

Time-Shared Modular Redundancy for Fault-Tolerant FFT Processors

Vincenzo Piuri
Dept. Electronics and Information
Politecnico di Milano
piazza L. da Vinci 32, 20133 Milano, Italy
email piuri@elet.polimi.it

Earl E. Swartzlander, Jr.
Dept. Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712, USA
email e.swartzlander@compmail.com

Abstract

This paper presents an efficient approach to concurrent error detection and correction for FFT processors by using time-shared modular redundancy. Digits of each input operand are partitioned in disjoint subsets: the nominal operations are performed more than once on each subset by using different arithmetic units. Comparison of results allows detection and, possibly, correction of errors. The modified architectures for detection and correction are analyzed and evaluated.

1: Introduction

The Discrete Fourier Transform is one of the fundamental digital signal processing tools [1] in many signal and image processing applications related to a large variety of engineering fields. For simplicity's sake, we consider here only the case of the one-dimensional DFT, but the proposed approach to fault tolerance can be directly extended to deal with the multidimensional case. Let $x(k)$ be the input sample at time k , N the number of points of the transform, and $X(k)$ the transformed output

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} = \sum_{n=0}^{N-1} x(n)e^{-jk(2\pi/N)n} \quad 0 \leq k \leq N-1$$

Several Fast Fourier Transform (FFT) algorithms and related architectures have been proposed in the literature to compute the DFT efficiently. The scheme of the radix-2 Cooley-Turkey algorithm [1] is one of the most popular. For massive-computing applications (e.g., aerospace, telecommunications, real-time data analysis), we adopt an architecture that directly maps each functional block (i.e., each radix-2 butterfly operator) of this scheme onto a distinct digital device to achieve the maximum parallelism. Since the weights W_N^{kn} are given constants depending only on the size of the transform, they can be hardwired (i.e., no register is required). To increase the throughput, we use pipelining implemented by introducing master-slave registers between the stages of this FFT processor.

When the application has mission-critical characteristics, it becomes mandatory to introduce fault-tolerant capabilities in order to assure the correctness of the delivered results. If the system needs to operate in real time, error detection and correction must be realized concurrently with the nominal computation. Data coding (e.g., AN codes, residue codes) usually requires additional circuit complexity for the redundant operations as well as increased latency. Moreover, only single errors in the output can be dealt with at reasonable costs in terms of

circuit complexity, latency, and throughput; this often limits the actual fault model that can be considered. Traditional modular redundancy is robust for any kind of error affecting a single module, but has a high circuit complexity. Similarly, traditional time-redundancy solutions (e.g., alternating logic [2], recomputing with shifted operands [3], recomputing with rotated operands [4]) suffer from a large latency increase and throughput reduction, even though the circuit complexity is minimized. In the literature, some approaches have been proposed for FFT processors with concurrent error detection [5-7] and with error correction [8-12], but they have high circuit redundancy and performance degradation. General techniques for array processors based on checking with spare units shared by several processing elements (e.g., [13]) can be efficiently adapted to the FFT case, but they are not able to guarantee testing of all results.

In this paper, we consider time-shared modular redundancy [14,15] as an effective technique to perform concurrent error detection and, possibly, correction in dedicated FFT processors. Each operand is partitioned in groups of adjacent digits: computation for each group is repeated on different modules in parallel and the results are compared for error detection and correction. Time-shared modular redundancy is a comprehensive extension of the technique proposed in [16] for concurrent error detection in adders only. A good trade-off between circuit complexity increase and performance reduction is achieved by applying this technique at system level in an original way. The error model is very wide and comprehensive since it encompasses all possible errors in the output of a single module, implying that any fault multiplicity at the gate level can be accepted if the overall faulty area affects the computation of only one module. The error coverage is larger than that which can be provided by other techniques (like coding) at the same circuit complexity.

2: Time-shared modular redundancy for concurrent error detection and correction

The basic concept of time redundancy is to perform the computation multiple times in order to detect the presence of errors and, possibly, to correct them. Recomputation with the same functional modules and input operands can detect transient failures only, not permanent ones. Permanent failures can be detected either by changing the functional modules or the operands. Time-shared modular redundancy has been shown an effective in arithmetic systems [14,15], at reasonable circuit complexity. Basic arithmetic operations, like addition and multiplication, are divided in parts by splitting the operands: each part of the operands is treated by the corresponding partial operation. Fault tolerance is achieved by performing the partial operations on the partial operands by using different circuits in parallel.

