Exercising the NON-VON Primary Processing Subsystem[1]

Bruce K. Hillyer
and
David Elliot Shaw

The NON-VON Project
Department of Computer Science
Columbia University

Revised April 1983

Abstract

The Primary Processing Subsystem of the NON-VON supercomputer potentially may comprise thousands of custom nMOS integrated circuits. It is vital that faulty components be detected and located. This paper provides a collection of algorithms to exercise the Primary Processing Subsystem so that the manifestation of latent faults may be observed.

Background

The NON-VON project is designing and implementing portions of a massively parallel partitionable SIMD computer, which in a full scale version would have on the order of one million processors. Readers interested in background information concerning this effort may wish to examine some of the following references [Shaw, 1982; Shaw et al., 1981; Shaw, 1980; Shaw, 1979].

---

[1] This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-82-C-0427.

1 Introduction

There are several dimensions to the testing of electronic systems. One dimension is temporal, and spans from real-time error detection to off-line testing. Another dimension proceeds from concrete to abstract, with concerns such as jitter, speed, and waveforms at one end, and functional behavior of circuits at the other. A third dimension involves testing at various stages in the life cycle. Newly designed circuits are implemented and tested for design errors, while devices previously installed and working are tested to see whether they are still functioning correctly. Between these points we have chips which have just arrived from the maker, and which must be tested to see if there were errors during fabrication[Srini, 1977; Galiay et al., 1979].

The subject matter of this paper is the off-line functional testing of the chips used in the NON-VON Primary Processing Subsystem (PPS). The methods presented here for this purpose may also be applied to individual PPS chips when they are received from fabrication.

2 Exercising vs. Testing

Research in testing centers on finding minimal sets of inputs which will expose all possible faults of a certain class (such as stuck-faults). A principal aim of such research is to enable the testing of devices more quickly than by exhaustive application of all possible input sets [Sridhar et al., 1981]. A concomitant goal is to ensure that the chosen input set will force all potential faults to manifest themselves. It should be noted that for some circuits, there are faults for which it is impossible to test. Although some work has been done in the areas of pattern sensitive fault detection and detection of intermittent faults, in general it is not not known how to perform a complete test for these circuit errors in any feasible amount of time.

Our goal here is different. We are faced with a practical problem requiring a pragmatic solution. The NON-VON Primary Processing Subsystem (PPS) potentially may comprise thousands of custom nMOS chips. These chips must be tested after fabrication, and then again from time to time after they are installed in the PPS. We cannot expect to answer the question: "Is this chip working correctly?" with certainty. We seek an indication that the chips are working, and reasons to be confident in the correct functioning of our hardware. To this end, several programs are presented which will exercise the PPS.

The following collection of exercises was chosen after some thought, but is not the product of careful research in the area of device testing. No representation is made concerning the degree of fault coverage provided. The intention of the exercises is, however, to provide opportunities for the discovery of shorts, stuck-faults, pattern-sensitive faults within PE's and

interference among PE's. The exercises check functional properties of the PPS, and as such are independent of the particular VLSI implementation. It should be noted, however, that tests developed using information about the layout could be more efficient, and could be proven to detect all instances of certain classes of faults.

The total time required to perform this collection of exercises on a PPS consisting of 1 million PE's, with a NON-VON instruction executed every 2 us, is approximately 2 1/2 minutes. This time is in no sense optimal.

