2014 20th IEEE International Symposium on Asynchronous Circuits and Systems

An Asynchronous SDM Network-on-Chip Tolerating Permanent Faults

Guangda Zhang*, Wei Song*, Jim Garside*, Javier Navaridas* and Zhiying Wang†
*School of Computer Science, University of Manchester, Manchester, M13 9PL, United Kingdom
Email: {zhangga, songw, jgarside, javier.navaridas}@cs.man.ac.uk
†School of Computer Science, National University of Defense Technology, Changsha, 410073, China
Email: zywang@nudt.edu.cn

***Abstract*—Asynchronous circuits have been used to implement Networks-on-Chip (NoCs), resulting in asynchronous NoCs where the links are usually implemented as quasi-delay-insensitive (QDI) pipelines to tolerate delay variations. With the ageing process of circuits, permanent faults may happen on links at runtime, causing both data errors and deadlocks of the network. This paper presents an asynchronous Spatial Division Multiplexing (SDM) NoC which tolerates permanent faults on the QDI links. Using a time-out mechanism, a general fault-detection technique can locate the permanent fault in the deadlocked NoC. To recover the network, a *Drain*&*Release* technique releases fault-free network resources on the deadlocked path. The SDM NoC physically divides every link and buffer into multiple independent virtual circuits. By configuring the switch allocator, the faulty virtual circuit is blocked so that it will not be allocated to any packets. The succeeding traffic requesting the same link will go through other fault-free virtual circuits and the network function is recovered. With regard to intermittent faults, the previously blocked virtual circuit can be resumed when the fault disappears. Experimental results show the asynchronous SDM NoC can detect and recover from permanent faults with reasonable overhead.**

***Keywords*—network-on-chip; asynchronous; permanent fault; quasi-delay-insensitive; spatial division multiplexing; deadlock**

I. Introduction

Advanced semiconductor technology makes it possible to integrate multiple processing cores on a single chip, leading to a multi-core era. The increasing number of processing cores results in a growing demand for scalable and efficient on-chip communication fabrics. Network-on-Chip (NoC) [1] has emerged as an infrastructure promising to support this kind of communication. It has been extensively studied by both the academic and industrial communities.

Built from routers and links, NoCs can be either synchronous or asynchronous. Synchronous NoCs depend on distributing clocks with little skew over long distances; asynchronous NoCs remove this problem. An asynchronous NoC is implemented as an asynchronous circuit where all communication is controlled by handshakes [2]–[4]. This provides clear benefits, including simplified timing closure and reduced power consumption, which are especially attractive as the network is scaled up. Quasi-delay-insensitive (QDI) circuits [2] are a family of timing-robust asynchronous circuits. Since links of NoCs may be long, they are usually implemented in a QDI fashion so that delay variations are tolerated. However, the large number of long wires are vulnerable to faults which cause unexpected transitions. Fault-tolerance has become an urgent issue in reliability-critical digital systems.

Permanent faults may happen at runtime with the ageing process and will never disappear [5]. The resulting permanent errors will bring lifetime reliability problems, even causing chips to be discarded. For critical and ultra-expensive systems, keeping them working is important even with some performance loss, leading to a demand for runtime permanent-fault-tolerant systems. Most of permanent faults can be modelled as stuck-at faults, including stuck-at-0 (s-a-0) and stuck-at-1 (s-a-1) [6]. Some other permanent faults, such as stuck-open faults, may have floating values [6].

Permanent-fault-tolerance has rarely been studied on asynchronous NoCs. Handling runtime permanent faults on asynchronous circuits, especially QDI ones, is more difficult than on synchronous circuits. Besides data errors, which also happen on synchronous circuits, permanent faults on QDI circuits can halt the handshake process and cause deadlocks [7]. Considering a NoC, the deadlock may spread over the network, permanently blocking the traffic and paralysing the NoC.

This paper targets the design of an asynchronous NoC which tolerates runtime stuck-at permanent faults emerging on links. The management of permanent faults includes the fault detection and the network recovery. A new time-out mechanism utilizing characteristics of the deadlocked network is proposed to detect and locate positions of permanent faults. This fault-detection technique is general and it does not rely on any transient-fault-tolerant techniques. During the network recovery phase, a *Drain&Release* technique releases fault-free network resources (routers and links) which were deadlocked. Spatial Division Multiplexing (SDM) [8], [9] is employed where each link and buffer are divided into multiple independent virtual circuits. By configuring the switch allocator, the faulty virtual circuit is blocked. The succeeding traffic can go through the fault-free virtual circuits of the same link and the network function is recovered adhered with some performance degradation. Considering intermittent faults (which may happen as an early symptom of permanent faults) [10], a recovery mechanism can resume the use of the previously blocked virtual circuit.

The proposed fault-detection technique relies on analysing the pattern of the deadlocked network. As long as a permanent or intermittent fault on links deadlocks the NoC, the faulty link can be detected and the network is recovered afterwards. Asynchronous SDM NoCs using the proposed techniques are implemented and demonstrated with detailed experimental results.

