Proceedings of the 1989 International Conference on Systolic Arrays,
Killarney, Ireland

EXPLOITING THE INHERENT FAULT TOLERANCE OF ASYNCHRONOUS ARRAYS*

Rodney M. Goodman
Kathleen Kramer
Dept of Electrical Engineering
California Institute of Technology
Pasadena, CA 91125

Anthony McAuley
Bell Communications Research
Morristown, NJ 07950

*this work is supported in part by NSF grant No. MIP 8711568

INTRODUCTION

Architectures using four state coding, a data driven technique for implementing bit-level wavefront arrays, and their responses to various faults are described. They are shown to be resistant to soft and hard error propagation. This resistance to propagation of errors is the basis of a simplified approach to fault tolerance not possible with conventional clocked systolic arrays.

1. FOUR STATE CODING

With clockless (self-timed, or asynchronous) wavefront arrays the timing is controlled by the elements themselves, not by a common clock [1]. In order to keep computations ordered, it is necessary to employ more than two states to represent a single bit of information. Four state asynchronous communication code employs two states to represent logical 1 (P1 and Q1) and two states for logical 0 (P0 and Q0), where P and Q can be thought of as two phase representations. This can be coded in binary logic using two digits: 00 = logical 0 (P0), 01 = logical 1 (Q1), 10 = logical 0 (Q0), and 11 = logical 1 (P1). If one alternates between the P and Q phase representations, there is always exactly one bit changed for every transition - so there are no races when the output changes.

A detailed paper on four state coding, which describes its advantages and disadvantages is currently being prepared for publication [15]. Its major drawback is area (true for asynchronous designs in general); but it claims three important advantages over the equivalent synchronous systolic array: faster throughput, reduced design complexity and greater reliability. For the purposes of this paper we will illustrate how four state asynchronous coding works using a simple serial shift register.

Figure 1a shows three stages in the middle of a longer shift register. It consists of three asynchronous delay cells (D-cells), whose truth table is shown in figure 1c. Of the 8 possible input states, half change the output state and the other half leave it in its existing state. The D-cell only passes Q phase data when the next element has P phase data, and only passes P phase data when the next

element has Q phase data. Each cell in the shift register will alternate between P and Q phase as data is shifted through from left to right. Figure 1b shows a simplified view of figure 1a, which will be used in the rest of this paper. Here each line represents three wires: two for the flow of information (P0, Q0, P1 or Q1) going in the direction of the arrow and one for the phase acknowledgement (P or Q) going in the opposite direction.

2. FAULTS IN MOS CIRCUITS AND THEIR DETECTION

The need for increased processing speeds (in communications, defense, complex modeling and real time image processing) is coupled with a need for increased reliability. For many applications, processing errors in the individual processing elements must be detected and corrected in order to provide assurance that the results are correct.

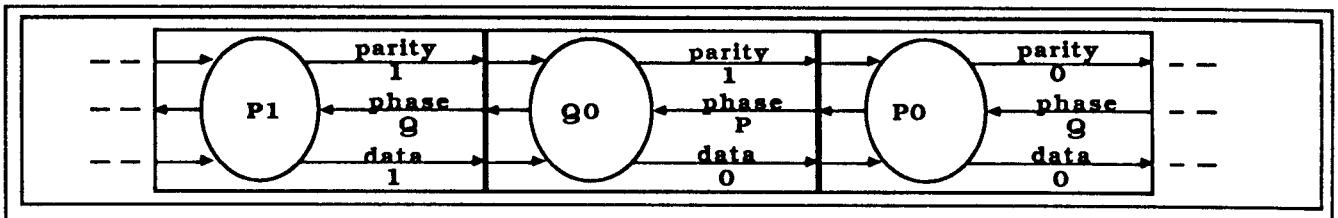
On-line detection of both permanent (or hard) errors and temporary (or soft) errors is necessary in order to provide fault-tolerant computing [9]. The continuing decrease in the minimum feature sizes has tended to increase the severity and frequency of both hard and soft errors [4]. The impact of failures upon the validity of the circuit operation may vary from none to catastrophic. While hard errors may seriously affect system performance, soft errors tend to be more difficult to detect and fix, potentially causing more harm in an application. It has been estimated that in most systems 80-90% of physical faults are soft [10]. It is in mitigating the effects of these faults that we believe asynchronous logic can have an important impact.

