

Module 12 - Behavioral VHDL Tutorial and Exercises

Copyright 1995-1999 SCRA

All rights reserved. This information is copyrighted by the SCPA, 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.

Module 12 - Behavioral VHDL Lab Tutorial

Module 12 Tutorial

1 Getting started

- 1.1 Create a directory for the Module 12 lab material, e.g:

```
mkdir m12_ex  
cd m12_ex
```

- 1.2 Copy the source files for the VHDL that you will compile and simulate from the source directory, e.g:

```
cp $VHDL_SRC/mult_controller_behav.vhdl .  
cp $VHDL_SRC/full_mult_str.vhdl .
```

The lab is also available in the m12_lab.tar file.

This includes a behavioral description of the control unit for the multiplier and a structural description that instantiates the datapath and the control unit together.

- 1.3 There is also a command file that has been created that will generate the forces necessary to drive the full multiplier through an example multiplication sequence. Copy it over now, e.g:

```
cp $VHDL_SRC/m12_ex/full_mult.do .
```

- 1.4 The Mentor Graphics QuickVHDL simulator needs a work directory for the compiled VHDL files. Create this directory with the appropriate command for the version you are running, e.g:

```
qhlib work
```

- 1.5 The full multiplier uses the datapath constructed for Module 11 as well as the package from the Module 10 lab. You can access those compiled descriptions by mapping logical libraries to their actual locations in the file system. The exact command to do this is specific to the VHDL tools being used. For many versions of the Mentor Graphics QuickVHDL simulator, it is done using the following command:

```
qhmap gate_lib ../m10_ex/work  
qhmap mult_lib ../m11_ex/work
```

Module 12 - Behavioral VHDL Lab Tutorial

2 Examine and compile the state machine code for this lab

- 2.1 Open the file **mult_controller_behav.vhdl** file using a text editor or a VHDL editing environment. You will see the following VHDL description:

```
-----
-- Eight Bit Multiplier Controller Behavioral Example  --
-- RASSP E&F Module # 12 Behavioral VHDL              --
-- Robert Klenke UVa 28 May 1996                      --
-----

LIBRARY gate_lib;
USE gate_lib.resources.all;

ENTITY mult_controller_behav IS
  PORT(reset  : IN level; -- global reset signal
        start  : IN level; -- input to indicate start of process
        q0     : IN level; -- q0 ,input from data path
        clk    : IN level; -- clock signal
        a_enable : OUT level; -- clock enable for A register
        a_reset : OUT level; -- Reset control for A register
        a_mode  : OUT level; -- Shift or load mode for A
        c_enable : OUT level; -- clock enable for c register
        m_enable : OUT level; -- clock enable for M register
        q_enable : OUT level; -- clock enable for Q register
        q_mode  : OUT level; -- Shift or load mode for Q
END mult_controller_behav;

ARCHITECTURE state_machine OF mult_controller_behav IS

  SUBTYPE count_integer IS integer RANGE 0 to 8;
  TYPE states IS (idle,initialize,test,shift,add);
  SIGNAL present_state : states := idle;
  SIGNAL next_state    : states := idle;
  SIGNAL present_count : count_integer := 0;
  SIGNAL next_count    : count_integer := 0;

  BEGIN

    CLKD : PROCESS(clk,reset)
    BEGIN
      IF(reset = '1') THEN
        present_state <= idle;
        present_count <= 0;
      ELSIF(clk'EVENT AND clk = '1' AND clk'LAST_VALUE = '0') THEN
        present_state <= next_state;
        present_count <= next_count;
      END IF;
    END PROCESS CLKD;

    STATE_TRANS : PROCESS(present_state,present_count,start,q0)
    BEGIN
      next_state <= present_state; -- default case
      next_count <= present_count; -- default case
    END PROCESS STATE_TRANS;
  END ARCHITECTURE state_machine;
```

