Low-cost fault-tolerant switch allocator for network-on-chip routers

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

Emmanouil Kalligeros
Information and Communications Systems Eng.
University of the Aegean
Karlovassi 83200, Samos, Greece
kalliger@aegean.gr

ABSTRACT

Reliable operation that can be checked on-line is of paramount importance to current and future systems-on-chips that are implemented in very deep submicron technologies. In such systems, the communication among architectural modules is handled by a modular network-on-chip infrastructure that should be sufficiently protected from transient faults that may affect its correct operation. The error protection mechanism should cover all fault scenarios and incur the minimum area/energy/delay overhead. In this paper, we propose such an on-line checking mechanism for the switch allocator of the router that detects every possible single transient or permanent fault in the arbiters and handles it appropriately, thus preserving the reliable operation of the switch.

Keywords

fault-tolerance, on-line testing, switch allocation, logic design

1. INTRODUCTION

Interconnection networks lie at the kernel of any complex SoC and provide a modular communication infrastructure that parallelizes the communication between system's modules by utilizing a network of switches connected with multiple point-to-point links. The switches are the basic building blocks of such interconnection networks and their design critically affects the performance of the whole system. The main role of the switches is to guide the incoming packets to the appropriate output, allowing them to move closer to their final destination. When two or more packets request the same output, only one wins and the rest have to wait for their turn in the input buffers of the switch. The inputs that are allowed to send their data are determined by the switch allocator. The switch allocator accepts the requests from each input and decides which one to grant in order to produce a valid connection pattern for the crossbar's multiplexer that handle the actual data switching.

INA-OCMC '12, January 25, 2012, Paris, France

The performance of the network's switches has increased in the recent years by better architecting and taking also advantage of the benefits of technology scaling [1]. However, in the future, such benefits will not come any more for free. The reduced transistor sizes and supply voltages have made logic gates more susceptible to single event transients (or upsets) that, when allowed to propagate to storage elements, lead to soft errors that alter the state of the system and may corrupt its functionality [2]. When such a transient fault hits one of the network's switches, several erroneous conditions arise that need to be efficiently handled on-line in order to mitigate their possible catastrophic effects.

In this paper, we focus on enhancing the reliability of the switch allocation logic by providing a self-checking capability to the arbiters of the switch [3]. The output of each arbiter is checked in every cycle for invalid decisions, blocking away any cases that cause incorrect network operation, such as packet misrouting, packet loss or packet corruption. Instead of treating every faulty scenario differently, we propose a new error detection mechanism that maps all the examined faulty cases to the same two erroneous conditions that are later detected by a newly introduced checker module. The proposed checker introduces insignificant area and delay overhead, thus providing a viable solution to both low cost and high-performance switches.

Relative to previous state-of-the-art on fault-tolerant switch allocation [4], [5], [6], this work (a) minimizes the number of employed checker modules by employing a new mapping technique, (b) covers in detail all possible single faults, and (c) allows for early system warning when the checker itself presents an internal fault.

In the following, we present in Section 2 the proposed fault tolerant arbiter architecture. In Section 3 we describe the new checker module, analyze its behavior in the presence of faults and measure its area/delay characteristics. Finally, conclusions are drawn in Section 4.

2. ARBITER ON-LINE TESTING

A generic arbiter, as shown in Fig. 1, consists of two parts; the arbitration logic that decides which request to grant based on the current state of the priorities, and the priority update logic that decides, according to the current grant vector, which inputs to promote. The width of the priority state associated with each input depends on the complexity of the priority selection policy. For example, a single priority bit per input is enough for simple round-robin policy, while for more complex weight-based policies such as age-based allocation, multi-bit priority state is needed. The arbitration

logic, besides the grant signals, produces also a no-request (NR) flag that declares the case that the arbiter did not receive any requests. This flag is already supported by the majority of the arbiters since it can be produced directly from the arbitration logic without any hardware overhead. Also, in many cases, such as wormhole switching, the arbiter's decision should be locked until the last flit of a packet (tail flit) leaves the switch. In this case, actual arbitration is performed only among head flits that inherit their grants to the following flits of the same packet. As long as the lock signal is asserted, the output of the arbiter remains unchanged and points to the same winning input.