In the case of concurrent detection, comparison of two results is sufficient to detect errors due to any kind of fault confined in one of the functional modules. The operands and the basic operations need therefore to be split into two parts and two identical modules perform the redundant partial operations in parallel: simple comparison allows determination of correctness of the results. A self-checking comparator should be used to perform the comparison. In the case of correction, majority voting is used to deduce the correct output. If the fault is confined within a single functional module, only three copies must be analyzed and voted: the operands and the operations are thus partitioned in three parts. A self-checking voter should be used.

From the analysis of the circuitry required by the time-shared modular redundancy, this technique is most efficient if we move the checking to the highest level of the system structure. This is possible whenever carries do not need to be immediately propagated, i.e., when the overall computation - viewed as a cascade of multiplications and additions - can be rearranged by applying the distributive property of the product with respect to the summation. This is the case for the inner product, convolution [15], as well as the FFT algorithm.

3: Concurrent error detection and correction in the FFT processor

Application of time-shared redundancy for concurrent error detection in FFT processor can be performed efficiently at system level by avoiding to store, compare and recirculate carries locally in each arithmetic unit. Two iterations of the nominal computation must be performed on different arithmetic units and two copies of the final transformed data $X(k)$ must be produced in order to execute the error checking. In fact, we are interested in verifying the final outputs only, without any concern about the intermediate results. Each B -bits input operand $x(k)$ of the FFT processor is partitioned in two parts having half length of the nominal operand (i.e., $\lceil B/2 \rceil$ bits); as in the multiplication case, the weights W_N^{kn} are not partitioned. Computation on each part of the input operands is performed independently and carries are propagated toward the processor outputs for addition and generation of the final results $X(k)$. Comparison on the two copies of each result $X(k)$ allows for detecting the presence of errors due to faults.

The resulting redundant system structure for the FFT processor with concurrent error detection capacity is shown in Figure 1. Each functional block still deals with the same kind of data, but by applying the time-shared redundancy. A modified functional block is derived for each functional block of the nominal structure by duplicating each arithmetic unit and the interconnection paths: however, each of the two instances deals with only half of the operand bits. Practically, from an high-level point of view, two identical redundant FFT processors are created, working in parallel on the half-sized operands; the overall structure of each of these redundant FFT processors is identical to that of the nominal one, apart of the final carry propagation required in the fault-tolerant case. Each modified functional block in the first stage of the FFT processor with detection capabilities is preceded by a input partitioning block, while the modified functional blocks in the last stage are followed by merging and comparing blocks.

The input partitioning blocks (see Figure 2a) are the first main difference with respect to the nominal FFT processor. These blocks include by multiplexers to partition the input samples $x(k)$ and present the suitable part in parallel to the corresponding modified functional block at the given time. During the first iteration of the time-shared redundancy approach, the lowest-significant part of the input samples $x(k)$ is delivered in parallel to the corresponding modified functional block in the first stage of the FFT processor; during the second iteration, the highest-significant part is selected. For each primary input of the fault-tolerant FFT processor, we use a two-input multiplexer to choose between the lowest- and the highest-significant parts of the corresponding operand. In the case of a fault in one of these multiplexers, an erroneous input is presented to one instance of the arithmetic units in the modified functional block, while the other instance has the correct value (under the single faulty module assumption). This structure guarantees error detection also for faults in the input multiplexers. Alternatively, we can adopt only one multiplexer to perform inputs partitioning since the two redundant FFT processors work in parallel on the same operand part during each iteration. This approach reduces the circuit complexity of each block, but possible faults in multiplexers themselves will not be detected as errors in the final results since the same erroneous input is presented to both instances of the arithmetic units in the functional block: these multiplexers must be therefore considered hard-core.

The modified functional blocks (see Figure 2b) are similar to the nominal ones except for the size of the input operands and, as a consequence, the size of the arithmetic units and interconnection paths. The weights W_N^{kn} are not modified. During the first iteration of the time-shared redundancy approach, all operations are performed on the least-significant part of the input samples $x(k)$ with the full-sized weights W_N^{kn} , in accordance to the nominal FFT algorithm; the least-significant partial result is generated for $X(k)$ at the output of the

corresponding modified functional block in the last stage of the FFT processor. During the second iteration, the same operations are applied to the most-significant part of the input samples $x(k)$ and the most-significant partial result is computed for each transformed $X(k)$. Note that the least- and the most-significant partial results are not the two (lowest and highest) parts of the results $X(k)$ since carries still need to be propagated from the first result to the second one.

