SIMULATION EFFECTIVENESS

RESEARCH REPORT*

ALICE C. PARKER

DIGITAL INTEGRATED SYSTEMS CENTER REPORT

DISC/83-2

DEPARTMENT OF ELECTRICAL ENGINEERING-SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90089-0781

APRIL 1983

---

*Funding for this research was provided by IBM Corporation Grant S 956501 LX A B22.

as that which is determinable given a simulation of the design and a given test set; i.e. it is correctness relative to a test set. As simulation effectiveness increases, simulation correctness approaches correctness. Figures 4.1 and 4.2 describe some of the correlations that exist between these quantities.

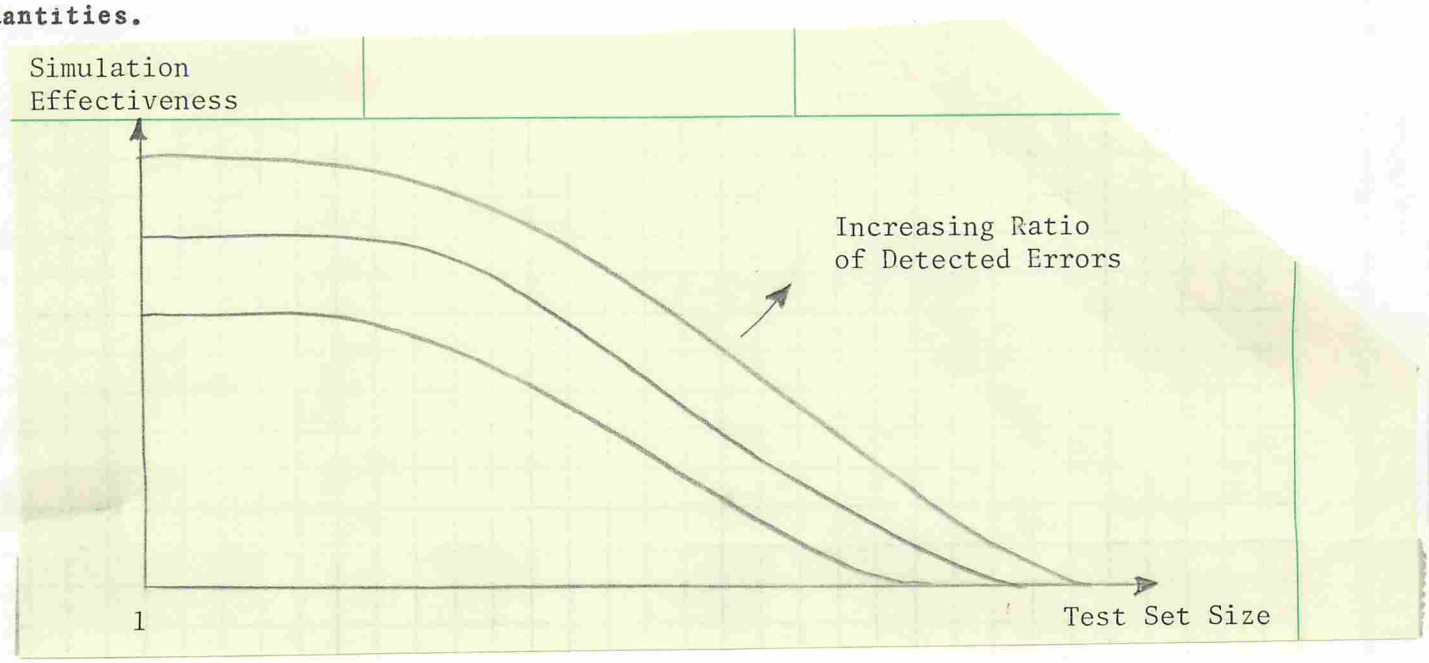


Figure 4-1: The relationship between Test-Set Size and Simulation Effectiveness

The Survey Results

In order to classify errors into types, and to determine which error types escape early detection by simulation, we conducted a survey of designers, both within and outside of IBM. A sample survey form is attached as Appendix A. Within IBM, a number of designers were surveyed, and sixty-two errors were described. Outside of IBM, designers in one company were interviewed informally, one company provided a comprehensive report on design errors, key individuals in other companies were sent survey forms for general information, and individual designers were surveyed about specific projects.

Although the survey results are incomplete, and many of the surveys have missing or unclear answers, some general trends have emerged.

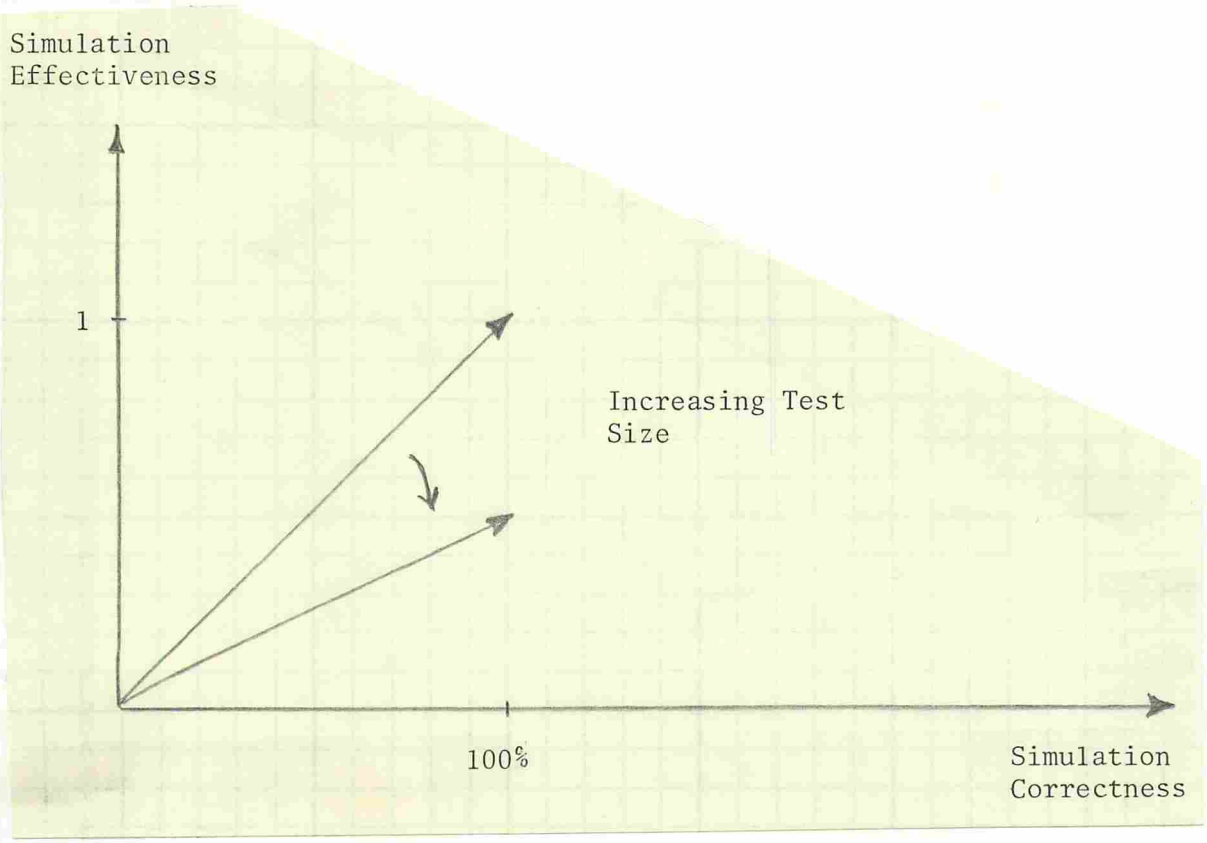


Figure 4-2: The Relationship Between Simulation Effectiveness and Simulation Correctness

