

Efficient Time Redundancy for Error Correcting Inner-Product Units and Convolvers

Yuang-Ming Hsu, Vincenzo Piuri*, and Earl E. Swartzlander, Jr.

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712, USA
ymhsu@pine.ece.utexas.edu, e.swartzlander@compmail.com

*Department of Electronics and Information
Politecnico di Milano
piazza L. da Vinci 32, I-20133 Milano, Italy
piuri@elet.polimi.it

Abstract

Fault tolerance can be achieved by using time redundancy with modest hardware overhead at the expense of computation time. In this paper, the REcomputing with Triplication With Voting (RETWV) technique is applied to complex arithmetic units, such as inner product units and convolvers for concurrent error correction. Hardware complexity, delay, and throughput of the RETWV concurrent error correcting inner product units are analyzed and compared. It is seen that RETWV designs can be faster than the conventional design. That is, in addition to their concurrent error correcting capability, the throughput of RETWV designs is higher than that of their nonredundant counterparts. This result is significant because this shows that the RETWV technique, which is a time redundancy approach, can be used in high performance systems.

1. Introduction

Inner product units and convolvers are fundamental elements in signal processing systems [1]. They have very similar structures as can be seen in the following. Let $\mathbf{W} = [w_k]$ and $\mathbf{X} = [x_k]$ be the N -dimensional weight and the sample vectors, respectively. The inner product P is defined by

$$P = \mathbf{W} \cdot \mathbf{X} = \sum_{k=1}^N w_k x_k$$

For each operation, a different set of samples is presented at the inputs of the inner product unit. Convolution C_i is defined for a continuous sequence of input samples $(x_1, x_2, x_3, \dots)$, with respect to a weight kernel $\mathbf{W} = [w_k]$ as

$$C_i = \sum_{k=1}^N w_k x_{i-k}$$

The N -sized window of samples is shifted at each time step i . Therefore, inner products and convolutions are actually identical operations with different data input schemes.

The basic parallel architectures of an inner product unit and a convolver are shown in Figure 1 and Figure 2, respectively. The input samples are supplied in parallel in each module, while weights are stored in the internal registers. The structure is highly regular and modular. Each module M_k receives the input sample x_k , the weight w_k , and the partial sum S_{k-1} from the previous module. The bit products of x_k and w_k are generated by an array of AND gates indicated by the blocks with crosses in the figure. An adder sums up the bit products with S_{k-1} and generates the new partial summation S_k .

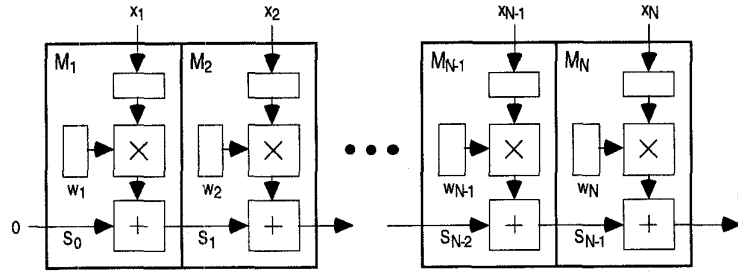


Figure 1. Block diagram of an inner product unit.

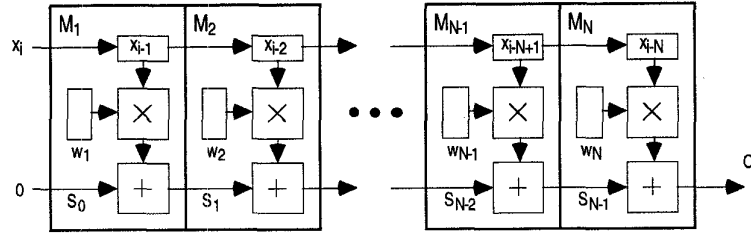


Figure 2. Block diagram of a convolver.

In the case of inner product, all the input samples x_k need to be available at the same time, one for each module. In the case of convolution, however, only one input sample at a time is supplied from the external environment. The sequence of input samples is progressively shifted along the array of modules in each computation. This is achieved by using a cascade of registers in the modules. Since a convolver and an inner product unit are identical except for the arrangement of the registers for the samples x_k , inner product units are considered in the following discussion without loss of generality.

