Deadlock Recovery in Asynchronous Networks on Chip in the Presence of Transient Faults

Guangda Zhang*, Jim Garside*, Wei Song† and Javier Navaridas*, Zhiying Wang‡
*School of Computer Science, University of Manchester, Manchester, M13 9PL, United Kingdom
Email: {zhangga, jgarside, javier.navaridas}@cs.man.ac.uk
†Computer Laboratory, University of Cambridge, Cambridge, CB3 0FD, UK
Email: ws327@cam.ac.uk
‡School of Computer Science, National University of Defense Technology, Changsha, 410073, China
Email: zywang@nudt.edu.cn

***Abstract*—Asynchronous Networks-on-Chip (NoCs) have been proposed as a promising infrastructure to provide scalable and efficient on-chip communication for many-core systems. Using the Quasi-delay-insensitive (QDI) implementation, asynchronous NoCs could achieve timing-robustness. However, the advancing semiconductor technology leads to shrinking transistor dimensions and increasing chip density, accelerating the occurrence of faults, especially transient faults. Transient faults emerging on QDI circuits could cause not only data errors (symbol corruption and insertion), but also deadlock. When the deadlock happens on asynchronous NoCs, it can spread over the whole network and paralyse its function. This deadlock has not been fully studied while most traditional fault-tolerant techniques cannot deal with it. Using a new model built for QDI pipelines, the formation and behaviour of the deadlock caused by transient faults are systematically studied. Using the summarized deadlock patterns, the fault position can be precisely located and the fault type can be diagnosed. A fine-grained recovery mechanism is proposed to recover the network from different deadlocks. As a design case, an asynchronous NoC is designed which can recover from the deadlock caused by both transient and permanent faults on links. Detailed experimental results are given.**

I. Introduction

With the developing of semiconductor technologies, more and more intellectual property (IP) cores can be integrated on a single chip, proposing an urgent requirement for scalable, efficient and reliable on-chip communication. Network-on-Chip (NoC) [1] emerges as a promising infrastructure to support this kind of communication. It can be implemented using either synchronous or asynchronous circuits. Most existing NoCs are synchronously built. Controlled by handshake protocols instead of global clocks, asynchronous NoC has many promising advantages over the synchronous one [2]. It divides the whole chip into multiple independent synchronous domains, constructing a Globally-Asynchronous, Locally-Synchronous (GALS) system [3] (Fig. 1a), enabling individual voltage/frequency control and simplifying the chip-level timing closure. Quasi-delay-insensitive (QDI) circuits [2] are a family of asynchronous circuits that can tolerate delay variations. Using the QDI implementation, the resulting QDI NoC is timing-robust and could tolerate Process-Voltage-Temperature (PVT) variations.

On the other hand, increasing chip density, shrinking transistor dimensions, increasing clock frequency and decreasing critical charge continuously intensify the fragility of electronic devices to environmental variations, accelerating the occurrence of faults, especially transient faults [4]. Transient faults usually last for a short period. They can be provoked by various sources, including noise [5], electromagnetic interference, electrostatic discharge [4] and radiation [6] (Fig. 1b). Besides transient faults, permanent faults may also happen at runtime with the ageing process, bringing lifetime reliability problems [7]. As a result, high reliability has become an essential design objective for critical electronics.

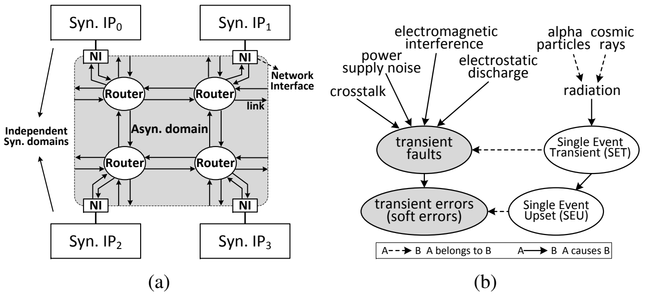
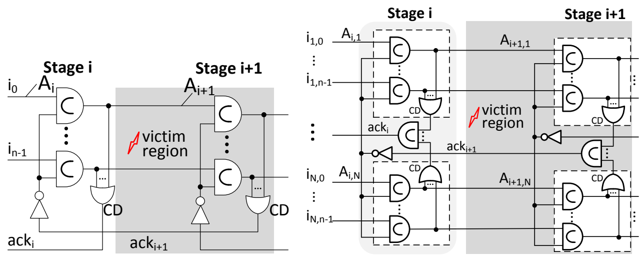


Fig. 1. (a) A Globally-Asynchronous, Locally-Synchronous (GALS) system constructed by an asynchronous NoC (b) Sources of transient faults

In traditional synchronous NoCs, transient faults typically cause data errors. The packet transmission can be assured to be running because the synchronous sampling proceeds under the control of the clock. In QDI circuits without the clock, time is elastic. Data is encoded into delay-insensitive (DI) codes to contain timing information [2]. As a result, transient faults emerging on asynchronous NoCs could cause not only data errors (including symbol corruption and symbol insertion), but also deadlock [8], [9]. This deadlock is different from the conventional one happening in the routing layer [10]. Most traditional fault-tolerant techniques, including the error detection/correction codes, cannot deal with it. This deadlock has not been fully studied. It could spread over the whole asynchronous NoC and paralyse its function. The error detection and correction on synchronous NoCs have been fully studied [11]; recovery of the deadlock caused by transient faults on asynchronous NoCs represents a different challenge.

This paper focuses on the deadlock caused by transient faults emerging on asynchronous NoCs. A new model is built to demonstrate the handshake process of 4-phase 1-of-n QDI



(a) Single-word (b) Multi-word (N-word)

Fig. 2. 4-phase 1-of-n QDI pipelines

pipelines and the effect of transient faults. Using this pipeline model, the formation and behaviour of the deadlock caused by transient faults are systematically studied. Common deadlock patterns are summarized, using which the fault position can be precisely located. Utilizing the difference between deadlock patterns caused by transient and permanent faults, the fault type can be diagnosed. A fine-grained recovery mechanism is proposed to recover the network from the deadlock so that system reboot is avoided when deadlock happens. Finally, an asynchronous NoC using the proposed techniques is implemented which can recover from the deadlock caused by both transient and permanent faults on links. Since the proposed techniques do not rely on the network structure, they can be used in any 4-phase QDI pipelines to deal with the deadlock caused by transient faults.

