

CPE 487: Digital System Design

Spring 2018

Lecture 14

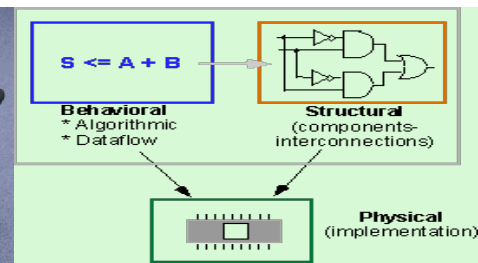
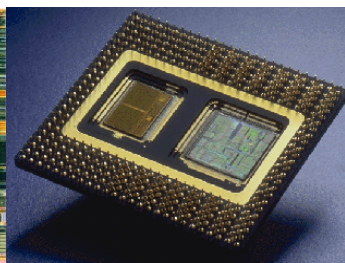
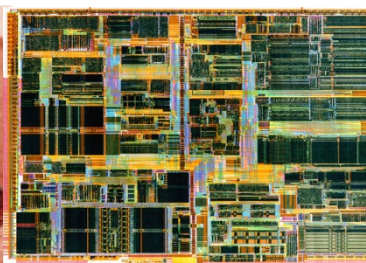
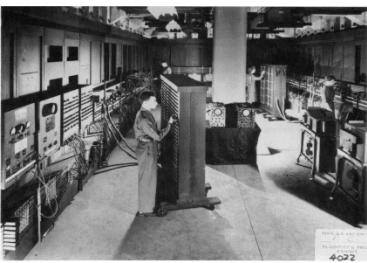
Test Bench Design

Bryan Ackland

Department of Electrical and Computer Engineering

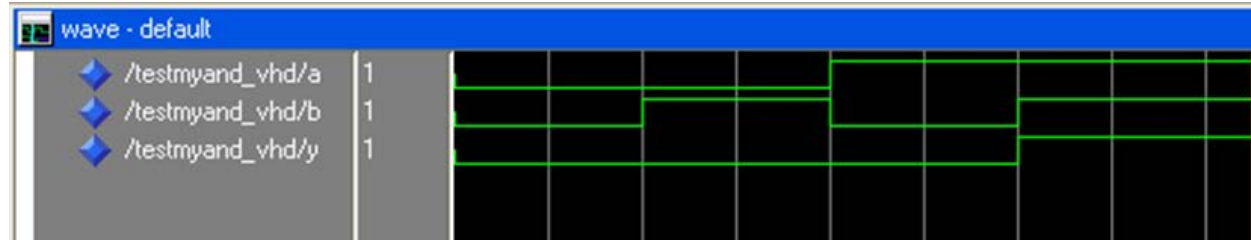
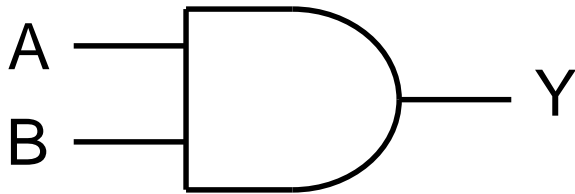
Stevens Institute of Technology

Hoboken, NJ 07030

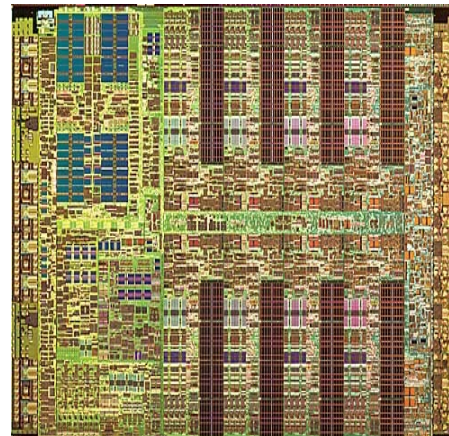


Testing Digital Circuits

- In our class examples, circuits were limited to a few input & outputs
 - Easy to set up test bench with processes to explicitly drive the inputs
 - Look at output waveforms to check correct operation



But how do we test this one?



*Processor for
SONY PlayStation 3*

Test Vectors

- For large complex designs, designers will set up a suite of test vectors
 - One or more files that contain a sequence of inputs and expected outputs

<i>inputs</i>	<i>expected outputs</i>
0010011100000110	11XX010110
1101001101100101	11X0111010
0100011010011010	1101101110
1110011111010100	0110001110

- Input vectors are typically applied to circuit under test at some regular clock rate. Outputs are read at same rate and compared to expected outputs
- It can take many millions of vectors to adequately test a complex chip e.g., a microprocessor

