# HDL Lecture 6

Design Verification

Testbenches

Generate Statements

## Design Verification

- Design Verification
  - **Crucial and time consuming part of design process**
    - Rule of thumb: 2x time to write design
- Goal is to verify that design behaves as expected
  - **Must verify correct result for valid inputs**
  - **Must verify correct result for invalid inputs**
- Simulation verifies design BEFORE synthesis and place & route.
  - **Design errors cost more as design cycle progresses**
    - EX: ASIC $10K's for each layout change

## VHDL Simulators

- Most simulators provide real-time interaction of inputs
  - **Inputs assigned, simulation time advanced, outputs inspected**
- Test Benches
  - **VHDL Model that applies test vectors to design (UUT)**
  - **Outputs test vectors can be observed via waveform, recorded to file or compared to expected value within test bench.**
  - **Advantages**
    - Allows input and output vectors to be documented
    - More methodical approach than visual inspection
    - Once defined, test bench can be rerun multiple times
    - Test bench can be used for both functional and gate level (timing) simulation

## Types of Test Benches

- Class 1
  - Test Bench generates inputs signals for model.
  - Output signals must be verified manually
  - No timing verified.
- Class 2 (self-checking)
  - Test Bench generates inputs signals for model.
  - Test Bench verifies the correct value of output signals
  - No timing verified.
- Class 3 (self-checking w/ timing)
  - Test Bench generates inputs signals for model.
  - Test Bench verifies the correct value of output signals
  - Test Bench verifies timing of model.

## Test Bench

```
--Library commands & packages

ENTITY <testbench_identifier> IS
END <testbench_identifier>;                 -- VHDL-1987 version
END ENTITY < testbench_identifier > ;    -- VHDL-1993 version

ARCHITECTURE <identifier> OF < testbench_identifier> IS
        --architecture declaration section
        --Signal Declarations (define signal for all ports on UUT)
        --UUT Component Declaration if no package
BEGIN
       Unit Under Test (UUT) Component instantiation statements
Process or functions for stimulus
END <architecture identifier> ;                  -- VHDL-1987 version
END ARCHITECTURE <architecture identifier>;   -- VHDL-1993 version

CONFIGURATION <cfg_identifier> OF < testbench_identifier> IS
     FOR <identifier>
     END FOR;
END <cfg_identifier> ;
```

# Example: AND Gate Test Bench

```
LIBRARY work;
USE work.MyAndGate_pkg.ALL;

ENTITY tb_MyAndGate IS
END tb_MyAndGate;

ARCHITECTURE test OF tb_MyAndGate IS

    SIGNAL a, b, c : bit;

BEGIN

    UUT: MyAndGate PORT MAP (
        a_in => a,
        b_in => b,
        --
        c_out => c
    );
```

**UUT is defined in package. If not then you must use component declaration in architecture**

**Test bench ENTITY has no PORT**

***Define all ports on UUT as signals***

**Unit Under Test Component instantiation**

## Example: AND Gate Test Bench

```
    stimulus: PROCESS
    BEGIN
        -- set inputs a = b = 0
        a <= '0';
        b <= '0';
        WAIT 20 ns;

        -- set inputs a = 1 and b = 0
        a <= '1';
        b <= '0';
        WAIT 20 ns;

        WAIT;   -- stop simulation
    END PROCESS;

END ARCHITECTURE test ;

CONFIGURATION cfg_tb_MyAndGate OF tb_MyAndGate IS
    FOR test
    END FOR;
END cfg_tb_MyAndGate;
```

*Define test vectors to apply to UUT*

# ASSERT Statement

```
ASSERT <expression>  <report> <severity>;              -- VHDL-1987 version
<label>: ASSERT <expression>  <report> <severity>;  -- VHDL-1993 version
```

- Assertion statements can be used to check for condition.
  - **Used in test benches to report problems.**
- <*expression*>
  - **If expression is NOT TRUE then ASSERT is executed**
- <*report*>
  - **Optional clause to define a string to provide more information to user during simulation**
  - **Default string is "Assertion Violation" if clause not defined**
- <*severity*>
  - **Optional clause to indicate how bad the violation is**
  - **Severity level can be NOTE, WARNING, ERROR, or FAILURE**
    - Default is ERROR if not defined
  - **Simulator uses level to determine if simulation should continue**

## ASSERT Statement

- Example

```
Fifo_PAF: ASSERT (FifoCount < FIFO_PAF_c)
        REPORT "FIFO ALMOST FULL" SEVERITY NOTE;
```

  - Displays message only when FifoCount is greater than or equal to FIFO_PAF_c

```
Fifo_FF: ASSERT (FifoCount < FIFO_SIZE_c) REPORT "FIFO OVERFLOW"
             SEVERITY FAILURE;
```

  - Displays message only when FifoCount is greater than or equal to FIFO_SIZE_c

## REPORT Statement

<*label*>: **REPORT** <*report*> <*severity*>;  *( 1076-1993 version)*

- Report Statement
  - **Used to trace out model execution by creating a trace of where execution is**
- Similar to ASSERT statement except not expression
- Report is NOT supported in 1076-1987
- <*severity*> clause
  - **Same as ASSERT statement except default is NOTE**

Example

```
FIFO_CK: REPORT REPORT “IN FIFO CHECK LOGIC” SEVERITY NOTE;
```

## FILE IO

- File access is part of the *textio* LIBRARY
- File access is not synthesizable
- File access used mostly for test benches
  - **Provides an easy way to store stimulus without changing code**
    - Once code is written, many different stimulus files can be applied to design
  - **Provides an easy way to store output results**
    - Results can be saved to file for verification (better then visual inspection)
  - **Example use: Image Processing designs**
    - Send in an known image, output an image
- VHDL-93
  - **VHDL93 allows files to be explicitly opened and closed during simulation**

