ROVING EMULATION AS APPLIED TO A (255, 223) RS ENCODER SYSTEM

M.A. Breuer
F. Cohen
A. A. Ismaeel

DIGITAL INTEGRATED SYSTEMS CENTER REPORT
DISC/83-1

DEPARTMENT OF ELECTRICAL ENGINEERING-SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90089

December 1982

This work was carried out in part at the University of Southern California, Electrical Engineering-Systems Department, and was supported in part under Contract No. N00039-80-C-0641 administered by the Department of the Navy, Naval Electronics Systems Command (ONR), Pasadena, California.

Table of Contents

List of Tables

List of Figures

ABSTRACT

In this report we discuss the application of the concept of roving emulation to testing a digital system implementing a (255, 223) RS-encoder. The encoder has previously been designed as a single VLSI NMOS chip. For our purpose, we decomposed the chip into 5 main subcircuits. Each subcircuit was then checked using the concept of emulation, i.e. software duplication. The five subsystems were tested in a round robin order. This report discusses the computations envolved in computing the expected error latency for each subcircuit. The results of this analysis indicate that the expected error latency for the entire system is approximately one tenth of a second. This very effective form of testing was achieved without making any hardware modifications to the encoder chip except its decomposition into five subcircuits. No design for testability nor test pattern generation was required.

1. BACKGROUND

The concept of roving emulation is intended to be a general purpose facility for on-line testing of computer systems during their normal operation. Performance degradation is usually small. Testing is accomplished by duplicating the operation of applicable subsystem, one at a time. Duplication is carried out as a background process using a roving emulation engine. This engine is a special purpose digital system added to the original system, and used to emulate the operation of the subsystems to be tested. Roving emulation testing does not require the generation of tests nor extensive hardware for BIT or BITE. Hence it is a viable testing methodology when fault masking is not required.

In operation, the roving emulator would be either a board or large chip in a larger computer system. Its job is to rove through the system watching each subsystem for some period of time, keeping data on its initial state, final state, and interactions over that period. It then emulates that processor using the initial state and input data and compares the output and final state data with the observed data. If there is a difference, then an error in the subsystem is detected and appropriate action is indicated. The emulator then continues to cycle through the subsystems. Figure 1 indicates a general configuration of a system containing a roving emulator.

The measure we have selected to use in evaluating the performance of roving emulation testing is the expected value of the error latency time.

More details on the description of the roving emulator and the computation of error latency can be found in [1] and [2]. Reference [3] deals with the design of the roving emulator.

This report deals with appling the roving emulation test philosophy to a (255, 223) RS encoder chip designed here at USC [4,5]. This chip was partitioned into five subcircuits, which for our purpose are modeled as five separate chips. The roving emulator then emulates each one of these subcircuits in turn. We have determined that if a fault exists in any one of these subcircuits, then the expected error latency time is only one – tenth of a second.

In section 2 of this report we present the basic theory used in calculating error latency. Section 3 describes the encoder chip and the derivation of the important parameters required to calculate the error latency, such as the probability of detection.

Section 4 briefly describes the structure of the roving emulation engine, and in section 5 we calculate the time this engine requires in emulating each of the subcircuits in the encoder system.

Section 6 describes the hardware required to couple the emulator to the encoder, and finally in section 7 we calculate the expected value of error latency.

2. ERROR LATENCY EQUATIONS

Equations obtained from queueing models will be used to evaluate the expected value of error latency for the system. The expected value of error latency is equal to the average time a customer (fault) waits (exists) in the system before acquiring the required service (being detected).

Let E_i be the expected value of error latency for the system given that module i is faulty. Then we have

$$E_i = \text{average waiting time} + \text{average service time}$$

$$= \sum_{j=1}^{i} T_j - T_i + (n'_i - 1) \sum_{j \neq i} T_j + n'_i T_i \qquad (1)$$

where T_i is the testing period and n'_i is the average number of quantums needed to complete service.

Knowing E_i for each module, and using PF_i, the probability that module i is faulty given that one module in the system is faulty, we have

$$E = \sum_{i=1}^{n} PF_i E_i \qquad (2)$$

where E is the expected value of error latency for the system given that one of the modules is faulty.

Finally we note that $n'_i =$. At this point we need to find a method to calculate PD_i for digital circuits (i.e. combinational and sequential circuits).

2.1 Combinational Circuits

Since the inputs to any combinational circuit have the property of being independent of previous inputs, the probability of detecting a fault in a combinational circuit by applying m or less input vectors, given that a fault exists, is

$$PD = 1 - (1-q)^m \qquad (3)$$

In this equation q is the probability of detecting a fault in the combinational circuit, assuming that one exits, when one input vector is applied. We will adopt the single stuck-at-fault model in this analysis, hence our fault set will contain all possible single stuck-at-fault on every wire in the circuit. We calculate q for each fault in the set. Assuming that all failures are equally likely, then we calculate the average q for all the faults. This q is the one we use in equation (3).

2.2 Sequential Circuits

Any sequential circuit can be represented by the structure shown in Figure 2. We will adopt this model in the following analysis to aid us in calculating the PD for sequential circuits given m input vectors. We will assume a that only one subcircuit (input, output, or memory circuit) is faulty. We assume that the probability of failure for any subcircuit is linearly proportional to the number of wires in the circuit.

For the combinational input we will assume a single stuck-at-fault model from which we can evaluate q as explained earlier. The same assumption will also hold for the output subcircuit. We let q_1 and q_2 be the average input and output circuit probability of detection for one input vector respectively. For the memory we will assume only single stuck-at-faults on the input lines. This kind of fault will mask up to half of the memory states.

We define the propagation probability of a digital circuit as the probability that two different inputs to the circuit yield different outputs. Let C be a digital circuit with n input lines and m output lines. Let Z be the output set which contains all possible output vectors of the circuit. Let Y be the input set which contains all the input vectors of the circuit. We create a new set X by dividing the input set into $|X|$ subsets such that each subset contains all the input vectors that produce the same output. Note that $|X|$ stands for the dimension of the set X. Hence, given two input vectors, the propagation probability will be equal to the probability that the two vectors lie in different elements of the set X.

Let

$x_i \in X \qquad i = 1,2,\ldots, |X|$

and

$x_{ij} \in x_i \qquad j = 1,2,\ldots, |x_i|.$

Then the propagation probability is given by the following expression

$$\sum_{i=1}^{|Z|} \sum_{j=1}^{|x_i|} P[x_{ij}] \left(\frac{1 - \sum_{j=1}^{|x_i|} P[x_{ij}]}{1 - P[x_{ij}]} \right) \qquad (1)$$

where $P[x_{ij}]$ is the probability of applying the input x_{ij}. For the memory part of a sequential circuit the propagation probability will depend on the kind of flip-flop used in the memory elements. For example, since there is a direct mapping between the input and the output in the D flip-flop, the propagation probability is equal to 1 for a memory consisting of D flip-flops. We let A and B be the propagation probabilities for the input and the output circuit, respectively.

We have develop a computer routine which will calculate the average probability of detection for a certain number of vector given all the previous parameters.

3. THE ENCODER CHIP

The encoder chip is a VLSI architecture which is designed to realize a (255, 223) RS-encoder over $GF(2^8)$ using Berlekamp's bit-serial multiplier algorithm. The information symbols, where each symbol is represented by an eight bit byte, are fed into the chip serially through pin DIN and the encoded word is transmitted out from a pin DOUT.

The RS-encoder chip has six pins. VDD and GND are the power pins. CLK, DIN, LM, and DOUT are input-output pins. The input and LM signals are synchronized by the CLK signal.

The RS-encoder chip circuit is divided into five partitions. Figure 3 shows a block diagram of the RS-encoder chip. We will attempt to test the RS-encoder chip using the roving emulator as if it were a digital system consisting of five different modules. All we have available are the input and output pins shown in Figure 3 for each module. We will examine each module separately in order to evaluate the parameters needed for the roving emulator performance analysis.

3.1 Control Unit

The control unit is divided into 4 portions. Figure 4 shows a block diagram of the control unit. We divided the control unit into smaller blocks so that we can analyze each block precisely. The portions are the control signal generator and three divide-by-2 circuits. The control signal START resets all registers as well as the divide-by-2 counters before the encoding process begins. Control signal LD is high every eight clock cycles. To the RE this module has one input and three output pins. We will attempt to evaluate the parameters needed for each portion, then combine them to obtain the parameters for the entire module.

3.1.1 Control Signal Generator

Figure 5 shows a logic diagram for the control signal generator. The circuit is a Mealy type sequential circuit. It has one input LM and two outputs START and SL.

