Microprocessors and Microsystems 38 (2014) 826–842

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Protecting QDI interconnects from transient faults using delay-insensitive redundant check codes

Guangda Zhang [a,*], Wei Song [a], Jim Garside [a], Javier Navaridas [a], Zhiying Wang [b]

[a] School of Computer Science, University of Manchester, Manchester M13 9PL, UK
[b] School of Computer, National University of Defense Technology, Changsha 410073, China

ARTICLE INFO

Article history:
Available online 18 April 2014

Keywords:
Asynchronous circuits
Quasi-delay-insensitive (QDI) interconnects
Fault tolerance
Transient fault
1-of-n
Error-correcting code

ABSTRACT

Asynchronous circuit design is a promising technology for large-scale multi-core systems. As a family of asynchronous circuits, Quasi-delay-insensitive (QDI) circuits have been widely used to build chip-level long interconnects due to their tolerance to delay variations. However, QDI interconnects are vulnerable to faults. Traditional fault-tolerant techniques for synchronous circuits cannot be easily used to protect QDI interconnects. This paper focuses on protecting QDI interconnects from transient faults. The first contribution of this paper is a fault-tolerant delay-insensitive redundant check code named DIRC. Using DIRC in 4-phase 1-of-n QDI pipelines, all 1-bit and some multi-bit transient faults are tolerated. The DIRC and basic pipeline stages are mutually exchangeable. Arbitrary basic stages can be replaced by DIRC ones to strengthen fault-tolerance. This feature permits designers to use DIRC flexibly according to the practical design requirement. The second contribution is a redundant technique protecting the acknowledge wires (RPA). Experimental results indicate that DIRC pipelines have moderate area and speed overheads. Compared with unprotected basic pipelines, the average speed decrease of DIRC pipelines is less than 50%, with the 128-bit 1-of-2 DIRC pipeline only 28% slower. In severe environments with multi-bit transient faults, the fault-tolerance capability of DIRC pipelines increases thousands-fold.

1. Introduction

The advancing semiconductor manufacturing technology continuously increases the number of transistors and wires integrated in a single chip, bringing a multi-core or many-core era. However, the reduced transistor dimensions, the increase of clock frequency and the decrease of critical charge intensify the frailty of electronic devices to environmental variations, which consequently accelerates the occurrence of transient faults [1,2]. The vulnerability of circuits to faults is aggravated [2]. For these reasons, fault-tolerance becomes an essential design objective for critical digital systems, especially those in highly specialized fields such as aerospace, military and medical equipment.

Coming along with the multi-core era, it is increasingly difficult to distribute a high-speed clock signal over a very-large-scale integration circuit with acceptable clock skews, which is a challenge for traditional synchronous designs. Asynchronous circuits provide a fundamental solution to problems caused by clocks [3] due to their clockless nature. They also present potential advantages in power consumption, modularity, composability and robustness, which has attracted many researches in recent years [3,4].

Networks-on-chip (NoCs) [5] are the state-of-the-art infrastructures for scalable and efficient on-chip communications [4]. Implementing a NoC using asynchronous circuit has the potential of simplifying timing closure, reducing power and resolving chip-level synchronization issues. Consequently, asynchronous NoCs are thought to be better candidates than their synchronous counterparts for current and future multi-core systems [4,6–8].

As an important issue in the design of asynchronous NoCs, the large number of long wires, which are exposed to environmental noises and faults, may lead to delay variations and glitches [9]. Quasi-delay-insensitive (QDI) circuits [3] are a family of asynchronous circuits that tolerate all delay variations but they are vulnerable to glitches. Occasionally, an erroneous transition may be accepted as a valid signal, producing a fault in QDI circuits.

This paper proposes a new coding scheme that significantly improves the tolerance of 4-phase 1-of-n QDI interconnects to transient faults, namely the delay-insensitive redundant check (DIRC) code [10]. The DIRC coding scheme tolerates all 1-bit transient faults and some multi-bit transient faults. Furthermore, it can be easily adopted in all existing 1-of-n QDI pipelines to provide

* Corresponding author.
E-mail addresses: zhangga@cs.man.ac.uk, zhanggd.nudt@gmail.com (G. Zhang).

http://dx.doi.org/10.1016/j.micpro.2014.04.002
0141-9331/

fault-tolerance. Since the DIRC code is systematic, it allows the DIRC stages to be placed arbitrarily in an unprotected QDI pipeline, thus some certain pipeline segments can be protected. According to the practical fault-tolerance requirement, the DIRC pipeline using different construction patterns can provide enough fault-tolerance for the communication infrastructure with a moderate and reasonable hardware overhead, making DIRC especially attractive to large-scale communication-centered fabrics such as NoCs and buses.

Another fault-tolerant technique, RPA (redundant protection of acknowledge wires), is proposed to protect the acknowledge wires from 1-bit transient faults. It has extremely low area overhead and can be used independently of DIRC.

The QDI pipeline using DIRC and RPA provides both timing-robustness and fault-tolerance. A 128-bit DIRC 1-of-2 pipeline is only 28% slower than an unprotected basic pipeline, while the fault-tolerance capability increases thousands-fold.

This paper is organized as follows: Section 2 introduces some necessary background of asynchronous circuits and the impact of transient faults on QDI interconnects. Section 3 presents related works. Section 4 describes the DIRC coding scheme and Section 5 demonstrates the hardware implementation of the DIRC pipeline. Section 6 evaluates the hardware overhead of DIRC pipelines and studies different DIRC construction patterns. Section 7 reveals detailed experimental results of DIRC pipelines. Finally this paper is concluded in Section 8.

2. QDI pipelines and the impact of transient faults

2.1. Asynchronous protocols and 4-phase QDI pipelines

Asynchronous communication is controlled by handshake protocols, including 4-phase and 2-phase ones [3]. 4-phase protocols are level-triggered protocols which require a reset (or return-to-zero) phase (Fig. 1a). 2-phase protocols are edge-triggered where each transition of the acknowledge signal starts a new data transmission (Fig. 1b). In a 4-phase pipeline, valid data are separated by spacers while in a 2-phase pipeline, data symbols are transmitted consecutively. In general, 4-phase protocols are more widely used than 2-phase ones for their implementation simplicity.

QDI circuits assume that both the delays of gates and wires are arbitrary, positive and bounded. No delay assumption other than the isochronic-forks [3] is used. As a result, QDI circuits are robust to delay variations. For these requirements, QDI circuits use delay-insensitive (DI) codes (or unordered codes [11–13]) to encode data, whose arrival can be detected using a completion detector (CD). The 1-of-n code is one of the most utilized DI codes due to its simplicity [4,6–8]. As the two extensively used 1-of-n codes, 1-of-2 and 1-of-4 codes are presented in Fig. 2. This paper focuses on improving the fault-tolerance of 4-phase QDI interconnects using 1-of-n codes.

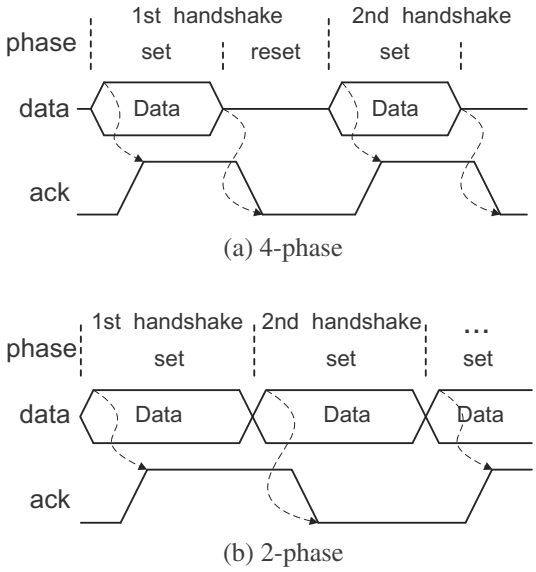


Fig. 1. Handshake protocols.

Fig. 3 shows an unprotected 4-phase 1-of-n QDI pipeline, which contains *S* parallel sub-pipelines. Each sub-pipeline contains *n* data wires transmitting one 1-of-n code. To latch incoming data, each pipeline stage is divided into *S* slices, each of which contains *n* C-elements acting as asynchronous latches. Acting as a sub-CD, an OR-gate notifies the arrival of a 1-of-n code. To synchronize the data transmission of all sub-pipelines, the CDs of all slices are connected to a multi-input C-element producing a common acknowledge signal (*ack*), whose positive transition denotes a full data word is latched. The inverse of this common acknowledge signal (*iack*) is then used to drive the previous stage. The OR-gates and the multi-input C-element are usually regarded as the CD of a pipeline stage. On long interconnects, buffers are often inserted to increase signal strength.

2.2. Impact of transient faults on QDI pipelines

Transient faults, including positive ('0' → '1' → '0') and negative ('1' → '0' → '1') ones, can be provoked by noise [14], electromagnetic interference, electrostatic discharge [2] or radiation [15]. A transient fault usually lasts for a short period and causes an error when it is captured by a memory component (such as a C-element). The typical phenomenon of a transient fault is a bit-flip, also known as a glitch.

This paper concentrates on protecting the long wires between two pipeline stages from 1-bit transient faults (Fig. 3). Multi-bit faults rarely occur although they result in more severe damage than 1-bit faults.

In the presence of transient faults, QDI pipelines behave differently from synchronous ones due to their clockless nature. A fundamental difference between QDI and synchronous circuits is the timing reference of data words. In synchronous circuits, the clock signal acts as a timing reference controlling the data transmission. Each bit of a data word typically has a constant latency requirement which can be agreed between the transmitter and the receiver. Faults are therefore only able to corrupt the value of some data words. QDI circuits have no such timing reference. Data-validity is implicitly encoded within the data word, which controls its own transmission. As a result, the corrupted data words caused by faults can disorder the data transmission. In other words, faults on QDI pipelines may produce fake data-validity or remove the correct data-validity, resulting in erroneous data insertions or data removals. Conventional fault-tolerant techniques used in synchronous circuits cannot be easily adopted in QDI pipelines.

A fault model is built to analyse the impact of 1-bit transient fault. Fig. 4 shows two adjacent pipeline slices. The fault may occur on any of the *n* data wires. To identify the position of a fault, the

1-of-2 codes	Binary
00	*Spacer*
01	0
10	1

(a) 1-of-2codes

1-of-4 codes	Binary
0000	*Spacer*
0001	00
0010	01
0100	10
1000	11

(b) 1-of-4codes

Fig. 2. 1-of-n codes.

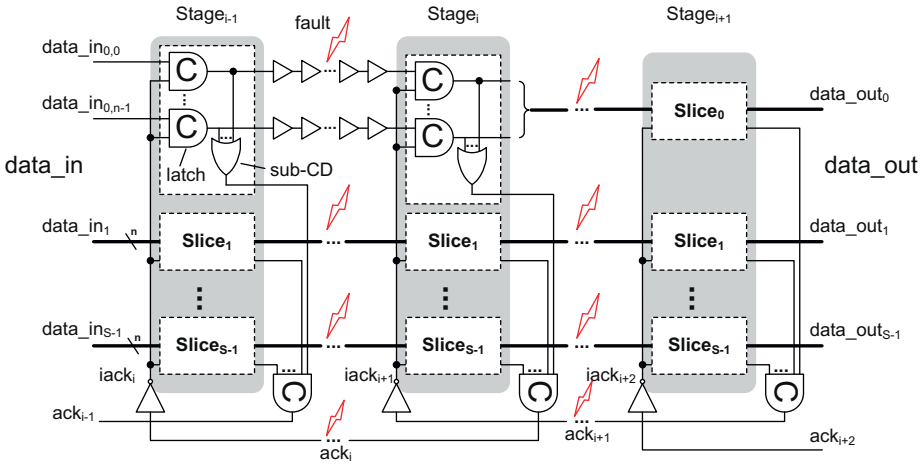


Fig. 3. A 4-phase 1-of-n QDI asynchronous pipeline.

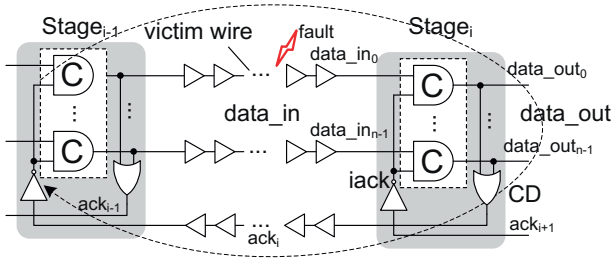


Fig. 4. Fault happens on a sub-pipeline for one 1-of-n code.

wire that should carry the correct '1' is defined as the "active" wire while the others are "inactive" wires (Fig. 5). The wire affected by a fault is denoted as the "victim" wire (Fig. 4).

Not every transient fault causes an error. Some transient faults are masked automatically during their propagation by intrinsic masking factors of the circuit [15]. For the asynchronous latch built from C-elements (Fig. 6), when *iack* is low, positive faults at the input are masked. When *iack* is high, negative faults are masked. However, if the transition of a fault is the same as the level of *iack*, the fault may be latched, resulting in a transient error. The key issue of fault-tolerant QDI pipelines is to prevent faults from being incorrectly latched. To analyse all the fault occasions of a sub-pipeline stage, the proposed fault model divides one handshake period into two intervals (*iack*+ and *iack*−).

2.2.1. Fault with a positive iack

When *iack* is high, the pipeline is susceptible to positive faults. Both active and inactive wires can be the victim, leading to an invalid 1-of-n code insertion or 2-of-n code insertion.