This paper is organized as following. Section 2 reviews the RETWV concurrent error correcting technique. The architecture of the RETWV inner product unit is presented in Section 3. The hardware complexity and delay of the RETWV inner product unit designs are evaluated in Section 4. Finally, a conclusion summarizes the discussion.

2. The RETWV concurrent error correcting technique

A system is required to be highly reliable in mission-critical applications. There are many strategies to build a highly reliable system. In all cases, redundancy is introduced in the system to detect or correct the result of failures in the system. This redundancy can be in the form of hardware, information, or time. In the hardware redundancy approach, function modules in a system are duplicated or triplicated so that comparison or majority voting can be performed to detect or correct errors. It is obvious that the hardware overhead of this approach is very high while the extra delay is minimal. Coding techniques are used in the information redundancy approach. However, both the hardware complexity and delay can be very high. On the other hand, time redundancy is an approach to achieve fault tolerance without introducing too much hardware overhead at the expense of more computation time.

The basic concept of time redundancy is to repeat computations one or more times. Recomputation using the same functional modules and input operands can detect transient faults but not permanent faults. It is necessary either to change the input operands or the functional modules in the recomputation steps to detect permanent faults. Several time redundancy techniques for concurrent error detection have been proposed in the literature [2]-[5]. The *REcomputing with Duplication With Comparison* (REDWC) technique proposed by Johnson [5] has been successfully extended for concurrent error correction using *REcomputing with Triplication With Voting* (RETWV) [6]. The RETWV technique is also called *Time Shared TMR* [7].

A concurrent error correcting ripple carry adder implemented using RETWV is shown in Figure 3. The original adder as well as the input operands are divided into three parts. In the first computation, the lower third of the operands are fed into the three reduced-size adders in parallel and the partial sum is voted and stored in the registers. In the second computation, the middle third of the operands and the carry from the first computation are fed into the same reduced-size adders in parallel. Again, the second partial sum is voted and stored in another set of registers. Finally in the third computation, the reduced-size adders work on the higher third of the operands and the carry from the second computation. The partial sum is voted and concatenated with the contents of the registers to give the final result. The operands are selected by 3-to-1 multiplexers.

The block diagram of a concurrent error correcting array multiplier based on RETWV is shown in Figure 4. In this case, only one of the input operands, *i.e.*, a , is divided into three parts. The operation of this multiplier is similar to that of the RETWV adder. The RETWV technique can be applied to inner product units as shown in the next section.

3. The RETWV inner product unit

The architecture of a concurrent error correcting inner product unit based on RETWV is shown in Figure 5. The overall architecture is composed of N arithmetic modules which generate the partial results and a merging module that votes on the final outputs. In each arithmetic module, the input data x_k is partitioned into three parts; while the weight w_k is kept full word length. The array multiplier is divided into three parts as described in Section 2 and the operation is performed in the following three phases.

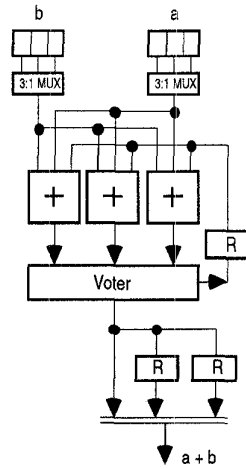


Figure 3. RETWV error correcting adder.

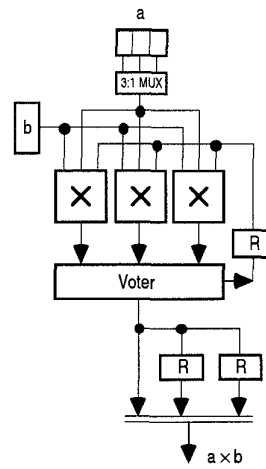


Figure 4. RETWV error correcting multiplier.