2.1 Transient faults or soft errors

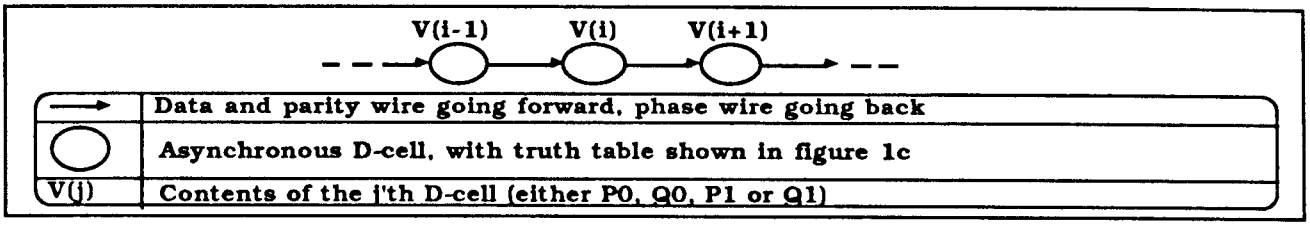
Environmental factors, such as electromagnetic interference and radiation, cause all circuits to receive undesirable charges in the form of alpha particles and cosmic rays [9]. The soft error rate in MOS circuits [2] is almost exponentially dependent on Qcrit, the critical charge necessary to change the logic value [3]. As processes are scaled Qcrit tends to decrease by the square of the scale factor. The decrease in Qcrit more than compensates for any improvement in hit probability caused by smaller devices (assuming device area scales by the square of the minimum feature size). Assuming the improved density is used to make larger arrays, keeping the active area unchanged, then scaling down by a factor of 2 results in the soft error rate increasing by a factor of between 2 and 4 [3].

2.2 Detecting and correcting soft errors

The most straightforward approach to detecting and correcting soft errors is replication. It is applicable to most circuits; and, although it costs a great deal of area, it is not very complex to implement. Triplication, like triple repetition coding, enables single error correction. An erroneous output from one circuit will be corrected if the other two circuits are outputting the correct value. With duplication, only error detection can be done, since we do not know



a) Three stages from a four state shift register.

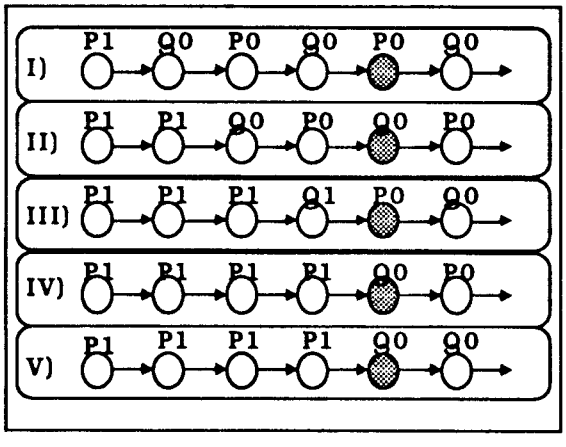


b) Simplified representation of the shift register.