Design Verification vs. Product Testing

- During the design process, we develop **functional test vectors**
 - Apply vectors similar to what might be expected in normal operation of chip
 - Look for correct functionality across broad range of expected inputs
 - Purpose of these **functional vectors** is to **debug the design**
 - Each time design is refined, we reapply the functional vectors to check that no mistake has been made in moving from behavioral modeling to implementation
- Once design has been synthesized into gates, and checked once more using the functional vectors, we develop a new set of **manufacturing test vectors**
 - Now checking for **manufacturing defects**

Testing for Manufacturing Defects

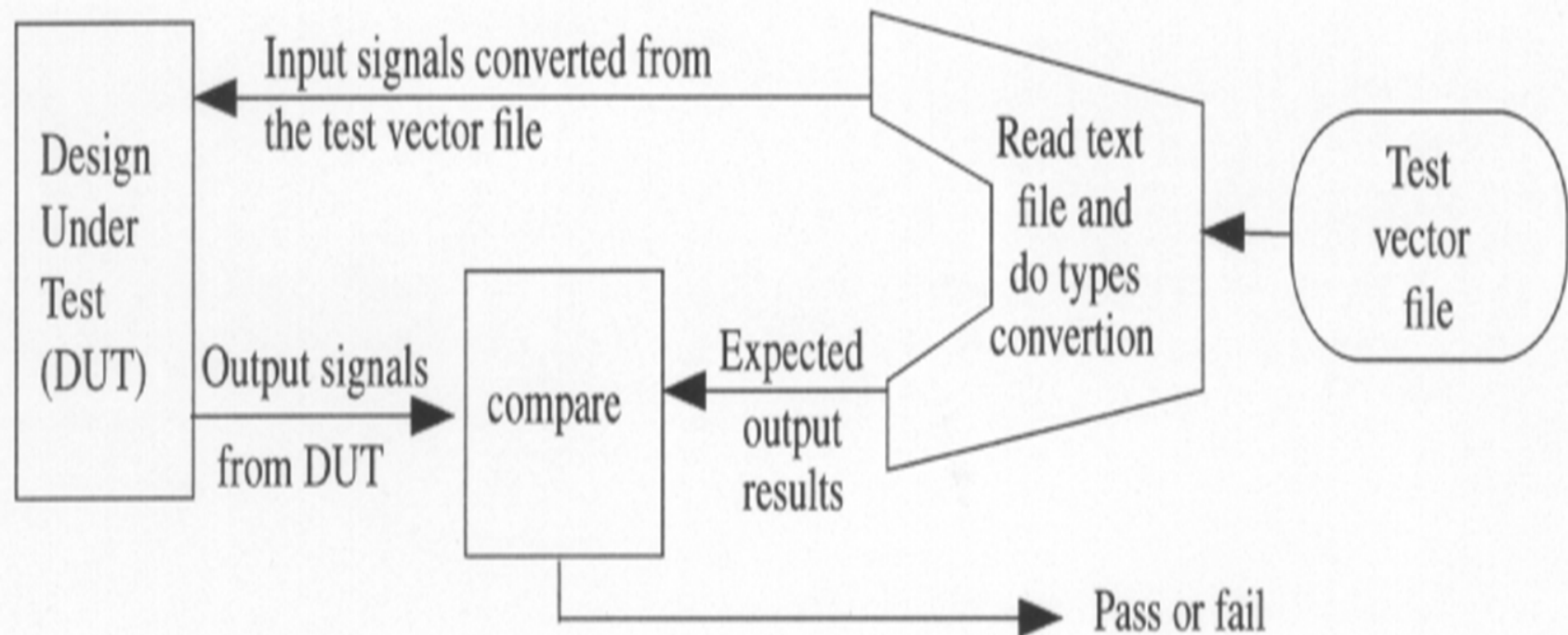
- Popular approach is to assume that all manufacturing faults can be modeled as a single node being **stuck-at** either a '0' or a '1'
- Ideally, develop a set of test vectors that would show an error if any node is stuck-at '0' or '1'
 - Not practical as it would take too many vectors to exhaustively test for all stuck at faults – tradeoff between test time and cost of sending bad chips into the field
 - Ratio of tested faults to all possible faults is known as fault coverage – 90% is considered good for a large, complex chip

Production chip tester.