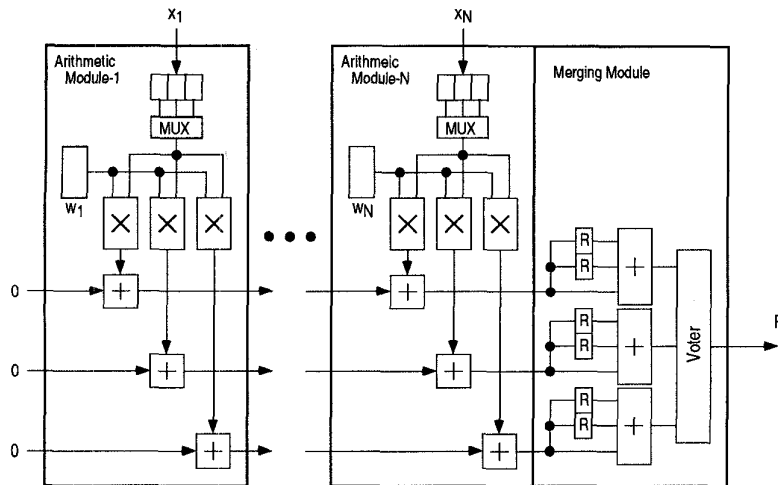


Figure 5. Block diagram of the RETWV inner product unit.

- Phase L: The lower third of x_k is multiplied by w_k in the three reduced-size array multipliers in parallel (where $k = 1, 2, \dots, N$). The bit products are accumulated in the adders together with the partial sum S_{k-1}^L from the previous arithmetic module $k-1$. The output of the N -th arithmetic module, S_N^L , is stored in the registers of the merging module.
- Phase M: The middle third of x_k is multiplied by w_k . The bit products are accumulated in the adders together with the partial sum S_{k-1}^M from the previous arithmetic module $k-1$. The output of the N -th arithmetic module, S_N^M , is stored in the second sets of registers in the merging module.
- Phase H: The higher third of x_k is multiplied by w_k . The bit products are accumulated in the adders together with the partial sum S_{k-1}^H from the previous arithmetic module $k-1$. The output of the N -th arithmetic module, S_N^H , is merged with S_N^L and S_N^M in the three adders in the merging module. The partial sums are aligned by proper wiring in the adders: S_N^H shifted to the left by $2\lceil n/3 \rceil$ bits with respect to S_N^L , S_N^M shifted to the left by $\lceil n/3 \rceil$ bits. The final result (the inner product) is obtained by taking a majority vote among the three adder outputs.

The size of the adders is assumed to be the same in all arithmetic modules. This ensures the regularity of the arithmetic modules. In the N -th module, the operands of the adder are the bit product P_k^T (where $T = L, M$, or H) with $n + \lceil n/3 \rceil$ bits, and the partial sum S_{N-1}^T , which has $n + \lceil n/3 \rceil + \lceil \log_2(N-1) \rceil$ bits. Therefore, the sum has $n + \lceil n/3 \rceil + \lceil \log_2(N-1) \rceil + 1$ bits.

The *latency* of the computation is the sum of the time for bit product generation, the propagation delay of the partial sums through all of the N arithmetic modules, the adder delay in the merging module, and the time for the majority voting. The *throughput* of this RETWV inner product unit is the reciprocal of this latency. The throughput can be improved by pipelining. In Figure 6, registers are inserted between arithmetic modules so that all arithmetic modules are isolated from each other. Although the latency is increased in this case, the throughput is improved dramatically as can be seen in the design evaluations in the next section.

4. Design evaluation

In this section, the hardware complexity and delay of the RETWV inner product units are analyzed based on a technology-independent approach. The pipelined case is considered here.

4.1 Hardware complexity

The hardware complexity of the RETWV inner product units is analyzed at the gate level. The data are assumed to be m -bit integers.

In a conventional inner product unit with window size N , there are N arithmetic modules. Each arithmetic module has three registers for the sample, weight, and the partial sum from the previous module. There is also an m -bit by m -bit array multiplier and an adder for accumulating the partial sums. In an RETWV inner product unit with window size N , there are N arithmetic modules and one merging module. In the arithmetic module, one third of both the array multiplier and the adder is triplicated. The number of registers is the same as that of the conventional design. In the merging module, the registers and the merging adders are triplicated. The voter is $n + 3\lceil n/3 \rceil + \lceil \log_2 N \rceil + 1$ bit wide. The gates used in the gate level analysis are inverters, 2 and 3-input AND gates, and 2 and 3-input OR gates. The gate count of the inner product units for various wordsizes and window sizes is listed in Table 1. The relative gate count is listed in Table 2.