3 Fault Location

The exercises given here to detect failures in an arbitrarily large tree of NON-VON PE's can also be used for fault location down to the chip level. This is made possible by an important property of the scheme used to allocate PE's to chips. Leiserson's technique for chip-level layout of a tree machine, given in [Leiserson 1981], places a complete subtree and one additional interior node of the tree on each chip. All three I/O ports of the interior node are accessible, as is the root of the complete subtree. As a consequence, the exercises can be applied recursively to locate a faulty chip. In particular, if an exercise fails, the two children of the current root are physically accessible. Each may be exercised as a new root by removing the chip containing the current root and inserting a suitable bypass connector. If both subtrees are ok, then the current root is bad. If just one subtree shows a fault, the exercises may be applied to the failing subtree. If both subtrees fail, we may conclude there are multiple failures in the PPS, and examine both subtrees independently. The process will end in one of two ways. Either a bad interior node will be found when both of its subtrees pass the test, or the failure will be pursued until a subtree contained entirely inside a chip is found bad. In either case, a single faulty chip is located by the application of the exercise procedure at most twice for each level in the tree of PE's. After a faulty chip is replaced, the exercises are used again since multiple faults may be present. For example, one application of the location procedure will only find the bottom-most bad chip in a chain of faulty descendants.

4 The Exercises

4.1 Exercise 1

Goal: The aim of this exercise is to check 1-bit linear neighbor communication in both directions, as well as the correct functioning of the A1 and I01 flags.

Instructions tested: LOADA1 IO1, STOREA1 IO1, SEND1 LN, SEND1 RN, RECV1 LN, RECV1 RN

Time: Linear in the number of PE's, taking about 8 seconds for a million PE PPS running 1/2 million instructions per PE per second.

Method:

For this test, we consider the entire PPS to have an LSSD-like shift register linking the A1 and IO1 flag registers of all PE's along the linear-neighbor communication path. A long, random sequence of bits is to be shifted into the linear-neighbor path through the PE's, and examined as it comes out the other end. A stuck-at-0 or stuck-at-1 fault in an IO line or A1 or IO1 flag will be manifested by a constant output. Because of the size and randomness of the test pattern, there also is an opportunity to observe interference errors and intermittent problems.

NON-VON has two instructions which will shift bits through the PEs to the right. The first is RECV1 LN, and the second is SEND1 RN. In a loop, it would be necessary either to alternate these two instructions, or to transfer between IO1 and A1 at each iteration. There are analogous instructions for shifting left. Rather than performing just one of these instructions repeatedly, on each iteration this exercise takes a random choice among the four possible instructions, with a bias of, say, 90% for shifting to the right. Note that an extra load or store may be necessary to make an iteration compatible with that which preceeded.

Comments and limitations: If the bit string which comes out the right end of the linear-neighbor sequence of the PPS matches the input string, we can be confident in the functioning of the facilities which have been exercised so far [Agrawal and Vishwani, 1981]. Note that we don't know whether the contents of A1 or IO1 will be corrupted by setting or clearing some other flag or register. When performing further tests, after marking (in A1) PE's which fail the test, we can shift out the A1 contents and know precisely which PE's were bad.

4.2 Exercise 2

Goal: This exercise checks the functioning of some logical operations.

Instructions tested: STOREA1 B1, LOADA1 B1, SET, CLEAR, NEGATE, OR, XOR

Time: Linear in the number of PE's, taking about 30 seconds for a million PE PPS running 1/2 million instructions per PE per second.

Method:

```
STOREA1 B1     ; save the random bits left in A1 from exercise 1

      ; check if SET and CLEAR work
SET            ; assume temporarily we can set A1 in all PE's
{shift A1 out so the CP can check that A1 is really set}
CLEAR          ; check if CLEAR works
{shift A1 out so the CP can check it}
SET            ; check if SET works
{shift A1 out so the CP can check it}

      ; check if we can load the random bits from B1 into an A1 which
      ; contains both 0 and 1 to start with.
SET
LOADA1 B1      ; are the random bits still there?
{shift A1 out so the CP can check it}
CLEAR
LOADA1 B1      ; make sure we can load both 0 and 1 from B1
{shift A1 out so the CP can check it}

      ; try out NEGATE
LOADA1 B1
NEGATE
STOREA1 B1

      ; check if we can load the complemented random bits from B1 into an
      ; A1 which contains both 0 and 1 to start with.
SET
LOADA1 B1
{shift A1 out so the CP can check it}
CLEAR
LOADA1 B1
{shift A1 out so the CP can check it}


      ;  test OR
SET            ; 1 OR 1
STOREA1 B1
OR
{shift A1 out so the CP can check it}

SET            ; 0 OR 1
STOREA1 B1
CLEAR
OR
{shift A1 out so the CP can check it}

CLEAR          ; 1 OR 0
STOREA1 B1
SET
OR
{shift A1 out so the CP can check it}
```

```
CLEAR          ; 0 OR 0
STOREA1 B1
OR
{shift A1 out so the CP can check it}


      ;  test XOR
SET            ; 1 XOR 1
STOREA1 B1
XOR
{shift A1 out so the CP can check it}

SET            ; 0 XOR 1
STOREA1 B1
CLEAR
XOR
{shift A1 out so the CP can check it}

CLEAR          ; 1 XOR 0
STOREA1 B1
SET
XOR
{shift A1 out so the CP can check it}

CLEAR          ; 0 XOR 0
STOREA1 B1
XOR
{shift A1 out so the CP can check it}
```

