ASYNC FIFO USING CDC PROJECT

USING VERILOG

Marwan Khaled Mohamed

Contents

1 Design Architecture Overview

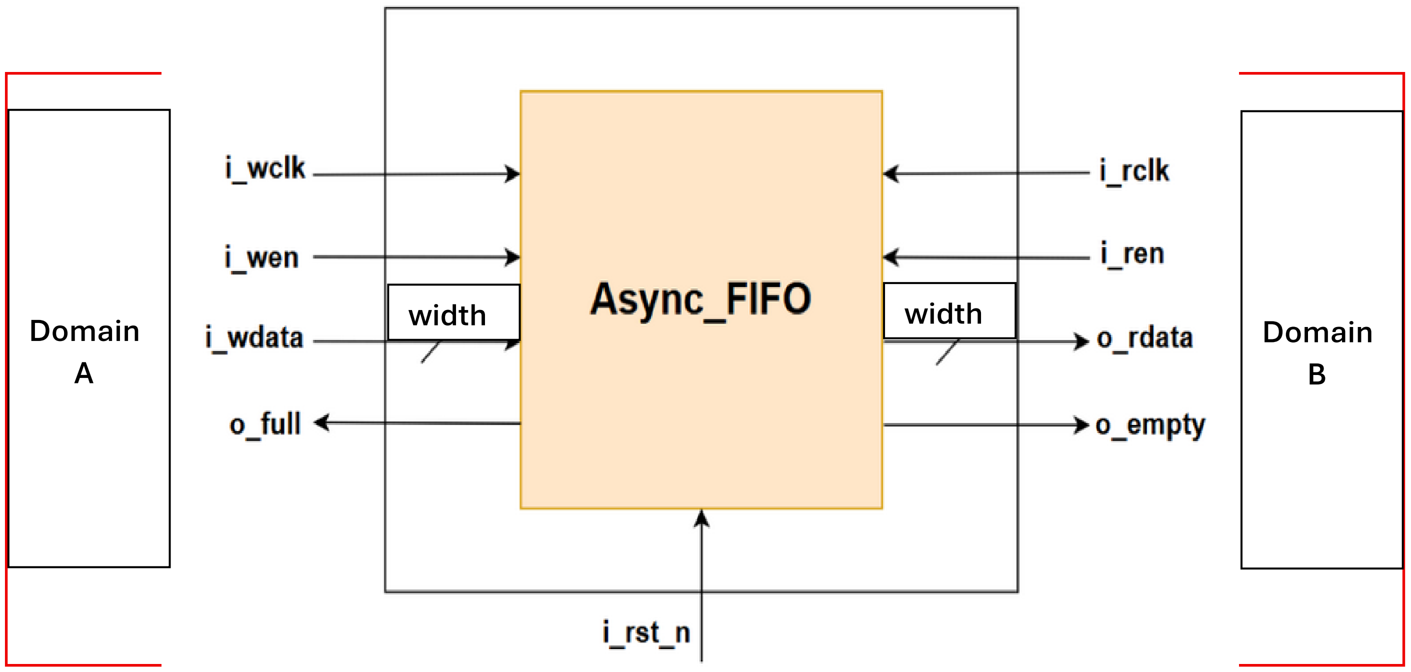
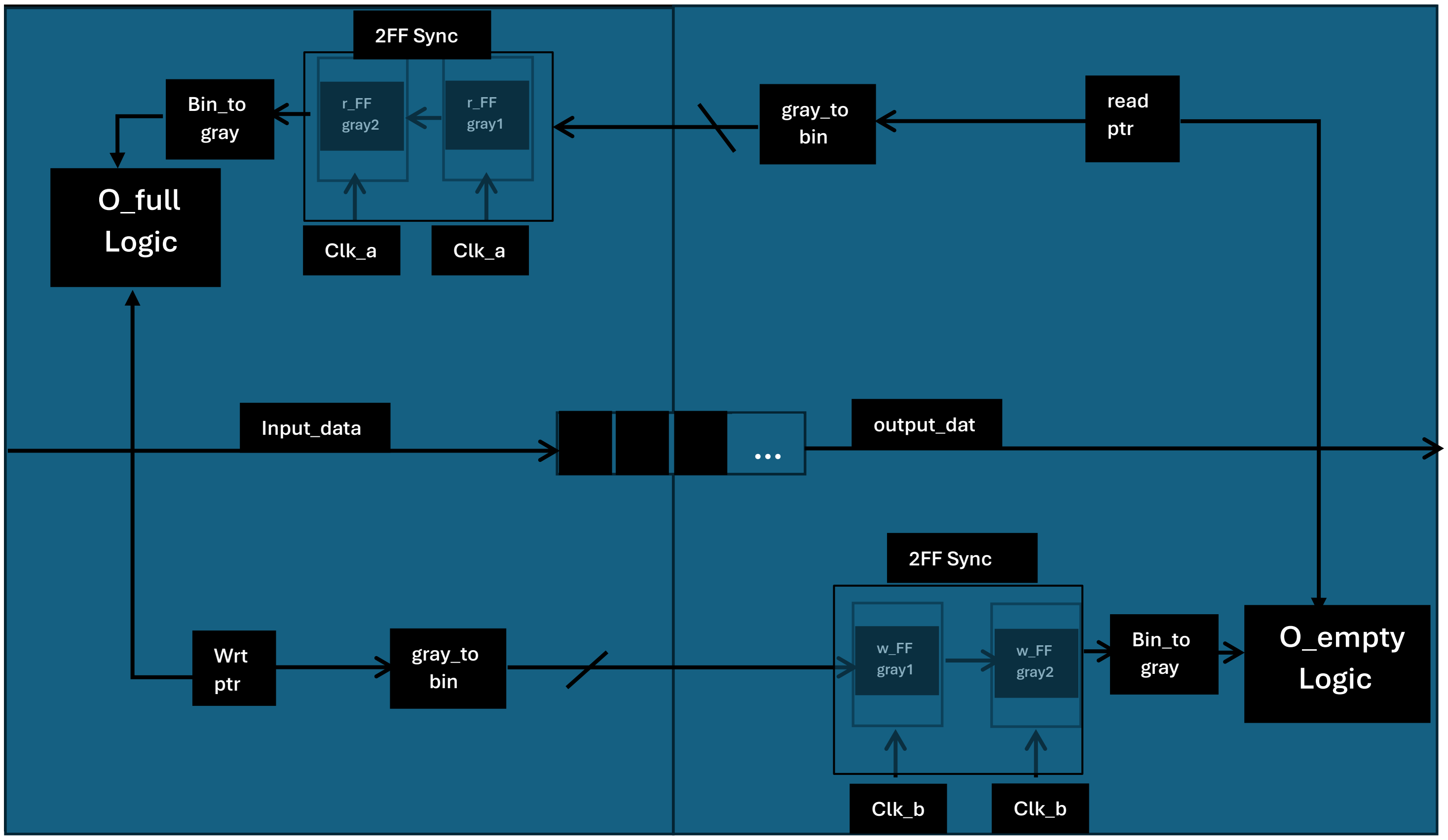