Figure 4-2: The Relationship Between Simulation Effectiveness and Simulation Correctness

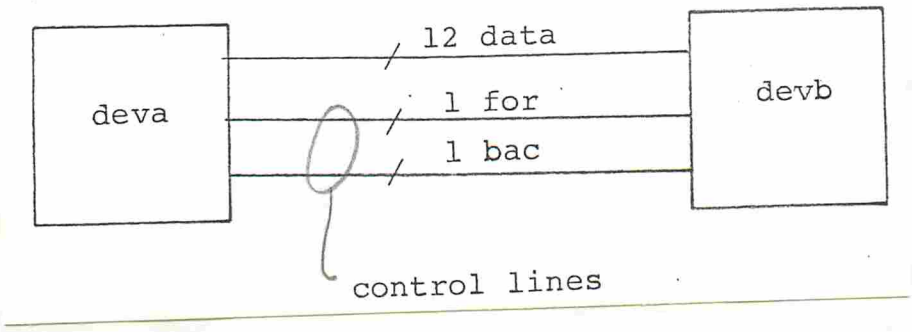


Figure 5-1: The Hardware Example Used for Simulations

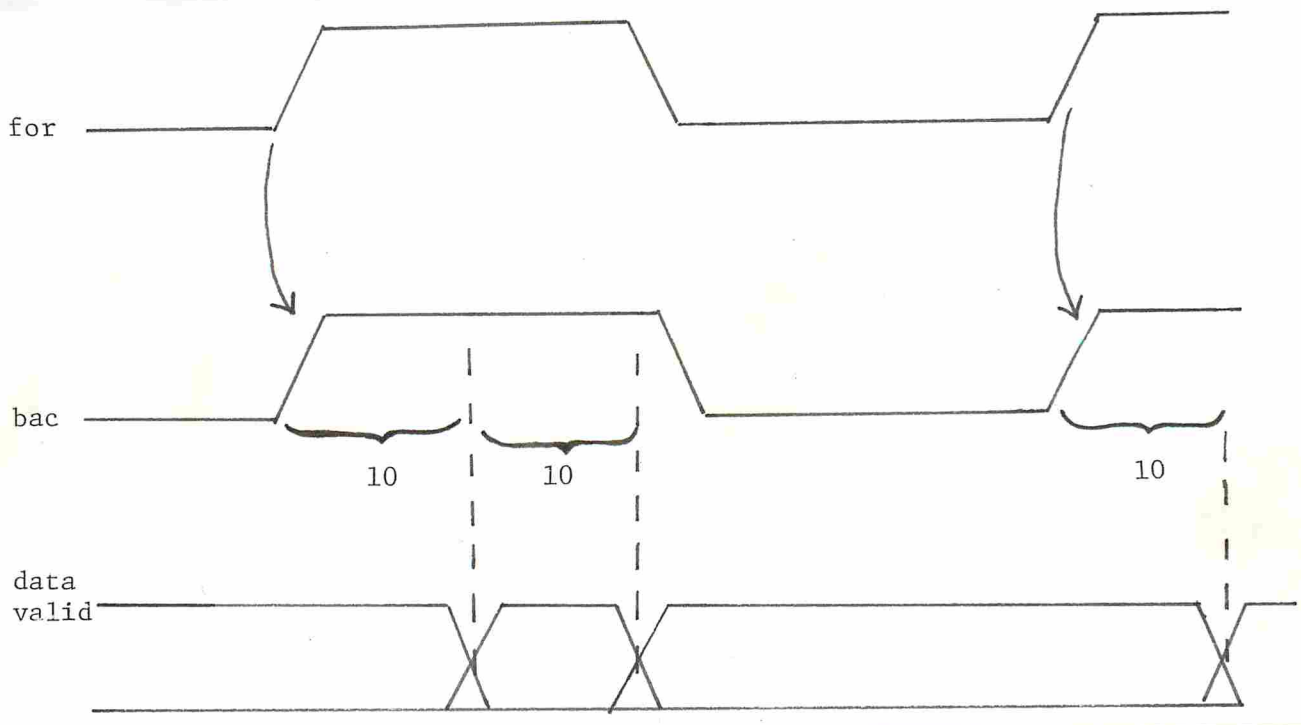


Figure 5-2: Timing of the Protocol for the Example Hardware

Figure 5-1: The Hardware Example Used for Simulations

Figure 5-2: Timing of the Protocol for the Example Hardware

and "restart" conditions respectively. L[A,f](P) is a predicate which is the disjunct of all the preconditions of all the "leave f" actions which may be executed, when A is executed with a precondition of P. In other words L[A,f](P) is a predicate which is satisfied by any state which may be in effect when a "leave f" is executed within A, after A has been started in a state satisfying the precondition P. Similarly R[A,f](P) is a predicate characterizing all states which may hold when a "restart f" is executed within A, with A having been started in some state satisfying P. All the possible actions in ISPB and the definition of T and B (Behavior Expression) for them are listed below.

The Pre-to-Postcondition Transformer T

A	T[A](P)
$u \leftarrow e$	$\exists u'\ P\langle u \leftarrow u' \rangle \wedge u = e\langle u \leftarrow u' \rangle$
$x_i \leftarrow e$	P
$u \leftarrow x_i$	$\exists u', j'_i\ P\langle u \leftarrow u', j_i \leftarrow j'_i \rangle \wedge j_i = j'_i + 1 \wedge u = x_i(j_i)$
leave f_k	false
restart f_k	false
skip	P
if b then A_1 else A_2	$T[A_1](P \wedge b) \vee T[A_2](P \wedge \sim b)$
$A_1;A_2$	$\exists \underline{w}'_1, \underline{w}'_2 (T[A_1]((P \wedge \underline{w}_1 = \underline{w}'_1)\langle \underline{w}_2 \leftarrow \underline{w}'_2 \rangle) \wedge T[A_2]((P \wedge \underline{w}_2 = \underline{w}'_2)\langle \underline{w}_1 \leftarrow \underline{w}'_1 \rangle))$
A_1 next A_2	$T[A_2](T[A_1](P))$
call f_k	$\exists I \{L[A_k, f_k](P_k(P,I)) \vee T[A_k](P_k(P,I))\}$

where $f_k := A_k$

Here P_k is defined to be the unique least fixpoint of the equation

$P_k(P,I) = ((I = 1) \wedge P) \vee ((I > 1) \wedge R[A_k, f_k](P_k(P, I-1)))$

DEFINITION OF B

A	B[A](P)
$u \leftarrow e$	$\Lambda:/T[A](P)/$
$x_i \leftarrow e$	$W(x_i):/(P \wedge x_i = e)/$
$u \leftarrow x_i$	$R(x_i):/(T[A](P))/$
leave f_k	$\Lambda:/P/$
restart f_k	$\Lambda:/P/$
skip	$\Lambda:/P/$
if b then A_1 else A_2	$B[A_1](P \wedge b) + B[A_2](P \wedge \sim b)$
$A_1;A_2$	$B[A_1](P) \parallel B[A_2](P)$
A_1 next A_2	$B[A_1](P) \cdot B[A_2](T[A_1](P))$
call f_k	$[B[A_k](P_k(P,I_k))]^* I_k$
where $f_k := A_k$	