II. Pipeline models and deadlock analyses

A. Modelling 4-phase 1-of-n QDI pipelines

Buffers are usually inserted to the input and output of a router to support specific flow control and improve the network performance. Therefore, a generic asynchronous router can be modelled as a QDI pipeline where the input/output buffers construct pipeline stages. A packet traversing through multiple routers will experience a long QDI pipeline. Fig. 2a shows a single-word 4-phase 1-of-n QDI pipeline with two stages built from half-buffer latches. The OR-gates are completion detectors (CD) which acknowledge the preceding stages the successful latching of a complete data word (*ack*+) or a spacer (*ack*-). Fig. 2b shows a multi-word pipeline built from N single-word sub-pipelines. To synchronize all sub-pipelines, their CDs are connected to a multi-input C-element to produce a common acknowledge signal (*ack*) to the preceding stage.

A fault may occur on any of the n data wires of a single-word 1-of-n pipeline or the *ack* wire. To identify the fault position, the wire carrying '1' in a handshake cycle is defined as the "active" wire while the others are "inactive". The faulty wire (or the output wire of the faulty gate) is defined as the "victim" wire. For an N-word ($N \geq 1$) wide pipeline (Fig. 2b), $A_{i,j}$ ($1 \leq j \leq N$) belonging to the j-th sub-pipeline is the input active wire of *Stage i*. Produced at *Stage i*, ack_i is the *ack* signal to the preceding *Stage* (*i*-1).

A linear pipeline with depth d can be divided into $(d-1)$ *segments*, each of which contains two continuous pipeline stages. In the proposed pipeline model, *segment* is used as the object of study. The state of one *segment* S_i ($1 \leq i \leq d$) contains input active wires and output *ack* wires of both stages (*Stage i* and *Stage* (*i*+1)). Therefore, we have S_i=($\{A_{i,1}...A_{i,N}\}$, ack_i, $\{A_{i+1,1}...A_{i+1,N}\}$, ack_{i+1}). The initial state after reset is S_i = ({0...0}, 0, {0...0}, 0).

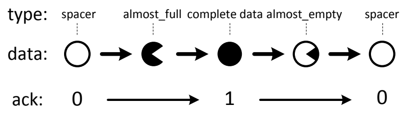


Fig. 3. State transitions of a 4-phase QDI pipeline stage

Production Rule Set (PRS) for basic logic gates [12] is borrowed to describe the behaviour of a pipeline *segment*. A production rule with a form of $A \wedge B \longrightarrow C\uparrow, D\downarrow$ is equivalent to $A \wedge B \longrightarrow C\uparrow$ and $A \wedge B \longrightarrow D\downarrow$, meaning that if both A and B are *true*, the assignments $C\uparrow$ and $D\downarrow$ are fired. The firing of $C\uparrow$ and $D\downarrow$ is independent and has no sequence. According to the 4-phase handshake protocol, the PRS for an N-word pipeline *segment* S_i is ($1 \leq j \leq N$):

$$A_{i,1} \wedge ... \wedge A_{i,j} ... \wedge A_{i,N} \wedge \neg ack_{i+1} \longrightarrow ack_i \uparrow \tag{1}$$

$$A_{i,j} \wedge \neg ack_{i+1} \longrightarrow A_{i+1,j} \uparrow \tag{2}$$

$$\neg A_{i,1} \wedge ... \wedge \neg A_{i,j} ... \wedge \neg A_{i,N} \wedge ack_{i+1} \longrightarrow ack_i \downarrow \tag{3}$$

$$\neg A_{i,j} \wedge ack_{i+1} \longrightarrow A_{i+1,j} \downarrow \tag{4}$$

The PRS of the environment of *segment* S_i is defined as follows:

$$ack_i \longrightarrow A_{i,j} \downarrow \tag{5}$$

$$\neg ack_i \longrightarrow A_{i,j} \uparrow \tag{6}$$

$$A_{i+1,1} \wedge ... \wedge A_{i+1,j} ... \wedge A_{i+1,N} \longrightarrow ack_{i+1} \uparrow \tag{7}$$

$$\neg A_{i+1,1} \wedge ... \wedge \neg A_{i+1,j} ... \wedge \neg A_{i+1,N} \longrightarrow \neg ack_{i+1} \downarrow \tag{8}$$

B. Deadlock caused by transient faults

A transient fault could happen on any wires and gates of a pipeline, including pipeline stages (including the latch and CD), *data* and *ack* wires. The proposed model uses two assumptions to analyse the pipeline:

1) Though multi-bit faults could happen in practical, only 1-bit fault is considered to simplify the proposed model since it is enough to deadlock a QDI pipeline.
2) It is assumed that a fault could happen only after *Stage i* in *segment* S_i, which is called the "victim" region (the grey area in Fig. 2). *Stage i* is the pre-fault stage while *Stage* (*i*+1) is the post-fault one. Thus, a fault could affect only A_{i+1} and ack_{i+1} while the input side (A_i and ack_i) is fault-free. If a fault happens on *Stage i*, it will be considered in *segment* S_{i-1} rather than S_i.

In a 4-phase 1-of-n QDI pipeline, data is encoded as delay-insensitive (DI) symbols which are either *complete* or *incomplete*. Incomplete data exists between a complete data word and a spacer. Fig. 3 shows the normal state transition of a 4-phase QDI pipeline stage. For an N-word 1-of-n pipeline (N>2), two types of incomplete data are defined: *almost_full* and *almost_empty*. The *almost_empty* data has only one 1-of-n symbol. After receiving one more 1-of-n symbol, the *almost_full* symbol will become complete. It has been reported that a transient fault on 4-phase 1-of-n QDI pipelines could cause symbol corruption (1-of-n is changed to 2-of-n), symbol insertion (a new 1-of-n code is inserted to the original data sequence, leading to misalignment) or deadlock [8], [9]. This paper focuses on the deadlock caused by a 1-bit transient fault.