Figure 1:Design Architecture

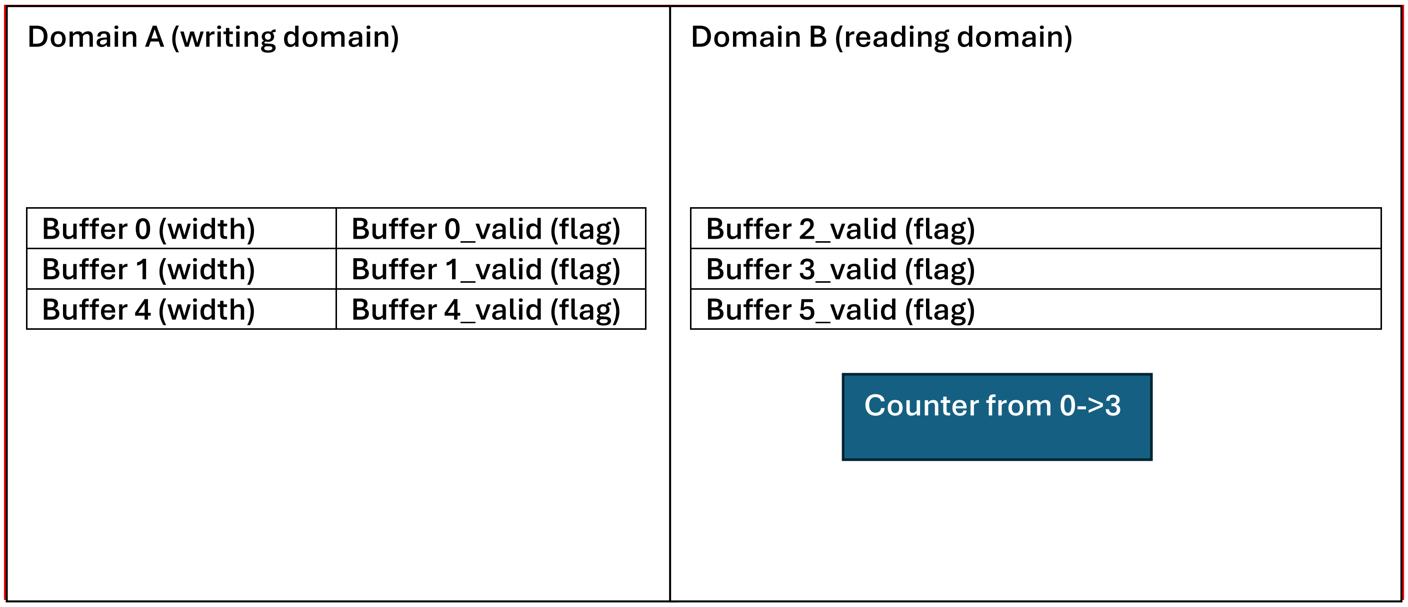
2 Detailed Architecture for FIFO



I made two configs for CDC :

1- FIFO to handle the different rates
2- Gray coding for data incoherence

3 Extra blocks

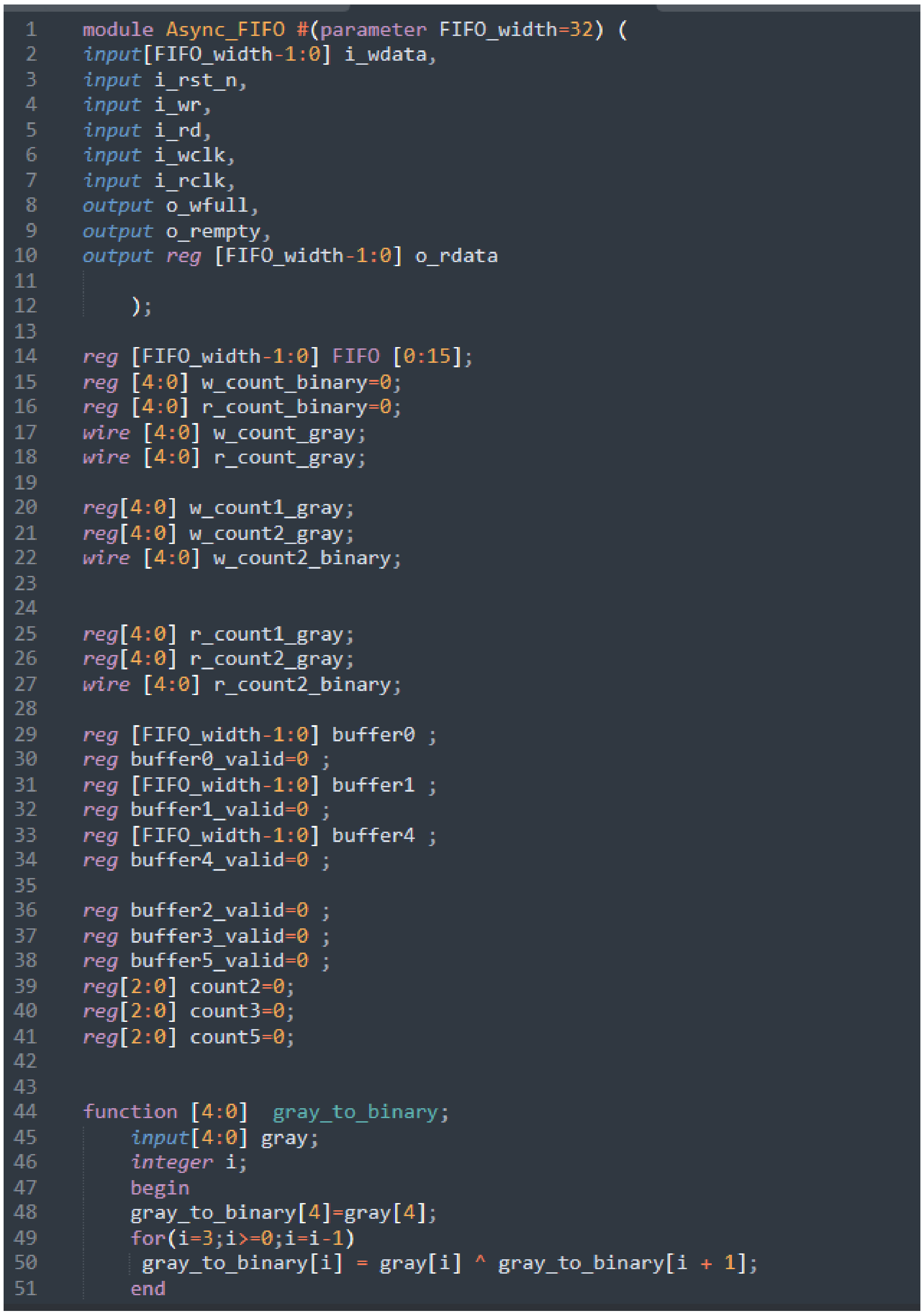


Because of 2FF sync we have delay and because of this delay we have 3 important corner cases and I have added these 2 blocks to handle the problem
1-small buffer in domain A: to handle the corner case when you want to write in domain A but domain A thinks the FIFO is full while there's a place empty because you also have read from domain b but domain A wants 2 period of delays to sense the change so I put 3 elements buffer to save this elements temporary until the delays end

2- some flags in domain B: to handle the second corner case when you want to read from FIFO But domain B thinks that it's empty while you have just written a new element in domain A but the change needs 2 clock cycle to propagate so I rise the flag until the change arrive