Comments and limitations: For this exercise, it was first determined that A1 may be loaded and stored with respect to B1, and that B1 is not altered as a result of A1 being set or cleared. Subsequently, OR and XOR were tested exhaustively. This is the last exercise in which it will be necessary to shift A1 out frequently, as we may now use OR to accumulate 1-bits (indicating a detected failure, for example), and shift out these 1-bits at the end. XOR will be useful for comparing a result or stored value with what was expected.

4.3 Exercise 3

Goal: This exercise checks the functioning of load and store of A1 and B1 with respect to several of the flag registers. Many of these are needed for exercise 4, which checks inorder RESOLVE.

Instructions tested: {LOAD,STORE} {A1,B1} {A1,B1,C1,X1,Y1,Z1,IO1}

Time: Linear in the number of PE's, taking about 10 seconds for a million PE PPS running 1/2 million instructions per PE per second.

Method: First, we determine that X1 can be used to hold a value which will not be corrupted by changes to other flag registers. Then we use X1 to hold a failure indication; testing the other registers and OR'ing failure bits into X1. Finally, X1 is shifted out.

```
      ; Set X1 and see that it remains set
SET
STOREA1 X1
CLEAR
STOREA1 A1
STOREA1 B1
STOREA1 C1
STOREA1 Y1
STOREA1 Z1
STOREA1 IO1
LOADA1 X1
{shift out A1 so the CP can check it}
LOADB1 X1     ; check that B1 reads the same value as A1
XOR
{shift out A1 so the CP can check it}

      ; Clear X1 and see that it remains clear
CLEAR
STOREA1 X1
SET
STOREA1 A1
STOREA1 B1
STOREA1 C1
STOREA1 Y1
STOREA1 Z1
STOREA1 IO1
LOADA1 X1
{shift out A1 so the CP can check it}
LOADB1 X1     ; check that B1 reads the same value as A1
XOR
{shift out A1 so the CP can check it}

      ; Note that X1 is clear at this point

      ; Now that X1 is known to be ok, we do an O(4 * 6^2) test on
      ; A1, B1, C1, Y1, Z1, and IO1, checking that setting and clearing
      ; any of them leaves all other values unchanged.  Set/clear their
      ; value by setting/clearing A1, and storing it.  For each flag
      ; check (where possible), load both A1 and B1 from the register under
      ; test, and XOR them.  Any resulting 1 signifies an error; OR it into
      ; X1.  Then, SET or CLEAR to give A1 the expected (correct) value for
      ; the flag under test, load B1 with that register, and XOR again,
      ; ORing the result into X1.

      ; Next, repeat the 6-squared process described above, but for each
      ; STOREA1 <flag> used there, use the sequence:
```

```
        ;   {STOREA1 B1, NEGATE, STOREA1 <flag>, STOREB1 <flag>}.
        ; This will show that B1 can store as well as A1.

        ; Finally, load X1 into A1, and shift it out so the CP can see if
        ; there are any 1-bits (indicating faults).
```

Comments and limitations: We have not tested for interference from the 8-bit data path, nor from tree communication. At this point we can be fairly confident in the correct operation of the 1-bit data path, however.

