Lecture 15

File I/O Based Testbench

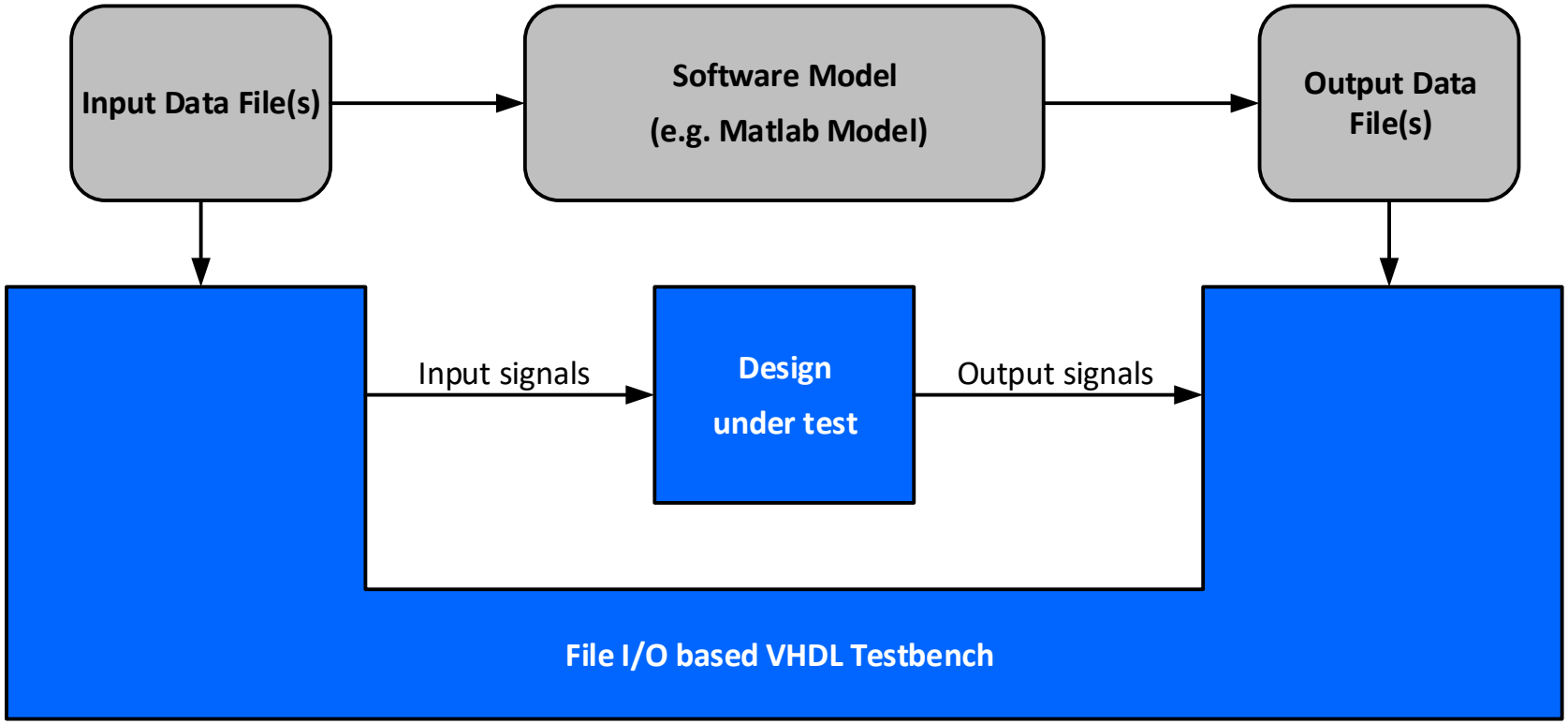
Overview

- **Aim**
  - To understand the use of TEXTIO in reading and writing text files and then write file I/O based testbench

- **Topics Covered**
  - TEXTIO
  - READ and WRITE
  - TEXTIO for stimulus and outputs
  - STD_LOGIC_TEXTIO
  - Example – File I/O based testbench for a floating-point multiplier VHDL design

File I/O based VHDL Testbenches

- **The File I/O based VHDL testbench includes code to read an input data file and feed the input data to the input data signals, save the output data signals to an output data file and possibly code to compare the output data with the expected data from the software model.**



Introduction to Subprograms

- Subprogram may be either procedures or functions
- Procedures may return values via their arguments
- Functions return a single result value as part of an expression
- Subprograms may be overloaded

```
S <= to_signed(I, 8);
```

Function from ieee.numeric_std

```
Write(L, A);
```

```
read(L, A);
```

Procedures from std.textio

Reading and Writing Text Files

- **Text files are used because they are portable**
- **Standard reading and writing procedures are provided in STD.TEXTIO**
- **A different operator (:=) is used for variable assignment than signal assignment to reflect the different update behaviour between variables and signals**

```
use std.textio.all;
...
FWRITE: process is
  variable L: line;
  variable W: bit;
begin
  ...
  W := '1';
  write(L, W);
  writeline(F, L);
end process FWRITE;
```

Variables are declared within a prcocess

Data type from TEXTIO

Variable assignment

Procedures from TEXTIO

Overloaded Write Procedures

- **Overloaded subprogram avoids having to invent a large number of subprogram names to cover all the combinations of operations and type**