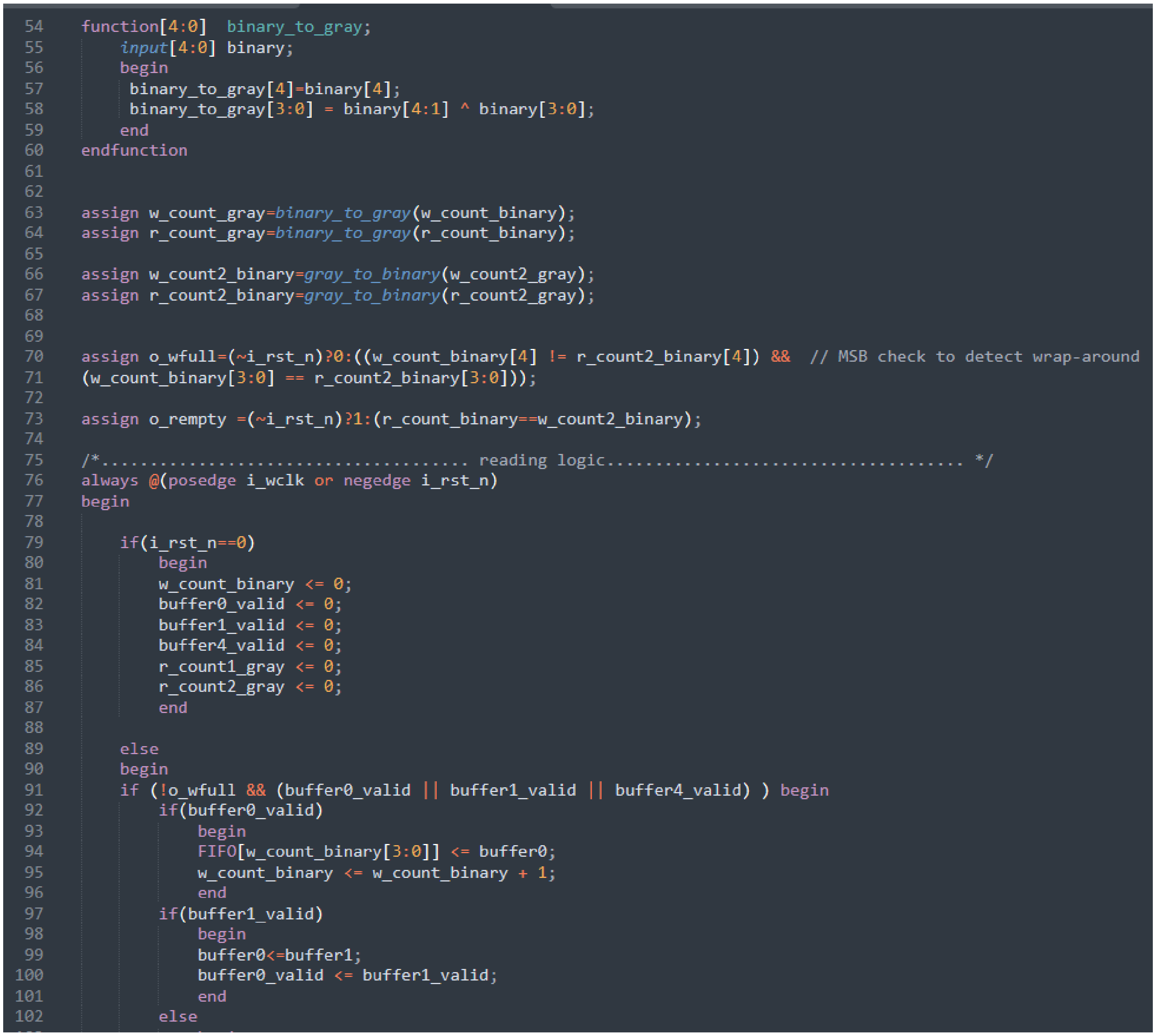
3- counter in domain B: to handle the last corner case when you want to read you wait for 2 or 3 cycles if FIFO still empty that means no one wrote in domain A so noting to read

4 Design FIFO code



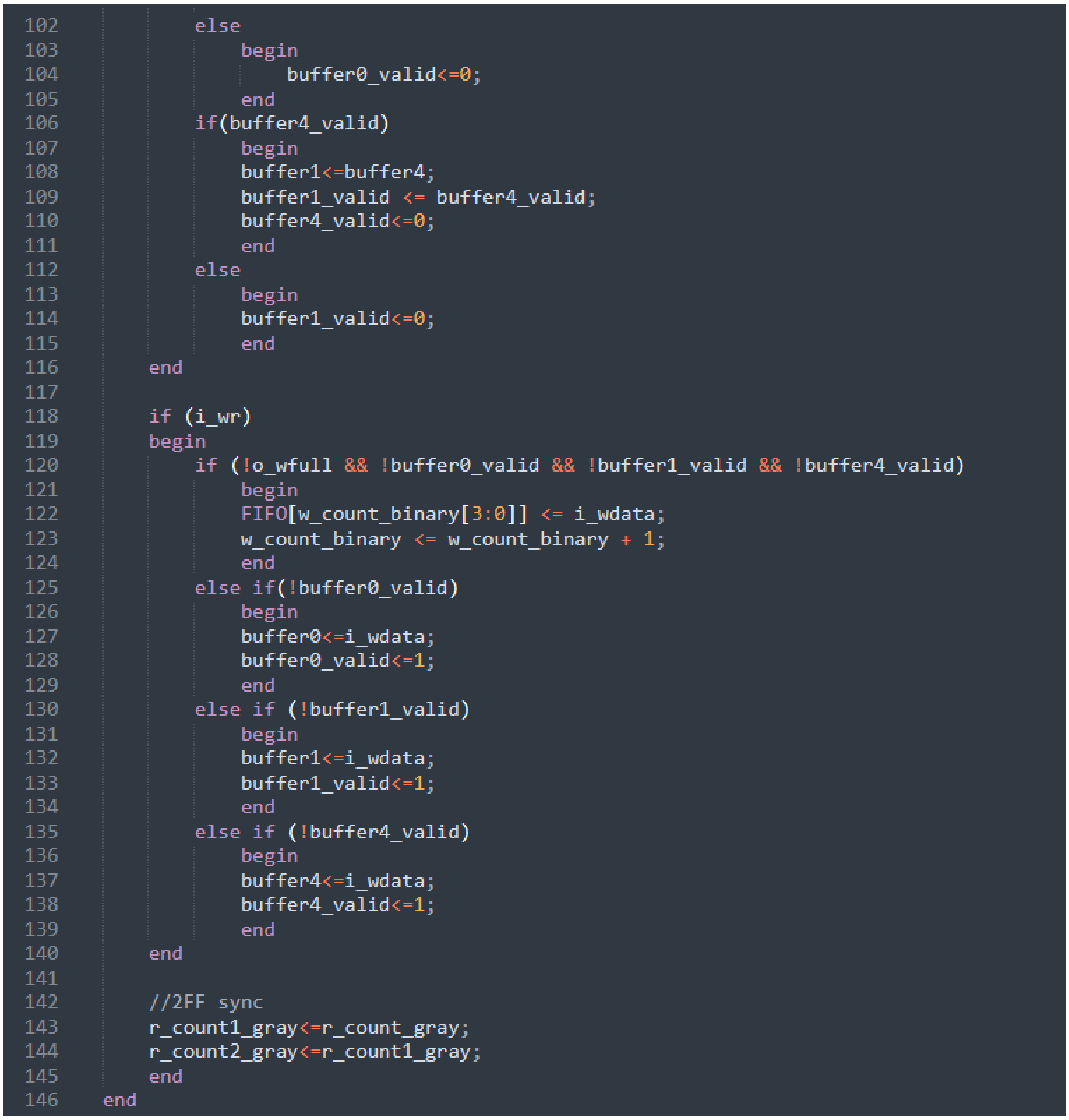
```verilog
module Async_FIFO #(parameter FIFO_width=32) (
input[FIFO_width-1:0] i_wdata,
input i_rst_n,
input i_wr,
input i_rd,
input i_wclk,
input i_rclk,
output o_wfull,
output o_rempty,
output reg [FIFO_width-1:0] o_rdata

    );

reg [FIFO_width-1:0] FIFO [0:15];
reg [4:0] w_count_binary=0;
reg [4:0] r_count_binary=0;
wire [4:0] w_count_gray;
wire [4:0] r_count_gray;

reg[4:0] w_count1_gray;
reg[4:0] w_count2_gray;
wire [4:0] w_count2_binary;


reg[4:0] r_count1_gray;
reg[4:0] r_count2_gray;
wire [4:0] r_count2_binary;

reg [FIFO_width-1:0] buffer0 ;
reg buffer0_valid=0 ;
reg [FIFO_width-1:0] buffer1 ;
reg buffer1_valid=0 ;
reg [FIFO_width-1:0] buffer4 ;
reg buffer4_valid=0 ;

reg buffer2_valid=0 ;
reg buffer3_valid=0 ;
reg buffer5_valid=0 ;
reg[2:0] count2=0;
reg[2:0] count3=0;
reg[2:0] count5=0;


function [4:0]  gray_to_binary;
    input[4:0] gray;
    integer i;
    begin
    gray_to_binary[4]=gray[4];
    for(i=3;i>=0;i=i-1)
     gray_to_binary[i] = gray[i] ^ gray_to_binary[i + 1];
    end
```

