Transient Fault Tolerant QDI Interconnects Using Redundant Check Code

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

***Abstract*—Asynchronous logic is a promising technology for building the chip-level interconnect of multi-core systems. However, asynchronous circuits are vulnerable to faults. This paper presents a novel scheme to improve the robustness of asynchronous systems. Our first contribution is a fault tolerant delay-insensitive redundant check coding scheme named DIRC. Using DIRC in 4-phase 1-of-n quasi-delay-insensitive (QDI) interconnects, all 1-bit and some multi-bit transient faults can be tolerated. The DIRC and the basic 4-phase 1-of-n pipeline stages are mutually exchangeable so that arbitrary basic stages can be replaced by DIRC stages to strengthen the fault-tolerance of long wires. Our second contribution, RPA, is a redundant technique to protect the acknowledge wires from transient faults – an issue that has long been disregarded by the community. The DIRC pipelines (using DIRC plus RPA) were simulated using the UMC 0.13μm standard cell library and compared with the basic pipelines. Detailed experimental results show that the 128-bit DIRC 1-of-4 pipeline is only 13% slower than the basic one but increases fault-tolerance hundred-folds when multi-bit transient faults are considered.**

Keywords- asynchronous interconnects; transient faults; fault tolerance; quasi-delay-insensitive circuits;

I. INTRODUCTION

The continuous shrinking of device geometry significantly increases the number of transistors and the density of wires integrated in a single chip which opens a vast range of opportunities. However, together with these opportunities several issues arise: the reduced supply voltage, the increased clock frequency and the broadened clock distribution network not only introduce extra difficulty in delivering the global clock signal all over the chip with acceptable clock skews, but also worsen the vulnerability of the circuits to faults [1-3]. For this reason, fault-tolerance becomes an essential design objective for critical digital systems, especially in highly specialized fields such as aerospace, military and medical equipment.

Asynchronous circuits provide a fundamental solution to problems caused by clocks [4] due to their clockless nature. They also have potential advantages in power consumption, modularity, composability and robustness, which has attracted many researchers in recent years.

Networks-on-chip (NoC), a kind of scalable and efficient on-chip communication infrastructure [5], has some attractive features which make them specially well suited for exploiting the advantages of asynchronous circuits: simplify timing closure, reduce power and resolve chip-level synchronization issues. Consequently, asynchronous NoCs are thought to be better candidates for current multi-core systems than their synchronous counterparts [6]. As an important issue in the design of NoCs, the large numbers of long wires are exposed to noises and faults, which lead to the variation of propagation delay and glitches [7].

Quasi-delay-insensitive (QDI) circuits [4] are a family of asynchronous circuits which tolerate nearly all delay variations as all wires and gates are assumed with positive infinite latencies. Unlike synchronous circuits, QDI circuits are unaffected by delay variation and skew. However, QDI circuits are not so robust to glitches compared with their robustness to delay variation. Without the sense of time, any erroneous transition or glitch may be accepted as a valid signal, making QDI circuits vulnerable to faults.

This paper proposes a novel transient fault tolerant coding scheme for 4-phase 1-of-n QDI interconnects. Using delay-insensitive redundant check (DIRC) code, the proposed scheme tolerates, with reasonable overhead, all 1-bit transient faults and some multi-bit transient faults. The DIRC code can be easily used in existing 1-of-n QDI pipelines to tolerate transient faults.

A novel fault-tolerant technique, RPA (redundant protection of acknowledge wires), is also proposed to protect the acknowledge wires from 1-bit transient faults. The QDI pipeline using DIRC and RPA provides both timing-robustness and fault-tolerance. A 128-bit DIRC 1-of-4 pipeline is only 13% slower than a basic QDI pipeline while its fault-tolerant capability increases hundred-folds.

This paper is organized as follows: Section II introduces the background of asynchronous protocols, transient faults and their impact on QDI interconnects. Section III presents related works. Section IV describes the proposed DIRC coding scheme and Section V demonstrates the hardware implementation of the DIRC pipeline. Section VI evaluates the DIRC pipeline with detailed experimental results and comparisons. Finally this paper is concluded in Section VII.

II. BACKGROUND

A. Asynchronous Protocols and 4-phase QDI Pipelines

Asynchronous communication protocols can be classified into 4-phase and 2-phase protocols [4]. As shown in Fig. 1, 4-phase protocols are level-triggered protocols where a reset or return-to-zero phase is required after transmitting a valid data. 2-phase protocols are edge-triggered where each transition of the acknowledge signal starts a new data transmission. 4-phase protocols are more widely used than 2-phase ones for the sake of implementation simplicity [8]. On

`/13 $26.00 © 2013 IEEE
DOI 10.1109/DSD.2013.11

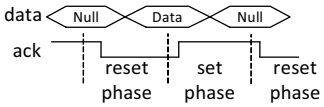

Figure 1. 4-phase protocol.

TABLE I. 1-OF-4 CODES

1-of-4 codes	Information (in binary)
0000	Null
0001	00
0010	01
0100	10
1000	11

most QDI interconnects, data are transmitted using delay-insensitive (DI) codes, also named unordered codes [8][9], where 1-of-n codes are the most utilized ones. As an extensively utilized case of 1-of-n codes, the 1-of-4 code is presented in Table I. This paper focuses on the fault-tolerance of 4-phase 1-of-n QDI interconnects.