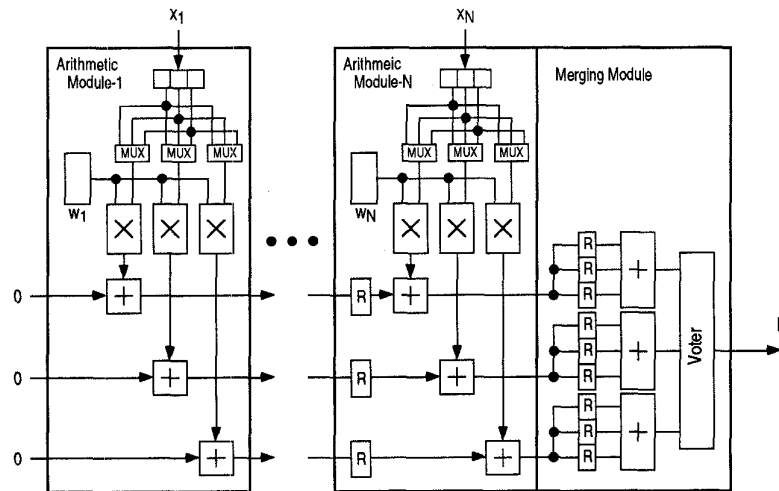


Figure 6. Block diagram of the pipelined RETWV inner product unit.

Table 1. Gate count of the inner product units.

Window size $N =$	5	8	10	15	20
Conventional:					
Data wordsize $m = 9$	5945	9772	12426	18771	25424
Data wordsize $m = 16$	16347	26516	33398	50313	67536
Data wordsize $m = 21$	26977	43596	54778	82443	110416
Data wordsize $m = 30$	52511	84580	106062	159477	213200
RETWV error correcting:					
Data wordsize $m = 9$	9592	14305	18017	26012	35097
Data wordsize $m = 16$	24032	36647	45627	66792	89047
Data wordsize $m = 21$	34072	52249	64937	95372	126012
Data wordsize $m = 30$	62092	96163	119447	176372	234387

Table 2. Relative gate count of the RETWV inner product units.

Window size $N =$	5	8	10	15	20
Data wordsize $m = 9$	161.4%	146.4%	145.0%	138.6%	138.1%
Data wordsize $m = 16$	147.0%	138.2%	136.6%	132.8%	131.9%
Data wordsize $m = 21$	126.3%	119.9%	118.6%	115.7%	114.9%
Data wordsize $m = 30$	118.3%	113.7%	112.6%	110.6%	109.9%

4.2 Delay

As can be seen from the tables in the previous section, the hardware overhead of the RETWV inner product units is very low compared to the traditional TMR design, which has hardware complexity of more than 300%. The *delay* is the time between data application at the input and data available at the output. In terms of full adder delays, the delay of a conventional inner product unit with window size N is $N(2m + \lceil \log_2(N-1) \rceil)$, where m is the number of bits in the samples and weights. The delay of an RETWV inner product unit is $(N+3)(m + 2\lceil n/3 \rceil + \lceil \log_2(N-1) \rceil)$. The delay of the inner product units of various number of bits and window sizes is listed in Table 3. The relative delay is listed in Table 4.

Table 3. Delay of the inner product units.

Window size $N =$	5	8	10	15	20
Conventional:					
Data wordsize $m = 9$	100	168	220	330	460
Data wordsize $m = 16$	170	280	360	540	740
Data wordsize $m = 21$	220	360	460	690	940
Data wordsize $m = 30$	310	504	640	960	1300
RETWV error correcting:					
Data wordsize $m = 9$	144	198	247	342	460
Data wordsize $m = 16$	248	341	416	576	759
Data wordsize $m = 21$	304	418	507	702	920
Data wordsize $m = 30$	424	583	702	972	1265

Table 4. Relative delay of the RETWV inner product units.