1. *Invalid 1-of-n code insertion*: A positive fault on an inactive wire may be latched and outputted before the arrival of the next valid code, producing an invalid 1-of-n code insertion and an early drop of *iack*. As shown in Fig. 7a, if the valid code comes so late that a full reset phase of the invalid code has finished, the invalid code is inserted into the code sequence. Otherwise, if the valid code overlaps with the reset phase (*iack*−) caused by the invalid code (Fig. 7b), it may be erased as the previous stage is mistakenly reset by the wrong *iack*.
   A positive fault on the active wire may also lead to cases of Fig. 7a and b when the fault makes *iack* go low earlier than the arrival of the valid data. The difference is the inserted 1-of-n code by the fault is *valid*. Therefore, the situation of Fig. 7b will not make errors while the situation of Fig. 7a will output one valid code twice, which can be taken as an invalid spacer insertion.
2. *Invalid 2-of-n code insertion*: As shown in Fig. 7c, a positive fault on one of the inactive wires is latched and outputted, causing an overlap period between the invalid '1' (fault) and the valid '1' (original), resulting in an invalid 2-of-n code at the output.

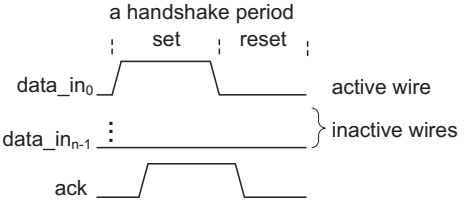
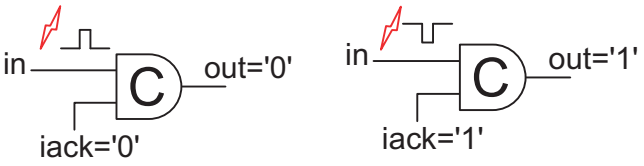


Fig. 5. Active and inactive wires.



(a) Masked positive fault (b) Masked negative fault

Fig. 6. Faults masked by C-elements.

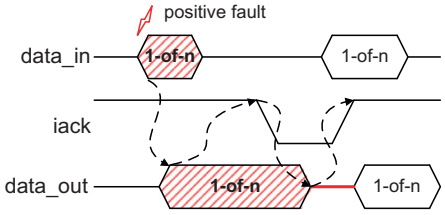
2.2.2. Fault with a negative iack

When *iack* is low, all positive faults are masked and the pipeline is only vulnerable to negative faults happening on the active wire.

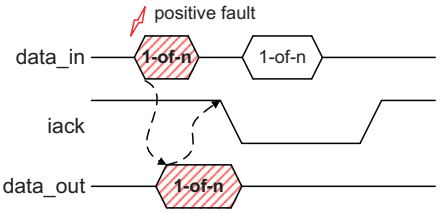
1. *Early spacer* (not errors): A negative fault happening on the active wire may cause an early spacer (Fig. 8a), which is tolerated by QDI pipelines.
2. *Invalid 1-of-n code insertion*: A negative fault may lead to a premature reset phase. If *iack* already goes high when the transient fault elapses, as shown in Fig. 8b, the original valid code will be outputted twice, leading to an invalid code insertion.

2.2.3. Deadlocks caused by transient faults

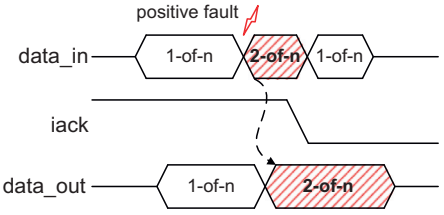
Invalid code insertions may cause deadlocks when considering a full pipeline with multiple sub-pipelines. As an example, Fig. 9a depicts the correct operation of two adjacent pipeline stages (*Stage*$_{i-1}$ and *Stage*$_i$), each of which contains two 1-of-n slices. d_0



(a) Invalid 1-of-n code insertion (valid code is outputted later)

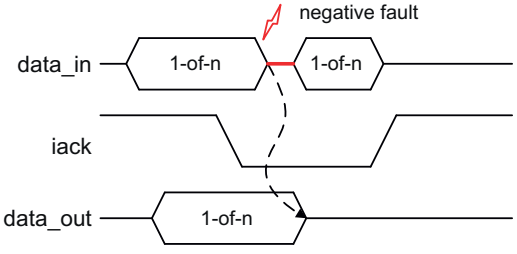


(b) Invalid 1-of-n code insertion (valid code is erased)

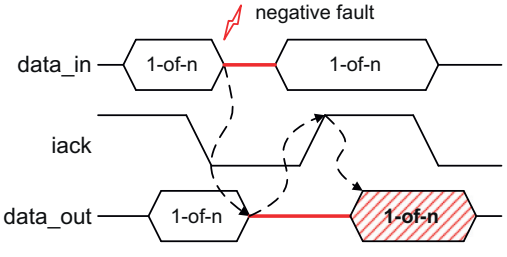


(c) Invalid 2-of-n code insertion

Fig. 7. Transient faults happen when *iack* is high.



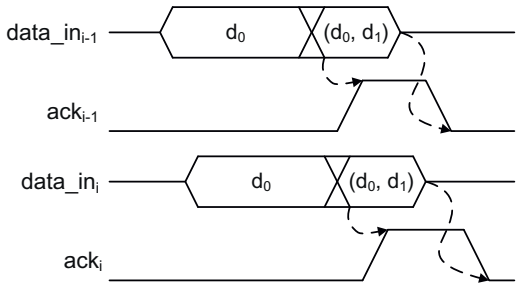
(a) Early spacer



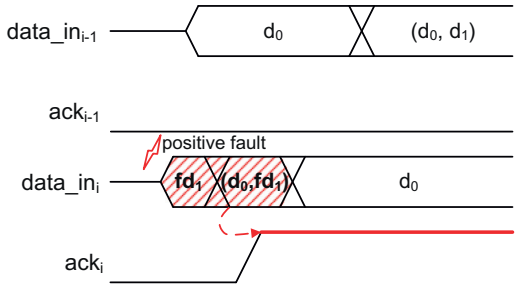
(b) Invalid 1-of-n code insertion

Fig. 8. Transient faults happen when *iack* is low.

and d_1 are two 1-of-n codes. A positive fault at the input of $Stage_i$ creates a false 1-of-n code fd_1 on $data\_in_i$ which is mistakenly latched and sets the acknowledge signal ack_i high (Fig. 9a). Consequently, the valid d_1 which arrives late at $Stage_{i-1}$ cannot be latched due to the high ack_i. $Stage_{i-1}$ keeps waiting for a low ack_i to latch d_1 and then the data can be removed from $data\_in_i$ in the next reset



(a) Correct situation



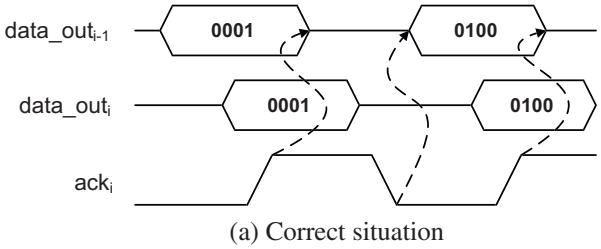
(b) Deadlocked situation

Fig. 9. Deadlock caused by transient faults on data wires.

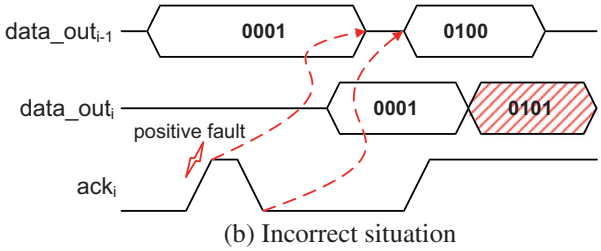
phase while $Stage_i$ keeps waiting for the removal of $data\_in_i$, leading to a deadlock.

2.2.4. Transient faults on acknowledge wires

Fault-tolerance of acknowledge wires is a serious issue which has been ignored by many existing fault-tolerant designs. Faulty acknowledge signals may lead to faulty situations similar to the above ones. An example is shown in Fig. 10 where $data\_out_{i-1}$ and $data\_out_i$ are outputs of $Stage_{i-1}$ and $Stage_i$ respectively. ack_i is the acknowledge signal of $Stage_i$. Fig. 10a shows the correct situation of these signals. Fig. 10b shows the faulty situation where a positive fault happens on ack_i indicating that a valid code has been latched by $Stage_i$, causing $Stage_{i-1}$ to reset. When the fault disappears, $Stage_{i-1}$ starts a new transmission of "0100", which may be latched with the first data word, leading to an invalid 2-of-n code ("0101") insertion.



(a) Correct situation



(b) Incorrect situation

Fig. 10. Transient faults on acknowledge wires.

2.3. Summary

In summary, a 1-bit transient fault on QDI pipelines may cause an invalid 1-of-n code insertion or an invalid 2-of-n code insertion, which may even lead to a deadlock, resulting in a failure of the pipeline. Acknowledge signals are equally critical to the handshake process. Faults on acknowledge wires also cause various errors. This paper tries to use separate methods to protect both data wires and acknowledge wires from transient faults.

3. Background and related work

Fault-tolerance can be achieved by redundancy, including spatial, temporal and information redundancy. These fault-tolerant techniques can be classified into two categories: *code redundancy* and *circuit redundancy* techniques. Code redundancy uses error-detecting codes or error-correcting codes to detect or correct errors. The others belong to circuit redundancy using special hardware designs to achieve fault-tolerance.

3.1. Systematic codes

Using code redundancy, the original data words are encoded into systematic or non-systematic codes [13,16]. A non-systematic code has only one inseparable field so that extra decoders are required to extract the original data.

A systematic code, on the other hand, comprises of an information field and a check field. The information field contains the original data while the check field is a redundant check word used to correct errors. One of the benefits of using systematic codes is that the original data words are untouched during their transmission so that no extra data extraction is needed. This feature allows generating check words and correcting errors at discontinuous pipeline stages, the pipeline segment between which is protected. Compared with the pipeline using non-systematic codes where the check word generation and error-correction operations are usually executed at adjacent stages (including the data extraction), the pipeline using systematic codes can be constructed in a more flexible way to protect some specific pipeline segments according to the practical fault-tolerance requirement. As a result, the area overhead can be largely reduced (Section 6.2).

3.2. Traditional fault-tolerance methods

Belonging to systematic codes, parity check codes, cyclic redundancy check codes [17] and Hamming codes [18] have been widely used for fault-tolerance purposes. However, using them along with DI codes, the generated check word and the original data must be DI as well, which is a big challenge. Most existing asynchronous designs [19,20] that use these traditional error-detecting or error-correcting codes are (partly) bundled-data designs rather than QDI, which are similar to synchronous circuits.

Triple Modular Redundancy (TMR) is another well-known technique. Using TMR, all components (usually including the voter [21]) are tripled, making the circuit more than three times larger. A synchronization among the three replicas is also required, which compromises the speed. TMR is expensive and often used in special fields with an extremely high requirement on fault-tolerance.

3.3. Non-QDI designs

For code redundancy techniques, an unordered systematic code [22] using parity check codes can correct 1-bit error and detect completion simultaneously on 4-phase asynchronous links. Using several fault-tolerant methods [19], a self-timed bundled-data link is able to tolerate transient and permanent errors but introduces large area overhead due to the extra de-interleaving and Hamming decoding processes.

Another unordered systematic code named Zero-Sum [13,23] provides both 1-bit error correction and 2-bit error detection for asynchronous links. Its extensions [13] provide more fault-tolerance with the price of a larger hardware overhead. A time-out mechanism is required to constrain the bit arrival intervals.

For circuit redundancy technique, Ogg et al. [24] proposed a technique utilizing the phase relationships between data symbols and a redundant reference symbol to achieve transient-fault-tolerance. An asynchronous interfacing scheme using parity check codes has been proposed for globally asynchronous locally synchronous (GALS) systems [20]. TMR has been used in a GALS system to achieve fault-tolerance [25].

Although these techniques seem promising, none of them can be easily used in QDI interconnects. This paper focuses on providing fault-tolerance for QDI interconnects with code redundancy while keeping the timing-robust nature.

3.4. QDI designs

As for QDI designs, Bainbridge and Salisbury [9] proposed a series of fault-tolerant techniques which increase the robustness of the circuits but not enough for fault-immunity. Jang and Martin [21] proposed a duplicated double-checking technique which tolerates 1-bit faults and some multi-bit faults. The fault-tolerant buffer using this technique is three times larger and runs twice slower than the normal one. Some circuit redundancy techniques [26,27] can also be used in QDI circuits, but their duplication is area-consuming.

In addition, Pontes et al. [28,29] used temporal redundancy to tolerate transient faults on 1-of-n QDI pipelines. Peng and Manohar [30] proposed a failure-detection technique for pipelined QDI circuits which can achieve fail-stop with respect to permanent and transient errors. Kuang et al. [31] studied the fault-tolerance of Null convention logic QDI circuits.

4. Delay insensitive redundant check coding scheme

4.1. Arithmetic rules of m-of-n codes

Before presenting the fault-tolerant coding scheme, some arithmetic rules of m-of-n ($1 \leqslant m < n$) codes are defined.

Let i be an integer less than n ($0 \leqslant i < n$); its 1-of-n code representation is an n-bit vector $D^n(i)$ where the $(i+1)$ th bit is high (e.g. $D^4(2)$ = "0100"). The basic arithmetic rule of 1-of-n codes is defined as (1)–(3).