Definition 1 (Deadlock): In a *deadlocked* QDI pipeline, no transitions could be fired any more and the pipeline gets stuck at a "stable" state.

For the half-buffer latch built from C-elements, it means inputs of the C-element fail to match so that no transition could happen at its output.

Theorem 1: A transient fault happening on a single-word 4-phase 1-of-n QDI pipeline can cause symbol corruption and insertion, but not deadlock.

Proof: It has been proved that a transient fault could cause symbol corruption and insertion on a single-word 4-phase 1-of-n QDI pipeline [8], [9]. Here we need to prove only that a transient fault cannot deadlock the pipeline.

Assuming a transient fault on the "victim" region of *segment* S_i could deadlock the pipeline, the input active wire and the *ack* wire to the pre-fault stage *Stage i*, (A_i, ack_{i+1}), should get stuck at either (11) or (00) to avoid any state transitions according to the deadlock definition.

a) (A_i, ack_{i+1}) gets stuck at (11).

A_i comes from the external environment of this *segment* and it is fault-free. A positive fault may fire $ack_{i+1}\uparrow$ but will disappear after some time. A_{i+1} should keep positive to make ack_{i+1} high and ack_i should keep negative to make A_i high according to (7) and (6). Because a transient fault can make A_{i+1} positive for only some time, a positive A_i should have been latched and outputted by *Stage i* to keep A_{i+1} high in the 1-bit fault case. As a result, ack_i will go high, leading to a negative A_i according to (5), which is in contradiction to the assumption.

b) (A_i, ack_{i+1}) gets stuck at (00).

Similarly, a negative fault may fire $ack_{i+1}\downarrow$ but will disappear after some time. A negative A_i should have been latched and outputted by *Stage i* to make ack_{i+1} negative. However, this will make ack_i low, permitting A_i to go high according to (6), which is in contradiction to the assumption.

Therefore, a transient fault on a single-word 4-phase 1-of-n QDI pipeline cannot cause deadlock. ■

In a multi-word pipeline, multiple sub-pipelines are synchronized using a multi-bit wide C-element connected to their CDs, which is different from a single-word pipeline (Fig. 2). In other words, there are two kinds of synchronization points in a multi-word pipeline. One is the asynchronous latch which synchronizes the forward and backward events ($A_{i,j}$ and ack_{i+1}). The other is the CD which synchronizes the parallel multiple forward events ($A_{i,1}, ..., A_{i,N}$ in Fig. 2b) and generates the backward event (ack_i). A 2-word pipeline is used as an example in the following, which can be extended to all multi-word or m-of-n (2≤m<n) pipelines.

Theorem 2: In a deadlocked multi-word 4-phase QDI pipeline, the two adjacent *ack* signals of the faulty *segment* cannot be equal.

Proof: Assume that ack_i and ack_{i+1} of *segment* S_i are equal. Since *Stage i* is in the fault-free region (Fig. 2b), the part of the *segment* state $(\{A_{i,1}A_{i,2}\}, ack_i, ack_{i+1})$ should be either $(\{11\}, 0, 0)$ or $(\{00\}, 1, 1)$ in a deadlock state.

If ack_{i+1} is negative, $ack_i\uparrow$ will surely be fired according to (1) which is in contradiction to the assumption. If ack_{i+1} is positive, $ack_i\downarrow$ will surely be fired according to (3) which is in contradiction to the assumption as well. Therefore, ack_i and ack_{i+1} cannot be equal in a deadlock state. In other words, the *segment* state $(\{A_{i,1}A_{i,2}\}, ack_i, ack_{i+1})$ should be either ({11},0,1) or ({00},1,0) ($A_{i,j}$ is equal to ack_{i+1} according to the *deadlock* definition). ■

Theorem 3: A transient fault could deadlock a multi-word 4-phase 1-of-n QDI pipeline.

Proof: Since it is assumed that a transient fault happens only in the victim region (Fig. 2), *Stage i* is fault-free. If the pipeline could be deadlocked, the final state of *segment* S_i must contain either $(\{A_{i,1}A_{i,2}\}, ack_i, ack_{i+1})$ =({11},0,1) or ({00},1,0) according to Theorem 2.

a) $(\{A_{i,1}A_{i,2}\}, ack_i, ack_{i+1})$ = ({11},0,1).

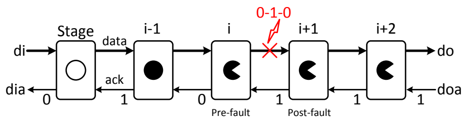
In this case, the two positive inputs $\{A_{i,1}A_{i,2}\}$={11} should not be latched and outputted together by *Stage i* (otherwise, ack_i is positive according to (1), which is in contradiction to the precondition).

1) If neither $A_{i,1}$ nor $A_{i,2}$ is latched, both $A_{i+1,1}$ and $A_{i+1,2}$ are negative eventually according to (4) (a transient fault cannot change the final value of ack_{i+1} in this case), leading to a negative ack_{i+1} according to (8), which is in contradiction to the precondition.
2) Assume only one of $\{A_{i,1}A_{i,2}\}$ has been latched by *Stage i* ($A_{i,1}$ for example, it means the positive transition of $A_{i,2}$ is delayed). In this case, a positive transient fault could make ack_{i+1} high according to (7). The *segment* state is shown in (9). According to the PRS (Section II-A), no transitions could happen as long as the environment of the *segment* does not change which means the pipeline is deadlocked.