1522-8681/14 $31.00 © 2014 IEEE
DOI 10.1109/ASYNC.2014.10

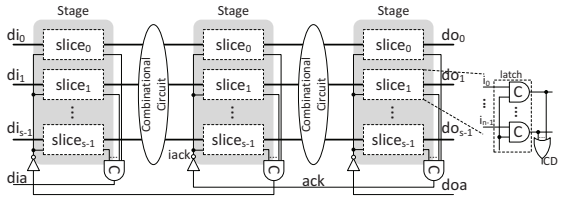
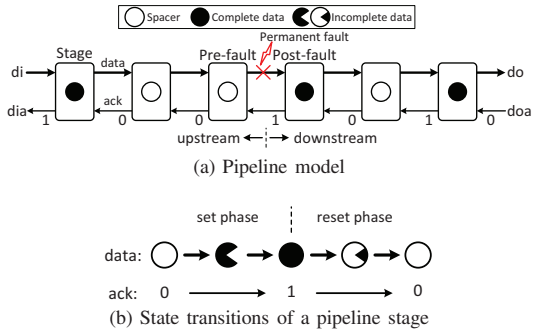


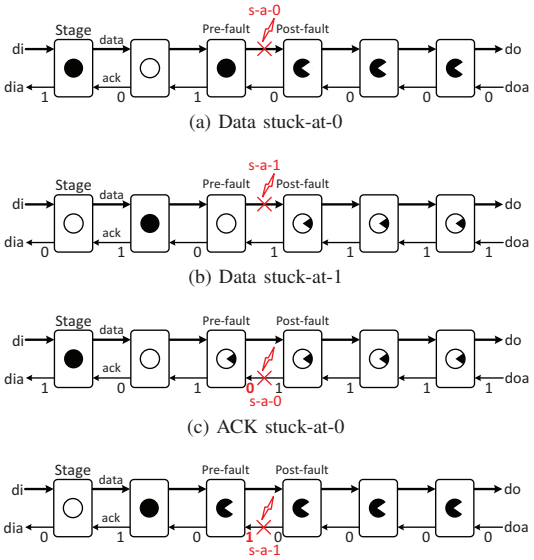
Fig. 1. A 4-phase 1-of-n QDI pipeline



(a) Pipeline model

(b) State transitions of a pipeline stage

Fig. 2. Modelling 4-phase QDI pipelines



(a) Data stuck-at-0

(b) Data stuck-at-1

(c) ACK stuck-at-0

(d) ACK stuck-at-1

Fig. 3. Deadlocked pipelines caused by permanent faults on link wires

II. Background

A generic asynchronous router can be modelled as a pipeline where the input/output buffers, crossbar and other components comprise the pipeline stages. A packet traversing through multiple routers experiences a long pipeline.

A. Modelling QDI pipelines

Fig. 1 shows a basic 4-phase QDI pipeline [2]. Each pipeline stage contains multiple 1-of-n slices which are asynchronous latches built from C-elements. An OR-gate acting as a completion detector (CD) notifies the arrival of a 1-of-n code. To synchronize slices, all CDs are connected to a multi-input C-element producing a common acknowledge signal (*ack*), whose positive transition denotes a complete word is latched.

A 4-phase QDI pipeline can be modelled as Fig. 2a, which depicts a transient state of the pipeline with all stages holding data or a spacer. Encoded as delay-insensitive (DI) codes [2], the data is either complete or incomplete. Complete data causes *ack* high while a spacer resets *ack* to low. Incomplete data exists between a complete data word and a spacer. The state transition of a pipeline stage is shown in Fig. 2b, which depicts the 4-phase handshake containing a set and a reset phase.

B. Deadlock caused by permanent faults

Unlike synchronous circuits, permanent faults cause deadlocks in QDI circuits, which disables traditional code redundant methods used in synchronous circuits. A permanent fault in a synchronous circuit typically causes a permanent error polluting the succeeding data. The circuit continues under the control of the clock, making it possible to detect the fault using accumulated history statistics of the circuit, i.e. error syndromes which are usually obtained by using transient-fault-tolerant techniques [11]. If the error syndromes satisfy specific patterns or some time-out conditions [5], [11], [12], the fault is taken as permanent and the recovery process is invoked; otherwise, it is transient or intermittent.

Controlled by handshake protocols, QDI circuits are event-driven. A permanent fault not only corrupts data, but also halts the handshake process, resulting in a deadlock. Most existing transient-fault-tolerant QDI designs [13], [14] are also deadlocked when a permanent fault happens. In this case, error syndromes cannot be easily obtained, making traditional fault-detection techniques [5], [11], [12] fail. In a NoC, the deadlock reserves some network resources, which may cause more deadlocks of the network, reducing the network performance and finally paralysing the whole network.

III. Detecting Permanent Faults on Inter-router Links of Asynchronous NoCs

A. Deadlock analysis

A permanent fault divides a QDI pipeline into "upstream" and "downstream" stages (Fig. 2a). The first upstream stage connected to the faulty link is the "pre-fault" stage while the first downstream stage is the "post-fault" stage. Permanent faults happening on a 4-phase QDI pipeline can be classified into four classes (Fig. 3):

1) **Data stuck-at-0** (Fig. 3a) A stuck-at-0 fault on a data wire prevents a '1' from being transmitted to the post-fault stage. As a result, all the downstream stages are stuck at the set phase. They keep waiting for the lost '1'.
2) **Data stuck-at-1** (Fig. 3b) A stuck-at-1 fault on a data wire prevents all downstream stages from being reset as they keep holding the invalid '1'.
3) **Ack stuck-at-0** (Fig. 3c) A stuck-at-0 fault on the *ack* wire prevents the pre-fault stage from being reset. As a result, all downstream stages keep waiting for a spacer. Their *ack* signals are all '1's
4) **Ack stuck-at-1** (Fig. 3d) A stuck-at-1 fault on the *ack* wire prevents the pre-fault stage from latching new in-

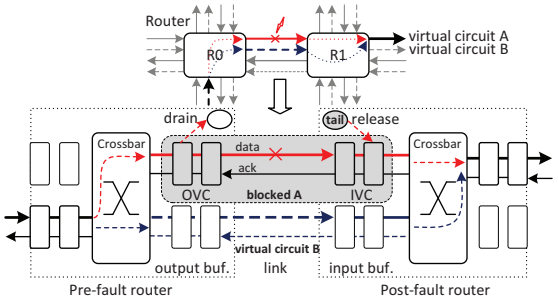


Fig. 4. Connection between two adjacent SDM routers (2 virtual circuits)

coming '1's, making all downstream stages stuck at the set phase. All *ack* signals of downstream stages are '0's.