To complete the computation of output $X(k)$, we need therefore to add the two corresponding partial results. The final merging and comparing block (see Figure 2c) connected to the output of each modified functional block in the last stage of the FFT processor is introduced to perform this specific task. Since the least- and the most-significant partial results for each output are produced at two different times (at the end of the first and the second iteration, respectively) by the same circuits, the least-significant partial result needs to be stored in a register in order to save it during the generation of the most-significant partial result. Then, the two partial results can be added after a suitable alignment that takes into account the initial alignment of the two parts of the input samples $x(k)$. The most-significant partial result must be in fact shifted to the left by $\lceil B/2 \rceil$ bits with respect to the least-significant one; this can be simply accomplished by proper wiring from the outputs of the register and the inputs of the block to the inputs of the adder. The output of one of the adder is delivered as the final result $X(k)$ generated by the corresponding functional block in the last stage of the FFT processor. In parallel, the two final results are compared to verify the presence of possible errors. A self-checking comparator should be preferred to guarantee that also possible faults in the comparator itself are dealt with.

To improve the system throughput dramatically, pipelining can be applied also in the fault-tolerant FFT processor as well as in the nominal one. This can be accomplished by inserting a master-slave register in each data flow between each stage of the FFT processor as well as in each data path delivering the highest-significant partial result from the output of the modified functional blocks in the last stage to the adder in the corresponding merging and comparing block. The overall architecture of the pipelined FFT processor is identical to the one presented in Figure 1, but the modified functional blocks and the merging and comparing blocks are different since they must include a master-slave pipeline register for each input (see Figure 3). In this case a new operation can be initiated when any individual block (either a functional block or a merging and comparing block) has stored the steady output coming from the previous stage into the input pipeline register and has generated a steady output for the subsequent stage. In the non-fault-tolerant architecture as well as in the fault-tolerant ones, the system latency is increased with respect to the corresponding non-pipelined structures; however, throughput enhancement is so relevant that the latency degradation can be easily accepted in all applications in which throughput is prevalent.

The use of time-shared redundancy at system level in complex arithmetic systems like the FFT processors can be extended also to deal with concurrent error correction, still avoiding to store, vote and recirculate carries locally in arithmetic units. Now three iterations of the nominal computation are required on different arithmetic units so as to generate three copies of the final transformed data $X(k)$. Majority voting of these results allow to detect the presence of possible errors due to faults (of any multiplicity) in the arithmetic units generating one single result and to correct them. Intermediate results of functional blocks does not need to be corrected since the approach is able to correct the possible error even working only on the final system outputs. Each B -bits input operand $x(k)$ of the FFT processor is now partitioned in three parts of $\lceil B/3 \rceil$ bits each, while the weights W_N^{kn} are kept entirely. Three independent iterations of the FFT algorithm are applied in parallel on each part of the input samples: carries are propagated toward the processor outputs for final addition. The three copies of the final transformed data $X(k)$ are produced: on them, voting is applied.

The resulting architecture of the FFT processor is shown in Figure 4: it is the direct extension of the one presented in Figure 3. Each modified functional block is composed of three independent instances of the circuits implementing the nominal operation instead of only two.

4: Architectural evaluation and conclusions

To abstract from a specific integration technology, the circuit complexity is measured as number of logic gates [17], due to the direct connection to the silicon area required by the implementation; its abstract value in area units can be obtained by multiplying this number by the average gate area in the specific integration technology. The latency is measured as number of gate levels along the longest path from the inputs to the outputs [17]; its abstract value in time units is the product of this number by the average gate delay. The throughput is the inverse of the time between two consecutive complete sets of transform outputs; its abstract value in outputs per time units is the product of this number by the inverse of the clock cycle time. The actual physical values of the above figures of merit in a specific integration technology may be different from the abstract values since several physical effects must be taken into account (e.g., the actual fan-in and fan-out, the load capacitances, the interconnection paths). These exact values can be derived only from a detailed analysis of the actual implementation.

The circuit complexity increase (in percentage) with respect to the nominal architectures (with or without pipelining) is shown in Table 1 for different numbers B of bits of the input samples and for some characteristic values of the transform window size N . The typical cost for detection ranges from 10 to 20%, while for correction it is from 50 to 75%. This redundancy is definitely smaller than the one required by other time-redundancy approaches as well as by physical modular redundancy; besides, it is also competitive with respect to data coding whenever multiple errors are to be dealt with.

