State synchronization technique based on present input and healthy state for repairable TMR systems

Rui Yao, Jun Wu, Meiqun Wang, Xueyan Zhong, Ping Zhu, and Jiemei Liang

1 College of Automation and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 211106, China
2 Beijing Institute of Control Engineering, Beijing 100190, China
3 Nanjing Institute of Railway Technology, Nanjing, Jiangsu 210031, China

Abstract: We present a real-time state synchronization approach, called PIHS3TMR here, for the improvement of the real-time performance of the repairable triple modular redundancy systems. In our approach, the repaired module’s state synchronization with the other modules is performed by constructing its state directly according to the present input of the system and the present states of the fault-free modules. Experimental results show that, with very small hardware resource overhead and maximum frequency decline, the proposed approach can obtain state synchronization in one clock cycle, which is hundreds of times faster than state-of-the-art state restoration techniques for a Lion2 CPU. And there is no interruption or delay in system functionality during the synchronization process.

Keywords: state synchronization, present input, healthy state, TMR system

Classification: Integrated circuits

References

[1] Z. Yang, et al.: “A new automatic method for testing interconnect resources in FPGAs based on general routing matrix,” IEICE Electron. Express 12 (2015) 20150747 (DOI: 10.1587/elex.12.20150747).
[2] M. Ebrahimi, et al.: “Low cost scan-chain-based technique to recover multiple errors in TMR systems,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 21 (2013) 1454 (DOI: 10.1109/TVLSI.2012.2213102).
[3] K. S. Morgan, et al.: “A comparison of TMR with alternative fault-tolerant design techniques for FPGAs,” IEEE Trans. Nucl. Sci. 54 (2007) 2065 (DOI: 10.1109/TNS.2007.910871).
[4] S. Y. Yu and E. J. McCluskey: “On-line testing and recovery in TMR systems for real-time applications,” ITC Dig. Tech. Papers (2001) 240 (DOI: 10.1109/TEST.2001.966639).
[5] J. Losq: “A highly efficient redundancy scheme: self-purging redundancy,” IEEE Trans. Comput. C-25 (1976) 569 (DOI: 10.1109/TC.1976.1674656).
[6] H. Asadi and M. B. Tahoori: “Analytical techniques for soft error rate modeling
1 Introduction

Nowadays, embodied systems are used more and more widely in applications such as networks, spaceborne electronic systems, storage systems and process control systems. In particular, the systems based on field programmable gate arrays (FPGAs) have gained a steadily increasing interest in those applications because of their high performance, low nonrefundable-engineering and fast time-to-market [1]. However, such systems often have both timing constraints and fault tolerance requirements. E.g., with the technology down scaling towards nanometer area, soft errors caused by radiation effects, especially single event upsets (SEUs), have brought serious reliability challenges to such systems both in space applications and ground-level applications. Moreover, maintenance and repair are usually very expensive and time-consuming. Therefore, it is crucial to provide fault-tolerant techniques with minimum performance overhead for these systems [2].

Redundancy is the conventional approach to provide fault tolerance. In real-time systems, N-modular redundancy, duplex pairs and state machine encoding are frequently used. Among them, triple-modular redundancy (TMR) is the most well-known and widely utilized. And research results suggest that none of the other
techniques can provide greater reliability than TMR. Furthermore, the other techniques often require more resources [3].

Although TMR increases the system’s reliability by masking errors which occur at one module, it is incapable of fault recovery. So its reliability may be inferior to the dual redundant system and even the single module system when one module fails. Therefore, TMR is only effective for short time mission unless repair is possible [4]. To lengthen mission times, several techniques have been developed, e.g., self-purging redundancy [5], state restoration approaches [6, 7], etc. And for designs implemented on SRAM-based FPGAs, scrubbing mitigation [8, 9] and dynamic partial reconfiguration [10] can also be utilized. These techniques can repair the faulty circuit, but they may not restore the proper circuit state, so the repaired module will fail to synchronize with the fault-free modules.

The simplest approach for state synchronization of the repairable TMR system is to reset the entire system and let three modules restart together. However, to do this means to stop the present transaction, and the system cannot provide service during the process. So it is unfit for tight deadline applications. To avoid resetting the system, rollback recover technique can be exploited [11], in which the system’s states are backed up in some checkpoints during a program execution, and the system is restored to the previous fault-free checkpoint when an error is detected. Although rollback imposes performance improvement, it may violate the real-time requirements of safety-critical applications. Another method to obtain correct state is called roll-forward recovery, which copies correct state from a fault free redundant module to the repaired module to avoid re-computation [12]. However, when the state in the repaired module being restored, other modules are still operational and their states may have changed already, so the state synchronization cannot be obtained.