Module 12 - Behavioral VHDL Lab Tutorial

```
CASE present_state IS
  WHEN idle =>
    IF(start = '1') THEN
      next_state <= initialize;
    ELSE
      next_state <= idle;
    END IF;
    next_count <= present_count;
  WHEN initialize =>
    next_state <= test;
    next_count <= present_count;
  WHEN test =>
    IF(present_count < 8) THEN
      IF(q0 = '0') THEN
        next_state <= shift;
      ELSE
        next_state <= add;
      END IF;
    ELSE
      next_state <= idle;
    END IF;
    next_count <= present_count;
  WHEN add =>
    next_state <= shift;
    next_count <= present_count;
  WHEN shift =>
    next_state <= test;
    next_count <= present_count + 1;
  WHEN OTHERS =>
    next_state <= idle;
    next_count <= present_count;
END CASE;
END PROCESS STATE_TRANS;
```

```
OUTPUT : PROCESS(present_state)
BEGIN
  CASE present_state IS
    WHEN idle =>
      a_enable <= '0';
      a_reset  <= '1';
      a_mode   <= '1';
      c_enable <= '0';
      m_enable <= '0';
      q_enable <= '0';
      q_mode   <= '1';
    WHEN initialize =>
      a_enable <= '1';
      a_reset  <= '0';
      a_mode   <= '1';
      c_enable <= '0';
      m_enable <= '1';
      q_enable <= '1';
      q_mode   <= '1';
    WHEN test =>
      a_enable <= '0';
      a_reset  <= '1';
```

Module 12 - Behavioral VHDL Lab Tutorial

```
a_mode <= '1';
c_enable <= '0';
m_enable <= '0';
q_enable <= '0';
q_mode <= '1';
WHEN add =>
  a_enable <= '1';
  a_reset <= '1';
  a_mode <= '1';
  c_enable <= '1';
  m_enable <= '0';
  q_enable <= '0';
  q_mode <= '0';
WHEN shift =>
  a_enable <= '1';
  a_reset <= '1';
  a_mode <= '0';
  c_enable <= '0';
  m_enable <= '0';
  q_enable <= '1';
  q_mode <= '0';
WHEN OTHERS =>
  a_enable <= '0';
  a_reset <= '1';
  a_mode <= '1';
  c_enable <= '0';
  m_enable <= '0';
  q_enable <= '0';
  q_mode <= '1';
END CASE;
END PROCESS OUTPUT;

END state_machine;
END resources;
```

- 2.2 Compile the VHDL code, e.g:

```
qvhcom mult_controller_behav.vhdl
```

- 2.3 Compile the vhdl code for the full multiplier structural description, e.g:

```
qvhcom full_mult_str.vhdl
```

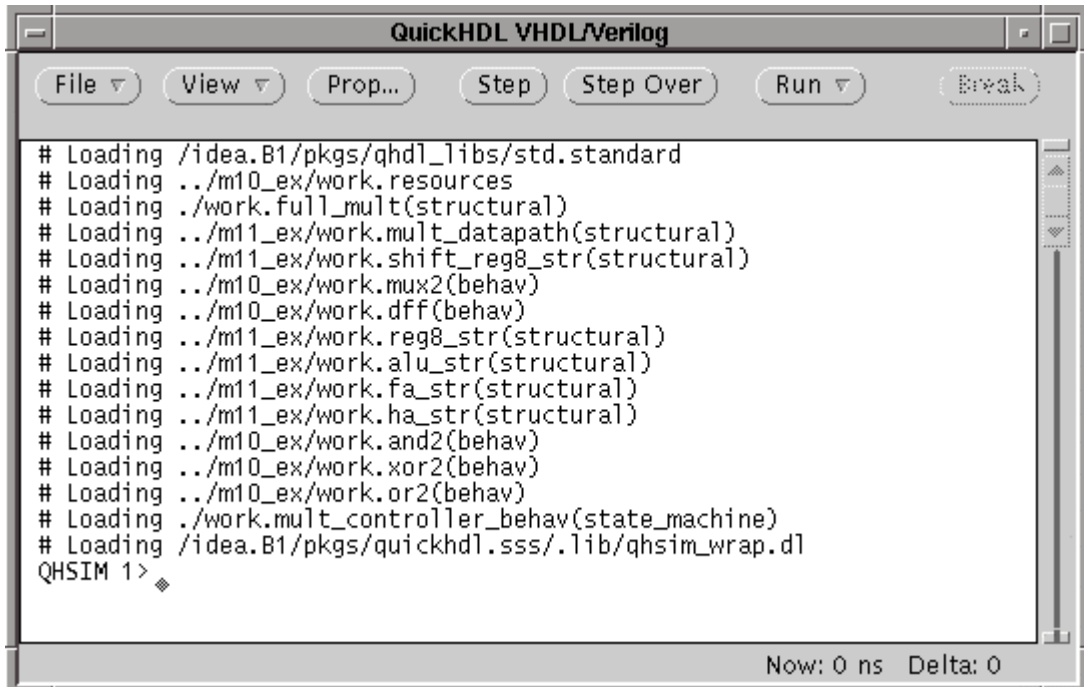
3 Simulate the compiled code

- 3.1 Start up the Mentor Graphics VHDL simulator. The specific command may vary depending on the version you are using, e.g.:

