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.

![Graph showing the relationship between test set size and simulation effectiveness.](image)

**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 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 $T$

$$T[A](P)$$

$$u\leftarrow e \quad \exists u' P(u\leftarrow u') \land u = e\leftarrow u'$$

$$x_i\leftarrow e \quad P$$

$$u\leftarrow x_i \quad \exists u', j P(u\leftarrow u', i\leftarrow j) \land j = i + 1 \land u = x_i$$

leave $f_k$ false

restart $f_k$ false

skip $P$

if $b$ then $A_1$

else $A_2$

$A_1;A_2 \quad T[A_1](P\land b) \lor T[A_2](P\land \neg b)$

$A_1;next A_2 \quad T[A_2](P) \land T[A_1](P)$

call $f_k \quad \exists \ldots$ where $f_k = A_k$

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

$$P_k(P,0) = ((I=1) \land P) \lor ((I>1) \land R[A_k,f_k](P_k(P,I-1)))$$

**DEFINITION OF B**

$$B[A](P)$$

$$u\leftarrow e \quad A:/T[A](P)/$$

$$x_i\leftarrow e \quad W(x_i):=(P \land x_i = e)/$$

$$u\leftarrow x_i \quad R(x_i):=(T[A](P))/$$

leave $f_k$ false

restart $f_k$ false

skip $A:/P/$

if $b$ then $A_1$ else $A_2$

$A_1;A_2 \quad B[A_1](P\land b) + B[A_2](P\land \neg b)$

$A_1;next A_2 \quad B[A_1](P) + B[A_2](P)$

call $f_k \quad \ldots$ 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

$$P_k(P,0) = ((I=1) \land P) \lor ((I>1) \land R[A_k,f_k](P_k(P,I-1)))$$
The process is a recursive one, where A is a projection with a range in the space, which may be an intersection or a direct sum. After A and A's range are fixed, the A is the intersection of A and the A's range, which may be a projection or a direct sum. After the A and A's range are given, the A is the intersection of A and the A's range.

The expression I(A) = (I, /I) = (I, /I)
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;

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> NEXT
          data _ b NEXT
          for _ ≠ NEXT
          bac _ ≠
        END
      END
      END
  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>;  ! if 'decs' is two
  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
  ! 'for' transitions from '0' to '1'
  ! delay forever
END

MAIN PROCESS deva;

! Second device
CLOCK 1;
LINE
  data<11:0>,
  for<>,
  bac<>;
END 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 /;
INIT Adata:=0 WHEN for EQL /;
INIT Aread:=0 WHEN bac EQL #;

! Starting conditions for Assign, Adata, and Aread

PROCESS Adata;
BEGIN
  bac _ /;
  delay 10 NEXT
  data _ soa
END;

! delay by 10 clock cycles
! write onto data lines

PROCESS Aread;
BEGIN
  a _ data;
  DELAY 75 NEXT
  for _ /
END;

! read from data lines
! and delay 75 clock cycles

PROCESS Assign;

BEGIN
  soa _ #7775 NEXT
  flag _ #
END;

! 'soa' is assigned octal value 7775
! lower flag

BEGIN !dummy process!
  DELAY WHILE 1
END;

! idle forever!

BEGIN !main process!
  flag _ /;
  DELAY WHILE 1
END

! raise flag to start Assign
! idle forever!
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 transferred 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
  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
  END

! delay has been changed from 20 to 5 ! thus causing data to be read early.
ELSE
BEGIN
   b<1> _ NOT b<1> NEXT
   data _ b NEXT
   for _ # NEXT
   dac _ #
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
         data _ b NEXT
         for _ # NEXT
         bac _ #
      END
   END

! b<0> _ NOT b<0> is missing
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>,
zero<1:0> := '00;
one<1:0> := '01;
two<1:0> := '10;

BEGIN
  delay 20 NEXT
  b =: data ;
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
  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;
VITTAL: DEVB DATA FOR BAC;
#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
#PR FOR
#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
   --> 0.011us DATA
   LOGIC= 6 SIZE= 11 PERIOD= 0.000us
   VALUES ON WIRE= 111111111111
   --> 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

! Information about FLAG
! SIZE= 0 means '1' bit
! value of FLAG is '1'
!

