Concurrent Error Correction in Iterative Circuits by Recomputing with Partitioning and Voting

Hussain Al-Asaad and Edward Czeck

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

This paper presents a novel technique for the design of iterative circuits with concurrent error correction capabilities. The new method is called "recomputing with partitioning and voting" (RWPV). It uses a combination of hardware and time redundancy to achieve fault tolerance while providing the same error correction capabilities as found in hardware TMR or time redundancy computation. RWPV error correction is obtained with small hardware and time overhead, as compared to over 200% overhead in either hardware or time for TMR or time redundancy.

1 Introduction

To achieve fault tolerance, redundancy is often used in the form of hardware, software, information, or time. The most common form of redundancy is hardware. For example; duplication with comparison (DWC) can detect an error while triple modular redundancy (TMR) can correct an error [1]. Hardware redundancy impacts physical weight, size, power consumption, and cost. DWC and TMR methods use at least 100% and 200% hardware redundancy respectively.

To overcome these difficulties of hardware redundancy, time redundancy has received much attention [2-5]. The main idea of time redundancy is to reduce the amount of extra hardware at the expense of performance.

The paper is divided into the following sections: Section 2 presents concurrent error detection/correction techniques, their associated overheads in redundancy, and their limitations in fault coverage. Section 3 discusses a timing model used to evaluate our technique. Section 4 details our new scheme -- Recomputing with Partitioning and Voting -- RWPV. Section 5 presents results of a timing simulation for some common circuits with RWPV applied. Finally, Section 6 concludes the paper.

2 Concurrent error checking

Hardware concurrent error detection duplicates the system and then compares the results to detect errors. Time redundant concurrent error detection repeats the computation, then compares the results to detect the errors. To detect permanent faults using time redundancy, the repeated computation is performed differently.

Various techniques which use time redundancy are: alternating logic [2], recomputing with shifted operands (RESO) [3] and recomputing with swapped operands (RESWO) [4]. The differences in the above techniques are their encoding and decoding functions. Recently, a new technique, recomputing using duplication with comparison (REDWC), was developed for iterative circuits. REDWC has been proved to be a good technique for error detection[5]. The method which accomplishes error detection resembles that of duplication with comparison. A time redundant calculation completes the operation and obtain the final result.

The error detection schemes described above use either hardware or time redundancy to detect errors. Table-1 shows the associated overhead for each of these schemes.

Table 1 Redundancy used in different error detection and correction methods.

Scheme	HARDWARE redundancy	TIME redundancy
Error detection schemes		
Alternating Logic	0%-100%	>100%
RESO	≈0%	>100%
RESWO	0%-100%	0%-100%
REDWC	0%-100%	0%-100%
DWC	>100%	≈0%
Error correction schemes		
TMR	>200%	≈0%
RESO	≈0%	>200%

A typical error correction hardware scheme is TMR with its disadvantage of high hardware redundancy -- at least 200%. On the other hand, RESO has limited capabilities of error correction by using at least 200% time redundancy. Table-1 shows the associated overhead for each of the above error correction techniques.

To overcome the weaknesses of the above approaches in error correction, an extension of REDWC is presented

0-8186-3830-3/93 $03.00 © 1993 IEEE

(N/3)units. Each primary input vector A is also divided into three (N/3) bit parts AL, AM and AH. In addition, we need the following:

1) 3-to-1 Multiplexer (MUX1) for each primary input of (IT) to select between L, M, and H parts.
2) 2-to-1 Multiplexer (MUX2) for each secondary input entering ITL. Similarly for ITM and ITH.
3) A Latch (LC) for each secondary output of ITL. Similarly for ITM and ITH.
4) (N/3)-bit voter (VOTER) for each primary output of IT.
5) One bit voter (V) for each secondary output of ITL. Each (V) will vote among a secondary output of ITL, ITM and ITH.
6) Two (N/3) bit latches (LATCH) for each primary output of IT to store the first (SL) and the second (SM) parts of the result.

The complete computation is done in three phases:

PHASE 1: The L part of primary inputs is selected to enter the three parts of the iterative circuit ITL, ITM and ITH. External values are applied to the secondary input of each of ITL, ITM and ITH. The primary outputs of ITL, ITM and ITH are voted and the result is then saved in the latch of the lowest part to get (SL).

PHASE 2: The M part of primary inputs is selected to enter the three parts of the iterative circuit ITL, ITM and ITH. In this phase, the secondary outputs of ITL, ITM and ITH from the first phase are used as secondary inputs to ITL, ITM and ITH. This justifies the need of latches at the secondary output of each ITL, ITM and ITH. The primary outputs of ITL, ITM and ITH enter the voter to vote on the results. The voter result is saved in the latch of the medium part to get (SM).