Fig. 2 shows two adjacent stages of a basic 1-of-4 asynchronous QDI pipeline. Each stage is composed of a group of asynchronous latches and a completion detector (CD). The asynchronous latches are constructed by C-elements [4]. The CD is a tree of OR-gates and C-elements, detecting the completion of the incoming data and generating the acknowledge signal (*ack*) whose inverse (*inv_ack*) drives the latches of the previous stage. To reduce wire delay, buffers are inserted on long wires between stages to increase signal driving strength.

B. Transient Faults

Faults can be permanent faults or transient faults [10]. Transient faults are the most prevalent faults and nearly 80% of errors are caused by them [11]. A transient fault can be provoked by internal or external sources [12]. The internal sources include electrostatic discharge, power transients, capacitive coupling, inductive coupling and crosstalk, while the external sources include electromagnetic coupling and strikes of neutron or alpha particles. Transient faults usually last for a short period and cause errors when they are captured by memory components. The typical phenomenon caused by transient fault is a bit-flip, also known as a glitch. Note that glitches can be short or long. Long glitches rarely happen and cause complicated faulty situations which can be reduced to those created by short ones (Section II.C).

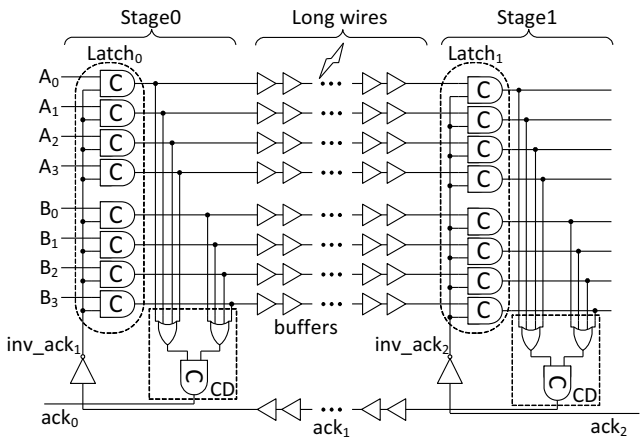

Figure 2. Two adjacent stages of a basic 1-of-4 asynchronous QDI pipeline.

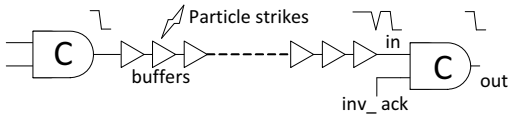

Figure 3. Particle strikes buffers of long wires.

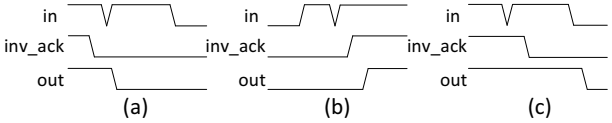

Figure 4. Negative transient faults on high-level data signals.

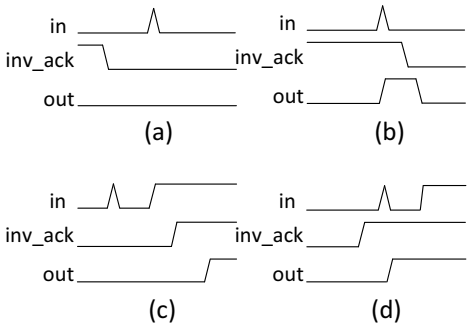

Figure 5. Positive transient faults on low-level data signals.

This paper focuses on the tolerance of transient faults on long wires (Fig. 2).

C. Impact of Transient Faults on 4-phase QDI Interconnect

Transient faults may occur on any wires at any time and cause glitches or erroneous transitions. Glitches include positive and negative ones. When glitches are captured by memory components, they become erroneous transitions. It is assumed that the 1-bit transient fault is incurred by particle strikes (Fig. 3). Multi-bit transient faults may largely increase the failure possibility of QDI pipelines but they rarely happen.

1) Negative transient faults on high-level data signals

Fig. 4 shows the situations when negative transient faults happen on high-level data signals. In Fig. 4(a) and Fig. 4(b), faults happen when the inverted acknowledge signal *inv_ack* is low; in Fig. 4(c), the fault happens when *inv_ack* is high. If *inv_ack* goes low earlier than the negative glitch (Fig. 4(a)), a premature reset operation is invoked. For the other two situations, faults are filtered by the C-element. None of the three situations will cause an error.

2) Positive transient faults on low-level data signals

Low-level data signals can be classified into two kinds. One is the data signal that keeps low as shown in Fig. 5(a) and Fig. 5(b) while the other one will go high in the next transmission or phase as shown in Fig. 5(c) and Fig. 5(d). For the former kind, if a positive fault happens when the *inv_ack* is low (Fig. 5(a)), the fault will be filtered by the C-element. Fig. 5(b) shows that if the fault happens when *inv_ack* is high, the output of the current C-element will go high incorrectly, which causes a *positive erroneous*

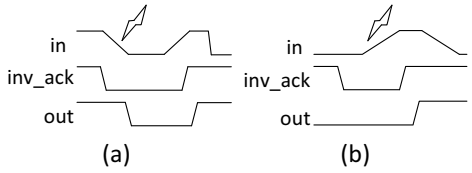

Figure 6. Long transient faults on incoming data signals.

transition. For the later kind of data signals (Fig. 5(c) and Fig. 5(d)), in the worst case the positive fault causes a premature set operation (Fig. 5(d)) and no erroneous transitions are incurred.

