Nang Thiri Wutyi
20113
9/30/2024

Verilog Lab HW 2

1. ReductionOperators

```
module ReductionOperators();
   initial begin
      // Bit Wise AND reduction
      $display (" & 4'b1001 = %b", (&4'b1001));
      $display (" & 4'bx111 = %b", (&4'bx111));
      $display (" & 4'bz111 = %b", (&4'bz111));

      // Bit Wise NAND reduction
      $display (" ~& 4'b1001 = %b", (~&4'b1001));
      $display (" ~& 4'bx001 = %b", (~&4'bx001));
      $display (" ~& 4'bz001 = %b", (~&4'bz001));

      // Bit Wise OR reduction
      $display (" | 4'b1001 = %b", (|4'b1001));
      $display (" | 4'bx000 = %b", (|4'bx000));
      $display (" | 4'bz000 = %b", (|4'bz000));

      #10 $finish;
   end
endmodule
```

Results

```
[2024-09-28 10:24:36 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
 & 4'b1001 = 0
 & 4'bx111 = x
 & 4'bz111 = x
 ~& 4'b1001 = 1
 ~& 4'bx001 = 1
 ~& 4'bz001 = 1
 | 4'b1001 = 1
 | 4'bx000 = x
 | 4'bz000 = x
design.sv:18: $finish called at 10 (1s)
Done
```

ShiftOperators

```
module ShiftOperators();
initial begin
```

```
        // Left Shift
        $display (" 4'b1001 << 1 = %b", (4'b1001 << 1));
        $display (" 4'b10x1 << 1 = %b", (4'b10x1 << 1));
        $display (" 4'b10z1 << 1 = %b", (4'b10z1 << 1));
        // Right Shift
        $display (" 4'b1001 >> 1 = %b", (4'b1001 >> 1));
        $display (" 4'b10x1 >> 1 = %b", (4'b10x1 >> 1));
        $display (" 4'b10z1 >> 1 = %b", (4'b10z1 >> 1));
        #10 $finish;
end
endmodule
```

Results

```
[2024-09-28 10:27:33 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
 4'b1001 << 1 = 0010
 4'b10x1 << 1 = 0x10
 4'b10z1 << 1 = 0z10
 4'b1001 >> 1 = 0100
 4'b10x1 >> 1 = 010x
 4'b10z1 >> 1 = 010z
design.sv:12: $finish called at 10 (1s)
Done
```

ConcatenationOperator

```
module ConcatenationOperator();
   initial begin
      // Concatenation
      $display ("{4'b1001,4'b10x1} = %b", {4'b1001, 4'b10x1});
      #10 $finish;
   end
endmodule
```

Results

```
[2024-09-29 00:57:27 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
{4'b1001,4'b10x1} = 100110x1
design.sv:5: $finish called at 10 (1s)
Done
```

ReplicationOperator

```
module ReplicationOperator();

   initial begin
      // replication
      $display ("{4{4'b1001}} = %b", {4{4'b1001}});
```

```
        // replication and concatenation
        $display ("{4{4'b1001,1'bz}} = %b", {4{4'b1001,1'bz}});

        #10 $finish;
    end
endmodule
```

Results

```
[2024-09-29 01:02:44 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
{4{4'b1001}} = 1001100110011001
{4{4'b1001,1'bz}} = 1001z1001z1001z1001z
design.sv:10: $finish called at 10 (1s)
Done
```

2. **Addbit module**

```
// Addbit Module
module addbit (
    input a,        // First bit
    input b,        // Second bit
    input cin,      // Carry in
    output sum,     // Sum output
    output cout     // Carry out
);

    // Full adder logic
    assign sum = a ^ b ^ cin;       // Sum is the XOR of the inputs and carry in
    assign cout = (a & b) | (cin & (a ^ b)); // Carry out is generated based on inputs and
carry in

endmodule
```

