

Module 11 : Structural 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 11 lab will provide practice with structural VHDL models. The basic gates created and compiled for the Module 10 lab will be reused through the use of library mappings. Simple components will be used to form larger, more useful structures.

1.2. The first VHDL file that you will need is **aoi2_str.vhdl**.

2. Examine and compile the and/or/inverter model

2.1. Open the file **aoi2_str.vhdl** using a text editor or a VHDL editing environment.

```
-----  
--      And/OR Invert Structural Example      --  
--      RASSP E&F Module # 11 Structural VHDL  --  
--      Robert Klenke UVa 19 April 1996      --  
-----  
  
LIBRARY gate_lib;  
USE gate_lib.resources.all;  
  
ENTITY aoi2_str is  
  
    GENERIC(trise : delay := 12 ns;  
            tfall : delay := 9 ns);  
  
    PORT(a : IN level;  
          b : IN level;  
          c : IN level;  
          d : OUT level);  
  
END aoi2_str;  
  
ARCHITECTURE structural OF aoi2_str IS  
  
    COMPONENT and2  
        GENERIC(trise : delay;  
                tfall : delay);  
        PORT(a : IN level;  
              b : IN level;  
              c : OUT level);  
    END COMPONENT;  
  
    COMPONENT or2  
        GENERIC(trise : delay;  
                tfall : delay);
```

Module 11- Structural VHDL Tutorial

```
    PORT(a : IN level;
          b : IN level;
          c : OUT level);
END COMPONENT;

COMPONENT inv
  GENERIC(trise : delay;
          tfall : delay);
  PORT(a : IN level;
        b : OUT level);
END COMPONENT;

FOR ALL : and2 USE ENTITY gate_lib.and2(behav);
FOR ALL : or2  USE ENTITY gate_lib.or2(behav);
FOR ALL : inv  USE ENTITY gate_lib.inv(behav);

SIGNAL and_out : level; --signal for output of and gate
SIGNAL or_out  : level; --signal for output of or gate

BEGIN

    AND_1 : and2 GENERIC MAP(trise => trise, tfall => tfall)
            PORT MAP(a => a, b => b, c => and_out);

    OR_1  : or2  GENERIC MAP(trise => trise, tfall => tfall)
            PORT MAP(a => and_out, b => c, c => or_out);

    INV_1 : inv  GENERIC MAP(trise => trise, tfall => tfall)
            PORT MAP(a => or_out, b => d);

END structural;
```

2.2. Before the **aoi2_str.vhdl** model will compile correctly, some configuration of the libraries is needed. Notice the library clause "LIBRARY gate_lib;" at the beginning of the **aoi2_str.vhdl** file. This tells the compiler that a library called "gate_lib" is available for use.

2.3. Notice that the **and2**, **or2**, and **inv** gates are bound to the components in the "gate_lib" library. In order for the compiler to use these gates (which were compiled during the Module 10 lab), we must create a *library mapping*.

2.4. Library mapping commands are specific to the VHDL tools being used. Consult your compiler's documentation for more information on library mappings. Be sure that the logical library name "gate_lib" refers to the working library created in Module 10. Typical names for working libraries are

"work" and "worklib".

2.5. Once the library mapping is correctly created, compile the **aoi2_str.vhdl** model. The model should compile without any errors.