Developing a Functional Test Bench

- Develop a set of functional test benches using simple example: full adder
- Move towards fully automated model:

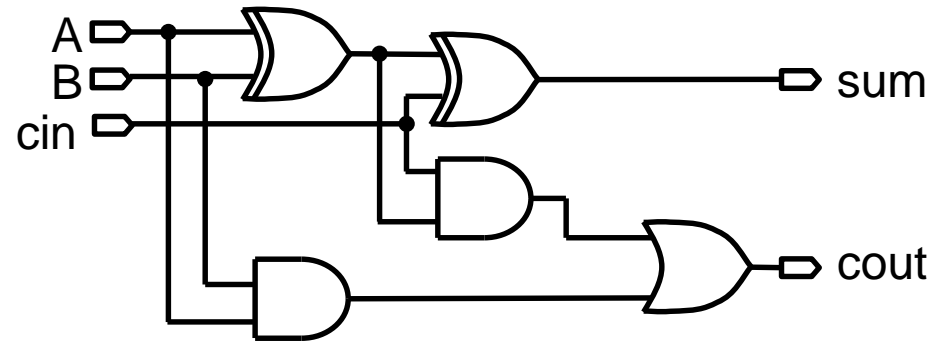


Device Under Test: Full-Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity fadder is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          cin : in  STD_LOGIC;
          sum : out  STD_LOGIC;
          cout : out  STD_LOGIC);
end fadder;

architecture gates of fadder is
    signal S1,S2,S3: std_logic;
begin
    S1 <= A xor B after 5 ns;
    S2 <= cin and S1 after 3 ns;
    S3 <= A and B after 3 ns;
    sum <= S1 xor cin after 5ns;
    cout <= S2 or S3 after 3 ns;
end gates;
```



TB1: A Simple Test Bench

```
ENTITY fadd_tb1 IS
END fadd_tb1;

ARCHITECTURE behavior OF fadd_tb1 IS
  -- Component Declaration for UUT
  COMPONENT fadder
  PORT(
    A : IN  std_logic;|
    B : IN  std_logic;
    cin : IN  std_logic;
    sum : OUT std_logic;
    cout : OUT std_logic
  );
  END COMPONENT;

  --Inputs
  signal A : std_logic := '0';
  signal B : std_logic := '0';
  signal cin : std_logic := '0';

  --Outputs
  signal sum : std_logic;
  signal cout : std_logic;

BEGIN
  -- Instantiate the UUT
  uut: fadder PORT MAP (
    A => A,
    B => B,
    cin => cin,
    sum => sum,
    cout => cout
  );
```

```
TB: process
  constant PERIOD: time:= 20ns;
  BEGIN

    A<='0'; B<='0'; cin<='0';
    wait for PERIOD;

    A<='0'; B<='1'; cin<='0';
    wait for PERIOD;

    A<='1'; B<='0'; cin<='0';
    wait for PERIOD;

    A<='1'; B<='1'; cin<='0';
    wait for PERIOD;

    A<='0'; B<='0'; cin<='1';
    wait for PERIOD;

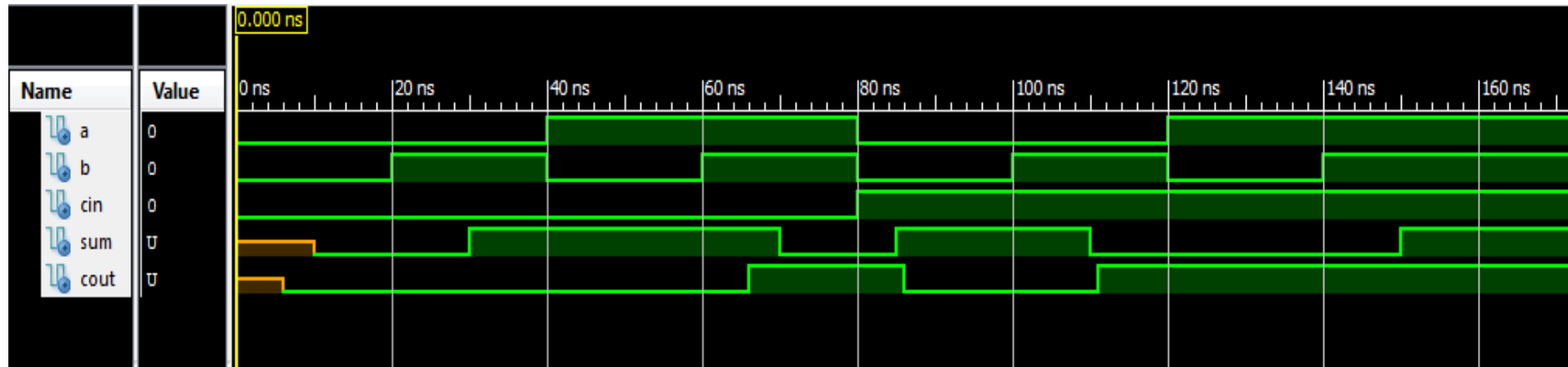
    A<='0'; B<='1'; cin<='1';
    wait for PERIOD;

    A<='1'; B<='0'; cin<='1';
    wait for PERIOD;