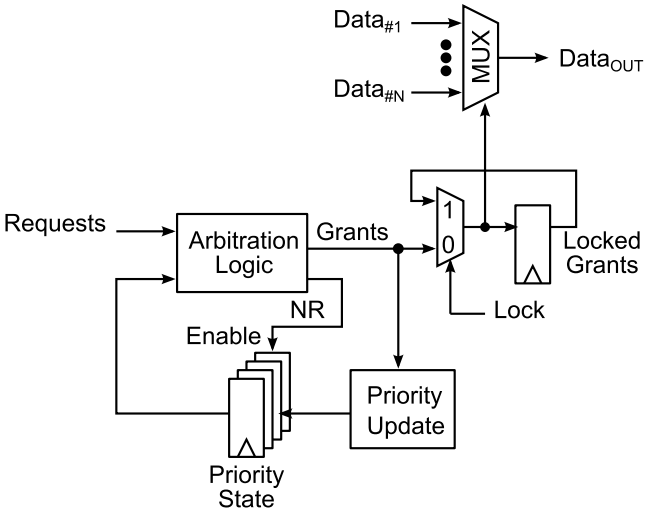

Figure 1: The baseline organization of an output arbiter and a multiplexer.

An arbiter's decision is considered valid when it grants only one of the incoming requests. If there aren't any active requests it should not provide any grants. In this way, the output multiplexer is configured to send to the output either one valid flit or none. Also, as long as the lock signal is asserted, the arbiter is not allowed to change its valid output. In the presence of a single error, permanent or transient, either inside the arbiter or at its output lines (grant signals) the behavior observed at the outputs of the arbiter may change as follows:

Multiple grants (MG): The arbiter produces multiple grants at each output that would cause the multiplexer either to merge in the output link the bits of many different flits or to send at the output the wrong flit. The exact behavior depends on the internal design of the multiplexer. This case should be avoided since it causes flit corruption or misrouting and a higher-level error recovery mechanism needs to be employed in order to recover the correct status of the network [7]. Therefore, we always need to detect this case and stop the data at the output of the multiplexer before leaving the switch. Additionally, the appearance of multiple grants to an output arbiter may cause an input to receive a wrong number of grants leading possibly to an unwanted multicast or an early flit dequeue. When a multiple-grant error is detected, the grants returned to the inputs from the erroneous arbiter should be nullified. In this way, it is guaranteed that the inputs always receive valid grants.

No grant (NG): The arbiter does not produce any grants even if there are active requests that can be served. This case does not affect the correctness of the network operation and it only degrades network performance by reducing the throughput of the switch. Therefore, we don't worry much about this case. When the faulty node that caused the no-grants case recovers to its correct value, the arbiter will soon return to its normal operation leaving the network unaffected. If NR=1 then the arbiter correctly did not produce any grants. However, when NR=0 it means that this happened due to a fault. Therefore, if both the bits of the grant vector and the NR flag are all equal to 0, an error exists in the arbiter.

Wrong grant (WG): The arbiter produces only one grant that makes the output look valid, *i.e.*, follows the onehot code, but it corresponds either to an inactive request or points to a position different from the one with the highest priority.

- Under a single fault, it is impossible for the arbiter to give a grant to an inactive request without simultaneously granting at least one active request[1]. Therefore, a wrong grant will always correspond to the multiple grants error condition, possibly involving inactive requests too.
- However, it is possible for the arbiter to grant an active request that does not have the highest priority. This condition may arise due to an error in the priority state. In this case, we can start serving the new input, even if another with higher priority should have won. Since the grant vector is valid, this decision is locked as a normal one and applied to the crossbar for all the flits of the packet.