2.6. A test bench for the model is provided in the file **aoi2_str_test.vhdl**. Open and compile this test bench file.

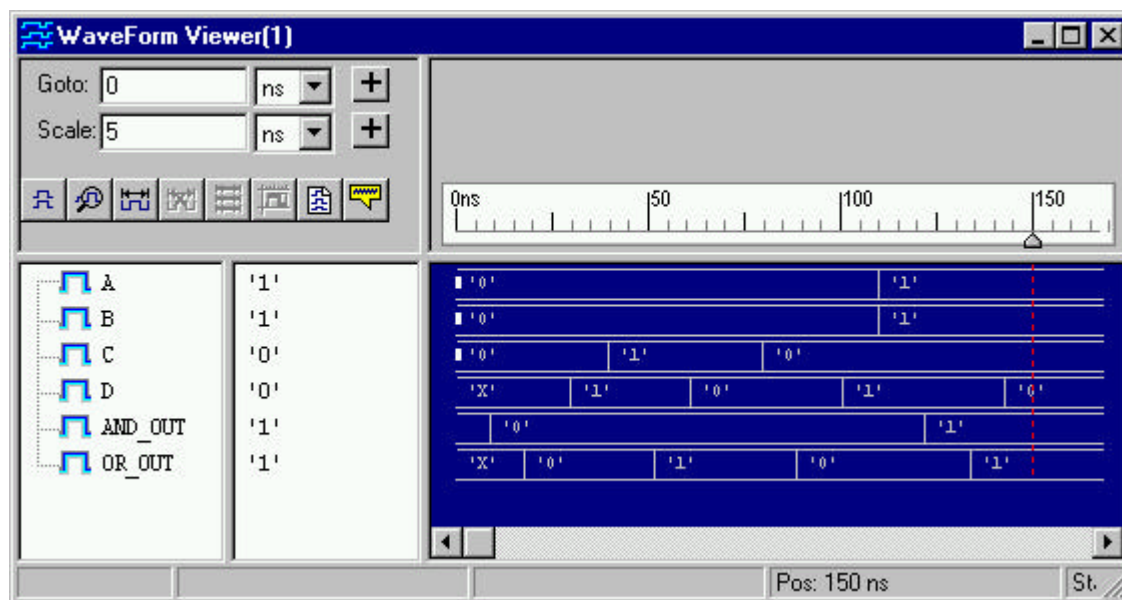
3. Simulate the compiled code

3.1. Start the VHDL simulator. Remember to select an entity or architecture as the design root, if required by your simulator.

3.2. Before adding any signals to the waveform window, note the hierarchy of signal names in the model. The test bench, which is the highest level of hierarchy, has signals are "A_SIG", "B_SIG", "C_SIG", and "D_SIG". The **aoi2_str** structural entity has the input signals "A", "B", and "C"; output signal "D"; and internal signals "AND_OUT" and "OR_OUT". Lower levels of hierarchy are also present.

3.3. Open a waveform window, then add the signals from the **aoi2_str** component. As listed above, these are "A", "B", "C", "D", "AND_OUT", and "OR_OUT".

3.4. Run the simulator for sufficient time to fully exercise the circuit (at least 150 nanoseconds in this example). The signal waveforms should look something like this:



4. Examine and compile the 8-bit shift register model

Copyright ©1995-1999 SCRA

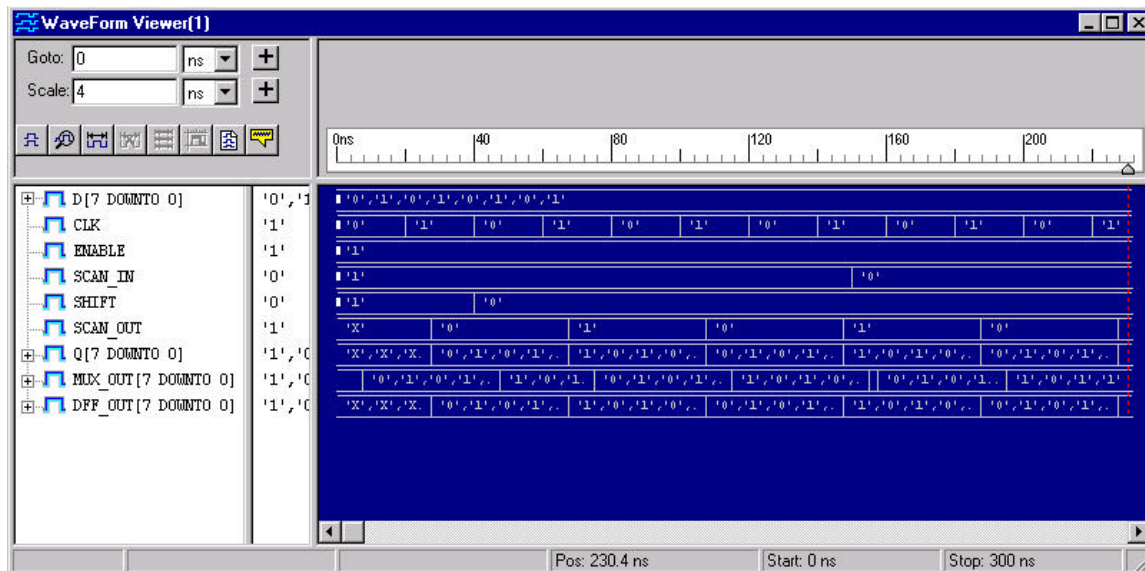
See first page for copyright notice,

Distribution restrictions and disclaimer

- 4.1. Open the file **shift_reg8_str.vhdl** using a text editor or a VHDL editing environment.
- 4.2. Compile the **shift_reg8_str.vhdl** model.
- 4.3. A test bench for the model is provided in the file **shift_reg8_str_test.vhdl**. Open and compile this test bench file.

5. Simulate the compiled code

- 5.1. Start the VHDL simulator.