3) Long transient faults on data signals

In rare situations, the duration of transient faults can be long. Long transient faults lead to complicated faulty situations which can be eventually reduced to the situations of short transient faults discussed above. Fig. 6 depicts two examples of long transient faults on data signals. In Fig. 6(a), the long negative fault breaks the high-level input into two parts. Before the second part goes low, the *inv_ack* goes high already, which leads to a *positive erroneous transition*. In Fig. 6(b), the positive fault is long and remains active when *inv_ack* goes high, which again leads to a *positive erroneous transition*. Both situations can be reduced to the faulty situation shown in Fig. 5(b).

Leaving aside faults on the acknowledge wires, the only situation that leads to an error in a 4-phase 1-of-n QDI interconnect is a *positive erroneous transition* (Fig. 5(b), Fig. 6(a) and Fig. 6(b)).

4) Transient faults on acknowledge wires

The fault-tolerance of acknowledge wires is a serious issue which has been ignored by many existing fault-tolerant designs. Acknowledge signals are critical to QDI pipelines. Faulty acknowledge signals may lead to erroneous data or data missing. For example, in Fig. 7 out_0 and out_1 are the outputs of two continuous pipeline stages (S_0 and S_1) respectively while ack_0 and ack_1 are the corresponding acknowledge signals. Fig. 7(a) shows the correct situation without faults where two 1-of-4 data words ("0001" and "0100") are transmitted. Fig. 7(b) shows the faulty situation where a positive fault happens on ack_1. The data word "0001" has not been captured by S_1 and the positive fault creates a pseudo ack_1 indicating that a valid data word has been received by S_1 causing S_0 to reset. When the fault disappears, S_0 starts a new transmission. At this time, if S_1 has output the previous "0001", it will output the new "0100" continuously without a Null, which is equivalent to an invalid "0101". If S_1 has not output "0001", the previous "0001" may be rewritten into "0100", which leads to data missing.

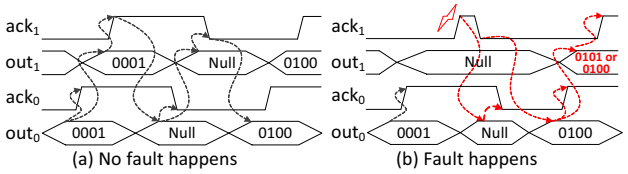

Figure 7. Transient faults on acknowledge wires.

III. Related Work

Most fault-tolerant techniques rely on redundancy, such as spatial redundancy, temporal redundancy or information redundancy [13]. These techniques can be classified into two categories: code redundant or hardware redundant techniques. Code redundant techniques use error-detecting codes (EDC) or error-correcting codes (ECC) to detect or correct errors, while hardware redundant techniques use extra hardware to provide fault-tolerance.

For code redundant techniques, a systematic error-correction unordered (ECU) coding scheme [14] using parity check codes can correct 1-bit error and detect completion simultaneously on 4-phase asynchronous links. Another ECU code named Zero-Sum [8] provides both 1-bit error correction and 2-bit error detection for asynchronous links. Utilizing several fault-tolerant methods, a self-timed bundled-data link [11] is able to tolerate transient and permanent errors but introduces large area overhead due to the extra de-interleaving and Hamming decoding processes. For hardware redundant technique, phase relationship has been used to tolerate transient faults [15]. Although these techniques seem promising, none of them can be easily implemented in QDI interconnects.

As for QDI designs, Jang et al. [3] proposed a duplicated double-checking technique which tolerates 1-bit faults and some multi-bit faults. The main difference with other communication-centric fault-tolerant techniques is that it can be used to build computational circuits. The fault-tolerant pre-charged buffers using this technique are only twice slower than the normal pre-charged buffers. Bainbridge et al. [7] proposed a series of fault-tolerant techniques for QDI designs but none of them are robust enough to qualify fault-immunity. In addition, Kuang et al. [16] studied the fault-tolerance of Null convention logic QDI circuits.

Pontes et al. [17] added temporal redundancy to 1-of-n codes to eliminate faults on 4-phase QDI pipelines, which is the closest work to ours. The proposed method encodes 1-of-n codes to 2-of-(n+1) codes and provides temporal redundancy using feedback latches. Erroneous data is filtered by double-checking. However, the acknowledge wires are not protected.

IV. Delay Insensitive Redundant Check Coding Scheme

The proposed DIRC code is based on systematic code, a widely utilized error-correcting coding paradigm. A systematic code [8][18] comprises of two fields: an information field and a check field. The information field contains the original data while the check field is generated from the data and can be used to recover the original data when errors occur. DIRC applies a systematic coding scheme to 1-of-n codes: the information field contains several data words denoted by 1-of-n codes and the check field is a single 1-of-n code generated from the 1-of-n data words. Thanks to the unordered nature of the 1-of-n codes, the check word generating and error correcting processes are implemented using QDI circuits with reasonable overhead.