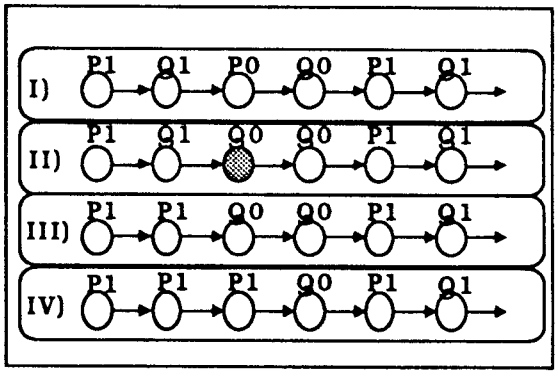
Input parity, data (phase, data)	Acknowledge phase	Output parity, data (phase, data)
00 (P0)	P	NO CHANGE
00 (P0)	Q	00 (P0)
01 (Q1)	P	01 (Q1)
01 (Q1)	Q	NO CHANGE
10 (Q0)	P	10 (Q0)
10 (Q0)	Q	NO CHANGE
11 (P1)	P	NO CHANGE
11 (P1)	Q	11 (P1)

c) Truth table for the four state shift register cell (D-cell).

Figure 1 The four state shift register.



a) with hard error.



b) with soft error.

Figure 2 Shift register with hard and soft errors.

which output is correct if they disagree. However, if we can somehow mark one circuit as having gone wrong, like an erasure in a communication channel, we can again have single error correction; and for a third less area cost. We will show that asynchronous logic can be made to output erasures rather than errors.

More sophisticated detecting schemes have been employed in some applications [7]. Examples of this are single error correcting and double error detecting codes for memories, parity bits for data buses, residue codes for ALU's, watchdogs and redundant links in switches [5] [12] [13] [14]. In general a circuit is self testing for a set of faults **F**, if for every fault in **F**, the circuit produces a non-code output [6] [8]. However, though these schemes are more area efficient, they lack the simplicity and generality of replication coding.

3. FAULTS IN A FOUR STATE SHIFT REGISTER

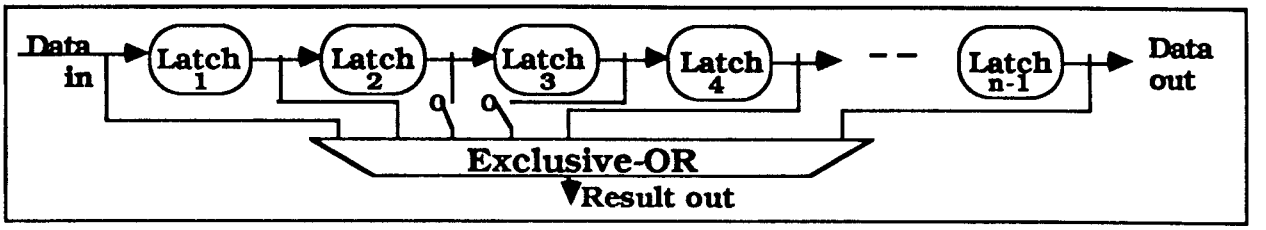
In section 2, we described how a four state shift register was supposed to operate. Although the resulting shift register is functionally similar to a clocked shift register, it is affected by faults very differently. In particular, the asynchronous shift register can react to an error by either "sticking" or "slipping."

3.1 Sticking

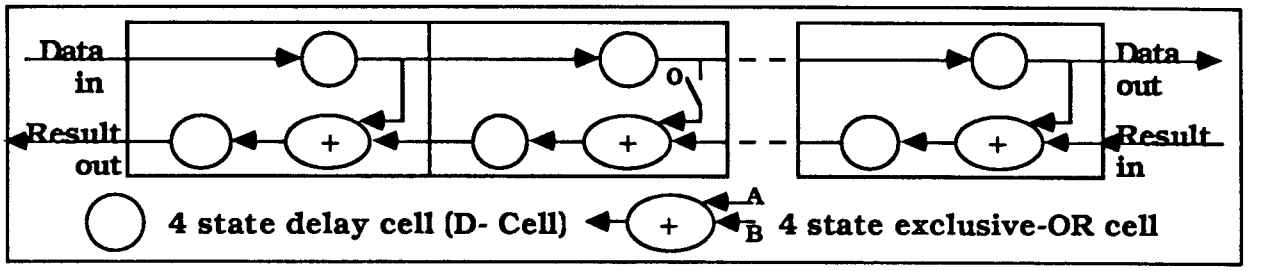
Figure 2a illustrates how a hard error could cause the register to "stick." The marked cell in figure 2aI has a permanent fault, the data bit is "stuck-at-zero." There are no problems as long as this bit is not supposed to change; but when P1 is input as in figure 2aIV, the cell cannot acknowledge the data and no further data can be received from or sent to this cell. Although sticking may appear to be undesirable behavior, in fact it can act as error detection. Regular circuit behavior would result in a parity transition, a new output, within some reasonable interval of time. Here, there is no such transition, so it can be recognized that no new output has been received from this circuit. In effect, the circuit has output an erasure. Circuit duplication would result in a repetition code of length two. Duplication, not triplication, is thus sufficient for single error correction, as only erasure correction is required.