Figure 2:FIFO Code part1



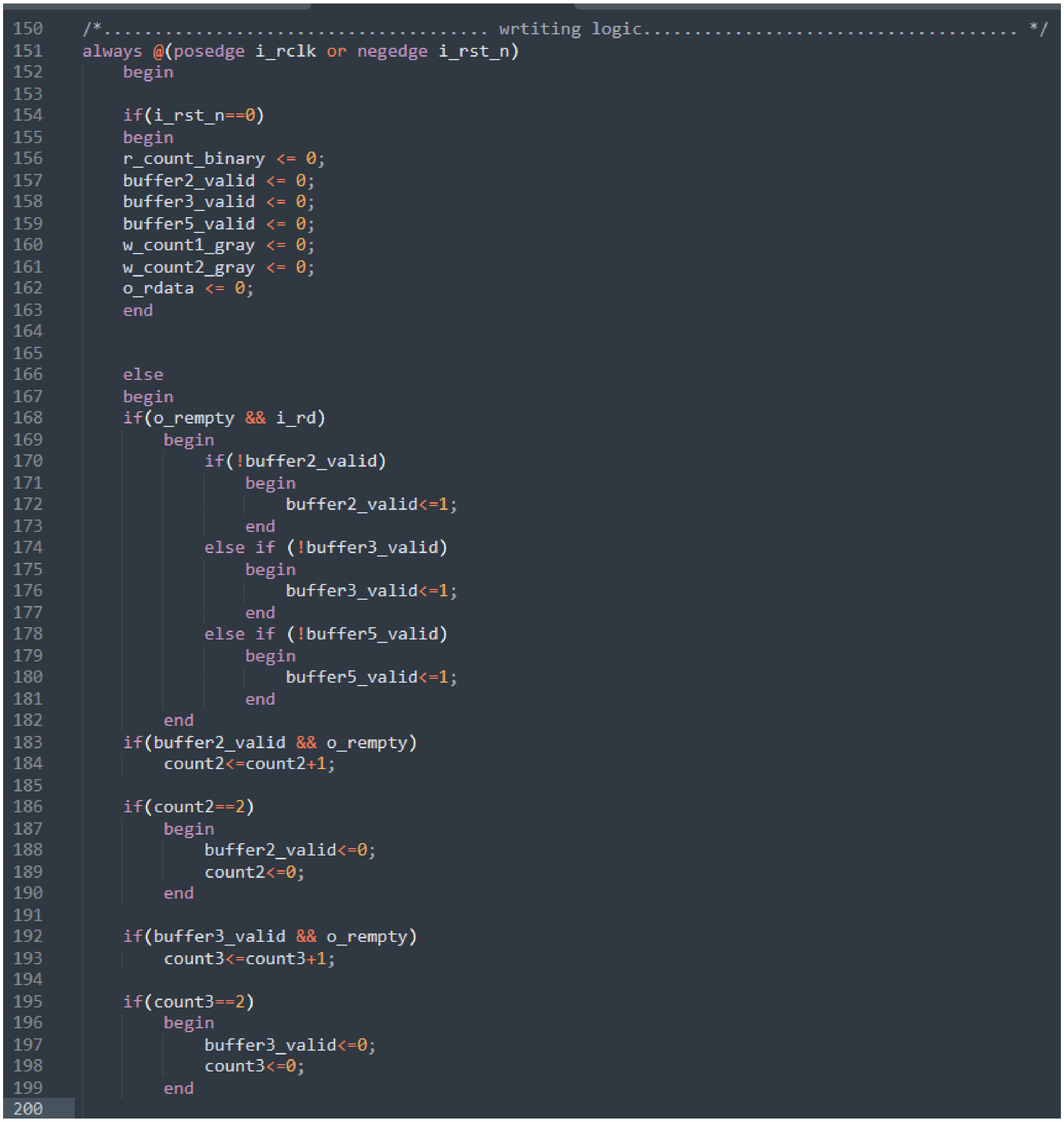
```
function[4:0]  binary_to_gray;
    input[4:0] binary;
    begin
     binary_to_gray[4]=binary[4];
     binary_to_gray[3:0] = binary[4:1] ^ binary[3:0];
    end
endfunction


assign w_count_gray=binary_to_gray(w_count_binary);
assign r_count_gray=binary_to_gray(r_count_binary);

assign w_count2_binary=gray_to_binary(w_count2_gray);
assign r_count2_binary=gray_to_binary(r_count2_gray);


assign o_wfull=(~i_rst_n)?0:((w_count_binary[4] != r_count2_binary[4]) &&  // MSB check to detect wrap-around
(w_count_binary[3:0] == r_count2_binary[3:0]));

assign o_rempty =(~i_rst_n)?1:(r_count_binary==w_count2_binary);

/*..................................... reading logic.................................... */
always @(posedge i_wclk or negedge i_rst_n)
begin

    if(i_rst_n==0)
        begin
        w_count_binary <= 0;
        buffer0_valid <= 0;
        buffer1_valid <= 0;
        buffer4_valid <= 0;
        r_count1_gray <= 0;
        r_count2_gray <= 0;
        end

    else
    begin
    if (!o_wfull && (buffer0_valid || buffer1_valid || buffer4_valid) ) begin
        if(buffer0_valid)
            begin
            FIFO[w_count_binary[3:0]] <= buffer0;
            w_count_binary <= w_count_binary + 1;
            end
        if(buffer1_valid)
            begin
            buffer0<=buffer1;
            buffer0_valid <= buffer1_valid;
            end
        else
```