It can be concluded that, all the above cases share the same pattern: ***the deadlock caused by a permanent fault leads to a steady state where all downstream pipeline stages have the same ack while the ack signals in upstream stages are alternately valued***. This deadlock pattern is the key to detecting permanent faults on QDI pipelines.

B. Fault detection on asynchronous NoCs

A permanent fault on a link of the NoC will deadlock the reserved path built by a packet. The faulty link divides the deadlocked path into *upstream* and *downstream* parts. The *upstream* routers hold the fault-free flits of the packet while the *downstream* routers may hold polluted flits. The first *upstream* router before the faulty link is termed as a "pre-fault" router while the first *downstream* router after the faulty link is the "post-fault" one. By checking each pair of adjacent routers and exchanging some information, the faulty link can be located.

A time-out mechanism is applied to each router to monitor all links. The intermediate link is the faulty one when the following conditions are satisfied by the two adjacent routers:

1) Transitions are detected on neither routers (indicating an idle, a temporarily blocked or a deadlocked link);
2) The *ack* signals at the preceding router are alternately valued. For the succeeding router, all pipeline stages on the reserved path have the same *ack*.

IV. Recovering Asynchronous NoCs from Permanent Faults

A. SDM NoC

Many existing NoCs use virtual channels [15] sharing one physical link. A fault on the link will pollute the traffic in all virtual channels. As a result, techniques such as adaptive routings are often used to detour around the defective link to recover the network [15].

This paper describes SDM [8], [9] used to construct the NoC. Assuming an SDM router has p w-bit input/output ports, every link and buffer are physically divided into m (w/m)-bit wide virtual circuits so that the crossbar becomes a $pm \times pm$ (w/m)-bit wide one [8]. Fig. 4 illustrates the connection of two adjacent SDM routers with two virtual circuits. Each virtual circuit contains an output virtual circuit (OVC), an input virtual circuit (IVC) and intermediate link wires. A single fault on the link affects only the traffic in one virtual circuit.

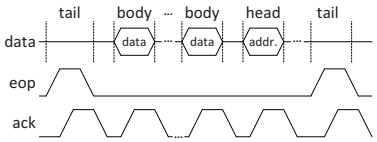


Fig. 5. The flit sequence in a virtual circuit

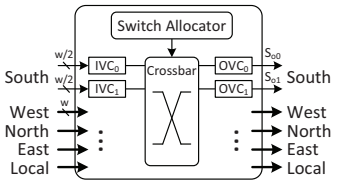


Fig. 6. An SDM router (2 virtual circuits)

B. Network recovery from permanent faults

Two operations are needed to recover the SDM NoC from a permanent fault. The first is to block the faulty virtual circuit. By configuring the allocator of the pre-fault router, the faulty virtual circuit (A in Fig. 4) is blocked so that no further packets will be allocated to the defective link. The traffic will go through other fault-free virtual circuits (B in Fig. 4). The second is to release fault-free network resources on the deadlocked path, achieved by a *Drain*&*Release* method:

- ***Drain*** Since the flits remaining in the *upstream* routers are fault-free, by creating a sink at the output of the pre-fault router (Fig. 4), all the remaining flits can be drained out from the *upstream* routers.
- ***Release*** The release of *downstream* routers can be achieved by creating a tail flit at the input of the post-fault router (Fig. 4). The fake tail flit will be transmitted through all *downstream* routers to release them one by one (incomplete data in *downstream* routers should be cleaned first).

V. Implementation of an Asynchronous SDM NoC Tolerating Permanent Faults

This section presents a 2D-mesh asynchronous SDM NoC to demonstrate how the above permanent-fault-tolerant techniques can be implemented. 4-phase 1-of-4 protocol is applied to both routers and links. The NoC employs an XY-dimension-ordered routing algorithm and wormhole routing [16]. Each packet is divided into head, body and tail flits (Fig. 5). The destination address is stored in the head flit followed by a sequence of body flits. A tail flit, indicated by an end-of-packet (*eop*) signal, is used to separate consecutive packets.

A. Structure of a basic asynchronous SDM router

The SDM router with five directions is shown in Fig. 6. Fig. 7 presents the input/output of two adjacent SDM routers. Each input virtual circuit (IVC) contains pipelined input buffers, an XY-controller (generating routing requests to the allocator) and a buffer controller (controlling the flit flow)

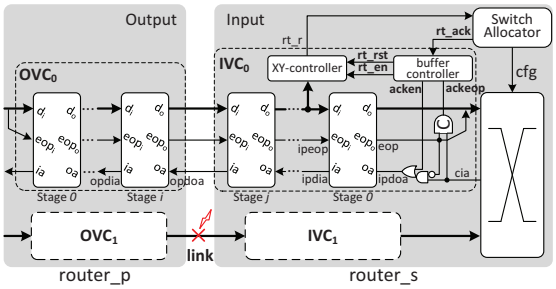
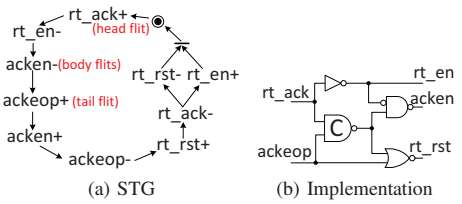


Fig. 7. An unprotected link between adjacent SDM routers (2 virtual circuits)



(a) STG (b) Implementation

Fig. 8. Buffer controller [9]

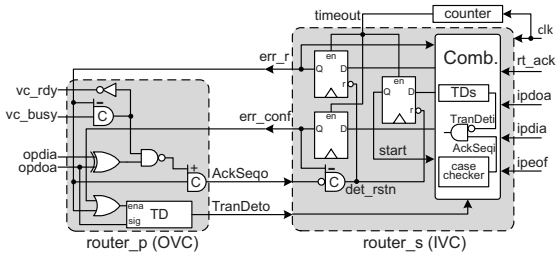
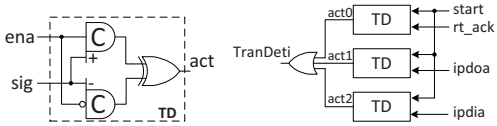


Fig. 9. Fault-detection circuits monitoring one virtual circuit



(a) Transition detector (b) Transition detectors at input

Fig. 10. Structure of transition detectors

while each output virtual circuit (OVC) contains output buffers. Details of the XY-controller and the crossbar can be found in [9]. Fig. 8 presents the Signal Transition Graph (STG) of the buffer controller and its implementation.