For two integers a and b,

$$D^n(a) = D^n(a \bmod n) \tag{1}$$

$$-D^n(a) = D^n(-a) = D^n(-a \bmod n) = D^n(n-a) \tag{2}$$

$$D^n(a) + D^n(b) = D^n((a+b) \bmod n) \tag{3}$$

Transient faults may mutate a 1-of-n code into m-of-n ($m \geqslant 2$). To extend the arithmetic rules of 1-of-n codes to m-of-n codes, a position set $A^m = (a_0, a_1, \ldots, a_{m-1})$ ($1 \leqslant m \leqslant n$) is defined to identify the positions of the m '1's in an m-of-n code. Let $D^n(A^m) = D^n(a_0, a_1, \ldots, a_{m-1})$ be an m-of-n code. A union operation is used in (4) to construct an m-of-n code with m 1-of-n codes, which is denoted by the symbol "$\bigcup$".

$$\begin{aligned} D^n(A^m) = D^n(a_0, a_1, \ldots, a_{m-1}) &= \bigcup_{i=0}^{m-1} D^n(a_i) \\ &= D^n(a_0) \cup D^n(a_1) \cup \ldots \cup D^n(a_{m-1}) \end{aligned} \tag{4}$$

The arithmetic rules of m-of-n codes are shown in (5) and (6), which are unions of multiple 1-of-n operations.

$$-D^n(A^m) = -\bigcup_{i=0}^{m-1} D^n(a_i) = \bigcup_{i=0}^{m-1} [-D^n(a_i)] = \bigcup_{i=0}^{m-1} D^n(-a_i) \tag{5}$$

$$D^n(A^m) + D^n(B^{m'}) = \bigcup_{i=0}^{m-1} D^n(a_i) + \bigcup_{j=0}^{m'-1} D^n(b_j) = \bigcup_{j=0}^{m'-1} \bigcup_{i=0}^{m-1} D^n(a_i + b_j) \tag{6}$$

Taking a 1-of-4 code $D^4(3)$ and a 2-of-4 code $D^4(3,2)$ for example, we have:

$$-D^4(3,2) = -[D^4(3) \cup D^4(2)] = D^4(-3) \cup D^4(-2) = 0010 \cup 0100 = 0110,$$

$$D^4(3) + D^4(3,2) = [D^4(3) + D^4(3)] \cup [D^4(3) + D^4(2)] = D^4(2) \cup D^4(1) = 0110.$$

4.2. DIRC coding scheme

The proposed DIRC coding scheme is based on the arithmetic rules of m-of-n codes. Its definition is given below:

Definition 4.1 (*DIRC Code*). Let $X = (x_0, x_1, \ldots, x_{CN-1})$ $(CN \geqslant 2)$ be a data vector containing CN 1-of-n codes. A delay-insensitive redundant check word c can be generated from X using a check word generating process. Each x_i, as well as the check word c, is 1-of-n. Together they comprise a DIRC code word $(x_0, x_1, \ldots, x_{CN-1}, c)$ containing $(CN+1)$ 1-of-n codes.

It can be found that the DIRC code is systematic: the information field contains the original 1-of-n data while the check field is a single 1-of-n check word.

4.2.1. Check generating and error correcting processes

The check word generating process $f(X)$ is defined as (7):

$$c = f(X) = \sum X = x_0 + x_1 + \cdots + x_{CN-1} \tag{7}$$

An arbitrary data word x_j in a DIRC code word $(x_0, x_1, \ldots, x_{CN-1}, c)$ can be regenerated using the error correcting process $g(X_{\neq j})$ defined in (8), where $X_{\neq j} = (x_0, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{CN-1}, c)$ and x'_j is the regenerated data word of x_j.

$$x'_j = g(X_{\neq j}) = c - \sum_{i=0, i\neq j}^{CN-1} x_i = -\left[\left(-c + \sum_{i=0, i\neq j}^{CN-1} x_i\right)\right] \tag{8}$$

In (8), subtraction has been transferred to addition so that both the check generating and error correcting processes can be implemented using the same addition unit (Section 5). Fig. 11 gives examples of the DIRC coding scheme for several 1-of-n codes, where the $CN > 2$ case can also be inferred.

4.2.2. Error filtering

For any data word x_i $(0 \leqslant i < CN)$ in a DIRC code $(x_0, x_1, \ldots, x_{CN-1}, c)$, the regenerated data word x'_i is the same as x_i if no faults occur. Under the assumption of a 1-bit transient fault, either x_i or x'_i may be altered by a fault but not both. To obtain the error-free data word, an error filter $h(x_i, x'_i)$ is defined in (9), where "&" denotes the logical operation of a C-element. The final error-free data is $X'' = (x''_0, x''_1, \ldots, x''_{CN-1})$.

$$x''_i = h(x_i, x'_i) = x_i \;\&\; x'_i = x_i \;\&\; g(X_{\neq i}) \tag{9}$$

A fault on x_i is filtered using (9) because x'_i is calculated from other fault-free codes. If the fault adds an extra '1' to x'_i, the faulty '1' will be filtered since x_i is correct. What is important is whether a fault on $X_{\neq i}$ will erase the valid '1' in x'_i.

x_0	x_1	c
01	01	01
	10	10
10	01	10
	10	01

(a) 1-of-2 codes

x_0	x_1	c
001	001	001
	010	010
	100	100
010	001	010
	010	100
	100	001
100	001	100
	010	001
	100	010

(b) 1-of-3 codes

x_0	x_1	c
0001	0001	0001
	0010	0010
	0100	0100
	1000	1000
0010	0001	0010
	0010	0100
	0100	1000
	1000	0001
0100	0001	0100
	0010	1000
	0100	0001
	1000	0010
1000	0001	1000
	0010	0001
	0100	0010
	1000	0100

(c) 1-of-4 codes

Fig. 11. Examples of DIRC coding scheme when $CN = 2$.

Theorem 4.1. *Assume that $D^n(i), D^n(j)$ and $D^n(s)$ $(0 \leqslant i, j, s < n)$ are three 1-of-n codes where $D^n(i) + D^n(j) = D^n(s)$. $D^n(A^m)$ and $D^n(S^m)$ are two m-of-n $(1 \leqslant m \leqslant n)$ codes, and $D^n(A^m) + D^n(j) = D^n(S^m)$.*

If $i \in A^m, s \in S^m$; Otherwise, if $i \notin A^m, s \notin S^m$

Proof. If $i = a_k \in A^m$ $(0 \leqslant k < m)$, Eq. (10) can be inferred from the arithmetic rules of m-of-n codes.

$$\begin{aligned} D^n(A^m) + D^n(j) &= \bigcup_{t=0}^{m-1} D^n(a_t + j) = D^n(a_k + j) \cup \bigcup_{t=0, t\neq k}^{m-1} D^n(a_t + j) \\ &= D^n(i + j) \cup \bigcup_{t=0, t\neq k}^{m-1} D^n(a_t + j) \\ &= D^n(s) \cup \bigcup_{t=0, t\neq k}^{m-1} D^n(a_t + j) \end{aligned} \tag{10}$$

Because the union operation will not erase any '1's in the code words, it can be inferred that $s \in S^m$.

Else if $i \notin A^m, \forall a_t \in A^m$ $(0 \leqslant t < m)$, $D^n(a_t) \neq D^n(i)$.

$$D^n(A^m) + D^n(j) = \bigcup_{t=0}^{m-1} D^n(a_t + j) = D^n(S^m) \tag{11}$$

$$D^n(a_t + j) = D^n(a_t) + D^n(j) \tag{12}$$

Since $D^n(i) + D^n(j) = D^n(s), D^n(s) \neq D^n(a_t + j)$. Therefore, $s \notin S^m$. □

Theorem (4.1) implies that when faults occur on one of the code words in $X_{\neq i}$, the extra '1's brought by faults would add extra '1's to the result of the m-of-n operations (i.e. x'_i) but never erase the valid '1' in x'_i which ought to be produced by the original error-free code words. Therefore, C-elements are able to filter out the wrong bits.

Taking two 1-of-4 data words x_0 ("0010") and x_1 ("1000") for example, the check word c is "0001". Assuming a positive transient fault converts x_0 to the faulty "1010", the regenerated x'_0 and x'_1 are "0010" and "1010" respectively. For x'_1, the first '1' from left is generated from the invalid '1' in the faulty x_0 which can be filtered by C-elements. Using (9), we get x''_0 and x''_1 as follows:

$$x''_0 = x_0 \;\&\; x'_0 = 0010, \quad x''_1 = x_1 \;\&\; x'_1 = 1000$$

which are recovered error-free data words. If the check word is faulty while the two incoming data words are error-free, both regenerated data words will be erroneous. The errors can also be filtered by the C-elements.

Considering the impact of faults happening on data wires (Section 2.2), the 2-of-n code insertion (Fig. 7c) makes one of the operands of (9) a 2-of-n code. The faulty '1' will be filtered out by the C-element. The invalid 1-of-n code insertion has two cases.

One case is both the valid and the invalid 1-of-n code words are outputted. The invalid '1' will be filtered using (9). The other case is that the valid code is outputted twice, which can be taken as an invalid spacer insertion. The spacer will not be latched since the other operand of (9) is valid and contains '1'. As a result, the valid code is outputted only once. The deadlock which is the result of the invalid code insertion is also avoided. It can be concluded that, all 1-bit transient faults on 4-phase QDI pipelines can be tolerated using the DIRC to 1-of-n codes.

The proposed DIRC coding scheme can also tolerate some multi-bit transient faults. If multi-bit unidirectional transient faults happen in a single word while the other words of this DIRC code are fault-free, the faulty word can be corrected. (Unidirectional faults are either positive or negative faults, but not both, in a code word [11].)

4.3. Code evaluation

Table 1 summarizes the difference between the DIRC and several existing error-correction codes. Since a large number of existing asynchronous designs [4,6–8] use 1-of-n codes to encode data, the hardware support for 1-of-n implementation is also important. The DIRC code is the only one which is unordered, systematic and can be implemented using QDI circuits.

Coding efficiency can be measured by the code rate [32], as depicted in (13) where n is the code length or the number of occupied wires, and M is the number of binary codes it can represent. M equals to n for 1-of-n codes. Both code rates of the 1-of-2 and 1-of-4 codes are 0.50.

$$R = \frac{\log_2 M}{n} \tag{13}$$

Applying the proposed DIRC, one extra check word is added for every CN 1-of-n data words. The code rate of the DIRC (R_{DIRC}) can be expressed as (14), which decreases $1/(CN+1)$ compared with the code rate of the original 1-of-n code.

$$R_{DIRC} = \frac{CN\ \log_2 n}{n(CN+1)} \tag{14}$$

It can be inferred from (14) that the code rate of the DIRC code increases with CN. In the worst-case of $CN = 2$, the code rates of 1-of-2 and 1-of-4 codes are both 0.33, which is 33.3% less than their original code rates. When $CN = 5$, the code rate decreases only 16.7%. The code rates of most existing code redundancy techniques decrease further when applied to 1-of-n codes. For example, the code rate decreases by 42.9% when using the Hamming (7,4) code [18] to the 1-of-4 code. Applying the TRDIC code [28] which changes 1-of-n codes to 2-of-(n + 1) codes to obtain fault-tolerance, the code rate decreases by 33.3% for 1-of-2 codes and 20.0% for 1-of-4 codes respectively.

5. Hardware implementation of DIRC pipeline stages

Fig. 12 presents the implementation of a DIRC pipeline stage transmitting one DIRC code (x_0, x_1, c) $(CN = 2)$. It includes 1-of-n adders, error filters, CDs and an acknowledge generator (AckGen).

5.1. Implementation of 1-of-n adders and error filters

The check word generating and error correcting processes can be implemented using QDI circuits. The key components are 1-of-n adders. In Fig. 12, the upper two adders are used to regenerate data words while the bottom one is used to generate the check word. If no fault happens, operands of the adder are 1-of-n codes. When faults convert 1-of-n codes to erroneous m-of-n codes. The adder works under the m-of-n arithmetic rules.

The mathematical representation of the 1-of-n adder is depicted in (15):

$$s_i = \cup\{(a_j\ \&\ b_k) | i = (j+k) \bmod n, i, j, k \in [0, n)\} \tag{15}$$

where A $(a_{n-1}a_{n-2}\ldots a_0)$, B $(b_{n-1}b_{n-2}\ldots b_0)$ and S $(s_{n-1}s_{n-2}\ldots s_0)$ are all m-of-n codes, and the subscript denotes the bit index. S is the sum of A and B.

Proof. Because $S = A + B$, it can be inferred from (6) that:

$s_i = 1 (0 \leqslant i < n) \iff \exists j, k \in [0, n)$ s.t. $(j+k) \bmod n = i, a_j = 1$ and $b_k = 1$. Therefore, we have (15). □

Fig. 13 shows the hardware implementation of a 1-of-2 adder and a 1-of-4 adder. Only C-elements and OR-gates are employed to ensure the adder is QDI. Fig. 14 shows structures of 1-of-n adders when the adder has multiple operands (corresponding to different CN).

During the error correcting process, the negating operation of an m-of-n code is required in (8). According to (2) and (5), the negation of a 1-of-n code is merely a bit-reshuffle which is depicted in (16) where inv_A is the negated A (both A and inv_A are m-of-n codes).

$$inv\_A_i = A_{(n-i) \bmod n}, \quad \forall i \in [0, n) \tag{16}$$

The error filters described by (9) are combined with pipeline latches to improve area and speed performance. The resulting 3-input C-elements latch structure for data words is depicted in Fig. 12. The asynchronous latch for check words is built from 2-input C-elements.