4.4 Exercise 4

Goal: This exercise checks whether inorder resolve works.

Instructions tested: RESOLVE

Time: Linear in the number of PE's, taking about 40 seconds for a million PE PPS running 1/2 million instructions per PE per second.

Method: The pre-test checks whether: if the first and last A1 are set while all the rest are clear, then after RESOLVE, the last A1 will be clear. The main part of the test checks that if the vector of A1 values is 0000...01111...1, then after RESOLVE there will be a single A1 set, and it will be in the proper position. Further, if a single A1 is set, after RESOLVE it will still be set, and no other A1 will be. Finally, if no A1 is set, then after RESOLVE, no A1 will be set, and R1 in the control processor will be clear.

```
; Pre-test to see if a kill signal can be generated and propagated across the
;   PPS.  If we fail this test, there is no need to do the extensive one.

        ; Clear the A1 and IO1 registers
(nv-clear)
(nv-storea1 io1)

        ; For each of the following shifts, the CP inserts a 1 bit into its
        ; linear neighbor.  After the three shifts, the first and last A1 will
        ; be set, and all the rest will be clear.
(nv-recv1 ln)
(nv-send1 rn)
(nv-recv1 rn)

        ; Now RESOLVE, and the CP checks that the rightmost A1 was reset.
(nv-resolve)    ; CP checks (R1) is set
(nv-send1 rn)   ; CP checks that it receives a 0

; Main test of resolve.  X1 is clear initially, and has 1-bits indicating
```

```
;   faults OR'ed in after each iteration.  After the loop, the values in X1
;   are shifted out of the PPS, and examined by the CP.

        ; Clear X1 and IO1
(nv-clear)
(nv-storea1 X1)
(nv-storea1 IO1)

; At each iteration of the loop, we shift a 1-bit in from the right to get a
;   new test pattern of the form 000...011...1 , which is XOR'ed with the
;   previous test pattern to get a bit set in the single processor which
;   should win the RESOLVE.  We resolve twice, each time checking that only
;   the correct PE has A1 set.  The first iteration checks for correct
;   functioning when no A1 is set.

(do ((i 0 (1+ i)))
    ((> i number-of-PEs) (princ '|done, now shift out fault vector.|))
    (nv-loadb1 io1)      ; get old vector 000011
    (nv-recv1 rn)        ; shift to get   000111
    (nv-storea1 io1)     ; store new vector
    (nv-xor)             ; form the goal  000100
    (nv-storea1 b1)      ; store the goal in b1
    (nv-storea1 y1)      ; and in y1

    (nv-loada1 io1)      ; re-load new vector
    (nv-resolve)         ; test resolve, CP checks (R1)
    (nv-xor)             ; reveal any erroneous results
    (nv-loadb1 x1)       ; load the fault vector
    (nv-or)              ; or in any newly revealed errors
    (nv-storea1 x1)      ; save the fault vector back

    (nv-loadb1 y1)       ; re-load the goal vector
    (nv-loada1 y1)       ; load it as test vector too
    (nv-resolve)         ; test resolve, CP checks (R1)
    (nv-xor)             ; reveal any erroneous results
    (nv-loadb1 x1)       ; load the fault vector
    (nv-or)              ; or in any newly revealed errors
    (nv-storea1 x1))     ; save the fault vector back


; Move the error vector into A1 and shift it out for the CP to inspect

(nv-loada1 x1)

(do ((i 1 (1+ i)))
    ((> i number-of-PEs-divided-by-2))
    (nv-send1 rn)
    (nv-recv1 ln))
```

Comments and limitations: This exercise is expensive, but after it has been

completed, the remainder of the PPS functions may be checked quite rapidly, using the paradigm: "everybody do this test; here's the answer you should have gotten; if you got the wrong answer raise your hand; RESOLVE". In this way, the CP can be notified of test failures in constant time. Failures may be located by shifting out the A1 registers.