Initially a head flit is blocked before *Stage*0 of the input buffer. Using the destination address, the XY-controller (Fig. 7) generates a routing request *rt_r* to the switch allocator. The allocator will allocate an idle output buffer to the requesting packet and build a path in the crossbar (indicated by *rt_ack*+). Then XY-controller is disabled (*rt_en*-) and *Stage*0 is enabled (*acken*-). Body flits of this packet start traversing through the crossbar. When a tail flit is detected (*eop*+) and received by the output (*cia*+, leading to *ackeop*+), the buffer controller blocks *Stage*0 (*acken*+), withdraws the tail flit (*ackeop*-) and resets the XY-controller (*rt_rst*+, so that the request *rt_r* is cleared). As a result, the allocated path is released (*rt_ack*-). The input virtual circuit starts waiting for the next packet (*rt_en*+).

B. Fault-detection circuits for one virtual circuit

This paper focuses on runtime stuck-at faults on the inter-router links. Fault-detection is executed at the input and output of each pair of adjacent routers. Fig. 9 presents the structure of fault-detection circuits monitoring one virtual circuit between router_p and router_s. Assume the traffic flow is from router_p to router_s. The intermediate link is defective (this virtual circuit is faulty) if router_p is an *upstream* router and router_s is a *downstream* router of the faulty link (they are pre-fault and post-fault routers respectively).

One of the key components for fault-detection is a transition detector (TD) monitoring the transition of a certain signal. Fig. 10a presents a TD (a similar design was presented in [17]). The *sig* is the signal being monitored and *ena* is an active-high enable signal. Initially the detector is disabled by a low *ena* and it outputs '1'. When *ena* is high, the detector starts monitoring *sig*. The outputs of the two C-elements are either both high or both low, leading to a low *act*. If *sig* changes, only one C-element changes its output. As a result, *act* is set high denoting a transition is detected on *sig*. If *act* keeps low when TD is enabled, no transitions are detected.

In each input virtual circuit, three *ack* signals (*rt_ack*, *ipdia* and *ipdoa*) are monitored by three TDs (Fig. 10b) located in the combinational circuit (Comb.) in Fig. 9. The *start* signal enabling each TD is controlled by a state machine. During the detection process (*start* is '1' so that *TranDeti* is '0' initially), a high *TranDeti* denotes transitions are detected on these signals so that this input buffer is not deadlocked. If no transitions are detected (*TranDeti*-) for some time, indicating the input buffer is idle or blocked (which may be caused by network congestion or a permanent fault), the *ack* sequence in the input buffer is checked against the deadlock pattern (Section III-A). For a *downstream* router, a permanent fault causes one of the following cases in the input buffer (Fig. 7 and Fig. 9):

Case 1: As the general deadlock scenario for all kinds of stuck-at faults (Fig. 3), the allocator has allocated an output to this input buffer (*rt_ack*+) and two consecutive *ack* signals in the input buffer are equal. This is indicated by (*rt_ack* & *ipdia*==*ipdoa*). In this case, the head flit has been accepted and the input buffer holds the polluted flit without being cleared.

Case 2: An *Ack stuck-at-1* or a *Data stuck-at-0* fault may prevent this input from receiving a complete head flit (*ack* signals are low). As a result, the XY-controller cannot produce a valid request (*rt_r*) to the allocator. No output buffer is granted to this input (*rt_ack*-). This is indicated by (!*rt_ack* & !*ipdia* & !*ipdoa*).

Case 3: A *Data stuck-at-1* fault may affect the *eop* wire when the input is idle, creating a fake tail flit. The fake tail flit will reach the front of the queue (*Stage*0 in the input buffer, Fig. 7) and keep blocking the input buffer. Without receiving a valid head flit, the input buffer will not be granted by the allocator (*rt_ack*-). This case is indicated by (!*rt_ack* & *ipeop* & !*ipdoa*).

These cases are checked by a case checker in the input virtual circuit whose output is *AckSeqi* (Fig. 9). If none of them is satisfied, *AckSeqi* is low indicating the router_s is not a *downstream* router of the defective link. Network

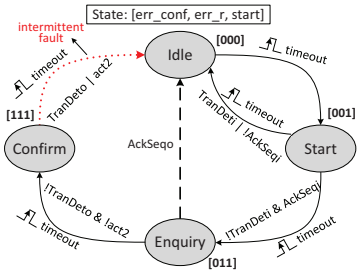


Fig. 11. The flow to detect permanent faults

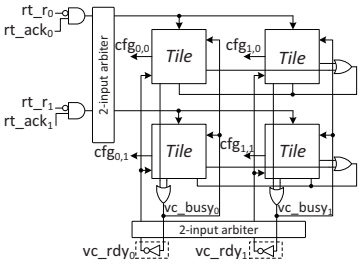


Fig. 12. Switch allocator [9], [18]

congestion may cause a temporary blockage of a packet, which is distinguished from a fault-caused deadlock because none of the above cases is satisfied.

In the output virtual circuit of router_p, a TD monitors the *opdoa* in Fig. 9, outputting a high *TranDeto* when transitions are detected. An XOR gate examines if two adjacent *ack* signals (*opdia* and *opdoa*) are equal (indicated by *AckSeqo*). For an *upstream* router of the defective link, no transitions can be detected and adjacent *ack* signals should be unequal. Therefore, a high *TranDeto* or *AckSeqo* indicates router_p is not *upstream*.

