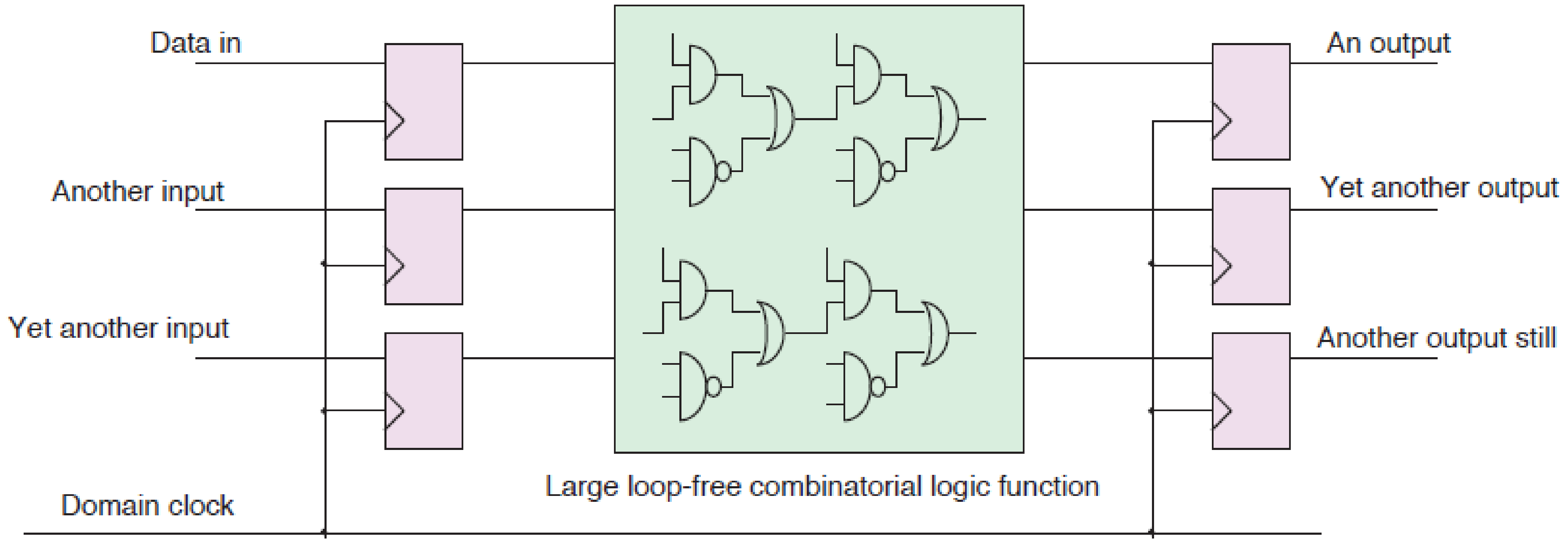
Part 1: Practical Basic Verilog Examples

Erwin

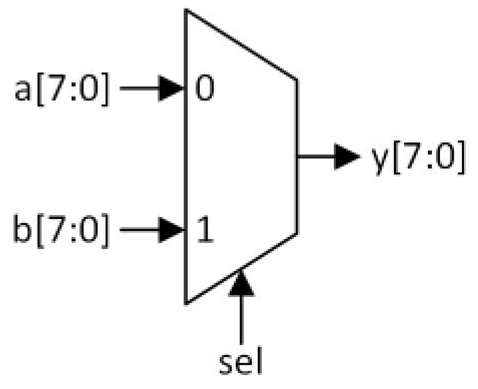
Verilog Module

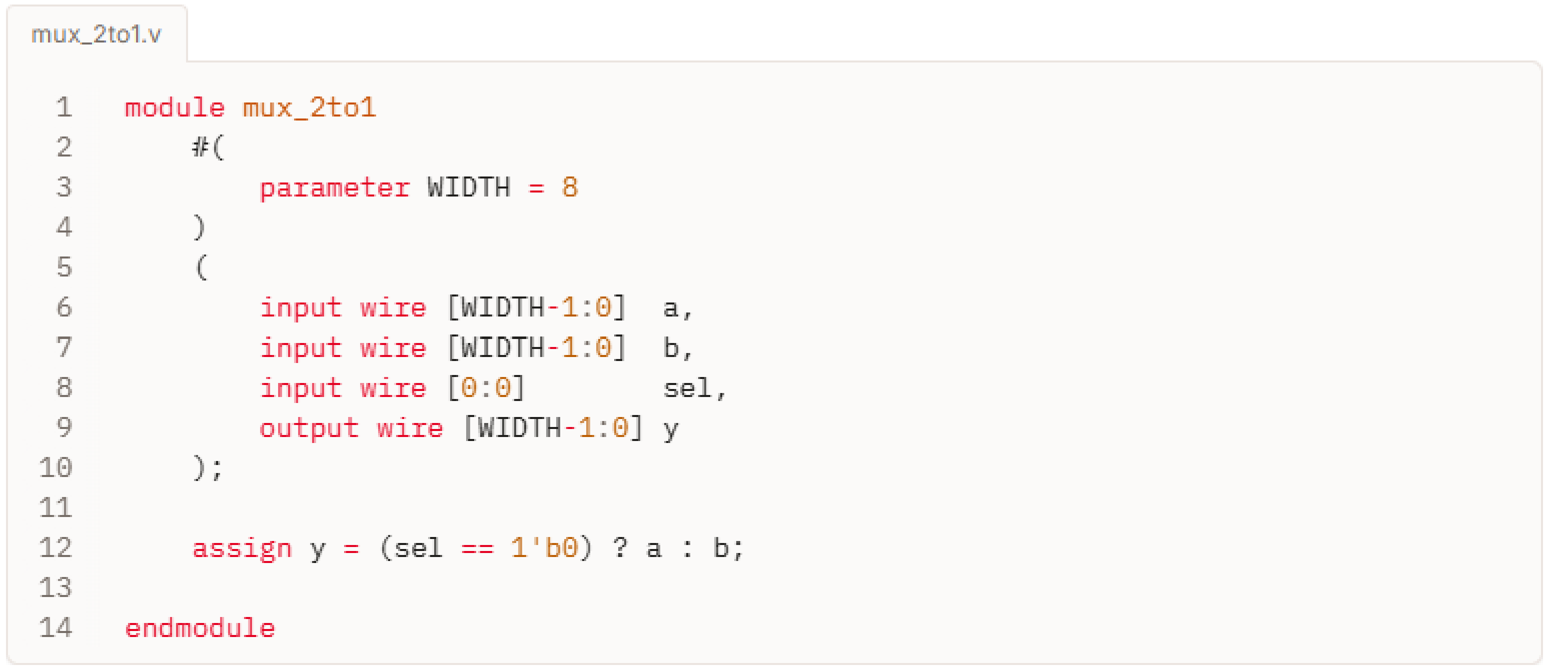
- **Combinational**: output depends only on the current inputs — no memory, no clock
- **Sequetial**: output depends on current inputs and past states.It requires a clock to update its state.



Combinational Cricuit Examples

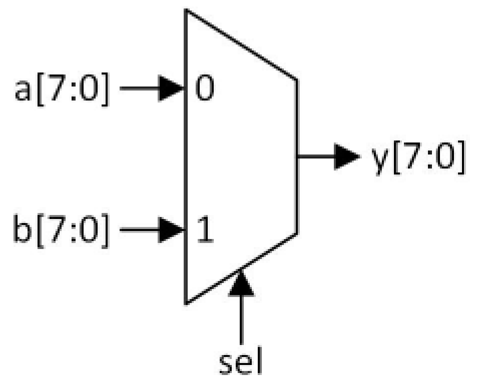
Multiplexer using Assign




mux_2to1.v

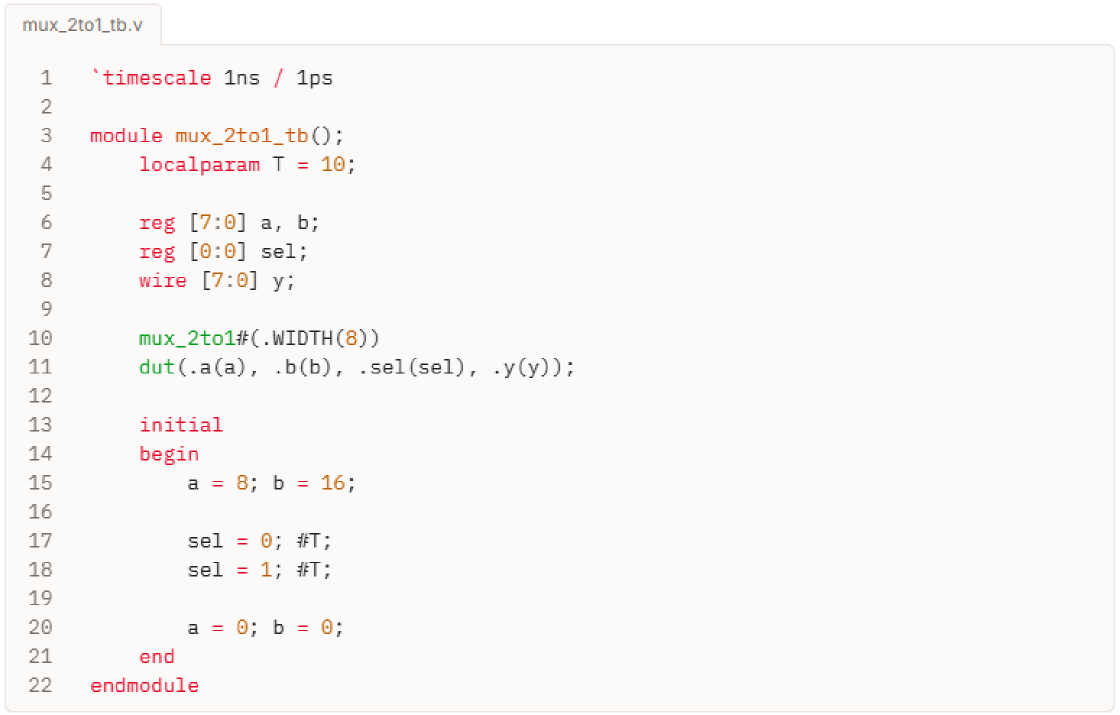
```verilog
module mux_2to1
    #(
        parameter WIDTH = 8
    )
    (
        input wire [WIDTH-1:0]  a,
        input wire [WIDTH-1:0]  b,
        input wire [0:0]        sel,
        output wire [WIDTH-1:0] y
    );

    assign y = (sel == 1'b0) ? a : b;

endmodule
```

Multiplexer using Always



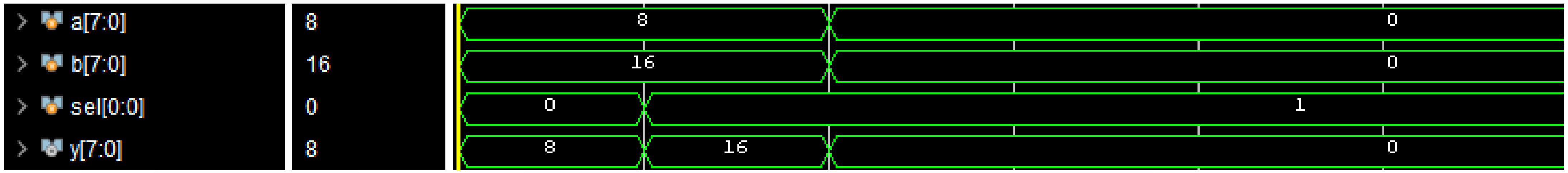
```verilog
module mux_2to1
    #(
        parameter WIDTH = 8
    )
    (
        input wire [WIDTH-1:0]  a,
        input wire [WIDTH-1:0]  b,
        input wire [0:0]        sel,
        output reg [WIDTH-1:0]  y
    );

    always @(*)
        if (sel == 1'b0)
            y = a;
        else
            y = b;

endmodule
```