3.2 Slipping

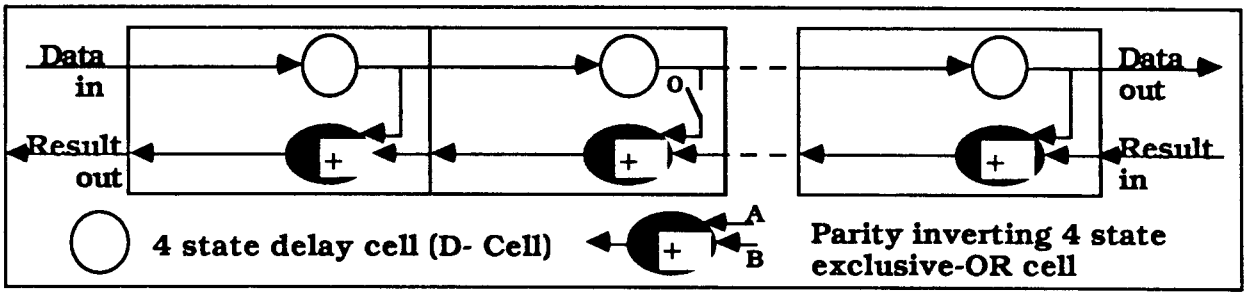
Figure 2b illustrates how a soft error could cause the register to "slip". The correct shift register contents of figure 2bI are changed by a "hit" into those of figure 2bII. Now the cells to the left of the error think that the data they sent had been received when it actually had not; and the the cells to the right would never see the data that had been changed in error. The eventual result is shown in figure 2bIV, with the leftmost P1 allowed to propagate two cells to the right; while the Q1 and P0 to its right were effectively erased. This behavior is likely to be catastrophic and needs to be avoided.



a) Example of an n-stage convolutional encoder.



b) Convolutional encoder (version 1).



c) Convolutional encoder (version 2).

Input A	Input B	Ackn. Input	Output
00	00	Q	00
00	11	Q	11
01	01	P	10
01	10	P	01
10	01	P	01
10	10	P	10
11	00	Q	11
11	11	Q	00
else			no change

normal exclusive-OR

Input A	Input B	Ackn. Input	Output
00	10	P	00
00	01	P	11
01	11	Q	10
01	00	Q	01
10	11	Q	01
10	00	Q	10
11	10	P	11
11	01	P	00
else			no change

parity inverting exclusive-OR

d) Four state exclusive-OR truth tables.

Figure 3 Convolutional encoders.

4. FAULTS IN A FOUR STATE CONVOLUTIONAL ENCODER

An n-stage convolutional encoder is a polynomial multiplier, where an input stream is multiplied by a fixed coefficient. The function is illustrated by figure 3a, where the input stream is stored in latches and the coefficient determines whether a particular latch output is included in the exclusive-OR operation. For large n the encoder would be built using a more regular systolic structure, consisting of n identical elements chained together. Two such elements, that employ four state cells, are shown in figures 3b and 3c. It is instructive to consider these two functionally equivalent architectures, since they exhibit different behavior under error conditions. We will start with a fault prone architecture and then describe an alternate fault resistant architecture.

