

Dennis Lang

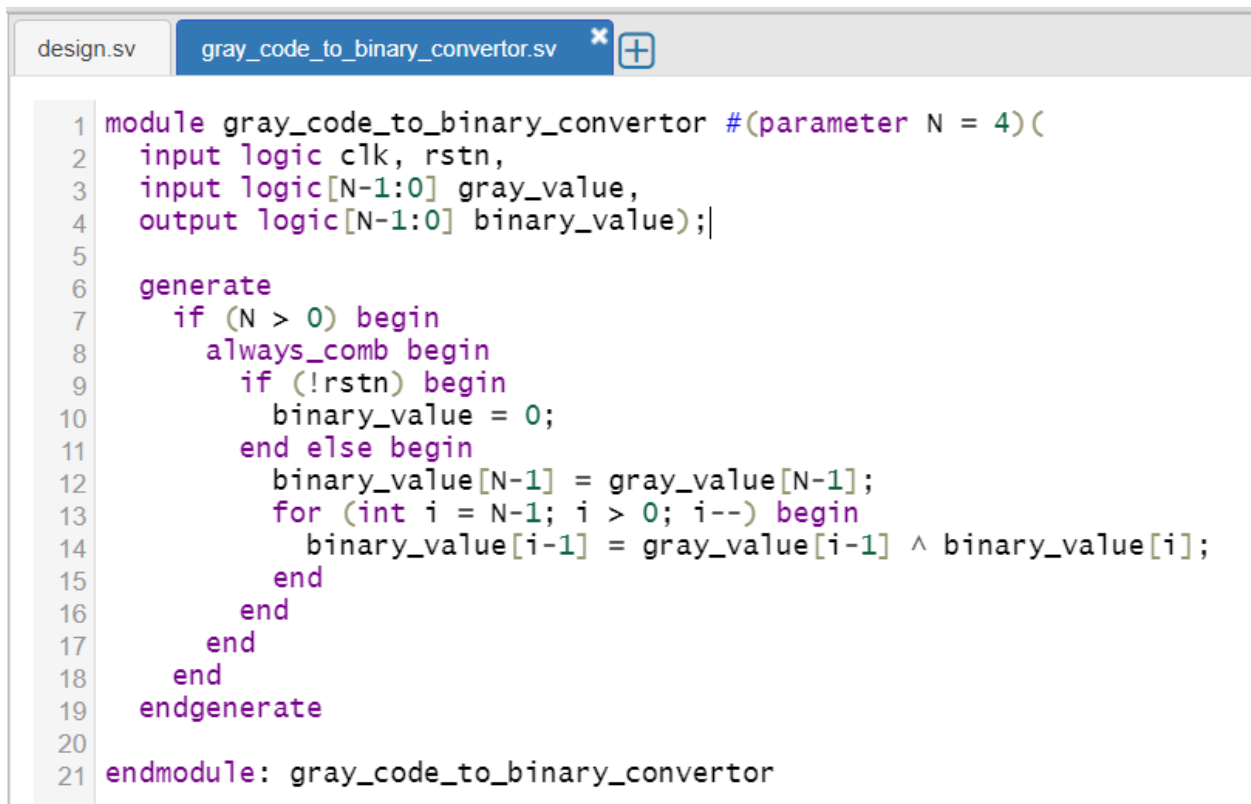
ECE 111 Winter 2024

Professor John Eldon

3 March 2024

## Lab 6

### Part A - Gray Code



The screenshot shows a code editor with two tabs: 'design.sv' and 'gray\_code\_to\_binary\_convertor.sv'. The code in the active tab is a Verilog module for converting Gray code to binary. It includes a parameter N=4, inputs for clock (clk), reset (rstn), and Gray code (gray\_value), and an output for binary code (binary\_value). The logic uses a generate block with an if statement for N > 0, an always\_comb block, and a for loop to calculate the binary value from the Gray code.

```
1 module gray_code_to_binary_convertor #(parameter N = 4)(
2   input logic clk, rstn,
3   input logic[N-1:0] gray_value,
4   output logic[N-1:0] binary_value);
5
6   generate
7     if (N > 0) begin
8       always_comb begin
9         if (!rstn) begin
10          binary_value = 0;
11        end else begin
12          binary_value[N-1] = gray_value[N-1];
13          for (int i = N-1; i > 0; i--) begin
14            binary_value[i-1] = gray_value[i-1] ^ binary_value[i];
15          end
16        end
17      end
18    end
19  endgenerate
20
21 endmodule: gray_code_to_binary_convertor
```