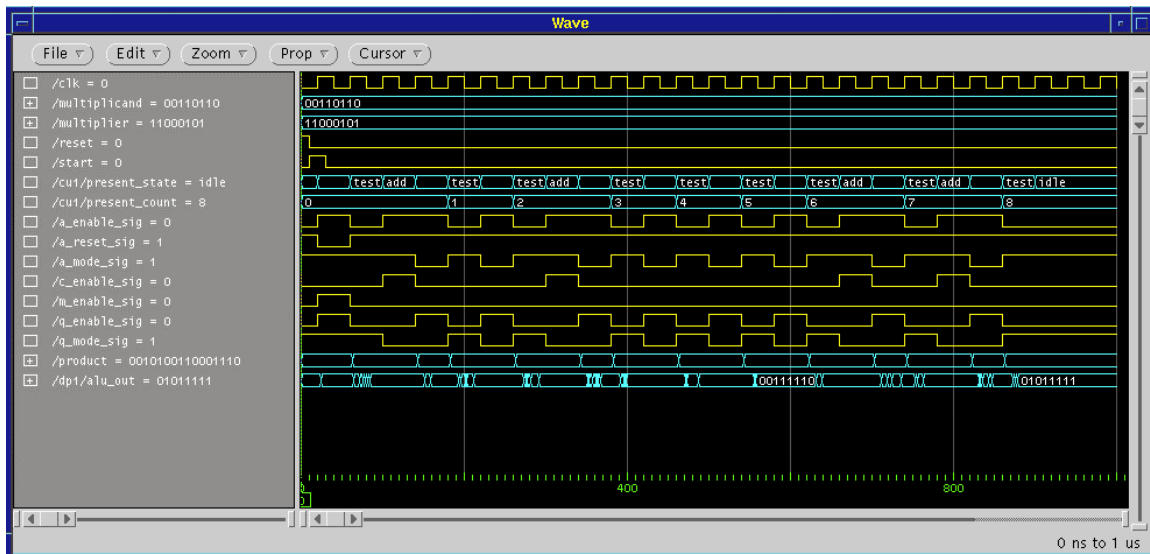
```
qhsim full_mult
```

Note that all of the lower level components used in the multiplier are loaded by the simulator. You should see a window similar to the following:

Module 12 - Behavioral VHDL Lab Tutorial



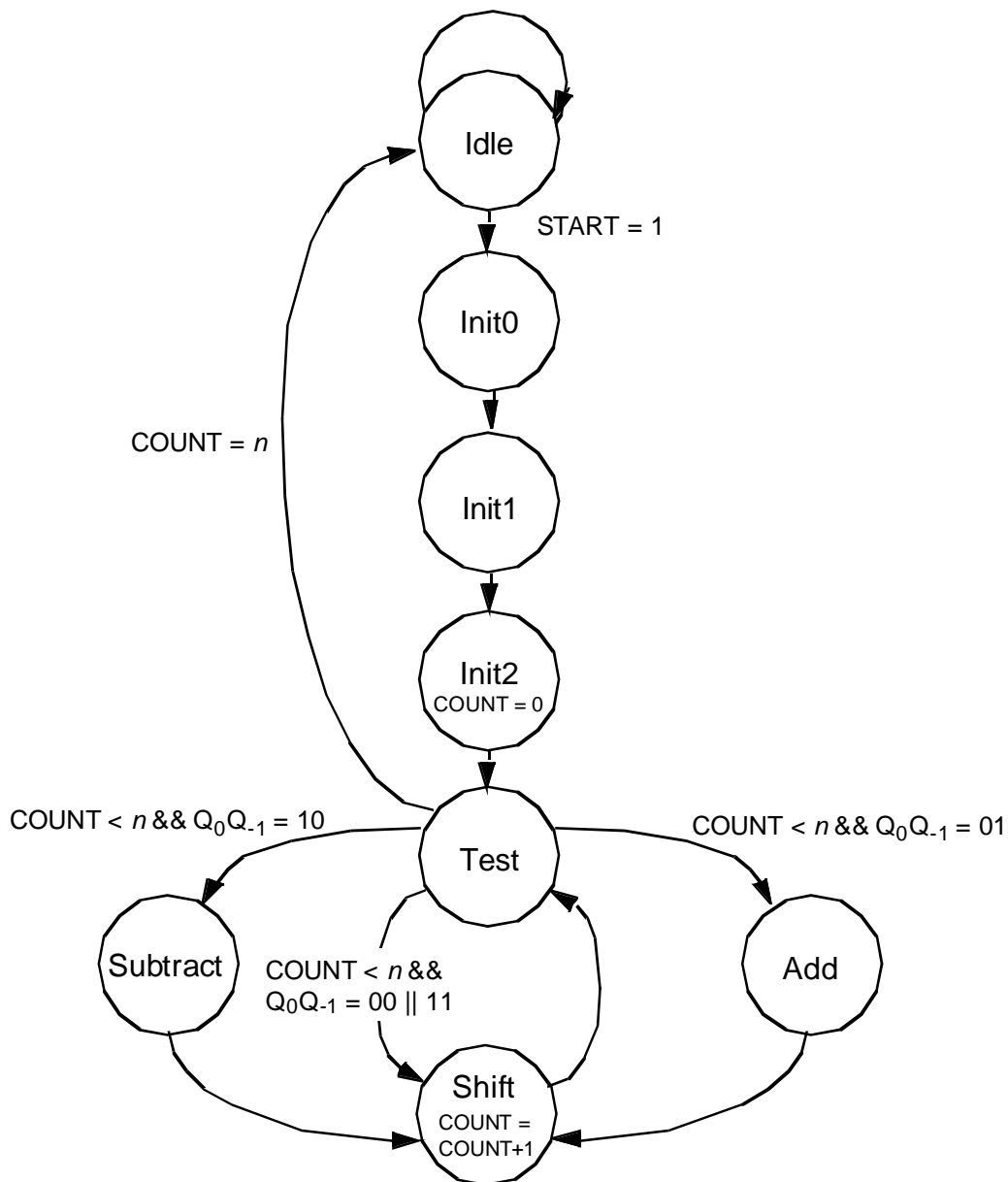
- 3.2 Using the *dofile* mechanism of the Mentor Graphics VHDL simulator, execute command file **full_mult.do**. The result should be a wave window that is similar to the following:



Module 12 - Behavioral VHDL Lab Tutorial

Assignment:

Develop a VHDL behavioral description of a control unit for the Booth's algorithm datapath you designed for Module 11. Note that your datapath will probably require several initialization states before it is ready to start the main loop of the algorithm. A generalized state diagram for the control unit with the outputs eliminated for clarity is shown below:



Develop a structural description for the full Booth's algorithm multiplier that instantiates the datapath and control unit together. Compile and simulate your design.

Module 12 Tutorial Continued (optional if time permits)

4 Copy, Compile and run the C source code for the quicksort example

- 4.1 You will begin this example by compiling and running a version of the quicksort routine coded in C. Copy that source code now from the source directory, e.g:

```
cp $VHDL_SRC/qsort.c .
```

- 4.2 Examine the qsort.c file using a text editor. You will see the following file:

```
/******  
/*  Quicksort Algorithm Behavioral Example - C Test Code  */  
/*      RASSP E&F Module # 12 Behavioral VHDL      */  
/*      Robert Klenke UVa 28 May 1996      */  
/******  
#include <stdio.h>  
  
/* use static allocation */  
#define MAXARRAY 100  
  
void quicksort(a,l,r)  
int *a;  
int l;  
int r;  
{  
    int v, t;  
    int i, j;  
  
    if(r>l) {  
        v = a[r];  
        i = l-1;  
        j = r;  
        do {  
            do {  
                i = i + 1;  
            } while(a[i]<v);  
            do {  
                j = j - 1;  
            } while(a[j]>v);  
            t = a[i];  
            a[i] = a[j];  
            a[j] = t;  
        } while(j>i);  
        a[j] = a[i];  
        a[i] = a[r];  
    }  
}
```