## FILE IO

**FILE <*file_handle*> : <*file_type*> OPEN <*file_mode*> IS "*file_name*"**

- <*file_handle*> defines the logical file handle to the file
- <*file_type*> defines the file type
  - **Ex: text, integer_file**
- <*file_mode*> defines how you want to open the file
  - **Ex: read_mode, write_mode, append_mode**
  - **VHDL-87 does not support file mode. It uses keywords IN and OUT.**

Examples:

```
FILE ImageIn : integer_file OPEN read_mode IS "foo.raw";
FILE ImageOut: text OPEN write_mode IS "foobar.raw";
```

- Use file_close(<*file_handle*>) to close the file

Examples:

```
file_close(ImageIn);
file_close(ImageOut);
```

## File IO

- READLINE function
  - Reads a complete line from the file
  - Example: `readline (CommandFile, InputLine);`
- READ function
  - Read up to the first space
  - Example: `read (InputLine, command);`
- WRITE function
  - Writes to a line
  - Example: `write (OutputLine, command);`
- WRITELINE function
  - Writes a complete line to the file
  - Example: `writeline (Outputfile, OutputLine);`

# File IO

- ENDFILE function
  - **Determines when we have reached the end of file**
  - **Returns a Boolean value**
  - **Example:** `endfile (InputLine);`
- Types needed for reading text files
  - **Input and output operations using textio are based on dynamic strings that are accessed using pointers of type LINE**
    - Example: `VARIABLE InputLine   : line;`
      - **This variable that we can store a string object (line)**
  - **Characters are a predefined enumerate type.**
    - Useful when reading a file to parse the line into characters
    - Example: `VARIABLE command     : character;`
  - **Integers are a predefined enumerate type.**
    - Useful when reading a file to parse the line into integers
    - Example: `VARIABLE data     : integer;`

## File IO Example

- Read Example

```
read_file : PROCESS
       FILE InputFile   text OPEN read_mode IS “input.txt";
       VARIABLE InputLine  : line;
       VARIABLE command    : character;
       VARIABLE data       : integer;
   BEGIN
       WHILE NOT endfile (InputFile) LOOP
           readline (InputFile, InputLine);
           read (InputLine, command);
           read (InputLine, data);
       END LOOP;
      file_Close (InputFile);

      REPORT "*** File Reading Done **";
      WAIT;
   END PROCESS read_file;
```

## File IO Example

- Write Example

NEED: use STD.textio.all;

```
 write_file : PROCESS
        FILE OutputFile:text OPEN write_mode IS “output.txt";
        VARIABLE OutputLine : line;
BEGIN
       write (OutputLine, string'(" The first line = ") );
       write (OutputLine, command);
       write (OutputLine, ' ');
       write (OutputLine, data_a);
       write (OutputLine, ' ');
       write (OutputLine, data_b);
       writeline (OutputFile ,OutputLine);

       file_Close (OutputFile);

       REPORT "*** File Writing Done **";
       WAIT;
    END PROCESS write_file;
```

R·I·T

## Generate Statements

## Generation Statements

```
<label>: FOR <identifier> IN range GENERATE
   -- concurrent statements
END GENERATE <label> ;
```

- Generate statements provide a means for a designer create replicated structures
- Generated statements can contain IF-THEN and loop constructs
  - **Used to handle cases where first or last structure is not the same as the others**

## Generation Statements

- Example #1

```
ARCHITECTURE GEN OF REG_BANK IS
    COMPONENT REG
        PORT(D, CLK, RESET : IN  std_ulogic;
             Q             : OUT std_ulogic);
    END COMPONENT;
BEGIN
    GEN_REG : FOR I IN 0 TO 3 GENERATE
        REGX : REG PORT MAP (
           D     => DIN(I),
           CLK   => CLK,
           RESET => RESET,
           Q     => DOUT(I)
        );
    END GENERATE GEN_REG;
END GEN;
```

## Generation Statements

- Example #1 w/o generate

```
ARCHITECTURE GEN OF REG_BANK IS
    COMPONENT REG
        PORT(D, CLK, RESET : IN  std_ulogic;
             Q             : OUT std_ulogic);
    END COMPONENT;
BEGIN
    -- same as generate statement
    -- positional association used to conserve space
    REG0 : REG PORT MAP ( DIN(0),CLK, RESET, DOUT(0) );
    REG1 : REG PORT MAP ( DIN(1),CLK, RESET, DOUT(1) );
    REG2 : REG PORT MAP ( DIN(2),CLK, RESET, DOUT(2) );
    REG3 : REG PORT MAP ( DIN(3),CLK, RESET, DOUT(3) );

END GEN;
```

## Generation Statements

- Example #2

```
ARCHITECTURE GEN OF RIPPLE IS

    COMPONENT FULLADD
        PORT (A, B, CIN   : IN  bit;
              SUM, CARRY : OUT bit);
    END COMPONENT;

    COMPONENT HALFADD
        PORT(A, B         : IN  bit;
             SUM, CARRY   : OUT bit);
    END COMPONENT;

    SIGNAL C  :     bit_vector(0 TO 7);
```

## Generation Statements

- Example #3

```
BEGIN
    GEN_ADD : FOR I IN 0 TO 7 GENERATE

        LOWER_BIT : IF I = 0 GENERATE
            U0 : HALFADD PORT MAP
                (A(I), B(I), S(I), C(I));
        END GENERATE LOWER_BIT;

        UPPER_BITS : IF I > 0 GENERATE
            UX : FULLADD PORT MAP
                (A(I), B(I), C(I-1), S(I), C(I));
        END GENERATE UPPER_BITS;

    END GENERATE GEN_ADD;

END GEN;
```