Multiplexer Testbench



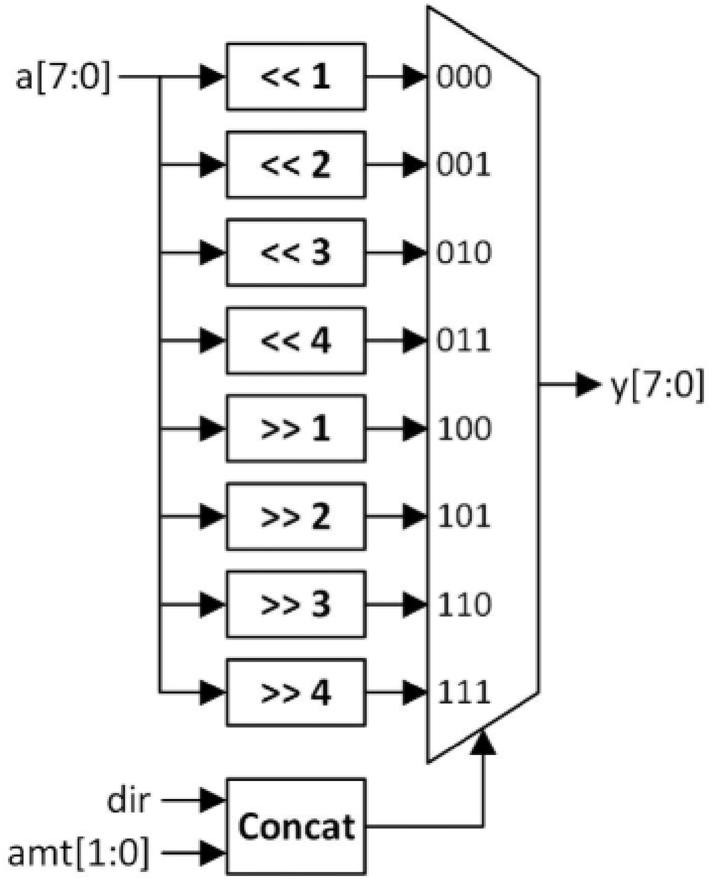
mux_2to1_tb.v

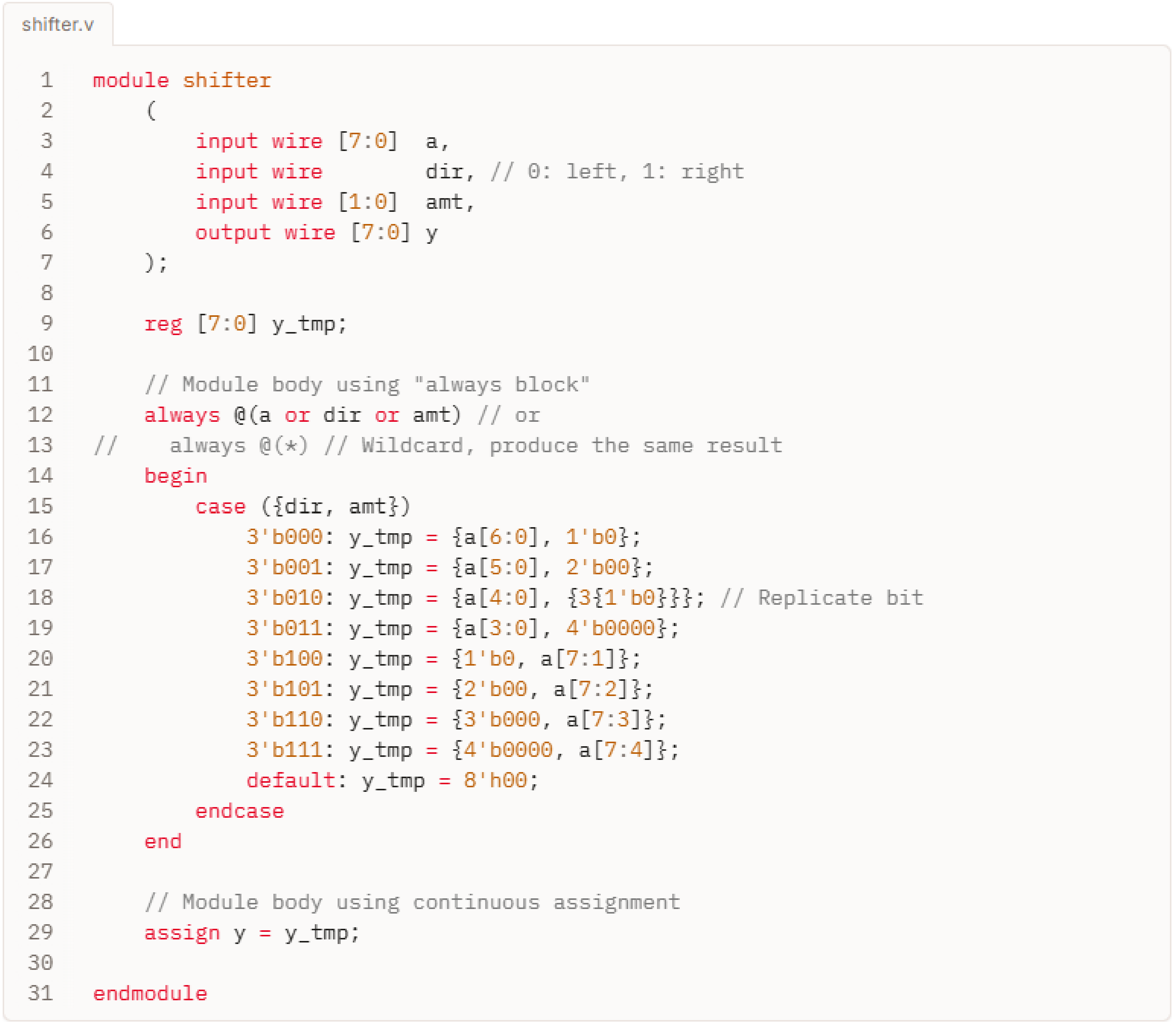
```verilog
`timescale 1ns / 1ps

module mux_2to1_tb();
    localparam T = 10;

    reg [7:0] a, b;
    reg [0:0] sel;
    wire [7:0] y;

    mux_2to1#(.WIDTH(8))
    dut(.a(a), .b(b), .sel(sel), .y(y));

    initial
    begin
        a = 8; b = 16;

        sel = 0; #T;
        sel = 1; #T;

        a = 0; b = 0;
    end
endmodule
```



```
a[7:0]     8     8            0
b[7:0]     16    16           0
sel[0:0]   0     0    1
y[7:0]     8     8    16      0
```

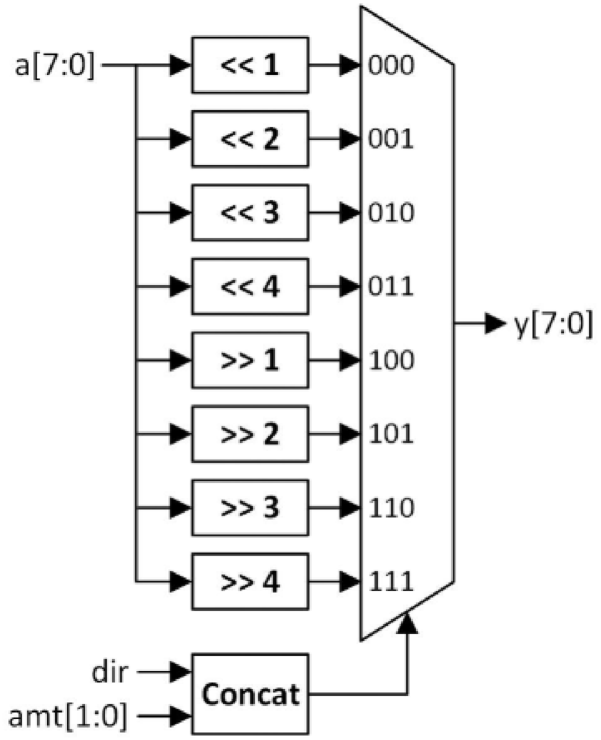
Shifter

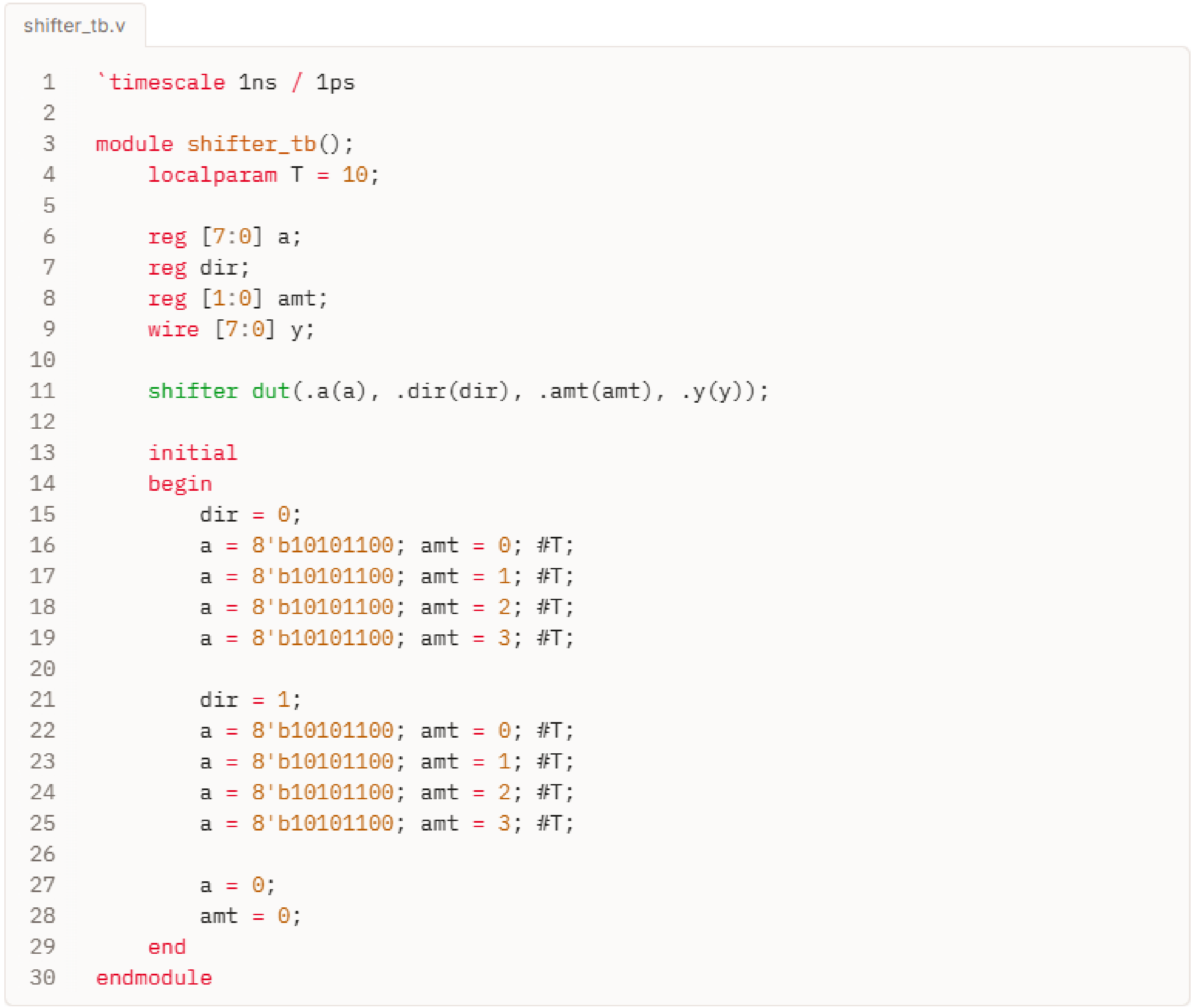



shifter.v

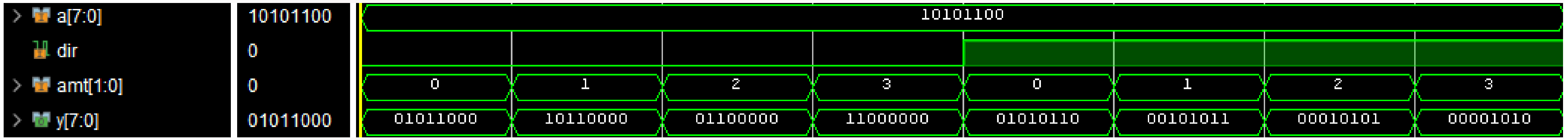
```verilog
module shifter
    (
        input wire [7:0]  a,
        input wire        dir, // 0: left, 1: right
        input wire [1:0]  amt,
        output wire [7:0] y
    );

    reg [7:0] y_tmp;

    // Module body using "always block"
    always @(a or dir or amt) // or
//    always @(*) // Wildcard, produce the same result
    begin
        case ({dir, amt})
            3'b000: y_tmp = {a[6:0], 1'b0};
            3'b001: y_tmp = {a[5:0], 2'b00};
            3'b010: y_tmp = {a[4:0], {3{1'b0}}}; // Replicate bit
            3'b011: y_tmp = {a[3:0], 4'b0000};
            3'b100: y_tmp = {1'b0, a[7:1]};
            3'b101: y_tmp = {2'b00, a[7:2]};
            3'b110: y_tmp = {3'b000, a[7:3]};
            3'b111: y_tmp = {4'b0000, a[7:4]};
            default: y_tmp = 8'h00;
        endcase
    end

    // Module body using continuous assignment
    assign y = y_tmp;

endmodule
```