Figure 3:FIFO Code part2



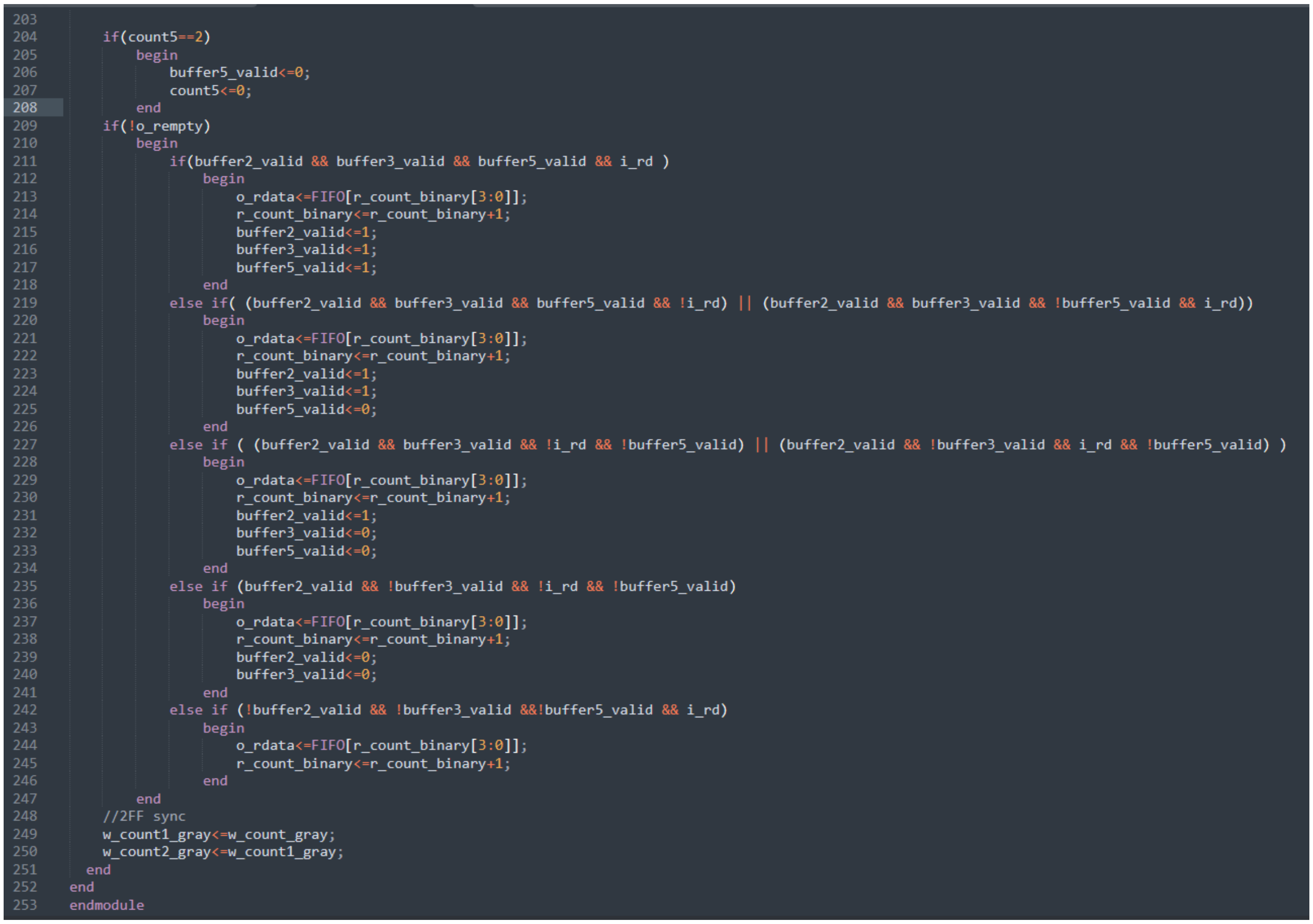
```
        else
            begin
                buffer0_valid<=0;
            end
        if(buffer4_valid)
            begin
            buffer1<=buffer4;
            buffer1_valid <= buffer4_valid;
            buffer4_valid<=0;
            end
        else
            begin
            buffer1_valid<=0;
            end
    end

    if (i_wr)
    begin
        if (!o_wfull && !buffer0_valid && !buffer1_valid && !buffer4_valid)
            begin
            FIFO[w_count_binary[3:0]] <= i_wdata;
            w_count_binary <= w_count_binary + 1;
            end
        else if(!buffer0_valid)
            begin
            buffer0<=i_wdata;
            buffer0_valid<=1;
            end
        else if (!buffer1_valid)
            begin
            buffer1<=i_wdata;
            buffer1_valid<=1;
            end
        else if (!buffer4_valid)
            begin
            buffer4<=i_wdata;
            buffer4_valid<=1;
            end
    end

    //2FF sync
    r_count1_gray<=r_count_gray;
    r_count2_gray<=r_count1_gray;
    end
end
```

Figure 4:FIFO Code part3



```
/*.................................... wrtiting logic.................................... */
always @(posedge i_rclk or negedge i_rst_n)
    begin

    if(i_rst_n==0)
    begin
    r_count_binary <= 0;
    buffer2_valid <= 0;
    buffer3_valid <= 0;
    buffer5_valid <= 0;
    w_count1_gray <= 0;
    w_count2_gray <= 0;
    o_rdata <= 0;
    end


    else
    begin
    if(o_rempty && i_rd)
        begin
            if(!buffer2_valid)
                begin
                    buffer2_valid<=1;
                end
            else if (!buffer3_valid)
                begin
                    buffer3_valid<=1;
                end
            else if (!buffer5_valid)
                begin
                    buffer5_valid<=1;
                end
        end
    if(buffer2_valid && o_rempty)
        count2<=count2+1;

    if(count2==2)
        begin
            buffer2_valid<=0;
            count2<=0;
        end

    if(buffer3_valid && o_rempty)
        count3<=count3+1;

    if(count3==2)
        begin
            buffer3_valid<=0;
            count3<=0;
        end