In order to identify all the above error conditions, we need to check only two cases at the output of the arbiter. The first case involves the detection of multiple-asserted grants and the second one involves the detection of all-deasserted grants. In both cases, when referring to the grant vector of the arbiter we should include also the NR flag. Practically, we don't care about which module of the arbiter is actually the faulty one. For example, a possible fault in the priority state or the priority update logic may promote by mistake a low priority input. If this error does not translate to multiple grants or no grants as done in the case of MG and WG, it is just a performance error similar to NG and does not affect the correct operation of the network.

To protect the arbiter and the switch as a whole from the three possible fault scenarios, we design a new checker that checks the output of the arbiter and decides, in parallel to multiplexing, if the arbiter's decision ruins the correct operation of the switch. The checker reports an error only when it observes multiple-asserted grants, including the NR bit, or none of them. In every case, the checker is driven by the grant signals connected to the select lines of the multiplexer.

The operation of the output logic, shown in Fig. 2, is the following: When lock is equal to 0, we are free to take a new decision, checking at the same time the validity of the

[1]This behavior holds for all arbiters that are based on priority encoding, carry lookahead-like structures and the matrix arbiter, practically covering almost all possible implementations [8].

outputs of the arbiter. If an error is detected, lock stays at 0 (this is done since the head flit is not dequeued; no flit dequeueing is performed when an error is detected - see below) and the arbiter retries until the checker reports an error-free grant vector. When this happens, lock is set to 1 and the output of the switch starts serving the flits of the granted packet. After lock=1, the checker and the multiplexers receive the output of the register that locked the valid grant/NR vector. In this stage, the operation of the output of the switch is practically decoupled from the operation of the arbiter, and thus the system is protected by any fault that occurs in the arbiter in the meantime. Additionally, if a fault appears at the register that holds the onehot locked grants, then, under the single-fault assumption, this fault will translate either to a multiple grant or to a no grant case and will be detected by the checker. When the effect of such a fault is removed, the output of the switch will continue serving the flits of the selected input.

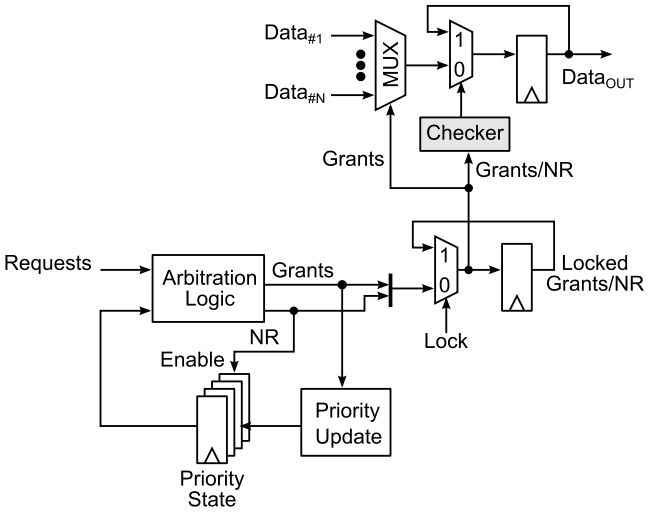

Figure 2: The on-line testable arbiter architecture.

As shown in Fig. 2, only when the checker decides that the output of the arbiter is valid, the data at the output of the multiplexer are latched by the output register and leave the switch. Also, the checker's output is used for masking the arbiter's grants. In case of an error, this mask protects the inputs from receiving multiple grants or a wrong grant that would cause the input to dequeue a flit that is not yet to leave the switch.

3. FAULT-TOLERANT CHECKER

The proposed checker receives as input the grant signals of the arbiter along with its NR bit. An error is detected when more than two input lines of the checker are asserted or when none of them is asserted. Equivalently, the checker reports a correct input when there is a single 1 in the input vector, *i.e.*, the checker's input follows the onehot code.