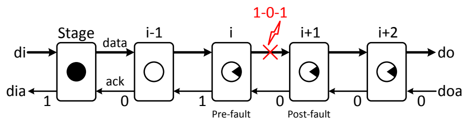
$$\begin{aligned}(\{A_{i,1}A_{i,2}\}, ack_i, \{A_{i+1,1}A_{i+1,2}\}, ack_{i+1})\\ = (\{11\}, 0, \{10\}, 1)\end{aligned} \tag{9}$$

In this case, the normal transmission of $A_{i,2}\uparrow$ is so slow that it arrives at *Stage i* later than $ack_{i+1}\uparrow$. As a result, $A_{i,2}\uparrow$ is blocked before *Stage i* by the positive ack_{i+1} forever. Since $A_{i,2}\uparrow$ cannot be latched, ack_i will be negative forever and the input positive $A_{i,1}$ cannot be reset though it has been latched by *Stage i*.

Stage (*i,j*) denotes the *i*th pipeline stage of sub-pipeline *j* ($1 \le j \le N$). In this deadlocked state, all downstream stages from *Stage* (*i*,1) in the first sub-pipeline keep holding the same 1-of-n word and waiting for being reset. For the second sub-pipeline where the transmission of $A_{i,2}\uparrow$ is delayed, all downstream stages from *Stage* (*i*,2) keep holding spacers. The positive ack_{i+1} blocking the latching of $A_{i,2}\uparrow$ may be caused by a positive transient fault on the input *data* wire or the half-buffer latch of *Stage*(*i*+1,2). The fault first gets latched by the latch of *Stage*(*i*+1,2), which triggers the CD and sets ack_{i+1} high. After the fault disappears, all the latched fault bit is



(a) A positive fault



(b) A negative fault

Fig. 4. Deadlocked pipelines caused by transient faults on link wires

cleared. A positive fault happening on CD of *Stage* (i+1) could fire the $ack_{i+1} \uparrow$ directly.

b) $(\{A_{i,1}A_{i,2}\}, ack_i, ack_{i+1})$ = ({00},1,0).

Similarly, only one of the $(A_{i,1}A_{i,2})$={00} could have been latched by *Stage i*. The transmission of the negative $A_{i,2}$ is so slow and later than $ack_{i+1} \downarrow$ which can be caused by a negative transient fault on the input *data* wire to *Stage* (i+1,2), the half-buffer latch or the CD of *Stage* (i+1,2). As a result, the deadlock state is:

$$\begin{aligned}(\{A_{i,1}A_{i,2}\}, ack_i, \{A_{i+1,1}A_{i+1,2}\}, ack_{i+1})\\ = (\{00\}, 1, \{01\}, 0).\end{aligned} \tag{10}$$

The above proof can be extended to all multi-word (or m-of-n, 2≤m<n) pipelines. Therefore, a transient fault could deadlock a multi-word 4-phase QDI pipeline. Along with the deadlock, a transient fault could cause symbol corruption and insertion as well, whose proof is omitted in this paper. The network will continue and the polluted packets can be corrected using other fault-tolerant techniques. This paper focuses on the deadlock of multi-word QDI pipelines which could cause serious consequence when it happens on QDI NoCs. ■

C. Deadlock analyses

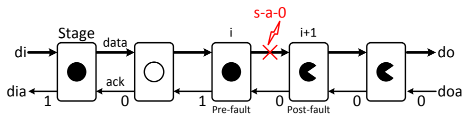
For an N-word wide 4-phase QDI pipeline, four binary sets are defined as follows ($\mathbb{A}_i = \{A_{i,j} | 1 \leq j \leq N\}$):

- **complete_data** ($\mathbb{A}_{i,complete}$): all elements are '1's.
- **almost_full** ($\mathbb{A}_{i,almost\_full}$): all elements are '1's except for the fault-related one which is '0'.
- **spacer** ($\mathbb{A}_{i,spacer}$): all elements are '0's.
- **almost_empty** ($\mathbb{A}_{i,almost\_empty}$): all elements are '0's except for the fault-related one which is '1'.

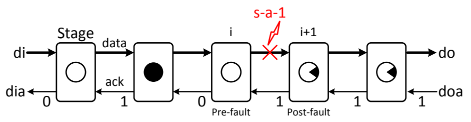
It can be concluded from Theorem 3 that possible deadlock states of an N-word 4-phase QDI pipeline caused by a transient fault are (11) and (12). Fig. 4 presents two examples that pipelines are deadlocked by transient faults on link wires. Data symbols listed in Fig. 3 are used.

$$\begin{aligned}(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1})\\ = (\mathbb{A}_{i,complete}, 0, \mathbb{A}_{i+1,almost\_full}, 1)\end{aligned} \tag{11}$$

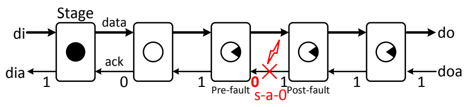
$$\begin{aligned}(\mathbb{A}_i, ack_i, \mathbb{A}_{i+1}, ack_{i+1})\\ = (\mathbb{A}_{i,spacer}, 1, \mathbb{A}_{i+1,almost\_empty}, 0)\end{aligned} \tag{12}$$



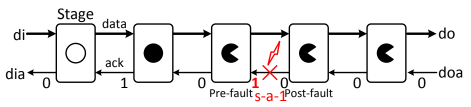
(a) Data stuck-at-0



(b) Data stuck-at-1



(c) ACK stuck-at-0



(d) ACK stuck-at-1

Fig. 5. Deadlocked pipelines caused by permanent faults on link wires [7]

T$_{skew}$ is used to represent the data skew between the two slowest parallel sub-pipelines. The loop latency T$_{loop}$ can be expressed as (13) where T$_{forward}$ and T$_{backward}$ represent the propagation delay of the forward *data* and the backward *ack* respectively. T$_{forward}$ and T$_{backward}$ can be expressed as (14) and (15) where t$_{latch}$, t$_{CD}$, t$_{dwire}$ and t$_{awire}$ are propagation delays of the asynchronous latch, the CD, *data* wires and the *ack* wire, respectively. It can be concluded that (16) is the necessity that a transient fault deadlocks a 4-phase QDI pipeline.

$$T_{loop} = T_{forward} + T_{backward} \tag{13}$$

$$T_{forward} = 2t_{latch} + t_{dwire} \tag{14}$$

$$T_{backward} = t_{CD} + t_{awire} \tag{15}$$

$$T_{skew} > T_{loop} \tag{16}$$

D. Deadlock detection and fault diagnosis

It can be found from Fig. 4 that the deadlock caused by a transient fault has the same pattern as the pattern caused by permanent faults [7]. The common deadlock pattern shared by both faults is:

1) No transitions are detected on the pipeline;
2) All pipeline stages downstream of the fault have the same *ack* while the *ack* signals in stages upstream of fault are alternately valued.