Procedures in TEXTIO for all standard VHDL types

```
procedure WRITE (L: inout LINE; VALUE: in BIT);
procedure WRITE (L: inout LINE; VALUE: in BIT_VECTOR);
procedure WRITE (L: inout LINE; VALUE: in BOOLEAN);
procedure WRITE (L: inout LINE; VALUE: in CHARACTER);
procedure WRITE (L: inout LINE; VALUE: in INTEGER);
procedure WRITE (L: inout LINE; VALUE: in REAL);
procedure WRITE (L: inout LINE; VALUE: in STRING);
procedure WRITE (L: inout LINE; VALUE: in TIME);
```

```
WRITE (L, FALSE);
WRITE (L, 0);
WRITE (L, 1.0);
WRITE (L, 10 NS);
WRITE (L, "Text");

WRITE (L, STRING'("Text"));
```

```
BOOLEAN
INTEGER
REAL
TIME
Ambiguous

Qualified
```

Note!

TEXTIO Output

- **Two stages:**
  - Use procedure WRITE to write items to a line
  - Use procedure WRITELINE to write the line to a file, then clear the line buffer, ready for further calls to WRITE

```
1 use std.textio.all;
  ...
  signal A, B, G: bit_vector(3 downto 0);
  ...
  Monitor: process is
2   file F: text open write_mode is "test.txt";
    variable L: line; 3
  begin
    wait until rising_edge(Clock);
    wait for Settling_time;

    write(L, NOW, Left, 10);
    write(L, A);
    write(L, B, Right, 5);
    write(L, G, Right, 6);

    writeline(F, L);
  end process Monitor;
```

NOW = simulation time

10	4	5	6
0 NS	0000	0000	0000
50 NS	1111	1111	1111
100 NS	0101	0101	0101

TEXTIO Input

- **Two stages:**
  - READLINE reads the next line from the file into the line buffer
  - READ reads the next item from the line buffer into a variable from the start of the line buffer

```
1 use std.textio.all;
  ...
  signal A, B, G: bit_vector(3 downto 0);
  ...
  Stimulus: process is
2   file F: text open read_mode is "test.txt";
    variable L: line;  3
    variable Avalue, Bvalue: bit_vector(3 downto 0);
  begin
    while not endfile(F) loop
      readline(F, L);
      read(L, Avalue);
      read(L, Bvalue);
      wait until rising_edge(Clock);
      A <= Avalue;
      B <= Bvalue;
    end loop
    wait;
  end process Stimulus;
```

TRUE at end of file

You must read into variables

0000	0000
1111	1111
0101	0101

Procedure READ

- READ skips spaces, except for types Character and String
- For data types CHARACTER and STRING, READ reads exactly the number of characters required by the length of the value parameter, starting with the next character on the line

```
    100 NS     99    ABCDEF     27
```

```
    variable Tim: time;
    variable Int: integer;
    variable Cha: character;
    variable Str: string(1 to 6);
  begin
    readline(F, L);
    read(L, Tim);   -- Tim = 100 NS
    read(L, Int);   -- Int = 99
    read(L, Cha);   -- Cha = ' '
    read(L, Str);   -- Str = "  ABCD"
    read(L, Int);   -- ERROR!
```

Package STD_LOGIC_TEXTIO

- **The WRITE and READ procedures defined in TEXTIO do not support STD_LOGIC, so we need STD_LOGIC_TEXTIO**
- **STD_LOGIC_TEXTIO is not an IEEE standard package**


```
procedure READ(L: inout LINE;
               VALUE: out STD_ULOGIC/STD_LOGIC_VECTOR);

procedure WRITE(L: inout LINE;
               VALUE: out STD_ULOGIC/STD_LOGIC_VECTOR;
               JUSTIFIED: in SIDE := RIGHT;
               FIELD: in WIDTH := 0);

procedure HREAD/OREAD(L: inout LINE;
                      VALUE: out STD_LOGIC_VECTOR);

procedure HWRITE/OWRITE(L: inout LINE;
                        VALUE: in STD_LOGIC_VECTOR;
                        JUSTIFIED: in SIDE := RIGHT;
                        FIELD: in WIDTH := 0);
```

Hex/Octal

Using STD_LOGIC_TEXTIO

- **Use STD_LOGIC_TEXTIO to rewrite the earlier file reading example for type STD_LOGIC_VECTOR**

```
use STD.TEXTIO.all;
use IEEE.STD_LOGIC_TEXTIO.all;
...
signal A, B, G: STD_LOGIC_VECTOR(3 downto 0);
...
Stimulus: process is
  file F: TEXT open READ_MODE is "vectors.txt";
  variable L: LINE;
  variable Avalue, Bvalue: STD_LOGIC_VECTOR(3 downto 0);
begin
  while not ENDFILE(F) loop
    READLINE(F, L);
    READ(L, Avalue);
    READ(L, Bvalue);
    wait until Rising_edge(Clock);
    A <= Avalue;
    B <= Bvalue;
  end loop
  wait;
end process Stimulus;
```

You need both packages

Floating-Point Multiplier

- **With double precision**
- **Files:**
  - FpMultiplier.vhd - VHDL design code
  - tb_FpMultiplier.vhd - File I/O based VHDL testbench code
  - RandomFpMulTest.m - Matlab code to generate the test data
  - InputX.txt - data file for the test data from Matlab
  - InputY.txt - data file for the test data from Matlab
  - OutputR_matlab - data file for the test data from Matlab
  - OutputR - data file for the result for VHDL simulation