The proposed checker constitutes a binary-tree structure, which is constructed by identical nodes. Each node receives a pair of 3-bit wide inputs (HZF) from two nodes of the above tree level and generates 3 outputs, which follow the same encoding as the inputs (see Fig. 3). The inputs and the outputs of the checker nodes are also in onehot form and their meaning is the following:

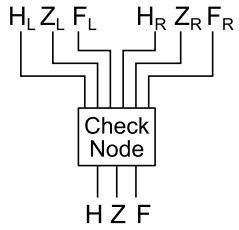

Figure 3: The interface of the basic building block of the checker

- H (oneHot bit): It means that there is exactly one input with value 1 in the examined subset of checker inputs. In this case, HZF = 100.
- Z (Zero bit): all of the examined checker inputs are 0; then, HZF = 010.
- F (Fault bit): It is set when there are two or more 1s in the examined subset of checker inputs. When this happens, HZF = 001.

The functionality of a checker node is quite simple: if any of its inputs is equal to 001 (fault case), then the output should be also 001. Additionally, the output should indicate a fault (001) if both inputs denote onehot (100). This means that there are two disjoint subsets of checker inputs that are onehot encoded, *i.e.*, there is a single 1 in both subsets, and hence, at least two 1s exist at the checker's inputs. Therefore, $F = F_L + F_R + H_L \cdot H_R$. It is now easy to understand that an onehot indication at the outputs of a node (100) should be generated if one of its input denotes onehot (100) and the other zero (010); consequently, $H = H_L \cdot Z_R + H_R \cdot Z_L$. Finally, the zero output should be set only when both node inputs indicate zero (010); as a result, $Z = Z_L \cdot Z_R$.

The adoption of onehot encoding within the proposed checker offers two significant advantages: a) in the presence of a single fault in the checker, the affected HZF triplet will assume an invalid (non-onehot) value, and b) it simplifies the implementation of the checker nodes. Note also that a zero-indicating triplet (010) at the inner nodes of the proposed checker does not imply a fault, since it can be later combined with an onehot-indicating triplet (100). However, if the zero combination (010) appears at the outputs of the checker, this means that the checker inputs are erroneous (all inputs are zero).

In order to generate the HZF triplets from the checker's inputs, a preprocessing stage is required. Each preprocessing (PP) node receives one input A_i and produces the corresponding HZF triplet. The boolean functions of the PP node outputs are trivial: $H = A_i$, $Z = \overline{A_i}$, and $F = 0$. The proposed checker, along with a running example for 4 inputs is shown in Fig. 4.

3.1 Fault analysis

We examine all four possible cases regarding the reliable operation of the arbiter and the checker:

Arbiter correct – Checker correct: Correct operation.

Arbiter faulty – Checker correct: The checker identifies the error (checker output: 001 or 010) and no data are

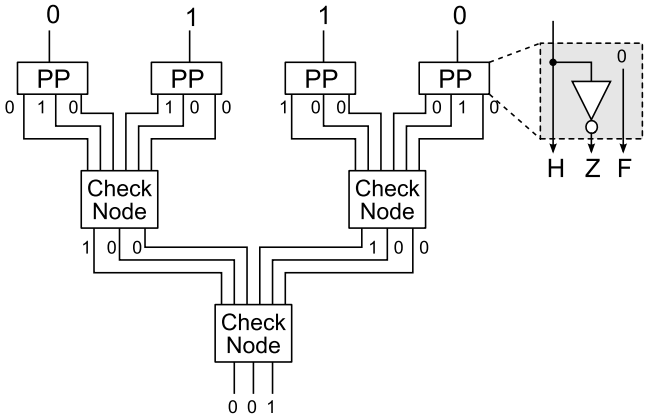

Figure 4: The structure and operation of a 4-input checker

leaving the corresponding output port. Also, the grants that return to the input ports are appopriately masked.