A. Arithmetic Rules of m-of-n Codes

Let i be an integer lower than n ($0 \le i < n$), its 1-of-n code representation is an n-bit vector $D^n(i)$ where the $(i+1)th$ bit is high while the other n-1 bits are low. For example, the 1-of-4 codes for 1 and 3 are $D^4(1)$ ("0010") and $D^4(3)$ ("1000") respectively. We can define the basic arithmetic rules of 1-of-n codes as (1) ~ (3). For two integers a and b,

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

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

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

According to Section II.C, *positive erroneous transitions* may happen on data signals of 4-phase QDI pipelines under the assumption of 1-bit transient fault. As a result, 1-of-n codes may mutate into m-of-n ($m \ge 2$) codes. We use position vector $A^m=(a_0, a_1, \ldots, a_{m-1})$ ($1 \le m \le n$) to identify the positions of the m '1's in an m-of-n code (the order of a_i is arbitrary). 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 the m-of-n code with m 1-of-n codes, which is denoted by the symbol "$\bigcup$".

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

Take a 2-of-4 code "1100" for example. It can be considered as a union of two 1-of-4 codes $D^4(3)$ and $D^4(2)$. Both $D^4(3,2)$ and $D^4(2,3)$ denote "1100". The extended m-of-n arithmetic rules are shown in (5) and (6).

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

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

As described in (4) ~ (6), an m-of-n code is a union of m 1-of-n codes while the arithmetic of m-of-n codes is a union of 1-of-n operations. Taking 1-of-4 code $D^4(3)$ and 2-of-4 code $D^4(3,2)$ for example, we have:

$$\begin{aligned} -D^4(3,2) &= D^4(-3) \bigcup D^4(-2) = \text{"0010"} \bigcup \text{"0100"} = \text{"0110"}, \\ D^4(3) + D^4(3,2) &= [D^4(3) + D^4(3)] \bigcup [D^4(3) + D^4(2)] \\ &= D^4(2) \bigcup D^4(1) = \text{"0110"} \end{aligned}$$

B. DIRC Coding Scheme

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

DIRC Code: Let $X=(x_0, x_1, \ldots, x_{CN-1})$ be a data vector containing CN data words ($CN \ge 2$). A DI 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 a 1-of-n data word. They together comprise a DIRC code (x_0, x_1, …, x_{CN-1}, c) containing (CN+1) 1-of-n words.

Assuming that a data channel is divided into GN (GN>0) groups or data vectors, each group contains CN 1-of-n data words. Applying the DIRC code, the width of the new DIRC data channel, i.e. the number of wires of the DIRC channel, is $GN \cdot (CN+1) \cdot n$ while the wire number of the former data channel without check words is $GN \cdot CN \cdot n$. The number of redundant wires is $GN \cdot n$, which is exactly the number of wires occupied by the GN redundant check words.

A 32-bit wide 1-of-4 (n=4) data channel, for example, occupies 64 wires which comprise 16 1-of-4 data words. The total 16 data words can be divided into 8 groups (GN=8). Then each group contains 2 data words (CN=2) which comprise a data vector. Added with redundant check words, each DIRC code contains a data vector and a 1-of-4 check word. The width of the new DIRC channel is $GN \cdot (CN+1) \cdot n$=96 and the number of redundant wires is $GN \cdot n$=32.

If the data channel is divided into 4 groups (GN=4), correspondingly each DIRC code contains 4 data words (CN=4) and a check word. The width of the DIRC channel is $GN \cdot (CN+1) \cdot n$=80 and the number of redundant wires is $GN \cdot n$=16. It is found that with the increasing of CN, the required redundant wires are reduced and the code rate increases but the complexity of the implemented circuit becomes extremely large according to the synthesized results. Therefore, this paper only discusses the situation of CN=2.

Let function $f(X)$ be the check word generating process, it can be defined as (7) where the addition operation obeys the m-of-n arithmetic rules.

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

For arbitrary data word x_j ($0 \le j < CN$) in a DIRC code (x_0, x_1, …, x_{CN-1}, c), it can be regenerated from the other data words and the corresponding check word c using the error correcting process $g(X_{\neq j})$ defined in (8), where $X_{\neq j}=(x_0, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{CN-1}, c)$ ($0 \le j < CN$)) and x_j' is the regenerated data word of x_j.

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

In (8), subtraction has been transferred to addition. The whole design uses the same addition units. Fig. 8 gives examples of the DIRC coding scheme for several 1-of-n codes. The check word generating process and the error-correcting process follows (7) and (8) respectively, both of which obey the m-of-n arithmetic rules. Taking the 1-of-4 code for example, a data vector contains two data words x_0 ("0010") and x_1 ("1000"). Using (7) and (8), we have:

$$\begin{aligned} c &= x_0 + x_1 = D^4(1) + D^4(3) = D^4(0) = \text{"0001"} \\ x_0' &= c - x_1 = D^4(0) - D^4(3) = D^4(-3) = \text{"0010"} \\ x_1' &= c - x_0 = D^4(0) - D^4(1) = D^4(-1) = \text{"1000"} \end{aligned}$$

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

Figure 8. Examples of DIRC coding scheme.