Table 6-2: Definitions of T and B, taken from [4]

P_k is defined to be the unique least fixpoint of the equation

$$P_k(P,I) = ((I=1 \wedge P) \vee ((I>1 \wedge R[A_k, f_k](P_k(P, I-1)))$$

```
! The following two SLIDE descriptions describe two devices communicating
  with each other over a  data bus.  The communication is controlled by
  a pair of control lines. !


MAIN PROCESS devb;                          ! First device

CLOCK 1;

LINE
   data<11:0>,                              ! data bus
   for<>,                                   ! control line
   bac<>;                                   ! control line

INIT B:0 WHEN bac EQL /;                    ! PROCESS B starts executing on
                                            ! the rising edge of bac.

PROCESS B;

REGISTER
     b<11:0>,                               ! 12 bit register used by process B.
     decs<1:0>;                             ! 2 bit register.

COMB
   zero<1:0> := '00,                        ! binary value 00 is assigned to zero.
   one<1:0>  := '01,
   two<1:0>  := '10;

BEGIN
  delay 20 NEXT                             ! delay by 20 clock cycles.
  b _ data NEXT                             ! contents of 'data' are transferred
                                            ! to 'b'
  decs<1:0> _ b<1:0>;                       ! two LSB's of 'b' are assigned  to
                                            ! 'decs'
  IF decs EQL zero THEN                     ! if 'decs' is zero then write 'b'
                                            ! onto 'data'
   BEGIN
    data_b NEXT
    for _ ≠ NEXT                            ! 'for' makes a transition from '1'
                                            ! to '0'
    bac _ ≠
   END
  ELSE IF decs EQL one THEN                 ! if 'decs' is one and
   BEGIN
    IF PARE(b<11:0>) THEN                   ! if parity of 'b' is even
     BEGIN
      b<0> _ NOT b<0> NEXT                  ! LSB of 'b' is complemented
      data _ b NEXT
      for _ ≠ NEXT
      bac _ ≠
     END
```

```
    ELSE                                    ! if 'decs' is one and
     BEGIN                                  ! if parity of 'b' is odd
       b<1> _ NOT b<1> NEXT                 ! LSB of 'b' is complemented
       data _ b NEXT
       for _ ≠ NEXT
       bac _ ≠
     END
    END
   ELSE IF decs EQL two THEN                    ! if 'decs' is two
    BEGIN
     data<11:6> _ b<5:0>;                       ! write 'b' onto 'data'
     data<5:0> _ b<11:6> NEXT                   ! swapping bytes
     for _ ≠ NEXT
     bac _ ≠
    END
   ELSE                                         ! if 'decs' is three
    BEGIN
     data _ 0 NEXT                              ! all data lines are made '0'
     for _ ≠ NEXT
     bac _ ≠
    END
  END;

  BEGIN
   for _ / ;                                    ! 'for' transitions from '0' to '1'
   DELAY WHILE 1                                ! delay forever
  END
```

```
MAIN PROCESS deva;                              ! Second device

CLOCK 1;

LINE
  data<11:0>,                                   ! Same external lines
  for<>,
  bac<>;

EXT REGISTER
  flag<>;

INIT Dummy:0 WHEN 1;                            ! Dummy process is always executing
                                                ! since its initiation condition is
                                                ! always true.
```

```
PROCESS Dummy;

        REGISTER
          soa<11:0>,
          a<11:0>;

        INIT Assign:0 WHEN flag EQL /;   ! Starting conditions for 'Assign',
        INIT Adata:0 WHEN for EQL /;     ! 'Adata', and 'Aread'
        INIT Aread:0 WHEN bac EQL ≠;

PROCESS Adata;
BEGIN
  bac _ /;
  delay 10 NEXT                          ! delay by 10 clock cycles
  data _ soa                             ! write onto data lines
END;

PROCESS Aread;

BEGIN
  a _  data;                             ! read from data lines
  DELAY 75 NEXT                          ! and delay 75 clock cycles
  for _ /
END;

PROCESS Assign;

BEGIN
  soa _ #7775 NEXT                       ! 'soa' is assigned octal value 7775
  flag _ ≠                               ! lower flag
END;

BEGIN !dummy process!
  DELAY WHILE 1                          ! idle forever!
END;

BEGIN !main process!
  flag _ /;                              ! raise flag to start 'Assign'
  DELAY WHILE 1                          ! idle forever!
END
```

```
! This description of Devb contains an error. This is an example of a
  timing error. The value of "data" is being read too early. The contents
  of "soa" are supposed to be written into "data", after which the value
  of "data" is read and written into register "b". In the description
  shown below the value of "data" is being read and transfered to "b"
  even before "data" is written to from "soa". This error is not caught
  by the simulator but the output of the simulation run is different from
  the expected output !

MAIN PROCESS devb;

CLOCK 1;

LINE
   data<11:0>,
   for<>,
   bac<>;

INIT B:0 WHEN bac EQL /;

PROCESS B;

REGISTER
    b<11:0>,
    decs<1:0>;

COMB
   zero<1:0> := '00,
   one<1:0>  := '01,
   two<1:0>  := '10;

BEGIN
  delay 5 NEXT                    ! delay has been changed from 20 to 5
                                  ! thus causing data to be read early.
  b _ data NEXT
  decs<1:0> _ b<1:0>;
  IF decs EQL zero THEN
   BEGIN
    data_b NEXT
    for _ ≠ NEXT
    bac _ ≠
   END
  ELSE IF decs EQL one THEN
   BEGIN
    IF PARE(b<11:0>) THEN
     BEGIN
      b<0> _ NOT b<0> NEXT
      data _ b NEXT
      for _ ≠ NEXT
      bac _ ≠
     END
```

```
    ELSE
     BEGIN
       b<1> _ NOT b<1> NEXT
       data _ b NEXT
       for _ ≠ NEXT
       bac _ ≠
     END
   END
  ELSE IF decs EQL two THEN
   BEGIN
    data<11:6> _ b<5:0>;
    data<5:0> _ b<11:6> NEXT
    for _ ≠ NEXT
    bac _ ≠
   END
  ELSE
   BEGIN
    data _ 0 NEXT
    for _ ≠ NEXT
    bac _ ≠
   END
END;

BEGIN
 for _ / ;
 DELAY WHILE 1
END
```

```
! The description shown below contains an example of a logical error,
  a missing conditional branch.  One of the statements ( b<0> _ NOT b<0> )
  in a particular conditional branch( if decs EQL one ) is missing.
  This error is also not detected by the simulator but the output of the
  simulation run is different from the expected output. !

MAIN PROCESS devb;

CLOCK 1;

LINE
   data<11:0>,
   for<>,
   bac<>;

INIT B:0 WHEN bac EQL /;

PROCESS B;

REGISTER
     b<11:0>,
     decs<1:0>;

COMB
   zero<1:0> := '00,
   one<1:0>  := '01,
   two<1:0>  := '10;

BEGIN
  delay 20 NEXT
  b _ data NEXT
  decs<1:0> _ b<1:0>;
  IF decs EQL zero THEN
   BEGIN
    data_b NEXT
    for _ ≠ NEXT
    bac _ ≠
   END
  ELSE IF decs EQL one THEN
   BEGIN
    IF PARE(b<11:0>) THEN
     BEGIN
                                          ! b<0> _ NOT b<0> is missing
      data _ b NEXT
      for _ ≠ NEXT
      bac _ ≠
     END
```

```
    ELSE
     BEGIN
       b<1> _ NOT b<1> NEXT
       data _ b NEXT
       for _ ≠ NEXT
       bac _ ≠
     END
    END
   ELSE IF decs EQL two THEN
    BEGIN
     data<11:6> _ b<5:0>;
     data<5:0> _ b<11:6> NEXT
     for _ ≠ NEXT
     bac _ ≠
    END
   ELSE
    BEGIN
     data _ 0 NEXT
     for _ ≠ NEXT
     bac _ ≠
    END
  END;

  BEGIN
   for _ / ;
   DELAY WHILE 1
  END
```

```
! The description below contains an example of a concurrent error, a
  resource conflict. "b" is being written into and read at the same time.
  This is a 'read/write' type of resource conflict. This error is not detected
  by the simulator or by the simulation run. There is another error, however,
  a 'write/write' type of resource conflict which is caught by the simulator. !

MAIN PROCESS devb;

CLOCK 1;

LINE
   data<11:0>,
   for<>,
   bac<>;

INIT B:0 WHEN bac EQL /;

PROCESS B;

REGISTER
     b<11:0>,
     decs<1:0>;
```

```
COMB
   zero<1:0> := '00,
   one<1:0>  := '01,
   two<1:0>  := '10;

BEGIN
   delay 20 NEXT
   b _ data ;                          ! a NEXT statement after 'b _ data'
   decs<1:0> _ b<1:0>;                 ! is missing causing the next statement
   IF decs EQL zero THEN               ! to be executed in parallel with it.
    BEGIN                              ! Thus 'b' is being written to and read
     data_b NEXT                       ! in parallel.
     for _ ≠ NEXT
     bac _ ≠
    END
   ELSE IF decs EQL one THEN
    BEGIN
     IF PARE(b<11:0>) THEN
      BEGIN
       b<0> _ NOT b<0> NEXT
       data _ b NEXT
       for _ ≠ NEXT
       bac _ ≠
      END
     ELSE
      BEGIN
        b<1> _ NOT b<1> NEXT
        data _ b NEXT
        for _ ≠ NEXT
        bac _ ≠
      END
    END
   ELSE IF decs EQL two THEN
    BEGIN
     data<11:6> _ b<5:0>;
     data<5:0> _ b<11:6> NEXT
     for _ ≠ NEXT
     bac _ ≠
    END
   ELSE
    BEGIN
     data _ 0 NEXT
     for _ ≠ NEXT
     bac _ ≠
    END
  END;

  BEGIN
   for _ / ;
   DELAY WHILE 1
  END
```

```
[PHOTO:  Recording initiated  Sat 29-May-82 3:10PM]

[Link from SPEAR, TTY 167]

 TOPS-20 Command processor 4(560)

! THE SIMULATION RUN BELOW IS OF THE EXAMPLE WITHOUT AN INJECTED ERROR

@r corr
SLIDE/Multi-Level Simulator Version 1.0
Welcome and Good Luck!!

#GET TRAP.IL
#ALL
VISHAL: DEVA     DATA FOR BAC FLAG;                !Vishal is an instance of Deva
VITTAL: DEVB     DATA FOR BAC;                     !Vittal is an instance of Devb
#SIMU                    !data is connected to data, for to for, bac to bac
%Simulation time parameters for VISHAL : DEVA may be bound now
%finished
%Simulation time parameters for VITTAL : DEVB may be bound now
%finished
#PR DATA                                           ! Probe data, for, bac, flag
#PR FOR
#PR BAC
#PR FLAG
#GO J1
--->       0.000us FLAG                            ! Information about FLAG
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us     ! SIZE= 0 means '1' bit
VALUES ON WIRE-    1                               ! value of FLAG is '1'
--->       0.000us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->       0.000us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->       0.001us FLAG                            ! refers to FLAG at .001us.
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0                               ! value of FLAG is '0'
--->       0.011us DATA                            ! refers to DATA at .001us
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us     ! size=11 means 12 bits
VALUES ON WIRE-    111111111101                    ! value is 111111111101
--->       0.023us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    111111111100
--->       0.024us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