Shifter Testbench




shifter_tb.v

```
`timescale 1ns / 1ps

module shifter_tb();
    localparam T = 10;

    reg [7:0] a;
    reg dir;
    reg [1:0] amt;
    wire [7:0] y;

    shifter dut(.a(a), .dir(dir), .amt(amt), .y(y));

    initial
    begin
        dir = 0;
        a = 8'b10101100; amt = 0; #T;
        a = 8'b10101100; amt = 1; #T;
        a = 8'b10101100; amt = 2; #T;
        a = 8'b10101100; amt = 3; #T;

        dir = 1;
        a = 8'b10101100; amt = 0; #T;
        a = 8'b10101100; amt = 1; #T;
        a = 8'b10101100; amt = 2; #T;
        a = 8'b10101100; amt = 3; #T;

        a = 0;
        amt = 0;
    end
endmodule
```

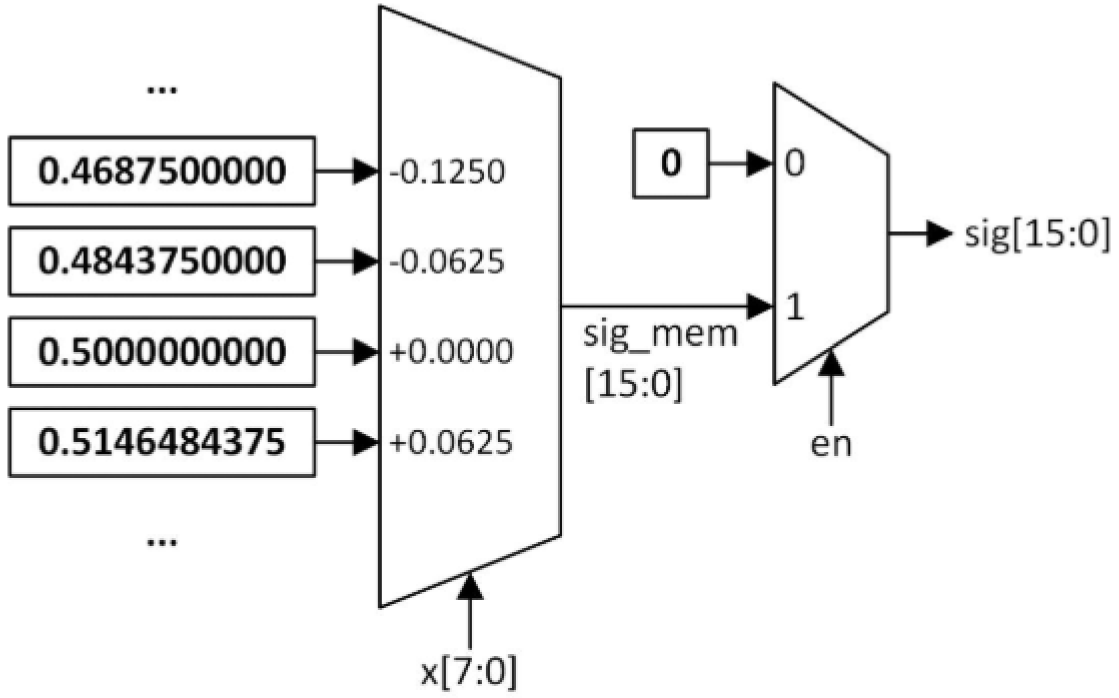


Signal	Value								
a[7:0]	10101100	10101100							
dir	0	0	0	0	0	1	1	1	1
amt[1:0]	0	0	1	2	3	0	1	2	3
y[7:0]	01011000	01011000	10110000	01100000	11000000	01010110	00101011	00010101	00001010

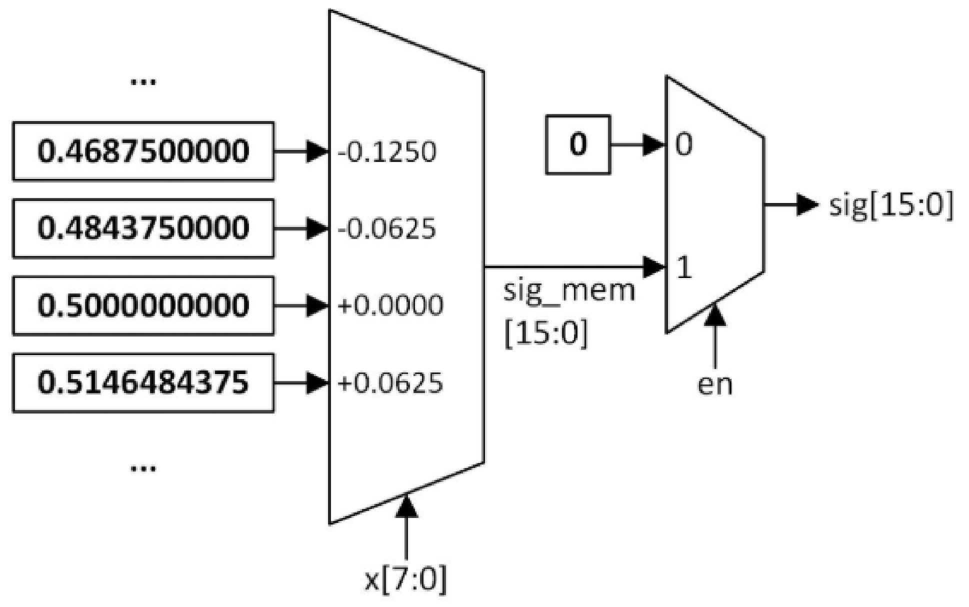
Math Look-Up Table

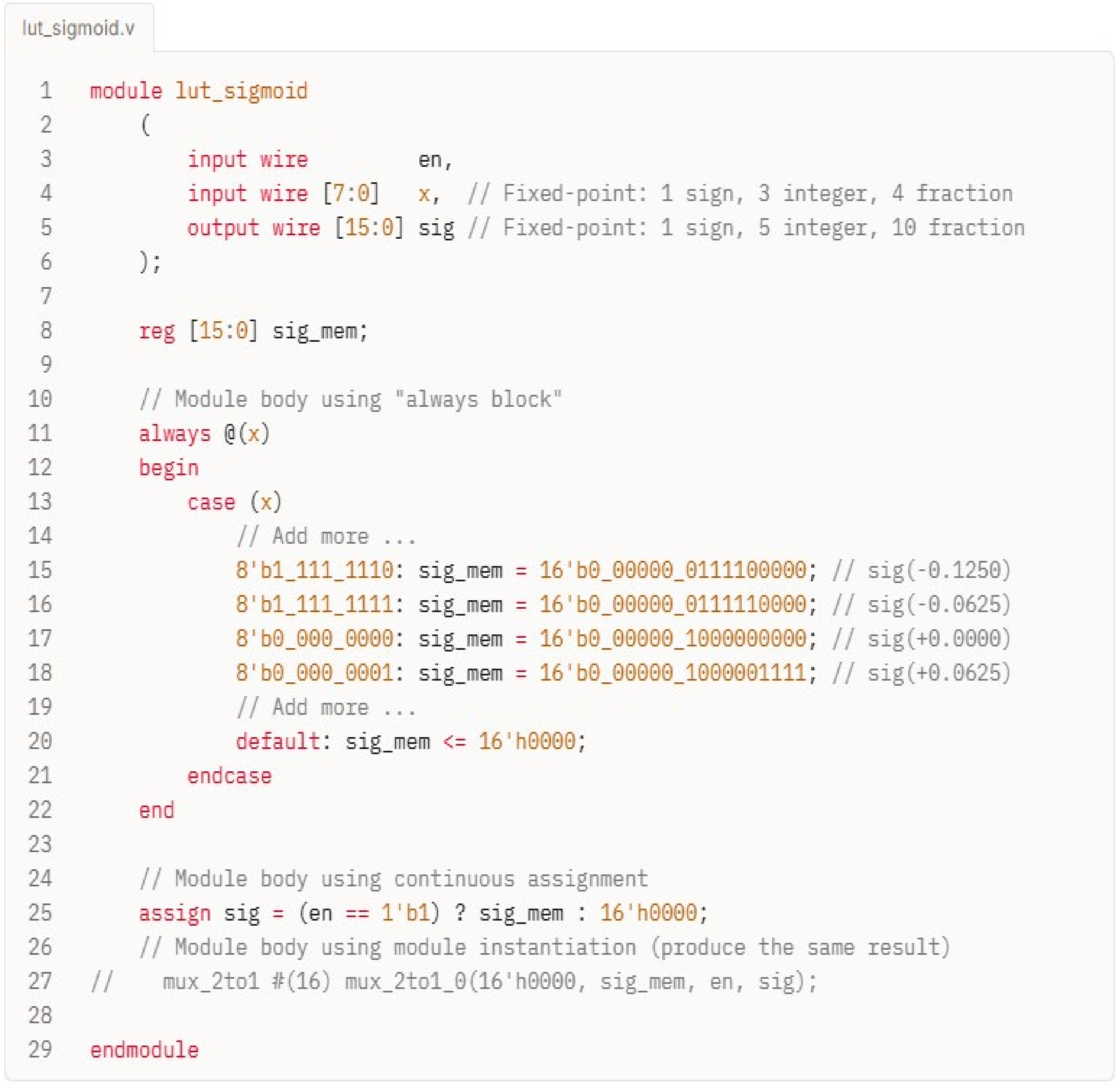
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

x	σ(x)
...	...
-0.1250	0.4687500000
-0.0625	0.4843750000
+0.0000	0.5000000000
+0.0625	0.5146484375
...	...