For the input circuit we evaluate q for each possible stuck-at-fault on the wires. Table 1 shows the faults and their corresponding q's for the input circuit. From the table

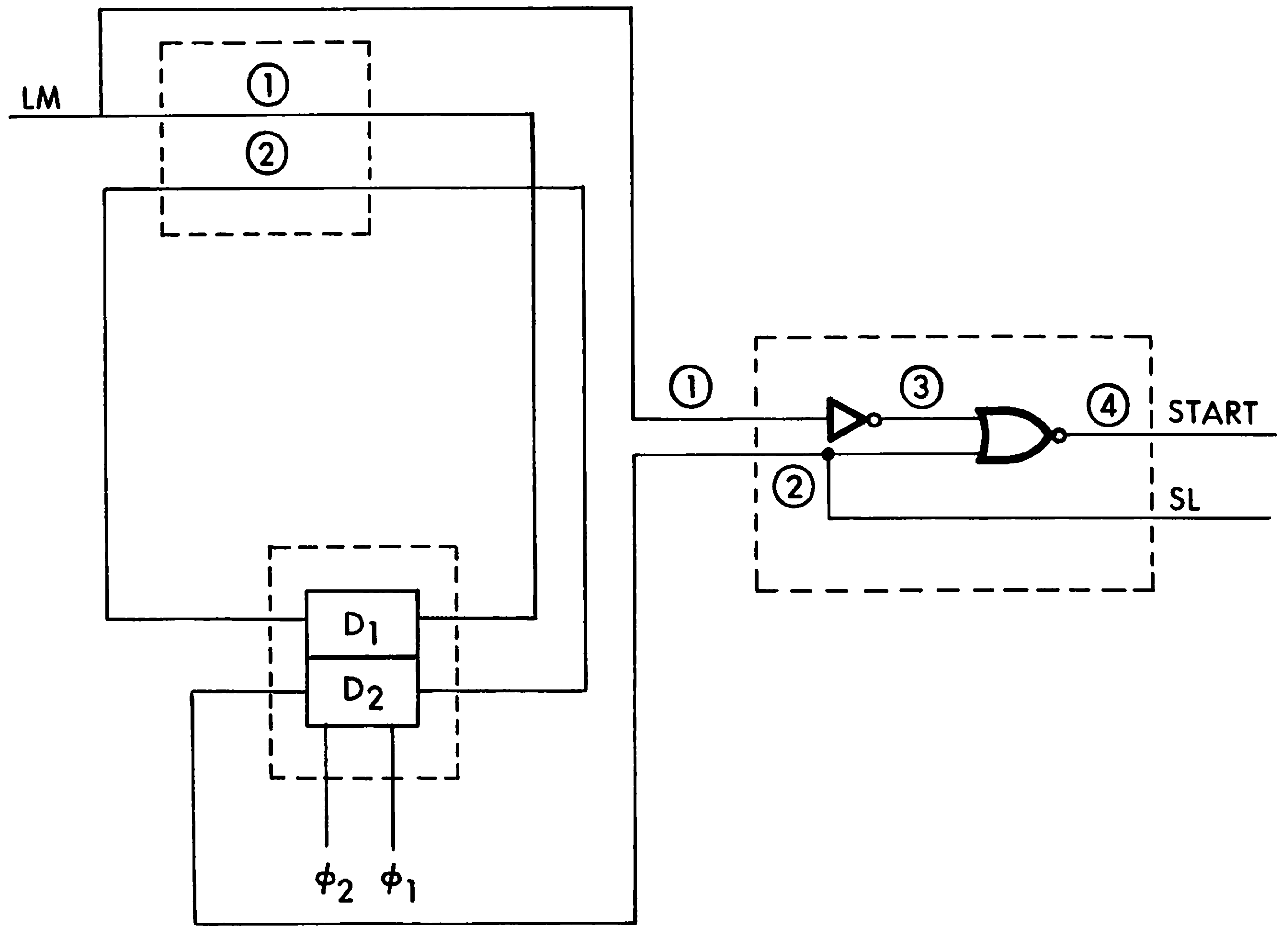

Figure 5. Control signal generator logic diagram

we obtain $q_1 = 1/4[1/2 + 1/2 + 1/2 + 1/2] = 1/2$. Since we have a direct mapping between input and output lines in the input circuit, the parameter A is equal to 1.

For the memory circuit we will assume single stuck at fault at the input lines only. For each possible stuck at fault only half of the states are accessible, hence q_m for the fault set is equal to 1/2.

From the truth table of the output circuit we obtain B = 2(1/4 x 3/3) + 2/4 x 2/3 = 5/6. Table 2 shows all possible single stuck-at-faults and their q's. From the table we obtain $q_2 = 1/8[1/2 + 1 + 1/2 + 1] = 3/8$.

To calculate the probability of detecting a fault in the circuit we will assume that the failure rate is linearly proportional to the number of wires in a circuit. From this assumption we can obtain an approximation to the failure rate for each part of the sequential circuit. The total number of wires in the control signal generator is 8. We will assume that only one part is faulty. Then the probability that the input circuit is faulty is 1/4, and the probabilities of failure for the memory and the output circuit are 1/4 and 1/2 respectively.

3.1.2 Divide-by-2 Counter 1

Figure 6 shows the circuit logic diagram for the first divide-by-2 counter. The circuit is a Moore type sequential

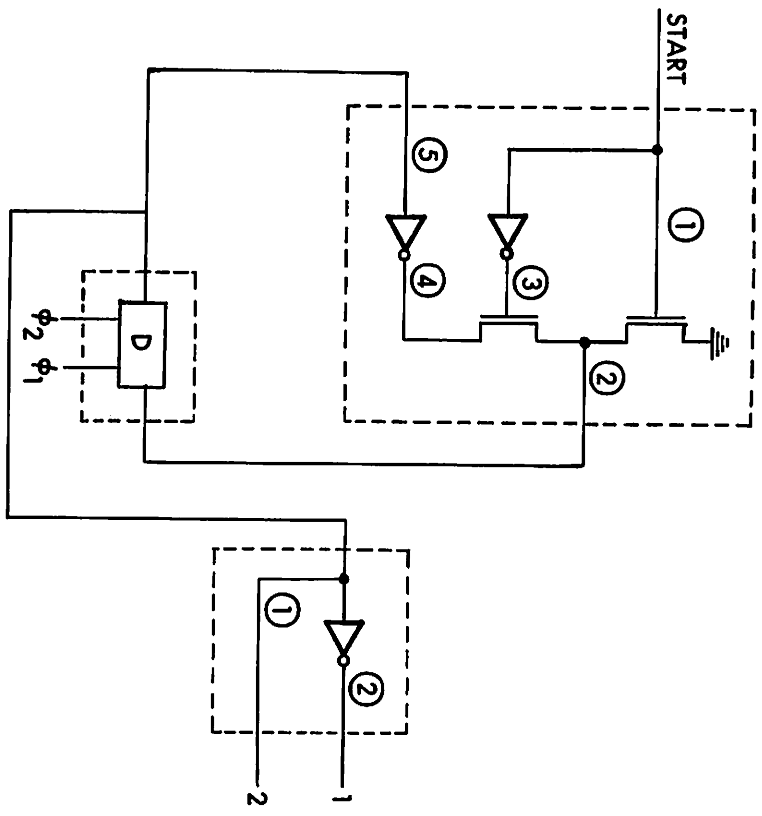

Figure 6. Divide by 2 counter 1-Logic diagram

machine. It has one input START, one memory element, and two outputs 1 and 2.

From the truth table of the input circuit we have

$A = 1/4 \times 3/3 + 3/4 \times 1/3 = 1/2$

Using the single-stuck-at-fault assumption and the data in table 3, we obtain

$q_1 = 1/10[\,1/2 + 1 + 1/2 + 1/2 + 1/4\,] = 11/40$

For the memory we obtain $q_m = 1/2$.

We obtain $B = 1$ and $q_2 = 1/2$ for the output circuit (see table 4).

Now we calculate the probability of failure for each subcircuit, assuming that one unit is faulty. The total number of wires is 8. So the probability of failure for the input, memory, and output circuits are 5/8, 1/8, and 2/8 respectively.

Table 1. Input Circuit Fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/2	1/2
2	1/2	1/2

Table 2. Output Circuit Single-stuck-at-fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/4	1/4
2	1/2	1/2
3	1/4	1/4
4	1/4	3/4

Table 3. Input Circuit Single Stuck-at-fault Set

Line Number	q	
	s - a - 0	s - a- 1
1	1/4	1/4
2	1/4	3/4
3	1/4	0
4	1/4	1/4
5	1/4	1/4

Table 4. Output Circuit Single Stuck-at-Fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/2	1/2
2	1/2	1/2

3.1.3 Divide-by-2 Counters 2 & 3

Figure 7 shows the circuit diagram for the second divide-by-2 counter. The circuit is a Mealy type machine. It has three inputs (START, 1, and 2), one memory element, and two outputs (1 and 2).