    A<='1'; B<='1'; cin<='1';
    wait for PERIOD;

    wait; -- will wait forever
  END process;
END;
```


TB1: Visually Check Simulation Result



This is a Lite version of ISim.

Time resolution is 1 ps

Simulator is doing circuit initialization process.

Finished circuit initialization process.

ISim>

Assert Statement (revisited)

assert *boolean-expression*

[**report** *string-expression*]

[**severity** *expression*];

- The *boolean-expression* is evaluated and if the expression is false, the *string-expression* specified in report statement is displayed in the simulator window
- The severity statement then indicates to the simulator what action should be taken in response to the assertion failure.
- Severity: NOTE, WARNING, ERROR, FAILURE.

TB2: Using Assert Statement

```
..
TB: process
  constant PERIOD: time:= 20ns;
  BEGIN

    A<='0'; B<='0'; cin<='0';
    wait for PERIOD;
    assert(sum='0' and cout='0')
      report "Test FAILED" severity error;

    A<='0'; B<='1'; cin<='0';
    wait for PERIOD;
    assert(sum='1' and cout='0')
      report "Test FAILED" severity error;

    A<='1'; B<='0'; cin<='0';
    wait for PERIOD;
    assert(sum='1' and cout='0')|
      report "Test FAILED" severity error;

    A<='1'; B<='1'; cin<='0';
    wait for PERIOD;
    assert(sum='0' and cout='1')
      report "Test FAILED" severity error;
```

```
A<='1'; B<='1'; cin<='0';
wait for PERIOD;
assert(sum='0' and cout='1')
  report "Test FAILED" severity error;

A<='0'; B<='0'; cin<='1';
wait for PERIOD;
assert(sum='1' and cout='0')
  report "Test FAILED" severity error;

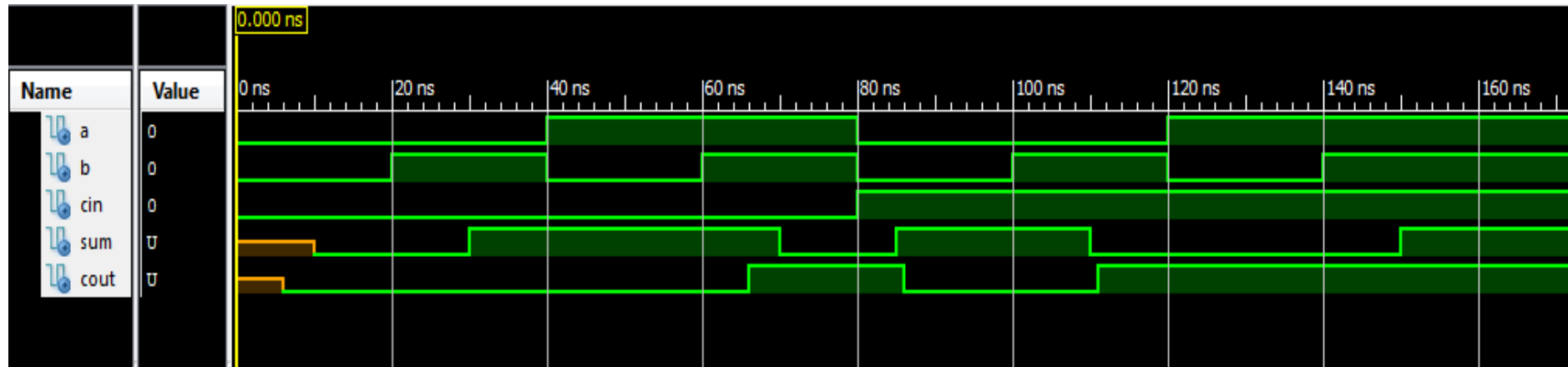
A<='0'; B<='1'; cin<='1';
wait for PERIOD;
assert(sum='0' and cout='1')
  report "Test FAILED" severity error;

A<='1'; B<='0'; cin<='1';
wait for PERIOD;
assert(sum='0' and cout='1')
  report "Test FAILED" severity error;

A<='1'; B<='1'; cin<='1';
wait for PERIOD;
assert(sum='1' and cout='1')
  report "Test FAILED" severity error;