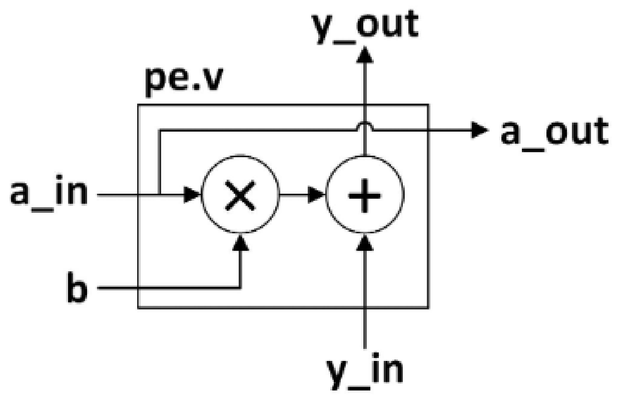
Math Look-Up Table

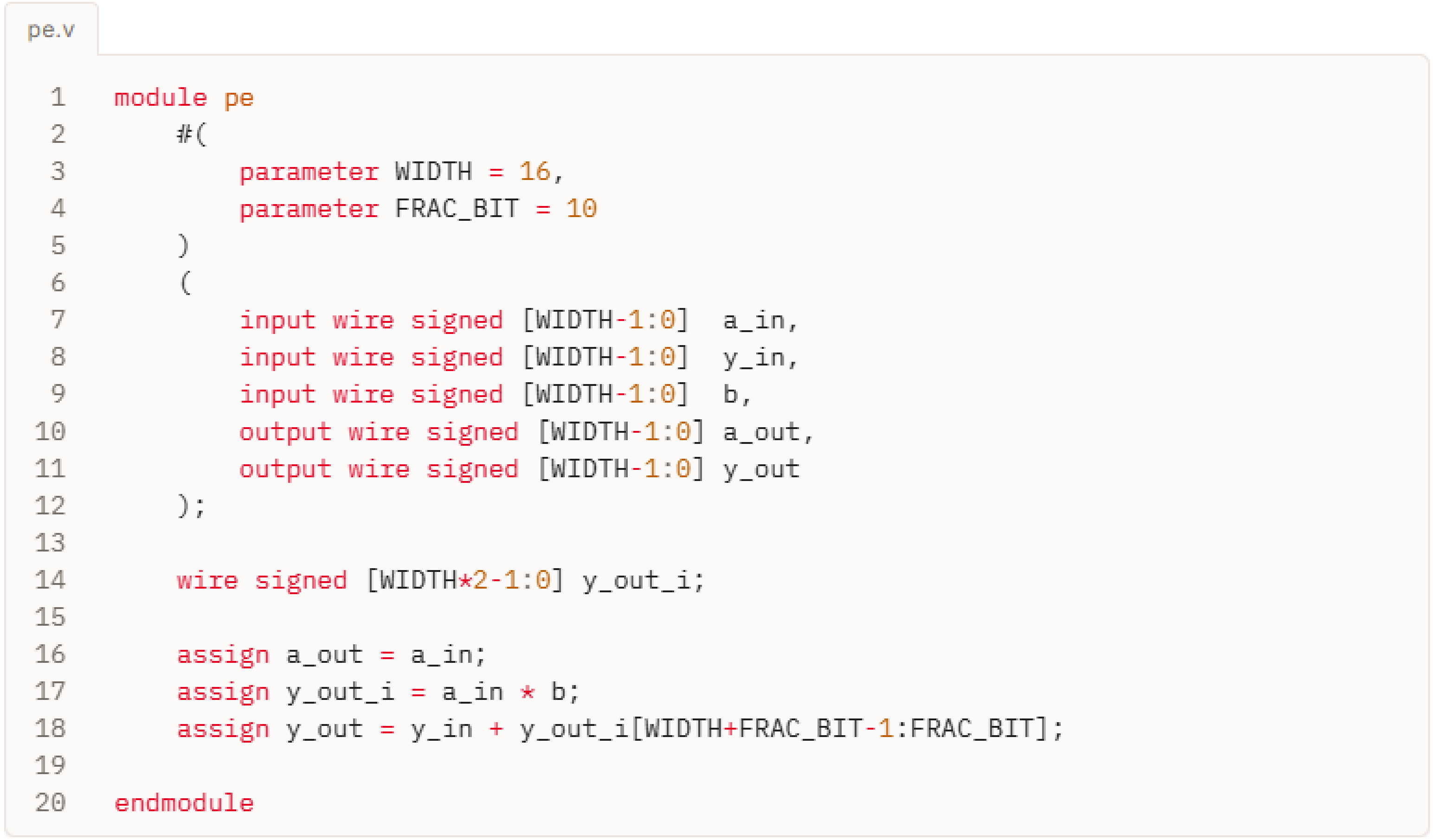



lut_sigmoid.v

```verilog
module lut_sigmoid
    (
        input wire          en,
        input wire [7:0]    x,  // Fixed-point: 1 sign, 3 integer, 4 fraction
        output wire [15:0] sig // Fixed-point: 1 sign, 5 integer, 10 fraction
    );

    reg [15:0] sig_mem;

    // Module body using "always block"
    always @(x)
    begin
        case (x)
            // Add more ...
            8'b1_111_1110: sig_mem = 16'b0_00000_0111100000; // sig(-0.1250)
            8'b1_111_1111: sig_mem = 16'b0_00000_0111110000; // sig(-0.0625)
            8'b0_000_0000: sig_mem = 16'b0_00000_1000000000; // sig(+0.0000)
            8'b0_000_0001: sig_mem = 16'b0_00000_1000001111; // sig(+0.0625)
            // Add more ...
            default: sig_mem <= 16'h0000;
        endcase
    end

    // Module body using continuous assignment
    assign sig = (en == 1'b1) ? sig_mem : 16'h0000;
    // Module body using module instantiation (produce the same result)
//    mux_2to1 #(16) mux_2to1_0(16'h0000, sig_mem, en, sig);

endmodule
```

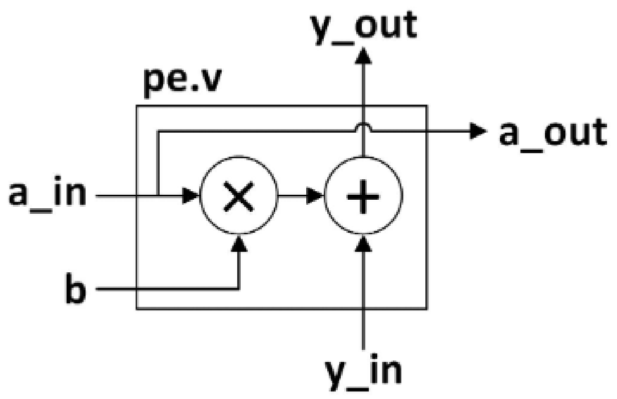
Multiply and Add

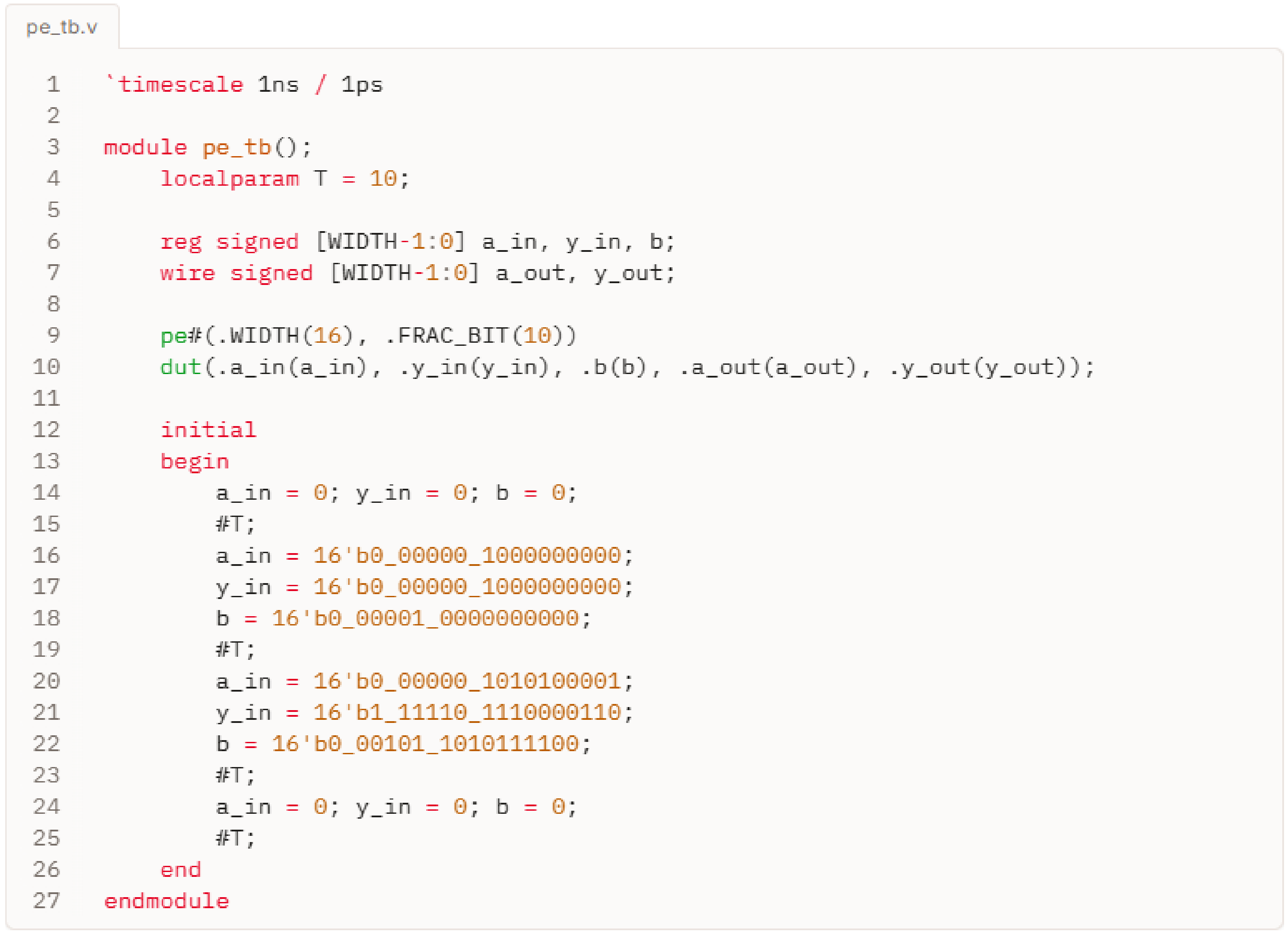



pe.v

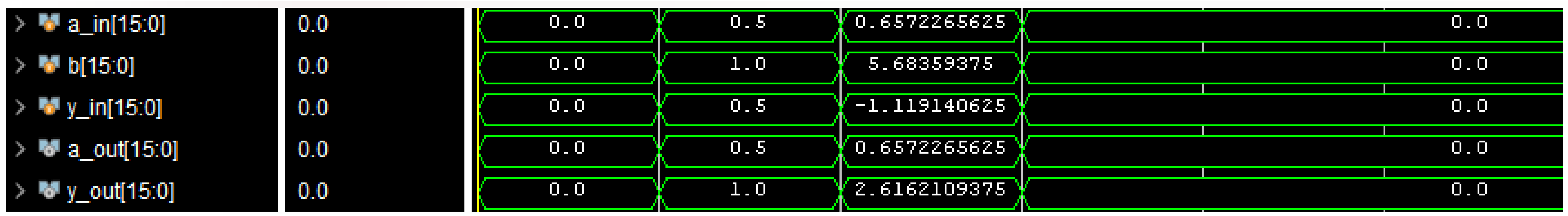
```verilog
module pe
    #(
        parameter WIDTH = 16,
        parameter FRAC_BIT = 10
    )
    (
        input wire signed [WIDTH-1:0]  a_in,
        input wire signed [WIDTH-1:0]  y_in,
        input wire signed [WIDTH-1:0]  b,
        output wire signed [WIDTH-1:0] a_out,
        output wire signed [WIDTH-1:0] y_out
    );

    wire signed [WIDTH*2-1:0] y_out_i;

    assign a_out = a_in;
    assign y_out_i = a_in * b;
    assign y_out = y_in + y_out_i[WIDTH+FRAC_BIT-1:FRAC_BIT];

endmodule
```

Multiply and Add




pe_tb.v

```verilog
`timescale 1ns / 1ps

module pe_tb();
    localparam T = 10;

    reg signed [WIDTH-1:0] a_in, y_in, b;
    wire signed [WIDTH-1:0] a_out, y_out;

    pe#(.WIDTH(16), .FRAC_BIT(10))
    dut(.a_in(a_in), .y_in(y_in), .b(b), .a_out(a_out), .y_out(y_out));

    initial
    begin
        a_in = 0; y_in = 0; b = 0;
        #T;
        a_in = 16'b0_00000_1000000000;
        y_in = 16'b0_00000_1000000000;
        b = 16'b0_00001_0000000000;
        #T;
        a_in = 16'b0_00000_1010100001;
        y_in = 16'b1_11110_1110000110;
        b = 16'b0_00101_1010111100;
        #T;
        a_in = 0; y_in = 0; b = 0;
        #T;
    end
endmodule
```