PHASE 3: In this phase, the same happens as phase 2, but the output of the voter is not latched because the complete result of the computation is ready at the output. The result of (V) is used in this phase as the secondary output of IT.

The computing time using RWPV per phase is:

$$T_2 = T_1[(N/3)-1]+\max\{\beta_{co}+\lambda_{co},\beta_s+\phi_s\}$$

where T_1 is defined in equation (1).

Hence the computing time of each phase of RWPV is:

$$T_P = T_2+\max\{T_{m1},T_{m2}\}+T_v$$

where T_{m1} is the propagation delay of the primary input multiplexers (MUX1), T_{m2} is the propagation delay of the secondary input multiplexers (MUX2) and T_v is the delay of the voter (VOTER).

Note that the delay in phase *i* of the storage latches (LATCH) and those of the latches at the secondary outputs (LC) is overlapped with the next phase *i+1*. Hence the computing time of RWPV iterative circuit is:

$$T_{RWPV} \approx 3*T_p$$

Then the time redundancy in using RWPV as compared to a non-redundant implementation is:

$$[100*(3*\max\{T_{m1},T_{m2}\}+3*T_v+2*\max\{\beta_{co},\alpha_{co}\}+2*\lambda_{co}+2*\max\{\beta_{co}+\lambda_{co},\beta_s+\phi_s\}-4*(\beta_{co}+\lambda_{co}))] / [\max\{\beta_{co},\alpha_{co}\}+\lambda_{co}+(N-2)*[\beta_{co}+\lambda_{co}]+\max\{\beta_{co}+\lambda_{co},\beta_s+\phi_s\}] \quad (4)$$

From the above equation, we can see that the time redundancy is inversely proportional to N.

The advantages of the above model is its use in RWPV iterative circuit design. Iterative circuits can be designed in different forms and using the delay model we can determine which design form is better under RWPV.

RWPV can correct any single error in the iterative circuit, input multiplexers, and the one bit latches. Moreover, RWPV can correct multiple errors if the errors occur in one part of the circuit, i.e the L, M, or H.

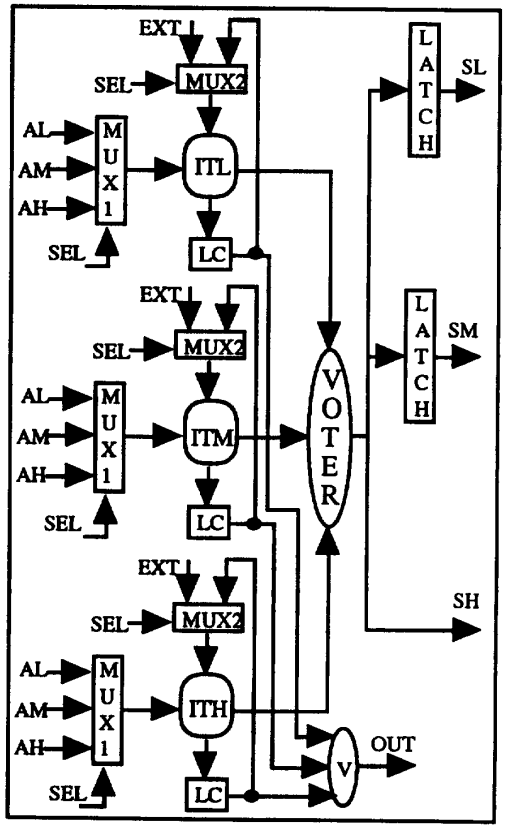

Figure 3 Structure of RWPV circuit.

5 Simulation

RWPV was simulated using VHDL for three different iterative circuit designs. These are:

which allows error correction capability. Our new approach is called recomputing with partitioning and voting (RWPV). It achieves error correction with typically less than 200% overhead by combining hardware and time redundancy. A description of RWPV follows in Section 4.

3 Circuit model

An iterative circuit consists of N identical units connected in a linear form. The inputs and outputs of the iterative circuit are divided into two major types: primary inputs (outputs) and secondary inputs (outputs). The secondary outputs of unit *i-1* become the secondary inputs of unit *i*. Unit *i* cannot perform its operation until unit *i-1* has finished.

A delay model of iterative circuits, Figure-1, is developed to understand the propagation delays in such circuits. It has six parameters which can be extracted from the design of a unit cell. These parameters are:

α_{co} The minimum time needed for the primary input data to propagate towards the secondary output until reaching a point which depends on a secondary input.