Without considering permanent faults, the position of the transient fault causing deadlock can be located using the proposed techniques in [7]. If both transient and permanent faults are considered, the fault type should be diagnosed first and then different recovery methods are used. Fig. 5 presents four cases that stuck-at permanent faults deadlock the pipeline, which has been fully studied [7]. Comparing Fig. 4 with Fig. 5, it can be found that, transient and permanent faults

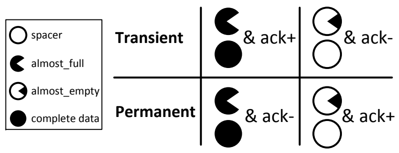


Fig. 6. Deadlock patterns of stages downstream of the fault

cause different combinations of the input *data* symbol and the generated *ack* (ack_{i+1}) at the post-fault stage (*Stage* (i+1)).

For the deadlock caused by a transient fault, the input to the post-fault stage is an *almost_full/complete_data* symbol with a positive *ack* to the preceding stage, or an *almost_empty/spacer* symbol with a negative *ack*. For permanent faults, the input to the post-fault stage is an *almost_full/complete_data* symbol with a negative *ack*, or an *almost_empty/spacer* symbol with a positive *ack*. Fig. 6 concludes the difference which is the key to diagnose the fault type.

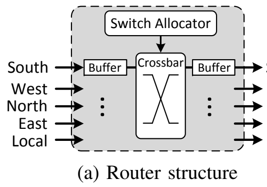
III. Implementation of an asynchronous NoC

This section presents the implementation of a 2D-mesh asynchronous NoC to demonstrate the deadlock detection, fault diagnosis and network recovery processes. Both transient and permanent faults are considered. The NoC uses 4-phase 1-of-4 protocol and works in a QDI fashion.

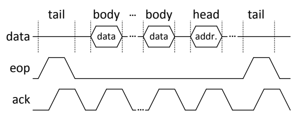
A. Router structure

Fig. 7a presents an asynchronous wormhole router with five bidirectional ports. Pipelined buffers using 4-phase 1-of-4 protocols are inserted at both the input and output ports of the router. A switch allocator controls the connection between the input and output ports through the crossbar. The NoC employs an XY-dimension-ordered routing algorithm and wormhole routing [10]. Packets are divided into head, body and tail flits (Fig. 7b). Address information is stored in the head flit while a tail flit indicated by a positive end-of-packet (*eop*) is used to separate consecutive packets.

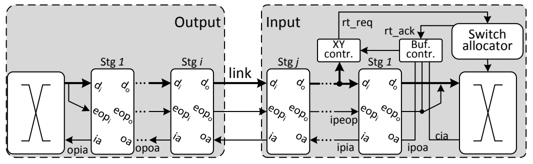
Fig. 7c shows the connection between two continuous routers. When a new packet arrives, its head flit is blocked before the first stage (*Stg* 1) by the Buffer controller, awaiting the XY controller to generate a routing request (*rt_req*) to the



(a) Router structure



(b) Flit sequence



(c) Connection between two routers

Fig. 7. An asynchronous wormhole router

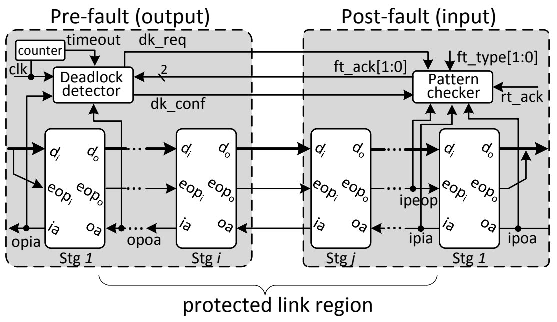


Fig. 8. Deadlock detection on the link of the NoC

switch allocator. After a free output is allocated to the input request, denoted by a positive *rt_ack*, the Buffer controller enables the transmission of the packet through the crossbar until a tail flit is noticed through the *eop* wire. Details about the router implementation can be found in [7], [13].

B. A time-out mechanism

The proposed techniques (Section II-D) are general and can be used in any 4-phase QDI pipelines to detect the deadlock and diagnose the fault type. In this design, they are used to protect links of the NoC (including pipelined input/output buffers and link wires) from deadlock. In [7], a time-out mechanism was proposed to detect the deadlock caused by permanent faults. This paper proposes an improved version which could detect the deadlock caused by both transient and permanent faults.

To detect the deadlocked link, extra circuits are added to the output and input of each pair of adjacent routers to monitor intermediate pipeline stages and link wires (Fig. 8). Faults can happen at any gates and wires between the leftmost (*Stg* 1 at the output) and the rightmost (*Stg* 1 at the input) stages of the link (which is the protected region). Since the common deadlock pattern shared by transient and permanent faults is valid only when the status of the pipeline is stable (deadlocked), a time-out mechanism can be used to control the deadlock detection. As Fig. 8 shows, a deadlock detector is added to each output of a router, which also controls the pattern checker at the input of the succeeding router. Each router has a counter to generate a *timeout* signal controlling a state machine in the deadlock detector, so that the detector is a sync/async hybrid circuit. If the monitored pipeline stages have been inactive for a long timeout period, the detector and pattern checker are assumed safe to sample asynchronous signals. Synchronizers are added to the sync/async interface to further reduce the possibility of metastability. If sampled asynchronous signals satisfy the deadlock pattern, the deadlock is detected and located.

Four flip-flops (*perm_fault*, *dk_conf*, *dk_req* and *start*) construct the state machine in the deadlock detector. The local clock signal used to drive the state machine and the counter can be easily got from the local synchronous IP cores. There is no requirement on its skew, jitter and frequency [7]. Fig. 9a presents the state graph, which has five states:

Idle: This is the default state after reset with all flip-flops being low.