4. 1 Version 1 - a fault prone four state convolutional encoder architecture

Three elements from a chain that make up a convolutional encoder are shown in figure 3b. It is a fairly straightforward implementation; but is optimum in terms of the throughput achievable. The two latch cells and the exclusive-OR cell that make up an element, have truth tables as shown in figure 1c and figure 3d, respectively. The top latch performs the same function as a latch in figure 3a, delaying the input as it passes from left to right. The bottom cell on the right hand side performs part of the overall exclusive-OR operation as the partial result is passed from right to left. The final latch in the bottom left is employed to improve throughput. Thus, the top cells act as a shift register, with each cell receiving its inputs from the previous element, and outputting its information both to the next element and to the exclusive-OR cell. The bottom cells act as an accumulator for the result, with the right hand cell doing the actual convolution and the left hand cell acting as a buffer.

If one of the cells that make up this convolutional encoder, shown in figure 3b, receives enough radiation to spontaneously change its output, its behavior will depend on the state of its neighbors. From our knowledge of four state coding we know that a single bit error will turn the phase (i.e P to Q or Q to P). In order to illustrate this point consider five of the cells in isolation. Specifically let us pick three cells from an element and the two right hand cells from the element to its left. Now these five cells will each be in one of four states (P0, Q0, P1 or Q1), however, for our purposes we will just note whether they contain P or Q phase data. There will then be 2^5 possible phase states. If each cell obeys its truth table (see figure 1c and figure 3d), then the legal transitions from a given state will be very limited. A convenient way of describing the possible sequences is by means of a de Bruijn diagram [16]. Figure 4a shows the resulting de Bruijn diagram for our five cells. It has four loops: two of size one (which we will term the "sticking" loops), one of size ten (which we will call the "slipping" loop) and one of size 20. (which we will call the "desired" loop).

Assume initially the encoder has been reset so that it is in the "desired" loop. Then, provided there are no errors in the system, the five cells will remain permanently in this loop. However, an error can cause the system