## Edited Testbench - Gray Code

testbench.sv    gray\_code\_to\_binary\_convertor\_testbench.sv

SV/Verilog Testb

```
1 //gray code to binary convertor testbench code
2 `timescale 1ns/1ns
3 module gray_code_to_binary_convertor_testbench;
4 parameter N=4;
5 logic clock, rstn;
6 logic [N-1:0] gray_value, binary_value;
7
8 // Instantiate design under test
9 gray_code_to_binary_convertor #(N(N)) design_instance(
10 .clk(clock),
11 .rstn(rstn),
12 .gray_value(gray_value),
13 .binary_value(binary_value)
14 );
15
16 initial begin
17     $dumpfile("dump.vcd"); $dumpvars;
18
19     // Initialize Inputs
20     rstn = 0;
21     clock = 0;
22
23     // Wait 20 ns for global reset to finish and start counter
24     #20;
25     rstn = 1;
26
27     // Drive gray value
28     for(int i=0; i<16; i++) begin
29         #20;
30         gray_value = i;
31     end
32
33     // Wait for 20ns
34     #20ns;
35     rstn = 0;
36     #20ns;
37     // terminate simulation
38     $finish();
39 end
40
41 // Clock generator logic
42 always@(clock) begin
43     #10ns clock <= !clock;
44 end
45
46     always@(posedge clock or rstn) begin
47         $display(" time=%0t, reset_n=%b clk=%b gray_value=%b
48 binary_value=%b ", $time, rstn, clock, gray_value, binary_value);
49     end
50
51 endmodule
```

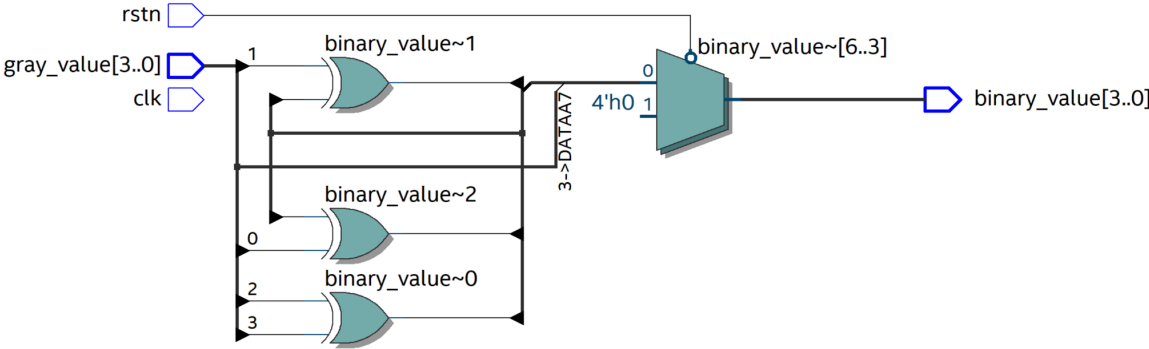
# Synthesis Resource Usage + RTL Netlist Schematic (Gray)

Compilation Report - gray\_code\_to\_binary\_convertor

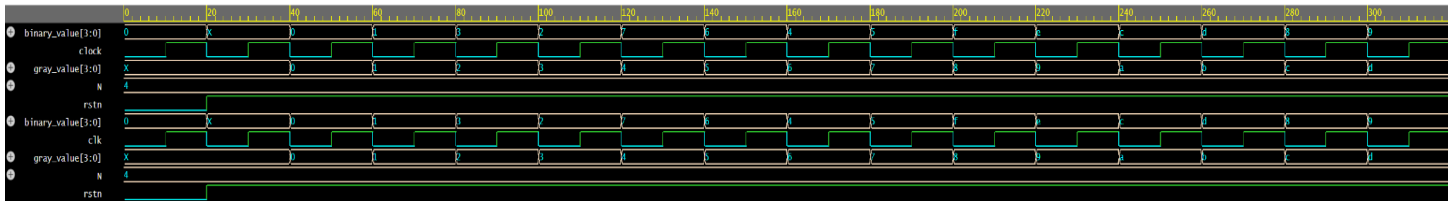
Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated ALUTs Used	4
1	-- Combinational ALUTs	4
2	-- Memory ALUTs	0
3	-- LUT_REGS	0
2	Dedicated logic registers	0
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	4
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	1
4	-- 4 input functions	1
5	-- <=3 input functions	2
8		
9	Combinational ALUTs by mode	
1	-- normal mode	4
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	4
12		
13	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
3	-- LUT_REGS	0
14		
15		
16	I/O pins	10
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	gray_value[3]-input
21	Maximum fan-out	4
22	Total fan-out	28
23	Average fan-out	1.17