Start: After a time-out period, *start* is set high, enabling the transition detector (Fig. 9b) to monitor the *ack* signal (*opoa*) at the output. Initially the transition detector is disabled and outputs '0'. When it is enabled to work (*start*+), any transitions of the monitored signal (*sig*) will set the output high. When the 2nd *timeout* comes, the deadlock detector either transits to **Enquiry** (no transitions are detected and two adjacent *ack* signals at the output (*opia* and *opoa* in Fig. 8) are complementary, satisfying the deadlock pattern at the pre-fault stage) or gets reset to **Idle** (transitions are detected or *opia*==*opoa*).

Enquiry: In this state, *dk_req* signal is set high by the deadlock detector to enquire the input of the succeeding router about the deadlock pattern. The pattern checker samples asynchronous signals from *Stg* 1 of the input buffer. Since the output of the preceding router has satisfied the deadlock pattern of the pre-fault stage, the link will be confirmed to be deadlocked if no transitions are detected during the 3rd timeout period and one of the following cases keeps true: ① the input buffer is transmitting a packet (a path connected to the output through the crossbar has been allocated, denoted by a high *rt_ack*) and the two consecutive *ack* signals connected to stage *Stg* 1 keep equal (*ipia*==*ipoa*) due to a fault; ② a fault may destroy the head flit so that the head of a packet is blocked before *Stg* 1 (!*rt_ack*). Consequently, both the two consecutive *ack* signals are low (!*ipia* and !*ipoa*); ③ an unallocated input buffer (!*rt_ack* and !*ipoa*) with a fake incoming tail flit (*ipeop*) may get blocked before *Stg* 1 of the input.

Using the fault type information collected from the fault diagnosis circuit at the 1st pipeline stage (*Stg* 1) of the post-fault router (Section III-C), the result from the pattern checker is encoded into a 1-of-2 signal (*ft_ack*[1:0]). If the deadlock pattern passes the check, *ft_ack* equals to either 2'b01 or 2'b10 indicating which kind of faults causes this deadlock (transient or permanent). The state machine will transit to **DK_Confirm** when the next *timeout* comes. If none of the above cases is matched or transitions are detected, the pattern checker outputs 2'b00, resetting the state machine to **Idle** immediately.

DK_Confirm: When the 3rd *timeout* arrives, a positive ($|ft\_ack$) will set *dk_conf* high to announce that a deadlock has been detected and the fault position is located (in the protected link region in Fig. 8). The deadlock must be caused by either a transient (*ft_ack*==2'b01) or a permanent fault (*ft_ack*==2'b10) in this paper. Then different recovery methods are provoked. If the deadlock is caused by a permanent fault, the state machine will get stuck at this state to block the defective link; Otherwise, the deadlock is caused by a transient fault and the state machine will go to **TF_Confirm**.

TF_Confirm: If the fault deadlocks the link is diagnosed as a transient one, the state machine goes to **TF_Confirm** to resume the previously blocked link to use when the 4th *timeout* comes. When the 5th *timeout* arrives, the state machine goes back to the initial **Idle** state and the detection process restarts.

It can be inferred that it needs two to four time-out periods to detect the deadlock (or the permanent fault), and four to six time-out periods to recover the link from the deadlock caused by a transient fault. This implemented NoC protects only the link (including the intermediate pipeline stages and link wires). The proposed methods can also protect the crossbar of the router if a pair of the deadlock detector and the pattern checker are put at the output and input of the same router so that the full data-path is protected.

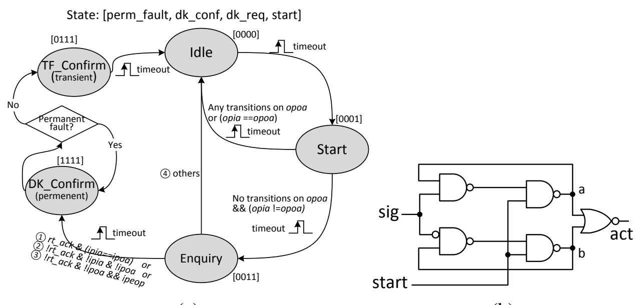

State: [perm_fault, dk_conf, dk_req, start]
Idle
[0000]
timeout
TF_Confirm (transient)
[0111]
timeout
No
Permanent fault?
Yes
[1111]
DK_Confirm (permenent)
Any transitions on opoa or (opia ==opoa)
timeout
[0001]
Start
④ others
No transitions on opoa && (opia !=opoa)
timeout
timeout
① rt_ack & (ipia==ipoa) or
② !rt_ack & !ipia & !ipoa or
③ !rt_ack & !ipoa && ipeop
Enquiry
[0011]
(a)
sig
start
a
b
act
(b)

Fig. 9. (a) State machine (b)Transition detector [14]

C. Fault diagnosis

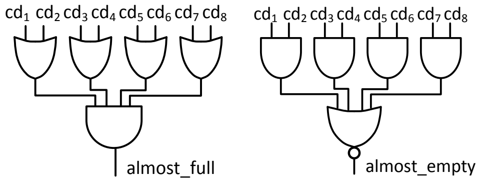
When the deadlock is detected and located (**DK_Confirm**), fault diagnosis is required to determine the fault type and then relevant methods are used to recover the network. According to Section II-C and II-D, we know that different faults cause different patterns at the post-fault pipeline stage (*Stg* 1 of the input in Fig. 8) which are summarized in Fig. 6. The key operation to diagnose fault type is to differentiate the almost_full/complete_data from almost_empty/spacer symbols.