4.5 Exercise 5

Goal: This exercise checks EN1, BROADCAST1, and the 1-bit ALU.

Instructions tested: ENABLE, {LOAD,STORE} {A1,B1} EN1, BROADCAST1, all LOGICAL, ADD1, SUB1

Time: Constant time using the resolve paradigm, taking about 1 ms.

Method:

EN1 and ENABLE are checked by the following sequence of operations:

```
        ; does ENABLE disturb anything?
SET, ENABLE, CLEAR, RESOLVE     ; no PE should respond

        ; try again after storing a 0 into EN1 from A1 or B1
STOREA1 EN1
SET, ENABLE, CLEAR, RESOLVE     ; no PE should respond
STOREA1 B1, STOREA1 IO1
SET, ENABLE, CLEAR, RESOLVE     ; no PE should respond

        ; try again after storing a 1 into EN1 from A1 or B1
SET, STOREA1 EN1
SET, ENABLE, CLEAR, RESOLVE     ; no PE should respond
SET, STOREA1 B1, STOREA1 IO1
SET, ENABLE, CLEAR, RESOLVE     ; no PE should respond
```

We check BROADCAST1 in four ways. We broadcast 0 and 1 into A1 when its previous state is set and clear. After each of these four operations, A1 is tested to see if it contains the correct value. This may be done by storing A1 into B1, then using SET or CLEAR to put the correct answer into A1. After XOR and RESOLVE, (R1) will be set if there was a fault, and clear if not.

The 1-bit ALU is tested exhaustively. For the logical functions there are 64 trials, since there are 4 input pairs to supply to each of the 16 logical functions. After each trial, the result is copied to B1 and checked as was done in the test of BROADCAST1. Checking ADD1 and SUB1 is similar, except that there are 8 input sets to supply (A1,B1,C1), and 2 output values (A1,C1) to be checked for each input trial.

Comments and limitations: The check of EN1/ENABLE assumes an all-or-none type of failure. That is, we suppose a PE either ignores EN1 for the purpose of instruction execution (a fault), or obeys it. No check is performed to see if some proper subset of the instructions are executed even when EN1 is clear. Also, we do not check the operation of inter-PE communication while some PE's are disabled.

4.6 Exercise 6

Goal: This exercise checks rotations, 8-bit broadcast, the 8-bit registers, and COMPARE.

Instructions tested: ROT{L,R}{A,B}, BROADCAST8, COMPARE, {LOAD,STORE}{A8,B8}{A8,B8,C8,X8,Y8,Z8,IO8,MAR}

Time: Constant time using the resolve paradigm, taking about 1 second.

Method:

The rotate instructions link the 1-bit and 8-bit data paths. If a bit is placed into A1, and then ROTRA is performed 9 times, the bit will move from A1 through all positions of A8, and back to A1. The left rotation, and the rotations on B1/B8 work in a similar manner. The rotations may be tested by iterating the sequence {check the bit in A1, broadcast a new bit to A1, rotate once}. A random pattern of bits may thus be shifted through A1/A8 and checked in a manner analogous to Exercise 1, but on a smaller scale. The B registers may be exercised in a similar way.

BROADCAST8 may be checked by following it with a rotation of the result into A1, checking it bitwise. As test values, the bytes 0, 255, and 0 again should be enough.

To check STOREA8 B8, we may use 0 and 255 as test values (call them "opposites"). The concern here is that an explict store from A8 to B8 should set the value of B8, while setting the value of A8 should not affect the contents of B8. The following operations will check for this. Using 0 first and then 255 as test values, broadcast the opposite of the test value to A8, store it to B8, broadcast the test value to A8, store it to B8, broadcast the opposite to A8, then use rotation on B to check bitwise that B8 contains the correct value. A similar test should be performed subsequently using 01010101 and 10101010 to check for adjacent-bit interference.

The correctness of COMPARE could be determined quickly by sophisticated testing using information about the actual circuit. We choose the simple but exhaustive test as an expedient. By broadcasting to A8 and copying

to B8 we may compare all 64K combinations, checking the results produced in A1 and B1. This exhaustive test takes about 1 second.