5.2. Generation of check words

In the hardware implementation shown in Fig. 12, the check word is generated from the incoming data words rather than the recovered data words (the implementation using recovered data to generate the check word is shown in Fig. 15). Consequently, the check word generating process and the error correcting process run in parallel which reduces the forward delay. However, the newly generated check word would be erroneous when the incoming data words are wrong. Since the wrong data would be

Table 1
Comparison of different error-correcting codes.

Codes	Unordered	Systematic	1-of-n Implementation	QDI Implementation
Hamming [18]	No	Yes	No	No
Blaum and Bruck [12]	Yes	Yes	No	Unprovided
Cheng and Ho [22]	Yes	Yes	No	Unprovided
Zero-Sum [23,13]	Yes	Yes	No	No
TRDIC [28,29]	Yes	No	Yes	Yes
DIRC	Yes	Yes	Yes	Yes

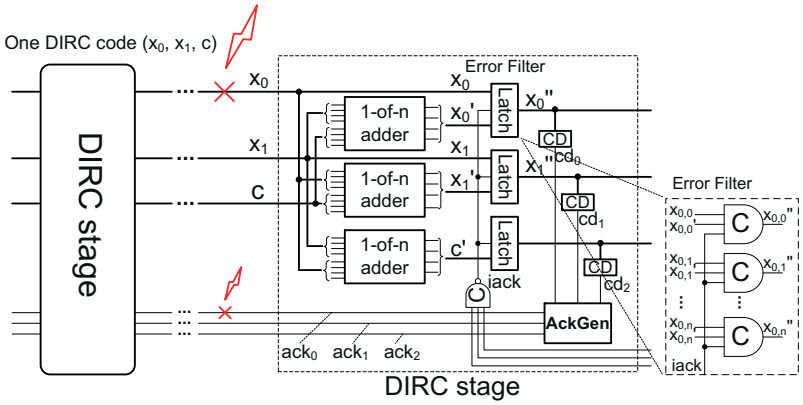


Fig. 12. Implementation of a DIRC pipeline stage transmitting one DIRC code ($CN = 2$).

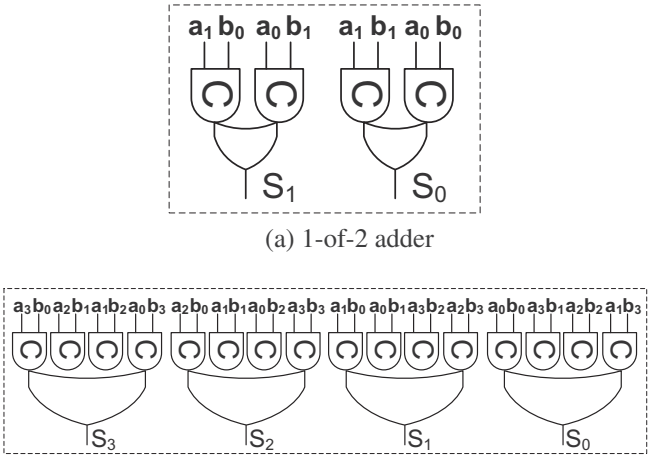


Fig. 13. Implementation of 1-of-n adders.

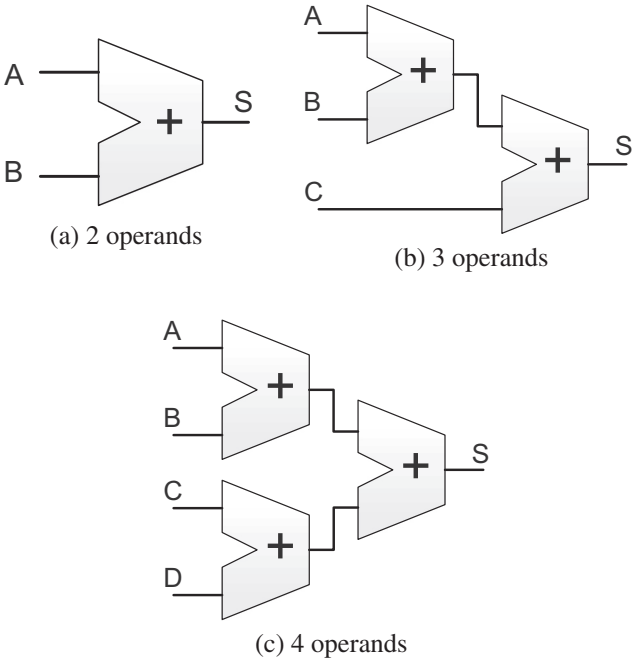


Fig. 14. Construction of 1-of-n adder trees with multiple operands.

corrected and not propagated to the next stage, the check word would not be an issue as long as no fault occurs on the corrected data again. Under the assumption of a 1-bit transient fault, executing the error-correction and check word generation processes in parallel is acceptable since the possibility of faults occurring on wires of adjacent stages is extremely low in practice.

5.3. Redundant protection of acknowledge wires (RPA)

A new redundant technique named RPA is proposed to protect acknowledge wires from transient faults. As shown in Fig. 16a, three C-elements are used to build an acknowledge generator (AckGen) which outputs three acknowledge signals (ack_0, ack_1, ack_2). The three inputs of AckGen are cd_0, cd_1 and cd_2, coming directly from the completion detection circuit. The original CD of a pipeline stage (Fig. 16b) can be easily divided into three sub-CDs by cutting off the bottom one or two C-elements when the number of 1-of-n slices of a pipeline stage is at least three (Fig. 16c). The generation of acknowledge signals is presented in (17) where "&" denotes the logical operation of a C-element. Finally, at the input side of the previous stage, an inverted 3-input C-element is used to generate the *iack* to asynchronous latches (Fig. 12).

$$\begin{cases} ack_0 = cd_0 \ \& \ cd_1 \\ ack_1 = cd_0 \ \& \ cd_2 \\ ack_2 = cd_1 \ \& \ cd_2 \\ iack = \neg(ack_0 \ \& \ ack_1 \ \& \ ack_2) \end{cases} \tag{17}$$

It can be found that the *iack* flips only when cd_0, cd_1 and cd_2 are all set high or all reset low. Any one of the three acknowledge signals relies on two sub-CDs and any one sub-CD decides two acknowledge signals, which ensures that a 1-bit transient fault on acknowledge wires will be masked. Only one extra C-element is added to the original CD (Fig. 16). The area overhead brought by RPA is negligible compared with the large number of data latches. This technique can be implemented independently of DIRC.

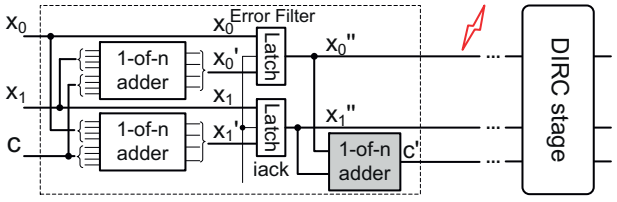


Fig. 15. The pipeline stage using the recovered data to generate the check word.

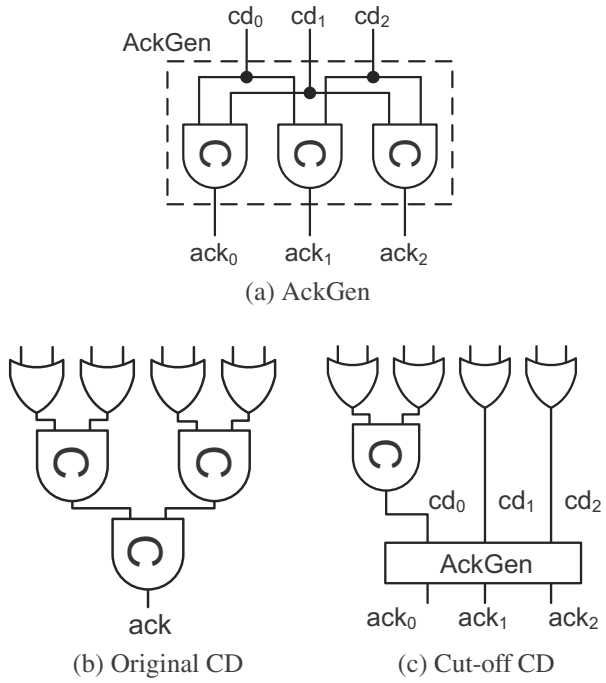


Fig. 16. Implementation of the acknowledge generator (AckGen).

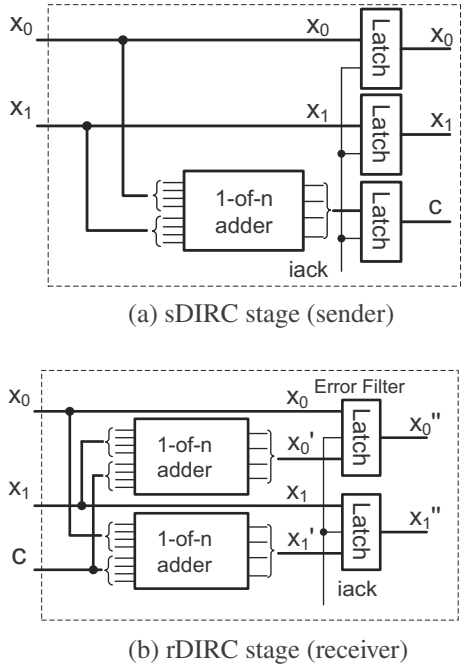


Fig. 18. Incomplete DIRC stages.

6. Case study: DIRC pipelines with different construction patterns

DIRC and RPA can be used in basic QDI pipelines (Fig. 3) to provide fault-tolerance with little modification. Fig. 17 presents a DIRC pipeline with three stages. An original pipeline stage with S ($S \geqslant 2$) 1-of-n slices is divided into GN ($GN \geqslant 1$) groups, each of which contains CN ($CN = S/GN$) slices. Applying DIRC, each group has an extra 1-of-n check word, resulting in a DIRC stage with $GN(CN+1)$ slices.

In a DIRC pipeline, the complete DIRC stages (Fig. 12) are placed between a pair of incomplete DIRC stages, which are sDIRC and rDIRC ones (Fig. 18). The sDIRC stage only generates check words while the rDIRC stage only corrects errors. Their sizes are smaller than the size of a complete one.

This section studies the DIRC pipelines with different construction patterns. Latency and area models are built to evaluate the hardware overhead of different DIRC pipelines.

6.1. Latency analysis

The saturation throughput of a QDI pipeline is determined by the equivalent period of two adjacent pipeline stages which form a loop (Fig. 4). A latency model is built to evaluate the loop latency and speed overhead of DIRC pipelines.

To make the model simple, it is assumed that cell delays for positive and negative transitions are the same while the wire delay is zero. The loop latency T can be expressed as (18), where t_L, t_{CD} and t_{Comb} are the propagation latency of an asynchronous latch, the CD and other combinational circuits respectively. The equivalent period can be approximated to $2T$.

$$T = 2t_L + t_{CD} + t_{Comb} \tag{18}$$

Assume a basic pipeline stage (denoted by a subscript of B) contains S slices. In our implementation (Fig. 3), the asynchronous latch is built from 2-input C-elements whose propagation latency is t_{c2}. CD is a tree constructed by n-input OR-gates and 2-input C-elements. Its delay $t_{CD,B}$ is determined by the propagation latency

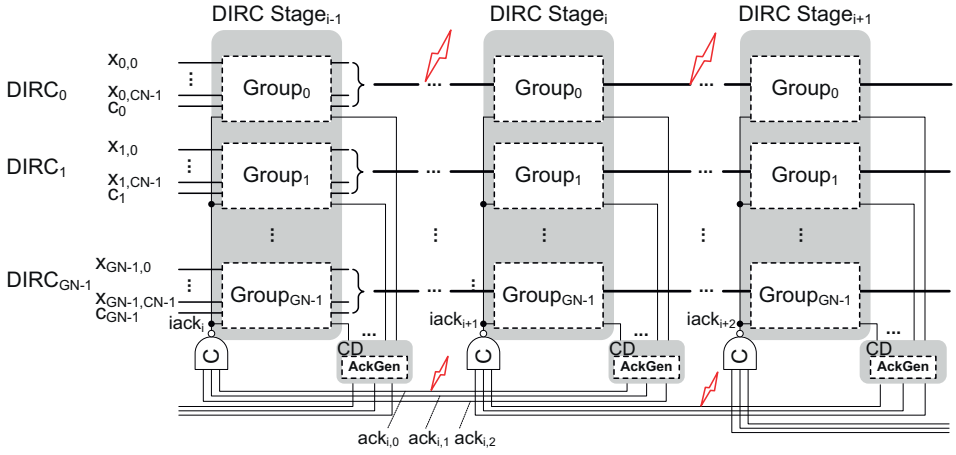


Fig. 17. A DIRC QDI pipeline.

of an n-input OR-gate (t_{OR}) and the tree depth ($\lceil \log_2 S \rceil$). The loop latency of a basic pipeline T_B can be estimated using (19) (the delay of the inverter used to invert the *ack* signal at the input of the latch is ignored).

$$t_{L,B} = t_{c2} \quad t_{CD,B} = t_{OR} + \lceil \log_2 S \rceil t_{c2} \quad t_{Comb,B} = 0 \tag{19}$$

For a DIRC pipeline using both DIRC and RPA, 1-of-n adders are added to the data path. The asynchronous latch for data words uses 3-input C-elements instead of 2-input ones. The redundant check words increase the tree depth of the CD (the new depth is $\lceil \log_2 (S + GN) \rceil \approx \lceil \log_2 S \rceil + 1$). Using RPA does not change the tree depth but only induces a delay of a 3-input C-element at the input side. Therefore, the loop latency of a DIRC pipeline T_{DIRC} can be estimated using (20):

$$\begin{aligned} &t_{L,DIRC} = t_{c3} \quad t_{Comb,DIRC} = t_{adder} \\ &t_{CD,DIRC} \approx t_{OR} + (\lceil \log_2 S \rceil + 1) t_{c2} + t_{c3} \end{aligned} \tag{20}$$

where t_{c3} and t_{adder} are the propagation latency of a 3-input C-element and 1-of-n adders respectively. t_{c3} and t_{c2} depend on the used cell library and the implementation of C-elements. Their values can be close. According to the implementation of 1-of-n adders (Section 5.1), t_{adder} can be estimated as (21).

$$t_{adder} = \lceil \log_2 CN \rceil (t_{c2} + t_{OR}) \tag{21}$$

Therefore, the loop latency of a DIRC pipeline can be expressed as (22).

$$T_{DIRC} = T_B + 3t_{c3} - t_{c2} + t_{adder} \tag{22}$$

It can be concluded that, using DIRC plus RPA, t_{adder} is the only variable determining the pipeline speed overhead while the other items in (22) are constants. The propagation latency of the adder is proportional to $\lceil \log_2 CN \rceil$. Usually $CN \ll S$ (since a large CN induces a large area overhead, which will be discussed in the next section), so that $t_{adder} \ll T_B$. In this case, the loop latency of a DIRC pipeline is generally less than $2T_B$. In other words, the period of a DIRC pipeline is less than twice as longer as the unprotected basic pipeline.