```

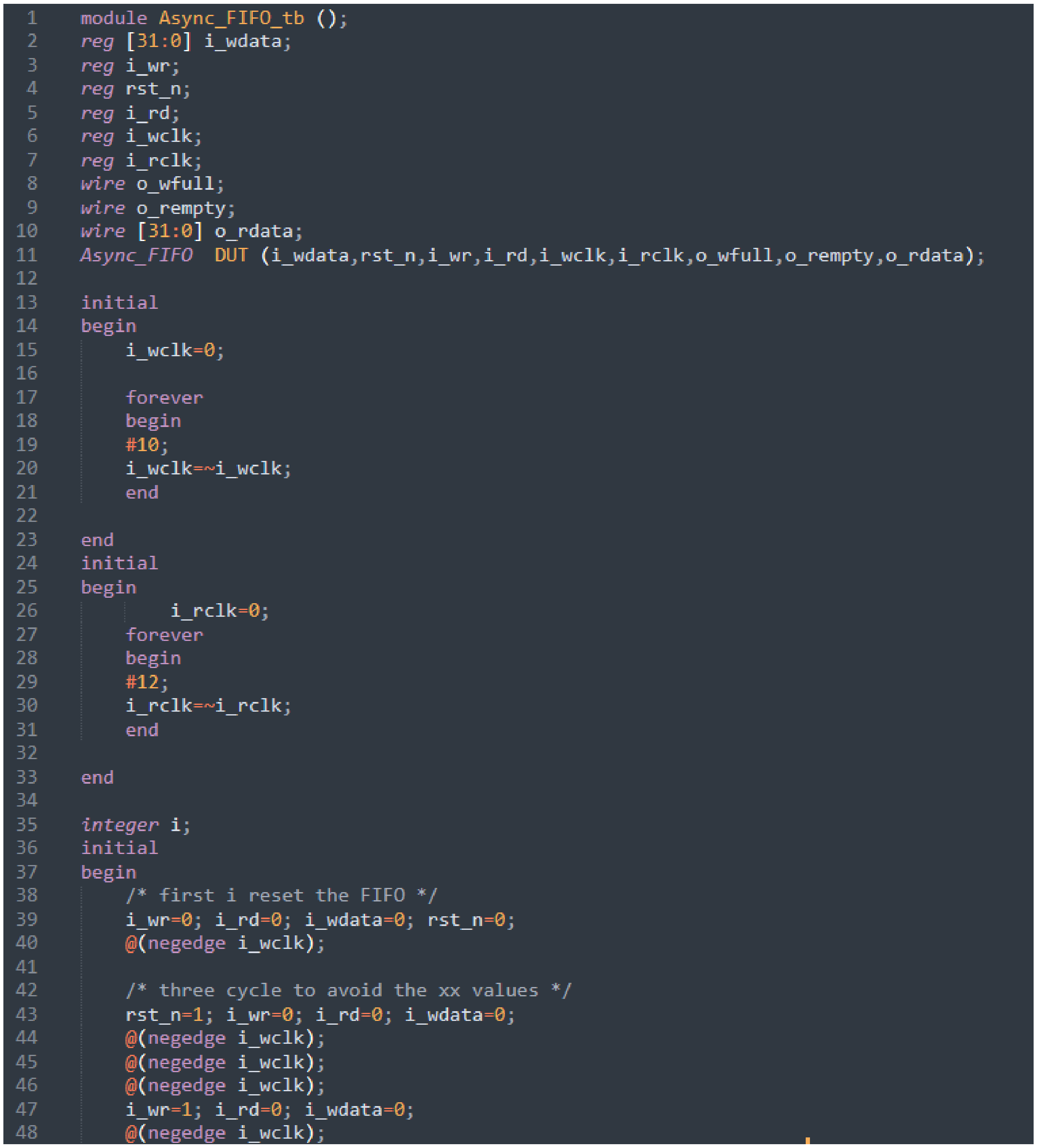
Figure 5:FIFO Code part4



```verilog
    if(count5==2)
        begin
            buffer5_valid<=0;
            count5<=0;
        end
    if(!o_rempty)
        begin
            if(buffer2_valid && buffer3_valid && buffer5_valid && i_rd )
                begin
                    o_rdata<=FIFO[r_count_binary[3:0]];
                    r_count_binary<=r_count_binary+1;
                    buffer2_valid<=1;
                    buffer3_valid<=1;
                    buffer5_valid<=1;
                end
            else if( (buffer2_valid && buffer3_valid && buffer5_valid && !i_rd) || (buffer2_valid && buffer3_valid && !buffer5_valid && i_rd))
                begin
                    o_rdata<=FIFO[r_count_binary[3:0]];
                    r_count_binary<=r_count_binary+1;
                    buffer2_valid<=1;
                    buffer3_valid<=1;
                    buffer5_valid<=0;
                end
            else if ( (buffer2_valid && buffer3_valid && !i_rd && !buffer5_valid) || (buffer2_valid && !buffer3_valid && i_rd && !buffer5_valid) )
                begin
                    o_rdata<=FIFO[r_count_binary[3:0]];
                    r_count_binary<=r_count_binary+1;
                    buffer2_valid<=1;
                    buffer3_valid<=0;
                    buffer5_valid<=0;
                end
            else if (buffer2_valid && !buffer3_valid && !i_rd && !buffer5_valid)
                begin
                    o_rdata<=FIFO[r_count_binary[3:0]];
                    r_count_binary<=r_count_binary+1;
                    buffer2_valid<=0;
                    buffer3_valid<=0;
                end
            else if (!buffer2_valid && !buffer3_valid &&!buffer5_valid && i_rd)
                begin
                    o_rdata<=FIFO[r_count_binary[3:0]];
                    r_count_binary<=r_count_binary+1;
                end
        end
    //2FF sync
    w_count1_gray<=w_count_gray;
    w_count2_gray<=w_count1_gray;
  end
end
endmodule
```