For any data word x_i ($0 \le i < CN$) in a DIRC code (x_0, x_1, …, x_{CN-1}, c), the regenerated data word x_i' equals to x_i when no fault occurs. Under the assumption of 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, a filter function $h(x_i, x_i')$ is defined in (9), where '&' is used to denote the logical operation of a C-element.

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

As described in Section II.C, only *positive erroneous transitions* may occur on data wires under the assumption of 1-bit transient fault. Faults only add extra '1's to 1-of-n codes and convert them into m-of-n ($2 \le m < n$) codes rather than erase any valid '1's. On the other hand, the m-of-n arithmetic operations are unions of multiple 1-of-n operations. When faults occur, the extra '1's brought by faults would add extra '1's to the results of these m-of-n operations but never erase the '1's which ought to be produced by the error-free inputs. Since one of the two operands of (9) (x_i and x_i') must be correct, the C-elements are able to filter out the wrong one.

Taking two 1-of-4 data words x_0 ("0010") and x_1 ("1000") for example, the corresponding check word c is "0001". Assuming a transient fault converts x_0 to the faulty "1010", the regenerated data x_0' and x_1' are "0010" and "1010" respectively. For x_1', the second '1' from right corresponds to the invalid '1' in the faulty x_0 which can be filtered by C-elements. Using (9), we get $x_0''=x_0 \& x_0'$="0010" and $x_1''=x_1 \& x_1'$="1000" which are recovered data words. If the check word is erroneous while the two incoming data words are error-free, both the two regenerated data words will be erroneous but, again, will be filtered by the C-elements.

The final error-free data vector is $X''=(x_0'', x_1'', \ldots, x_{CN-1}'')$.

V. Hardware Implementation

Fig. 9(a) presents a DIRC QDI pipeline stage which includes 1-of-n adders, asynchronous latches, a CD and an acknowledge generator (AckGen).

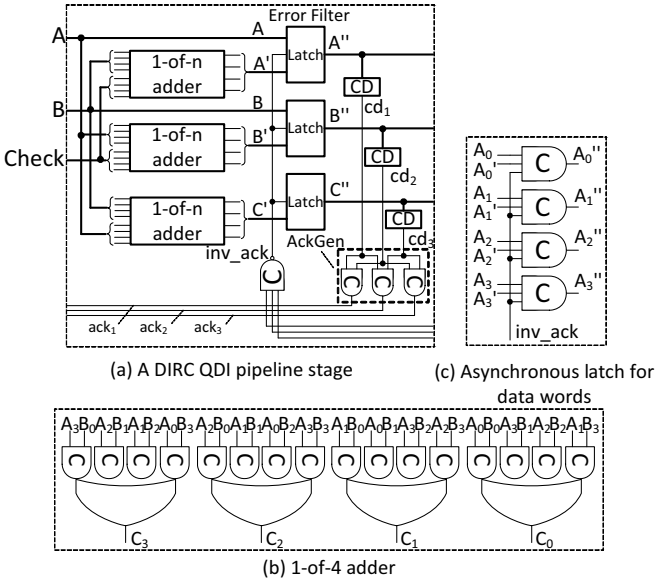

Figure 9. A DIRC QDI pipeline stage and relevant components.

A. Implementation of 1-of-n Adder and Error Filters

According to the proposed DIRC coding scheme, the key component of the whole design is the QDI 1-of-n adder. When faults convert a 1-of-n code to an erroneous m-of-n code, the 1-of-n adder works under the extended m-of-n arithmetic rules. The mathematical representation of the QDI 1-of-n adder is:

$$C_i = \bigcup\{(A_j \,\&\, B_k) \mid i = (j+k) \bmod n,\ i, j, k \in [0, n)\} \tag{10}$$

where A, B and C are 1-of-n codes (would be m-of-n codes when faults occur), and the subscript denotes the bit index. As an example, Fig. 9(b) shows an implementation of a QDI 1-of-4 adder.

During the error correcting process, we need to implement the negating operation of m-of-n codes as shown in (8). According to (5), a negated m-of-n code is a union of m negated 1-of-n codes. The negation of 1-of-n code is merely a bit-reshuffle according to (2). Assuming the *inv_A* is the negated A, we have (11).

$$inv\_A_i = A_{(n-i) \bmod n} \ (0 \le i < n) \tag{11}$$

Therefore, the negating operation of m-of-n codes incurs no hardware overhead.

In our implementation, 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. 9(c).

B. Generation of Check Words

In our implementation, the check word is generated from the incoming data words rather than the recovered data words. 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 may be erroneous because the incoming data words may be

wrong. If the current stage outputs an erroneous check word and the outputting data word in the same DIRC code encounters faults during its transmission, the error-correcting process of the next stage will fail because both operands of (9) are erroneous. Under the assumption of 1-bit transient fault, it is acceptable as the adjacent pipeline stages are related circuits and the possibility of faults on adjacent stages is extremely low in practice.