In order to synchronize the repaired module’s state with the other modules’, a modified roll-forward recovery approach for TMR systems has been proposed in [4], which combines roll-forward recovery with checkpoints. At checkpoints, upon detection of an error, all three modules stop their functionality, the system’s inputs are buffered, and the correct states are copied from the faulty-free modules to the repaired module. When the recovery process is finished, the buffered inputs will be processed. There is no re-computation delay introduced, and it seems that the present transaction is not interrupted owing to input buffering. However, large buffers are required to store the inputs; meanwhile, the system will not provide service actually during the state synchronization process. Therefore, this method cannot fulfill the real-time requirements of safety-critical applications, and it requires detailed information about the function of all registers of TMR modules. Therefore, a general purpose roll-forward error recovery technique based on multiple scan chains for TMR systems has been proposed in [13], called ScTMR. Thereafter, a modified version of ScTMR, called SMERTMR, has been presented in [2] to locate and remove as well latent faults that cause mismatch between the internal states of the TMR modules.

In general, the state synchronization for TMR systems in [2, 4, 13] are all initiated at checkpoints: either at the present checkpoint by buffering the present
inputs of the system [4], or at a certain checkpoint after real-time transactions being finished [2, 13]. Most important of all, the systems cannot provide service during the state synchronization process, thus the system’s functionality will be interrupted or delayed. Therefore, all these approaches cannot meet the critical mission deadline requirement for real-time applications.

In this paper, a real-time present-input and healthy-state based state synchronization technique for TMR systems, called PIHS3TMR here, is presented. In PIHS3TMR, there is no need to set any checkpoints, and the circuit recovery and state synchronization process are initiated immediately whenever a fault is detected. Moreover, the system can provide normal service during the state synchronization process, so there will be no interruption or delay in system functionality. In addition, no buffers are needed to store the system’s inputs, since the repaired module’s state is constructed according to the present input and the present states of the fault-free modules directly. Therefore, the time overhead is negligible. So PIHS3TMR technique is qualified for real-time applications with tight deadline requirement.

2 Proposed state synchronization technique

2.1 Review on the role of state synchronization in repairable TMR systems

The model of the general repairable TMR systems is shown in Fig. 1 indicated by full lines. The core functionality of the system is performed by a TMR structure composed of three repairable modules named Module 1, Module 2 and Module 3, as well as a voter; the system’s input $X$ is fed to all three Modules; and its output $Y$ comes from the voter by voting the outputs of three Modules, i.e., $Y_1$, $Y_2$ and $Y_3$. The fault detector (FD) is responsible for fault detection and location. And the repair controller (RC) performs repair-controlling. In operation, if a fault occurs at a certain module, the FD will detect it and give the information to the RC; and the RC will repair the faulty module, until it recovers from fault.

To date, the faulty module’s repair is often performed by partial scrubbing [9], dynamic partial reconfiguration [10, 14] or evolvable hardware [15, 16], etc. But all these approaches can only recover the logic of combinational circuits in the system; they cannot insure the state of sequential circuits of the repaired module is synchronized with those of the other modules. Failing to do so means that the repaired module cannot yet keep up with the pace of the fault-free modules, thus the system cannot regain the original fault-tolerant capability.

![Fig. 1. Model of the repairable TMR systems.](image-url)
To obtain state synchronization, synchronization control circuits and signals should be added, as is shown in Fig. 1 indicated by dotted lines. And the entire workflow of the system is shown in Fig. 2. Once an error at a certain module is detected, the circuit of the faulty module is repaired at first, and then the state of the repaired module is synchronized with the other modules.

2.2 Proposed state synchronization technique

2.2.1 Model of the sequential logic circuits

The sequential circuit is capable of remembering, and its output is determined by both the present input and the previous output. As is shown in Fig. 3, most of the sequential circuits can be modeled as state machines composed of combinational circuits (CLCs) and flip-flops (FFs). In Fig. 3, where \( X = [x_0, x_1, \ldots, x_i] \) and \( Z = [z_0, z_1, \ldots, z_j] \) represent the input and the output of the circuit, respectively. \( D = [d_0, d_1, \ldots, d_k] \) is the excitation input of FFs, and \( Q = [q_0, q_1, \ldots, q_m] \) is the state of the circuit, i.e., the state of FFs. And \( CP = [cp_0, cp_1, \ldots, cp_n] \) is the clock input of the circuit. Then the relationship among \( Z, D, Q \) and \( X \) can be described as

\[
Z^n = F_1(X^n, Q^n). \tag{1}
\]

\[
D^n = F_2(X^n, Q^n). \tag{2}
\]

\[
Q^{n+1} = F_3(D^n, Q^n). \tag{3}
\]