β_{co} The minimum time needed for the secondary input data to propagate towards the secondary output until reaching a point which depends on a primary input.

α_s The minimum time needed for the primary input data to propagate towards the primary output until reaching a point which depends on a secondary input.

β_s The minimum time needed for the secondary input data to propagate towards the primary output until reaching a point which depends on a primary input.

λ_{co} Is equal to the maximum propagation delay from (A secondary input to secondary output minus β_{co}) or (A primary input to secondary output minus α_{co}).

ϕ_s Is equal to the maximum propagation delay from (A secondary input to primary output minus β_s) or (A primary input to primary output minus α_s).

The computation time delay using the above circuit model is derived as follows:

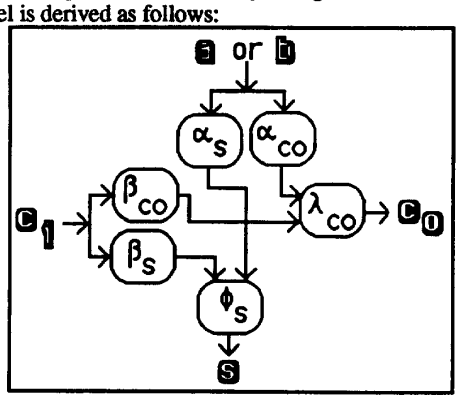

Figure 1 Iterative circuit delay model.

Let $T_1(i)$ be the time needed for the secondary output of unit *i* to settle after an input change. Then,

$$T_1(i)=(\max\{\beta_{co},\alpha_{co}\}+\lambda_{co})+(i-1)*(\beta_{co}+\lambda_{co}) \quad (1)$$

The total computing time of the iterative circuit is:

$$T= T_1(N-1)+\max\{\beta_{co}+\lambda_{co},\beta_s+\phi_s\} \text{ If } \alpha_s<T_1(N-1)+\beta_s \quad (2)$$

$$T= \max\{\beta_{co}+\lambda_{co}+T_1(N-1),\alpha_s+\phi_s\} \text{ If } \alpha_s>T_1(N-1)+\beta_s \quad (3)$$

where N is the number of elements in the iterative circuit. In most cases, the value of α_s is much smaller than $T_1(N-1)+\beta_s$ since $T_1(N-1)$ is a linear function of N -- later stages are waiting for the secondary inputs rather than the primary inputs. Hence equation 2 is often used.

Consider the example of ripple carry adder; a unit of this adder is shown in Figure-2. The model parameters are extracted from the circuit using the previous definitions. If the delay of a 2-input XOR gate is 8 ns and a 2-input NAND gate is 5 ns then we have:

$$\alpha_{co}=8 \text{ ns}, \beta_{co}=0 \text{ ns}, \alpha_s=8 \text{ ns}.$$

$$\beta_s=0 \text{ ns}, \lambda_{co}=10 \text{ ns}, \phi_s=8 \text{ ns}.$$

Hence the computing time of this iterative circuit without RWPV is:

$$T = 8 \text{ ns} + 10 \text{ ns} + (N-2)*10 \text{ ns} + 10 \text{ ns} = 8 \text{ ns} + N*10 \text{ ns}.$$

This result can be verified easily from the circuit or by circuit simulation.

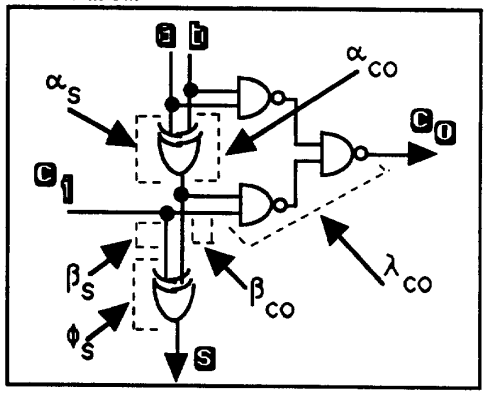

Figure 2 Ripple carry adder cell.

4 New method - RWPV

Our method divides the iterative array and input data into three parts. Each third of the array executes three times over the data part. Since the delay through the array is a linear function of array length, and the time required to perform one-third of the operation three times is approximately equal to performing the operation one time, this procedure requires little time overhead. Figure-3 shows the structure of RWPV while the following paragraphs describe the operation.

Consider an iterative circuit (IT) of N units, partitioned into three parts ITL, ITM and ITH each of which is

A) N bit Ripple carry adder (*RIPP ADD*).