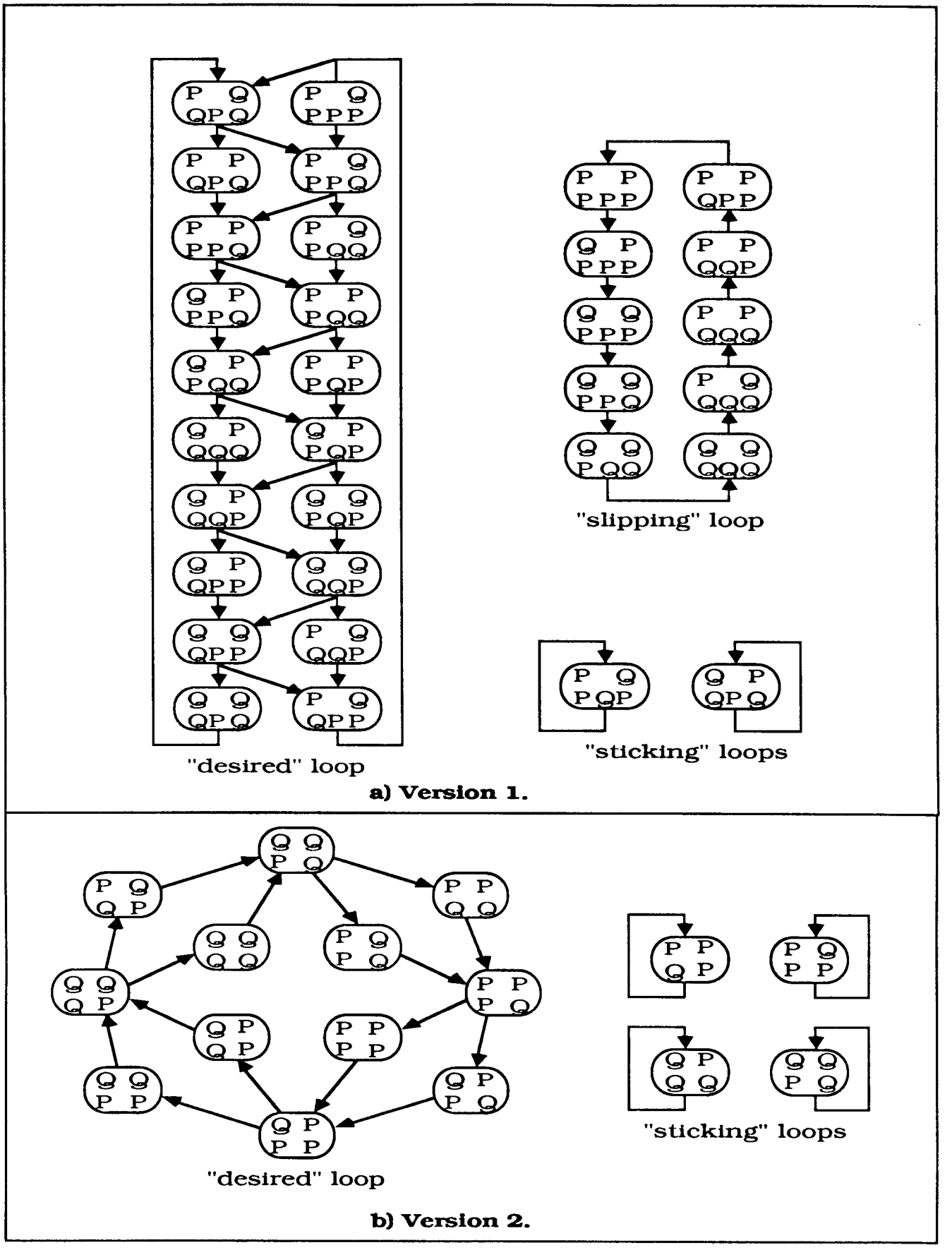


Figure 4 De Bruijn diagrams for the convolutional encoders.

to spontaneously jump into any possible state, including those of the "slipping" and "sticking" loops. If an error moves the system into either of the "sticking" loops the system will come to a complete standstill and the error will be detected. But, if an error moves the system into the "slipping" loop, two very undesirable, non-reversible events occur. First, two bits of data will be totally obliterated. This is easiest to see if the middle cell, initially in the Q state, was changed by radiation into the P state, and the remaining cells were already in the P state. Then, not only do we lose the Q bit, but we also lose one of the P's to its left or right since there is no longer any way of distinguishing them. The rest of the system will have no way of knowing that this data is lost, so the error goes undetected. A second problem that occurs is that the new loop introduces a bottleneck. This "slipping" loop only allows data to pass through at half the rate of the "desired" loop.

We can conclude that although this encoder is optimal in terms of throughput under normal conditions, it can exhibits some changes which could be catastrophic to system function as a result of a soft error. It is therefore not recommended for use.

4. 2 Version 2 - a fault resistant four state convolutional encoder architecture

The convolutional encoder shown in figure 3c makes use of a technique called phase inversion. That is, a cell A will input information from cell B with the parity bit inverted, making cell B appear to cell A as if it were in the opposite phase from its true state; and cell B will see inverted versions of cell A's acknowledgement, so that it receives the acknowledgement it is expecting. In figure 3c, the exclusive-OR cell is marked in black because it is using phase inversion. This encoder works in a very similar manner to the previous one and is functionally identical. It too is optimal in terms of throughput and is actually more efficient in area, since it employs one less cell. However, its response to soft errors is very different.

A de Bruijn diagram for two elements from this encoder is shown in figure 4b. This diagram shows five loops: four of size one ("sticking" loops) and one of size twelve ("desired" loop). If we again assume that the system is initialized to start in the "desired" loop, then again, provided there are no system errors, it will remain permanently in this loop. However, now errors can only move the system into a "sticking" loop.