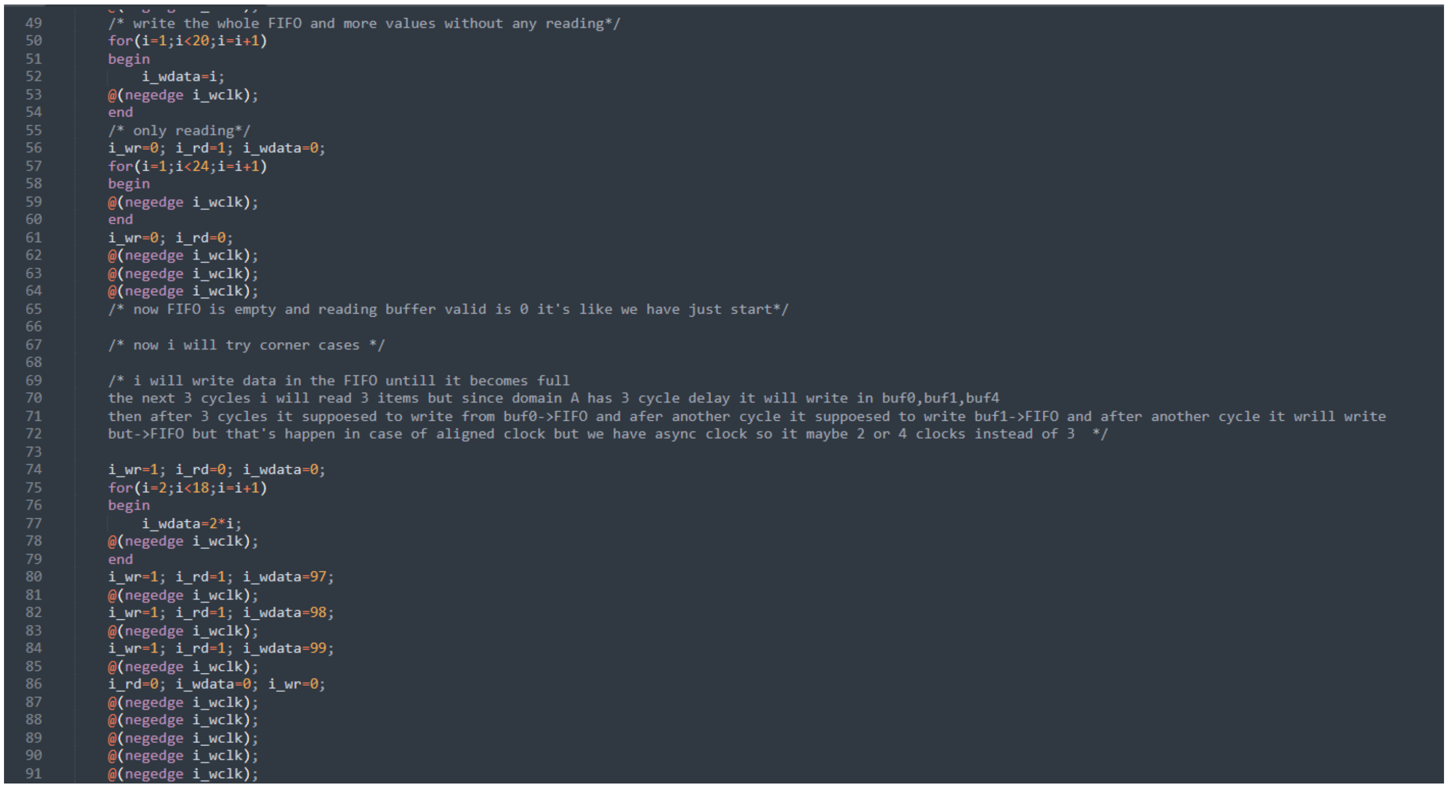
Figure 6:FIFO Code part5

5 Testbench code



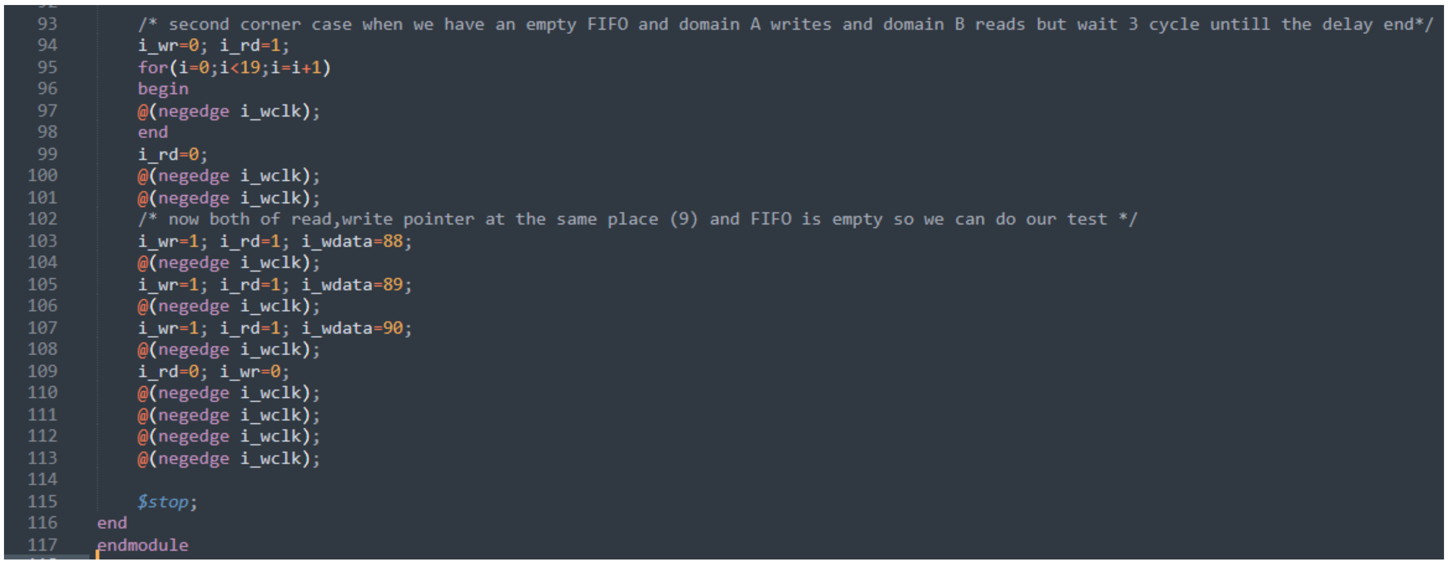
```
module Async_FIFO_tb ();
reg [31:0] i_wdata;
reg i_wr;
reg rst_n;
reg i_rd;
reg i_wclk;
reg i_rclk;
wire o_wfull;
wire o_rempty;
wire [31:0] o_rdata;
Async_FIFO  DUT (i_wdata,rst_n,i_wr,i_rd,i_wclk,i_rclk,o_wfull,o_rempty,o_rdata);

initial
begin
    i_wclk=0;

    forever
    begin
    #10;
    i_wclk=~i_wclk;
    end

end
initial
begin
        i_rclk=0;
    forever
    begin
    #12;
    i_rclk=~i_rclk;
    end

end

integer i;
initial
begin
    /* first i reset the FIFO */
    i_wr=0; i_rd=0; i_wdata=0; rst_n=0;
    @(negedge i_wclk);

    /* three cycle to avoid the xx values */
    rst_n=1; i_wr=0; i_rd=0; i_wdata=0;
    @(negedge i_wclk);
    @(negedge i_wclk);
    @(negedge i_wclk);
    i_wr=1; i_rd=0; i_wdata=0;
    @(negedge i_wclk);
```

Figure 7:Testbench part1



```
/* write the whole FIFO and more values without any reading*/
for(i=1;i<20;i=i+1)
begin
    i_wdata=i;
@(negedge i_wclk);
end
/* only reading*/
i_wr=0; i_rd=1; i_wdata=0;
for(i=1;i<24;i=i+1)
begin
@(negedge i_wclk);
end
i_wr=0; i_rd=0;
@(negedge i_wclk);
@(negedge i_wclk);
@(negedge i_wclk);
/* now FIFO is empty and reading buffer valid is 0 it's like we have just start*/

/* now i will try corner cases */

/* i will write data in the FIFO untill it becomes full
the next 3 cycles i will read 3 items but since domain A has 3 cycle delay it will write in buf0,buf1,buf4
then after 3 cycles it suppoesed to write from buf0->FIFO and afer another cycle it suppoesed to write buf1->FIFO and after another cycle it wrill write
but->FIFO but that's happen in case of aligned clock but we have async clock so it maybe 2 or 4 clocks instead of 3  */

i_wr=1; i_rd=0; i_wdata=0;
for(i=2;i<18;i=i+1)
begin
    i_wdata=2*i;
@(negedge i_wclk);
end
i_wr=1; i_rd=1; i_wdata=97;
@(negedge i_wclk);
i_wr=1; i_rd=1; i_wdata=98;
@(negedge i_wclk);
i_wr=1; i_rd=1; i_wdata=99;
@(negedge i_wclk);
i_rd=0; i_wdata=0; i_wr=0;
@(negedge i_wclk);
@(negedge i_wclk);
@(negedge i_wclk);
@(negedge i_wclk);
@(negedge i_wclk);
```

Figure 8:Testbench part2



```
    /* second corner case when we have an empty FIFO and domain A writes and domain B reads but wait 3 cycle untill the delay end*/
    i_wr=0; i_rd=1;
    for(i=0;i<19;i=i+1)
    begin
    @(negedge i_wclk);
    end
    i_rd=0;
    @(negedge i_wclk);
    @(negedge i_wclk);
    /* now both of read,write pointer at the same place (9) and FIFO is empty so we can do our test */
    i_wr=1; i_rd=1; i_wdata=88;
    @(negedge i_wclk);
    i_wr=1; i_rd=1; i_wdata=89;
    @(negedge i_wclk);
    i_wr=1; i_rd=1; i_wdata=90;
    @(negedge i_wclk);
    i_rd=0; i_wr=0;
    @(negedge i_wclk);
    @(negedge i_wclk);
    @(negedge i_wclk);
    @(negedge i_wclk);

    $stop;
end
endmodule
```