  wait; -- will wait forever
END process;
END;
```

TB2: Using Assert – no errors



This is a Lite version of ISim.

Time resolution is 1 ps

Simulator is doing circuit initialization process.

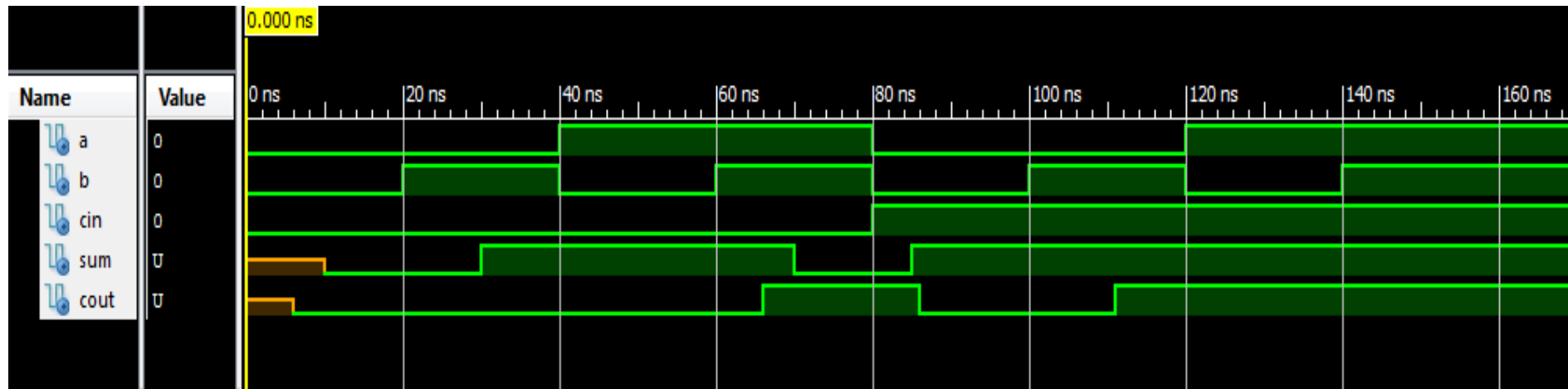
Finished circuit initialization process.

ISim>

TB2: Using Assert – Introduce error

```
architecture gates of fadder is
  signal S1,S2,S3: std_logic;
begin
  S1 <= A xor B after 5 ns;
  S2 <= cin and S1 after 3 ns;
  S3 <= A and B after 3 ns;
  sum <= S1 or cin after 5ns;
  cout <= S2 or S3 after 3 ns;
end gates;
```

Use or instead of xor



This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.

at 120 ns: Error: Test FAILED
at 140 ns: Error: Test FAILED

ISim> |

TB3: Using Test Vector Array

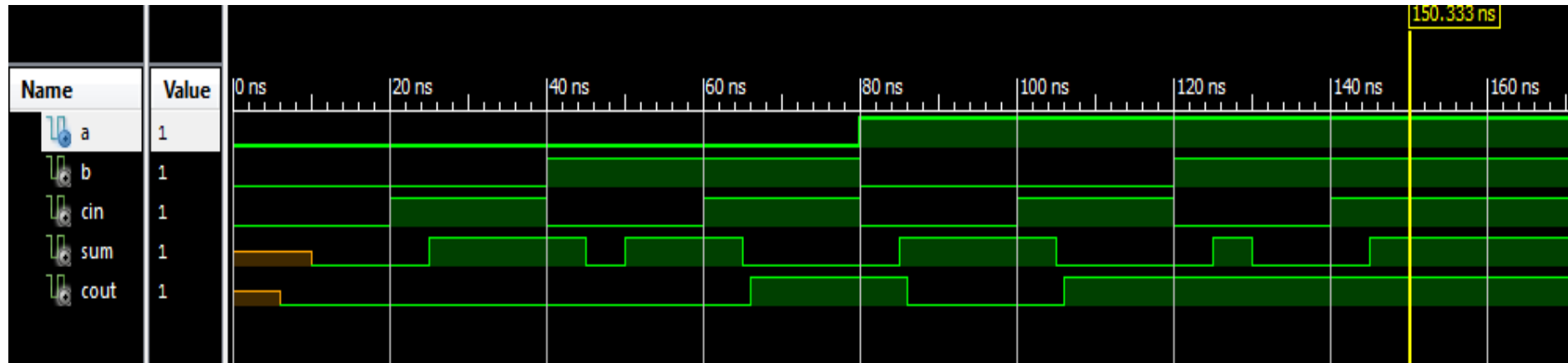
```
ARCHITECTURE behavior OF fadd_tb3 IS
  -- Component Declaration for UUT
  COMPONENT fadder
  PORT (
    A,B,cin : IN  std_logic;
    sum,cout : OUT std_logic
  );
  END COMPONENT;
  --Inputs
  signal A : std_logic := '0';
  signal B : std_logic := '0';
  signal cin : std_logic := '0';
  --Outputs
  signal sum : std_logic;
  signal cout : std_logic;
```

```
type test_array is array(integer range<>)
  of std_logic_vector(0 to 4);
constant test_vector: test_array(0 to 7):=(
('0', '0', '0', '0', '0'),
('0', '0', '1', '1', '0'),
('0', '1', '0', '1', '0'),
('0', '1', '1', '0', '1'),
('1', '0', '0', '1', '0'),
('1', '0', '1', '0', '1'),
('1', '1', '0', '0', '1'),
('1', '1', '1', '1', '1')
);
```