We can conclude that this encoder is to be preferred because it does not exhibit the potentially catastrophic changes (slipping) as a result of soft error that were exhibited by the previous architecture.

4. 3 More general architectures

Some general comments can be made regarding the relationship between architecture and error characteristics. First, slippage will always occur when there are cells receiving from only one cell and outputting to only one cell. This trivially results from the shift register case and is also true for the delay cell in the bottom left of figure 3b. A necessary condition for sticking is that there are two different paths from one cell to another. For example, the top

delay cells in the encoder element has two paths to their exclusive-OR cell: one direct and one through the next element. The element marked in black in figure 5a is in a configuration in which a hard or soft error on its output results in sticking. Additional inputs to the cell or outputs from the cell would not take away this quality. (However, taking away from the configuration would.) In figure 5b every cell in the mesh, except the upper left and lower right endpoint, is in the configuration of figure 5a. The entire mesh will therefore stick if any one of the cells gets an error.

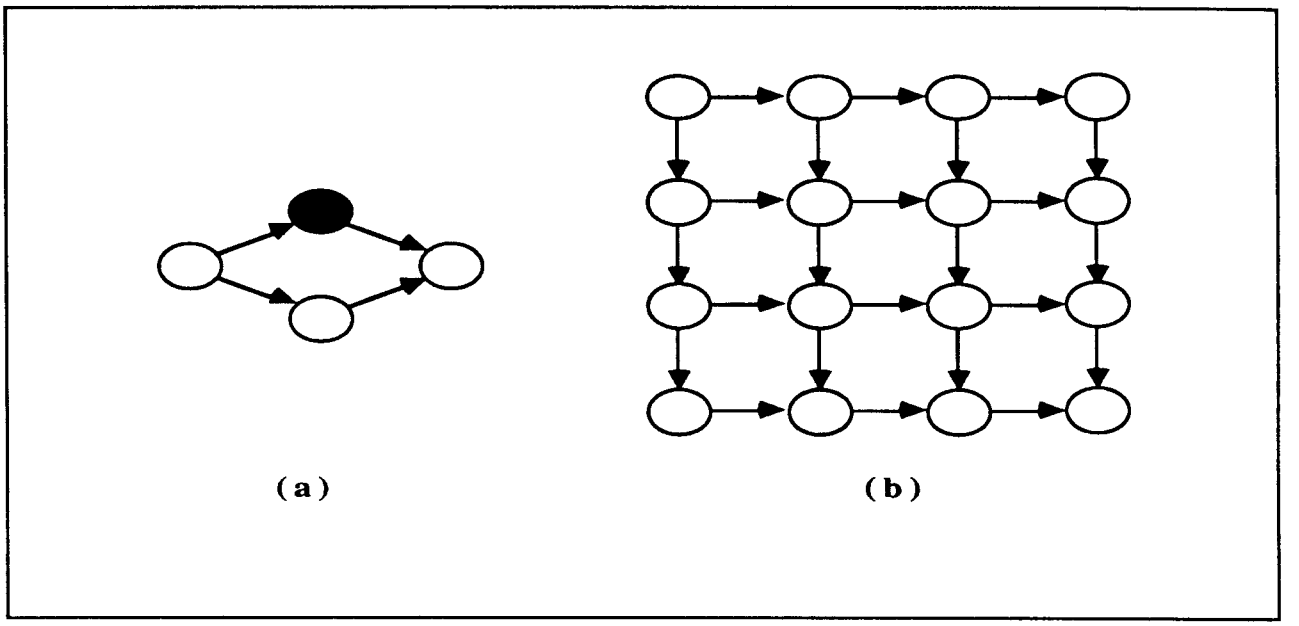


Figure 5 Mesh Configuration

4. 4 Future research

So far we have only considered the qualitative effects of an error, not their probabilities. What we really want to know is 'what is the probability of an error making a four state system: a) remaining in the "desired" loop (and hence going undetected; but not affecting the system in the long run), b) going into a "sticking" loop (and hence being detected) and c) going into a "slipping" loop (and hence going undetected; and permanently slowing the system).' The answer to this question is still the subject of research.

