Module 24

Example Verilog Testbench

cadence

This module examines a testbench application that uses pseudocode.

Module Objective

In this module, you:

- Create a testbench that executes pseudo-code

Topics

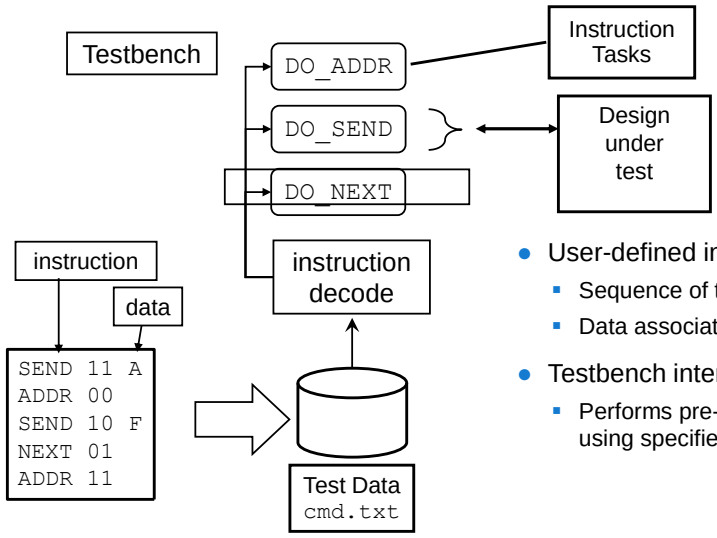
- A script driven testbench in Verilog-1995
- A script driven testbench in Veirlog-2001

cadence

Your objective is to create a testbench that executes pseudo-code.

This module presents a testbench that executes pseudo-code, and then sidles off to take another look at the various simulation outputs.

Script-Driven Testbench



- User-defined instructions specify:
  - Sequence of test operations.
  - Data associated with operations.
- Testbench interprets instructions:
  - Performs pre-defined operations using specified data.

It is possible to define a set of instructions by which the operation of a testbench can be controlled. This collection of user-defined instructions is sometimes called pseudocode, and a testbench usually reads these instructions from a text file. This type of testbench is also know as a script-driven testbench.

The testbench is written to interpret the text of each instruction and then perform that operation. We will see an example of this on the following slides.

The benefits of a script-driven testbench are many, as in these examples:

- Higher level of abstraction for creating test sequences and data.
- Sequence of tests and data can be easily changed by editing the external file.
- A text file is human readable and is easier to understand than in-line, looped, or arrayed stimulus.
- Tests and testbench are modular – additional tests can be easily added.

One issue is that user-defined instructions are more likely to be written by hand, and therefore contain errors, than a file of stimulus data created by, for example, a graphics tool. Therefore, our testbench needs to carefully check the instruction data it is trying to read.

Script-Driven Testbench Verilog-1995

```
...
 reg [15:0] cmdarr [0:6];
 reg [15:0] cmd;
 integer i;

 parameter SEND = 4'h0, ADDR = 4'h1, NEXT = 4'h2;

 initial begin
   $readmemh("data.txt", cmdarr);
   for (i=0;i<7;i=i+1)
     case (cmd[15:12])
       SEND:do_send(cmd[11:4], cmd[3:0]);
       ADDR:do_addr(cmd[11:4]);
       NEXT:do_next(cmd[11:4]);
       default:$display("unknown command %h",cmd);
     endcase
   end
...
```

Array for test data

Instruction coding

Read data into array

Decode instruction

Call tasks with specified data

data.txt

```
0_11_A //SEND 11 A
1_00_0 //ADDR 00
0_10_F //SEND 10 F
2_01_0 //NEXT 01
1_11_0 //ADDR 11
...
```

- You encode instructions in hex.
- $readmemh reads instructions into array.
- Testbench decodes array instructions into control and data.
  - Testbench is simple.
  - Test data is not (machine code).

495

cadence

In this example, there are three forms of instruction allowed in the input text file: *ADDR XX*; *NEXT XX* and *SEND XX Y* where *XX* is a hex address and *Y* is a hex number.

In Verilog 1995, the usual way to read file data is to use $*readmemb* or $*readmemh* to read every line of test data into a suitably sized array. This requires you to encode test instructions and data into hex or binary numbers.

Once the test data is read into the array, you can then index each line of data ,decode the instruction bits and execute the required operations.

With this approach, the testbench is simple, but the test data is more difficult to read, write and maintain.

Script-Driven Testbench Verilog-2001

```
...
reg [4*8-1:0] cmd;
reg [1:0] addr;
reg [3:0] data;
integer fid, c1, i;

initial begin
  fid = $fopen("cmd.txt", "r");
  while(!$feof(fid)) begin
    c1 = $fscanf(fid, "%4c %2h", cmd, addr);
    case (cmd)
      "SEND":begin
             c1 = $fscanf(fid, " %h", data);
             do_send(addr, data);
             end
      "ADDR":do_addr(addr);
      "NEXT":do_next(addr);
      default:$display("unknown command %s", cmd);
    endcase
    c1 = $fscanf(fid, "\n");
  end
 end
```

Open file in read mode

Detect EOF

Read 4 character command, a space, then 2 digit hex address

If SEND, read single digit hex data

Read end of line

cmd.txt

```
SEND 11 A
ADDR 00
SEND 10 F
NEXT 01
ADDR 11
...
```

- Verilog-2001 can read formatted data from files:
  - Test data is more readable (assembly language).

cadence

Verilog 2001 enhances file IO with a series of C-like system tasks and functions. This allows formatted test data to be read. The example above works as follows:

- $fopen opens the cmd.txt file in read mode.
- $feof detects the End Of File (EOF)
- $fscanf reads formatted data from the file. The format string "%4c %2h" reads 4 characters followed by a space, then 2 hex digits. $fscanf assigns data matching the format specifiers to the cmd and addr variables. cmd is a concatenation of 4 bytes representing the ASCII encoding for the command string.
- The case statement decodes the command and calls the tasks with the required address data. If the command is SEND, we must read the data from the file. $fscanf reads a space, followed by a single hex digit which is assigned to data.
- The case default statement handles unknown commands.
- After the case, another $fscanf reads the end of line (“\n”) character to advance the file read to the next line. Note that assignment variables in the $fscanf call are optional.

Module Summary

You can now create a testbench that is driven from a script and is also called a pseudocode testbench.

In this module, you learned how to write:

- A script-driven testbench using Verilog 1995 *readmemh* system task.
- A script-driven testbench using Verilog 2001 *fopen* system task.

cadence

This module examined a testbench application using pseudocode, and then sidled off to take another look at various simulation outputs.

Labs

Lab 24-1 Developing a Script-Driven Testbench Using Verilog 1995

- You write a testbench using the Verilog 95 $*readmemh* system task.

Lab 24-2 Developing a Script-Driven Testbench Using Verilog 2001

- You write another testbench using the Verilog 2001 $fopen system task.

cadence

Try to write two testbenches. First, write a Verilog 1995 vreadmem_test.v that uses the system task $readmemh for getting the required commands from a data.txt into the procedure call within the testbench.

For the second testbench, write a Verilog 2001 vfopen_test.v that uses the system task $fopen to get the file containing the cmds to be opened by the procedure call within the testbench.