Table 5a shows the truth table for the input circuit. From the truth table we find

$A = 2/8 \times 6/7 + 6/8 \times 2/7 = 6/14$

Table 5a. Input Circuit Truth Table

Input	Output
0 0 1 0	1
0 0 1 1	0
0 1 0 0	0
0 1 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	0
1 1 0 1	0

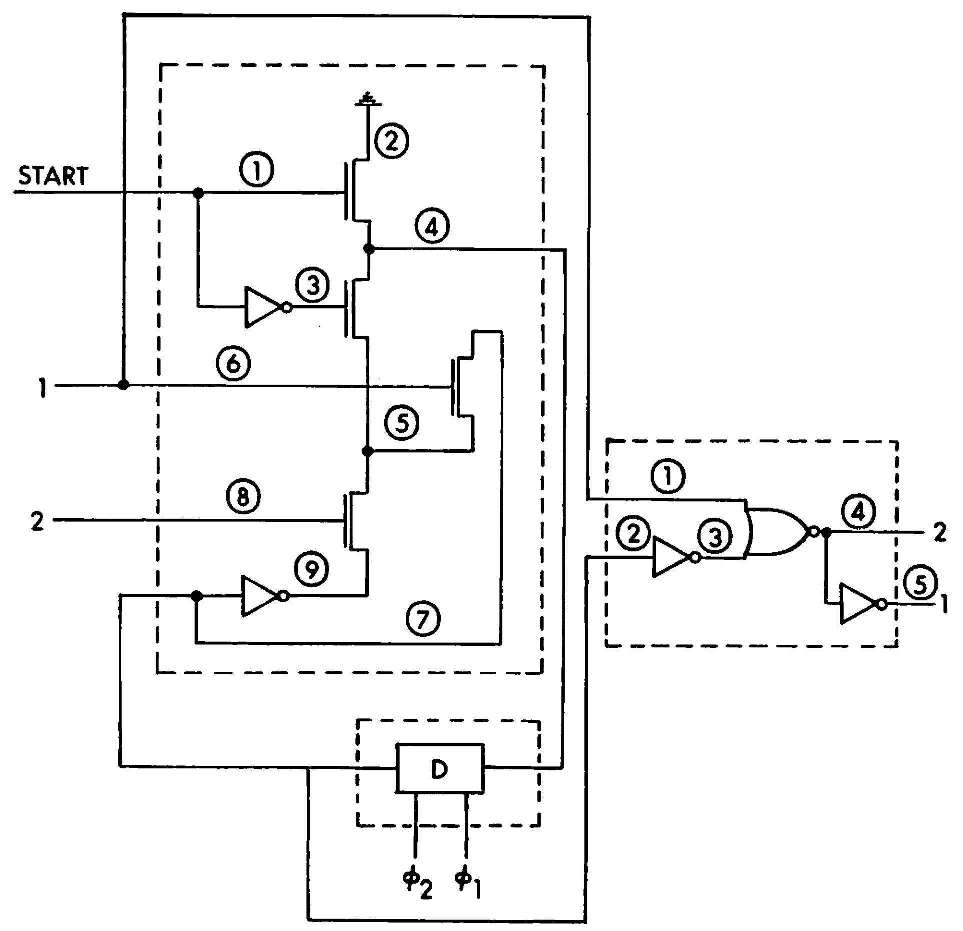

Figure 7. Divide by 2 counter 2-Logic diagram

Table 5b shows all the possible single stuck-at-faults with their corresponding values of q. From the table we obtain

$$q_1 = 1/18[1/2 + 1/2 + 1/4 + 1 + 1/4 + 1/8 + 1/4 + 1] = 31/144$$

Table 5b. Input Circuit Single Stuck-at-Fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/4	1/4
2	0	1/2
3	1/4	0
4	1/4	3/4
5	1/4	1/8
6	1/8	1/8
7	1/4	1/4
8	1/8	1/8
9	1/8	1/8

For the memory element we have $q_m = 1/2$. For the output subcircuit we have B = 1/2 and from Table 6 we have

$$q_2 = 1/10[1/2 + 1 + 1 + 1] = 7/20$$

Table 6. Output Circuit Single Stuck-at-fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/4	1/4
2	1/4	1/4
3	1/4	1/4
4	1/4	3/4
5	3/4	1/4

For the second divider the total number of wires is 15. The probability of failure for the input, memory, and output subcircuits are 9/15, 1/15, and 5/15 respectively.

We have the same values for the last divide-by-2 counter except for the output subcircuit. It has only one output so q_2 for the output circuit is 1/8[1/2 + 1/2 + 1/2 + 1] = 5/16. Finally for the last divide-by-2 circuit the probability of failure for the input, memory, and output circuits are 9/14, 1/14 and 4/14 respectively.

For a specific window size, that we will evaluate later, we calculate the probability of detecting a fault for each partition independently as if all input and output pins are observable. Now for the entire module we assume that one of the partitions is faulty. The probability of any partition being faulty is a function of the number of wires in the partition. The total number of wires in the module is 45. The probability that the control generator is the faulty partition is 8/45. The probabilities of the first, second, and third divide-by-2 partitions being faulty are 8/45, 15/45, and 14/45 respectively. Now we notice that if a partition is buried (e.g. the second divide-by-2 counter 4) and faulty then we need the error to propagate through all the other partitions to be observable on the output of the module so that it can be detected. From Figure 8 we can deduce the following probability of detecting a fault for m input vectors

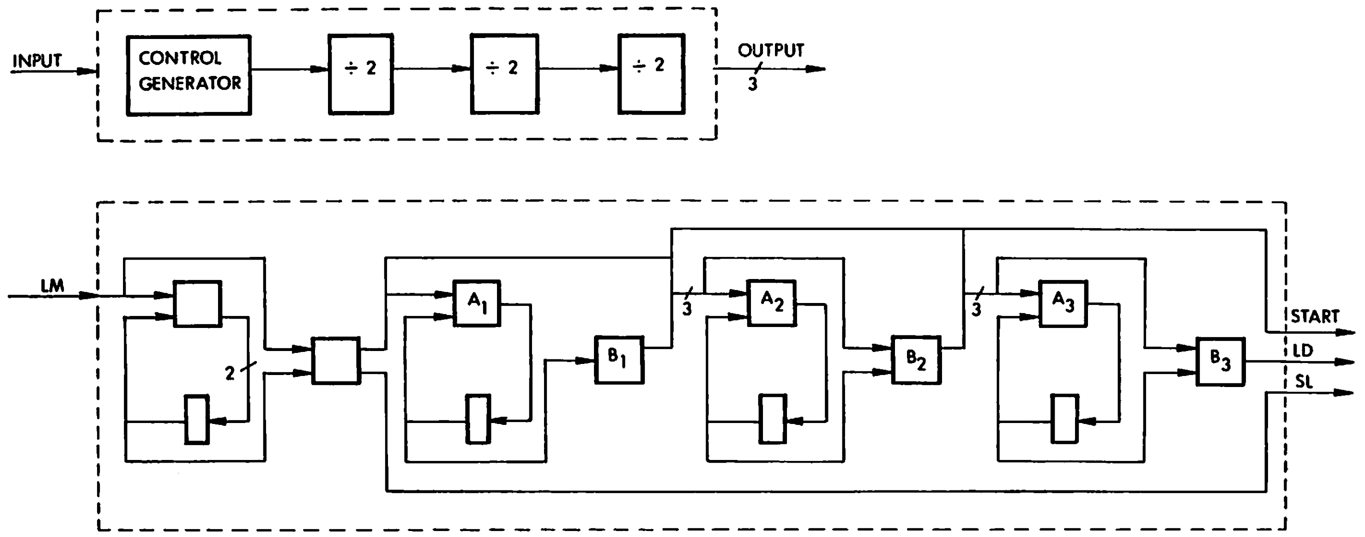

Figure 8. The control unit module

given that one partition is faulty:

$$P_{d(\text{control unit})}(m) = P_1 P_{d_1}(m) + P_2 P_{d_2}(m, A_2, B_2, A_3, B_3)$$
$$+ P_3 P_{d_3}(m, A_3, B_3) + P_4 P_{d_4}(m) \qquad (4)$$

P_d for each partition can be easily obtained by executing the routine for evaluating the probability of detection for sequential circuits.

3.2 Quotient Unit

The quotient unit will be divided into nine partitions. One partition is a combinational circuit and the rest are sequential circuits. Figure 9 shows a block diagram of the quotient unit. We divided the module so that we have seven partitions that are similar. We will examine partition 1, 2, and 3 as representatives of the module variations.

3.2.1 Partition 1

Partition 1 is a combinational circuit with six inputs and two outputs. Figure 10 shows a logic diagram of partition 1. The inputs are START, LD, T_f, SL, d_0, and d_1. The outputs are y'_7, which is an input to bit seven of the R register, and z_6, which is an input to bit six of the Q register. Table 7 shows the single stuck-at-fault set with their corresponding q values. From the table we find q to be equal to 0.1938.