Module 12 - Behavioral VHDL Lab Tutorial

```
a[r] = t;
quicksort(a,l,i-1);
quicksort(a,i+1,r);
}
}
main(argc,argv)
int argc;
char *argv[];
{
    int nelements, i, tempint, temppointer, min, iarray[MAXARRAY];
    FILE *in_fd, *out_fd;

    if(argc < 2) {
        fprintf(stderr,"Usage:qsort <infile> <outfile>\n");
        exit(0);
    }

    if ((in_fd = fopen(argv[1], "r")) == 0) {
        fprintf(stderr, "Usage: qsort <input_file> <output_file>\n");
        exit(0);
    }
    if ((out_fd = fopen(argv[2], "w")) == 0) {
        fprintf(stderr, "Usage: qsort <input_file> <output_file>\n");
        exit(0);
    }

    /* read in file and set number of elements */
    nelements = 0;
    while((fscanf(in_fd,"%d\n",&iarray[nelements]) != EOF)
        && (nelements < MAXARRAY))
        nelements++;

    /* find mininum element and place in element zero for sentinel */
    tempint = iarray[0];
    temppointer = 0;
    for(i=1;i<nelements;i++)
        if(iarray[i] < tempint) {
            tempint = iarray[i];
            temppointer = i;
        }
    if(temppointer != 0) {
        iarray[temppointer] = iarray[0];
        iarray[0] = tempint;
    }

    /* do the quicksort! */
    quicksort(iarray,0,nelements-1);

    /* write out results */
    for(i=0;i<nelements;i++)
        fprintf(out_fd,"%d\n",iarray[i]);
}
```

Module 12 - Behavioral VHDL Lab Tutorial

- 4.3 Compile the C code using an available C compiler, e.g.:

```
cc -o qsort qsort.c
```

- 4.4 Create an unsorted input file named **test.in** using a text editor and add the values shown below:

```
123
40
56
987
9
928374
89273
743
823
52
7
127638
83
2
187
1879
897234
3483
84502
284734
2
8729
947
```

- 4.5 Run the quicksort routine on the **test.in** file:

```
qsort test.in test.out
```

The result will be a **test.out** file like this one:

```
2
2
7
9
40
52
56
83
123
187
743
```

Module 12 - Behavioral VHDL Lab Tutorial

823
947
987
1879
3483
8729
84502
89273
127638
284734
897234
928374

5 Copy, Compile and simulate the VHDL source code for the quicksort example

- 5.1 Copy the source files for the VHDL that you will compile and simulate from the source directory, e.g.:

```
cp $VHDL_SRC/qsrt_pkg.vhdl .
```

```
cp $VHDL_SRC/qsrt.vhdl
```

- 5.2 Examine the quicksort package file **qsrt_pkg.vhdl** and notice the similarity between the quicksort routine in VHDL and the same routine in C. The major difference is the way in which the loops are constructed because of the differences in the available loop constructs in the two languages.

Notice that the entity description has two generics for the input and output file names, but no ports. Also note that the architecture is coded as a single process statement that executes one time at the beginning of the simulation and then stops at a WAIT statement at the end of the process. The VHDL code for this process is also very similar to the corresponding C code for the main program.

- 5.3 Compile the VHDL files, e.g.:

```
qvhcom qsrt_pkg.vhdl
```

```
qvhcom -93 qsrt.vhdl
```

Note that the -93 option is used in some versions of Mentor Graphics QuickVHDL compiler to allow IEEE 1076-93 VHDL constructs such as the file_open routine.

- 5.4 Run the VHDL quicksort routine, e.g.:

```
qhsim -c -ginfile=test.in -goutfile=test.out2 qsrt
```

Module 12 - Behavioral VHDL Lab Tutorial

The -c option is used in many versions of the Mentor Graphics QuickVHDL simulator to run in the command line mode (no GUI), and the -ginfile=test.in and -goutfile=test.out2 options are used to provide the **test.in** and **test.out2** files for the top level generics of the qsort entity.

When the QuickVHDL command line prompt appears, run the simulation and then exit the simulator, e.g.:

```
QHSIM 1> run
```

```
QHSIM 2> quit
```

- 5.5 Verify that the **test.out2** file generated by the VHDL quicksort routine is the same as the test.out file produced by the C implementation. You can look at the files using a text editor, or you can use the Unix *diff* program. *diff* compares two files and lists any differences between them:

```
diff test.out test.out2
```

No output from this command indicates that the files were the same.