The latency percentage increase for the proposed architectures is given in Table 2, for different values of B and N . Typically, detection requires about 70% without pipelining, while increases to about 100% for pipelining; correction needs about 160% and 200% without and with pipelining, respectively. It is interesting to point out that, in the case of non-pipelined architectures, the latency is not double or tripled, even if the time-shared redundancy is repeating the whole computation twice or three times, respectively. This is due to the shorter operands treated by the arithmetic units. In the pipelined cases, storing the outputs of each stage requires to complete the propagation of all carries within each functional block before starting the operation of the subsequent ones.

The throughput percentage decrease is evaluated in Table 3, for different values of B and N . For concurrent detection and correction, it is about 40% and 60%, respectively. The results are similar for the pipelined architectures. This behavior is better than the other time redundancy approaches since the shorter operands allow to reduce the carry propagation time of each iteration of the time-shared redundancy approach: for detection and correction with other time redundancy techniques, we in fact expect a throughput percentage reduction of at least 50% and 66%, respectively.

The use of time-shared modular redundancy is therefore more effective and efficient to implement concurrent error detection and correction at system level in the 1-D FFT processors than other modular redundancy techniques, whenever dedicated solutions are needed and commercial off-the-shelf chips are not suitable (e.g., in embedded and single-chip systems, especially for low-power applications).

References

- [1] Oppenheim and R.W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.
- [2] D. Reynolds and G. Metze, "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] L.G. Chen and T.H. Chen, "Fault-tolerant serial-parallel multiplier," *IEE Proc. E*, Vol. 138, pp. 276-280, 1991.
- [5] F. Lombardi and J.C. Muzio, "Concurrent error detection and fault location in an FFT architecture," *IEEE J. Solid-State Circuits*, Vol. 27, pp. 728-736, 1992.
- [6] D.L. Tao and C.R.P. Hartmann, "A novel concurrent error detection scheme for FFT networks," *IEEE Trans. Parallel and Distributed Systems*, Vol. 4, pp. 198-221, 1993.
- [7] T.H. Chen and L.G. Chen, "Concurrent error detectable butterfly chip for real-time FFT processing through time redundancy," *IEEE J. Solid-State Circuits*, Vol. 28, pp. 537-547, 1993.
- [8] A. Antola, *et al.*, "Fault tolerance in FFT arrays: time redundancy approaches," *J. VLSI Signal Processing*, Vol. 4, pp.295-316, 1992.
- [9] Y.H. Choi and M. Malek, "A fault-tolerant FFT processor," *IEEE Trans. Comput.*, Vol. 37, pp. 617-621, 1988.
- [10] J.Y. Jou and J.A. Abraham, "Fault-tolerant FFT networks," *IEEE Trans. Comput.*, Vol. 37, pp. 548-561, 1988.
- [11] M. Tsunoyama and S. Naito, "A fault-tolerant FFT processor," *Proc. 1991 IEEE Fault Tolerant Computing Symposium*, pp. 128-135, 1991.
- [12] C.G. Oh and H.Y. Youn, "On concurrent error detection, location, and correction of FFT networks," *Proc. 1993 Fault Tolerant Computing Symposium*, pp. 596-605, 1993.
- [13] S. Goldberg, S. Upadhyaya, and W.K. Fuchs, "Recovery schemes for mesh arrays utilizing dedicated spares," *Proc. 1996 IEEE Int'l Symposium on Defect and Fault Tolerance in VLSI Systems*, Boston, MA, USA, pp. 318-326.
- [14] 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*, Venice, Italy, Oct. 1993, pp. 287-294.
- [15] Y.-M. Hsu, V. Piuri, and E.E. Swartzlander, Jr., "Efficient time redundancy for error correcting inner-product units and convolvers," *Proc. 1995 IEEE Int'l Workshop on Defect and Fault Tolerance in VLSI Systems*, Lafayette, LA, USA, pp. 287-294.
- [16] 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.
- [17] C. Mead, L. Conway, *Introduction to VLSI System*, Addison-Wesley, 1980.