5. CONCLUSIONS

Four state coding is useful in the design and implementation of high performance circuits. Its inherently fault detecting nature makes its usefulness in fault tolerant designs worth investigating. The questions being studied are what architectures will produce sticking rather than slippage in response to permanent and transient faults; and whether there are simple ways of analyzing a given architecture's response to faults.

6. REFERENCES

[1] R. M. F. Goodman and A. J. McAuley, "An Efficient Asynchronous Multiplier," International Conference on Systolic Arrays, San Diego, California, USA, May 25-26, 1988.

[2] P. M. Carter and B. R. Wilkins, "Influences on Soft Error Rates in Static RAM's," IEEE Journal of Solid State Circuits, Vol. SC-22, No.3, pp. 430 - 436, July 1988.

[3] B. Chappell, S. Schuster, G. A. Sai-Halasz , "Stability and SER Analysis of Static RAM Cells", IEEE Trans. on Electron Devices, Vol. ED-32, No. 2, pp. 463 - 470 , Feb. 1985.

[4] Niraj K. Jha, "Multiple Stuck-Open Fault Detection in CMOS Logic Circuits," IEEE Trans. on Computers, Vol. 37, No. 4, pp. 426-432, April 1988.

[5] I. Gazit and M. Malek, "Fault Tolerance Capabilities in Multistage Network-Based Multicomputer Systems," IEEE Trans. on Computers, Vol. 37, No. 7, pp. 788 - 798, July 1988.

[6] T. Nanya and T. Kawamura, "Error Secure Propagating Concept and its Application to the Design of Strongly Fault-secure Processors," IEEE Trans. on Computers, Vol. 37, No. 1, pp. 14 - 24, Jan. 1988.

[7] A. Mahmood and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors -- A Survey," IEEE Trans. on Computers, Vol. 37, No. 2, pp. 160 - 174, Feb. 1988.

[8] D. Nikolos, A. M. Paschalis, and G. Philokyprou, "Efficient Design of Totally Self-Checking Checkers for all Low-Cost Artihmetic Codes," IEEE Trans. on Computers, Vol. 37, No. 7, pp. 807 - 811, July 1988.

[9] T. J. Brosnan and N.R. Strader II, "Modular Error Detection for Bit-Serial Multiplication," IEEE Trans. on Computers, Vol. 37, No. 9, pp. 1043-1052, September 1988.

[10] O. Tasar and V. Tasar, "A Study of Intermittent Faults in Digital Computers," in AFIPS Conf. Proc., vol. 46, pp 807- 811, 1979.

[11] M. W. Sievers and A. Avizienis, "Analysis of a Class of Totally Self-checking Functions Implemented in a MOS LSI General Logic Sturcture," in Proc. FTCS-10, pp. 256-261, 1980.

[12] J. R. Connet, E. J. Pasternak, and B.D. Wagner, "Software Defenses in Real Time Control Systems," in Dig. Int. Symp. Fault Tolerance Comput., FTCS-2 , Newton, MA, pp. 94-99, June 19-21, 1972.

[13] J. S. Novak and L. S. Tuomenoksqa, "Memory Mutilation in Stored Program Controlled Telephone Systems," in Conf. Rec. 1970 Int. Conf. Commun., vol. 2, pp. 43-32 to 43-45, 1970.

[14] S. M. Ornsteirn et al, "Pluribus-A Reliable Multiprocessor," in Proc. AFIPS Conf., vol 44, Anaheim, CA May 19-22, 1975 pp. 551-559.

[15] A. J. McAuley, "Four State Asynchronous Archtectures," submitted to the IEEE Trans. on Computers.

[16] B. Bannister and D. Whitehead, Fundamentals of Digital Systems, McGraw Hill, London, 1973.