

Module 13 : Advanced Concepts In VHDL

Tutorial and Exercises

For the VeriBest Simulator

Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds “Unlimited Rights” in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice .

See the [RASSP Disclaimer file](#) for additional RASSP Disclaimer, Warranty and Limitation of Liability Information concerning the material, VHDL code and software developed under the RASSP programs or incorporated in RASSP material.

1. Getting Started

- 1.1. The Module 13 lab will demonstrate the use of functions and test vectors. Some VHDL models from Modules 10, 11 and 12 will be reused through use of library mappings.
- 1.2. The first VHDL files that you will need are **package.vhdl**, **test_and2.vhdl**, and **test_mult.vhdl**. These include behavioral descriptions of the testbenches for the **and2** gate used in Module 10 and the multiplier used in Module 12.
- 1.3. In order for these VHDL models to compile correctly, some configuration of the libraries is needed. Create a library mapping from the logical library name "gate_lib" to the working library created in module 10.
- 1.4. Create a library mapping from the logical library name "mult_lib" to the working library created in module 11.
- 1.5. Create a library mapping from the logical library name "behav_lib" to the working library created in module 12.
- 1.6. There are also two files that contain the input vectors used with these testbenches. They are **test_and2.vec** and **test_mult.vec**. You can examine these files with any text editor.

2. Open and compile the package containing some functions to be used in this lab

- 2.1. Open the file **package.vhdl** using a text editor or a VHDL editing environment.

```
LIBRARY gate_lib;
USE gate_lib.resources.ALL;

PACKAGE resources IS

    FUNCTION bit2level (bit_in : BIT) RETURN level;

    FUNCTION bit2level (bit_vector_in : BIT_VECTOR)
        RETURN level_vector;

END resources;

PACKAGE BODY resources IS

    FUNCTION bit2level (bit_in : BIT) RETURN level IS

        BEGIN

            CASE bit_in IS

                WHEN '0' => RETURN '0';
```

Module 13 - Advanced Concepts in VHDL

```
        WHEN '1' => RETURN '1';
        WHEN OTHERS => ASSERT FALSE
            REPORT "Cannot do a conversion to BIT"
            SEVERITY ERROR;
    END CASE;
    RETURN '0';
END bit2level;

FUNCTION bit2level (bit_vector_in : BIT_VECTOR) RETURN level_vector IS

    variable I : integer;
    variable level_vector_out : level_vector(bit_vector_in'RANGE);

BEGIN

    FOR I IN bit_vector_in'RANGE LOOP
        CASE bit_vector_in(I) IS
            WHEN '0' => level_vector_out(I) := '0';
            WHEN '1' => level_vector_out(I) := '1';
            WHEN OTHERS => ASSERT FALSE
                REPORT "Cannot do a conversion to BIT"
                SEVERITY ERROR;
        END CASE;
    END LOOP;
    RETURN level_vector_out;
END bit2level;

END resources;
```

2.2. Compile the **package.vhdl** model. The model should compile without any errors.

3. Examine and compile the testbench for the and2 gate

3.1. Open the file **test_and2.vhdl** using a text editor or a VHDL editing environment.

```
LIBRARY gate_lib;
USE gate_lib.resources.ALL;
USE work.resources.ALL;
USE std.textio.ALL;

ENTITY test_and2 IS END;

ARCHITECTURE test OF test_and2 IS
    FILE datain: TEXT OPEN READ_MODE IS "test_and2.vec";
    SIGNAL a, b, c : level;
```

Module 13 - Advanced Concepts in VHDL

```
BEGIN

Gate0: ENTITY gate_lib.and2(behav)
    PORT MAP (a,b,c);

Action: PROCESS

    VARIABLE L : LINE;
    VARIABLE okay : BOOLEAN := TRUE;

    VARIABLE sim_time : TIME;
    VARIABLE a_in,b_in : BIT;
    VARIABLE delay : TIME;

    BEGIN

    WHILE NOT(ENDFILE(datain)) LOOP
        READLINE(DATAIN,L);
        READ(L,sim_time,okay);
        READ(L,a_in,okay);
        READ(L,b_in,okay);
        IF okay THEN
            delay := sim_time-now;
            a <= bit2level(a_in) AFTER delay;
            b <= bit2level(b_in) AFTER delay;
            WAIT FOR delay;
        END IF;
    END LOOP;
    WAIT UNTIL FALSE;
    END PROCESS;

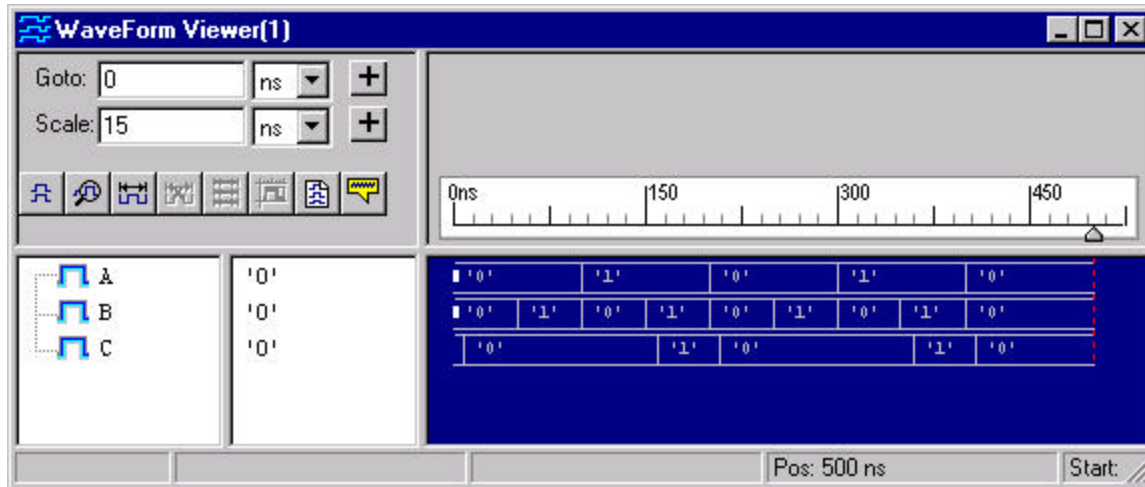
END test;
```

3.2. Compile the **test_and2.vhdl** model. The model should compile without any errors.

4. Simulate the compiled code

4.1. Start the VHDL simulator. Remember to select the entity **test_and2** (or its architecture **test**) as the design root, if required by your simulator.

- 4.2. Open a waveform window, then add the signals "A", "B", and "C" from the **and2** component.
- 4.3. Run the simulator for 500 nanoseconds. The signal waveforms should look something like this:



- 4.4. By editing the contents of the **test_and2.vec** file, you may change the input values and the times at which they will be assigned.

5. Testing the Booth Multiplier

- 5.1. Using a similar procedure to the one just used to test the and2 gate, we may test the Booth multiplier from the Module 12 lab using a testbench approach. First, compile the VHDL code from the file **test_mult.vhdl**.
- 5.2. Start the VHDL simulator. Remember to select the entity **test_mult** (or its architecture **test**) as the design root, if required by your simulator.
- 5.3. Open a waveform window, then add the signals from the full multiplier component. Alternately, if the simulator offers a list format for signal values, display a list of signals. For some simulations, a list may be more convenient to view than a waveform.
- 5.4. Run the simulator for 5000 nanoseconds.
- 5.5. Again, the input values for this simulation may be changed by editing the file **test_mult.vec**.