The final part of Exercise 6 will check the 8-bit registers. It consists of performing load and store from both A8 and B8 to each of the 8-bit registers. Both values 01010101 and 10101010 should be used. In order to check for interference among these registers, after each store, all 8 registers should be checked to see if the correct value is still there. This is similar to the checking done on the 1-bit flags in Exercise 3.

Comments and limitations: It is likely that more efficient testing of the ACU can be performed, since in the hardware it consists of two ganged 4-bit comparison units. The time penalty of about 1 second is not so severe as to preclude exhaustive testing, however.

4.7 Exercise 7

Goal: Testing of RAM and the 4 RAM I/O instructions.

Instructions tested: READRAM, READRAMR, WRITERAM, WRITERAMR

Time: Constant time, using the resolve paradigm, takes about 1 second.

Method: First, we see whether READRAM and WRITERAM set the MAR correctly, ignoring the actual data transferred with to or from the RAM. Next, we do a standard stuck-test on RAM, using READRAM and WRITERAM. The third check is to fill the RAM cells with distinct random numbers, and check that READRAMR and WRITERAMR work. Finally, we look for interference effects in RAM by spending about 1 second reading and writing random values from/to random addresses, checking for the correct value after each read. Further information about testing RAM may be found in [Knaizuk and Hartmann, 1977; Suk and Reddy, 1981; Hayes, 1975; and El. Test, 1981].

Comments and limitations: Because of our high processor to RAM ratio, we can test much more thoroughly than possible in a sequential environment.

4.8 Exercise 8

Goal: Exercising of 1- and 8-bit tree neighbor communication, and 8-bit linear neighbor communication.

Instructions tested: SEND1 {LC,RC}, RECV1 {LC,RC,P}, SEND8 {LC,RC,LN,RN},

RECV8 {LC,RC,P,LN,RN}

Time: Tree-neighbor testing is O(log n), but linear-neighbor testing is linear in the number of PE's, and takes about 8 seconds using the method of the first exercise.

Method: The 8-bit linear neighbor communication may be tested just as the 1-bit was in exercise 1. We ensure, however, that at the end of this portion of the exercise we have shifted data through the A8 registers in such a way that level number in the PPS of each PE is in its A8. The intention is to save this in X8, and match against it during the testing of tree neighbor communication. This testing includes two directions, corresponding with PE's receiving from parent or child. Testing in the upward direction may be performed by placing 01010101 in A8 in the leaves of the PPS, and 10101010 in the other PE's. The leaf value is subsequently propagated up the tree level by level, with appropriate checking after each upward move. This checking involves a comparison in all those PE's which should have received the leaf value, and another comparison in those which should not have. We may activate each of these two groups in turn by comparing the level number stored in X8 with a value broadcast by the CP, and storing A1 or B1 into EN1. For a second test of tree neighbor communication, we may have all PE's send 10101010 to their left children, send 01010101 to their right children, read them back, and check. Downward tree neighbor communication may be tested in a similar manner, except that there are two modes of communication to be checked: receiving from parent and sending to children.

Comments and limitations: Full consideration has not been given to testing for interference effects between siblings and parents/children.

4.9 Exercise 9

Goal: Exercising REPORT

Instructions tested: REPORT1, REPORT8

Time: Linear in the number of PE's, taking about 40 seconds.

Method: Checking the functioning of REPORT will require enabling the PE's one at a time. To do this, first we clear A1 and A8. Then we perform an operation called shifting a token through the PPS. That is, we enable just the first PE in the linear neighbor sequence, and then step through the linear sequence, disabling the currently enabled PE and enabling its right neighbor (we pass the token right). Thus each PE will be enabled, one at a time, in sequence. At each step, we test the PE which has the token. First, we SET and REPORT1 to test the 1-bit reporting. Then we rotate A1/A8 9 times, performing REPORT8 each time, with the CP

checking that the correct value was received. Finally, we clear A1 and report it out.