Adder Hier Module

```
// Adder Hier Module
module adder_hier (
    output [3:0] result_o,  // 4-bit sum output
    output carry_o,         // Carry out of the final addition
    input [3:0] r1_i,       // First 4-bit input
    input [3:0] r2_i,       // Second 4-bit input
    input ci_i              // Carry in input
);
```

```
    // Internal wires for carry between stages
    wire c1_w, c2_w, c3_w;

    // Instantiate addbit modules for each bit
    addbit u0 (
        .a(r1_i[0]), .b(r2_i[0]), .cin(ci_i),
        .sum(result_o[0]), .cout(c1_w)
    );

    addbit u1 (
        .a(r1_i[1]), .b(r2_i[1]), .cin(c1_w),
        .sum(result_o[1]), .cout(c2_w)
    );

    addbit u2 (
        .a(r1_i[2]), .b(r2_i[2]), .cin(c2_w),
        .sum(result_o[2]), .cout(c3_w)
    );

    addbit u3 (
        .a(r1_i[3]), .b(r2_i[3]), .cin(c3_w),
        .sum(result_o[3]), .cout(carry_o)
    );

endmodule
```

Testbench Module

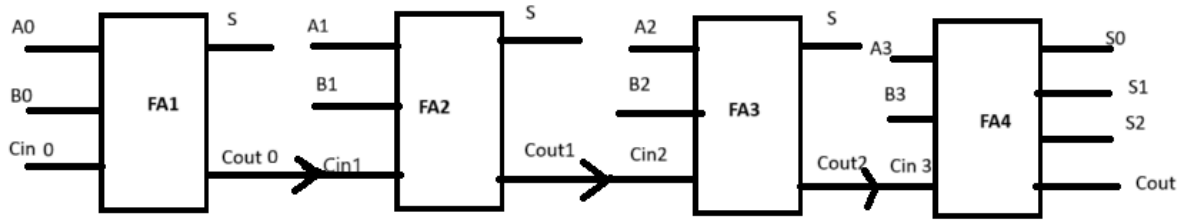
```
// Testbench Module
module tb_adder_hier();

    // Declare registers for inputs and wires for outputs
    reg [3:0] r1_r, r2_r;
    reg ci_r;
    wire [3:0] result_w;
    wire carry_w;

    // Instantiate the adder_hier module
    adder_hier UUT (
        .result_o(result_w),
        .carry_o(carry_w),
        .r1_i(r1_r),
```

```
    .r2_i(r2_r),
    .ci_i(ci_r)
);

// Drive the inputs
initial begin
    // Test case 1: Add 0 + 0 + 0
    r1_r = 4'b0000;
    r2_r = 4'b0000;
    ci_r = 1'b0;
    #10;  // Wait for 10 time units

    // Test case 2: Add 10 (0b1010) + 0 + 0
    r1_r = 4'b1010;
    r2_r = 4'b0000;
    ci_r = 1'b0;
    #10;

    // Test case 3: Add 10 (0b1010) + 2 (0b0010) + 0
    r1_r = 4'b1010;
    r2_r = 4'b0010;
    ci_r = 1'b0;
    #10;

    // Test case 4: Add 10 (0b1010) + 2 (0b0010) + 1 (carry-in)
    r1_r = 4'b1010;
    r2_r = 4'b0010;
    ci_r = 1'b1;
    #10;

    $finish;  // End simulation
end

// Monitor the inputs and outputs
initial begin
    $display("+-------------------------------------------------+");
    $display("| r1    | r2    | ci | result | carry |");
    $display("+-------------------------------------------------+");
    $monitor("| %b | %b |  %b  |  %b   |   %b   |", r1_r, r2_r, ci_r, result_w, carry_w);
    $display("+-------------------------------------------------+");
```

```
    end

endmodule
```