6.2. Construction of different DIRC pipelines

As a systematic code, the DIRC keeps the original data words so that the internal pipeline stages or combinational circuits between two DIRC stages can obtain the original data directly. This feature makes it possible to construct DIRC pipelines using different patterns, providing designers with flexible choices. Since different DIRC pipelines may have quite different area overhead, area models are built in this section to analyse the area overhead of DIRC pipelines. (The equivalent period of a DIRC pipeline is determined by the maximum loop and has a slight change with different patterns, so that only area overhead is studied.)

6.2.1. Area analysis

An area model for a single pipeline stage is first built to analyse the area of different pipeline stages, including the basic, the DIRC, the sDIRC and the rDIRC stages. Since the RPA technique protecting acknowledge wires causes little overhead compared with the DIRC and can be used independently, this area model evaluates only the area overhead brought by the DIRC.

The area of a single pipeline stage A can be expressed as (23), where A_L is the area of the asynchronous latch, A_{CD} is the area of the CD and A_{Comb} is the area of other combinational circuits. Wire area is assumed to be zero.

$$A = A_L + A_{CD} + A_{Comb} \tag{23}$$

For a basic pipeline stage with S 1-of-n slices, the asynchronous latch is built from 2-input C-elements and its area $A_{L,B}$ can be estimated as (24) where A_{c2} is the area of a 2-input C-element. The area of the CD (Fig. 16b) can be approximated to $A_{CD,B}$ shown in (25) where A_{OR} is the area of an n-input OR-gate. For the basic pipeline, $A_{Comb,B} = 0$. The area of a basic pipeline stage A_B can be estimated as (26) (the area of the inverter used to invert the *ack* signal is small and ignored).

$$A_{L,B} = S \cdot n \cdot A_{c2} \tag{24}$$

$$A_{CD,B} = S \cdot A_{OR} + (S-1) \cdot A_{c2} \approx S \cdot (A_{OR} + A_{c2}) \tag{25}$$

$$A_B = A_{L,B} + A_{CD,B} \tag{26}$$

Applying DIRC, the slices are divided into GN groups, each of which contains CN data words ($CN = S/GN$) and one check word. As a result, each DIRC stage contains $(S + GN)$ slices. As described in Section 5.1, 3-input C-elements are used to latch data and 2-input C-elements are used to latch check words in DIRC stages. The latch area $A_{L,DIRC}$ can be estimated as (27) where A_{c3} is the area of a 3-input C-element. The CD of a DIRC stage needs to detect $(S + GN)$ slices. Its area $A_{CD,DIRC}$ can be estimated as (28).

$$A_{L,DIRC} = S \cdot n \cdot A_{c3} + GN \cdot n \cdot A_{c2} = (A_{c3}/A_{c2} + 1/CN) \cdot A_{L,B} \tag{27}$$

$$\begin{aligned} A_{CD,DIRC} &= (S + GN) \cdot A_{OR} + (S + GN - 1) \cdot A_{c2} \\ &\approx (1 + 1/CN) \cdot A_{CD,B} \end{aligned} \tag{28}$$

A complete DIRC stage has $(S + GN)$ 1-of-n adders where S adders are used to regenerate data words and GN adders are used to generate check words (Fig. 12). The area of a single 1-of-n adder unit A_a increases with n and can be estimated as (29). When $CN > 2$, each adder becomes an adder tree containing $(CN - 1)$ 1-of-n adder units (Fig. 14). It is assumed that the area of adders regenerating data words and generating check words is $A_{adder,d}$ (i.e. $A_{Comb,rDIRC}$) and $A_{adder,c}$ (i.e. $A_{Comb,sDIRC}$) respectively. $A_{adder,d}$, and $A_{adder,c}$ can be expressed as (30) while their sum is A_{adder} (i.e. $A_{Comb,DIRC}$).

$$A_a = n^2 \cdot A_{c2} + n \cdot A_{OR} \tag{29}$$

$$\begin{cases} A_{Comb,rDIRC} = A_{adder,d} = S \cdot (CN - 1) \cdot A_a \\ A_{Comb,sDIRC} = A_{adder,c} = S \cdot (1 - 1/CN) \cdot A_a \\ A_{Comb,DIRC} = A_{adder} = S \cdot (CN - 1/CN) \cdot A_a \end{cases} \tag{30}$$

Assuming $A_{c3} = \alpha A_{c2}$ and $A_{OR} = \beta A_{c2}$ (usually $\alpha \approx 2$ and $\beta < 1$), (30) can be rewritten into (31). The area of a complete DIRC stage A_{DIRC} is depicted in (32).

$$\begin{aligned} &A_{adder,d} = (CN-1)(n+\beta) \cdot A_{L,B} \\ &A_{adder,c} = (1-1/CN)(n+\beta) \cdot A_{L,B} \\ &A_{adder} = (CN-1/CN)(n+\beta) \cdot A_{L,B} \end{aligned} \tag{31}$$

$$A_{DIRC} = (\alpha + 1/CN) \cdot A_{L,B} + (1 + 1/CN) \cdot A_{CD,B} + (CN - 1/CN)(n+\beta) \cdot A_{L,B} \tag{32}$$

Comparing (32) with (26), it can be found that most of the area overhead of a DIRC stage comes from 1-of-n adders. According to (29), the area of a single 1-of-n adder unit A_a increases with n^2. As a result, when applying the DIRC, 1-of-4 pipelines have larger area overhead than 1-of-2 pipelines with the same data width. For a 1-of-n DIRC pipeline stage with a fixed data width, A_{adder} increases approximately linearly with CN while the area overhead brought by the latch and CD decreases slightly (Eq. (32)). As a result, the area of a DIRC stage increases with CN.

The area of the incomplete DIRC stages, sDIRC and rDIRC, can also be estimated as (33).

$$\begin{cases} A_{sDIRC} = (1 + 1/CN) \cdot A_{L,B} + (1 + 1/CN) \cdot A_{CD,B} + (1 - 1/CN)(n+\beta) \cdot A_{L,B} \\ A_{rDIRC} = \alpha A_{L,B} + A_{CD,B} + (CN - 1)(n+\beta) \cdot A_{L,B} \end{cases} \tag{33}$$

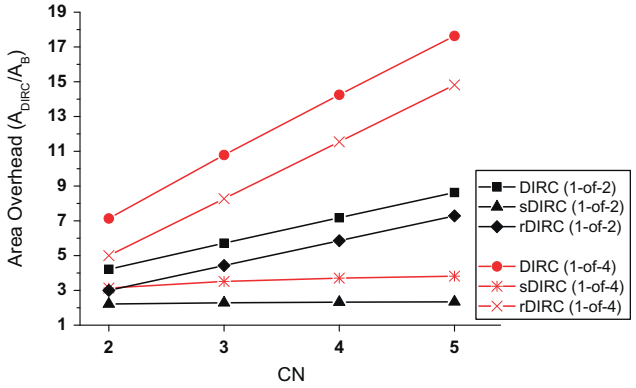


Fig. 19. Area overhead estimation of a single pipeline stage.

According to the Faraday UMC 0.13 μm standard cell library used in our implementation, it can be assumed that:

$$\alpha \approx 2; \quad \beta \approx 0.5 \tag{34}$$

Fig. 19 presents the area overhead estimation of different DIRC pipeline stages with different configurations. The baseline is the area of basic pipeline stages. The *Y*-axis uses the ratio of the area of DIRC (or sDIRC, rDIRC) stages to the area of the basic pipeline stages to measure the area overhead. Taking the 1-of-2 DIRC pipeline for example, when $CN = 2, A_{DIRC}/A_B$ is approximately 4.2; when $CN = 5$, the ratio rises to 8.6.

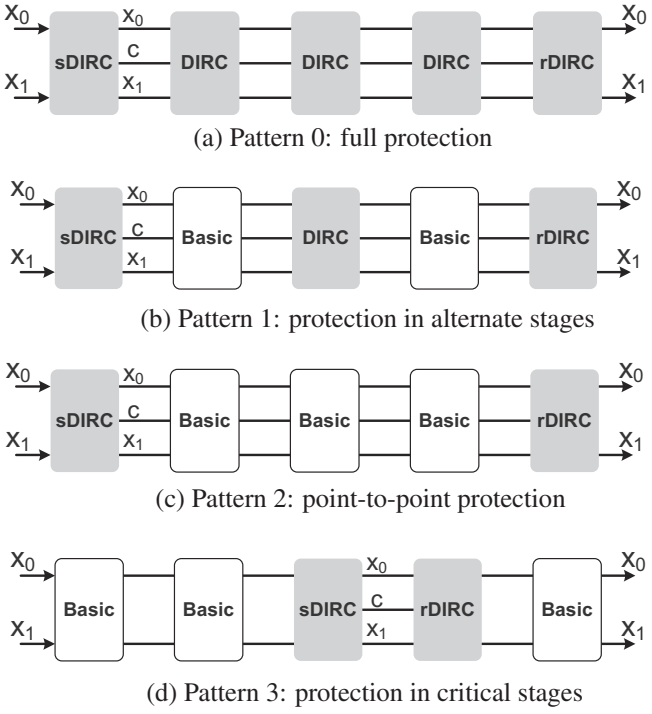
It can be found that, 1-of-4 pipeline stages have a steep gradient compared with 1-of-2 ones. The area overhead of all pipeline stages increases with *CN* while the complete DIRC stages have the largest area overhead. The sDIRC stage has the lowest gradient among the three kinds of DIRC stages. For an sDIRC stage with a fixed data width, the area of adders $A_{adder,c}$ increases with *CN* while the ratio of the area of the latch and the CD to their original area in a basic pipeline is $(1 + 1/CN)$, which decreases with *CN* (Eq. (33)). As a result, the ratio A_{sDIRC}/A_B increases slightly with a rising *CN* (the ratio increases from 2.21 to 2.34 for 1-of-2 pipelines and from 3.14 to 3.82 for 1-of-4 pipelines).

6.2.2. DIRC pipelines using different construction patterns

In Fig. 17, all pipeline stages are complete DIRC ones, which is a full-protection pattern (pattern 0 in Fig. 20a). This pattern makes the pipeline robust enough to tolerate all 1-bit transient faults between any two adjacent pipeline stages but incurs a large area overhead. Since the DIRC is systematic, this allows DIRC pipeline stages to be placed discontinuously to reduce the area overhead. This section builds area models for whole pipelines to evaluate the area overhead of different DIRC pipelines.

In a design, some parts may be critical or more susceptible to faults than other parts. In this case, DIRC stages can be used only in these parts to protect the communication. As Fig. 20 shows, DIRC stages can be placed using an arbitrary or specific pattern to provide protection, including the full protection, the protection in alternate stages, the point-to-point protection and the protection in critical stages. The protected pipeline segment starts from the sDIRC stage and ends with the rDIRC stage. Between the sDIRC and rDIRC stages, extra long wires are introduced for the redundant check words.