--->      0.025us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.101us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.101us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.112us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.124us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.125us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.126us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.202us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.202us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.213us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.225us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.226us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.227us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.303us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.303us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.314us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.326us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
```

```
-->        0.327us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.328us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.404us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.404us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.415us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE--    11111111101
-->        0.427us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE--    11111111100
-->        0.428us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.429us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.505us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.505us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.516us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE--    11111111101
-->        0.528us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE--    11111111100
-->        0.529us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.530us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    0
-->        0.606us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.606us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE--    1
-->        0.617us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE--    11111111101
```

```
--->        0.629us DATA
 LOGIC=     6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     11111111100
--->        0.630us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.631us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.707us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->        0.707us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->        0.718us DATA
 LOGIC=     6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     11111111101
--->        0.730us DATA
 LOGIC=     6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     11111111100
--->        0.731us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.732us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.808us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->        0.808us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->        0.819us DATA
 LOGIC=     6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     11111111101
--->        0.831us DATA
 LOGIC=     6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     11111111100
--->        0.832us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.833us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->        0.909us FOR
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->        0.909us BAC
 LOGIC=     6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
```

```
--->      0.920us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.932us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.933us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.934us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      1.000us
#EXIT

5 garbage collection(s) in 183 ms

End of SIMULA program execution.
CPU time: 8.30  Elapsed time: 57.06
@pop

[PHOTO:  Recording terminated  Sat 29-May-82 3:12PM]
```

```
[PHOTO:  Recording initiated  Sat 29-May-82 3:40PM]

[Link from SPEAR, TTY 167]

 TOPS-20 Command processor 4(560)

! THE SIMULATION RUN BELOW IS WITH AN INJECTED TIMING ERROR


@r timing
SLIDE/Multi-Level Simulator Version 1.0
Welcome and Good Luck!!

#GET TRAP J.IL
#BAD COMMAND.
#BAD COMMAND.
#ALL
VISHAL: DEVA     DATA FOR BAC FLAG;
VITTAL: DEVB     DATA FOR BAC;
#SIMU
%Simulation time parameters for VISHAL : DEVA may be bound now
%finished
%Simulation time parameters for VITTAL : DEVB may be bound now
%finished
#PR DATA
#PR  JFOR
#PR BAC
#PR FLAG
#GO  J1
```

```
--->       0.000us FLAG
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.000us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.000us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.001us FLAG
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.007us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     000000000000
--->       0.008us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.009us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.011us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     111111111101
--->       0.085us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.085us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.093us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     111111111100
--->       0.094us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.095us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.096us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     111111111101
--->       0.171us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.171us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     1
--->       0.179us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-     111111111100
--->       0.180us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
--->       0.181us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-     0
```

```
! The sequence of writes to
! the registers and lines is
! different from the expected
! output. The times at which
! the writes occur is also
! incorrect. The first write
! to DATA is 000000000000
! instead of 111111111101.
! All of this shows up in the
! simulation run but is not
! caught by the simulator.
```

```
--->      0.182us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.257us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.257us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.265us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.266us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.267us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.268us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.343us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.343us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.351us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.352us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.353us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.354us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.429us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.429us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.437us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.438us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.439us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
--->      0.440us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.515us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.515us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.523us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.524us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.525us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.526us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.601us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.601us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.609us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.610us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.611us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.612us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.687us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.687us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.695us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.696us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.697us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
--->      0.698us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.773us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.773us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.781us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.782us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.783us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.784us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.859us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.859us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.867us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.868us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.869us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.870us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.945us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.945us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.953us DATA
 LOGIC=    6 SIZE=  11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
--->      0.954us FOR
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.955us BAC
 LOGIC=    6 SIZE=   0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
-->       0.956us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111101
-->       1.000us
#EXIT

6 garbage collection(s) in 235 ms

End of SIMULA program execution.
CPU time: 8.90  Elapsed time: 1:15.03
@pop

[PHOTO:  Recording terminated  Sat 29-May-82 3:42PM]
```

```
[PHOTO:  Recording initiated  Sat 29-May-82 5:22PM]

[Link from SPEAR, TTY 167]

 TOPS-20 Command processor 4(560)


! INJECTED ERROR IS A LOGICAL ERROR


@r logicl
SLIDE/Multi-Level Simulator Version 1.0
Welcome and Good Luck!!

#GET TRAP.IL
#BAD COMMAND.
#BAD COMMAND.
#ALL
VISHAL: DEVA     DATA FOR BAC FLAG;
VITTAL: DEVB     DATA FOR BAC;
#SIMU
%Simulation time parameters for VISHAL : DEVA may be bound now
%finished
%Simulation time parameters for VITTAL : DEVB may be bound now
%finished
#PR DATA
#PR FOR
#PR BAC
#PR FLAG
#GO 1
-->       0.000us FLAG                              ! The sequence of writes is
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us     ! as expected, but the times
VALUES ON WIRE-    1                                ! at which these writes occur