Where \( n \) and \( n + 1 \) represent the \( n \)th clock cycle (CC) and the \((n + 1)\)th CC, respectively. \( Z^n, D^n \) and \( X^n \) represent the values of \( Z, D \) and \( X \) at the \( n \)th CC, respectively. \( Q^n \) is the FFs state at the \( n \)th CC, called the present state; and \( Q^{n+1} \) is the FFs state at the \((n + 1)\)th CC, called the next state. Eq. (1) is the expression of the output signals, called output equation; Eq. (2) is the expression of the FFs input signals, called excitation equation; and Eq. (3) is the expression of the FFs state in the next CC, called state equation. Obviously, \( Z \) and \( D \) are outputs of CLCs, so their present values are determined by the present input of CLCs; while \( Q \) is the output of FFs, so its next state \( Q^{n+1} \) is not only determined by the present FFs input \( D^n \), but also determined by the present state \( Q^n \).

Current repairable TMR systems can recover the circuit of the faulty module, but they cannot ensure the accordance of the repaired module’s present state to the faulty-free modules. So state restoration and synchronization are necessary.
2.2.2 Proposed state synchronization approach

To ensure the fault-free modules work as usual and provide normal service during the state restoration process, a state synchronization approach based on the present input and healthy state of the system for repairable TMR systems, called PIHS3TMR, is proposed, as is shown in Fig. 4.

In Fig. 4, the state machine of each module, denoted as Module $y$, is modified by introducing the healthy present state $Q_h$, the synchronization control signal $SC_y$ and 2-to-1 multiplexers (MUXs). And the state input signal $Q$ can be selected by MUXs from the native present state $Q_y$ and the healthy present state $Q_h$ under the control of the state synchronization control signal $SC_y$, as is shown in Eq. (4).

$$ Q = SC_y Q_h + \overline{SC_y} Q_y. $$  \hspace{1cm} (4)

During the normal operation process, the value of $SC_y$ is 0, and $Q_y$ is selected as the input of CLCs, so Module $y$ works independently according to its own state machine. Whereas in case that Module $y$ is repaired and state synchronization is required, the value of $SC_y$ is 1, and $Q_h$ is selected as the input of CLCs. Then the next state of Module $y$ is constructed immediately according to the healthy present state and the present operational input; thus the state of Module $y$ can synchronize with the other modules. Moreover, the modules without fault work as usual during the entire circuit recover and state synchronization process, and offer normal service all the time; hence the system’s functionality is not interrupted at all.

In Fig. 4, the healthy state can either be the voted result of the present states of three modules, or come from any of the present states of fault-free modules. However, inserting a voter right after each FF will impose significant area overhead.

---

Fig. 3. Model of the sequential logic circuit.

Fig. 4. Proposed state synchronization technique
to the circuit, and will increase the delay of the critical path. Therefore, the healthy state of each FF in one module adopted in this paper is from the present state of the corresponding FF of the neighbor module, which can be described as

\[ Q_{h1} = Q_2. \]  
\[ Q_{h2} = Q_3. \]  
\[ Q_{h3} = Q_1. \]

where \( Q_{hi} \) (i = 1, 2, 3) represents the healthy state of Module \( i \).

3 Results and discussions

The design of asynchronous sequential circuits is more complex than synchronous sequential circuits because of the clock domain crossing issues. Therefore, in this section, three synchronous sequential circuits, i.e., a mod16 hexadecimal up counter, a 16-word by 8-bit FIFO and a SpaceWire codec are used as case studies to verify the effectiveness of the proposed state synchronization technique. And the state synchronization time comparison of the proposed technique with state-of-the-art techniques for a Lion2 CPU is given.

3.1 Design results of mod16 up counter

The state transition diagram of a mod16 up counter with enable signal is shown in Fig. 5. The proposed PIHS3TMR system design for this counter is modeled by using very-high-speed integrated circuit hardware description language (VHDL). The simulation result in Xilinx’s ISE software is shown in Fig. 6. The implementation result comparison of the traditional TMR and the PIHS3TMR on a Xilinx Virtex-5 FPGA is shown in Table I.