C. Redundant Protection of Acknowledge Wires (RPA)

A new redundant technique named RPA is proposed to protect the acknowledge wires from transient faults. As shown in Fig. 9(a), three C-elements are used to build the acknowledge generator (AckGen) which outputs three acknowledge signals (ack_1, ack_2 and ack_3). The three inputs of AckGen are cd_1, cd_2 and cd_3, which are directly from three sub-CDs (the original CD can be easily divided into three sub-CDs). Then we have (12) where '&' denotes the logical operation of a C-element.

$$\begin{cases} ack_1 = cd_1 \,\&\, cd_2 \\ ack_2 = cd_1 \,\&\, cd_3 \\ ack_3 = cd_2 \,\&\, cd_3 \\ inv\_ack = \neg(ack_1 \,\&\, ack_2 \,\&\, ack_3) \end{cases} \tag{12}$$

Therefore, any one of the three acknowledge signals relies on two sub-CDs and any one sub-CD decides two acknowledge signals, which ensures that 1-bit transient fault on acknowledge wires can at most cause a premature operation but will not cause erroneous bit-flips. The *inv_ack* flips only when cd_1, cd_2 and cd_3 are all set high or all reset low. Considering the width of data wires, RPA protects the acknowledge wires from 1-bit transient faults with low overhead. Note that its design is general enough that it can be implemented independently of DIRC and with other QDI fault tolerance schemes.

VI. Experimental Results

A. Code Evaluation

Coding efficiency can be measured by code rate R [18], as shown in (13), where n is the length of the code or the number of occupied wires while M is the size of the code or the number of valid code words.

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

The code rates of the basic 1-of-2 and 1-of-4 code are both 50%. Applying the proposed DIRC technique, *CN* 1-of-n data words correspond to one 1-of-n check word. Averagely the number of redundant wires for each 1-of-n data word is *n/CN*. When *CN* is 2, the rates of the DIRC 1-of-2 and the DIRC 1-of-4 code are both 33.33%. We use R_b and R_{DIRC} to denote the rates of the basic 1-of-n code and the DIRC code respectively. For the DIRC code, the decreasing rate R_d of its code rate on the basic 1-of-n code is defined in (14).

$$R_d = 1 - \frac{R_{DIRC}}{R_b} \tag{14}$$

When *CN* is 2, the decreasing rate R_d of the DIRC code is 33.34% which means the code rate of the DIRC code decreases 33.34% compared with the basic 1-of-n code. Most code redundant techniques incur high decreasing rates when applied to 1-of-n codes. For example, the decreasing rate of the Hamming (7, 4) code on 1-of-4 codes is 42.86%. Pontes et al. [17] changes 1-of-n codes to 2-of-(n+1) codes to obtain fault-tolerance. The decreasing rate R_d of the newly built 2-of-3 code on the basic 1-of-2 code is 33.34% while for 1-of-4 code, R_d is only 20%. This technique has relatively high code efficiency but the overhead of the practical circuit can be large with increasing data width.

The DIRC coding scheme provides QDI interconnects with the ability of tolerating 1-bit transient faults, meaning that if only a single bit encounters faults, it will be tolerated. However, inferred from Section IV, the proposed DIRC technique can actually tolerate single-word transient faults. If multi-bit positive transient faults happen in a single word while the other words of this DIRC code are fault-free, the erroneous word can also be corrected.

B. Hardware Evaluation

The DIRC QDI pipelines (DIRC plus RPA, Fig. 9(a)) are implemented and synthesized using the UMC 0.13μm standard cell library. As a comparison, the basic pipelines without fault-tolerant mechanisms (Fig. 2) are also implemented. In the experiment, a data packet is divided into *GN* groups and each group has two data words (*CN*=2) (for the DIRC pipeline, each group is a DIRC code containing two data words and a check word). Table II shows detailed experimental results, including area, average forward delay and average equivalent period under different configurations.

Fig. 10(a) shows that the area of pipelines increases with their data widths. DIRC introduces some area overhead due to the error detection and correction mechanisms. On average this ratio is around 4.45 for the 1-of-2 pipelines while it is 7.47 for the 1-of-4 pipelines. Note that these ratios must be understood as worst-case ratios because the buffers on long wires are not included (when *CN*=2, only 50% extra wires are introduced by DIRC and 2 extra wires are introduced by RPA). Therefore, in practical designs, the real ratio should be small and approaching 1.5 as buffers consume a significantly larger area than the stages.

The forward delay of an asynchronous pipeline stage is the time needed by data to traverse the asynchronous latch, excluding the latency of the CD and acknowledge circuits. The equivalent period is the time of a complete data transmission, including the set and the reset phases of the 4-phase handshake protocol. The average forward delay and the average equivalent period in Table II are obtained from transmitting millions data through 20 stages of pipelines.

Fig. 10(b) shows that the pipeline delay increases slightly with data width due to the increased wire load model used in synthesis. The fault-tolerant mechanism of the DIRC pipelines causes an extra delay (DIRC-Basic). On average the delay overhead is around 0.3 ns for 1-of-2 pipelines