-->       0.000us FOR                               ! is incorrect. Also the second
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us     ! write to data is 111111111101
VALUES ON WIRE-    1                                ! instead of 111111111100. Thus
-->       0.000us BAC                               ! it shows up in the simulation
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us     ! run but is not caught by the
VALUES ON WIRE-    1                                ! simulator.
```

```
--->        0.001us FLAG
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.011us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.022us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.023us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.024us BAC
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.100us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     1
--->        0.100us BAC
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     1
--->        0.111us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.122us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.123us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.124us BAC
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.200us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     1
--->        0.200us BAC
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     1
--->        0.211us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.222us DATA
 LOGIC=     6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-     11111111101
--->        0.223us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.224us BAC
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     0
--->        0.300us FOR
 LOGIC=     6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-     1
```

```
--->      0.300us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.311us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.322us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.323us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.324us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.400us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.400us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.411us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.422us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.423us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.424us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.500us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.500us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.511us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.522us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.523us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.524us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.600us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
```

```
--->      0.600us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.611us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.622us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.623us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.624us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.700us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.700us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.711us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.722us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.723us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.724us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.800us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.800us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.811us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.822us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.823us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.824us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.900us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
```

```
--->      0.900us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
--->      0.911us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.922us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
--->      0.923us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      0.924us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
--->      1.000us
#EXIT

6 garbage collection(s) in 236 ms

End of SIMULA program execution.
CPU time: 8.38  Elapsed time: 58.75
@pop

[PHOTO:  Recording terminated  Sat 29-May-82 5:23PM]
```

```
[PHOTO:  Recording initiated  Sat 29-May-82 5:33PM]

[Link from SPEAR, TTY 167]

 TOPS-20 Command processor 4(560)


! INJECTED ERROR IS A CONCURRENCY ERROR


@r concur
SLIDE/Multi-Level Simulator Version 1.0
Welcome and Good Luck!!

#GET TRAP.IL
#BAD COMMAND.
#BAD COMMAND.
#ALL
VISHAL: DEVA     DATA FOR BAC FLAG;
VITTAL: DEVB     DATA FOR BAC;
#SIMU
%Simulation time parameters for VISHAL : DEVA may be bound now
%finished
%Simulation time parameters for VITTAL : DEVB may be bound now
%finished
```

```
#PR DATA
#PR F JOR
#PR BAC
#PR FLAG
#GO 1
--->        0.000us FLAG                                  ! The sequence of writes is as
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us         ! expected, but the times at
VALUES ON WIRE-    1                                     ! which they occur is incorrect
--->        0.000us FOR                                   ! The 'read/write' resource
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us         ! conflict is not caught by the
VALUES ON WIRE-    1                                     ! simulator. However, the
--->        0.000us BAC                                   ! omission of the NEXT
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us         ! statement causes a 'write/
VALUES ON WIRE-    1                                     ! write' resource conflict
--->        0.001us FLAG                                  ! which is caught by the
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us         ! simulator.
VALUES ON WIRE-    0
--->        0.011us DATA
 LOGIC=    6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant          ! Error caught by the
--->        0.022us DATA                                  ! simulator.
 LOGIC=    6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-    11111111100
--->        0.023us FOR
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    0
--->        0.024us BAC
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    0
--->        0.100us FOR
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    1
--->        0.100us BAC
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    1
--->        0.111us DATA
 LOGIC=    6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
--->        0.122us DATA
 LOGIC=    6 SIZE=   11 PERIOD=          0.000us
VALUES ON WIRE-    11111111100
--->        0.123us FOR
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    0
--->        0.124us BAC
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    0
--->        0.200us FOR
 LOGIC=    6 SIZE=    0 PERIOD=          0.000us
VALUES ON WIRE-    1
```

```
-->        0.200us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.211us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
-->        0.222us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
-->        0.223us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.224us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.300us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.300us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.311us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
-->        0.322us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
-->        0.323us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.324us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.400us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.400us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.411us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
-->        0.422us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
-->        0.423us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.424us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
-->        0.500us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.500us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.511us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111101
%More than one write to B within one time instant
-->        0.522us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111100
-->        0.523us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.524us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.600us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.600us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.611us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111101
%More than one write to B within one time instant
-->        0.622us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111100
-->        0.623us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.624us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.700us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.700us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.711us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111101
%More than one write to B within one time instant
-->        0.722us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    111111111100
-->        0.723us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
```

```
-->        0.724us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.800us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.800us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.811us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
-->        0.822us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
-->        0.823us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.824us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.900us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.900us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    1
-->        0.911us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111101
%More than one write to B within one time instant
-->        0.922us DATA
 LOGIC=    6 SIZE=   11 PERIOD=        0.000us
VALUES ON WIRE-    11111111100
-->        0.923us FOR
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        0.924us BAC
 LOGIC=    6 SIZE=    0 PERIOD=        0.000us
VALUES ON WIRE-    0
-->        1.000us
#EXIT

6 garbage collection(s) in 244 ms

End of SIMULA program execution.
CPU time: 8.34  Elapsed time: 58.08
@pop

[PHOTO:  Recording terminated  Sat 29-May-82 5:34PM]
```