The area of a pipeline can be estimated as the total area of all pipeline stages and long wires. For the purpose of simplicity, it is assumed that all long wires between two adjacent stages have the same area (short internal wires are unconsidered). A_w is the area of $S \cdot n$ wires between two continuous basic stages. Applying the DIRC, the wire number becomes $S \cdot n \cdot (1 + 1/CN)$. As a result,



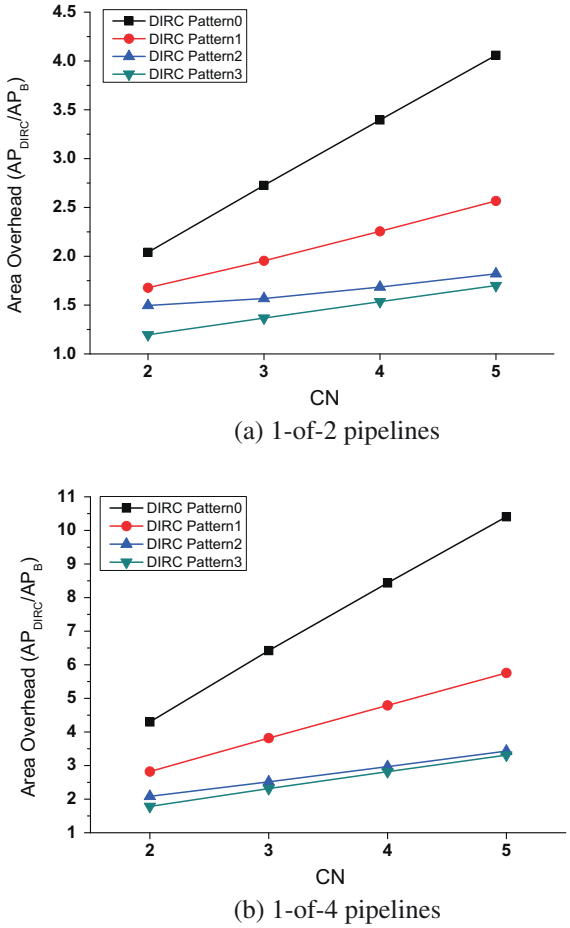
(a) Pattern 0: full protection

(b) Pattern 1: protection in alternate stages

(c) Pattern 2: point-to-point protection

(d) Pattern 3: protection in critical stages

Fig. 20. Different placement of DIRC stages.



(a) 1-of-2 pipelines

(b) 1-of-4 pipelines

Fig. 21. Area overhead estimation of different pipelines ($A_w = 0$).

in a protected pipeline segment, the wire area between two continuous stages is $(1+1/CN)\cdot A_w$ (acknowledge wires are ignored due to their small number). The original basic pipeline stages between the sDIRC and rDIRC are expanded due to the check words. The area of an expanded basic pipeline stage (A_{eB}) can be estimated as (35). The area of a whole pipeline (AP) is estimated in (36):

$$A_{eB} = (1+1/CN)\cdot A_L + A_{CD,DIRC} \approx (1+1/CN)\cdot A_B \quad (35)$$

$$AP \approx a\cdot A_B + b\cdot A_{eB} + c\cdot A_{sDIRC} + d\cdot A_{DIRC} + e\cdot A_{rDIRC} + f\cdot A_w \quad (36)$$

where a, b, c, d, e, f are constants for a specific pipeline.

Taking a 5-stage pipeline for example, the area of a basic pipeline AP_B can be estimated as (37):

$$AP_B = 5A_B + 4A_w \quad (37)$$

If the pipeline is fully protected (using three complete DIRC stages and two incomplete DIRC stages) as shown in Fig. 20a, the pipeline area $AP_{DIRC,P0}$ can be estimated as (38):

$$AP_{DIRC,P0} = 3A_{DIRC} + A_{sDIRC} + A_{rDIRC} + 4(1+1/CN)\cdot A_w \quad (38)$$

Protecting alternate stages (Fig. 20b), three DIRC stages are separated by two expanded basic stages. The area of this pipeline $AP_{DIRC,P1}$ is shown in (39).

$$AP_{DIRC,P1} = 2A_{eB} + A_{DIRC} + A_{sDIRC} + A_{rDIRC} + 4(1+1/CN)\cdot A_w \quad (39)$$

The area of pipelines with a point-to-point protection ($AP_{DIRC,P2}$) and a critical-stage protection ($AP_{DIRC,P3}$) can be estimated as (40) and (41) respectively.

$$AP_{DIRC,P2} = 3A_{eB} + A_{sDIRC} + A_{rDIRC} + 4(1+1/CN)\cdot A_w \quad (40)$$

$$AP_{DIRC,P3} = 3A_B + A_{sDIRC} + A_{rDIRC} + 3A_w + (1+1/CN)\cdot A_w \quad (41)$$

The area overhead of a whole 5-stage pipeline can be divided into two cases ($A_w = 0$ and $A_w > 0$) to discuss.

(1) The wire area is assumed to be zero ($A_w = 0$).
Using (35)–(41), an area overhead estimation of the pipelines using different patterns is summarized in Fig. 21. The area of the basic pipeline is used as the baseline. The Y-axis represents the area overhead brought by the DIRC, which is the ratio of the area of DIRC pipelines to the area of the corresponding basic pipelines (AP_{DIRC}/AP_B). It can be found that, for all pipelines, the area overhead increases approximately linearly with CN. The pipeline protecting only critical stages (pattern 3) brings the least redundant circuit.
It can be concluded that, DIRC pipelines using different construction patterns may have a quite different area overhead. If the DIRC is only used to protect some specific or critical pipeline segments, the incurred area overhead can be largely decreased.
(2) The wire area is non-zero ($A_w = k\cdot A_B$) ($k > 0$).

In practical pipelines, the data wires between two continuous stages may be long and occupy a large area due to the large numbers of inserted buffers. Assuming that $A_w = k\cdot A_B$ $(k>0)$, k is the ratio of the area of long wires to the area of the latch in a basic pipeline stage. Its value is decided by the practical circuit after fabrication which can be various. The area overhead of different pipelines is re-estimated using the previous model. Since the pipeline with a full protection (pattern 0) has the largest area overhead than other pipelines, the fully-protected DIRC pipeline is first presented to analyse the effect of k on the worst-case area overhead (Fig. 22).

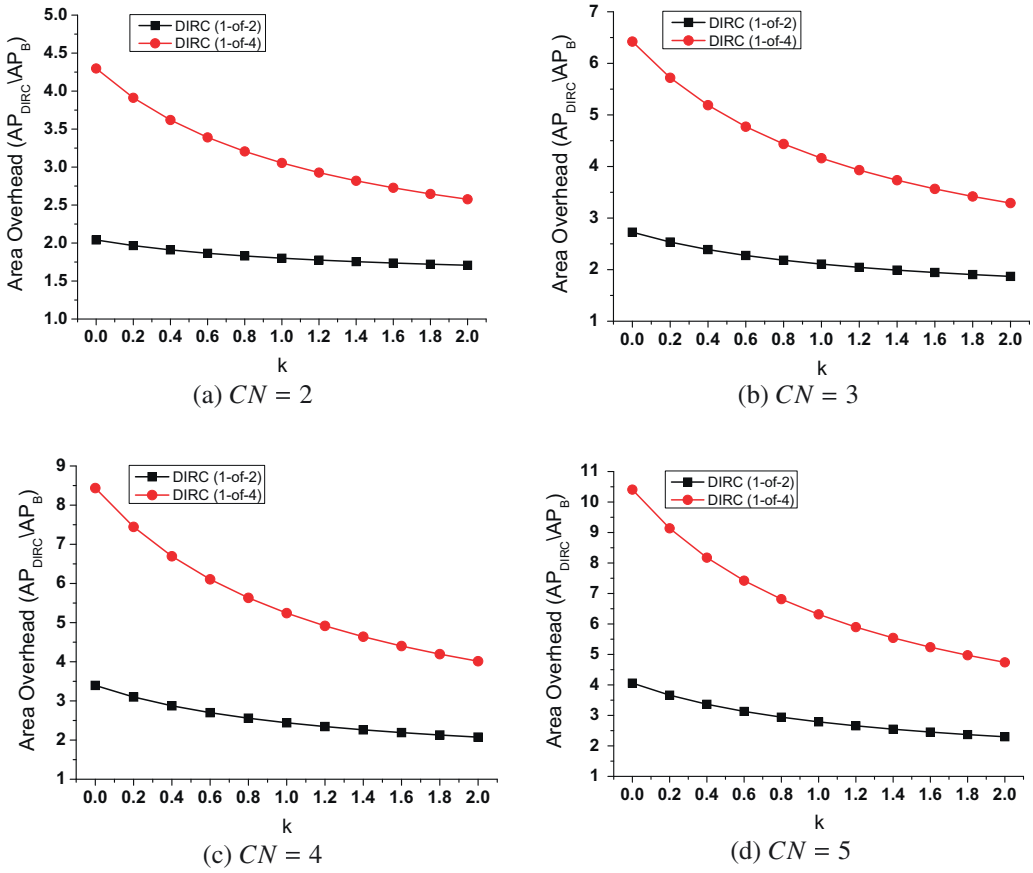


Fig. 22. Area overhead estimation of fully-protected pipelines with different CN ($A_w = k\cdot A_B$).

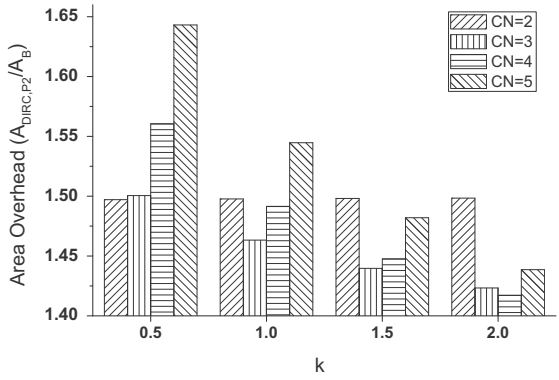
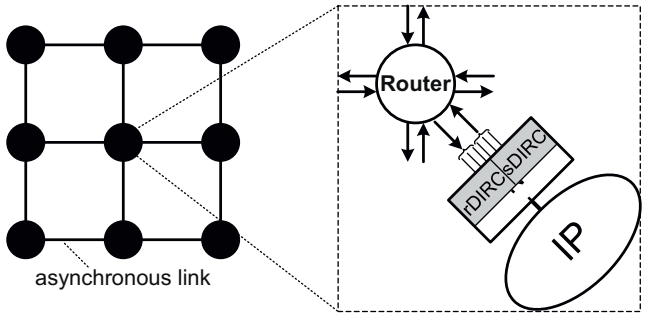
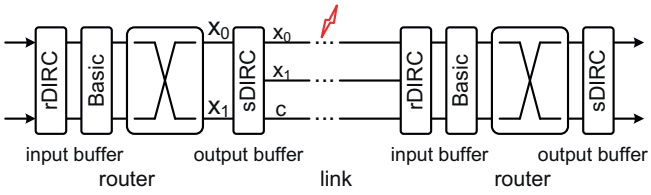


Fig. 23. Area overhead estimation of 1-of-2 pipelines using the point-to-point protection pattern.



(a) Using DIRC stages at network interfaces



(b) Using DIRC to protect a specific link

Fig. 24. DIRC applied to asynchronous NoCs.

For the fully-protected DIRC pipeline, the number of long wires between two continuous DIRC stages is $S \cdot (1 + 1/CN) \cdot n$.

In Fig. 22, the *Y*-axis still denotes the area ratio AP_{DIRC}/AP_B. The 1-of-4 pipelines have a larger area overhead than 1-of-2 pipelines. The area overhead reaches to maximum when $k = 0$ and then decreases with k. In an extreme condition when wires consume a significantly larger area than pipeline stages, the ratio will approach to $(1 + 1/CN)$. A small k denotes that the pipeline stages are the main contributor of the whole area, which is usually the practical case. Since the area of complete DIRC stages increases quickly with *CN* (Fig. 19), $CN = 2$ is more acceptable than a large *CN*. Although the wire number can be reduced by increasing *CN*, the area of the whole pipeline still increases with *CN*.

If the pipeline is constructed using other patterns, the area overhead can be further reduced. As an example, Fig. 23 presents the area overhead of 1-of-2 pipelines using the point-to-point protection pattern (pattern 2 in Fig. 20c). Since the area overhead of incomplete DIRC stages is lower than that of a complete DIRC stage (Fig. 19), increasing *CN* may not lead to a very large area of pipeline stages but reduces the wire area a lot. As a result, the pipeline area can be reduced. Fig. 23 shows cases where a large *CN* can reduce the area overhead. This feature provides flexible choices for designers to design a fault-tolerant system with an acceptable overhead according to the practical requirement.

6.2.3. Using DIRC in asynchronous NoCs

Since the DIRC code is systematic and supports a QDI implementation, it is especially suitable for large-scale asynchronous NoCs.

Built from asynchronous routers and links, a 3×3 mesh asynchronous NoC is illustrated in Fig. 24a. Network interfaces (NIs) are used to connect synchronous IP cores to the asynchronous network, implementing the transformation between the synchronous and asynchronous domain. The large numbers of long asynchronous links connecting routers are susceptible to transient faults. DIRC stages can be flexibly distributed in the network to protect the on-chip communication.