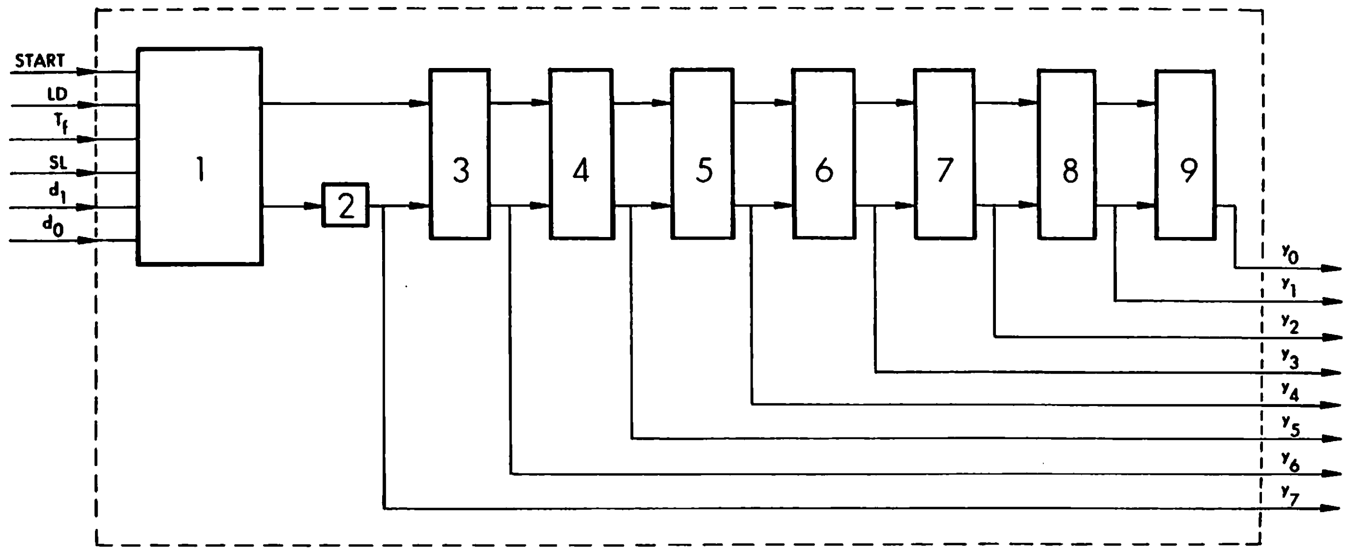

Figure 9. Quotient unit block diagram

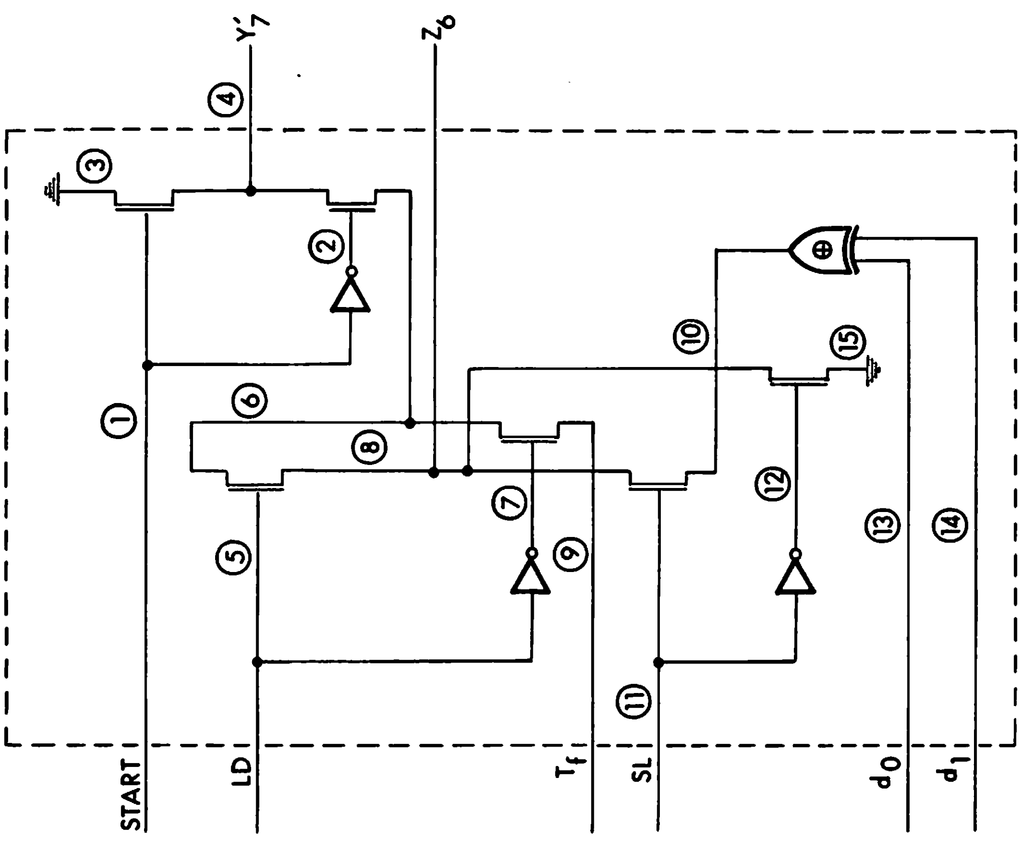

Figure 10. Partition 1-Logic diagram

Table 7. Partition 1 Single Stuck-at-fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	8/64	12/64
2	20/64.	0
3	0	32/64
4	12/64	52/64
5	8/64	8/64
6	16/64	16/64
7	12/64	0
8	16/64	16/64
9	8/64	8/64
10	16/64	16/64
11	16/64	12/64
12	12/64	16/64
13	8/64	8/64
14	8/64	8/64
15	0	8/64

3.2.2 Partition 2

Shown in Figure 11 is a logic diagram of partition 2. The circuit is a Moore type sequential machine with one input, one memory element, and one output. The circuit parameters can be easily obtained to be $q_1 = 1/2$, $A = 1$, $q_m = 1/2$, $q_2 = 1/2$, and $B = 1$. Now we assume that one part (input, output, or memory circuit) is faulty. We want to calculate the probability of detecting a fault in the circuit. The total number of wires is 3, hence the probability of each part being the faulty one is 1/3, given that one part is faulty.

3.2.3 Partition 3

Figure 12 shows a logic diagram of partition 3. Partition 3 is a Mealy type sequential circuit with five inputs (START, LD, y_1, and z_1, and z_0), two memory elements and one output (y_0). Table 8 shows a single stuck-at-fault set for the input circuit wires from which we obtain q_1 to be 0.2552. From the truth table of the input circuit we have $A = 0.3548$. For the memory $q_m = 1/2$, and for the output circuit $q_2 = 1/2$ and $B = 1$.

For this sequential circuit we have a total of 15 wires, so the probabilities of failure for the input, memory, and output circuit are 12/15, 2/15, and 1/15 respectively.

Now for each partition we can find the probability of detecting a fault if one exists. This can be accomplished by