Fault diagnosis circuit is put at the CD of the post-fault pipeline stage (*Stg* 1 at the input of the post-fault router, Fig. 8). Its input comes from the CD of each sub-pipeline stage (Fig. 2). Therefore, if the pipeline is N-word wide, the fault diagnosis circuit has N inputs. Assume that one 1-of-n pipeline is *N*-word wide (*N*>2) and the set of completion detection signals from all CDs of the post-fault pipeline stage is S_{CD}=$\{cd_1, ...cd_N\}$. *Num*(1) denotes the number of '1's in S_{CD} and *Num*(0) denotes the number of '0's. Under the assumption of 1-bit fault, only one "*cd*" in the S_{CD} could be different from the others at most. It can be inferred that:

- If *Num*(1)>*Num*(0), the latched data symbol is an almost_full or complete_data one;
- If *Num*(1)<*Num*(0), the latched data symbol is an almost_empty one or a spacer;
- *Num*(1) and *Num*(0) cannot be equal.

If N==1, the deadlock can be caused only by a permanent fault according to Theorem 1 in Section II-B. If N==2, the symbol type can be decided by adding one more redundant word (so that N==3) with which symbol corruption caused by transient faults can be tolerated [9].

Fig. 10 shows the implementation of fault diagnosis circuits for an 8-word wide pipeline, with which we can compare the value of *Num*(1) and *Num*(0). Fig. 10a shows the circuit that decides if the latched data by the pipeline stage is an almost_full/complete_data symbol or not. Since at most one "*cd*" in S_{CD} could be affected and different from the others under the assumption of 1-bit fault, if two or more '1's are detected by the diagnosis circuit (leading to a positive *almost_full*), it can be assured that *Num*(1)>*Num*(0) and the latched data is an almost_full or complete_data symbol.


cd$_1$ cd$_2$ cd$_3$ cd$_4$ cd$_5$ cd$_6$ cd$_7$ cd$_8$
almost_full
cd$_1$ cd$_2$ cd$_3$ cd$_4$ cd$_5$ cd$_6$ cd$_7$ cd$_8$
almost_empty

(a) Almost_full/complete (b) Almost_empty/spacer

Fig. 10. Fault diagnosis circuits for an 8-word wide pipeline

Similarly, Fig. 10b shows the circuit deciding if the latched data is an almost_empty/spacer symbol or not. If two or more "*cd*"s are '0's, it can be assured that *Num*(0)>*Num*(1). The output *almost_empty* goes high and the latched data is an almost_empty symbol or a spacer.

Fault type is denoted by a 1-of-2 signal *ft_type*[1:0] whose default value is 2'b00. When a deadlock is detected, it should be either 2'b01 (a transient fault) or 2'b10 (a permanent fault), while 2'b11 is invalid (it is assumed that the transient and permanent fault cannot happen and deadlock the same pipeline *segment* at the same time). In a deadlocked state, the output of the diagnosis circuit *ft_type* has the same value as the output of the pattern checker *ft_ack*. Therefore, we have (17) corresponding to Fig. 6 (*ack* corresponds to the *ipoa* in Fig. 8).

$$\begin{aligned}ft\_type =\{&(almost\_full \ \&\ !ack) \mid (almost\_empty \ \&\ ack),\\ &((almost\_full \ \&\ ack) \mid almost\_empty \ \&\ !ack)\}\end{aligned} \tag{17}$$

For example, if S_{CD}={00010000} where cd_4 from the 4th sub-pipeline is affected by the fault, using the diagnosis circuit we have *almost_full*==0 and *almost_empty*==1. The fault type can be decided using (17). The proposed diagnosis circuit can be extended to the case of multi-bit faults.

D. A fine-grained network recovery

If the deadlock is caused by a permanent fault, the defective component should be blocked so that the following traffic could detour around the defective one and the network function is recovered. This has been fully studied where the Spatial-division Multiplexing (SDM) is used to implement the NoC [7]. As a result, each link is divided into multiple independent sub-links physically. If one sub-link is defective, the other fault-free ones can still be used to transfer packets with a slightly decreased performance. The defective sub-link can be blocked by configuring the switch allocator of the pre-fault router. The remaining flits in the fault-free deadlocked pipeline stages upstream of the fault can be drained out at the output of the pre-fault router (*Drain* operation) while the deadlocked pipeline stages downstream of the defective link are released by creating a fake tail flit at the input of the post-fault router (*Release* operation). Details can be found in [7].

Differently, the link deadlocked by a transient fault is not defective which should be reused again. A fine-grained recovery mechanism is proposed to avoid an expensive system reboot. When the state machine goes to **DK_Confirm** where the deadlock has been detected and the fault type is known, the fault deadlocking the link is regarded as permanent in default. All recovery operations dealing with the permanent faulty link (including *Drain*& *Release* and the blockage of the link) are executed. It is reasonable to assume that they can finish in one long time-out period. If the fault is transient, the state machine will transits to **TF_Confirm** where the blocked link is resumed to use. Otherwise, the fault is permanent (**DK_Confirm**) and the defective sub-link is blocked forever.

TABLE I. HARDWARE EVALUATION OF ASYNCHRONOUS NOCS

	Original	Protected	Overhead
Area (μm^2)	63446	72394	14.1%
Throughput (*MByte/s/node*)	693	648	-6.5%
Energy (*pJ/Byte*)	3.7	4.3	16 %

IV. Experimental Results

A. Performance evaluation

The asynchronous NoC router using the proposed deadlock management techniques are implemented using the 130nm standard cell library. Asynchronous cells, such as C-elements and MUTEX, are built using standard cells. The router has five input/output ports and each port is 32-bit wide. Using Spatial-division Multiplexing (SDM) [7], [13], each link is divided into two sub-links physically to support the recovery from permanent faults. Both the input and output buffers have two pipeline stages. 4-phase 1-of-4 protocol is used to implement the whole network. The original asynchronous SDM NoC using the same configuration is also designed for comparison. TABLE I presents the experimental results. It can be found that, after adding extra circuits dealing with the deadlock, the area of the protected router increases 14.1%.

A SystemC/Verilog mixed environment is built to evaluate the network performance. 16 post-synthesis routers (annotated with the gate latency) are connected to build a 4×4 2D-mesh network. Mounted synchronous processors and network interfaces are implemented in SystemC. The packet size is fixed to 64 bytes while the flit width equals to the width of the sub-link (2 bytes). Using the maximum rate, processors inject random packets into the network so that the network traffic is uniformly distributed. The clock and time-out frequencies are set to 100*MHz* and 1.5*MHz* respectively. It can be found from TABLE I that the saturation throughput of the protected NoC decreases by 6.5% compared to the unprotected one while the energy of the protected router increases 16%.