```
BEGIN
  -- Instantiate the UUT
  uut: fadder PORT MAP (
    A => A,
    B => B,
    cin => cin,
    sum => sum,
    cout => cout
  );
  TB: process
    constant PERIOD: time:= 20ns;
    variable testv: std_logic_vector(0 to 4);
    BEGIN
      for i in test_vector'range loop
        testv := test_vector(i);
        A <= testv(0);
        B <= testv(1);
        cin <= testv(2);
        wait for PERIOD;
        assert (sum=testv(3) and cout=testv(4))
          report "Test FAILED" severity error;
        end loop;

        wait; -- will wait forever
      END process;
  END;
```

TB3: Using Test Vector Array – no errors



```
# run 1000 ns  
Simulator is doing circuit initialization process.  
Finished circuit initialization process.  
ISim>
```

Reading and Writing Files – TEXTIO Package

- A **file** is a class of object in VHDL (like signal, variable & constant). Each file has a **type**.
- The standard VHDL library contains the **TEXTIO** package which provides a set of file types, data types and I/O procedures for simple text I/O to and from **files**
- To make the package visible:

```
use std.textio.all;
```
- TEXTIO read and write procedures are available for predefined types **bit**, **bit_vector**, **character** and **string**
- The **ieee.std_logic_textio** package overloads these procedures to support **std_logic** and **std_logic_vector**

Declaring and Opening Files

- New Types:

text -- a file of character strings

line – a string (to or from a text file)

- Example Declarations

file testfile: text; -- testfile is “file handle”

variable L: line; -- L is a single line buffer

- Procedure to open a file:

```
file_open(file_handle, filename, open_kind);
```

- where *open_kind* is one of (`read_mode`, `write_mode` or `append_mode`), for example:

```
file_open (testfile, “my_file.txt”, read_mode);
```

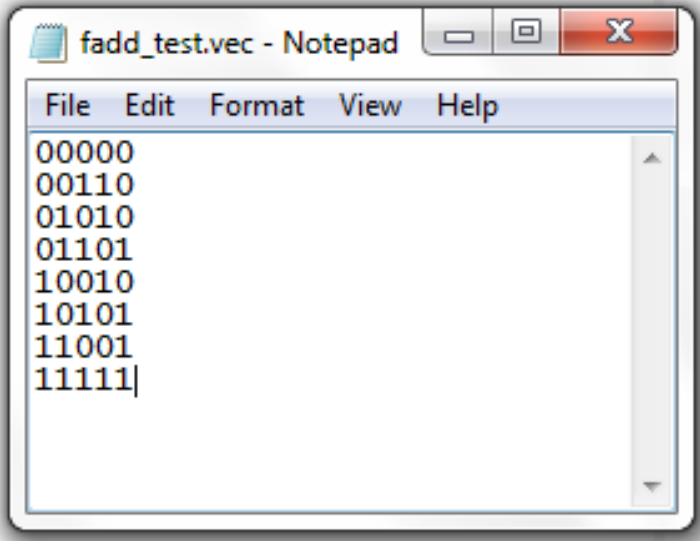
Reading Files

- `readline (file_handle, line_buffer);`
 - Read one line from "text" file `file_handle` into `line_buffer`
- `read(line_buffer, value);`
 - Read one item from `line_buffer` into variable `value`.
 - Variable `value` can be bit, bit_vector, character or string
 - Variable can also be std_logic or std_logic vector if IEEE `std_logic_textio` package is used.
- `endfile(file_handle);` --returns boolean (TRUE if at EOF)
- For example:

```
variable buf_in: line;  
variable bit0: std_logic;  
begin  
    readline(my_file, buf_in);  
    read(buf_in, bit0);
```

TB4: Reading Vectors from File

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL;  
USE std.textio.all;  
USE ieee.std_logic_textio.ALL;
```



```
fadd_test.vec - Notepad  
File Edit Format View Help  
00000  
00110  
01010  
01101  
10010  
10101  
11001  
11111
```

```
TB: process  
    constant PERIOD: time:= 20ns;  
    file vec_file: text;  
    variable buf_in: line;  
    variable testv: std_logic_vector(0 to 4);  
    BEGIN  
  
        file_open(vec_file,"fadd_test.vec", read_mode);  
        while not endfile(vec_file) loop  
            readline(vec_file, buf_in);  
            read(buf_in, testv);  
  
            --apply the stimulus from the vector  
            A <= testv(0);  
            B <= testv(1);  
            cin <= testv(2);  
            wait for PERIOD;  
            assert (sum=testv(3) and cout=testv(4))  
                report "Test FAILED" severity error;  
            end loop;  
  
            wait; -- will wait forever  
        END process;  
    END;
```

