, the requests travel horizontally until they meet an available output. A match between an input/output pair is actually achieved when both the incoming availability flag $DV[i][j]$ and the incoming request $DR[i][j]$ of the ith row and jth column, respectively, are both equal to 1. This condition is given to the output $Grant[i][j]$ of the cell.

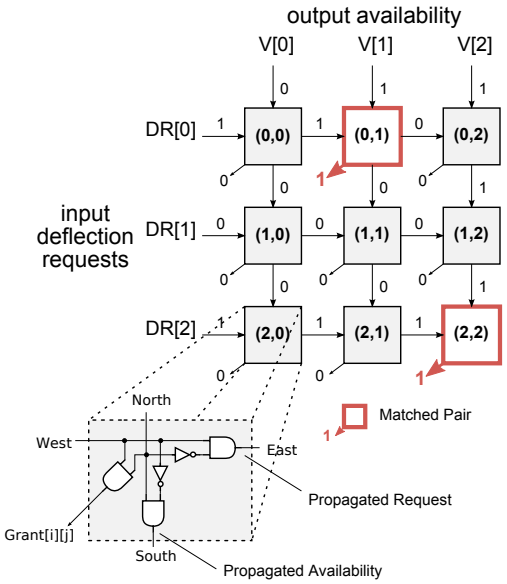

Figure 5: The 2D structure of the basic form of the load distributor.

3.1 Delay optimized load distributor

The basic circuit for the load distributor has a delay equal to the delay of $2n$ AND gates.[1] This delay is imposed by the order of assignment of inputs to outputs that we selected for the circuit of Fig. 5. However, in a deflection operation there is no need to maintain this exact order. The deflection requests can be assigned to any available output in any order. The only constraints that the load distributor should respect are the following: First each deflection request should be checked for a possible match against all availability flags irrespective the order of availability-flags traversal, and second, one availability flag should match to only one deflection request. Therefore, following this simple observation, we can perform a structural-only modification to the basic load-distribution module and derive a new circuit that is twice as fast. We will present the delay-optimized load distribution in three steps.

The first step involves how the deflection requests are connected to the load distributor. In the original circuit the deflection requests $DR[1]$ and $DR[2]$ are connected to the first column of the 2D array. In this way, even though they could match with ports 1 and 2 they had to wait first the availability of port 0. This constraint can be removed by connecting $DR[1]$ and $DR[2]$ directly to the main diagonal of the array. This modification followed by the appropriate re-wiring is shown in the 1st diagram of Fig. 6. After this modification the propagation delay of the deflection requests is symmetric for all inputs.

[1]Inverters are omitted since they are merged with logic after synthesis.

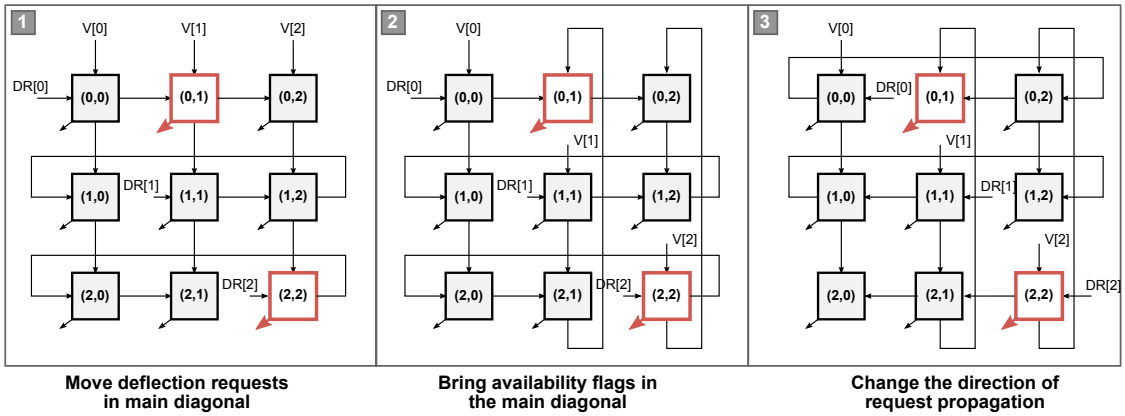

Figure 6: The transformation procedure to design a fast symmetric load distributor.