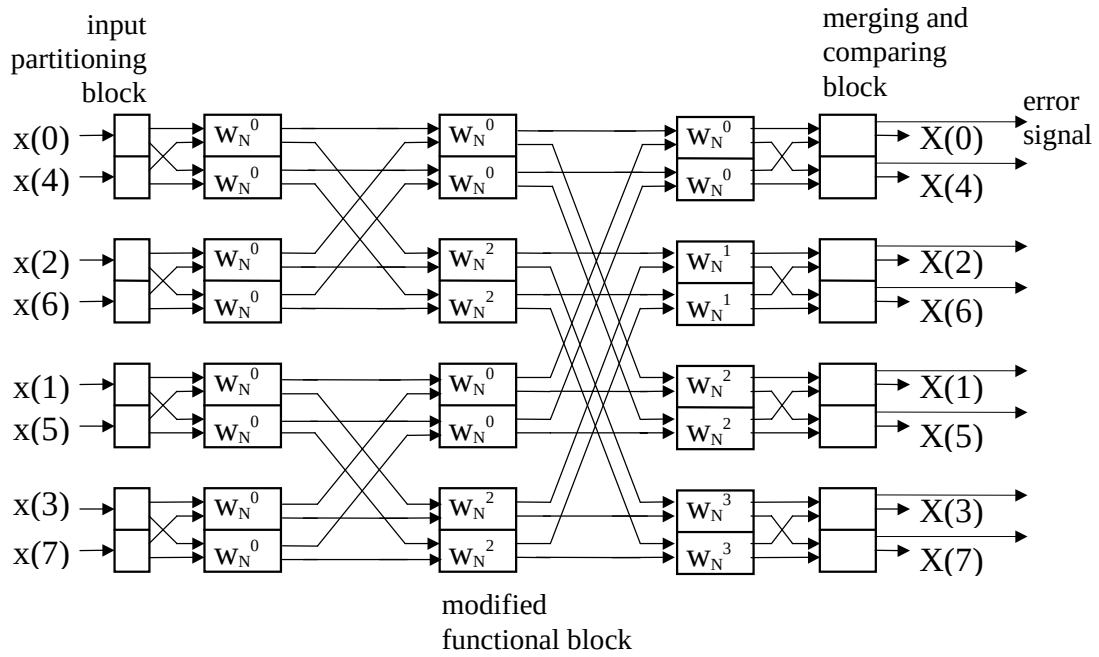


Figure 1: The modular architecture of the FFT processor with concurrent error detection based on time-shared modular redundancy.

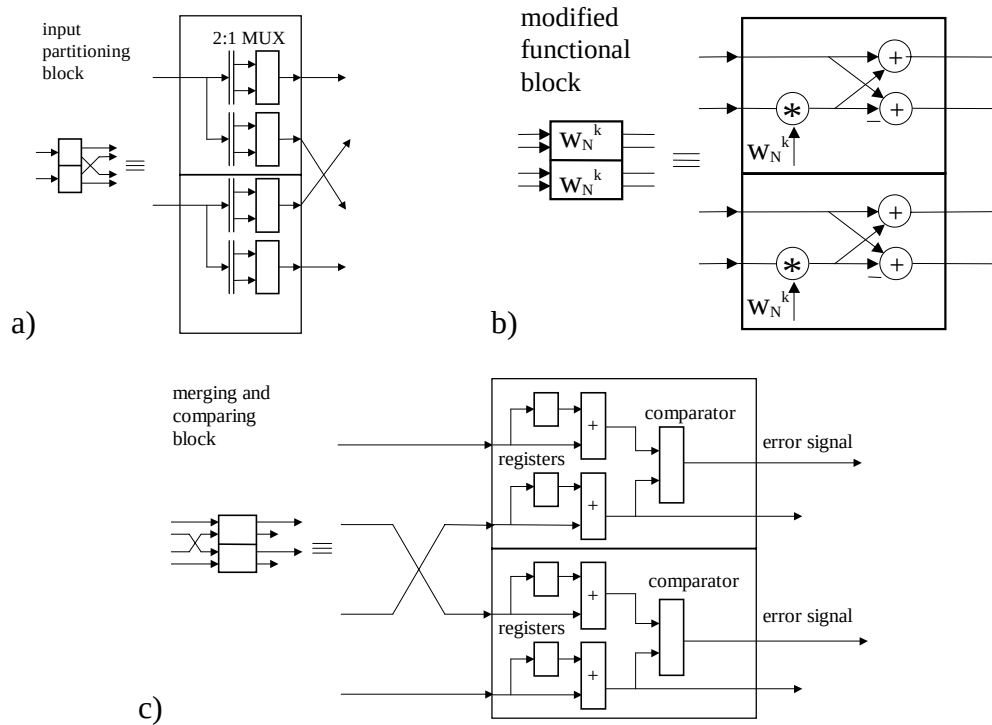


Figure 2: The structure of the redundant blocks for concurrent error detection: (a) the input partitioning block, (b) the modified functional block, and (c) the merging and comparing block.