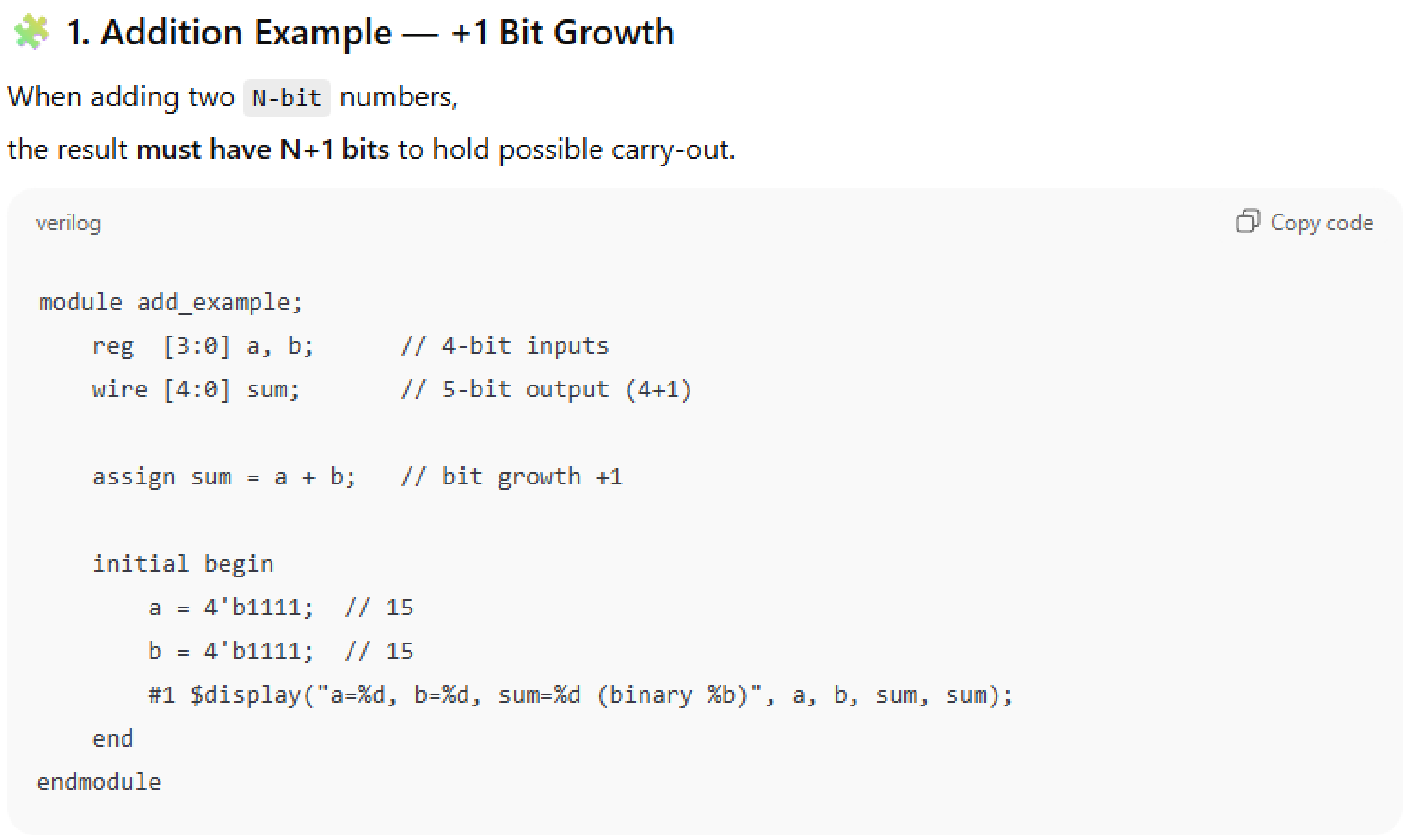
```
a_in[15:0]    0.0    0.0    0.5    0.6572265625     0.0
b[15:0]       0.0    0.0    1.0    5.68359375       0.0
y_in[15:0]    0.0    0.0    0.5    -1.119140625     0.0
a_out[15:0]   0.0    0.0    0.5    0.6572265625     0.0
y_out[15:0]   0.0    0.0    1.0    2.6162109375     0.0
```

Be Careful of Bit Growth!

- Result of Addition: bit +1
- Result of Multiplication: bit x2

https://zipcpu.com/dsp/2017/07/21/bit-growth.html

Be Careful of Bit Growth!



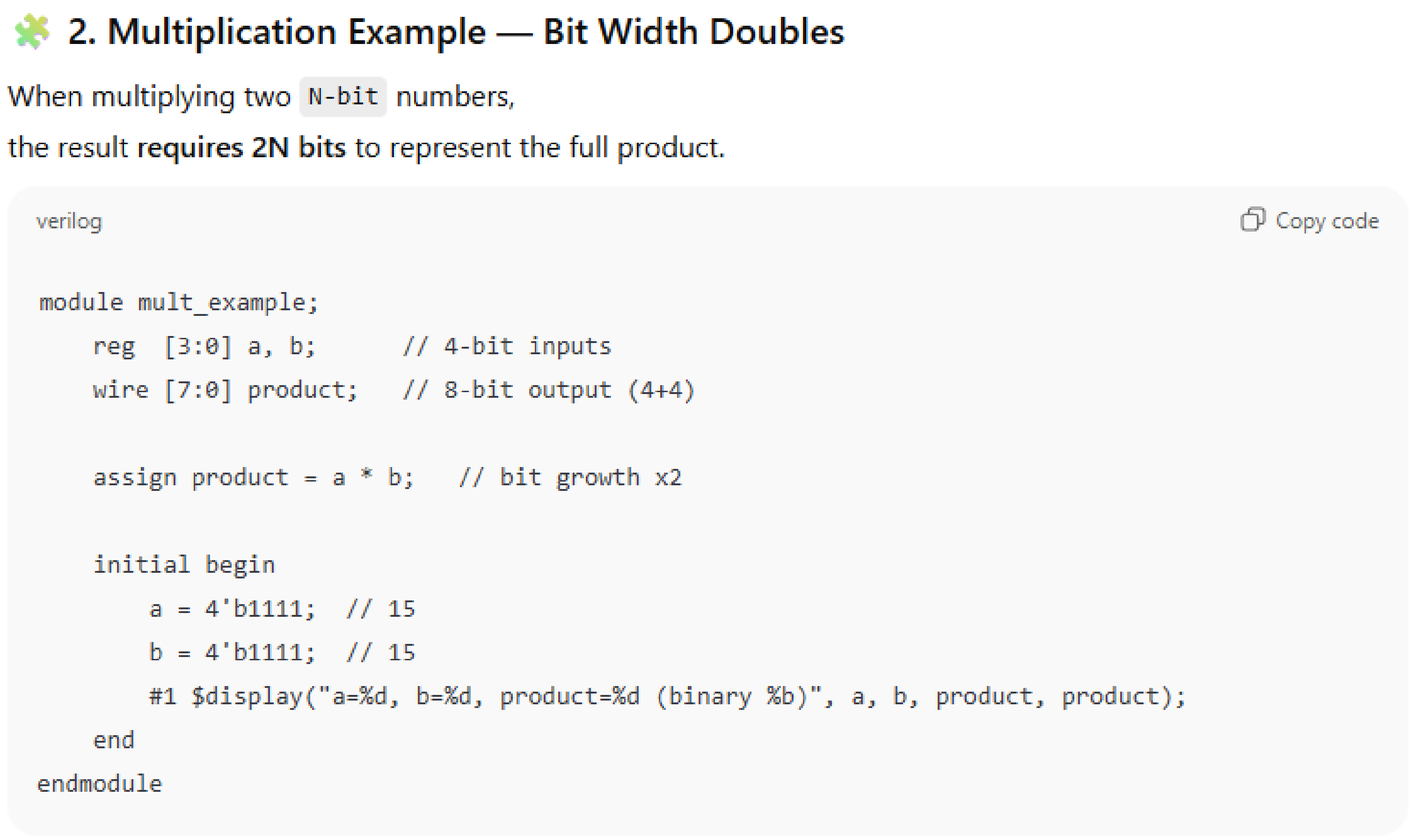
🧩 1. Addition Example — +1 Bit Growth

When adding two `N-bit` numbers,

the result **must have N+1 bits** to hold possible carry-out.

```verilog
module add_example;
    reg  [3:0] a, b;      // 4-bit inputs
    wire [4:0] sum;       // 5-bit output (4+1)

    assign sum = a + b;   // bit growth +1

    initial begin
        a = 4'b1111;  // 15
        b = 4'b1111;  // 15
        #1 $display("a=%d, b=%d, sum=%d (binary %b)", a, b, sum, sum);
    end
endmodule
```

Be Careful of Bit Growth!



🧩 2. Multiplication Example — Bit Width Doubles

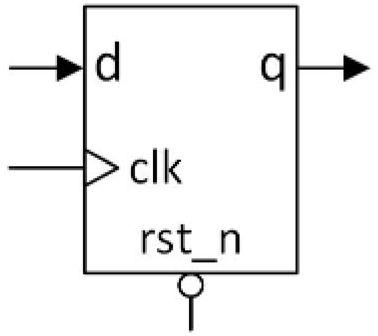
When multiplying two `N-bit` numbers,
the result **requires 2N bits** to represent the full product.

verilog   Copy code

```verilog
module mult_example;
    reg  [3:0] a, b;      // 4-bit inputs
    wire [7:0] product;   // 8-bit output (4+4)

    assign product = a * b;   // bit growth x2

    initial begin
        a = 4'b1111;  // 15
        b = 4'b1111;  // 15
        #1 $display("a=%d, b=%d, product=%d (binary %b)", a, b, product, product);
    end
endmodule
```

Sequential Cricuit Examples

Register

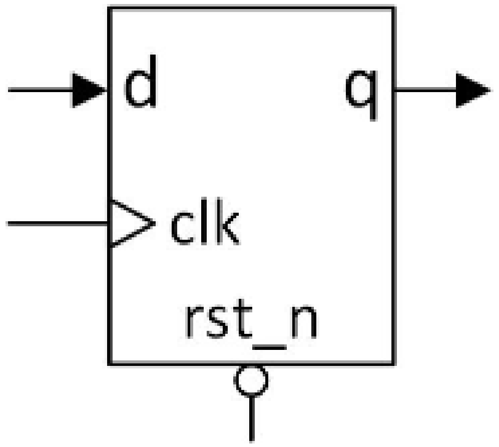


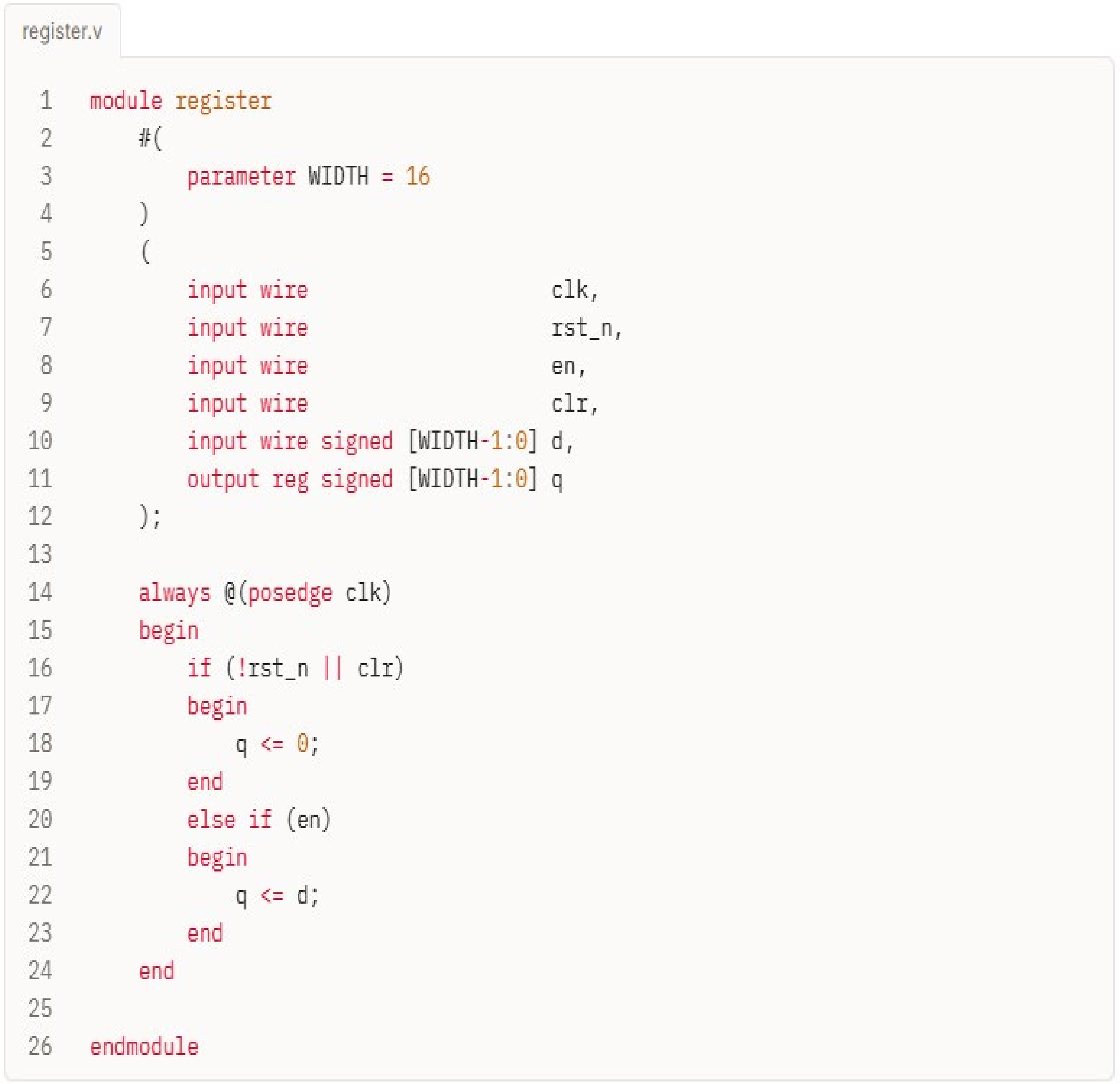
```
module dff
    (
    input wire clk,
    input wire rst_n,
    input wire d,
    output reg q
    );

    always @(posedge clk)
    if (!rst_n)
        q <= 0;
    else
        q <= d;

endmodule
```

clk	rst_n	q*
0	-	q
1	-	q
↑	0	0
↑	1	d

- **reg is not the same as register.**
- **reg data type, not an explicit flip-flop or hardware register.**