In Fig. 6, \( clk \) is the clock input, \( enable \) is the enable signal for counting; \( rst1-3, sc1-3 \) and \( zq1-3 \) are reset signals, state synchronization control signals and state output signals of Module 1–3, respectively; and \( rt \) is the voted state output signal. As can be seen from Fig. 6, in the 1st to 8th CCs, \( enable \) is valid, so counting is allowed. In CCs 1–5, three modules work normally, so the state of each module transforms according to the rule of \( 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \). In CC 6, \( rst1 \) is valid, which simulates the initial state \( zq1 = 0 \) of Module 1 after circuit repairing; \( zq2-3 \) then have transformed to 6; obviously, the state of Module 1 is inconsistent with other modules. However, with the proposed state synchronization technique, in CC 7, \( sc1 \) is valid; therefore, \( zq1 \) jumps into 7 from 0 immediately according to the healthy state \( zq2 \), and the value of \( enable \) in CC 6; so the states of all three modules are the same. And in CC 8, the values of \( zq1-3 \) all jump into 8. In CCs 9–10, \( enable \) is invalid, so the values of \( zq1-3 \) are intended to remain unchanged; but \( rst2 \) is valid in CC 10, so \( zq2 \) is reset to 0, which is inconsistent with \( zq1 \) and \( zq3 \). In CCs 11–14, \( enable \) is valid once again. Hence in CC 11, the values of \( zq1-3 \) all increase by 1; but \( zq2 \) is still inconsistent with \( zq1 \) and \( zq3 \). In CC 12, \( sc2 \) is valid, hence \( zq2 \) jumps into A from 1 immediately, thus synchronizing with \( zq1 \) and \( zq3 \). And in CCs 13–14, three modules work normally, and the values of \( zq1-3 \) increase by 1 in each CC. Most important of all, the counter offers normal service during the entire work process, without any interruption in system functionality.
As can be seen in Table I, comparing with the traditional TMR system implementation, the utilization of BUFG/BUFGCTRLs and registers of the PIHS3TMR system implementation are the same, and the number of LUTs used is about 31.6% more. The reason for more number of LUTs is that more logic resources (here 6 LUTs) are required to implement the MUXs. In addition, the maximum frequency of the PIHS3TMR system implementation is only a little slower, i.e., about 4% slower than that of the traditional one.

### Table I. Implementation results comparison for the counter

| Categories             | Available | Traditional TMR | Proposed TMR |
|------------------------|-----------|----------------|--------------|
|                        |           | Used | Utilization | Used | Utilization |
| Registers              | 44800     | 12  | 0%          | 12  | 0%          |
| LUTs                   | 44800     | 19  | 0%          | 25  | 0%          |
| BUFG/BUFGCTRLs         | 32        | 1   | 3%          | 1   | 3%          |
| Maximum frequency (MHz)| -         | 439.754 |         | 422.119         |

#### 3.2 Design results of the 16-word by 8-bit FIFO

FIFOs are commonly used in electronic circuits for buffering and flow control. A synchronous FIFO primarily consists of a dual-port RAM (DPRAM), a Status module, a write counter (WRC), and a read counter (RDC), as shown in Fig. 7. The DPRAM is used for storing buffering data; WRC and RDC are utilized to generate write and read address, respectively; and the Status is used to generate flag signals, including full, empty, as well as fc.dout that indicates the number of data unread. The other signals of the FIFO include: the global clock and reset signals clk and rst, the write and read enable signals wren and rden, as well as the data input and output signals wr.din and rd.dout.
The proposed PIHS3TMR system design for a 16-word by 8-bit FIFO is modeled by using VHDL, and the simulation results in ISE are shown in Fig. 8. The comparison between implementation results of the traditional TMR and the PIHS3TMR on a Virtex-5 FPGA is shown in Table II.