B) N bit Adder formed of 4-bit carry lookahead adders (Standard 74LS83) and the carry ripples between carry lookahead adders (*FAST ADD*).

C) N-bit ALU formed of 4-bit ALU (Standard 74LS181) and the carry ripples between carry lookahead ALU's (*FAST ALU*)

The time redundancy of the RWPV for the three designs is shown in Table-2, while the hardware redundancy is shown in Table-3. For the sake of comparison, the time and hardware redundancies of a TMR implementation are shown in Tables 4 and 5 respectively.

Table 2 Percent time redundancy for RWPV iterative circuits for different values of N.

N	12	24	36	48	96
RIPP ADD	25.00	12.50	8.33	6.25	3.13
FAST ADD	33.33	17.39	11.76	8.89	4.49
FAST ALU	264.71	150.00	104.50	82.14	42.59

Table 3 Percent hardware redundancy for RWPV iterative circuits for different values of N.

N	12	24	36	48	96
RIPP ADD	243.14	218.63	210.46	206.37	200.25
FAST ADD	147.62	132.74	127.78	125.29	121.58
FAST ALU	79.48	71.47	68.80	67.47	65.46

Table 4 Percent time redundancy for TMR iterative circuits for different values of N

N	12	24	36	48	96
RIPP ADD	9.17	4.58	3.06	2.29	1.15
FAST ADD	15.28	7.97	5.39	4.07	2.06
FAST ALU	21.57	12.22	8.53	6.55	3.40

Table 5 Percent hardware redundancy for TMR iterative circuits for different values of N.

N	12	24	36	48	96
RIPP ADD	252.94	252.94	252.94	252.94	252.94
FAST ADD	232.14	232.14	232.14	232.14	232.14
FAST ALU	217.31	217.31	217.31	217.31	217.31

From the simulation data, we conclude the following:

1-The hardware redundancy of the TMR iterative circuit is independent of N, decreases as the complexity of the iterative circuit increases, and is always greater than 200%.

2-The time redundancy of TMR iterative circuit decreases as N increases.

3-The hardware redundancy of an RWPV iterative circuit decreases as N increases or as the complexity of the iterative circuit increases.

4-The time redundancy of an RWPV iterative circuit is inversely proportional to N and depends on the actual design of the iterative circuit unit, (i.e, the parameters of the iterative circuit model described).

As a suggestion for evaluating RWPV in a single figure of merit, the parameter ρ is defined as:

$$\rho = \frac{\text{RWPV hardware overhead + RWPV time overhead}}{\text{TMR hardware overhead + TMR time overhead}}$$

Table-6 shows the values of ρ for the cases studied. ρ is always less than 1, except when the array size is small. Hence, the redundancy of an RWPV iterative circuit is typically less than that of the TMR.

Table 6 Values of ρ as a function of array size.

N	12	24	36	48	96
RIPP ADD	1.023	0.897	0.855	0.833	0.800
FAST ADD	0.732	0.626	0.587	0.568	0.538
FAST ALU	1.441	0.965	0.768	0.668	0.490

6 Conclusion

A new approach for concurrent error correction in iterative circuits was presented and evaluated. The technique allows for the on-line detection and correction of errors in an iterative array circuit. The advantage of this technique is that error correction is achieved with less than 200% redundancy in both time and hardware. Moreover, the time and redundancy overheads have been shown to decrease as the complexity and delay of the array increases.

References

[1] Siewiorek, D., Swarz, R., *Reliable Computer Systems: Design and Evaluation*, Digital Press, 1992.

[2] Reynolds, D., Metze, G. "Fault Detection Capabilities of Alternating Logic", *IEEE Transactions on Computers*, Vol C-27, No. 12, pp. 157-162, December 1978.

[3] Patel, J., Fung, L. "Concurrent Error Detection in ALU's by Recomputing with Shifted Operands", *IEEE Transactions on Computers*, Vol C-31, pp.417-422, July 1982.

[4] Hana, H., Johnson, B. "Concurrent Error Detection in VLSI Circuits Using Time Redundancy", *Proc. IEEE Southeastcon*, March 23-25, 1986, pp. 208-212.

[5] Johnson, B., Aylor, J., Hana, H. "Efficient Use of Time and Hardware Redundancy for Concurrent Error Detection in a 32-bit VLSI Adder", *IEEE Journal of Solid-State Circuits*, Vol. 23, No. 1, pp. 208-215, February 1988.

[6] Johnson, B., *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, 1989.

[7] Hsu, Y., Swartzlander, E. "Time Redundant Error Correcting Adders and Multipliers", *IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, pp 247-254, 1992, Published after initial submission of this paper.