Figure 4-1: Example of Entities-Resources and Timing Mappings

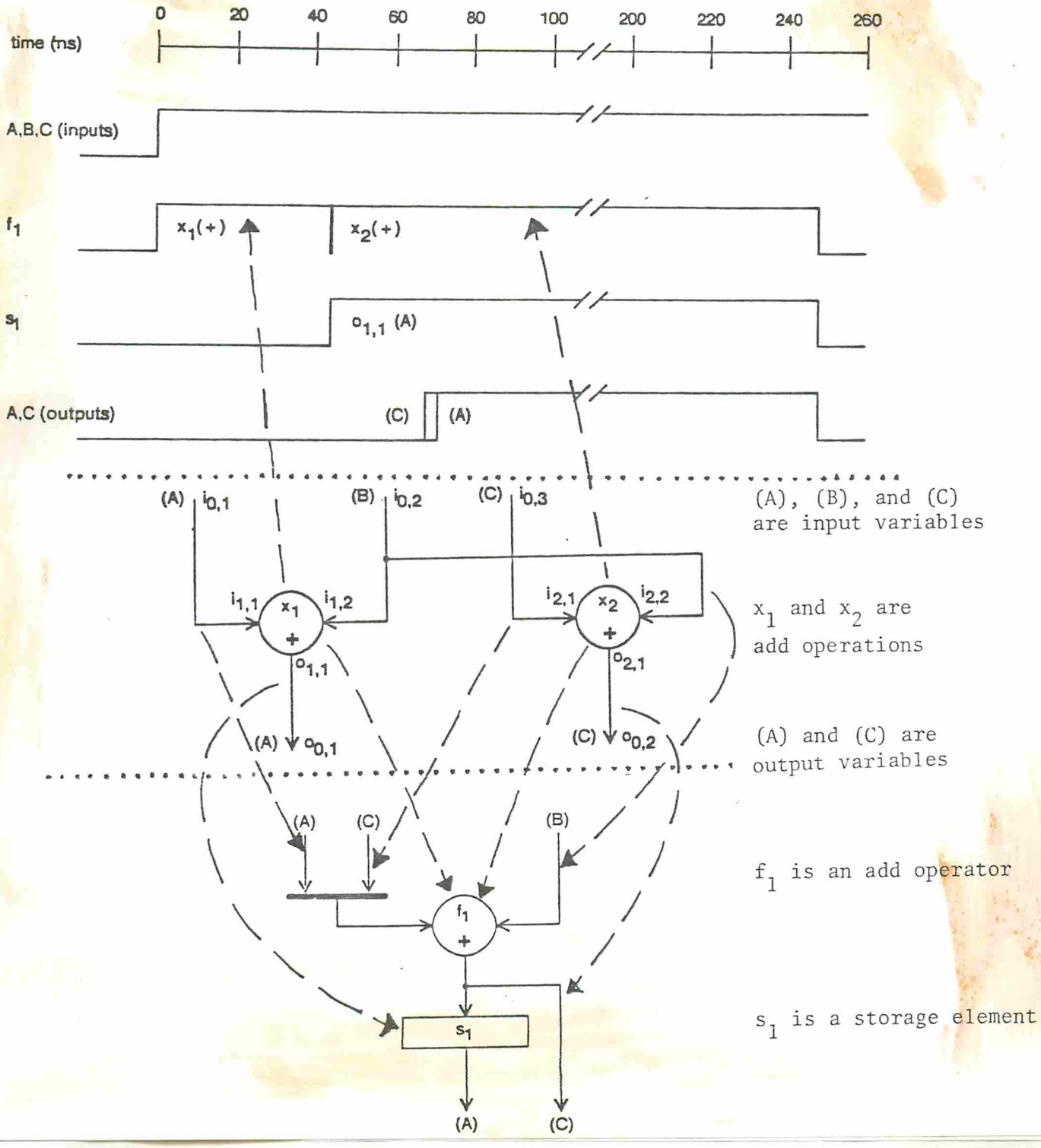


Figure 4-1: Example of Entities-Resources and Timing Mappings

Figure 4-2: Partitioning of Register-Transfer Design

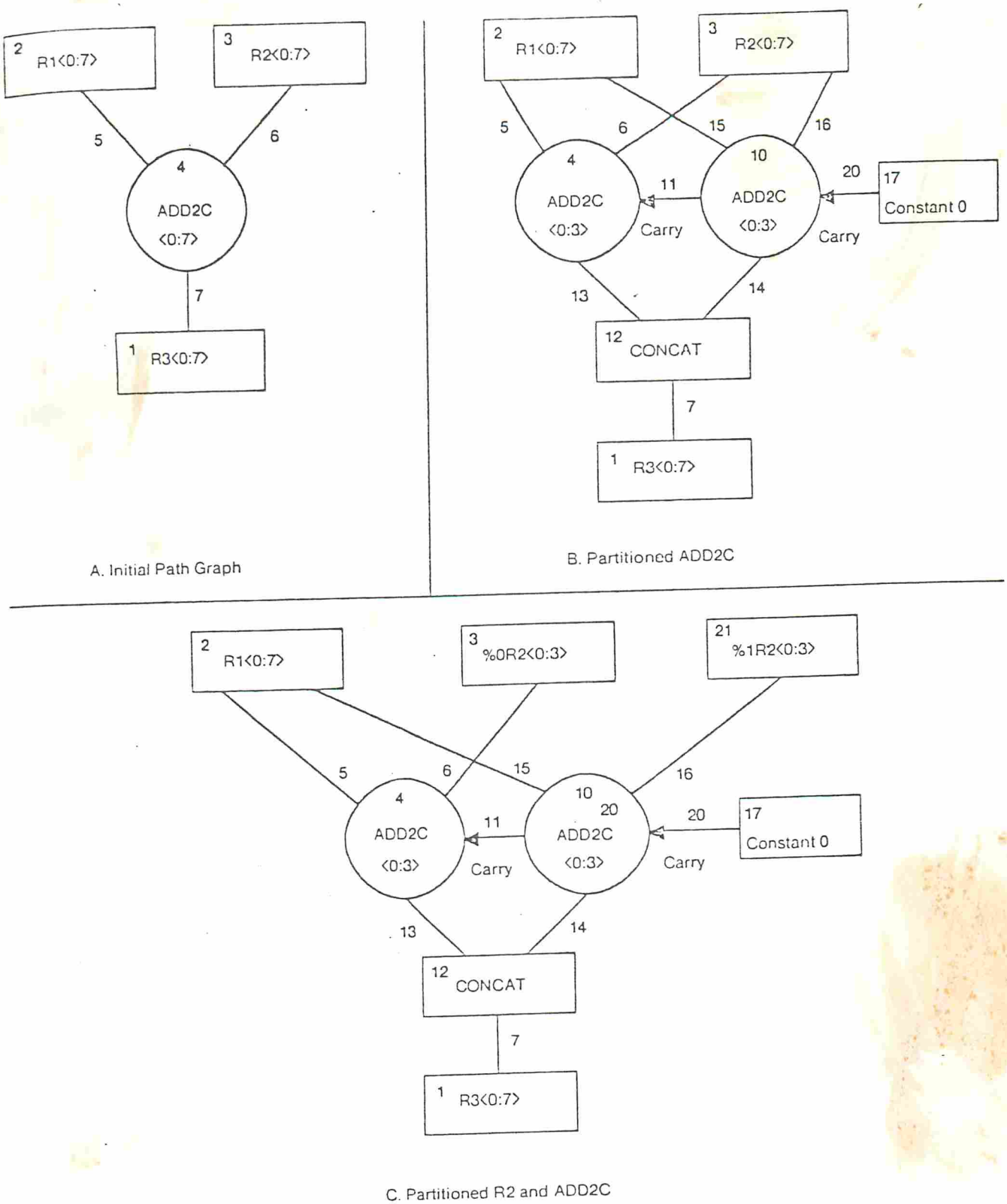
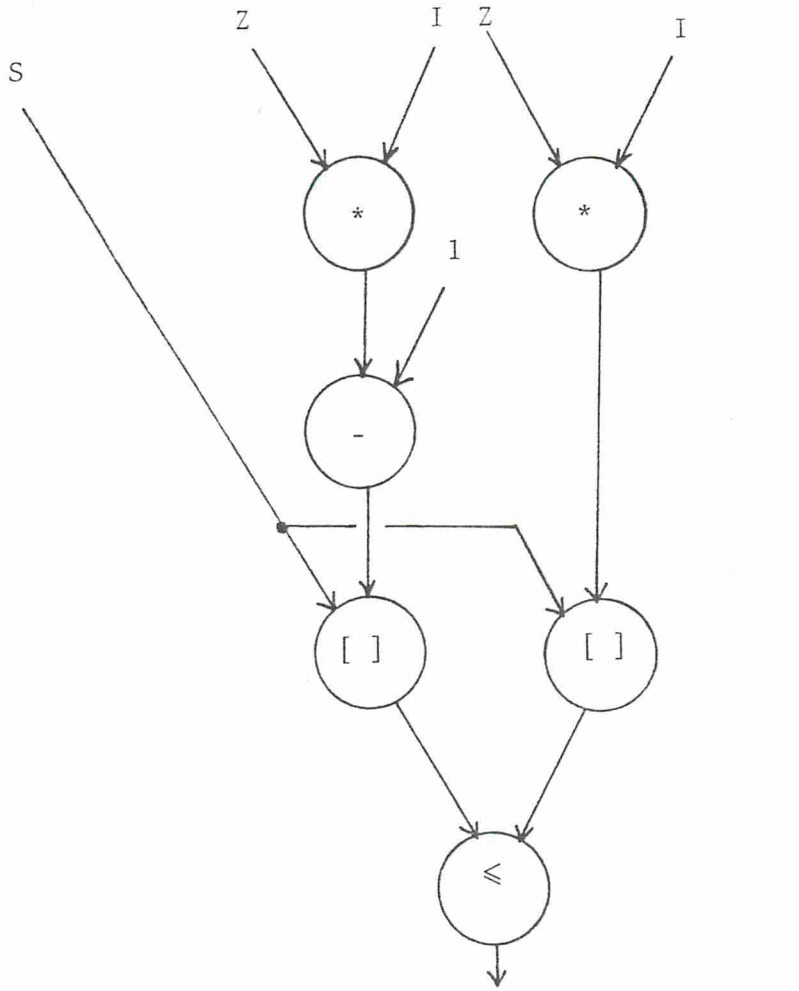


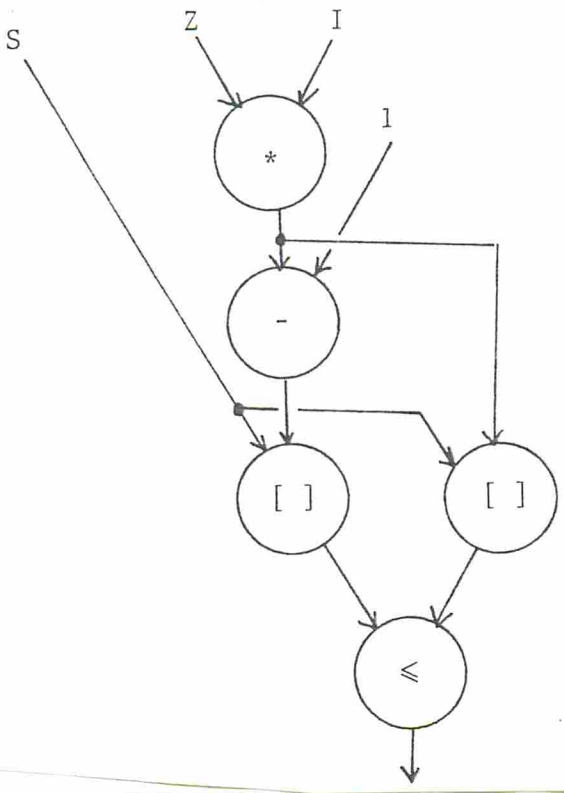
Figure 4-2: Partitioning of Register-Transfer Design

Figure 4-3: An Original and a Transformed Value Trace



Original Value Trace

If X[Z*I-1] ⩽ S[Z*I]



Value Trace
After Removal
of Redundant
Multiply

11

Figure 4-3: An Original and a Transformed Value Trace

Wagner transforms a hardware description according to certain axioms in order to prove the truth of assertions about the description. An example of this is the transformation from

```
/~CLEAR/ Q[0]<--0 ;    /~CLEAR/ Q[1]<--0;
```

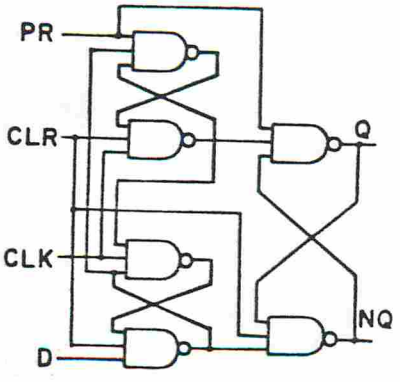
to

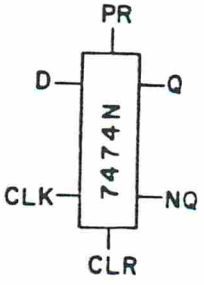
```
/~CLEAR/ Q[0:1]<--(0&0);
```

where & indicates concatenation.