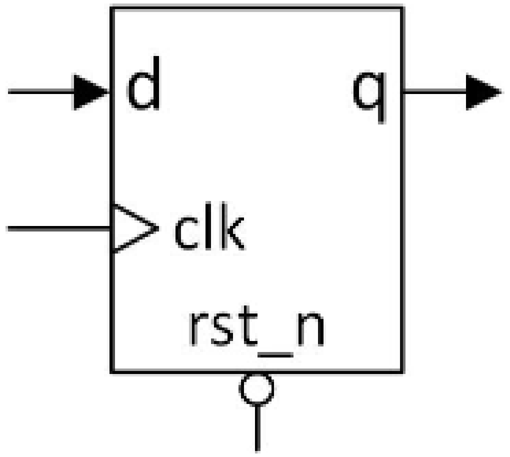
Register More Bits

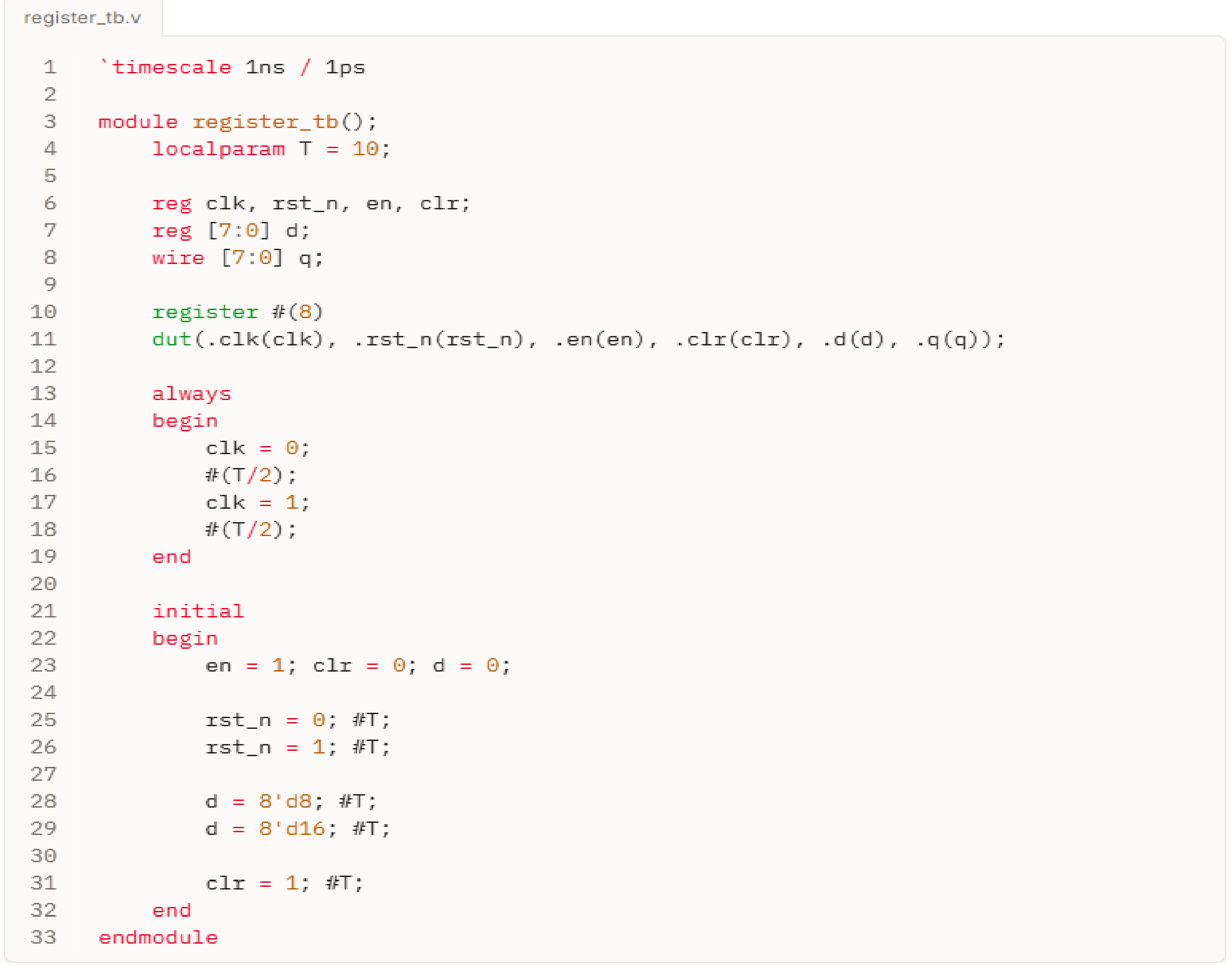



register.v

```verilog
module register
    #(
        parameter WIDTH = 16
    )
    (
        input wire                    clk,
        input wire                    rst_n,
        input wire                    en,
        input wire                    clr,
        input wire signed [WIDTH-1:0] d,
        output reg signed [WIDTH-1:0] q
    );

    always @(posedge clk)
    begin
        if (!rst_n || clr)
        begin
            q <= 0;
        end
        else if (en)
        begin
            q <= d;
        end
    end

endmodule
```

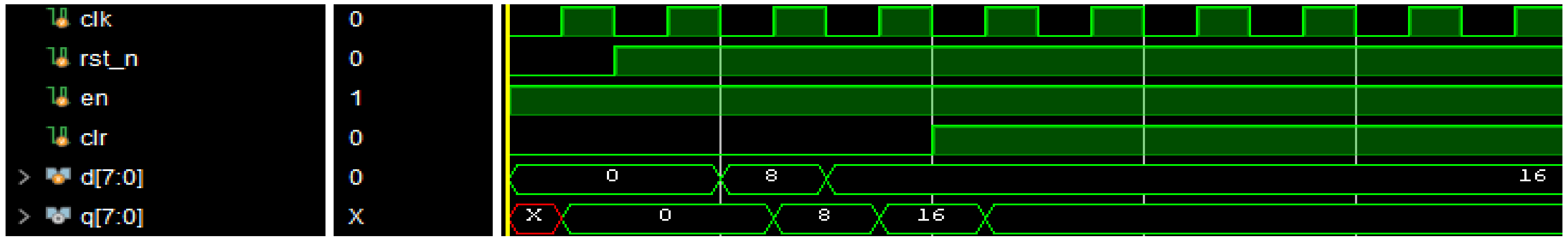
Register Testbench



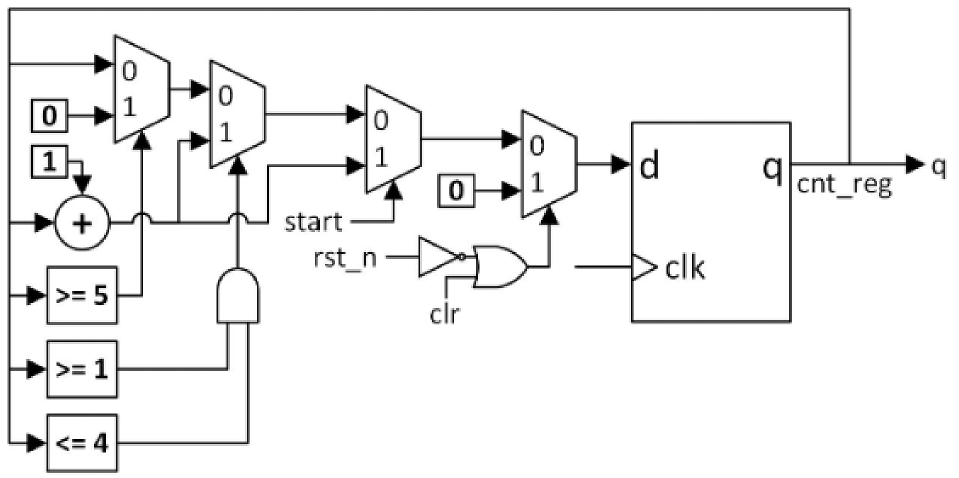


register_tb.v

```verilog
`timescale 1ns / 1ps

module register_tb();
    localparam T = 10;

    reg clk, rst_n, en, clr;
    reg [7:0] d;
    wire [7:0] q;

    register #(8)
    dut(.clk(clk), .rst_n(rst_n), .en(en), .clr(clr), .d(d), .q(q));

    always
    begin
        clk = 0;
        #(T/2);
        clk = 1;
        #(T/2);
    end

    initial
    begin
        en = 1; clr = 0; d = 0;

        rst_n = 0; #T;
        rst_n = 1; #T;

        d = 8'd8; #T;
        d = 8'd16; #T;

        clr = 1; #T;
    end
endmodule
```