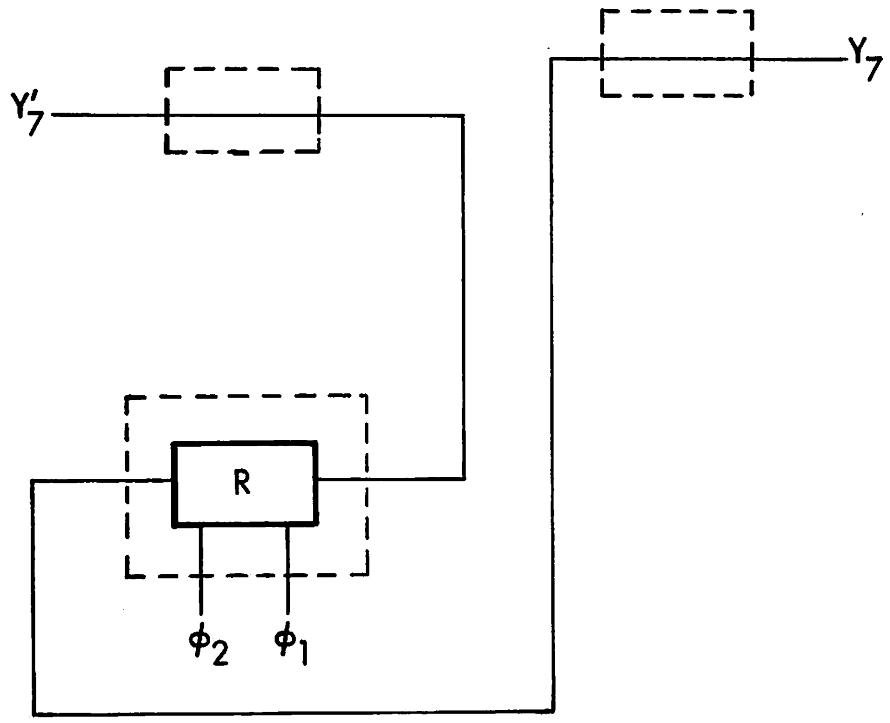

Figure 11. Partition 2-logic diagram

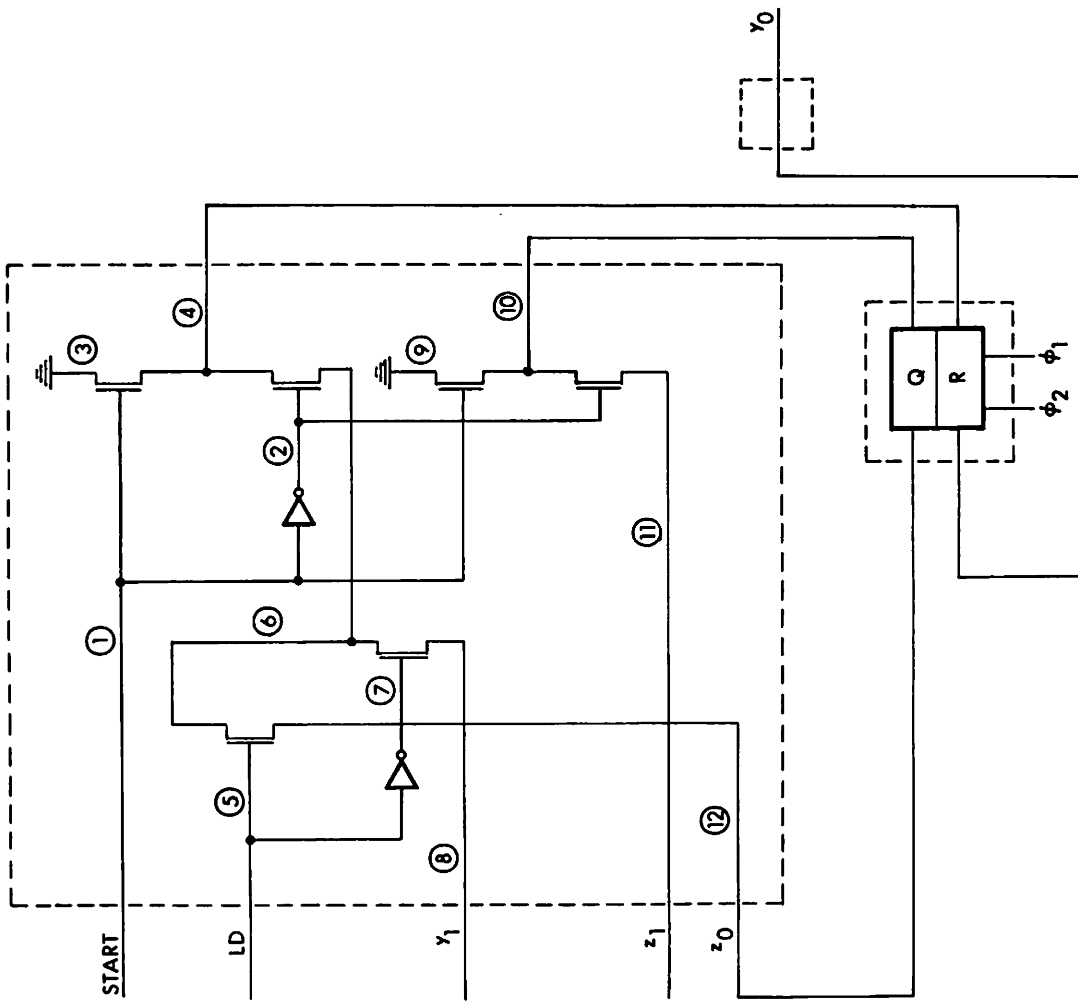

Figure 12. Partition 3-Logic diagram

calling the detection probability routine for sequential circuits.

Looking at Figure 9 and assuming that one of the partition is faulty we will evaluate the probability of detection for the entire module. The total number of wires in the module is 123, so the probability of failure for partition 1, 2, and 3 are 15/123, 3/123, and 15/123 respectively. The rest of the partitions are the same as partition 3. From Figure 9 we see that the output of partition 1 needs to propagate through partitions 2 and 3. The rest of the partitions have primary (observable) output leads, so the probability of detecting a fault for the entire module is

$$P_{d(\text{quotient unit})}(m) = P_1 P_{d_1}(m, A_3) + P_2 P_{d_2}(m) + 7\, P_3 P_{d_3}(m) \quad (5)$$

where m is the number of input vectors.

Table 8. Input Circuit Single Stuck-at-fault Set

Line Number	q·	
	s - a - 0	s - a - 1
1	3/8	3/8
2	3/8	0
3	0	1/2
4	1/4	3/4
5	1/8	1/8
6	1/4	1/4
7	1/8	1/8
8	1/8	1/8
9	0	1/2
10	1/4	3/4
11	1/4	1/4
12	1/8	1/8

3.3 Product Unit

The product unit is a combinational circuit with seven inputs and 34 outputs (see Figure 13). In this circuit we will consider a lower bound value for q. Figure 15 gives the Boolean function for each output pin. We note that T_2 is a function of the most number of input pins. From Figure 14 we find that q for T_2 is equal to 1/2. This means that half of the input set (2^7) are test vectors for any single stuck-at-fault on any line of the circuit shown in Figure 14. Since only one input line is left, then at least half of the total input set for the product unit is a test for any stuck-at-fault in the entire circuit. From equation (3) we obtain

$$P_{d(\text{product unit})}(m) = 1 - (1-q)^m \qquad (6)$$

where m is the number of input vectors. Finally we note that a realization of all the output functions gives us a total of 32 wires in the module.

3.4 Remainder Unit

The remainder unit is mainly a memory unit. It consists of 32-8 bit shift registers. Figure 16 shows a block diagram of the remainder unit. It has 33 input lines and one output line. We will divide the remainder unit into 33 partitions. Partition 1 is a simple 8-bit shift register. Partitions 2-32 are exclusive OR gates with an 8-bit shift registers. Partition 33 is an exclusive OR gate. We will examine each partition separately.

3.4.1 Partition 1

Figure 17 shows a logic diagram of partition 1. The circuit is a Moore type sequential circuit. It has one input, eight memory elements and one output. The circuit parameters are given as follows: $q_1 = 1/2$, $A = 1$, $q_m = 1/2$, $q_m = 1/2$, $q_2 = 1/2$, and $B = 1$. The number of wires in the circuit is 17. The failure probabilities of input, memory, and output circuit are 8/17, 8/17, and 1/17 respectively. The probability of detecting a fault in this circuit can be obtained from the sequential circuit routine.

3.4.2 Partition 2

Partition 2 is a Moore type sequential circuit with two inputs, eight memory elements, and one output (see Figure 18). From the logic diagram we can deduce that $q_1 = 1/2$, $A = 1/2$, $q_m = 1/2$, $q_2 = 1/2$, and $B = 1$. The total number of wires is 19. The probability of failure for each part is 10/19, 8/19, and 1/19, so the probability of detection can be calculated from the sequential circuit routine. The next 31 partitions will have the same parameters as partition 2.

3.4.3 Partition 33

Partition 33 is an exclusive OR circuit with q = 1/2. The total number of wires for the entire module is 609, so the failure probability of partition 1 is 17/609. The failure probability for the next 31 partitions is 19/609. The failure probability for the last partition is 3/609, so the detection probability in the remainer unit for m input vectors, assuming that one partition if faulty, is

$$P_{d(\text{remainder unit})}(m) = P_1 P_{d_1}(m, (1/2)^{32})$$

$$+PP_d(m,(1/2)) + PP_d(m,(1/2)^2) + \dots + PP_d(m,(1/2)^{31})$$

$$+P_{d_{33}}(m)\ P_{33} \qquad (7)$$

Note that most of the partitions are buried. The last partition is the only one with an observable output.

3.5 I/O Unit

This unit handles the input/output operations. The I/O unit is small enough so that we can evaluate the circuit parameters without dividing the circuit. Figure 19 shows a logic diagram of the I/O unit. The circuit is a Moore type sequential circuit with three inputs (DIN, SL, and d_0), three memory elements, and two outputs (DOUT, d_1). Using the truth table of the input circuit we obtain $A = 0.7333$. Table 9 shows all possible single stuck-at-faults and their corresponding q values. Using this table we find $q_1 = 0.3661$.