## Simulation + Explanation (Gray)



In the Resource Usage we can see that there are 4 combinational functions and 4 ALUTs with my three XORs as expected, as seen in the RTL netlist. The simulation waveform shows the binary outputs match the expected input in gray code, as does the result below.

```

# Loading sv_std.std
# Loading work.gray_to_binary_convertor_testbench(fast)
#
# run -all
# time=0, reset_n=0 clk=0 gray_value=xxxx binary_value=xxxx
# time=10, reset_n=0 clk=1 gray_value=xxxx binary_value=0000
# time=20, reset_n=1 clk=1 gray_value=xxxx binary_value=0000
# time=30, reset_n=1 clk=1 gray_value=xxxx binary_value=xxxx
# time=50, reset_n=1 clk=1 gray_value=0000 binary_value=0000
# time=70, reset_n=1 clk=1 gray_value=0001 binary_value=0001
# time=90, reset_n=1 clk=1 gray_value=0010 binary_value=0011
# time=110, reset_n=1 clk=1 gray_value=0011 binary_value=0010
# time=130, reset_n=1 clk=1 gray_value=0100 binary_value=0111
# time=150, reset_n=1 clk=1 gray_value=0101 binary_value=0110
# time=170, reset_n=1 clk=1 gray_value=0110 binary_value=0100
# time=190, reset_n=1 clk=1 gray_value=0111 binary_value=0101
# time=210, reset_n=1 clk=1 gray_value=1000 binary_value=1111
# time=230, reset_n=1 clk=1 gray_value=1001 binary_value=1110
# time=250, reset_n=1 clk=1 gray_value=1010 binary_value=1100
# time=270, reset_n=1 clk=1 gray_value=1011 binary_value=1101
# time=290, reset_n=1 clk=1 gray_value=1100 binary_value=1000
# time=310, reset_n=1 clk=1 gray_value=1101 binary_value=1001
# time=330, reset_n=1 clk=1 gray_value=1110 binary_value=1011
# time=350, reset_n=1 clk=1 gray_value=1111 binary_value=1010
# time=360, reset_n=0 clk=1 gray_value=1111 binary_value=1010
# time=370, reset_n=0 clk=1 gray_value=1111 binary_value=0000
# ** Note: $finish : gray_code_to_binary_convertor_testbench.sv(39)
# Time: 380 ns Iteration: 0 Instance: /gray_to_binary_convertor_testbench
# End time: 04:49:34 on Mar 04,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# *** Summary *****
# qrun: Errors: 0, Warnings: 0
# vlog: Errors: 0, Warnings: 5
# vopt: Errors: 0, Warnings: 0
# vsim: Errors: 0, Warnings: 0
# Totals: Errors: 0, Warnings: 5

```

Done

## Part B - N-bit Carry Lookahead Adder Code

```
design.sv  carry_lookahead_adder.sv  fulladder.sv  +
1  `include "fulladder.sv"
2
3  module carry_lookahead_adder #(parameter N = 4)(
4      input logic [N-1:0] A, B,
5      input logic CIN,
6      output logic [N:0] result
7  );
8
9      logic [N-1:0] sum;
10     logic [N:0] carry;
11
12     assign carry[0] = CIN;
13     genvar j;
14     generate
15         for (j = 0; j < N; j++) begin : carry_logic
16             assign carry[j+1] = (((A[j] ^ B[j]) & carry[j]) | (A[j] & B[j]));
17         end
18     endgenerate
19
20
21     // Instantiate four full adders
22     genvar i;
23     generate
24         for (i = 0; i < N; i++) begin : full_adder_instances
25             fulladder #(
26                 // You can customize parameters for the full adder here
27             ) fa_inst (
28                 .a(A[i]),
29                 .b(B[i]),
30                 .cin(carry[i]),
31                 .sum(sum[i]),
32                 .cout()
33             );
34         end
35     endgenerate
36
37     // Connect the sum outputs of the full adders to the result
38     assign result[N:0] = sum;
39
40 endmodule : carry_lookahead_adder
41
```