As an example, DIRC stages can be placed at the NIs (Fig. 24a), so that the check word generating and error-correcting operations are only executed when data passes NIs (which is a point-to-point protection pattern). A generic asynchronous router can be modeled as a pipeline where the input/output buffers, crossbar and other possible components are the pipeline stages. A packet traversing through multiple routers builds a data path which can be modeled

Code	Data width	GN	Basic Pipelines				DIRC Pipelines				DIRC/Basic			
			Area (μm^2)	*Delay (ns)*	*Period (ns)*	*Power (mW)*	*Area (μm^2)*	*Delay (ns)*	*Period (ns)*	*Power (mW)*	*Area*	*Period*	*Delay*	*Power*
1-of-2	4	2	193	0.071	1.34	0.41	960	0.260	2.10	0.76	4.97	1.57	3.67	1.86
	8	4	400	0.071	1.80	0.60	1791	0.266	2.65	1.06	4.48	1.47	3.75	1.76
	16	8	822	0.075	2.18	1.03	3321	0.279	3.04	1.79	4.04	1.39	3.73	1.74
	32	16	1727	0.075	2.76	1.74	6482	0.278	3.54	2.99	3.75	1.28	3.70	1.71
	64	32	3227	0.075	2.97	3.04	12673	0.290	3.92	5.34	3.93	1.32	3.86	1.76
	128	64	6481	0.078	3.32	5.55	24204	0.294	4.25	9.28	3.73	1.28	3.80	1.67
1-of-4	4	1	176	0.075	1.19	0.30	1424	0.257	1.93	0.56	8.09	1.63	3.44	1.91
	8	2	365	0.075	1.64	0.45	2842	0.266	2.28	0.93	7.79	1.39	3.56	2.07
	16	4	738	0.074	1.90	0.76	5535	0.277	2.80	1.39	7.50	1.48	3.73	1.84
	32	8	1357	0.075	2.29	1.17	10958	0.278	3.23	2.33	8.08	1.41	3.70	2.00
	64	16	2745	0.075	2.72	1.99	21711	0.301	3.64	4.28	7.91	1.34	4.01	2.15
	128	32	5664	0.079	3.09	3.80	43478	0.317	4.37	7.76	7.68	1.41	4.00	2.04

Fig. 25. Experimental results of the basic and DIRC pipelines ($CN = 2$).

as a long pipeline. DIRC can be used to protect any segments of the pipeline. Fig. 24b illustrates the situation where a specific link is protected. DIRC stages can also be placed to protect all or some critical routers.

7. Experimental results and evaluation

7.1. Hardware evaluation

To evaluate the hardware overhead of the proposed fault-tolerant techniques, the DIRC pipelines (DIRC + RPA, Fig. 17) are implemented and synthesized using the UMC 0.13 μm standard cell library. As a comparison, the unprotected basic pipelines (Fig. 3) are also implemented. Fig. 25 shows detailed experimental results for the $CN = 2$ case, including the area of a pipeline stage, forward delay, equivalent period and power consumption under different configurations. The time and power information are obtained from transmitting millions of data through 10-stage pipelines.

Since the wire area is decided by the practical design and the DIRC pipeline can be constructed using various patterns, the practical pipelines may have quite different area. To provide a general evaluation, only the area of complete DIRC stages is presented to demonstrate the area overhead brought by the DIRC and RPA. Fig. 26a compares the area of different pipeline stages, which increases with the pipeline data-width. DIRC and RPA introduce some area overhead due to the check generation and error correction mechanisms. On average, this ratio is around 4.15 for 1-of-2 pipeline stages and 7.84 for 1-of-4 ones (Fig. 26b), which is generally consistent with the area model proposed in Section 6.2.1.

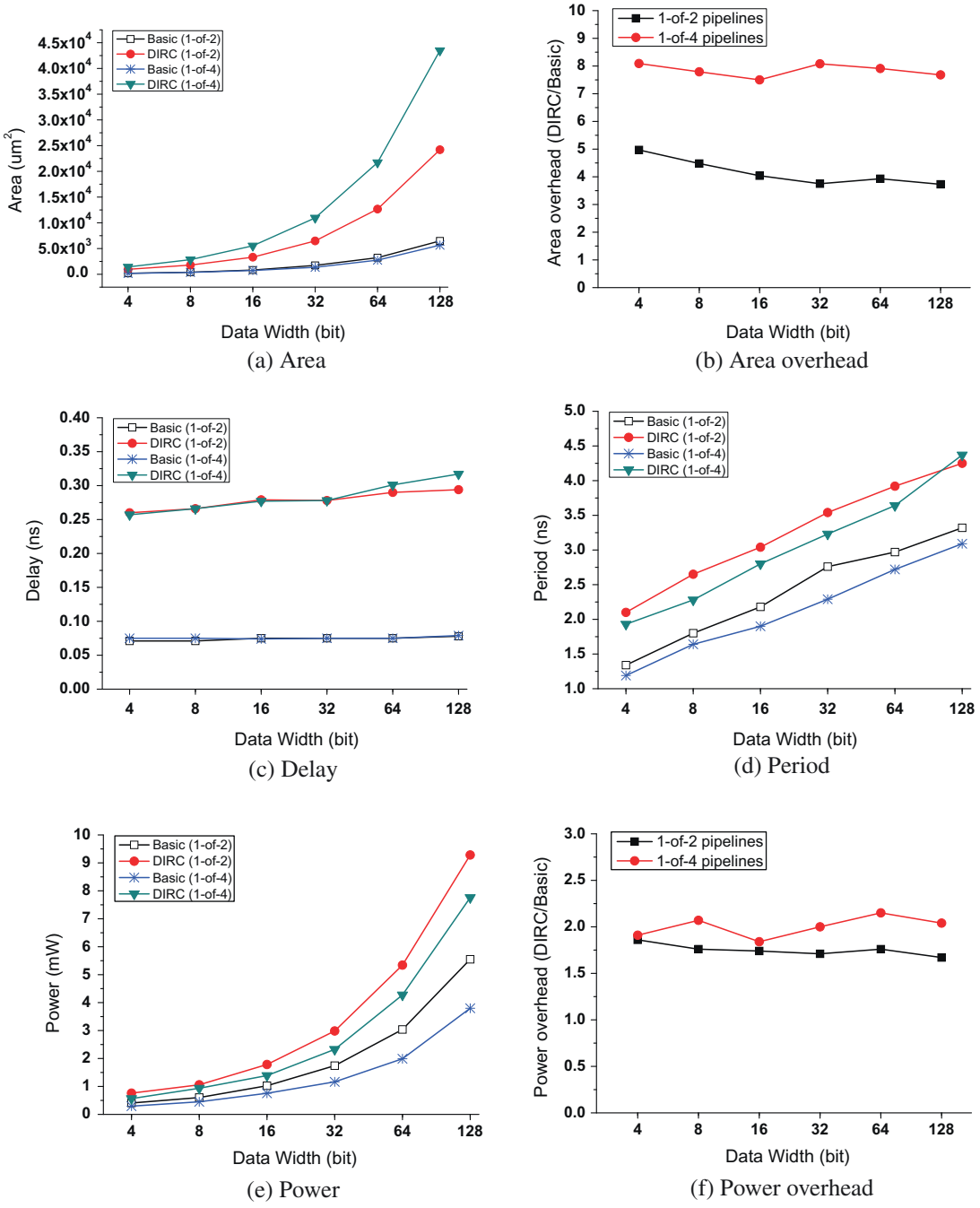


Fig. 26. Comparison between the basic and DIRC pipelines ($CN = 2$).

When the area of long wires is considered, this ratio will decrease. In practical designs where the pipeline is constructed using different patterns, the area overhead can be further reduced.

The forward delay of an asynchronous pipeline stage is the time needed by the data to traverse the asynchronous latch. Fig. 26c shows that the average forward delay increases slightly with data width due to the increasing area. On average, the forward delay of DIRC pipelines is 3.75 times the delay of basic pipelines. The fault-tolerant mechanism of the DIRC pipelines causes an extra delay (DIRC-Basic) which is only 0.21 *ns* on average for both 1-of-2 and 1-of-4 pipelines.

The equivalent period is an important factor affecting the maximum throughput of the pipeline (Section 6.1). Fig. 26d shows that the period increases with the data width of a pipeline. Since the CD tree is one level shallower in 1-of-4 pipelines than that of 1-of-2 pipelines, 1-of-4 pipelines have relatively shorter periods with the same data width. In most cases, the equivalent period of the DIRC pipeline is less than 1.5 times of the basic pipeline (Fig. 25). The period of the 128-bit wide 1-of-2 DIRC pipeline is only 1.28 times as long as that of the basic pipeline. Compared with the fault-tolerant design proposed by Jang and Martin [21] whose period is about twice slower than the basic one, the speed of DIRC pipelines is moderate and competitive.

The power consumption of pipelines also increases with the data width (Fig. 26e). The redundant circuit brought by DIRC and RPA causes more transition activities. On average, the power of 1-of-2 DIRC pipelines are 1.75 times greater than basic pipelines, while for 1-of-4 pipelines the ratio is about 2 (Fig. 26f).

To evaluate the effect of *CN* on the hardware overhead of DIRC pipeline stages, 60-bit wide 1-of-2 pipeline stages are implemented and synthesized. Fig. 27 compares the basic and the DIRC pipeline stages with different *CN*. It can be found that, the area of a pipeline stage increases approximately linearly with *CN*. The forward delay increases with *CN* since a larger *CN* means a deeper adder tree inserted to the data path. For the $CN = 3$ and $CN = 4$ cases, the adder trees have the same depth (both are 2-level) but the $CN = 3$ case requires more slices for the check words than the $CN = 4$ case, leading to a wider latch and a deeper CD tree. As a result, the speed of the DIRC stage with $CN = 3$ is a bit slower than the stage with $CN = 4$ (Fig. 27b and c). The power consumption also increases with *CN* due to the increased transition activities.

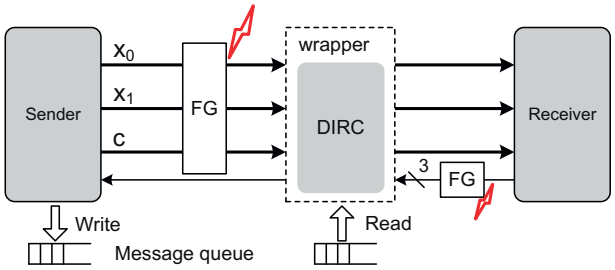


Fig. 28. Test environment for DIRC pipelines.

7.2. Evaluation of transient fault tolerance

To evaluate the fault-tolerance of the whole scheme (DIRC + RPA), a SystemC test environment is built (Fig. 28). It includes a sender, a receiver, a DIRC pipeline stage, fault-generators (FGs) and a stage wrapper. The DIRC pipeline stage is a synthesized gate-level netlist while other parts are SystemC models. The sender works as an sDIRC stage, producing random data and corresponding check words to the DIRC stage while the receiver works

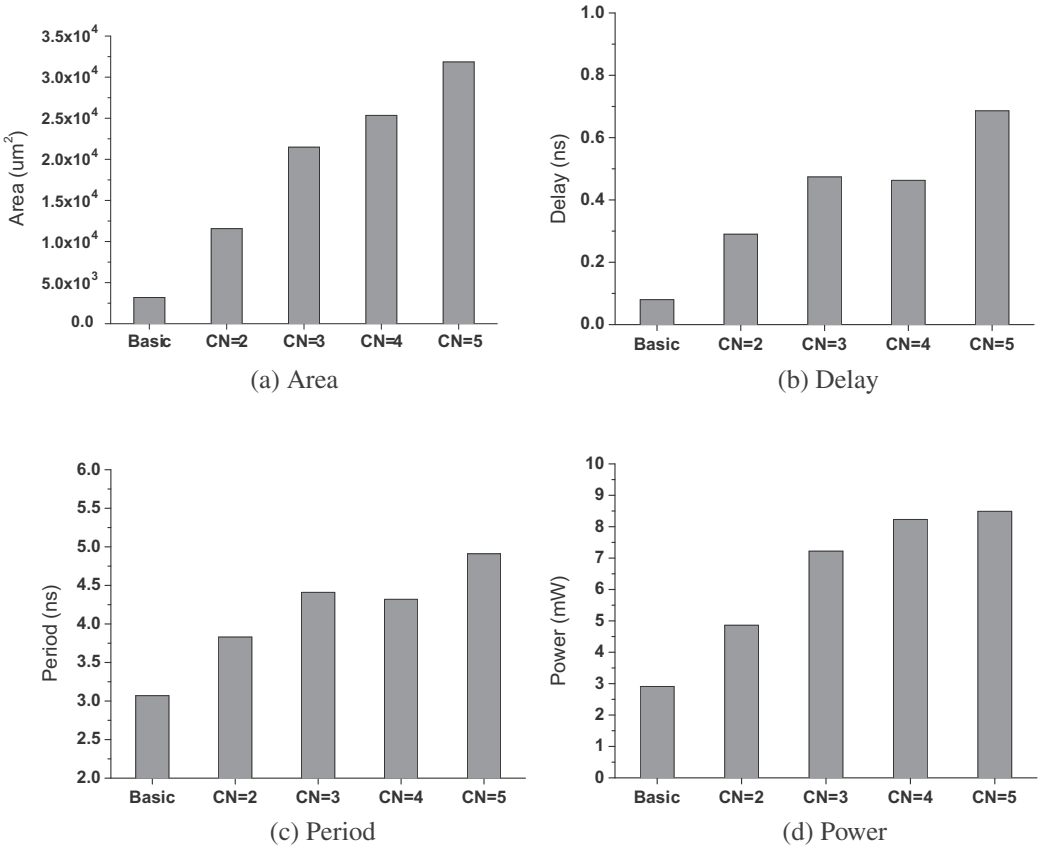


Fig. 27. Comparison between the basic and DIRC pipelines with different *CN*.

as an rDIRC stage. A fault-generator generates random faults on all wires including data wires (between the sender and the DIRC stage) and acknowledge wires (between the DIRC stage and the receiver). The stage wrapper checks the correctness of the output data and produces statistics. A shared message queue is used to store the error-free data being transmitted.

