Computation II: embedded system design (5EIB0)

Description Submission view

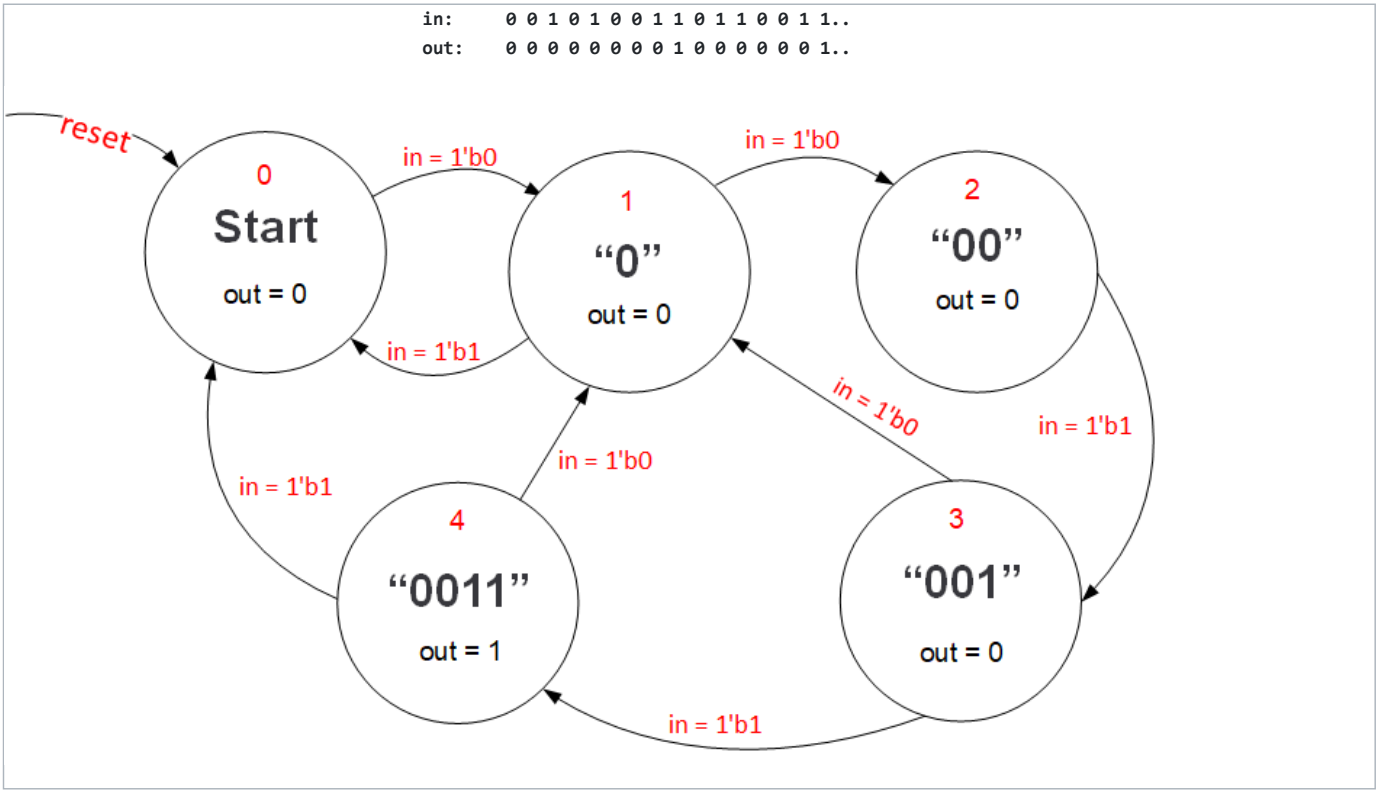
Available from: Wednesday, 3 July 2024, 6:00 PM
Due date: Wednesday, 3 July 2024, 9:00 PM
Requested files: seq_top_tb.v (Download)
Type of work: Individual work

Introduction

In this exercise, you will create the Verilog testbench for a Finite State Machine (FSM) which detects a finite string pattern including overlap. The FSM has the following properties:

- one input (**in**) and one output (**out**)
- one push button (**next**) to indicate a valid input
- output is asserted whenever the input sequence **... 0011 ...** is observed.

Example input/output behavior:



The diagram above illustrates the transitions of the FSM. It shows a Moore Machine where the output is defined by the state. The machine has five states. The state numbers are marked in red.

Note: All state transitions should happen when next rises from 0 to 1.

Assignment

Write a FSM testbench named **seq_top_tb.** It will drive the module named **seq_top** which takes **input signals clk, reset, in, next** and produces **output signals out, state_display**. Note that the FSM is reset when signal **reset** is high. Assume all input and output signals to be 1 bit wide except state_display which is 3 bits wide and little endian. You have to write the testbench which first initializes values and applies a reset and then tests the given finite string pattern recognizer for the following input sequence:

```
0 => 1 => 0 => 1 => 1 => 0 => 0 => 0 => 1 => 1 => 0
```

Print the following after each input stimuli:

```
$display("state %x, out %x", state_display, out);
```

Use the following template for writing the testbench:

```
`timescale 1ns / 1ps

module seq_top_tb();

    <<DECLARE SIGNALS TO DRIVE MODULE seq_top>>

        seq_top seq_top_inst(
                 .clk(clk),
                 .reset(reset),
              .next(next),
              .in(in),
                  .state_display(state_display),
                  .out(out));

    <<DECLARE CLOCK>>

        initial begin
        $dumpfile("dut.vcd");
        $dumpvars(0, seq_top_inst);

        <<DECLARE SIGNALS TO INITIALISE clk AND next VALUES>>


        <<DECLARE SIGNALS TO RESET seq_top>>

        $display("state %x, out %x", state_display, out);
```

```

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 1 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 1 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 1 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 1 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 1 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);

        <<INPUT 0 TO in AND TOGGLE next>>

        $display("state %x, out %x", state_display, out);
        #100
 $finish;
```

```
end
endmodule
```

STRICT NOTE: Just change the portions marked in red in the tesbench template. Also, make sure to deassert **next** (**next** = 1b'0) after every state transition.

Debug

Click on the symbol marked below to see the waveforms produced by your design. Please note that if your code has an error that prevents it being simulated it will not produce any waveforms.

Requested files

seq_top_tb.v

```
`timescale 1ns / 1ps

module seq_top_tb();

    <<DECLARE SIGNALS TO DRIVE MODULE seq_top>>

    seq_top seq_top_inst(
         .clk(clk),
         .reset(reset),
         .next(next),
         .in(in),
         .state_display(state_display),
         .out(out));

    <<DECLARE CLOCK>>

    initial begin
    $dumpfile("dut.vcd");
    $dumpvars(0, seq_top_inst);

    <<DECLARE SIGNALS TO INITIALISE clk AND next VALUES>>


    <<DECLARE SIGNALS TO RESET seq_top>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 1 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 1 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 1 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 1 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 1 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);

    <<INPUT 0 TO in AND TOGGLE next>>

    $display("state %x, out %x", state_display, out);
    #100
 $finish;
end
endmodule
```

VPL