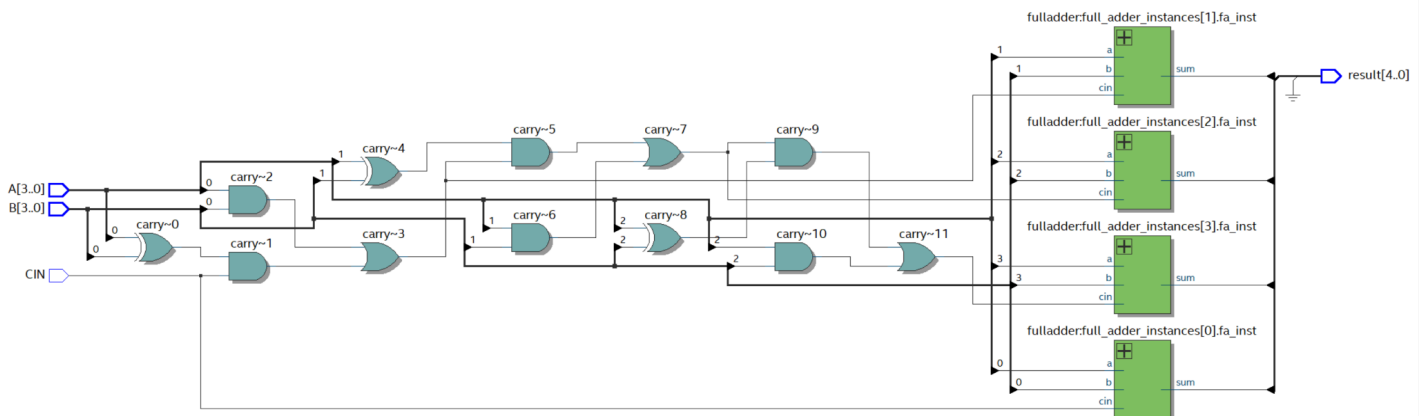
## Edited Testbench - Adder

testbench.sv    carry\_lookahead\_adder\_testbench.sv

```
1  `timescale 1ns/1ns
2  //Carry Lookahead Adder Testbench Code
3  module carry_lookahead_adder_testbench;
4      parameter N = 4;
5      logic[N-1:0] in0, in1;
6      logic carryin;
7      logic[N:0] sum;
8
9      // Instantiate design under test
10     carry_lookahead_adder #(N(N)) design_instance(
11         .A(in0),
12         .B(in1),
13         .CIN(carryin),
14         .result(sum)
15     );
16
17     initial begin
18
19         $dumpfile("dump.vcd"); $dumpvars;
20
21         // Initialize Inputs
22         in0=0;
23         in1=0;
24         carryin=0;
25         // Wait 100 ns
26         #100;
27         in0=2;
28         in1=1;
29         carryin=1;
30         #50
31         in0=1;
32         in1=1;
33         carryin=0;
34         #50
35         in0=2;
36         in1=2;
37         carryin=0;
38         #50
39         in0=3;
40         in1=1;
41         carryin=0;
42         #50;
43         in0=4;
44         in1=7;
45         carryin=1;
46         #50;
47         in0=15;
48         in1=2;
49         carryin=0;
50         #50;
51         in0=10;
52         in1=5;
53         carryin=0;
54         #50;
55     end
56
57     initial begin
58         $monitor(" time=%0t   A=%d   B=%d   CIN=%d\n", $time, in0, in1, carryin, sum);
59     end
60 endmodule
```