Results

```
[2024-09-30 06:15:06 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
+-----------------------------------------------+
| r1    | r2    | ci | result | carry |
+-----------------------------------------------+
+-----------------------------------------------+
| 0000 | 0000 |  0  |  0000   |   0   |
| 1010 | 0000 |  0  |  1010   |   0   |
| 1010 | 0010 |  0  |  1100   |   0   |
| 1010 | 0010 |  1  |  1101   |   0   |
design.sv:95: $finish called at 40 (1s)
Done
```



Circuit Block Diagram for Adder Hier Module

Synchronous DFF

```
// D Flip-Flop Module
`timescale 1ns / 1ps  // Set the time unit to 1ns and precision to 1ps

module DFFSynch(
    input d_i,      // Data input
    input rst_i,    // Reset input
    input clk_i,    // Clock input
    output reg q_o   // Output
);
    always @(posedge clk_i) begin
        if (rst_i)
            q_o <= 0;   // Reset output to 0
```

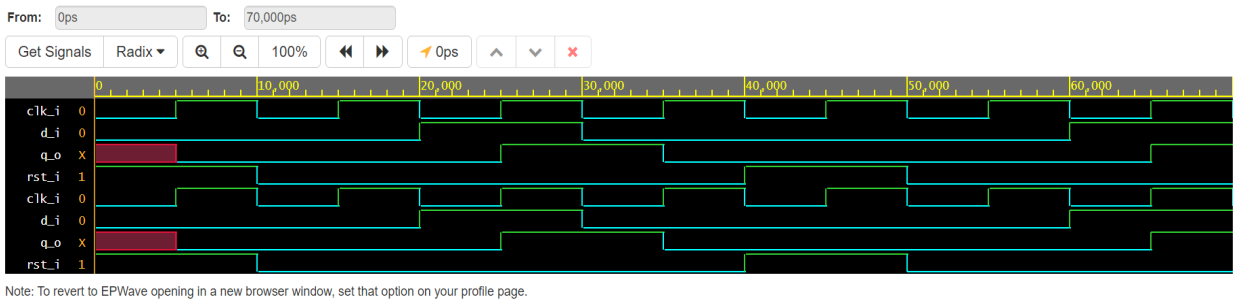
```
        else
            q_o <= d_i; // Update output with data input on clock edge
    end
endmodule
```

Testbench

```
// Testbench for D Flip-Flop
`timescale 1ns / 1ps  // Set the time unit and precision

module tb_DFFSynch;

    // Declare registers for inputs and wire for output
    reg d_i;         // Input data
    reg rst_i;       // Reset input
    reg clk_i;       // Clock input
    wire q_o;        // Output

    // Instantiate the DFF module
    DFFSynch dut (
        .d_i(d_i),
        .rst_i(rst_i),
        .clk_i(clk_i),
        .q_o(q_o)
    );

    // Clock generation
    initial begin
        clk_i = 0;  // Initialize clock
        forever #5 clk_i = ~clk_i;  // Toggle clock every 5 time units
    end

    // Test sequence
    initial begin
        // Set up the VCD file for waveform viewing
        $dumpfile("dump.vcd"); // Specify the name of the VCD file
        $dumpvars(0, tb_DFFSynch); // Dump all variables in the testbench

        // Monitor the output
        $monitor("Time: %0dns | d_i: %b | rst_i: %b | clk_i: %b | q_o: %b", $time, d_i, rst_i, clk_i, q_o);
```

```
        // Test case 1: Reset the DFF
        rst_i = 1; d_i = 0; #10;  // Apply reset
        rst_i = 0; #10;           // Release reset

        // Test case 2: Set d_i to 1
        d_i = 1; #10;             // Set d_i to 1, should update q_o on next clock

        // Test case 3: Set d_i to 0
        d_i = 0; #10;             // Set d_i to 0, should update q_o on next clock

        // Test case 4: Check reset again
        rst_i = 1; #10;           // Apply reset again
        rst_i = 0; #10;           // Release reset

        // Test case 5: Set d_i to 1 again
        d_i = 1; #10;             // Set d_i to 1, should update q_o on next clock

        // Finish the simulation
        $finish;
    end

endmodule
```

Results



Note: To revert to EPWave opening in a new browser window, set that option on your profile page.