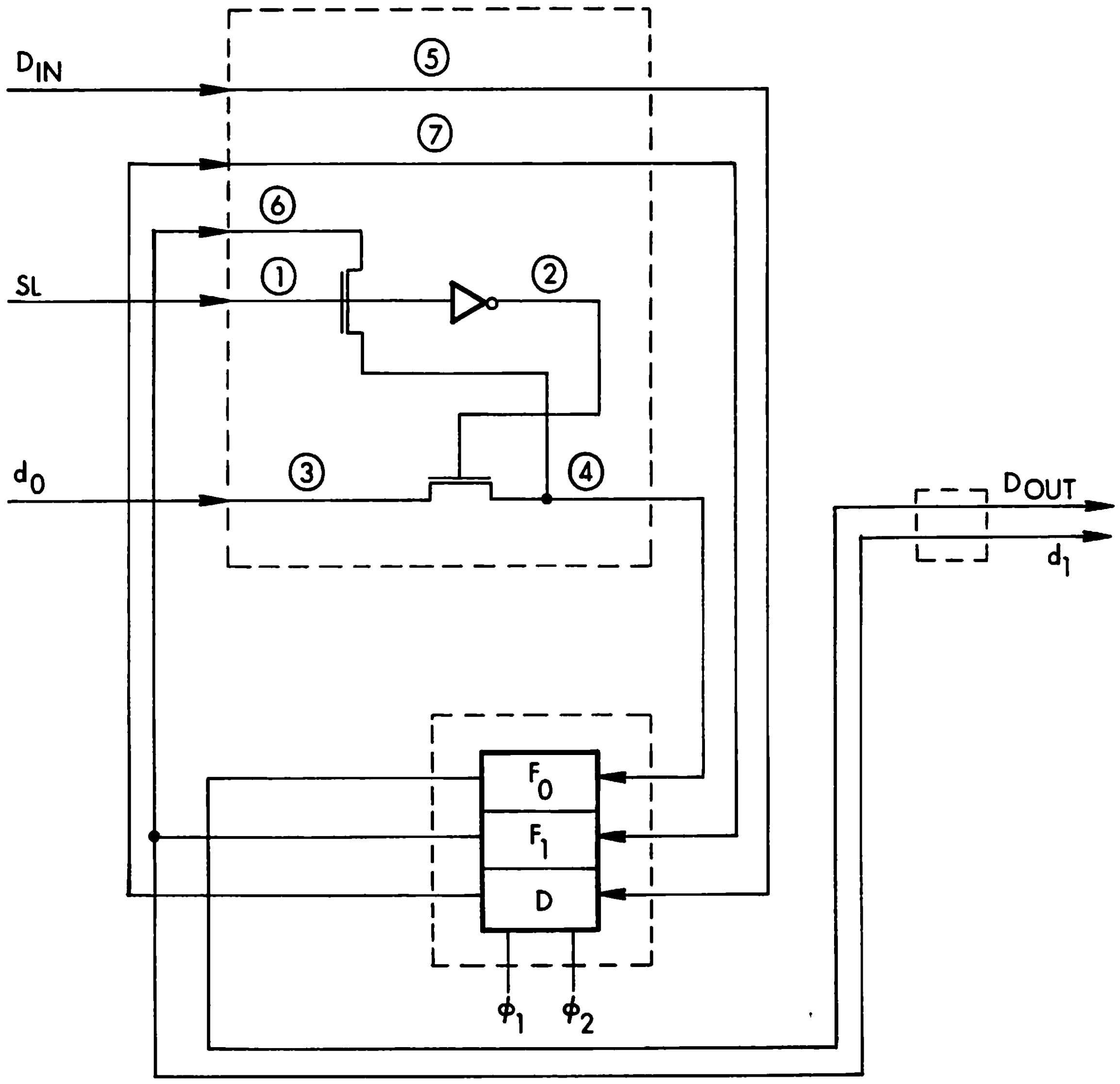

Figure 19. I/O unit Logic diagram

For the memory we have $q_m = 1/2$. The output circuit gives us $q_2 = 1/2$ and $B = 1$. The total number of lines in the circuit is 12, so the failure probability for the input, memory, and output circuits are 7/12, 3/12, and 2/12 respectively. Given these parameters and a specific window size we can evaluate the probability of detection.

Table 9. Input Circuit Single Stuck-at-Fault Set

Line Number	q	
	s - a - 0	s - a - 1
1	1/4	1/4
2	1/2	1/8
3	1/4	1/4
4	1/2	1/2
5	1/2	1/2
6	1/4	1/4
7	1/2	1/2

4. THE DESIGN OF THE ROVING EMULATION ENGINE

The current design of the roving emulation engine is essentially complete at the architectural level, with some portions complete to the register transfer level. Future work will include the implementation of hardware emulators for each of the individual components of the emulator and their configuration into systems in order to obtain performance data required to optimize the final design. Eventually it is hoped that the emulator will be implemented on a full wafer of silicon as a maximally configured system with external memory additions possible. In this state, the system will configure itself through self test so as to make the maximum possible number of resources available.

4.1 System Level Architecture

The USC roving emulator family of computer architectures consists of a group of optionally self timed subsystems. Each subsystem is designed for implementation on a single VHSIC chip. The subsystem can be structured in sets to provide performance and reliability tradeoffs. A system consists of the following sets of subsystems. (See Figure 20)

- A set of memory processors which maintain and dispatch global information regarding the state and description of the system being emulated to the 'delta-net' processors on request. They also store observations from the bus watcher providing the data necessary for emulation, and forward information to an output port to allow the results of emulation to be externally observed. A timing wheel is provided

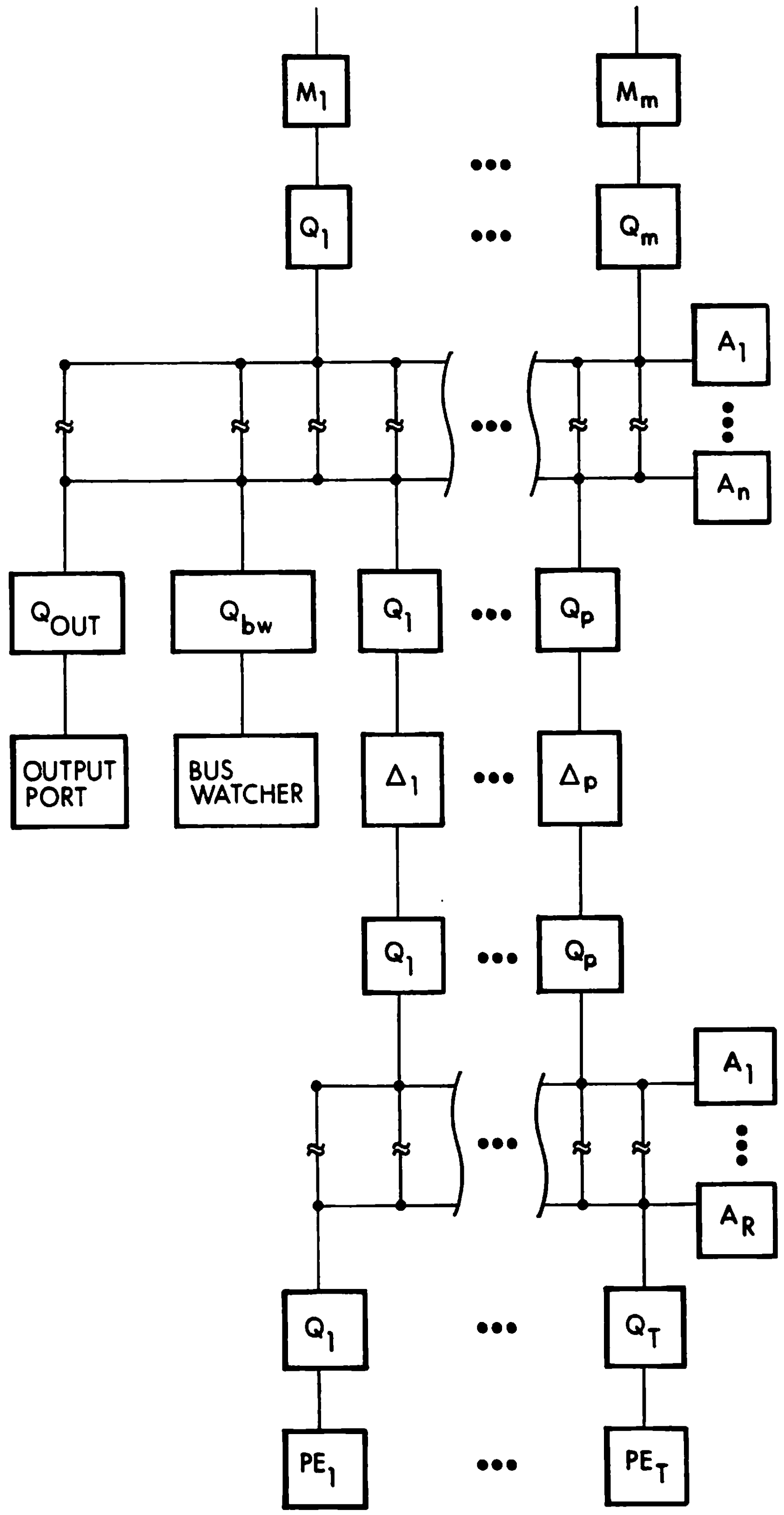

Figure 20. Roving emulator architecture
(queues are in pairs as are their links)

for time based simulation, and system state information is maintained in tables.

- A set of 'delta-net' processors which supervise the emulation by allocating code from externally compiled descriptions to processing resources. This implements the operating system features necessary to implement a 'delta-net' processor in hardware.

- A set of processing elements which perform emulation of various types of hardware. The processing elements themselves are strictly forward branching sequential instruction processors and have no looping capabilities. Detection hardware in the PEs allows them to detect changes requiring delta-net output.