C. A time-out mechanism detecting the permanent fault

Using the above fault-detection circuits, a time-out mechanism is used to detect the faulty virtual circuit between each pair of routers (router_p and router_s, Fig. 9). Each router has a counter producing a *timeout* signal, which controls a synchronous state machine at each input virtual circuit. The state machine has three flip-flops: *err_r*, *err_conf* and *start*. Its state transition graph is presented in Fig. 11. The state transition is only enabled by *timeout* except when returning to **Idle** from **Enquiry** (the dashed line in Fig. 11).

Idle is the default state after reset. The state machine transits to **Start** when the first *timeout* comes.

Start checks the input virtual circuit to decide if router_s is a *downstream* router of the defective link (Fig. 9). The state machine will return to **Idle** if either transitions are detected in the input buffer (*TranDeti+*), or the *ack* sequence of the input buffer violates the deadlock pattern of the *downstream* router (*AckSeqi-*, none of the three cases in Section V-B is satisfied). Otherwise, router_s is probably a *downstream* router (the input buffer may be idle rather than deadlocked by a fault). When the second *timeout* arrives, the state machine proceeds to **Enquiry** to examine the output behaviour of the preceding router by enabling the transition detector in the output virtual circuit (Fig. 9), starting the third time-out period.

At **Enquiry**, the state machine will be immediately reset to **Idle** (*AckSeqo+*) if either the *ack* sequence at the output buffer violates the deadlock pattern of *upstream* routers, or this output virtual circuit is idle (*vc_busy-*). If *AckSeqo* is low, and no transitions are detected in this output buffer (*TranDeto-*, Fig. 9) and the input buffer of router_s (*act2-*, Fig. 10b) during the third time-out period, this output buffer satisfies the deadlock pattern of the *upstream* router (the idle case is excluded). Bear in mind that router_s has been taken as a possible *downstream* router. It can be inferred that router_p is an *upstream* router while router_s is a *downstream* one. As a result, when the third *timeout* comes, the fault detection circuit in the input virtual circuit of router_s will be confirmed that router_s is the post-fault router (router_p is the pre-fault one). The faulty virtual circuit (containing the defective link wire) is precisely located. The state machine transits to **Confirm** (*err_conf+*) to invoke the network recovery operation.

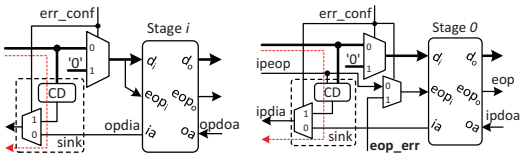
The state machine is driven by signals from the asynchronous circuit. An And-gate (in Comb. of Fig. 9) guarantees that only when these sampled signals are assumed stable (no transition activities for a time-out period, *TranDeti-*), the generated *AckSeqi* can trigger the state transition to **Enquiry**. Even if some of the monitored signals just toggle when *timeout* arrives, resulting in metastability, the state machine transits to either **Enquiry** where the deadlock pattern is re-examined, or **Idle** where the detection process restarts. The sync/async interface requires no synchronizer and the metastability does not affect the detection accuracy.

It can be inferred that it needs two to four time-out periods to detect the faulty virtual circuit from the occurrence of the deadlock. The state machine will get stuck at **Confirm** to block the faulty virtual circuit permanently. Considering long lasting intermittent faults, a recovery mechanism is required to revoke the asserted *err_conf* and resume the usage of the previously blocked virtual circuit when the fault disappears. A transition from **Confirm** to **Idle** is added (the red dotted arc in Fig. 11). When the fault disappears, some transitions may happen on data, *eop* or *ack* wires. These further cause transitions on *opdoa* or *ipdia* which can be detected by transition detectors (*TranDeto+* or *ack2+*). When the next *timeout* arrives, the state machine is reset to **IDLE**. The blocked virtual circuit can be reused. This method can also tackle permanent faults with floating values [6] to some extent. It can be concluded that, as long as a permanent or intermittent fault on the inter-router link causes a deadlock in the NoC, it can be detected using this fault-detection method.

D. Blocking the faulty virtual circuit

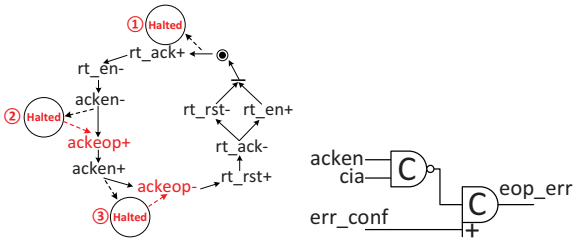
The network recovery process is invoked by a high *err_conf* indicating the defective link is located. The next step is to block the faulty virtual circuit and release the deadlocked fault-free network resources. The purpose of blocking the faulty virtual circuit is preventing the succeeding traffic from being allocated to it, which can be achieved by configuring the switch allocator of the pre-fault router.

In SDM routers the output has multiple separated virtual circuits so that the allocator needs to allocate one of the



(a) Output buffer (b) Input buffer

Fig. 13. Modified output/input buffer for network recovery



(a) Deadlocked STG (b) EOP-generator

Fig. 14. Buffer controller at deadlocked states

multiple resources to one request each time. Fig. 12 shows a multi-resource arbiter [9], [18] used to implement the switch allocator where two requests compete for one of the two resources (i.e. output virtual circuits). Two 2-input arbiters are used to select one request and one output virtual circuit respectively. The state of the output virtual circuit is indicated by *vc_rdy* or *vc_busy* (vc_rdy_i=!vc_busy_i). A low *vc_rdy* indicates the output virtual circuit has been allocated to a request and it cannot be reallocated to another request before it is released. Details can be found in [9], [18].