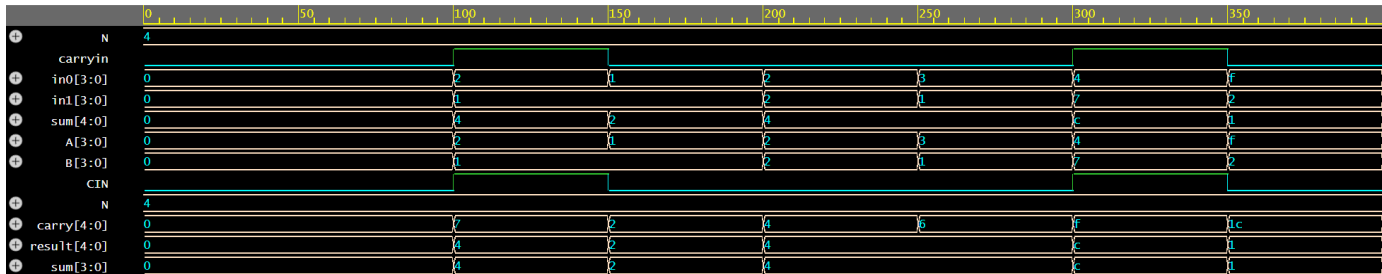
## Synthesis Resource Usage + RTL Netlist Schematic (Adder)

Analysis & Synthesis Resource Usage Summary		
	Resource	Usage
1	Estimated ALUTs Used	5
1	-- Combinational ALUTs	5
2	-- Memory ALUTs	0
3	-- LUT_REGS	0
2	Dedicated logic registers	0
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	5
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	3
4	-- 4 input functions	0
5	-- <=3 input functions	2
8		
9	Combinational ALUTs by mode	
1	-- normal mode	5
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	5
12		
13	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
3	-- LUT_REGS	0
14		
15		
16	I/O pins	14
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	A[0]~input
21	Maximum fan-out	3
22	Total fan-out	39
23	Average fan-out	1.18





## Simulation + Explanation (Adder)



In the Resource Usage we can see that there are 5 combinational functions and 5 ALUTs with my four full adders as expected, as seen in the RTL netlist. The simulation waveform shows the addition works including the CIN, as does the result below.

```

# Loading sv_std.std
# Loading work.carry_lookahead_adder_testbench(fast)
#
# run -all
# time=0   A= 0   B= 0   CIN=0   result= 0
#
# time=100  A= 2   B= 1   CIN=1   result= 4
#
# time=150  A= 1   B= 1   CIN=0   result= 2
#
# time=200  A= 2   B= 2   CIN=0   result= 4
#
# time=250  A= 3   B= 1   CIN=0   result= 4
#
# time=300  A= 4   B= 7   CIN=1   result=12
#
# time=350  A=15   B= 2   CIN=0   result= 1
#
# time=400  A=10   B= 5   CIN=0   result=15
#
# exit
# End time: 05:44:03 on Mar 04,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
# *** Summary ****
#   qrun: Errors:    0, Warnings:    0
#   vlog: Errors:    0, Warnings:    0
#   vopt: Errors:    0, Warnings:    0
#   vsim: Errors:    0, Warnings:    0
# Totals: Errors:    0, Warnings:    0
Finding VCD file...
./dump.vcd
[2024-03-04 10:44:03 UTC] Opening EPWave...
Done

```

## Lab 6

### Part C - Clock Divide By 3 Code

design.sv

clock\_divide\_by\_3.sv



```
1 module clock_divide_by_3 (  
2     input logic clkkin, reset,  
3     output logic clkout  
4 );  
5  
6     // Internal counter variables  
7     logic [1:0] counter;  
8     logic counter_plus_clkkin;  
9  
10    // Always block for the counter logic  
11    always_ff @(posedge clkkin or posedge reset) begin  
12        if (reset) begin  
13            // Reset the counter to 0  
14            counter <= 2'b00;  
15        end else begin  
16            // Increment the counter with modulo-3 behavior  
17            counter <= (counter == 2'b10) ? 2'b00 : counter + 1;  
18        end  
19    end  
20  
21    // Half clock cycle delay for counter[0]  
22    always_ff @(negedge clkkin) begin  
23        counter_plus_clkkin <= counter[0];  
24    end  
25  
26    // Assign the output clock signal  
27    assign clkout = counter_plus_clkkin | counter[1];  
28  
29 endmodule  
30
```