Writing Files

- `writeline (file_handle, line_buffer);`
 - Write one line to "text" file `file_handle` from `line_buffer`
- `write(line_buffer, value);`
 - Write variable value into `line_buffer`
 - Variable `value` can be bit, bit_vector, character or string
 - Variable can also be `std_logic` or `std_logic` vector if IEEE `std_logic_textio` package is used.
- For example:

```
variable buf_out: line;  
variable abc : bit_vector (3 downto 0);  
begin  
    write(buf_out, "abc is");  
    write(buf_out, abc);  
    writeline(my_file, buf_in);
```

TB5: Writing Results to File

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE std.textio.all;
USE ieee.std_logic_textio.ALL;

ENTITY fadd_tb5 IS
END fadd_tb5;

ARCHITECTURE behavior OF fadd_tb5 IS
    -- Component Declaration for UUT
    COMPONENT fadder
    PORT (
        A,B,cin : IN  std_logic;
        sum,cout : OUT std_logic
    );
END COMPONENT;
```

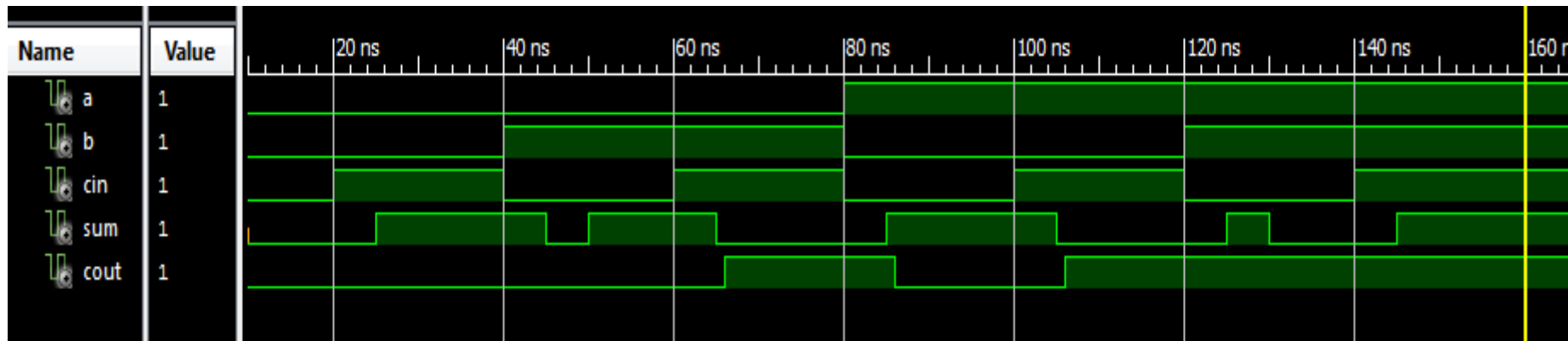
```
    --Inputs
    signal A : std_logic := '0';
    signal B : std_logic := '0';
    signal cin : std_logic := '0';
    --Outputs
    signal sum : std_logic;
    signal cout : std_logic;

BEGIN
    -- Instantiate the UUT
    uut: fadder PORT MAP (
        A => A,
        B => B,
        cin => cin,
        sum => sum,
        cout => cout
    );
```

TB5: Writing Results to File (2)