In Fig. 8, clk, rst1-3, sc1-3 are the same as those in Fig. 6; wren and rden are the enable signals for write and read operations; fc_dout1-3 are the numbers of unread data in FIFOs of Module 1–3, respectively; wr_din is the datum to be written; rd_dout1-3 and rd_dout are data read from FIFOs of Module 1–3 and the voted result, respectively; wr_ct and rd_ct are values of WRC and RDC of Module 1; and wr_count and rd_count are healthy states of wr_ct and rd_ct. As can be seen in Fig. 8, all three modules work normally in CCs 1–4. In CCs 1–2, wren is valid, so 2 data are written to FIFOs, and in each CC: wr_ct, wr_count, fc_dout1-3 increase by 1; rd_dout1-3 and rd_dout are unknown because of rden being invalid. In CCs 3–4, wren and rden are both valid, another 2 data are written to FIFOs, and 2 data are read from FIFOs. And in each CC, rd_dout1-3 and rd_dout are uniform, fc_dout1-3 remain unchanged, and wr_ct, wr_count, rd_ct and rd_count increase by 1. In CC 5, rst1 is valid, which simulates the initial states of Module 1 after circuit repairing: fc1_count, wr_ct and rd_ct are all reset to 0. At this time, since wren is valid, another datum is written to FIFOs of Module 2–3, so fc_count2-3 and wr_count increase by 1; obviously, the state of Module 1 is inconsistent with the other modules. In CC 6, wren is valid, another datum is written to FIFOs of three
modules (note that the address of Module 1 is different from Module 2–3), so \( fc\_dout1-3 \), \( wr\_ct \) and \( wr\_count \) all increase by 1; and the state of Module 1 is still inconsistent with other modules. Thanks to the proposed state synchronization technique, in CC 7, \( sc1 \) is valid; hence the state of Module 1 synchronizes with that of Module 2 and 3: \( fc\_dout1 \) jumps to 5 from 1, \( wr\_ct \) jumps to 7 from 1, and \( rd\_ct \) jumps to 2 from 0; and the states of all three modules are the same. In CCs 8–9, another 2 data are written to the same address. And in CCs 10–16, 7 data are read from FIFOs, since \( wren \) is invalid and \( rden \) is valid. Values of \( rd\_dout1-3 \) are the same in CCs 10–12 and in CCs 15–16; the reason for this is that the data written to FIFOs of three modules are the same before reset and after state synchronization. And values of \( rd\_dout1 \) in CCs 13–14 are unknown because Module 1 is not synchronized with other modules in CCs 5–6, so no data have been written to these cells. Moreover, the FIFO offers normal service during the entire operation process, without any interruption in system functionality.

Table II. Implementation results comparison for the FIFO

| Categories          | Available | Traditional TMR | PIHS3TMR |
|---------------------|-----------|-----------------|----------|
|                     |           | Used          | Utilization | Used       | Utilization |
| Registers           | 44800     | 66            | 0%         | 63         | 0%          |
| LUTs                | 44800     | 125           | 0%         | 164        | 0%          |
| BUFG/BUFGCTRLs      | 32        | 1             | 3%         | 1          | 3%          |
| Maximum frequency (MHz) | -       | 329.722       |            | 287.439    |             |

As can be seen in Table II, comparing with the traditional TMR system implementation, the utilization of BUFG/BUFGCTRLs of the PIHS3TMR system implementation is the same, the number of slice LUTs used is about 31.2% more, and the number of slice registers is about 4.5% less. In addition, the maximum frequency of the proposed TMR system implementation is about 12.8% slower than that of the traditional one.

3.3 Design results of the SpaceWire codec

In space missions, a huge mass of data interchanges are performed between the master computer and the peripheral equipment or the bulk memory, and some control commands are also required to transfer via the same communication network, so as to reduce system mass and complexity. Therefore, high-speed, strong radiation-tolerance and high reliability are demanded for the communication network. SpaceWire can provide high-speed (2–200 Mbits/s), bi-directional, full-duplex data links and networks to connect the SpaceWire enabled equipment using point to point data links and routing switches. SpaceWire codec is the atomic component of the network. It provides the interface for all the SpaceWire enabled equipment and the SpaceWire routing switches.

A SpaceWire codec often consists of a transmitter (Tx), a Receiver (Rx), a state machine (SM), a transmitter FIFO (TxFifo) and a receiver FIFO (RxFifo), as shown in Fig. 9. The Rx decodes the link data and sends it to the RxFifo. The RxFifo receives the user data from the Rx and transfers it to the host interface. The Tx
codes the user data and transmits it to the link. The TxFifo transmits the user data from the host interface to the Tx. The SM controls the overall operation of the link interface. The proposed PIHS3TMR system design for a SpaceWire codec is modeled by using VHDL, and the implementation result comparison with the traditional TMR system design is shown in Table III.