- 5.2. Open a waveform window, then add all of the signals associated with the **shift_reg8_str** structural model. Run the simulation. The resulting window should look something like this:



6. Examine and compile the unsigned 8-bit multiplier model

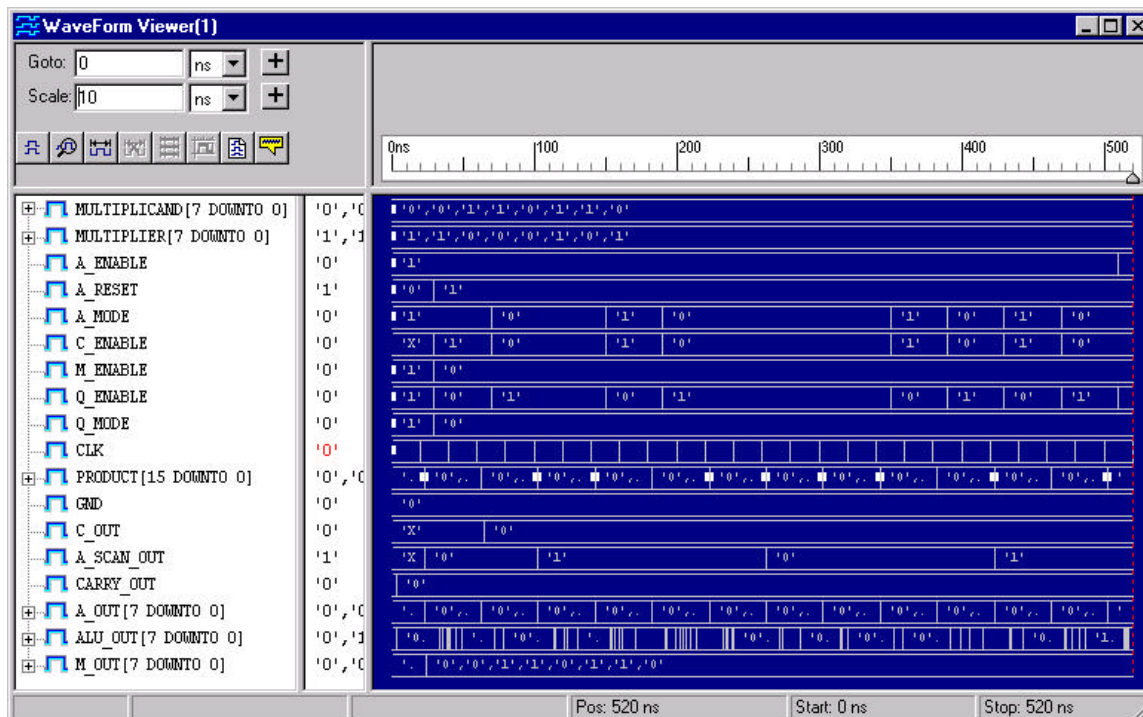
- 6.1. Open the VHDL source files:
reg8_str.vhdl
ha_str.vhdl
fa_str.vhdl
alu_str.vhdl
mult_datapath_str.vhdl
- 6.2. Compile the VHDL files.
- 6.3. A test bench for the multiplier model is provided in the file **mult_datapath_str_test.vhdl**. This test bench will perform an example