! refers to FLAG at .001us.
! value of FLAG is '0'
! refers to DATA at .001us
! size=11 means 12 bits
! value is 1111111111111


0.025us BAC

VALUES ON WIRE= 0

0.101us FOR

VALUES ON WIRE= 1

0.101us BAC

VALUES ON WIRE= 1

0.112us DATA

VALUES ON WIRE= 1111111101

0.124us DATA

VALUES ON WIRE= 11111111100

0.125us FOR

VALUES ON WIRE= 0

0.126us BAC

VALUES ON WIRE= 0

0.202us FOR

VALUES ON WIRE= 1

0.202us BAC

VALUES ON WIRE= 1

0.213us DATA

VALUES ON WIRE= 11111111101

0.225us DATA

VALUES ON WIRE= 11111111100

0.226us FOR

VALUES ON WIRE= 0

0.227us BAC

VALUES ON WIRE= 0

0.303us FOR

VALUES ON WIRE= 1

0.303us BAC

VALUES ON WIRE= 1

0.314us DATA

VALUES ON WIRE= 11111111101

0.326us DATA

VALUES ON WIRE= 11111111100
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
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 111111111111. All of this shows up in the simulation run but is not caught by the simulator.
<table>
<thead>
<tr>
<th>TimeStamp</th>
<th>Logic</th>
<th>Size</th>
<th>Period</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.182</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.257</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.265</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.266</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.267</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.268</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.343</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.343</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.351</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.352</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.353</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.354</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.429</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.429</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.437</td>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.438</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.439</td>
<td>BAC</td>
<td>6</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>Logic</td>
<td>Size</td>
<td>Period</td>
<td>Wire Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>--------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>6</td>
<td>11</td>
<td>0.440us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.515us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.515us</td>
<td>1</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.523us</td>
<td>11</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.524us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.525us</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.526us</td>
<td>11</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.601us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.601us</td>
<td>1</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.609us</td>
<td>11</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.610us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.611us</td>
<td>1</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.612us</td>
<td>11</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.687us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.687us</td>
<td>1</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.695us</td>
<td>11</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.696us FOR</td>
<td>6</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAC</td>
<td>0.697us</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>0.697us</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.697us FOR</td>
<td>0</td>
<td>0</td>
<td>0.000us</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 SPEAK, TTY 167]

TOPS-20 Command processor 4 (560)

! INJECTED ERROR IS A LOGICAL ERROR

@r logic
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

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

! The sequence of writes is ! as expected, but the times ! at which these writes occur ! is incorrect. Also the second ! write to data is 111111111101 ! instead of 111111111100. Thus ! it shows up in the simulation ! run but is not caught by the ! simulator.
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= 111111111101

0.322us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

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= 111111111101

0.422us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

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

0.522us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

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
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

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= 111111111101

0.922us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

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

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
! The sequence of writes is as expected, but the times at which they occur is incorrect. The 'read/write' resource conflict is not caught by the simulator. However, the omission of the NEXT statement causes a 'write/write' resource conflict which is caught by the simulator.

%More than one write to B within one time instant

% Error caught by the simulator.