![Structure diagram of the SpaceWire codec.](image)

As can be seen in Table III, comparing with the traditional TMR system implementation, the utilization of BUFG/BUFGCTRLs, Block RAM/FIFO and registers of the PIHS3TMR system implementation are the same, and the number of slice LUTs used is about 6.9% more. In addition, the maximum frequency of the proposed TMR system implementation is only about 4.2% slower than that of the traditional one. Obviously, the hardware resource overhead and the maximum frequency decline of the proposed state synchronization technique is negligible. The reason for this is that, for practical applications, the sequential logic circuit frequently occupies only a small portion of the total amount of hardware resources.

| Categories     | Available | Traditional TMR | PIHS3TMR |
|----------------|-----------|-----------------|----------|
|                | Used      | Utilization     | Used     | Utilization |
| Registers      | 44800     | 411 0%          | 411 0%   |
| LUTs           | 44800     | 799 1%          | 854 1%   |
| Block RAM/FIFO | 148       | 2 1%            | 2 1%     |
| BUFG/BUFGCTRLs | 32        | 1 3%            | 1 3%     |
| Maximum frequency (MHz) | -          | 334.728         | 321.203  |

3.4 Comparison of the proposed technique with state-of-the-art techniques

The performance comparison of the proposed PIHS3TMR technique with the previous techniques in [13], [2] and [4], is shown in Table IV. Where \( T_{\text{clk}} \) is the clock period of the system, \( L_{\text{sc}} \) is the number of FFs in the scan-chain, \( N_{\text{reg}} \) is the number of registers to be restored, and \( N_{\text{FF}} \) is the number of FFs in the circuit.

As can be seen in Table IV, PIHS3TMR has the best real-time performance among all techniques. The ideal state synchronization time (SST) of PIHS3TMR is only one CC for any systems, whereas the SSTs of the other techniques depend on the number of FFs or registers in the system. For a system with 2096 FFs in each module (the CPU of Lion2 processor [2]), 131 and 262 CCs are required for ScTMR and SMERTMR, respectively, when multiple scan chains with 16-bit width.
are utilized [2]; whereas 2096 and 4192 CCs are required, respectively, if a single
scan chain is adopted [2]. For RFTMR technique, 131 and 2096 CCs are required
when the register width is 16 bits and 1 bit, respectively. However, only one CC
is required for PIHS3TMR technique. Obviously, synchronization speed of
PIHS3TMR is hundreds of times faster than the other techniques. In addition,
the area overhead of PIHS3TMR is not more than other techniques, so its
scalability is good. Most important of all, systems that adopt PIHS3TMR technique
can provide normal service during the state synchronization process, without any
interruption or delay in system functionality.

| Table IV. Performance comparison of the techniques |
|-----------------------------------------------|
| Requirement of checkpoint | ScTMR [13] | SMERTMR [2] | RFTMR [4] | PIHS3TMR |
| Requirement of buffers | Yes | Yes | Yes | No |
| Requirement of buffers | No | No | Yes | No |
| Interruption or delay in system functionality | Interruption | Interruption | Delay | None |
| Time for state restore | $L_{sc} \times T_{clk}$ | $2L_{sc} \times T_{clk}$ | $N_{reg} \times T_{clk}$ | $T_{clk}$ |
| Extra resource overhead | $3N_F MUXs$ | $3N_F MUXs + N_{MUXs} \times N_{voters}$ | $3N_F MUXs + N_{MUXs} + N_{buffers}$ | $3N_F MUXs$ |

4 Conclusion

A real-time state synchronization approach called PIHS3TMR based on the present
input and the health state of the system for repairable TMR systems is presented. A
mod16 hexadecimal up counter, a 16-word by 8-bit FIFO and a SpaceWire codec
are used as case studies. Experimental results show that, comparing with the
traditional TMR system designs, the PIHS3TMR system designs have a resource
overhead of about 30% slice LUTs and a maximum frequency decline of less than
13% for applications where sequential logic circuits are dominant, such as the
counter and the FIFO; whereas the resource overhead and the maximum frequency
decline are very small for practical applications where sequential logic circuits
occupies only a small portion of total resources. Furthermore the PIHS3TMR
system can obtain state synchronization in one clock cycle, which is hundreds of
times faster than state-of-the-art state restoration techniques for a Lion2 CPU.
Moreover, the system can provide normal service during the state synchronization
process, so there is no interruption or delay in system functionality.

In summary, PIHS3TMR is a simple but effective technique. It can serve as a
common technique for repairable TMR systems to obtain state synchronization
after the faulty module’s circuit repairing. As for the fault model, the hardening of
the RC and the voters, we will consider further in our future research.

Acknowledgments

This work was supported by National Natural Science Foundation of China
(No. 61402226) and the Civil Space “12th Five-year” plan Pre-research Project
of Science and Technology.