Leveling by abstracting behavior from structure is performed by Leinwand and Lamdan [5] in order to perform verification. They extract a behavior, in terms of /condition/action statements, from a gate-level structure. An example taken from [5] is shown in 4-3.

flip - flop type "7474N".






```
Structural Definition of Element 7474N
Output Interfaces : Q NQ
Input Interfaces : D CLK PR CLR
Interconnections :
  7410N   #7256  #7254  CLK   #7255
  7410N   #7255  #7256  CLR   D
  7410N   #7257  PR     #7255 #7254
  7410N   #7254  #7257  CLR   CLK
  7410N   Q      NQ     PR    #7254
  7410N   NQ     Q      CLR   #7256
```

```
Functional Definition of Element 7474N
Output Interfaces : Q NQ
Input Interfaces  : D CLK PR CLR
Behavior :
/ *PR' + PR . *CLR' + CLR' . *PR + CLR . PR . *CLK /
  Q (-- *PR' + D . CLR . PR . *CLK
/ *CLR' + CLR . *PR' + PR' . *CLR + CLR . PR . *CLK /
  NQ (-- *CLR' + D' . CLR . PR . *CLK
```

Figure 4-4: Example of Leveling by Behavior Extraction

Wagner transforms a hardware description according to certain axioms in order to prove the truth of assertions about the description. An example of this is the transformation from

```
/~CLEAR/ Q[0]<--0 ;    /~CLEAR/ Q[1]<--0;
```

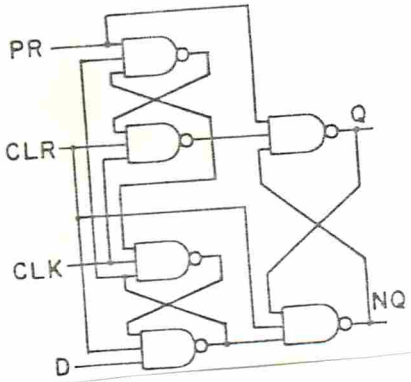
to

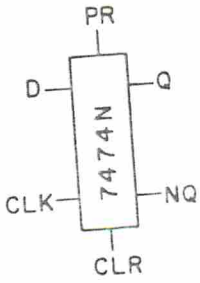
```
/~CLEAR/ Q[0:1]<--(0&0);
```

where & indicates concatenation.

Leveling by abstracting behavior from structure is performed by Leinwand and Lamdan [5] in order to perform verification. They extract a behavior, in terms of /condition/action statements, from a gate-level structure. An example taken from [5] is shown in 4-3.

flip - flop type "7474N".






```
-----------------------------------------------------------
Structural Definition of Element 7474N
-----------------------------------------------------------
Output Interfaces : Q  NQ
-----------------------------------------------------------
Input Interfaces  : D  CLK  PR  CLR
-----------------------------------------------------------
Interconnections :
  7410N    #7256   #7254    CLK    #7255
  7410N    #7255   #7256    CLR    D
  7410N    #7257   PR       #7255  #7254
  7410N    #7254   #7257    CLR    CLK
  7410N    Q       NQ       PR     #7254
  7410N    NQ      Q        CLR    #7256
-----------------------------------------------------------
-----------------------------------------------------------
Functional Definition of Element 7474N
-----------------------------------------------------------
Output Interfaces : Q  NQ
Input Interfaces  : D  CLK  PR  CLR
-----------------------------------------------------------
Behavior :
 / *PR' + PR . *CLR' + CLR'  . *PR + CLR . P
*CLK /
   Q (-- *PR' + D . CLR . PR . *CLK
 / *CLR' + CLR . *PR' + PR' . *CLR + CLR . P
*CLK /
   NQ (-- *CLR' + D' . CLR . PR . *CLK
-----------------------------------------------------------
```

Figure 4-4: Example of Leveling by Behavior Extraction

Wagner transforms a hardware description according to certain axioms in order to prove the truth of assertions about the description. An example of this is the transformation from

\~CLEAR\ Q[0] <-- 0 ;] \~CLEAR\ Q[1]<--0;

to

\~CLEAR\ Q[0:1]<--(0β0);

where β indicates concatenation.

Leveling by abstracting behavior from structure is performed by Leinwand and Lamdan [5] in order to perform verification. They extract a behavior, in terms of \condition\action statements, from a gate-level structure. An example taken from [5] is shown in 4-3.

Figure 4-4: Example of Leveling by Behavior Extraction

5. A SPECIFIC EXAMPLE: THE APPLICATION OF A SYNTHESIS MODEL TO VERIFICATION

The synthesis of register-transfer structure from register-transfer behavior can be performed by expressing the required behavior as a set of algebraic constraints on the design [4], [2]. These constraints are of two types:

1. Rules about the mapping from specific aspects of RT behavior to RT structure, including allocation of specific times to events.
2. General rules about how structures can be used to implement behavior.

Two examples of these rule types follow.

$$T_{OA}(O_a) = T_{XS}(x_a) + \sum_{d|f_d \in F_a} \sigma_{d,a} D_{FP}(f_d)$$

$$\sum_{d|f_d \in F_{a2}} \sigma_{d,a2} = 1$$

The first equation specifies that the time $T_{OA}(O_A)$ when the outputs O_a of operation x_a will be available is the time the operation began, $T_{XS}(x_a)$ and the propagation time through the operator, D_{FP}. Now, since we do not know which operator will actually be allocated to the operation x_a, we use the summation as a selection operation. $\sigma_{d,a}$ is a 0-1 variable which is set to one to indicate that operator f_d is used to implement operation x_a. Thus, the σ variable selects the proper propagation delay.

The second equation is a general one about design practices. It states that one and only one operator must be selected to implement an operation.

Now, we pose two verification problems - one straightforward, the other a little more complicated. First, suppose we wish to verify that a given set of data paths can correctly execute a required behavior, if we know the entity-

5. A SPECIFIC EXAMPLE: THE APPLICATION OF A SYNTHESIS MODEL TO VERIFICATION

The synthesis of register-transfer structure from register-transfer behavior can be performed by expressing the required behavior as a set of algebraic constraints on the design [4], [2]. These constraints are of two types:

1. Rules about the mapping from specific aspects of RT behavior to RT structure, including allocation of specific times to events.

2. General rules about how structures can be used to implement behavior.

Two examples of these rule types follow.

$$T_{OA}(O_a^{\ell}) = T_{XS}(x_a^{\ell}) + \sum_b \sigma(d,s)\, D_{PP}(t_b^{\ell})$$

$$\text{Op}\langle S\rangle\ \sigma(d,s2) = 1 \qquad d|f_{da}\, s\ F_a^{\ell}$$

$$d|f\ s\ F_{a2}^{\ell}$$

The first equation specifies that the time $T_{OA}(O_A^{\ell})$ when the outputs O_a^{ℓ} of operation x_a^{ℓ} will be available is the time the operation began, $T_{XS}(x_a^{\ell})$ and the propagation time through the operator, D_{PP}. Now, since we do not know which operator will actually be allocated to the operation x_a^{ℓ}, we use the summation as a selection operation. $\sigma^{d,s}$ is a 0-1 variable which is set to one to indicate that operator f_d^{ℓ} is used to implement operation x_a^{ℓ}. Thus, the σ variable selects the proper propagation delay.

The second equation is a general one about design practices. It states that one and only one operator must be selected to implement an operation.

Now, we pose two verification problems - one straightforward, the other a little more complicated. First, suppose we wish to verify that a given set of data paths can correctly execute a required behavior, if we know the entity-