B. Reliability analyses

Transient faults may cause symbol corruption, symbol insertion and deadlock on an asynchronous NoC. This paper targets on the deadlock occasion. The other faulty behaviour can be tolerated using fault-tolerant codes [9] or physical redundancy techniques [8], which is not the focus of this paper.

To verify the proposed deadlock recovery theorems, random faults are inserted to the built network during the simulation using scripts written in Tool Command Language. Faults could happen at pipelined buffers (including the asynchronous latch and CD) and link wires. It is straightforward that a permanent fault could cause deadlock on the asynchronous

NoC. For deadlock caused by a transient fault, data skew has to be inserted accompanied with the fault. For instance, along with a (positive/negative) transient fault inserted to $A_{i+1,1}$, a delay has to be inserted to $A_{i,1}$ to block its normal (positive/negative) transition to produce a deadlock on this *segment* (Fig. 2b). The fault level should be the same as the level of the delayed signal. Following this rule, deadlocks caused by transient faults are randomly generated through the network. Test results show that the fault type can be precisely diagnosed according to the deadlock pattern. The deadlock caused by transient or permanent faults on *data* or *ack* wires of links can be 100% detected and recovered.

It should be noticed that the deadlock recovery techniques bring extra circuits to original routers, which are not protected in this paper. They increase the router area by 14.1% which has a negative impact on the chip reliability. Given the fact that most runtime permanent faults are strongly related to the workload of the device [15], it is reasonable to assume that the overall reliability against permanent faults largely increases since there are far fewer activities on these extra circuits compared with the usual router logic, especially when a long time-out period is used. However, transient faults could happen randomly on the network. Considering our design, the state machine in the deadlock detector is critical and should be safe. Most of transient faults happening on detection circuits could be masked [6]. A transient fault may cause a wrong deadlock indication, provoking the recovery process. This could result data errors which can be tolerated using other techniques [8], [9]. The state machine would keep running under the control of the clock and finally goes back to the initial state. One possible risk is that a transient-fault-caused deadlock is mistaken into a permanent one due to a transient fault on the fault type signal. Under the assumption of 1-bit fault during a detection cycle, this cannot happen. Since the deadlock caused by a transient fault can paralyse the function of the whole network, using reasonable redundancy for deadlock detection and recovery could avoid expensive system reboot and make the network more reliable, especially when both transient and permanent faults are considered. Some other techniques such as scan chains [16] can be used to monitor and protect the behaviour of these redundant circuits periodically. In addition, a fault or the deadlock recovery operation can destroy a transmitting packet, resulting packet loss. Extra retransmission mechanisms should be employed to redeliver the lost packet. These are left as the future work.

V. Conclusion

A transient fault happening on a QDI pipeline with a delayed signal transition could cause deadlock, which has not been fully studied. This deadlock could spread over the whole asynchronous NoC and paralyse its function. By modelling the 4-phase QDI pipeline, this paper systematically studies the formation and behaviour of the deadlock caused by a transient fault. Common deadlock patterns caused by both the transient and permanent faults are summarized which can be used to detect the deadlock and diagnose the fault type. The proposed theorems apply to all 4-phase 1-of-n QDI pipelines and can be extended to m-of-n pipelines (m$\geq$2). As a design case, this paper implements an asynchronous NoC whose links (including the pipelined buffers and link wires) are protected. The deadlock caused by a transient or permanent fault on the link could be detected and diagnosed precisely. A fine-grained recovery mechanism is proposed to avoid the expensive system reboot and recover the network. Detailed experimental results show that the overhead brought by the redundant circuits are reasonable. The proposed deadlock recovery technique can be easily extended to protect the whole data-path of the NoC (including the crossbar).

This paper studies the deadlock which could cause serious consequence on the network. The deadlock does not happen so often as the symbol corruption (symbol insertion requires data skew as well) since only a transient fault accompanied with a long enough data skew could cause this deadlock. The future work is to combine the proposed deadlock recovery methods with fault-tolerant codes [9] or other redundancy techniques [8] to tolerate symbol corruption (or insertion) and extend the protection from the data-path to the control-path, so that a systematic protection of the NoC could be achieved.

Acknowledgment

The authors would like to thank various grants from the National Natural Science Foundation of China (61272144, 61402497 and 61402501), 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] J. Muttersbach, T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner, "Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems," in *Proc. of International ASIC/SOC Conference*, 1999, pp. 317–321.

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

[5] K. L. Shepard and V. Narayanan, "Noise in deep submicron digital design," in *Proc. of ICCAD*, 1996, pp. 524–531.

[6] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Tran. on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[7] G. Zhang, W. Song, J. Garside, J. Navaridas, and Z. Wang, "An asynchronous SDM network-on-chip tolerating permanent faults," in *Proc. of ASYNC*, May 2014, pp. 9–16.

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

[9] G. Zhang, W. Song, J. Garside, J. Navaridas, and Z. Wang, "Protecting QDI interconnects from transient faults using delay-insensitive redundant check codes," *Microprocessors and Microsystems*, vol. 38, no. 8, Part A, pp. 826 – 842, 2014.

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

[11] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch, "Methods for fault tolerance in Networks-on-chip," *ACM Computing Surveys*, vol. 46, no. 1, pp. 8:1–8:38, Jul. 2013.

[12] A. J. Martin, *Synthesis of asynchronous VLSI circuits*. No. CALTECH-CS-TR-93-28, California Institute of Technology, 1991.

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

[14] 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.

[15] 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.

[16] X.-T. Tran, Y. Thonnart, J. Durupt, V. Beroulle, and C. Robach, "Design-for-test approach of an asynchronous network-on-chip architecture and its associated test pattern generation and application," *IET Computers Digital Techniques*, vol. 3, no. 5, pp. 487–500, 2009.