- A set of queueing elements which act as input and output buffers between resources and arbiters. These are simply fifo queues implemented in hardware.

- A set of input and output busses with arbiters. This provides physical links for passing information between system resources.

- A bus watcher for observing emulated processors. This provides the data from observation of emulated processors to the system.

- An output port for transmitting the results of emulation to the system under test or external observers.

The current system architecture allows for:

- Multiple byte length data transfers between all resources.

- Up to 16 of each type of system resource (Memory, Delta-net, PE).

- Up to 16 busses and arbiters for supervising information transfers.

- Restructurable queued storage of information awaiting processing.

- Externally expandable resource memories.

- Independent or shared processor clocks and/or asynchronous components

In its minimal configuration, the emulator consists of one memory processor with external memory, two input and output busses, a single general purpose processing element, four queues, two bus arbiters, a bus watcher, a delta-net processor, and an output port. This comes out to a modest 11 component configuration which is capable of emulating virtually any computer system at a considerable slow down factor. Configurations of this size could easily to used in home microprocessors, long range space flight systems, or automatic test equipment in the field without substantial power, weight, or space utilization, and without significant performance degradation to the system under test.

In its maximal configuration, the current design could take over 200 components and could still fit on a large board in many commercial processors or on a single full wafer implementation with external memory. It is customizable to the performance characteristics of the system being emulated. From the standpoint of reliability, the maximal system could survive a large number of hardware faults and still be capable of operation with reduced performance. The performance characteristics of a full scale system depends heavily on the system being emulated, the description of that system, and the software used to generate internal code for the emulator.

5. EMULATION ENGINE PERFORMANCE

As an example of the analysis of a configuration, we have taken the design of the encoder and broken it up into the modules from which it was designed. This is an example of a circuit which up until now has been a multiple component system, and as such exemplifies the application of the emulator to such systems as well as its possible ramifications to testing chips having busses which are observable during normal operation. A major assumption used here is that the emulator has the same clock rate as the system under emulation.

The encoder chip is composed of five subunits:

1. Product Unit – the PLA described previously
2. Remainder Unit – 8 bit shift registers chained through xors
3. Quotient Unit – 8 bit shift registers with parallel load
4. Control Unit – combinational logic and an 9 bit counter
5. IO Unit – registers encoder chip and a switch

In operation, the chip goes through 256 clock cycles at a time, translating input to output. At the beginning of each 256 bit cycle, the chip clears its internal state, leaving all state information at 0s. This is rather handy from the point of view of the emulator since there is no need to dump the initial state of the system. Even if it was desired to dump the initial state, because of the large memory in the remainder without the addition of a large number of pins to the circuit. It would take almost 256 cycles to dump the state. This could be done, but has no advantage in this case, and in fact incurs a significant disadvantage since by removing the requirement to dump state, the system being emulated can execute with no performance degradation. In addition, we decided not to used the final state of the system since this would incur a large overhead with very little potential advantage. Because of this initialization behavior, windows of more than 256 cycles are meaningless, thus the window size is restricted to be from 1 to 256 cycles of the encoder clock.

Each of the modules in the system will have its own slow down factors for emulation. As an example, the PLA can do all 204 operations required to simulate it in one clock cycle. This slow down factor is also effected by the configuration of the emulator, since with 16 processing elements, it may be possible to do 204 operations in only 14 emulator clock cycles. Even more interestingly, the emulator might be configured to have enough memory to store the entire PLA transform in its memory, thus emulating the operation with a single table lookup. Each of these possibilities requires a cost performance tradeoff which must be analyzed in the light of external factors such as cost, power, area, and reliability.

The remainder unit is an example of a subsystem composed almost entirely of memory. It is usually very difficult to emulate memory because it requires at least as much memory in the emulator as in the system and a dump of the entire initial state. In this case however, the memory is not so large that it is unemulateable, and the state is initialized so as to be known every 256 cycles of the machine. For this reason, it is reasonable to emulate the memory using 33 8 bit memory locations, shifting each and performing the required xors on a single processing element. This much memory is available on our simple processing element, and the capability to shift and mask properly is available. The actual encoder chip does this entire process in a single clock cycle, whereas it requires at least a shift and 2 masks per register in our simple processing element. For this reason, we could expect a slow down factor of about 90 for the emulation of this special purpose memory. Again, it would be easy to use table lookups at the expense of space, or even to use a better processing at the expense of space, or even to use a better processing element than our simple one to produce significant improvements.

The quotient unit is simply a pair of shift registers with parallel load and output. This requires very little processing power in the emulator, and can easily be accomplished with 4 emulation cycles per encoder cycle.

The control unit has only 3 bits of state (used for the divide by 8 counter which produces the c2 output), and is relatively easy to emulate. Its purpose is the generation of nonoverlapping clocks, and as such it is an asynchronous device which , although it could be modeled syncnronously with slight loss of information, would best be modeled using the full power of the emulators delta-net capability. This would produce a slow down factor of nearly 20 for the entire control circuit as opposed to a slow down of only 4 or 5 for a synchronous model.

The IO unit is extremely simple to emulate, requiring only two memory bits and a gate. In this case, the emulator would produce a slow down factor of only 3 or 4, and no optimization would be worthwhile.

The previous analysis does not consider several points, such as the control overhead incurred by switching between emulation of different components and loading their data into the processing elements. Because of the queuing used in the emulator and the ability to off load sequential processing to the PEs this overhead should be expected to average about 4 to 5 cycles per operation, and this constant should be added to each emulation time.

6. A SIMPLE BUS WATCHER

A simple bus watcher would serve for the case of the encoder system, consisting only of a local memory, interface to the emulator via a prot into one of the system busses, and external interface via clocked parallel load registers and a single multiplexer. This simple architecture is pictured in Figure 21. It must be noted that many systems will not require specialized interface to the emulator, but rather will interface directly through system busses. In special cases, however, it is relatively easy to handcraft an interface to the emulator as required.

7. EVALUATION OF THE EXPECTED VALUE OF THE ERROR LATENCY

Since a codeword in the encoder contains 255 symbols, then the computation of a complete encoded codeword requires 255 symbol cycle. Each symbol is represented by 8 bits. The time required to complete a codeword is 8 x 255 = 2040 clock cycles. One convenient characteristic of the encoder is that the internal state, except the I/O unit, is reset to zero after a codeword cycle (2040 clock cycles). We will use this characteristic to our advantage so that we can avoid the interruption time to obtain initial and final state of a module. This will save us time, especially in the remainder unit, since it consist mainly of memory.