TABLE II. EXPERIMENTAL RESULTS OF THE BASIC AND DIRC QDI PIPELINES

Code	Data width	GN	Basic Pipelines			DIRC Pipelines			DIRC/Basic		DIRC-Basic
			Area (μm^2)	*Delay (ns)*	*Period (ns)*	*Area (μm^2)*	*Delay (ns)*	*Period (ns)*	*Area*	*Period*	*Delay (ns)*
1-of-2	4	2	158	0.089	2.007	745	0.385	2.969	4.72	1.48	0.296
	8	4	326	0.089	2.824	1459	0.393	4.215	4.48	1.49	0.304
	16	8	654	0.089	3.556	2891	0.395	4.602	4.42	1.29	0.306
	32	16	1318	0.089	3.647	5755	0.396	4.946	4.37	1.36	0.307
	64	32	2638	0.100	5.137	11479	0.398	7.541	4.35	1.47	0.298
	128	64	5278	0.127	6.746	22927	0.413	8.216	4.34	1.22	0.286
1-of-4	4	1	136	0.090	1.839	1084	0.590	3.37	7.97	1.83	0.500
	8	2	282	0.090	2.636	2145	0.598	3.668	7.61	1.39	0.508
	16	4	574	0.090	3.395	4275	0.610	4.817	7.45	1.42	0.520
	32	8	1166	0.090	3.485	8497	0.614	5.332	7.29	1.53	0.524
	64	16	2334	0.099	4.980	16921	0.635	7.040	7.25	1.41	0.536
	128	32	4670	0.128	6.586	33857	0.662	7.443	7.25	1.13	0.534

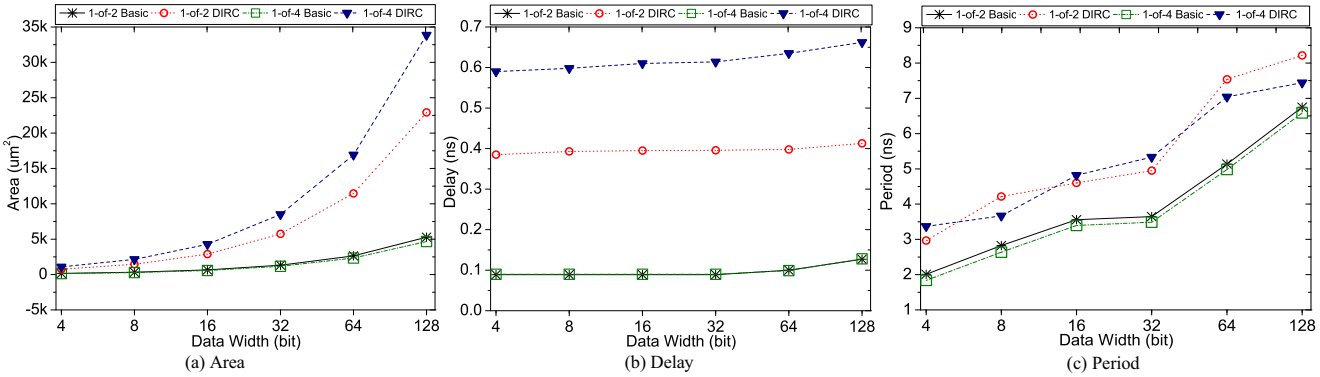

Figure 10. Comparison of area, average forward delay and average period between the basic and DIRC QDI pipelines.

while it is 0.52 ns for the 1-of-4 cases.

The average period is an important factor in evaluating pipeline because it affects the maximum achievable throughput. As shown in Fig. 10(c), the average equivalent periods increase with data width. The 1-of-4 pipelines have relatively shorter average periods than the 1-of-2 pipelines with the same data width, since the tree in the CD is one level deeper in 1-of-2 than in 1-of-4 pipelines. In most cases, the average periods of the DIRC pipelines are less than 1.5 times of the basic pipelines (as shown in Table II). Measured periods of the 128-bit wide DIRC 1-of-2 pipeline and the DIRC 1-of-4 pipeline are only 22% and 13% longer than their corresponding basic pipelines. Compared with the fault-tolerant designs proposed by Jang et al. [3] and Pontes et al. [17] whose average periods are 100% and 26% (at least) slower than basic ones, the overhead of DIRC pipelines is low.

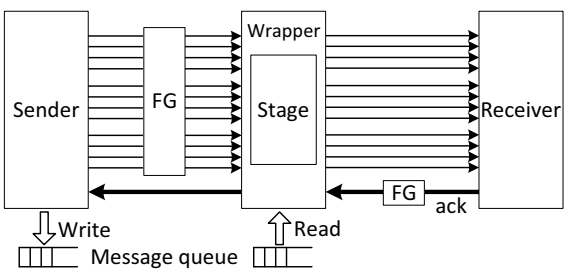

Figure 11. Test environment for DIRC QDI pipeline.

C. *Evaluation of Fault-tolerance*

In order to evaluate the fault-tolerant capability of the whole scheme (DIRC plus RPA), a SystemC test environment is built (as shown in Fig. 11). 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 simulated SystemC models. The sender has a random data generator, which produces random data to the stage. The sender also has a module like the 1-of-n adder to generate check words. Fault-generator generates random faults on all wires including the protected acknowledge wires. The stage wrapper checks the correctness of the outputting data and produces statistics. A shared message queue is used to store the error-free data being transmitted.