Arbiter correct – Checker faulty: The proposed checker is properly designed so that a single fault within its structure manifests itself at the outputs. Specifically, we have exhaustively verified that, assuming correct inputs, a fault affecting the value of any single line of the checker, will propagate to the outputs. The outputs' value will be different from the correct 100 value (not necessarily the 001 or the 010 fault-indicating output of normal checker operation, but any other output than the correct 100 – its exact value depends on the fault). Actually, the proposed checker is totally self checking under the single fault assumption, since it is both fault secure (in the presence of a fault, a non-valid output is produced), and self testing (all single faults are detected at the checker outputs by applying the set of non-faulty, *i.e.*, onehot, inputs) [9]. Even if we allow the correct, in this case, arbiter's decisions to affect the output of the switch, it is useful to know when the checker cannot ensure that the monitored circuit (*i.e.*, the arbiter) is working properly.

Arbiter faulty – Checker faulty: First of all, we have to note that this final case contravenes the single fault assumption, since faults occur in both the arbiter and its checker. However, this is an additional feature of the proposed checker; if a single fault occurs in the arbiter, this fault is reported at the checker's outputs, even in the presence of a single fault in the checker. As above, we have confirmed this behavior by exhaustively verifying that the outputs of the checker assume a different value from the correct one (100). We have to note that this property is due to the utilization of onehot encoding within the checker itself that does not allow a meaningful onehot triplet to be transformed to another one, under a single fault. In this case, problems would occur if 001, *i.e.*, fault, could change to 100, *i.e.*, onehot, or 100, *i.e.*, onehot, could change to 010, *i.e.*, zero, for multiple asserted grants, and if 010, *i.e.*, zero, could change to 100, *i.e.*, onehot, for all-deasserted grants at the arbiter's outputs.

3.2 Implementation results

We have implemented the on-line testable arbiter architecture shown in Fig. 2, in a 65nm CMOS technology using a standard-cell based design flow and compared it to the baseline structure of Fig. 1. The parameterized form of the circuits was described in VHDL, while synthesis, placement and routing were performed using Synopsys Design Compiler and Cadence SOC encounter, respectively. The arbiters of the switch followed the architecture of [10].

In the case of 4, 8 and 16 inputs, the area overhead of the checker is negligible ranging below 1%, when assuming at least a 32-bit wide datapath for the multiplexers. Also, the delay of the checking module increases the critical path of the switch for the examined cases by less than 5%, since, according to Fig. 2, the error flag of the checker controls the data loading operation of the register at each output port of the switch and runs in parallel to multiplexing.

4. CONCLUSIONS

The faults that appear in the arbiter's logic can affect the correct operation of the switch in several ways. The aim of this work is to reduce the possible faulty outcomes to only two errorneous cases that can be easily detected by the proposed low-cost on-line checker. When a fault is detected, packet loss, misrouting or corruption is avoided and the network continues its correct operation un-affected. Also, reliable operation is fully preserved since the checker is self-checked during the system's normal operation.

5. REFERENCES

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

[2] K. Mohanram and N.A. Touba, "Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits", Proc. of IEEE International Test Conference, pp. 893-901, 2003.

[3] S. Mitra and E. J. McCluskey, "Which concurrent error detection scheme to choose?," ITC 2000.

[4] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, C. R. Das "Exploring Fault-Tolerant Network-on-Chip Architecture", DSN, 2006.

[5] K. Constantinides, S. Plaza, J. Blome, B.Zhang, V. Bertacco, S. Mahlke, T. Austin, M. Orshansky, "BulletProof: a defect-tolerant CMP switch architecture, " in Proc. HPCA'06, pp.5-16, Feb. 2006.

[6] Q. Yu, M. Zhang, P. Ampadu, "Exploiting inherent information redundancy to manage transient errors in NoC routing arbitration," Proc. 5th ACM/IEEE Int. Symp. on Networks-on-Chip (NoCS'11), pp.105–112, May 2011

[7] S. Murali, T. Theocharides, N. Vijaykrishnan, M.J. Irwin, L. Benini, G. De Micheli, "Analysis of error recovery schemes for networks on chips," IEEE Design & Test of Computers, vol. 22, no. 5, pp. 434–442, 2005.

[8] 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.

[9] McCluskey, E. J., "Design techniques for Testable Embedded Error Checkers," IEEE Computer, Vol. 23, No. 7, pp. 84-88, July 1990.

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