To block the faulty virtual circuit, an asymmetric C-element is inserted before the inverter generating *vc_rdy* from *vc_busy* (Fig. 9). It ensures that the fault can be confirmed (*err_conf*+) only when *vc_busy* is high. As a result, when the defective link is detected, the allocator of the pre-fault router surely has allocated this link to one request (*vc_busy*+). This makes *vc_rdy* low permanently as long as the fault exists, preventing the following packets from using this defective link again. The succeeding packets requesting the same direction will be detoured to other fault-free virtual circuits.

E. Modifying output buffers for the Drain operation

To release the fault-free network resources on the upstream deadlocked path, a *Drain* operation is executed at the output of the pre-fault router. The modified output buffer is presented in Fig. 13a. Using a sink which sends back a low or high *ack* when detecting a spacer or a complete data word, the remaining fault-free flits blocked in *upstream* routers will traverse through the previously allocated path and get drained at this output. A multiplexer is used to clear data (including the *eop*) when a fault is detected so that the remaining flits from the upstream will not affect the defective link during the *Drain* operation. With the proceeding of the tail flit, the *upstream* routers are released one by one. The completion detector (CD) of the sink can reuse the CD of the preceding pipeline stage if the output buffer has multiple stages.

F. Modifying input buffers for the Release operation

The *Release* operation is executed at the input of the post-fault router to release the downstream deadlocked routers and links. The modified input buffer is shown in Fig. 13b. Three multiplexers are inserted before *Stage*0 to change the connection when a fault is detected (*err_conf*+): the uppermost multiplexer is used to clear the incomplete data in the downstream deadlocked path; the bottom one and a CD (which can reuse the CD of the preceding stage if the input buffer has multiple stages) comprise a sink generating *ack* (*ipdia*) to the preceding pipeline stage. The intermediate one is used to create a tail flit. At **Confirm**, the STG of the buffer controller may be halted at three places (①, ② and ③ in Fig. 14a).

① A fault may pollute a head flit or prevent a head flit from entering into the input of the post-fault router. As a result, the XY-controller cannot generate a valid request (*rt_r*) to the allocator. Without being granted (*rt_ack*-), the polluted head flit is blocked before *Stage*0 of the input buffer which is the end of the deadlocked path. This post-fault router is the only *downstream* router. No further operation is required.

For the other two cases, *rt_ack* to this input buffer is high (Fig. 7) and there are multiple *downstream* routers. The *Release* operation starts releasing downstream network resources by creating a tail flit. The circuit generating the tail flit (EOP-generator) is shown in Fig. 14b. When the faulty virtual circuit is located (*err_conf*+), *eop_err* goes to *Stage*0 of the input buffer and replaces the original *eop* (Fig. 13b and Fig. 7). It should be noticed all signals in the input buffer of the post-fault router are stable. The replacement of *eop* in a deadlocked asynchronous circuit is safe.

② A fault may deadlock the network when a body or tail flit is traversing through the link. As a result, the tail flit cannot reach *Stage*0 of the input buffer (*ackeop*-, Fig. 7). The buffer controller is halted at *acken*- (Fig. 14a). The uppermost multiplexer in Fig. 13b clears data ensuring a low *cia* (Fig. 7) after the fault is detected. Thus, a high *eop_err* (Fig. 14b) can replace the original low *eop* and be transmitted to the output, which is equivalent to creating a tail flit. The fake tail flit is detected by the buffer controller (*ackeop*+, Fig. 7). The STG transits from *Halted* to *ackeop*+ and finally reaches the initial state waiting for a new packet (Fig. 14). The fake tail flit will traverse through all *downstream* routers and release them one by one.

③ An *eop stuck-at-1* or an *Ack stuck-at-0* fault may block the buffer controller at *acken*+, which is equivalent to a tail flit without being withdrawn (*cia*-, Fig. 7). When the fault is detected, a low *eop_err* will replace the original *eop* and transmit to the output (leading to *ackeop*-). The STG will transits to *ackeop*- and finally reaches the initial state waiting for a new packet. With the withdrawal of the tail flit, *downstream* routers are released in sequence.

For all the cases, after the *Release* operation, the input virtual circuit of the post-fault router is halted and keeps waiting for a valid head flit (i.e. a new fault-free packet). When the fault disappears (it is intermittent), transitions will be detected at the input/output of the post/pre-fault router (the faulty bit will get drained at the sink of the input buffer of the post-fault router). Then the blocked link is resumed (*err_conf*-) and packets can be allocated to this virtual circuit again.

G. Some technical details

There is no requirement on the skew, jitter and frequency of the clock signal used by the fault-detection. The clock needs to drive only the state machine. It can be easily got from local synchronous IP cores or other clock sources. Different routers may use different clocks, producing different time-out periods. The only requirement is that the time-out period should be longer than the time needed by a packet to traverse a router, ensuring the *Drain&Release* operations are completed before the next *timeout* arrives (which starts a new fault-detection cycle). The possible signal transitions happening during the *Drain&Release* process will not cause transitions of the state machine. The state machine can leave **Confirm** only when the fault disappears (an intermittent fault case).

The fault-detection brings four extra wires to each virtual circuit (Fig. 9). Currently, these extra wires along with the added logic are not protected. They increase the area and may have a negative impact on the chip reliability. However, there are far fewer activities on these extra wires and logic than those on the usual data wires and router logic, especially when a long timeout period is used. Considering the fact that most runtime permanent faults coming with the ageing process are strongly related to the workload or activities of the device [19], these redundant circuits are expected to have a significant longer lifetime than the usual ones. The overall reliability of the system increases through protecting the links. Using some physical redundancy techniques [13], the extra circuits can be protected.