In this test environment, faults happen on any wires at any time, which mimics a real environment where not only 1-bit faults but also multi-bit faults may happen. Assuming that the occurrence of faults on a single wire is a Poisson process, the intervals between adjacent faults are randomized using an exponential distribution. Faults on different wires are inserted independently. The mean interval between faults is set to 1 μs while the duration of faults is randomized using a uniform distribution between 10 ps and 2 ns. These create a more comprehensive and severe fault environment than most in existing literature [21,24,28]. Data are transmitted continuously using the maximum injection rate. In total one million data packets were transmitted during the simulation.

The transient-fault-tolerance capability is evaluated by the mean time between failures (MTBF). Fig. 29a illustrates the MTBFs of different pipelines with $CN = 2$. Since the increasing number of wires lead to higher occurrence of faults, the MTBFs of all pipelines decrease with data width. When the data width is 4-bit, the MTBFs of the DIRC 1-of-2 and 1-of-4 pipelines are 2520 and 1748 times longer than the basic 1-of-2 and 1-of-4 pipelines respectively. When the data width rises to 128-bit, the ratio becomes 1117 for 1-of-2 pipelines and 1012 for 1-of-4 pipelines. For the basic 1-of-4 pipeline with a data width of 128 bits, 174647 out of 730498 transient faults result in errors during simulation period, while only 222 out of 1420558 transient faults lead to errors when using DIRC and RPA (Note that this test uses a multi-bit fault environment so that DIRC pipelines can make errors, while 1-bit transient faults will be fully tolerated). The resulting MTBF for the basic pipeline is 16 ns while it is prolonged to 16561 ns for the DIRC pipeline.

For DIRC pipelines using different *CN*s, Fig. 29b compares their MTBFs with the MTBF of a 60-bit wide 1-of-2 basic pipeline. It can be found that, under such a severe environment with multi-bit faults, the MTBFs of all DIRC pipelines are more than 1000 times longer than the basic pipeline. For the basic pipeline, 57067 out of 334679 faults result in errors while for the DIRC pipeline with $CN = 5$, only 62 out of 553143 faults result in errors. Most faults are filtered. A more severe multi-bit fault environment may largely reduce the MTBF of DIRC pipelines but this rarely happen. This also demonstrates the thousands-fold increment of fault-tolerance capability achieved by a DIRC pipeline.

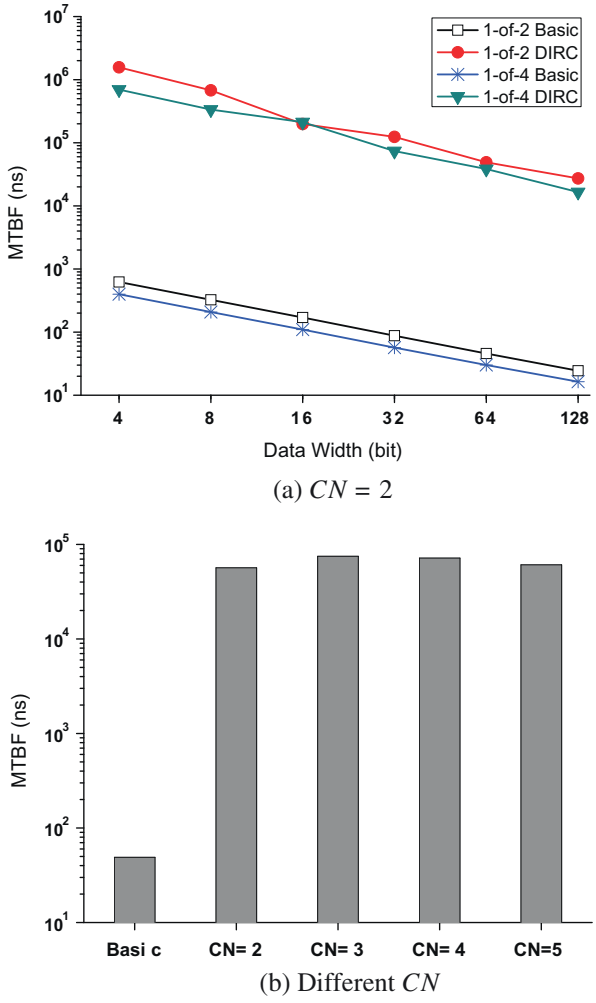


Fig. 29. Comparison of MTBF between the basic and DIRC pipelines.

8. Conclusions

This paper proposes a new unordered systematic fault-tolerant code, DIRC, providing the on-chip communication both timing-robustness and fault-tolerance. Using 1-of-n check words generated from multiple data words, the 4-phase QDI pipelines can tolerate all 1-bit and some multi-bit transient faults. Thanks to its simple structure, the DIRC stage can be easily used to replace all or arbitrary stages in existing 1-of-n QDI pipelines. Since DIRC is systematic, the DIRC pipelines can be constructed using different patterns to satisfy the practical fault-tolerance requirement with a reasonable hardware overhead. This feature makes DIRC especially suitable for large-scale communication-centric designs. A new redundant technique named RPA is also proposed to protect acknowledge wires from transient faults and can be used independently but causes little hardware overhead.

Latency and area models are built to evaluate the hardware overhead of different DIRC pipelines, providing designers a guidance to design a fault-tolerant system with an acceptable hardware overhead. Several DIRC and basic pipelines are implemented using the UMC 0.13 μm standard cell library. Detailed experimental results show that the DIRC pipelines achieve thousands-fold improvement on the fault-tolerance capability even in a severe simulation environment. The hardware overhead of DIRC pipelines (using DIRC plus RPA) is moderate and can be further reduced using different construction patterns. In most cases, the DIRC pipeline is less than 1.5 times slower than the basic one, while the fault-tolerance is measured to be more than 1000 times stronger. Furthermore, the DIRC coding scheme can be extended to m-of-n QDI interconnects [32] which is the future work.

Acknowledgement

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).

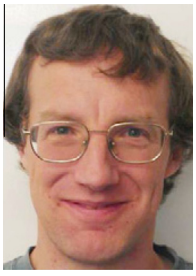
References

[1] R. Baumann, Soft errors in advanced semiconductor devices-part I: the three radiation sources, IEEE Trans. Dev. Mater. Rel. 1 (1) (2001) 17–22.
[2] C. Constantinescu, Trends and challenges in VLSI circuit reliability, IEEE Micro 23 (4) (2003) 14–19.
[3] J. Sparsø, S.B. Furber, Principles of Asynchronous Circuit Design: A Systems Perspective, Kluwer Academic Publishers, 2001.

[4] Y. Thonnart, P. Vivet, F. Clermidy, A fully-asynchronous low-power framework for GALS NoC integration, in: Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE), 2010, pp. 33–38.
[5] W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in: Proceedings of Design Automation Conference (DAC), 2001, pp. 684–689.
[6] F. Feliciian, S. Furber, An asynchronous on-chip network router with quality-of-service (QoS) support, in: Proceedings of IEEE International SOC Conference, 2004, pp. 274–277.
[7] T. Bjerregaard, J. Sparsø, Implementation of guaranteed services in the MANGO clockless network-on-chip, IEE Proc. Comput. Digital Tech. 153 (4) (2006) 217–229.
[8] J. Pontes, M. Moreira, F. Moraes, N. Calazans, Hermes-A – an asynchronous NoC router with distributed routing, in: Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2010, pp. 150–159.
[9] W.J. Bainbridge, S.J. Salisbury, Glitch sensitivity and defense of quasi delay-insensitive network-on-chip links, in: Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2009, pp. 35–44.
[10] G. Zhang, W. Song, J.D. Garside, J. Navaridas, Z. Wang, Transient fault tolerant QDI interconnects using redundant check code, in: Proceedings of Euromicro Conference on Digital System Design (DSD), 2013, pp. 3–10.
[11] B. Bose, On unordered codes, IEEE Trans. Comput. 40 (2) (1991) 125–131.
[12] M. Blaum, J. Bruck, Unordered error-correcting codes and their applications, in: Proceedings of International Symposium on Fault-Tolerant Computing (FTCS), 1992, pp. 486–493.
[13] M. Agyekum, S. Nowick, Error-correcting unordered codes and hardware support for robust asynchronous global communication, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 31 (1) (2012) 75–88.
[14] K.L. Shepard, V. Narayanan, Noise in deep submicron digital design, in: Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996, pp. 524–531.
[15] T. Karnik, P. Hazucha, Characterization of soft errors caused by single event upsets in CMOS processes, IEEE Trans. Dependable Secure Comput. 1 (2) (2004) 128–143.
[16] N.K. Jha, Separable codes for detecting unidirectional errors, IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. 8 (5) (1989) 571–574.
[17] W. Peterson, D. Brown, Cyclic codes for error detection, in: Proceedings of the IRE, vol. 49, 1961, pp. 228–235.
[18] R.W. Hamming, Error detecting and error correcting codes, Bell Syst. Tech. J. 29 (2) (1950) 147–160.
[19] T. Lehtonen, P. Liljeberg, J. Plosila, Online reconfigurable self-timed links for fault tolerant NoC, VLSI Des. 2007 (2007) 1–13.
[20] J. Lechner, M. Lampacher, T. Polzer, A robust asynchronous interfacing scheme with four-phase dual-rail coding, in: Proceedings of International Conference on Application of Concurrency to System Design (ACSD), 2012, pp. 122–131.
[21] W. Jang, A.J. Martin, SEU-tolerant QDI circuits, in: Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2005, pp. 156–165.
[22] F.-C. Cheng, S.-L. Ho, Efficient systematic error-correcting codes for semi-delay-insensitive data transmission, in: Proceedings of International Conference on Computer Design (ICCD), 2001, pp. 24–29.
[23] M.Y. Agyekum, S.M. Nowick, An error-correcting unordered code and hardware support for robust asynchronous global communication, in: Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE), 2010, pp. 765–770.
[24] S. Ogg, B. Al-Hashimi, A. Yakovlev, Asynchronous transient resilient links for NoC, in: Proceedings of IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis (CODES+ISSS), 2008, pp. 209–214.
[25] J. Lechner, V. Veeravalli, Modular redundancy in a GALS system using asynchronous recovery links, in: Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2013, pp. 23–30.
[26] Y. Monnet, M. Renaudin, R. Leveugle, Designing resistant circuits against malicious faults injection using asynchronous logic, IEEE Trans. Comput. 55 (9) (2006) 1104–1115.
[27] S. Almukhaizim, F. Shi, E. Love, Y. Makris, Soft-error tolerance and mitigation in asynchronous burst-mode circuits, IEEE Trans. VLSI Syst. 17 (7) (2009) 869–882.
[28] J. Pontes, N. Calazans, P. Vivet, Adding temporal redundancy to delay insensitive codes to mitigate single event effects, in: Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 142–149.
[29] J. Pontes, Soft Error Mitigation in Asynchronous Networks on Chip, Ph.D. thesis, The Pontifical Catholic University of Rio Grande do Sul (PUCRS), 2012.
[30] S. Peng, R. Manohar, Efficient failure detection in pipelined asynchronous circuits, in: Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2005, pp. 484–493.
[31] W. Kuang, E. Xiao, C. Ibarra, P. Zhao, Design asynchronous circuits for soft error tolerance, in: Proceedings of IEEE International Conference on Integrated Circuit Design and Technology (ICICDT), 2007, pp. 1–5.
[32] W.J. Bainbridge, W.B. Toms, D.A. Edwards, S.B. Furber, Delay-insensitive, point-to-point interconnect using m-of-n codes, in: Proceedings of IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2003, pp. 132–140.

Guangda Zhang received his B.S. degree in Computer Science and Technology from the National University of Defense Technology, China in 2010. He is currently pursuing his Ph.D. in Computer Science at the University of Manchester, UK. His current research interest mainly lies in fault-tolerant asynchronous circuits and asynchronous Networks-on-chip.

Wei Song received his B.S. and M.S. degrees in Electrical Engineering from Beijing University of Technology, China in 2005 and 2008 respectively. He received his Ph.D. in Computer Science at the University of Manchester, UK in 2012. His current research interests include high-speed asynchronous circuits, on-chip interconnection networks, systems-on-a-chip and dynamic reconfigurable logics.



Jim Garside gained his Ph.D. in Computer Science from the University of Manchester in 1987 for work in signal processing architecture. Post-doctoral work on parallel processing systems based on Inmos Transputers was followed by a spell in industry writing air traffic control software. Returning to academia gave an opportunity for integrated circuit design work, dominated by design and construction work on asynchronous microprocessors in the 1990s. More recently he has been involved with dynamic hardware compilation, GALS interconnection and the development of the hardware and software of the SpiNNaker neural network simulator.

Dr. Javier Navaridas is a Lecturer in Computer Architecture with the University of Manchester. He obtained his Ph.D. in Computer Engineering in 2009 from the University of the Basque Country which was rewarded with an Extraordinary Doctorate Award (Top 5%). He joined the University of Manchester with a prestigious Newton Fellowship in 2010. His research interests include interconnection networks for parallel and distributed systems, networks on chip for SoCs and Multiprocessors and characterization of applications behaviour. He has developed and maintained various simulation tools for different processes and systems.

Zhiying Wang received the Ph.D. degree in Computer Science and Technology from the National University of Defense Technology, China in 1989. He is currently a professor in the same university. He has contributed 10 invited chapters to book volumes, published 140 papers in archival journals and refereed conference proceedings, and delivered over 30 keynotes. His current research projects include asynchronous microprocessor design, nanotechnology circuits and systems based on optoelectronic technology, and virtual computer system. His main research fields include computer architecture, asynchronous circuit, computer security, VLSI design, reliable architecture and multi-core memory system.