Although $DR[0]$ can match directly with output 0 if it is available, this does not hold for $DR[1]$ and $DR[2]$ and the rest outputs. To give the opportunity to $DR[1]$ and $DR[2]$ to try in the first stage to match with their symmetric outputs 1, and 2, as done for $DR[0]$, we can move appropriately the entrance points of the availability flags $V[1]$ and $V[2]$. The placement of $V[1]$ and $V[2]$ to the main diagonal of the circuit is shown in the 2nd diagram of Fig. 6.

Up to now, we managed to speed up the circuit and removed some unnecessary dependencies. However, still the deflection requests $DR[1]$ and $DR[2]$ have to wait the complete propagation of $DR[0]$ before matching with an available output. This delay overhead can be easily removed by changing the direction of propagation of the requests in all rows of the circuit, as shown in the 3rd diagram of Fig. 6. In this way, we end up with a fully symmetric circuit that has a maximum delay of 3 AND gates in every path.

The design of a fast n-to-n load distributor can be performed by starting with a original $n \times n$ array similar to Fig. 5, connecting all deflection requests and availability flags along the main diagonal of the array and then changing finally the direction of request propagation in each row.

4. DELAY ANALYSIS AND COMPARISONS

Exact delay comparisons between the proposed architecture and the most efficient previous proposal shown in Fig. 1 requires logic synthesis and placement & routing for both modules. However, in this case, the delay difference in terms of logic levels between the two modules under comparison is that big, that just counting suffices to prove the delay savings expected by the proposed implementation. Please notice that the delay of both circuits is gate limited and the contribution of the internal wires is rather small.

The switch allocator shown in Fig. 1 has a delay equal to the delay of an n-input sorting network plus the delay of n serially connected arbiters. The fastest sorting network has a delay equal to $\frac{1}{2}\left(\log_2^2 n + \log_2 n\right)$ comparators, while the delay of each arbiter is roughly equal to the delay of $\log_2 n + 1$ logic gates [11], [12]. In the proposed case, the critical path passes through a max-selection unit that involves only $\log_2 n$ comparators in series and a load distributor that imposes an extra delay of n gates.

The delay overhead of demuxes and OR gates shown in the block diagram of the proposed switch allocator are not counted, since they are inherent to any switch allocator [13] even in the case of buffered switches, and exist also in a full implementation of the sorting-based switch allocator of [1].

5. CONCLUSIONS

A practical switch allocator for bufferless switches has been presented in this paper that removes the serial dependency of output port allocation found in previous proposals and offers significantly faster circuit implementations.The proposed switch allocator can act as a new tool to the hand of network architects that do not need to consider anymore switch allocation in the case of bufferless switches as a slow and hard-to-design module, thus allowing them to derive new more efficient architectures.

6. REFERENCES

[1] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks", ISCA-36, 2009.

[2] Z. Lu, M. Zhong, and A. Jantsch, "Evaluation of on-chip networks using deflection routing", GLSVLSI-16, 2006.

[3] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato, "Reducing packet dropping in a bufferless NoC", Euro-Par-14, 2008.

[4] M. Hayenga, N. Jerger, and M. Lipasti, "Scarab: A single cycle adaptive routing and bufferless network", MICRO-42, 2009.

[5] K. Goossens and A. Hansson, "The Aethereal Network on Chip after Ten Years: Goals, Evolution, Lessons, and Future", DAC, 2010.

[6] J. Kim. "Low-cost router microarchitecture for on-chip networks", MICRO-42, 2009.

[7] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating bufferless flow-control for on-chip networks", NOCS, 2010.

[8] S. Tota M. R. Casu, and L. Macchiarulo, "Implementation analysis of NoC: a MPSoC trace-driven approach", GLSVLSI-16,2006.

[9] Z. Hong and R. Sedgewick, "Notes on merging networks", ACM STOC, 1982

[10] K. Harteros, "Fast Parallel Comparison Circuits for Scheduling",ăTech. Report FORTH-ICS/TR-304, 2002.

[11] G. Dimitrakopoulos, N. Chrysos and K. Galanopoulos, "Fast arbiters for on-chip network switches", ICCD, 2008.

[12] G. Dimitrakopoulos, "Logic-level implementation of basic switch components", in Designing Network On-Chip Architectures in the Nanoscale Era, Jose Flich and Davide Bertozzi, Eds., CRC Press, 2010.

[13] D. Becker and W. Dally, "Allocator implementations for network-on-chip routers", ACM/IEEE SC, 2009.