This paper focuses on tolerating the stuck-at faults only on the inter-router links rather than the logic inside routers. With the ageing process, all transistors and wires in a chip will experience a degradation process and may face permanent faults. Although this paper has not targeted the permanent faults inside routers, to our best knowledge, there is no previous publication which fully tolerates the permanent faults on the links between QDI routers. It is believed that the method proposed in this paper can be extend to protect the switches inside QDI routers, which will be explored in the future.

VI. Experimental Results

A. Hardware evaluation

The protected asynchronous SDM routers using the proposed techniques are implemented using the UMC $130nm$ standard cell library. Asynchronous cells, such as C-elements, are built using standard cells. As a comparison, unprotected SDM routers are also implemented. Besides the implementation details revealed in Section V, the input buffer of a router has two pipeline stages while the output buffer has one.

Fig. 15a compares the area of different routers under different configurations. *DW* is the data width of a link (in *bits*) which has *VN* virtual circuits. On average, the area of the protected router increases by 17.0% compared with that of the unprotected router. As an example, the area of the unprotected router with *DW*=64, *VN*=2 is $92701\mu m^2$ while the area of the protected one using the same configuration increases by 18.3% and reaches $109684\mu m^2$. Notice that although the router with *DW*=64, *VN*=2 has a larger data width than the *DW*=32, *VN*=4 one, the latter router has larger area due to the more buffer controllers, and the larger allocator and crossbar caused by the larger *VN* [9].

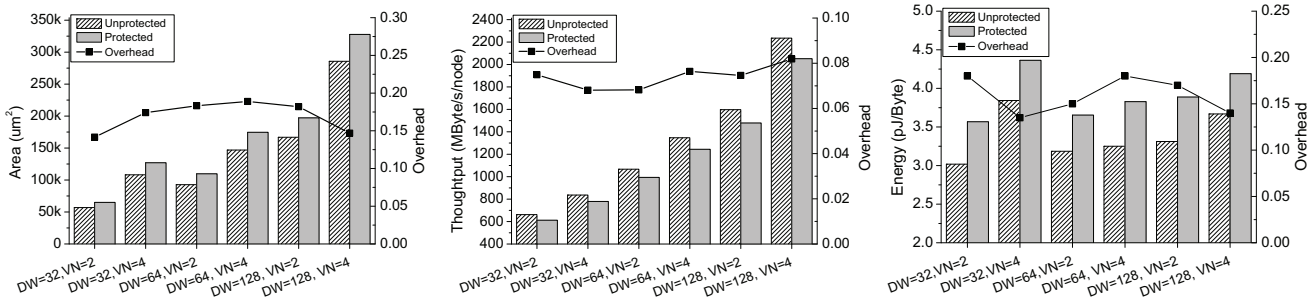
To evaluate the network performance, a SystemC/Verilog mixed environment is built to connect 16 post-synthesis routers (annotated with the gate latency), constructing a 4×4 2D-mesh NoC. The synchronous IP cores are SystemC models which inject data into the network using the maximum rate. The packet size is fixed at 64-byte. The flit size is the same as the data width of a virtual circuit. All packets are generated randomly and the network traffic is uniformly distributed. The clock and time-out frequencies are set to 100MHz and 1.5MHz respectively. Fig. 15b presents the saturation throughput of different NoCs which increases with *DW* and *VN*. Compared with the unprotected NoC, the throughput of the protected NoC decreases by 7.4% on average. Fig. 15c summarizes the energy consumption of different routers. On average the energy of protected SDM routers increases by 16.0% compared with unprotected ones.

B. Fault-tolerance evaluation

Using the built environment, permanent (or intermittent) faults can be inserted on any links at any time to verify the fault-tolerance of the network. The test results show that for a 1-bit fault in the NoC, the fault-detection achieves 100% detection accuracy and the network is successfully recovered. For intermittent faults which disappear after some time, the previously blocked virtual circuit can be reused again. As an example, Fig. 16 compares the throughput of the unprotected and protected NoCs (*DW*=64, *VN*=2) where a stuck-at-1 fault is inserted. The time-out period is set to $10\mu s$ which is far larger than the packet latency through a router (around $70ns$). The fault is inserted on the *East* link of router(1,2) to the Y+ direction at $30\mu s$, leading to a steep decrease of the network throughput. For the unprotected NoC (Fig. 16a), the throughput finally drops to 897MByte/s/node which is decreased by 16%. For the protected NoC, the throughput is around 993MByte/s/node initially. The fault is detected at $50.2\mu s$ (Fig. 16b). The throughput first decreases to 749MByte/s/node and then returns to 954MByte/s/node after the network is recovered, which is 6% higher than the final throughput of the unprotected NoC. The proposed technique can tolerate some multi-bit permanent faults if the resulting deadlocked paths have no intersections and the faults do not block a whole link. When multi-bit faults happen, multiple virtual circuits may be blocked, leading to a fully blocked link. Using adaptive routings [15], the network can be recovered.

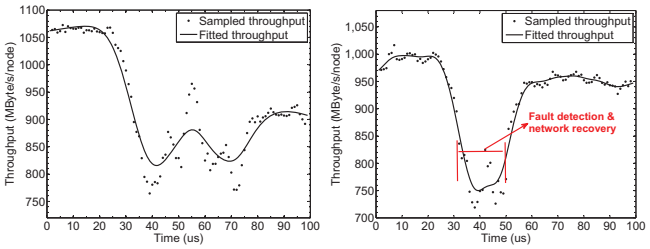
C. Comparison with related work

Most existing permanent-fault-tolerant NoCs are either synchronous [5], [12] or bundled-data designs [11]. Their fault-detection techniques cannot be easily used in asynchronous NoCs with QDI links [3] or QDI NoCs [4], [9]. A time-out mechanism using delay-lines was proposed in [15] to detect both transient and permanent faults on the LEDR [20] encoded links in an asynchronous NoC. Its router is a bundled-data self-timed design. Their method cannot be used in a QDI NoC but ours can. The fault-detection technique proposed in our paper uses a time-out mechanism controlled by clocks. It is general, independent of any transient-fault-tolerant techniques, and can be used in 4-phase QDI NoCs [4], [9] to detect permanent faults on links (including both *data* and *ack* wires). A method similar to the presented *Drain&Release* was proposed in [21]