## Edited Testbench - Clock

testbench.sv

clock\_divide\_by\_3\_testbench.sv

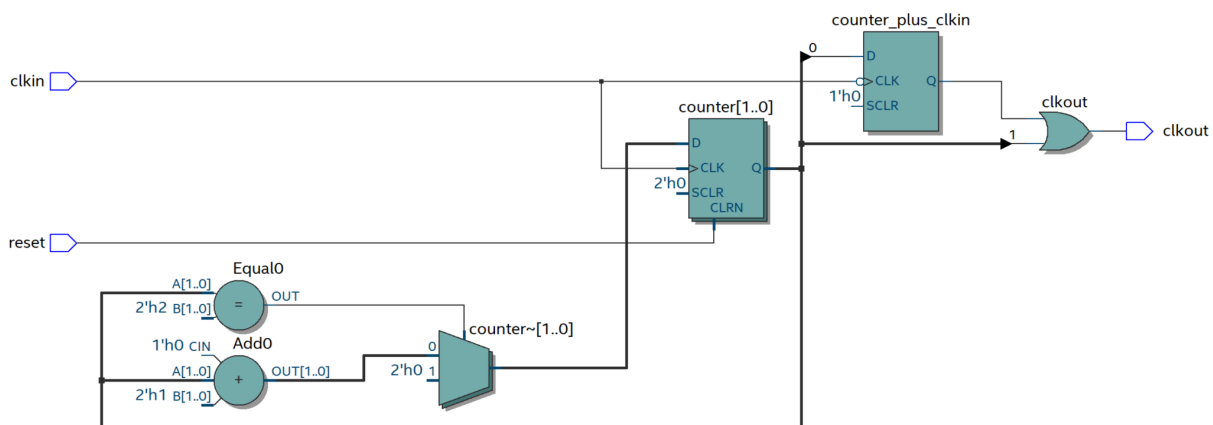


SV/Verilog Tes

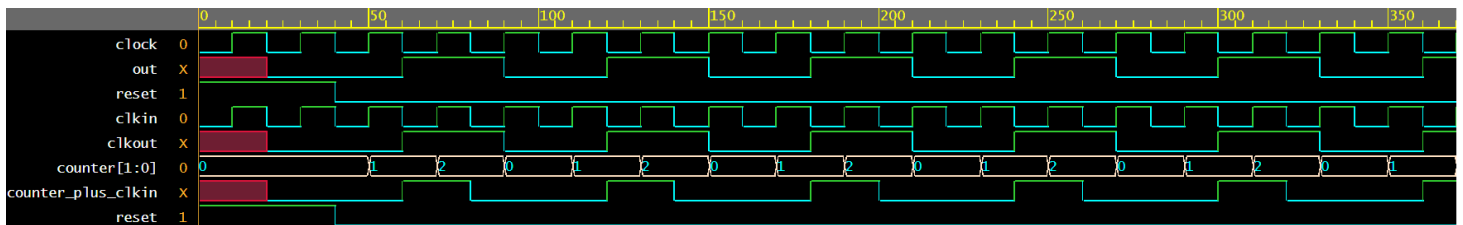
```
1 //Clock divide by 4 testbench
2 `timescale 1ns/1ns
3 module clock_divide_by_3_testbench;
4 logic clock, reset, out;
5
6 // Instantiate design under test
7 clock_divide_by_3 design_instance(
8 .clkkin(clock),
9 .reset(reset),
10 .clkout(out)
11 );
12
13 initial begin
14     $dumpfile("dump.vcd"); $dumpvars;
15
16 // Initialize Inputs
17 reset = 1;
18 clock = 0;
19
20 // Wait 40 ns for global reset to finish and start counter
21 #40;
22 reset = 0;
23
24 // Wait for some time
25 #340ns;
26
27 // terminate simulation
28 $finish();
29 end
30
31 // Clock generator logic
32 always@(clock) begin
33     #10ns clock <= !clock;
34 end
35
36 always@(posedge clock or reset) begin
37     $display(" time=%0t, reset=%b clockin=%b clockout=%b ",
38 $time, reset, clock, out);
39 end
40
41 always@(negedge clock or reset) begin
42     $display(" time=%0t, reset=%b clockin=%b clockout=%b ",
43 $time, reset, clock, out);
44 end
45
46 endmodule
```