Window size $N =$	5	8	10	15	20
Data wordsize $m = 9$	144.0%	117.9%	112.3%	103.6%	100.0%
Data wordsize $m = 16$	145.9%	121.8%	115.6%	106.7%	102.6%
Data wordsize $m = 21$	138.2%	116.1%	110.2%	101.7%	97.9%
Data wordsize $m = 30$	136.8%	115.7%	109.7%	101.3%	97.3%

It is interesting to note that for some window sizes and data wordsizes the delay of the RETWV design is less than that of the conventional design. That is, the RETWV designs can be *faster*. This is due to the fact that reducing the size of the array multipliers and adders in the arithmetic modules increases their speed, which makes them faster than the larger multipliers and adders in the conventional design.

Furthermore, since the designs are pipelined, an inner product is available at the output every clock cycle. The *clock cycle time* of a pipelined conventional inner product unit is the maximum propagation delay in one arithmetic module, which is $2m + \lceil \log_2(N-1) \rceil$ adder delays. In an RETWV design, the clock cycle time is the propagation delay in the merging module, which is slower than the arithmetic modules. The performance of the inner product units can be measured by their throughput, which is the reciprocal of the

clock cycle time. In terms of full adder delay, the clock cycle time of the inner product units is listed in Table 5. Notice that all the RETWV inner product units have lower clock cycle time than the conventional design. Therefore they are all faster than their conventional counterparts. The relative throughput of the RETWV inner product units is shown in Table 6. This shows that RETWV can be applied in high performance systems.

Table 5. Clock cycle time of the inner product units.

Window size $N =$	5	8	10	15	20
Conventional:					
Data wordsize $m = 9$	20	21	22	22	23
Data wordsize $m = 16$	34	35	36	36	37
Data wordsize $m = 21$	44	45	46	46	47
Data wordsize $m = 30$	62	63	64	64	65
RETWV error correcting:					
Data wordsize $m = 9$	18	18	25	23	23
Data wordsize $m = 16$	31	31	32	32	33
Data wordsize $m = 21$	38	38	39	39	40
Data wordsize $m = 30$	53	53	54	54	55

Table 6. Relative throughput of the RETWV inner product units.

Window size $N =$	5	8	10	15	20
Data wordsize $m = 9$	111.1%	116.7%	115.8%	115.8%	115.0%
Data wordsize $m = 16$	109.7%	112.9%	112.5%	112.5%	112.1%
Data wordsize $m = 21$	115.8%	118.4%	117.9%	117.9%	117.5%
Data wordsize $m = 30$	117.0%	118.9%	118.5%	118.5%	118.2%

5. Conclusion

In this paper, the REcomputing with Triplication With Voting (RETWV) technique is applied to the design of inner product units and convolvers. Concurrent error correction can be achieved without introducing too much hardware overhead. It is shown that for various data wordsizes and window sizes, the hardware overhead ranges from merely 10% to about 60%. This hardware overhead decreases with increasing data wordsize or window size. The relative delay of the RETWV designs also decreases with increasing data wordsize or window size. It is interesting to note that the RETWV designs are faster than the conventional ones for larger wordsizes or window sizes. More importantly, the throughput of the RETWV designs is always higher than that of the conventional designs. This result is significant because this shows that the RETWV technique, which is a time redundancy approach, can be used in high performance systems.

References

- [1] A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.
- [2] D. Reynolds and G. Metzger, "Fault detection capabilities of alternating logic," *IEEE Trans. Comput.*, Vol. C-27, pp. 1093-1098, 1978.
- [3] J. Patel and L. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," *IEEE Trans. Comput.*, Vol. C-31, pp. 589-595, 1982.
- [4] B. W. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems*, Section 3.6.2, pp. 144-145, Reading, MA: Addison-Wesley, 1989.
- [5] B. W. Johnson, J. H. Aylor, and H. H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder," *IEEE J. Solid-State Circuits*, Vol. 23, pp. 208-215, 1988.
- [6] Y.-M. Hsu and E. E. Swartzlander, Jr., "VLSI concurrent error correcting adders and multipliers," *Proc. 1993 IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 287-294.
- [7] Y.-M. Hsu, *Concurrent Error Correcting Arithmetic Processors*, Ph.D. Dissertation, Austin, TX: The University of Texas at Austin, 1995.