Comments and limitations: It may be possible to use a less extensive test to see that REPORT8 is working. Testing with 0 and 255 only, using BROADCAST8 to set the values, could save ten or fifteen seconds.

5 Coverage of Instructions

The following chart tells, for each instruction, which exercises check and use it. The "use" indication is not necessarily accurate for exercises which have not been fully coded.

Key:
- `-` not yet tested
- `t` tested here
- `u` used here
- `o` ok, previously tested

(Note that o may appear where u should be for exercise not yet fully coded.)

	Exercise								
	1	2	3	4	5	6	7	8	9
LOADA8									
A8	-	-	-	-	-	t	o	o	o
B8	-	-	-	-	-	t	o	o	o
C8	-	-	-	-	-	t	o	o	o
X8	-	-	-	-	-	t	o	o	o
Y8	-	-	-	-	-	t	o	o	o
Z8	-	-	-	-	-	t	o	o	o
IO8	-	-	-	-	-	t	o	o	o
MAR	-	-	-	-	-	t	o	o	o
LOADB8									
A8	-	-	-	-	-	t	o	o	o
B8	-	-	-	-	-	t	o	o	o
C8	-	-	-	-	-	t	o	o	o
X8	-	-	-	-	-	t	o	o	o
Y8	-	-	-	-	-	t	o	o	o
Z8	-	-	-	-	-	t	o	o	o
IO8	-	-	-	-	-	t	o	o	o
MAR	-	-	-	-	-	t	o	o	o
LOADA1									
A1	-	-	t	o	o	o	o	o	o
B1	-	t	t	o	o	o	o	o	o
C1	-	-	t	o	o	o	o	o	o

```
  X1............ - - t u o o o o o
  Y1............ - - t u o o o o o
  Z1............ - - t o o o o o o
  IO1........... t u t u o o o o o
  EN1........... - - - - t o o o o

LOADB1
  A1............ - - t o o o o o o
  B1............ - - t o o o o o o
  C1............ - - t o o o o o o
  X1............ - - t u o o o o o
  Y1............ - - t u o o o o o
  Z1............ - - t o o o o o o
  IO1........... - - t u o o o o o
  EN1........... - - - - t o o o o

STOREA8
  A8............ - - - - - t o o o
  B8............ - - - - - t o o o
  C8............ - - - - - t o o o
  X8............ - - - - - t o o o
  Y8............ - - - - - t o o o
  Z8............ - - - - - t o o o
  IO8........... - - - - - t o o o
  MAR........... - - - - - t o o o

STOREB8
  A8............ - - - - - t o o o
  B8............ - - - - - t o o o
  C8............ - - - - - t o o o
  X8............ - - - - - t o o o
  Y8............ - - - - - t o o o
  Z8............ - - - - - t o o o
  IO8........... - - - - - t o o o
  MAR........... - - - - - t o o o

STOREA1
  A1............ - - t o o o o o o
  B1............ - t t u o o o o o
  C1............ - - t o o o o o o
  X1............ - - t u o o o o o
  Y1............ - - t u o o o o o
  Z1............ - - t o o o o o o
  IO1........... t u t u o o o o o
  EN1........... - - - - t o o o o

STOREB1
  A1............ - - t o o o o o o
  B1............ - - t o o o o o o
  C1............ - - t o o o o o o
  X1............ - - t o o o o o o
  Y1............ - - t o o o o o o
```

```
  Z1........... - - t o o o o o o
  IO1.......... - - t o o o o o o
  EN1.......... - - - - t o o o o

READRAM........ - - - - - - t o o

WRITERAM....... - - - - - - t o o

READRAMR....... - - - - - - t o o

WRITERAMR...... - - - - - - t o o

ADD1........... - - - - t o o o o

SUB1........... - - - - t o o o o

ROTRA.......... - - - - - t o o o

ROTLA.......... - - - - - t o o o

ROTRB.......... - - - - - t o o o

ROTLB.......... - - - - - t o o o

LOGICAL
  CLEAR........ - t u u t o o o o
  SET.......... - t u o t o o o o
  NEGATE....... - t u o t o o o o
  AND.......... - - - - t o o o o
  OR........... - t u u t o o o o
  XOR.......... - t u u t o o o o
  EQU.......... - - - - t o o o o
  NAND......... - - - - t o o o o
  NOR.......... - - - - t o o o o
  NOP.......... - - - - t o o o o
    ........... - - - - t o o o o
    ........... - - - - t o o o o
    ........... - - - - t o o o o
    ........... - - - - t o o o o
    ........... - - - - t o o o o
    ........... - - - - t o o o o

BROADCAST8..... - - - - - t o o o

BROADCAST1..... - - - - t o o o o

REPORT8........ - - - - - - - - t

REPORT1........ - - - - - - - - t

SEND8
  LN........... - - - - - - - t o
```

```
  RN............ - - - - - - - t o
  LC............ - - - - - - - t o
  RC............ - - - - - - - t o

SEND1
  LN............ t o o o o o o o o
  RN............ t u u u o o o o o
  LC............ - - - - - - - t o
  RC............ - - - - - - - t o

RECV8
  LN............ - - - - - - - t o
  RN............ - - - - - - - t o
  LC............ - - - - - - - t o
  RC............ - - - - - - - t o
  P............. - - - - - - - t o

RECV1
  LN............ t u u u o o o o o
  RN............ t o o u o o o o o
  LC............ - - - - - - - t o
  RC............ - - - - - - - t o
  P............. - - - - - - - t o

ENABLE.......... - - - - t o o o o

COMPARE......... - - - - - t o o o

RESOLVE......... - - - t u u o o o
```