(a) Router area (b) Saturaion throughput (c) Energy consumption

Fig. 15. Performance evaluation of different NoCs



(a) Unprotected (b) Protected

Fig. 16. Throughput of NoCs with a stuck-at-1 fault

to deal with LEDR [20] encoded links rather than the 4-phase 1-of-n links in this paper. Techniques including spare wires replacement [11], splitting transmission [22] and fault-tolerant routings [12], [15] have been used to recover the network from permanent faults. Without relying on these conventional techniques, this paper makes use of the SDM to recover the network. To our best knowledge, this is the first research on permanent-fault-tolerant asynchronous SDM NoCs.

VII. Conclusion

The links of asynchronous NoCs are usually implemented as QDI pipelines to tolerate delay variations. This paper presents an asynchronous SDM NoC tolerating permanent faults on links. Such a fault will cause a deadlock, which can be precisely detected by monitoring the transition activity and the *ack* sequence of adjacent routers. A time-out mechanism controls the detection process. The fault-detection technique can locate a permanent or intermittent fault as long as it deadlocks the network. To recover the network, the router is implemented using SDM. Using a *Drain&Release* technique, the fault-free deadlocked network resources are released. By configuring the allocator, the defective link is blocked and all the following traffic to the same direction will be allocated to other virtual circuits. Consequently, the network function is recovered with some performance loss. For an intermittent fault, the previously blocked link can be resumed to use.

Acknowledgment

The authors would like to thank the various grants from the National Natural Science Foundation of China (61272144), the China Scholarship Council, and the Engineering and Physical Sciences Research Council (EP/I038306/1).

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. of DAC*, 2001, pp. 684–689.

[2] J. Sparsø and S. B. Furber, *Principles of Asynchronous Circuit Design: a Systems Perspective*. Kluwer Academic Publishers, 2001.

[3] R. Dobkin, R. Ginosar, and A. Kolodny, "QNoC asynchronous router," *Integration, the VLSI Journal*, vol. 42, no. 2, pp. 103–115, 2009.

[4] F. Clermidy, C. Bernard, R. Lemaire, J. Martin, I. Miro-Panades, Y. Thonnart, P. Vivet, and N. Wehn, "A 477mW NoC-based digital baseband for MIMO 4G SDR," in *Proc. of ISSCC*, 2010, pp. 278–279.

[5] T. Lehtonen, D. Wolpert, P. Liljeberg, J. Plosila, and P. Ampadu, "Self-adaptive system for addressing permanent errors in on-chip interconnects," *IEEE Tran. VLSI*, vol. 18, no. 4, pp. 527–540, 2010.

[6] S. A. Al-Arian and D. P. Agrawal, "Physical failures and fault models of CMOS circuits," *IEEE Tran. Circuits and Systems*, vol. 34, no. 3, pp. 269–279, 1987.

[7] C. LaFrieda and R. Manohar, "Fault detection and isolation techniques for quasi delay-insensitive circuits," in *Proc. of DSN*, 2004, pp. 41–50.

[8] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor, "Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip," *IEEE Tran. Computers*, vol. 57, no. 9, pp. 1182–1195, 2008.

[9] W. Song and D. Edwards, "Asynchronous spatial division multiplexing router," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 85–97, 2011.

[10] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," *IEEE Micro*, vol. 23, no. 4, pp. 14–19, 2003.

[11] T. Lehtonen, P. Liljeberg, and J. Plosila, "Online reconfigurable self-timed links for fault tolerant NoC," *VLSI Design*, 2007.

[12] C. Feng, Z. Lu, A. Jantsch, M. Zhang, and Z. Xing, "Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router," *IEEE Tran. VLSI*, vol. 21, no. 6, pp. 1053–1066, 2013.

[13] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits," in *Proc. of ASYNC*, 2005, pp. 156–165.

[14] G. Zhang, W. Song, J. D. Garside, J. Navaridas, and Z. Wang, "Transient fault tolerant QDI interconnects using redundant check code," in *Proc. of DSD*, 2013, pp. 3–10.

[15] M. Imai and T. Yoneda, "Improving dependability and performance of fully asynchronous on-chip networks," in *Proc. of ASYNC*, 2011, pp. 65–76.

[16] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2003.

[17] Y. Shi, S. B. Furber, J. Garside, and L. A. Plana, "Fault tolerant delay insensitive inter-chip communication," in *Proc. of ASYNC*, 2009, pp. 77–84.

[18] S. Golubcovs, D. Shang, F. Xia, A. Mokhov, and A. Yakovlev, "Modular approach to multi-resource arbiter design," in *Proc. of ASYNC*, 2009, pp. 107–116.

[19] R. Aitken, G. Fey, Z. T. Kalbarczyk, F. Reichenbach, and M. Sonza Reorda, "Reliability analysis reloaded: How will we survive?" in *Proc. of DATE*, 2013, pp. 358–367.

[20] M. E. Dean, T. E. Williams, and D. L. Dill, "Efficient self-timing with level-encoded 2-phase dual-rail (LEDR)," in *Proc. of Advanced Research in VLSI*, 1991, pp. 55–70.

[21] T. Yoneda, M. Imai, N. Onizawa, A. Matsumoto, and T. Hanyu, "Multi-Chip NoCs for automotive applications," in *Proc. of PRDC*, 2012, pp. 105–110.

[22] M. Zhang, Q. Yu, and P. Ampadu, "Fine-grained splitting methods to address permanent errors in network-on-chip links," in *Proc. of ISCAS*, 2012, pp. 2717–2720.