```
TB: process
    constant PERIOD: time:= 20ns;
    file vec_file, result_file: text;
    variable buf_in, buf_out: line;
    variable testv: std_logic_vector(0 to 4);
    BEGIN
        file_open(vec_file,"fadd_test.vec", read_mode);
        file_open(result_file,"fadd_test.out", write_mode);
        while not endfile(vec_file) loop
            readline(vec_file, buf_in);
            read(buf_in, testv);
            --apply the stimulus from the vector
            A <= testv(0);
            B <= testv(1);
            cin <= testv(2);
            wait for PERIOD;
            assert (sum=testv(3) and cout=testv(4))
                report "Test FAILED" severity error;
            write(buf_out, "Time="); write(buf_out, now);
            write(buf_out, ":A="); write(buf_out, testv(0));
            write(buf_out, ",B="); write(buf_out, testv(1));
            write(buf_out, ",cin="); write(buf_out, testv(2));
            write(buf_out, " ---> sum="); write(buf_out, sum);
            write(buf_out, ",cout="); write(buf_out, cout);
            writeline(result_file,buf_out);
        end loop;
        wait; -- will wait forever
    END process;
END;
```

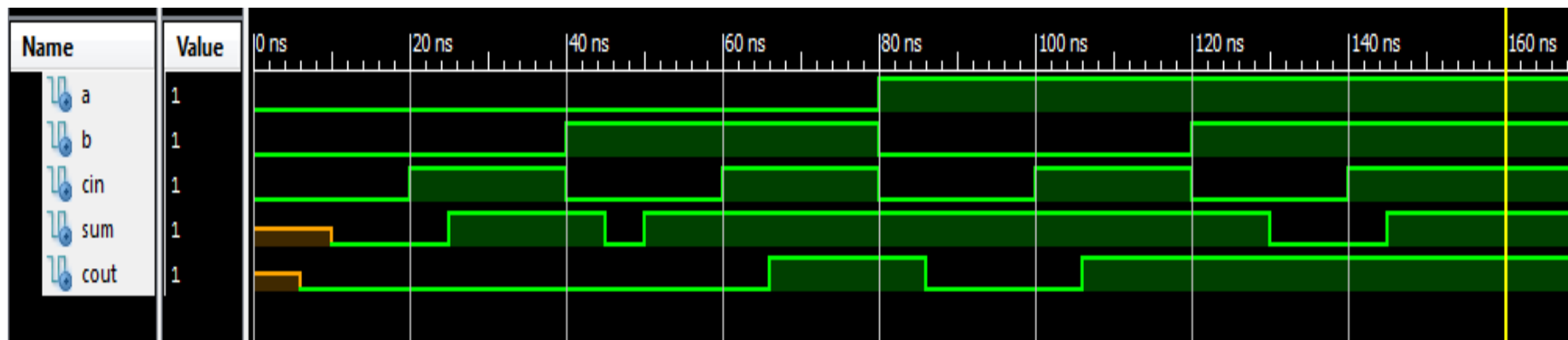
TB5: Writing Results to File (2)



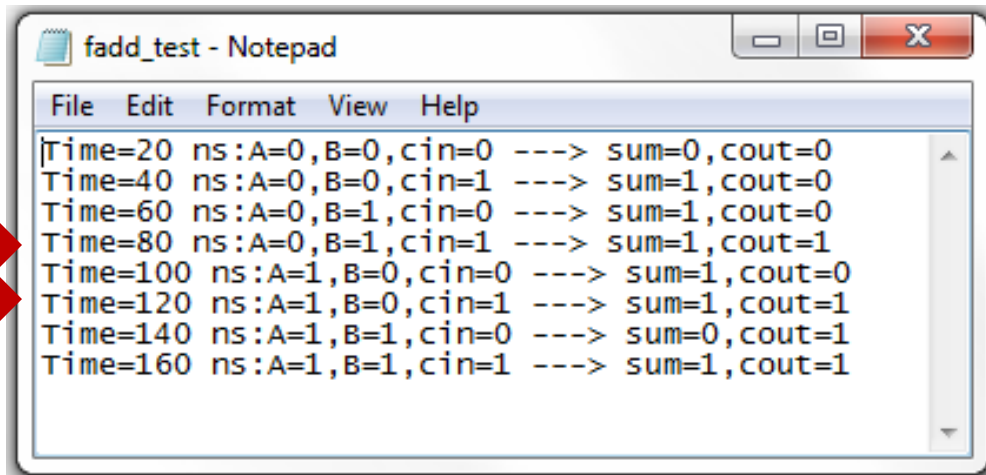
This is a Lite version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>

```
fadd_test.out - Notepad
File Edit Format View Help
Time=20 ns:A=0,B=0,cin=0 ---> sum=0,cout=0
Time=40 ns:A=0,B=0,cin=1 ---> sum=1,cout=0
Time=60 ns:A=0,B=1,cin=0 ---> sum=1,cout=0
Time=80 ns:A=0,B=1,cin=1 ---> sum=0,cout=1
Time=100 ns:A=1,B=0,cin=0 ---> sum=1,cout=0
Time=120 ns:A=1,B=0,cin=1 ---> sum=0,cout=1
Time=140 ns:A=1,B=1,cin=0 ---> sum=0,cout=1
Time=160 ns:A=1,B=1,cin=1 ---> sum=1,cout=1
```

TB5: Writing Results with OR/XOR error



```
architecture gates of fadder is
signal S1,S2,S3: std_logic;
begin
  S1 <= A xor B after 5 ns;
  S2 <= cin and S1 after 3 ns;
  S3 <= A and B after 3 ns;
  sum <= S1 or cin after 5ns;
  cout <= S2 or S3 after 3 ns;
end gates;
```



This is a Lite version of ISim.

Time resolution is 1 ps

Simulator is doing circuit initialization process.

Finished circuit initialization process.

at 80 ns: Error: Test FAILED
at 120 ns: Error: Test FAILED

ISim>

Example: Writing Test Vectors

- Write a set of test vectors to test the following 2-bit binary counter:

res	ck	up/ $\overline{\text{dn}}$	Q
0	X	X	00
1	↑	1	count up
1	↑	0	count dn