%More than one write to B within one time instant
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= 111111111101

%More than one write to B within one time instant
0.222us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

0.223us FOR
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

0.224us BAC
VALUES ON WIRE= 0

0.300us FOR
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

0.303us BAC
VALUES ON WIRE= 1

0.309us DATA
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

0.311us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

%More than one write to B within one time instant
0.322us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

0.323us FOR
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

0.324us BAC
VALUES ON WIRE= 0

0.400us FOR
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

0.402us BAC
VALUES ON WIRE= 1

0.403us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101

%More than one write to B within one time instant
0.422us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

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
LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 11111111101
%More than one write to B within one time instant

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 11111111101
%More than one write to B within one time instant

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 0

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 0 PERIOD= 0.000us
VALUES ON WIRE= 1

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111101
%More than one write to B within one time instant

LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 111111111100

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= 1111111111101
%More than one write to B within one time instant

0.822us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 1111111111100

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= 1111111111101
%More than one write to B within one time instant

0.922us DATA
LOGIC= 6 SIZE= 11 PERIOD= 0.000us
VALUES ON WIRE= 1111111111100

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

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

(A), (B), and (C) are input variables

x₁ and x₂ are add operations

(A) and (C) are output variables

f₁ is an add operator

s₁ is a storage element
Figure 4-2: Partitioning of Register-Transfer Design

A. Initial Path Graph

B. Partitioned ADD2C

C. Partitioned R2 and ADD2C
Figure 4-3: An Original and a Transformed Value Trace

Original Value Trace

If $x[z\ast l] \leq s[z\ast i]$ 

Value Trace
After Removal of Redundant Multiply
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

\[-\text{CLEAR/ } Q[0] \leftarrow 0; \quad /\sim\text{CLEAR/ } Q[1] \leftarrow 0;\]

to

\[-\text{CLEAR/ } Q[0:1] \leftarrow (0\&0);\]

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.

---

**Structural Definition of Element 7474N**

**Functional Definition of Element 7474N**

---

**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

\[ \sim \text{CLEAR/ } Q[0] \leftarrow 0; \sim \text{CLEAR/ } Q[1] \leftarrow 0; \]

to

\[ \sim \text{CLEAR/ } Q[0:1] \leftarrow (0 \& 0); \]

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.

Figure 4-4: Example of Leveling by Behavior Extraction
Where a preassembled component is to be modified or improved, it may be necessary to analyze the existing system to determine the nature and extent of the modifications required. The analysis should include an assessment of the existing components and their interconnections, as well as an evaluation of the overall system performance and efficiency. This analysis can help identify areas for improvement and provide a basis for developing a design modification plan. The goal is to enhance the system's reliability, efficiency, and effectiveness, ensuring that it meets the desired performance criteria and operates within specified constraints.
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.

\[
T_{OA}(O_a) = T_{XS}(x_a) + \sum \sigma_{d,a} \cdot D_{FP}(f_d)
\]

\[
\sum \sigma_{d,a2} = 1
\]

\[d | f \in F_a \]

\[d | f \in F_{a2} \]

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 \( \sigma \) 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-
A specific example: the application of a symmetric matrix to verification

The expression of operator-structure from register-register

The expression can be performed by expressing the register register as a set of

appropriate constants of the register. [1] [2] There constitutes an effect of the

there.

.5.1.1.6.2 A new function from specific aspects of T-propagation to T

.5.4.3.1.6.2 D.1.2.6.2 A new function from specific aspects of T-propagation to T

Two examples of these results follow.

\[ (a, b) \cdot p_T (x) = \sum \delta(x - \delta) = (0) \cdot p_T \]

\[ T = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \]

The first equation specifies that the time \( T(A, 0) \) when the output \( T \)

operation \( x \) will be satisfied in the time the operation \( \bar{x} \). How, therefore, we go out from

the propagation time changes the operation \( T \). When the operation \( \delta \) and \( \delta \) are

compared as a selection operation, \( \delta \) is the 0-1-constant \( \delta \) is set to

zero to influence the operation. If we pass up to implement operation \( \delta \). Then, the

appropriate nodes for proper propagation gates.

The second equation is a special case out of constant properties. It states

the case and only one operator must be included to implement an operation.

From now, we pose two verification hypotheses: one stabilization the control

interface more complicated. Hence, we have to verify from a given set of

gate levels can correctly execute a desired operation if we know the entity.