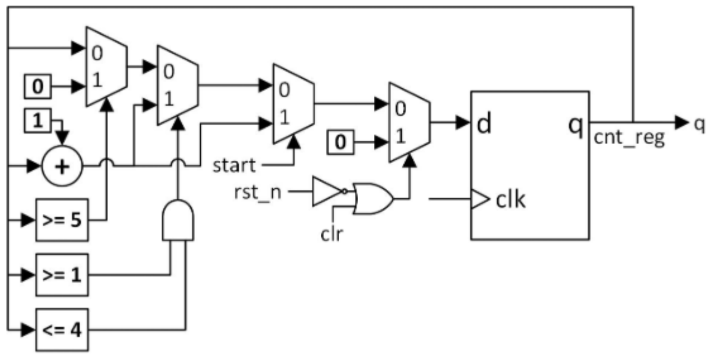
Counter

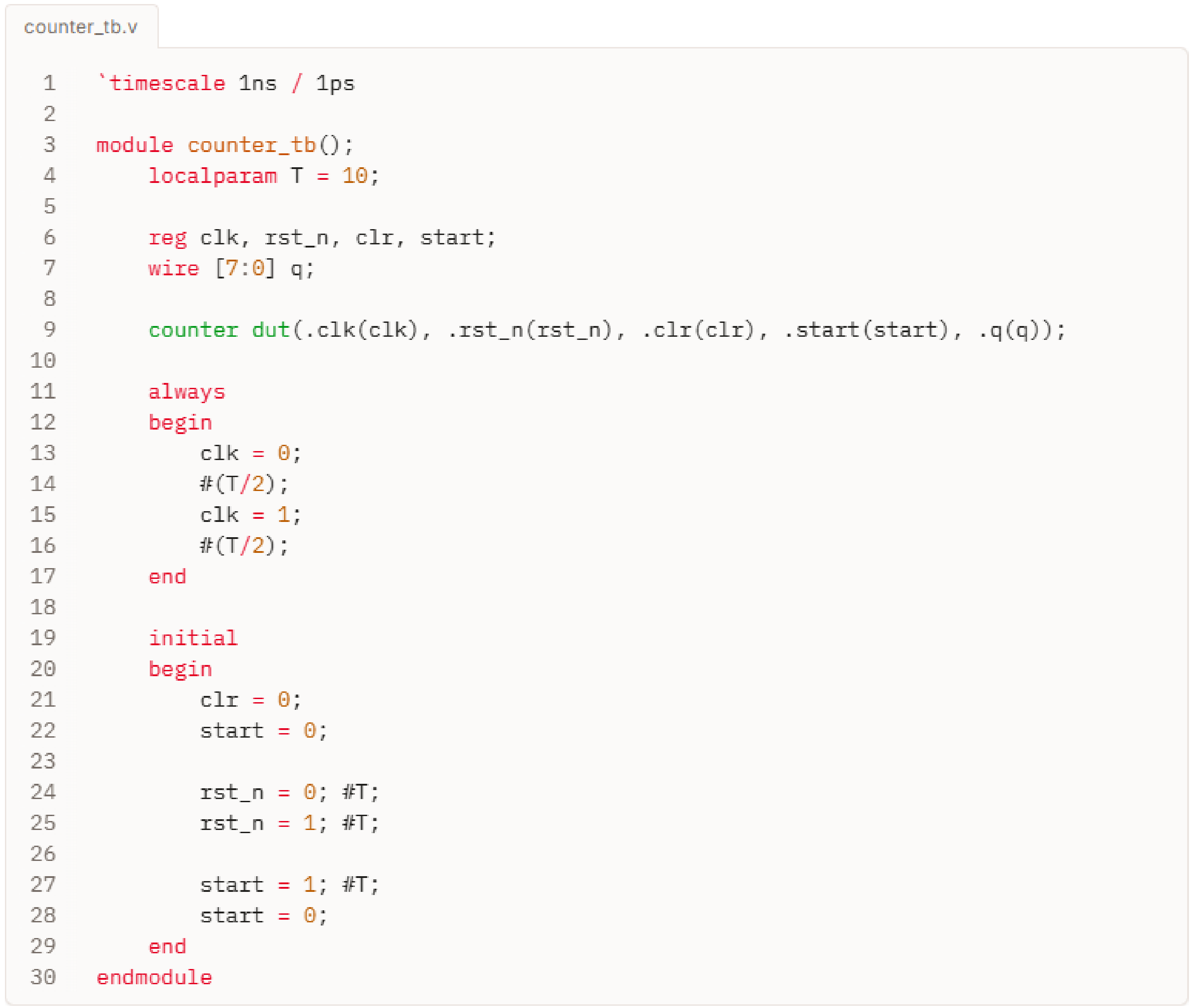


counter.v

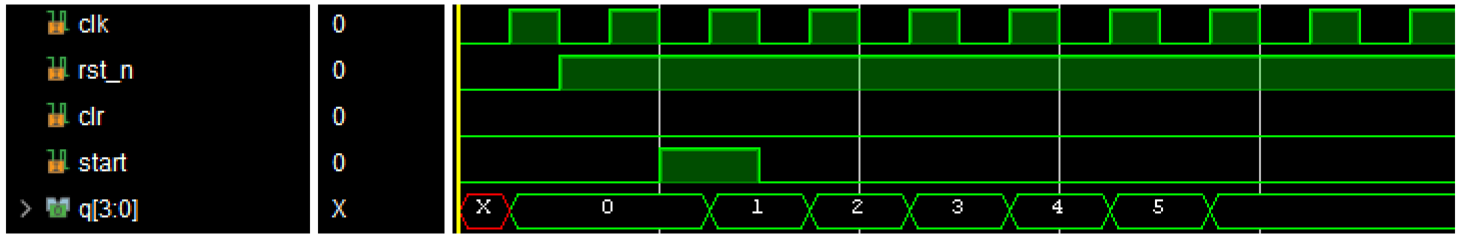
```
module counter
    (
        input wire        clk,
        input wire        rst_n,
        input wire        clr,
        input wire        start,
        output wire [3:0] q
    );

    reg [3:0] cnt_reg;

    always @(posedge clk)
    begin
        if (!rst_n || clr)
        begin
            cnt_reg <= 0;
        end
        else if (start)
        begin
            cnt_reg <= cnt_reg + 1;
        end
        else if (cnt_reg >= 1 && cnt_reg <= 4)
        begin
            cnt_reg <= cnt_reg + 1;
        end
        else if (cnt_reg >= 5)
        begin
            cnt_reg <= 0;
        end
    end

    assign q = cnt_reg;

endmodule
```

Counter




counter_tb.v

```verilog
`timescale 1ns / 1ps

module counter_tb();
    localparam T = 10;

    reg clk, rst_n, clr, start;
    wire [7:0] q;

    counter dut(.clk(clk), .rst_n(rst_n), .clr(clr), .start(start), .q(q));

    always
    begin
        clk = 0;
        #(T/2);
        clk = 1;
        #(T/2);
    end

    initial
    begin
        clr = 0;
        start = 0;

        rst_n = 0; #T;
        rst_n = 1; #T;

        start = 1; #T;
        start = 0;
    end
endmodule
```



Common Mistakes

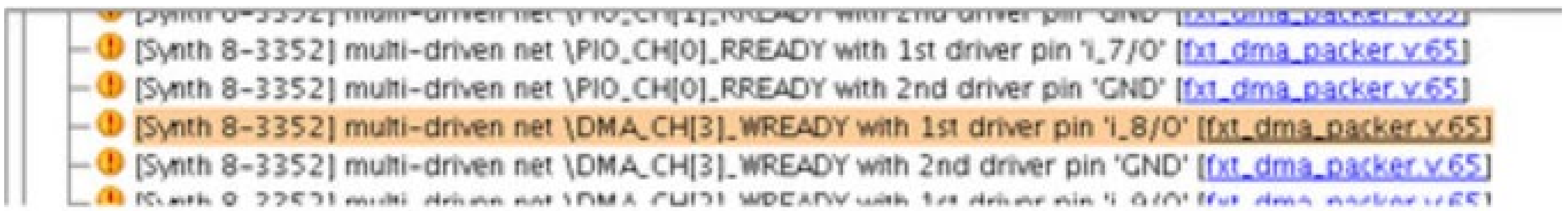
Variable Assigned in Multiple Always Blocks

```verilog
reg y, a, b, clear;

always @(*)
    if (clear)
        y = 1'b0;

always @(*)
    y = a & b;
```

Wrong

```verilog
always @(*)
    if (clear)
        y = 1'b0;
    else
        y = a & b;
```

Correct



```
[Synth 8-3352] multi-driven net \PIO_CH[1]_RREADY with 2nd driver pin 'GND' [fxt_dma_packer.v:65]
[Synth 8-3352] multi-driven net \PIO_CH[0]_RREADY with 1st driver pin 'i_7/O' [fxt_dma_packer.v:65]
[Synth 8-3352] multi-driven net \PIO_CH[0]_RREADY with 2nd driver pin 'GND' [fxt_dma_packer.v:65]
[Synth 8-3352] multi-driven net \DMA_CH[3]_WREADY with 1st driver pin 'i_8/O' [fxt_dma_packer.v:65]
[Synth 8-3352] multi-driven net \DMA_CH[3]_WREADY with 2nd driver pin 'GND' [fxt_dma_packer.v:65]
[Synth 8-3352] multi-driven net \DMA_CH[2]_WREADY with 1st driver pin 'i_9/O' [fxt_dma_packer.v:65]
```

Incomplete Branch and Incomplete Output Assignment

```
always @(*)
    if (a > b)       // eq is not assigned in this branch
        gt = 1'b1;
    else if (a == b) // gt is not assigned in this branch
        eq = 1'b1;
                     // final else branch is omitted
```

Wrong

```
always @(*)
    if (a > b)
    begin
        gt = 1'b1;
        eq = 1'b0;
    end
    else if (a == b)
    begin
        gt = 1'b0;
        eq = 1'b1;
    end
    else // i.e., a < b
    begin
        gt = 1'b0;
        eq = 1'b0;
    end
```

Correct

Incomplete Branch and Incomplete Output Assignment

```
always @(*)
    if (a > b)
    begin
        gt = 1'b1;
        eq = 1'b0;
    end
    else if (a == b)
    begin
        gt = 1'b0;
        eq = 1'b1;
    end
    else // i.e., a < b
    begin
        gt = 1'b0;
        eq = 1'b0;
    end
```

Correct

```
always @(*)
begin
    gt = 1'b0; // Default value for gt
    eq = 1'b0; // Default value for eq
    if (a > b)
        gt = 1'b1;
    else if (a == b)
        eq = 1'b1;
end
```

Correct (alternative)

Incomplete Branch and Incomplete Output Assignment

```
reg [1:0] s;

always @(*)
begin
    case (s)
        2'b00: y = 1'b1;
        2'b10: y = 1'b0;
        2'b11: y = 1'b1;
    endcase
end
```

Wrong

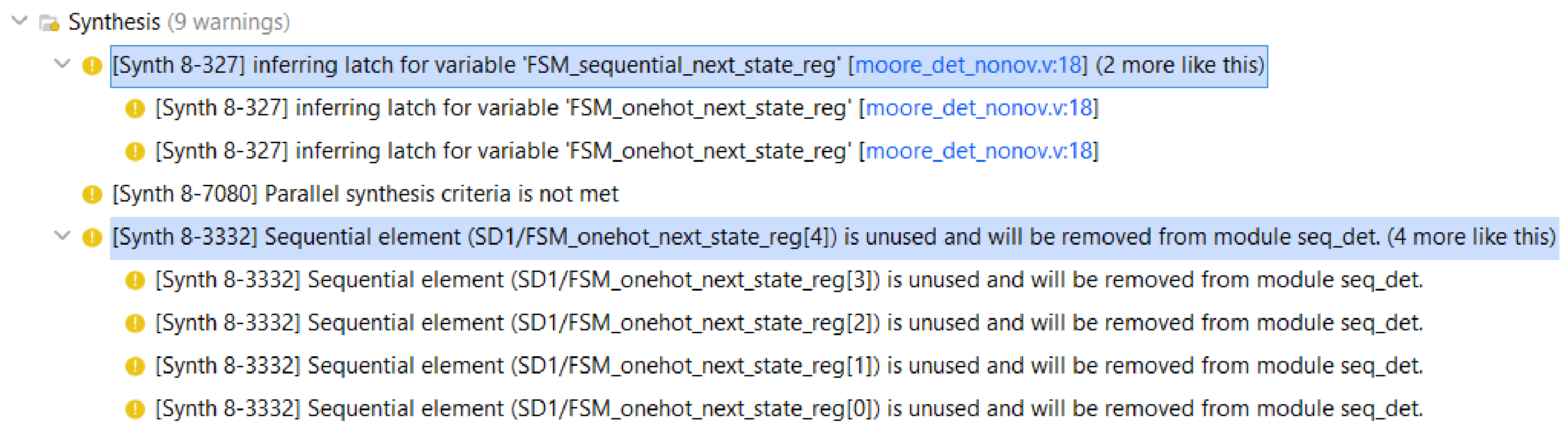
```
always @(*)
begin
    case (s)
        2'b00: y = 1'b1;
        2'b10: y = 1'b0;
        2'b11: y = 1'b1;
        default: y = 1'b0;
    endcase
end
```

Correct

```
always @(*)
begin
    y = 1'b0; // Default value for y
    case (s)
        2'b00: y = 1'b1;
        2'b10: y = 1'b0;
        2'b11: y = 1'b1;
    endcase
end
```

Correct

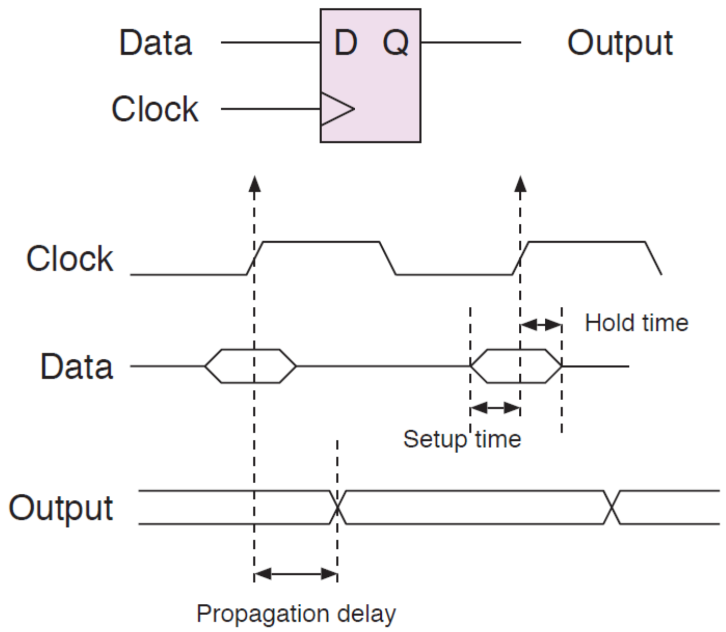
Incomplete Branch and Incomplete Output Assignment