Table 1: The circuit complexity increase (in percentage).

architecture	detection			detection with pipelining			correction			correction with pipelining		
window size N >	8	16	32	8	16	32	8	16	32	8	16	32
bit number B v												
8	34,5	24,8	19,1	40,4	29,0	22,4	112,6	91,5	79,2	117,5	94,9	81,8
9	28,6	20,7	16,1	34,4	24,9	19,4	101,1	83,4	73,2	106,6	87,4	76,3
12	21,8	16,3	13,1	27,5	20,4	16,3	86,0	73,2	65,8	92,5	77,9	69,5
15	17,5	13,5	11,1	22,8	17,3	14,2	76,5	66,7	61,1	82,9	71,4	64,8
16	19,2	14,8	12,3	24,4	18,6	15,2	78,3	68,2	62,4	84,8	72,9	66,1
20	14,1	11,3	9,8	18,7	14,7	12,4	68,2	61,2	57,2	74,2	65,6	60,7
24	10,7	9,0	8,2	14,7	11,9	10,4	61,3	56,4	53,7	66,7	60,4	56,8
28	10,5	9,0	8,3	14,2	11,7	10,3	59,9	55,6	53,2	65,0	59,3	56,1
32	8,4	7,6	7,2	11,6	9,9	9,0	55,7	52,7	51,0	60,3	56,1	53,7
33	7,4	6,9	6,7	10,6	9,2	8,5	54,0	51,5	50,1	58,5	54,8	52,7

Table 2: The latency increase (in percentage).

architecture	detection			detection with pipelining			correction			correction with pipelining		
window size N >	8	16	32	8	16	32	8	16	32	8	16	32
bit number B v												
8	71,7	74,0	75,4	111,7	103,6	98,6	157,6	161,0	163,1	217,6	205,5	148,4
9	69,7	72,7	74,6	112,3	103,2	98,8	154,6	159,0	161,9	218,5	204,9	148,6
12	69,0	72,8	75,2	114,7	106,4	102,2	153,6	159,2	162,9	222,0	209,6	152,8
15	68,6	72,9	75,7	116,1	108,3	104,3	152,9	159,4	163,5	224,2	212,5	155,4
16	70,4	74,4	77,0	119,1	110,8	106,4	155,6	161,6	165,5	228,7	216,2	158,1
20	68,9	73,6	76,6	118,6	111,1	107,1	153,4	160,4	165,0	228,0	216,7	158,9
24	67,9	73,0	76,4	118,3	111,3	107,5	151,9	159,6	164,6	227,5	217,0	159,5
28	68,9	74,0	77,3	120,4	113,2	109,2	153,3	161,0	166,0	230,6	219,8	161,6
32	68,1	73,5	77,0	119,9	113,1	109,3	152,2	160,2	165,6	229,9	219,7	161,7
33	67,6	73,1	76,7	119,3	112,7	109,1	151,4	159,7	165,1	229,0	219,1	161,4

Table 3: The throughput decrease (in percentage).

architecture	detection			detection with pipelining			correction			correction with pipelining		
window size N >	8	16	32	8	16	32	8	16	32	8	16	32
bit number B v												
8	41,7	42,5	42,9	37,5	38,8	39,7	61,1	61,6	61,9	58,3	59,2	59,8
9	41,0	42,1	42,7	37,0	38,7	39,7	60,7	61,4	61,8	58,0	59,1	59,8
12	40,8	42,1	42,9	37,8	39,6	40,7	60,5	61,4	61,9	58,5	59,7	60,5
15	40,7	42,1	43,0	38,2	40,1	41,3	60,4	61,4	62,0	58,8	60,1	60,9
16	41,3	42,6	43,5	39,0	40,8	41,9	60,8	61,7	62,3	59,3	60,5	61,3
20	40,8	42,4	43,4	38,9	40,9	42,1	60,5	61,6	62,2	59,3	60,6	61,4
24	40,4	42,2	43,3	38,8	40,9	42,2	60,3	61,4	62,2	59,2	60,6	61,4
28	40,7	42,5	43,6	39,4	41,4	42,7	60,5	61,6	62,4	59,6	60,9	61,8
32	40,5	42,3	43,5	39,3	41,4	42,7	60,3	61,5	62,3	59,5	60,9	61,8
33	40,3	42,2	43,4	39,1	41,3	42,6	60,2	61,4	62,2	59,4	60,8	61,7