In this test environment, both positive and negative 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 produced independently. The mean interval between faults is set to 1μs while the duration of faults is randomized using a uniform distribution between 10ps and 2ns. This creates a more comprehensive and severer faulty environment than most existing literatures [3][15][17]. Data are transmitted

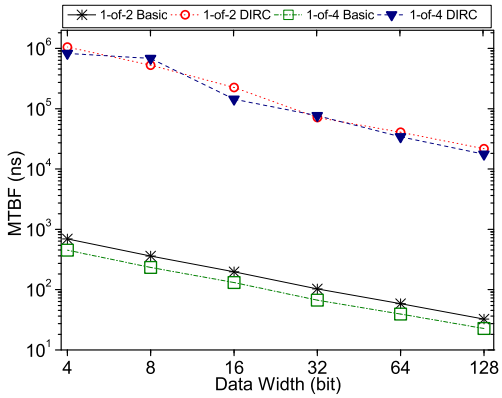

Figure 12. The comparison of MTBF between different pipelines.

continuously using the maximum injection rate. Totally one million data packets were transmitted during the simulation (due to the variable data rates, simulations are finished in 1.59ms to 7.19ms).

The plot in Fig. 12 shows the mean time between failures (MTBF) as a measure of fault-tolerance. Since the increasing number of wires lead to higher occurrence of faults, the MTBFs of all pipelines decrease with data width. When data width is 4-bit, the MTBFs of the DIRC 1-of-2 and 1-of-4 pipelines are 1525 and 1825 times longer than the basic 1-of-2 and 1-of-4 pipelines respectively. When data width rises to 128-bit, the times become 671 and 774 respectively. For the basic 1-of-4 pipeline with a data width of 128 bits, 280057 out of 810317 transient faults result in errors during the whole simulation, while only 331 out of 1119820 transient faults lead to errors when the DIRC code and the RPA technique is used. The resulting MTBF for the basic pipeline is 22.754 ns while it is prolonged to 17621.7 ns for the DIRC pipeline.

VII. Conclusions

This paper proposes a new efficient fault-tolerant DIRC coding scheme. Using a 1-of-n check word generated from multiple 1-of-n data words, 4-phase QDI interconnects can tolerate all 1-bit transient faults and some multi-bit transient faults. Thanks to its simple structure, this technique can be easily used to replace all or arbitrary stages in existing 1-of-n interconnects. A new redundant technique named RPA is also proposed to protect acknowledge wires from transient faults. Several DIRC and basic pipelines are implemented using the UMC 0.13μm standard cell library. Detailed experimental results show that the DIRC pipelines achieve hundred-folds fault-tolerant capability improvement in severe simulation environment with reasonable overhead. The 128-bit wide 1-of-4 DIRC pipeline is only 13% slower than the basic pipeline but the fault-tolerance is estimated more than 600 times stronger. Furthermore, the DIRC coding scheme can be extended to m-of-n QDI interconnects [19].

Acknowledgment

The authors gratefully acknowledge the contribution of the National Natural Science Foundation of China grant 61272144 and the China Scholarship Council.

References

[1] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," IEEE Micro, vol. 23, pp. 14-19, 2003.

[2] P. Song and R. Manohar, "Efficient failure detection in pipelined asynchronous circuits," in Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), 2005, pp. 484-493.

[3] W. Jang and A. J. Martin, "SEU-tolerant QDI circuits," in Proceedings of International Symposium on Asynchronous Circuits and Systems (ASYNC), 2005, pp. 156-165.

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

[5] N. Enright Jerger and L.-S. Peh, On-chip Networks, Morgan & Claypool Publishers, 2009.

[6] Y. Thonnart, P. Vivet, and 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.

[7] W. J. Bainbridge and S. J. Salisbury, "Glitch sensitivity and defense of quasi delay-insensitive network-on-chip links," in Proceedings of International Symposium on Asynchronous Circuits and Systems (ASYNC), 2009, pp. 35-44.

[8] M. Y. Agyekum and 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.

[9] M. Blaum and J. Bruck, "Unordered error-correcting codes and their applications," in Proceedings of International Symposium on Fault-Tolerant Computing (FTCS), 1992, pp. 486-493.

[10] S. Mukherjee, Architecture Design for Soft Errors, Morgan Kaufmann Publishers, 2008.

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

[12] F. L. Yang and R. A. Saleh, "Simulation and analysis of transient faults in digital circuits," IEEE Journal of Solid-State Circuits, vol. 27, pp. 258-264, 1992.

[13] D. Sorin, Fault Tolerant Computer Architecture, Morgan & Claypool Publishers, 2009.

[14] F.-C. Cheng and 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.

[15] S. Ogg, B. Al-Hashimi, and A. Yakovlev, "Asynchronous transient resilient links for NoC," in Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2008.

[16] W. Kuang, E. Xiao, C. M. Ibarra, and P. Zhao, "Design asynchronous circuits for soft error tolerance," in Proceedings of International Conference on Integrated Circuit Design and Technology (ICICDT), 2007, pp. 1-5.

[17] J. Pontes, N. Calazans, and P. Vivet, "Adding temporal redundancy to delay insensitive codes to mitigate single event effects," in Proceedings of International Symposium on Asynchronous Circuits and Systems (ASYNC), 2012, pp. 142-149.

[18] T. Verhoeff, "Delay-insensitive codes — an overview," Distributed Computing, vol. 3, pp. 1-8, 1988.

[19] W. J. Bainbridge, W. B. Toms, D. A. Edwards, and S. B. Furber, "Delay-insensitive, point-to-point interconnect using m-of-n codes," in Proceedings of International Symposium on Asynchronous Circuits and Systems (ASYNC), 2003.