REFERENCES

Agrawal, Vishwani D., "An Information Theoretic Approach to Digital Fault Testing", IEEE Transactions on Computers, vol. C-30, no. 8, August 1981.

Electronics Test, "Standard Patterns for Testing Memories", vol. 4, no.4, April 1981, pp. 22-26.

Galiay, J., Y. Crouzet, and M. Vergniault, "Physical versus Logical Fault Models in MOS LSI Circuits, Impact on Their Testability", Proceedings of the Ninth International Symposium on Fault-Tolerant Computing, Madison, Wisconson June 1979.

Hayes, John P., "Detection of Pattern-Sensitive Faults in Random-Access Memories", IEEE Transactions on Computers, vol. C-24, no. 2, February 1975.

Knaizuk, John Jr., and Carlos R. P. Hartmann, "an Optimal Algorithm for Testing Stuck-at Faults in Random Access Memories", IEEE Transactions on Comuters, vol. C-26, no. 11, November 1977.

Leiserson, Charles, Area-Efficient VLSI Computation, Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, October 1981.

Shaw, David Elliot, "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives", Stanford Computer Science Department Report STAN-CS-79-778, October 1979.

Shaw, David Elliot, "A Relational Database Machine Architecture", Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing, Asilomar, California, March 1980. (Also reprinted in joint issue of ACM SIGARCH, SIGIR and SIGMOD publications.)

Shaw, David Elliot, "The NON-VON Supercomputer", Columbia Computer Science Department Report, July 1982. (Submitted to IEEE Transactions on Computers.)

Shaw, David Elliot, Salvatore J. Stolfo, Hussein Ibrahim, Bruce Hillyer, Gio Wiederhold and J. A. Andrews, "The NON-VON Database Machine: A Brief Overview", Database Engineering, vol. 4, no. 2., December 1981.

Sridhar, Thirumalai, and John P. Hayes, "A Functional Approach to Testing Bit- Sliced Microprocessors", IEEE Transactions on Computers, vol. C-30, no. 8, August 1981.

Srini, Vason P., "Fault Diagnosis of Microprocessor Systems", Computer, January 1977.

Suk, D. S., and S. M. Reddy, "A March Test for Functional Faults in

Semiconductor Random Access Memories", IEEE Transactions on Computers, vol. C-30, no. 12, December 1981.