multiplication sequence. Open and compile this test bench file.

7. Simulate the compiled code

7.1. Start the VHDL simulator.

7.2. Open a waveform window, then add all of the signals associated with the **mult_datapath_str** structural model. Run the simulation for an appropriate amount of time (at least 500 nanoseconds for this example). The resulting window should look something like this:



Notice that during initialization, the **ALU** outputs all "0"s which are loaded into the **A** register to clear it. The **C** register is cleared, the multiplier is loaded into the **Q** register, and the multiplicand is loaded into the **M** register. the simulation then proceeds by performing **ADD** and **SHIFT** operations based on the status of the **Q₀** bit.

Module 11 Exercise

Assignment:

Using a simple GENERATE statement, create an 8 bit 2 to 1 multiplexor from a structure of **mux2** components. Compile and simulate the design to ensure correct operation. Do the same for an 8 bit 4 to 1 multiplexor using **mux4** components.

Using a GENERATE statement with an IF clause, create an 8 bit arithmetic shift register from **dff** and **mux2** components. Recall that in an arithmetic shift:

```
Q(7) <= D(7),  
Q(6 downto 0) <= D(7 downto 1), and  
Scan_out <= D(0).
```

Compile and simulate the design to ensure correct operation.

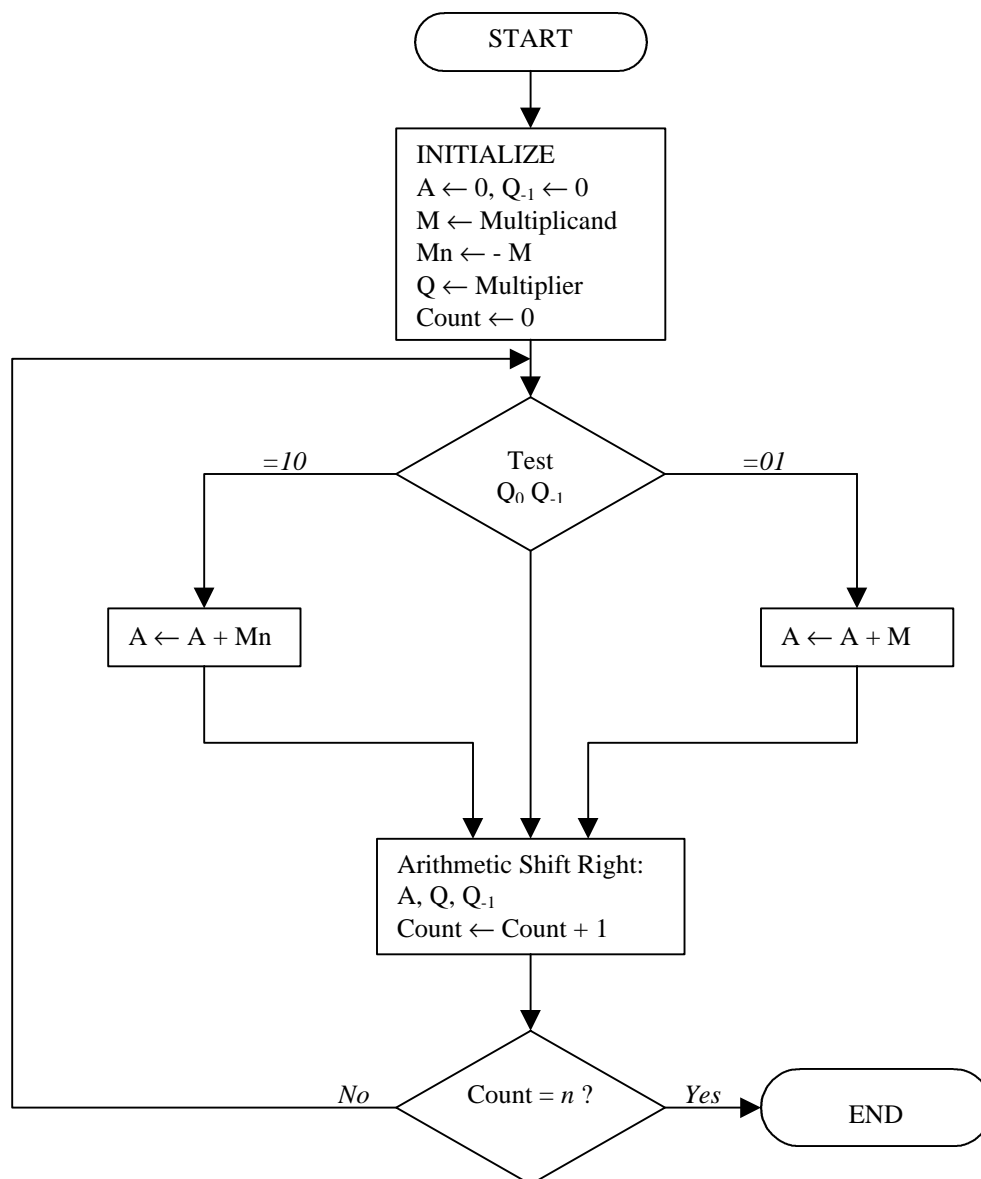
Using the components constructed above as well as the **reg8_str**, **shift_reg8_str**, **alu_str**, **mux2**, and **dff**, create an RTL datapath for a Booth's algorithm multiplier. A Booth's multiplier functions very similarly to the unsigned multiplier with the exceptions that it uses an arithmetic shift of the A and Q registers, the Q(0) bit is shifted out into a Q(-1) register, and the control of the shift/add operation is based on the Q(0) and Q(-1) bits. Finally instead of just adding M to A and shifting, the Booth's multiplier either adds or subtracts (adds the two's complement of) M with A. The flow chart in the following pages fully outlines Booth's algorithm for two's complement multiplication.

An example of 2-complement multiplication using Booth's Algorithm is shown below:

M = 11111011

Mn = 00000101

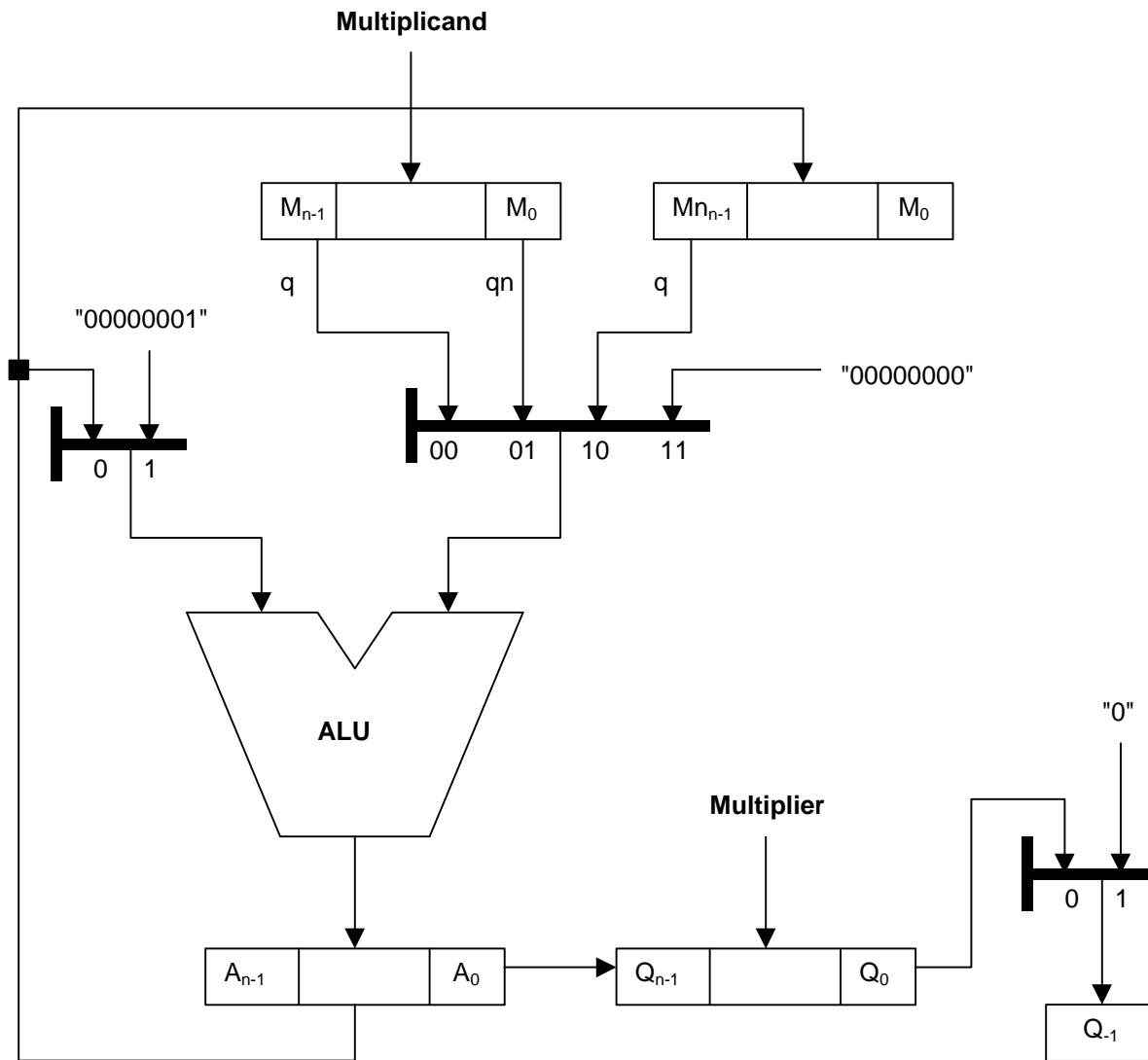
A	Q	Q ₋₁		
00000000	00000011	0	Initialization	
00000101	00000011	0	A ← A + Mn	First Cycle
00000010	10000001	1	Arithmetic Shift	
00000001	01000000	1	Arithmetic Shift	Second Cycle
11111100	01000000	1	A ← A + M	Third Cycle
11111110	00100000	0	Arithmetic Shift	
11111111	00010000	0	Arithmetic Shift	Fourth Cycle
11111111	10001000	0	Arithmetic Shift	Fifth Cycle
11111111	11000100	0	Arithmetic Shift	Sixth Cycle
11111111	11100010	0	Arithmetic Shift	Seventh Cycle
11111111	11110001	0	Arithmetic Shift	Eighth Cycle



Flowchart for Booth's Algorithm 2's-Complement Multiplication

Hint - A datapath very similar to the one used for the unsigned multiplier can be used. Instead of a single register entering the ALU for the Multiplicand, two registers can be used, one for the multiplicand (M) and another for the two's complement of M (Mn) entering the ALU through a 4 to 1 multiplexor. The two's complement of M can be generated by adding the inverse of M (available as the Qn outputs of the M register) to "00000001" using the ALU.

If you need help, an example datapath that can perform Booth's Algorithm multiplication is shown below.



Example Datapath for Booth's Algorithm 2's-Complement Multiplication