Asynchronous DFF

```
// Asynchronous D Flip-Flop Module
`timescale 1ns / 1ps  // Set the time unit and precision

module DFFAsynch(
    input d_i,      // Data input
    input rst_i,    // Reset input
    input clk_i,    // Clock input
    output reg q_o   // Output
```

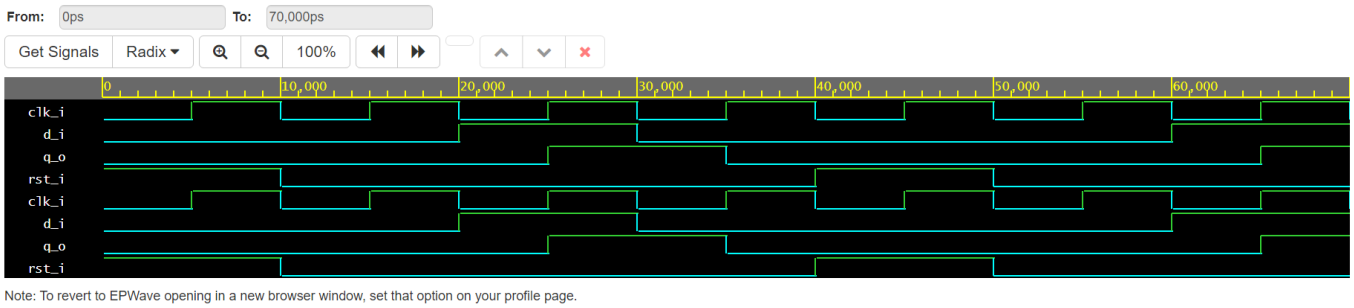
```
);
    always @(posedge clk_i or posedge rst_i) begin
        if (rst_i)
            q_o <= 0;   // Reset output to 0
        else
            q_o <= d_i; // Update output with data input on clock edge
    end
endmodule
```

Testbench

```
// Testbench for DFFAsynch
`timescale 1ns / 1ps  // Set the time unit and precision

module tb_DFFAsynch;

    // Declare registers for inputs and wire for output
    reg d_i;         // Input data
    reg rst_i;       // Reset input
    reg clk_i;       // Clock input
    wire q_o;        // Output

    // Instantiate the DFFAsynch module
    DFFAsynch dut (
        .d_i(d_i),
        .rst_i(rst_i),
        .clk_i(clk_i),
        .q_o(q_o)
    );

    // Clock generation
    initial begin
        clk_i = 0;  // Initialize clock
        forever #5 clk_i = ~clk_i;  // Toggle clock every 5 time units
    end

    // Test sequence
    initial begin
        // Set up the VCD file for waveform viewing
        $dumpfile("dump.vcd"); // Specify the name of the VCD file
        $dumpvars(0, tb_DFFAsynch); // Dump all variables in the testbench
```

```
        // Monitor the output
        $monitor("Time: %0dns | d_i: %b | rst_i: %b | clk_i: %b | q_o: %b", $time, d_i, rst_i, clk_i,
q_o);

        // Test case 1: Reset the DFF
        rst_i = 1; d_i = 0; #10;  // Apply reset
        rst_i = 0; #10;           // Release reset

        // Test case 2: Set d_i to 1
        d_i = 1; #10;             // Set d_i to 1, should update q_o on next clock

        // Test case 3: Set d_i to 0
        d_i = 0; #10;             // Set d_i to 0, should update q_o on next clock

        // Test case 4: Check reset functionality
        rst_i = 1; #10;           // Apply reset again
        rst_i = 0; #10;           // Release reset

        // Test case 5: Set d_i to 1 again
        d_i = 1; #10;             // Set d_i to 1, should update q_o on next clock

        // Finish the simulation
        $finish;
    end

endmodule
```

Results



Note: To revert to EPWave opening in a new browser window, set that option on your profile page.