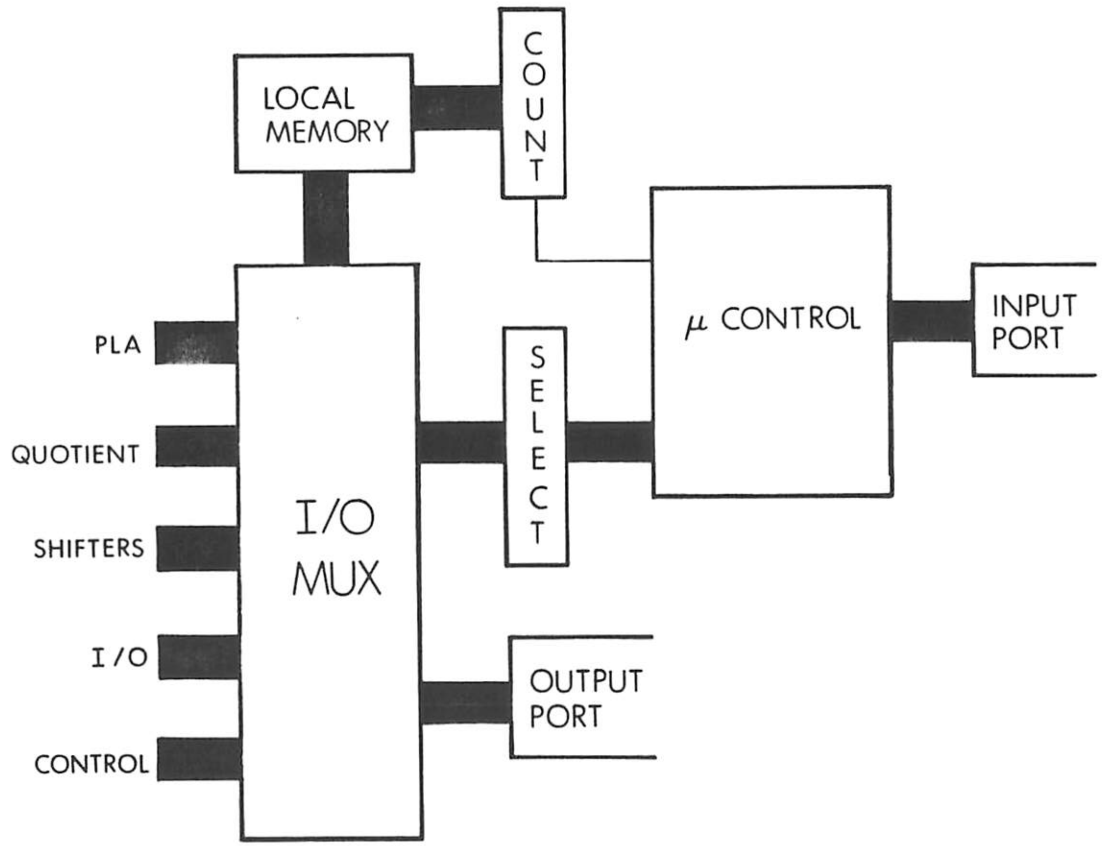

Figure 21. Roving emulator - special purpose bus watcher for encoder system

Fig 21

We will have the RE test each module in at least one codeword cycle. Hence we will make $T_i \leq 2040$ for $1 \leq i \leq 5$. T_i is the testing period for module i. Δ will be zero for all the modules except for the I/O unit. Since we have three memory elements in the I/O unit, then it is safe to assume that it takes three shifts to dump the state of I/O unit. We will assume during the whole course of analysis that detection is accomplished by comparing the output of the actual and the simulated circuit only. We note that since every module in the system is always utilized, then at every clock cycle we have a new output which means that m = w. Now we will determine the window size for each module, and then calculate the probability of detection assuming that one of the partitions is faulty.

7.1 Control Unit

Since k = 20, to obtain the window size we let w + 20w = 2040. Then we have

$$21w = 2040$$
$$w = \lfloor 2040/21 \rfloor = 97$$

Thus

$$k\,w = 97 \times 20 = 1940$$

But

$$T = w + 20w = 2037$$

Thus the system is idle 2040 – 2037 = 3 clock cycles before it starts testing the next module. For a window of size 97 we have $P_{d1}(97) = .7986$, $P_{d2}(97) = .9025$, $P_{d3}(97) = .6343$, and $P_{d4}(97) = .6082$. Substituting these values into equation (4) we obtain $P_{d(\text{control unit})}(97) = 0.3839$

7.2 Quotient Unit

Since k = 4, for the window calculation we obtain

$$w + 4w = 2040$$
$$5w = 2040$$

hence

$$w = \lfloor 2040/5 \rfloor = 408$$

Therefore,

$$w = 408,$$
$$kw = 1632, \text{ and}$$
$$T = 2040$$

For a window size of 408 we have

$P_{d_1}(408) = 1$,

$P_{d_2}(408) = .8112$, and

$P_{d_i}(408) = .9216$ for $3 \leq i \leq 9$

Substituting into equation (5) we obtain

$$P_{d(\text{quotient unit})}(408) = 0.8501$$

7.3 Product Unit

Since k = 204, we have that

$$205w = 2040$$
$$w = \lfloor 2040/205 \rfloor = 9$$

The emulation time is kw = 1845, and the idle time is

2040 – 1845 = 195 clock periods.

Using equation (6) we obtain $P_{d(product)}(9) = .998$

7.4 Remainder Unit

Since k = 90 then 91w = 2040 and w = 22. The emulation time is 1980, hence T = 2002. This gives us an idle time of 38 cycles. Using a window size of 22 we obtain

$$P_{d_i}(22) = .8694 \qquad 2 \leq i \leq 32$$

$$P_{d_{33}}(22) = 1$$

Using equation (7) we have $P_{d(remainder)}(22) = 0.0316$

7.5 I/O Unit

Since k=4, then we have 5w = 2040, hence w = 408. The emulation time is 1632 and the idle time is 0. Using the sequential circuits detection routine we obtain

$$P_{d(I/O\ unit)}(407) = .8251$$

7.6 Evaluation of E_1, E_2, E_3, E_4, and E_5

Let WA_i be the idle or wait time for module i. Let k_i and w_i be the emulation and window size for module i. Figure 22 shows a complete pass of the roving emulator through the whole system. Let E_i be the expected value of error latency for the system assuming that module i is faulty. Then, considering

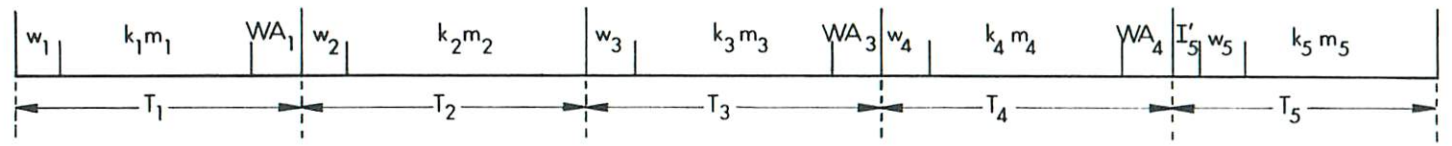

Figure 22. A roving emulator system cycle

22

each module, we have:

1) for control unit

$$n_1' = 2.6048$$

Thus the average wait time $= (n_1' - 1)(3x2040+2043+WA_1)$
$= 13105$ clock cycles

and the average service time $= n_1' \times T_1 = 5306$ clock cycles

Hence

$$E_1 = 13105 + 5306 = 18411 \quad \text{clock cycles}$$

2) for the quotient unit

$$n_2' = 1.1763$$

Then the average wait time $= (n_2' - 1)(3x2040 + 2043) + 2040$
$= 3479$ clock cycles

and the average service time $= n_2' \times T_2 = 2400$ clock cycles

Hence

$$E_2 = 3479 + 2400 = 5879 \quad \text{clock cycles}$$

3) for product unit

$$n_3' = 1.002$$

Then the average wait time $= (n_3' - 1)(3x2040+2043+WA_3) + 2x2040 =$ 4098 and the average service time $= n_3' \times T_3 = 1858$.

Hence

$$E_3 = 1858 + 4098 = 5956 \quad \text{clock cycles}$$

4) for remainder unit

$$n_4' = 31.6456$$

Then the average waiting time = $(n'_4 - 1)(3 \times 2040 + 2043 + WA_4) + 3 \times 2040 = 257445$ and the average service time = $n'_4 \times T_4 = 63354$.

Hence

$$E_4 = 63354 + 257445 = 320799 \quad \text{clock cycles}$$

5) for I/O unit

$$n'_5 = 1.2149$$

Then the average waiting time = $(n'_5 - 1)(4 \times 2040) + 4 \times 2040 = 9914$ and the average service time = $n'_5 \times T_5 = 2482$.

Hence

$$E_5 = 2482 + 9914 = 12396$$

Now we will calculate the average value of the error latency assuming that one of the modules is faulty. The total number of lines in the chip = 45 + 123 + 32 + 609 + 12 = 821. If we assume that the probability of failure for a module is directly proportional to the number of lines in the module, then we have

$$E = 1/821\ [828495 + 723117 + 59456 + 195366591 + 148752] = 240{,}105 \quad \text{clock cycles}$$

Hence on the average we need to encode $240{,}105/2{,}040 \cong 118$ words to detect a fault in one of the modules in the system.

Since the clock frequency of this chip is 2MHz, then the period of the clock is $.5 \times 10^{-6}$ sec or .5 usec. So on the average, assuming a continuous flow of information through the chip, the roving emulator will detect a fault in .5x240,105 = 0.1201 sec, which is very fast keeping in mind that this was a worst case analysis taking the minimum configuration of the roving emulator.

8. CONCLUSIONS

In this report we have demonstrated the feasibility of applying the concept of roving emulation to the testing of a digital system. We have presented the basic theory of how roving emulation works, and the basic equations used to calculate error latency. We have applied these results to an actual signal processing system, namely an encoder. The results indicate that the average time required to detect a fault is only a tenth of second. The system being tested consists of five different units, including a control unit, an arithmetic unit, and an I/O unit. The analysis techniques developed were applicable to all units of the system, again demonstrating the versatility of this new approach to testing.

9. REFERENCES

1. Task I: Design for Testability and Reliability, prepared by M.A. Breuer and J.P. Hayes, Annual Report for the period September 1980 – September 1981, prepared for the Naval Electronics System Command under Contract No. N00039-80-C-0641, (see Chapter 5).

2. M.A. Breuer and A.A. Ismaeel, "Roving emulation as a fault detection mechanism," submitted to the Int'l Fault Tolerant Computing Conference, June

1983.

3. M.A. Breuer and F. Cohen, "A Roving Emulation Engine," USC DISC report, under preparation.

4. T.K. Truong, I.S. Reed, I.-S. Hsu, K. Wang, and C.-S. Yeh, "The VLSI Design of a Reed - Solomon Encoder Using Berlekamp's Bit-Serial Multiplier Algorithm," submitted to IEEE Trans. on Computers.

5. M. Perlman and J.J. Lee, "Reed-Solomon Encoder - Conventional versus Berlekamp's Architecture," Inter Office Memo No. 3610-81-119, Jet Propulsion Laboratory, Calif., July 10, 1981.