### Synthesis Resource Usage + RTL Netlist Schematic (Clock)

Analysis & Synthesis Resource Usage Summary		
	Resource	Usage
1	Estimated ALUTs Used	3
1	-- Combinational ALUTs	3
2	-- Memory ALUTs	0
3	-- LUT_REGS	0
2	Dedicated logic registers	3
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	3
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	0
5	-- <=3 input functions	3
8		
9	Combinational ALUTs by mode	
1	-- normal mode	3
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	4
12		
13	Total registers	3
1	-- Dedicated logic registers	3
2	-- I/O registers	0
3	-- LUT_REGS	0
14		
15		
16	I/O pins	3
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	counter[1]
21	Maximum fan-out	3
22	Total fan-out	18
23	Average fan-out	1.50



## Simulation + Explanation (Clock)



In the Resource Usage we can see that there are 3 combinational functions, 3 ALUTs, and 3 registers with my flip-flop and modulo-3 as expected, as seen in the RTL netlist. The simulation waveform shows the output takes one cycle for three clkkin cycles, as does the result below.

```

# Loading sv_std.std
# Loading work.clock_divide_by_3_testbench(fast)
#
# run -all
# time=0, reset=1 clockin=0 clockout=x
# time=0, reset=1 clockin=0 clockout=x
# time=10, reset=1 clockin=1 clockout=x
# time=20, reset=1 clockin=0 clockout=x
# time=30, reset=1 clockin=1 clockout=0
# time=40, reset=0 clockin=1 clockout=0
# time=40, reset=0 clockin=1 clockout=0
# time=40, reset=0 clockin=0 clockout=0
# time=50, reset=0 clockin=1 clockout=0
# time=60, reset=0 clockin=0 clockout=0
# time=70, reset=0 clockin=1 clockout=1
# time=80, reset=0 clockin=0 clockout=1
# time=90, reset=0 clockin=1 clockout=1
# time=100, reset=0 clockin=0 clockout=0
# time=110, reset=0 clockin=1 clockout=0
# time=120, reset=0 clockin=0 clockout=0
# time=130, reset=0 clockin=1 clockout=1
# time=140, reset=0 clockin=0 clockout=1
# time=150, reset=0 clockin=1 clockout=1
# time=160, reset=0 clockin=0 clockout=0
# time=170, reset=0 clockin=1 clockout=0
# time=180, reset=0 clockin=0 clockout=0
# time=190, reset=0 clockin=1 clockout=1
# time=200, reset=0 clockin=0 clockout=1
# time=210, reset=0 clockin=1 clockout=1
# time=220, reset=0 clockin=0 clockout=0
# time=230, reset=0 clockin=1 clockout=0
# time=240, reset=0 clockin=0 clockout=0
# time=250, reset=0 clockin=1 clockout=1
# time=260, reset=0 clockin=0 clockout=1
# time=270, reset=0 clockin=1 clockout=1
# time=280, reset=0 clockin=0 clockout=0
# time=290, reset=0 clockin=1 clockout=0
# time=300, reset=0 clockin=0 clockout=0
# time=310, reset=0 clockin=1 clockout=1
# time=320, reset=0 clockin=0 clockout=1
# time=330, reset=0 clockin=1 clockout=1
# time=340, reset=0 clockin=0 clockout=0
# time=350, reset=0 clockin=1 clockout=0
# time=360, reset=0 clockin=0 clockout=0
# time=370, reset=0 clockin=1 clockout=1
# ** Note: $finish : clock_divide_by_3_testbench.sv(29)
# Time: 380 ns Iteration: 0 Instance: /clock_divide_by_3_testbench
# End time: 08:57:01 on Mar 04,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# *** Summary *****
# qrun: Errors: 0, Warnings: 0
# vlog: Errors: 0, Warnings: 0
# vopt: Errors: 0, Warnings: 0
# vsim: Errors: 0, Warnings: 0
# Totals: Errors: 0, Warnings: 0
Finding VCD file...
./dump.vcd
[2024-03-04 13:57:02 UTC] Opening EPWave...
Done

```