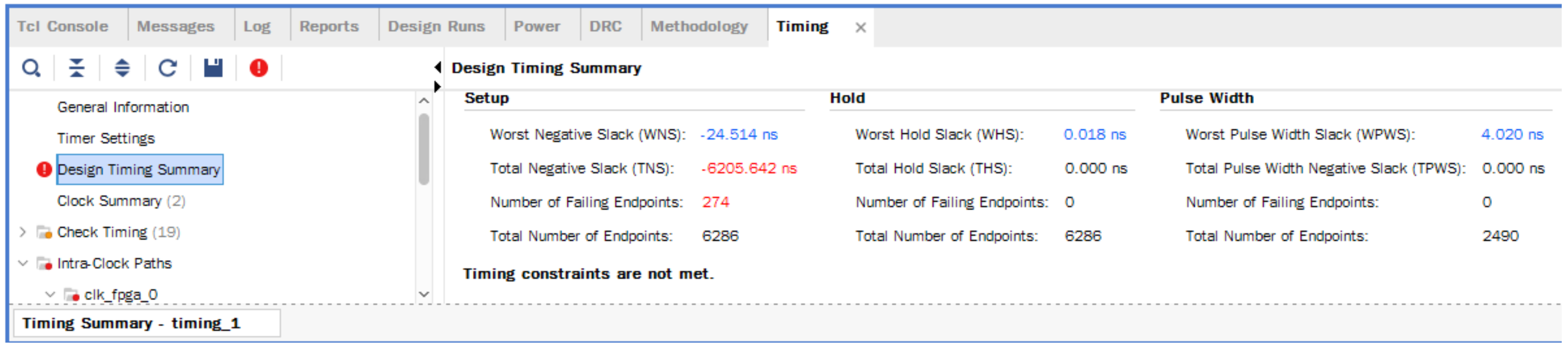


```
Synthesis (9 warnings)
  [Synth 8-327] inferring latch for variable 'FSM_sequential_next_state_reg' [moore_det_nonov.v:18] (2 more like this)
    [Synth 8-327] inferring latch for variable 'FSM_onehot_next_state_reg' [moore_det_nonov.v:18]
    [Synth 8-327] inferring latch for variable 'FSM_onehot_next_state_reg' [moore_det_nonov.v:18]
  [Synth 8-7080] Parallel synthesis criteria is not met
  [Synth 8-3332] Sequential element (SD1/FSM_onehot_next_state_reg[4]) is unused and will be removed from module seq_det. (4 more like this)
    [Synth 8-3332] Sequential element (SD1/FSM_onehot_next_state_reg[3]) is unused and will be removed from module seq_det.
    [Synth 8-3332] Sequential element (SD1/FSM_onehot_next_state_reg[2]) is unused and will be removed from module seq_det.
    [Synth 8-3332] Sequential element (SD1/FSM_onehot_next_state_reg[1]) is unused and will be removed from module seq_det.
    [Synth 8-3332] Sequential element (SD1/FSM_onehot_next_state_reg[0]) is unused and will be removed from module seq_det.
```

A design that has <u>multi-driver nets and/or inferred</u> latches may work in RTL simulation but will not work on an FPGA.

Setup and Hold Time



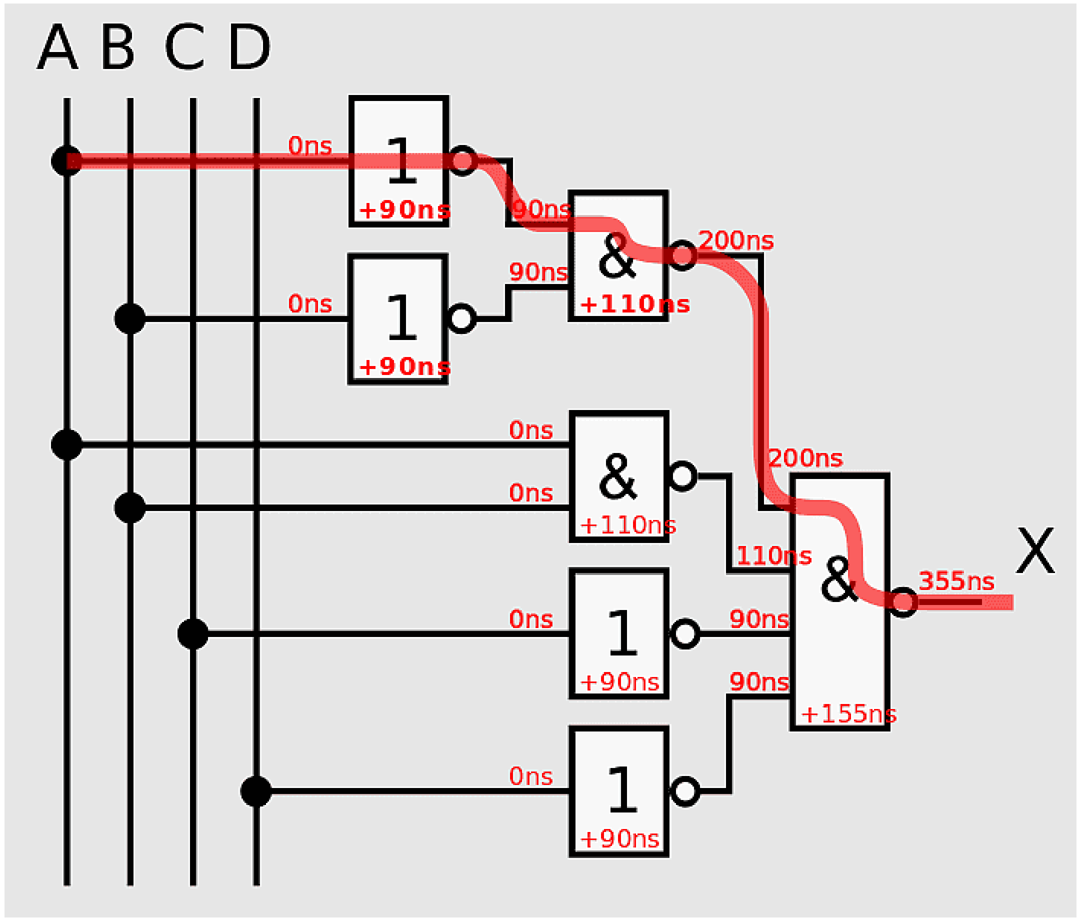


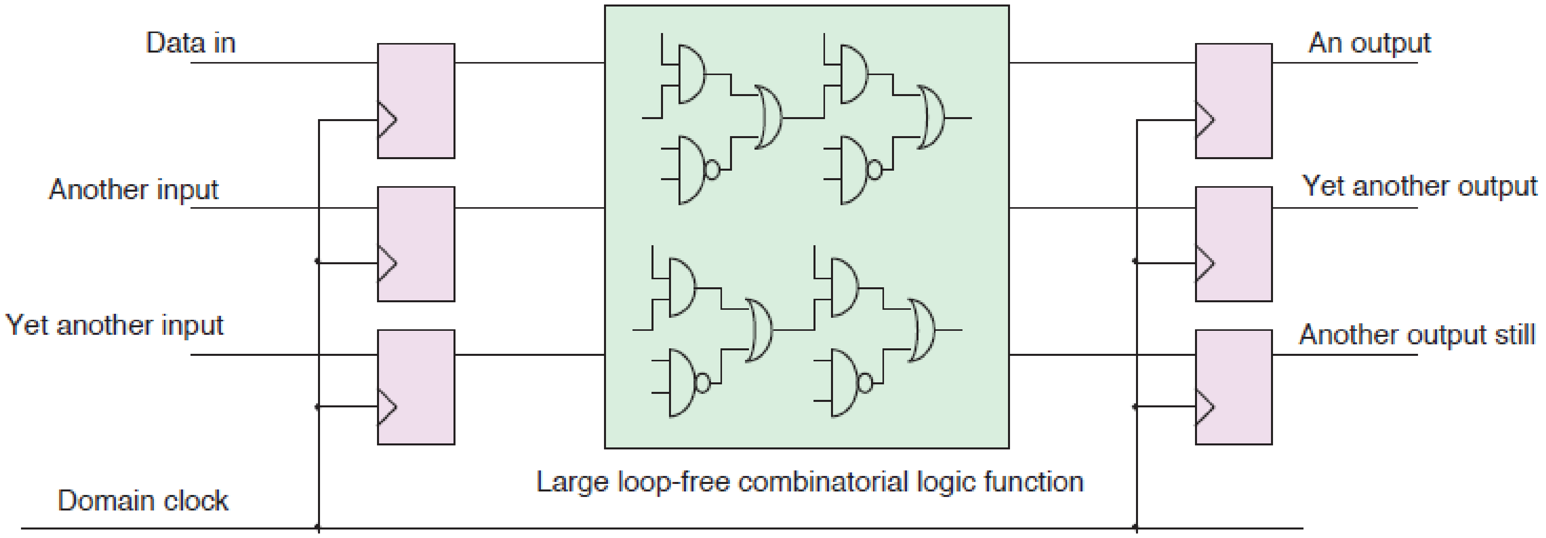
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-24.514 ns	Worst Hold Slack (WHS):	0.018 ns	Worst Pulse Width Slack (WPWS):	4.020 ns
Total Negative Slack (TNS):	-6205.642 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	274	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	6286	Total Number of Endpoints:	6286	Total Number of Endpoints:	2490

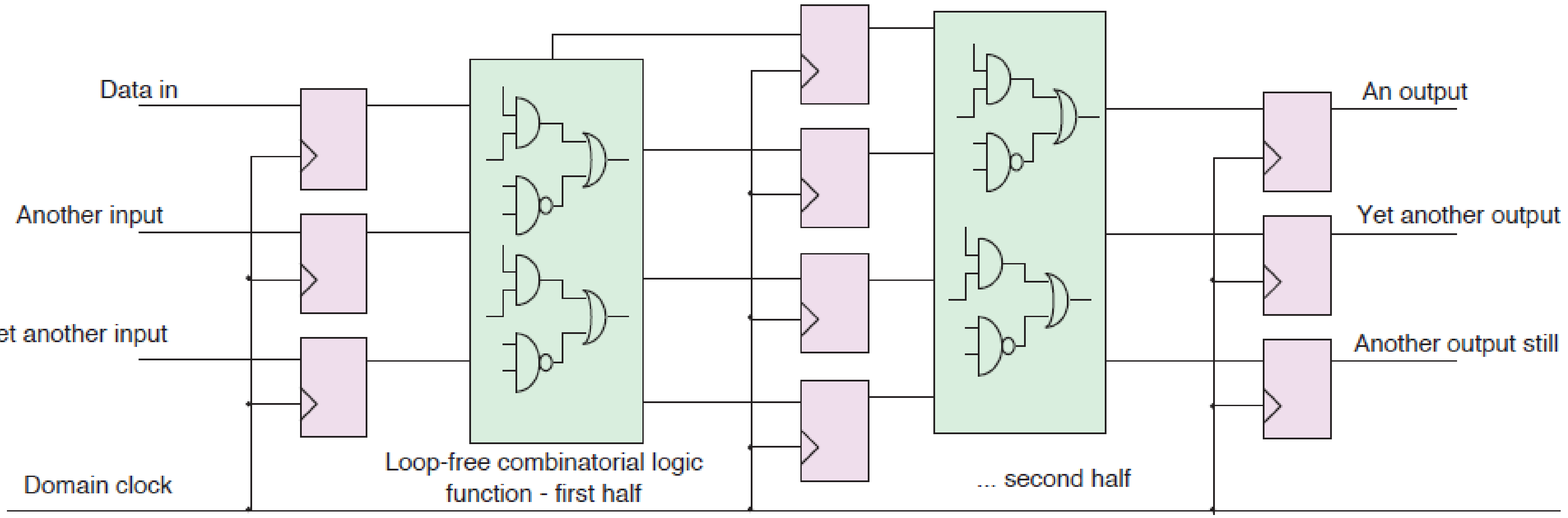
Timing constraints are not met.

Setup and Hold Time





Desired logic function



Same logic function - pipelined version.

Conlcusion

- Combinational examples
- Sequential examples
- Common errors