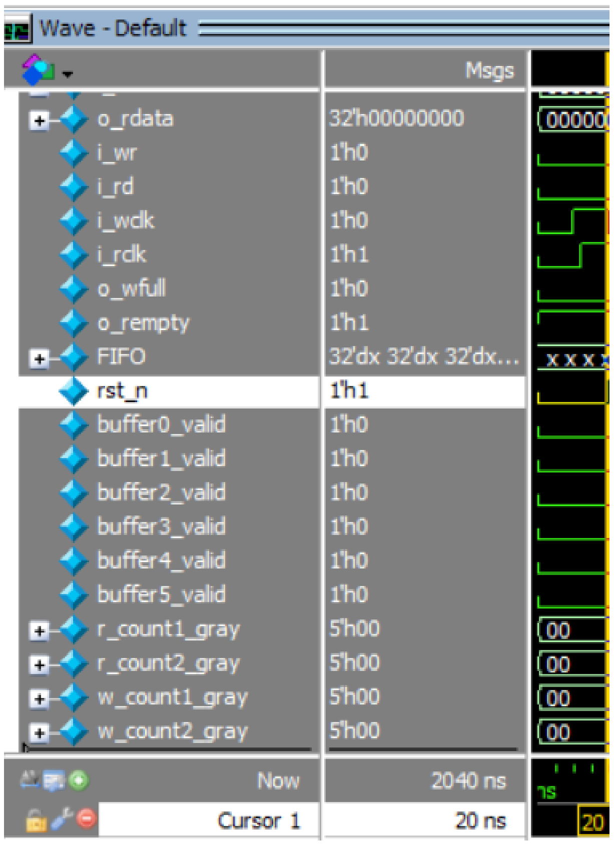
Figure 9:Testbench part3

6 Testbench plane

Case num	Description of the case	Expected output
1	Assert the reset_n	The all buffers and internal signal tied to 0
2	Write the whole FIFO and want to overwrite some elements	FIFO will be FULL then will save the next 3 elements and refuses the other overwritten elements until reading happens
3	Read the whole FIFO including the 3 extra elements	First the extra 3 elements in the buffer will be written then after that we will read the whole FIFO including them so now FIFO is empty
4 (first corner case)	I will write in the whole FIFO until it becomes full then I will try to read and write at the same time for 3 elements (cycles)	The FIFO will be full first, then when I want to write again it will save the values in the three buffers and once, I read the 3 elements the buffered values will be transferred from the buffer to the FIFO
5 (second corner case)	I will read the whole FIFO until it becomes empty then I will try to write and read at the same time for 3 elements (cycles)	The FIFO will be empty first, then when I want to read again it will rise the 3 buffers flags and once, I write the 3 elements the buffered values will be transferred from the the FIFO to the output

7 Waveforms

7.1 Case 1


Wave - Default

	Msgs
o_rdata	32'h00000000
i_wr	1'h0
i_rd	1'h0
i_wclk	1'h0
i_rclk	1'h1
o_wfull	1'h0
o_rempty	1'h1
FIFO	32'dx 32'dx 32'dx...
rst_n	1'h1
buffer0_valid	1'h0
buffer1_valid	1'h0
buffer2_valid	1'h0
buffer3_valid	1'h0
buffer4_valid	1'h0
buffer5_valid	1'h0
r_count1_gray	5'h00
r_count2_gray	5'h00
w_count1_gray	5'h00
w_count2_gray	5'h00
Now	2040 ns
Cursor 1	20 ns

Figure 10:wave case1

Assert the reset_n	The all buffers and internal signal tied to 0

7.2 Case 2

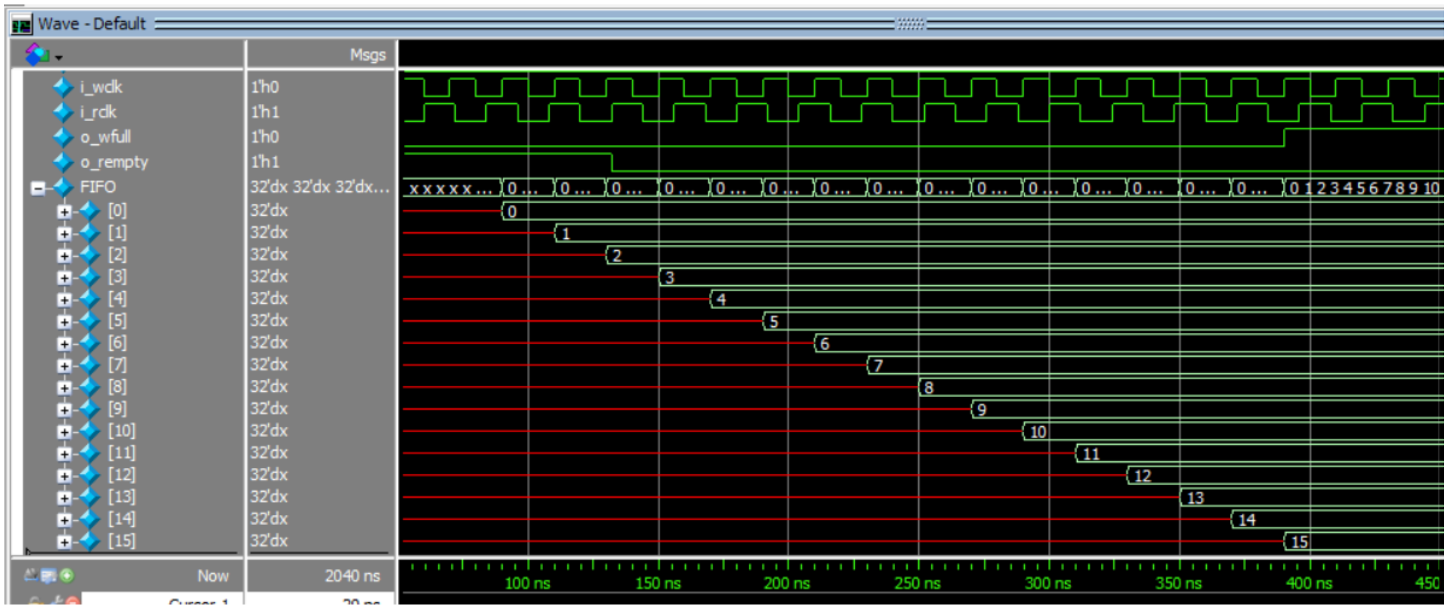
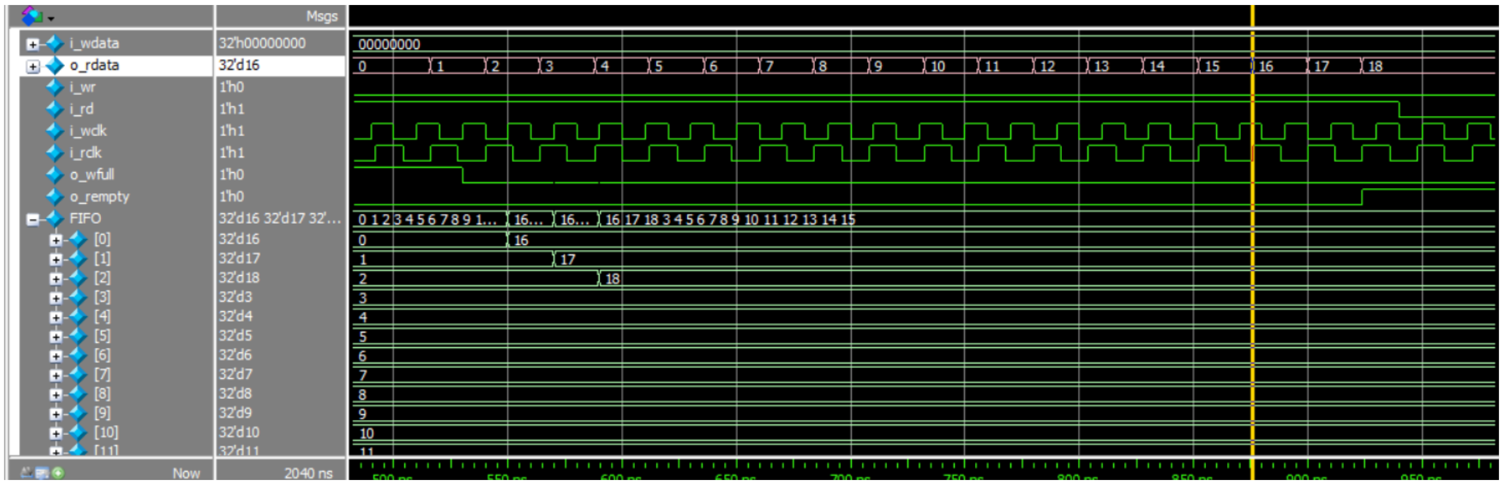


Figure 11:wave case2

Write the whole FIFO and want to overwrite some elements	FIFO will be FULL then will save the next 3 elements and refuses the other overwritten elements until reading happens

7.3 Case 3



Signal	Msgs
i_wdata	32'h00000000
o_rdata	32'd16
i_wr	1'h0
i_rd	1'h1
i_wclk	1'h1
i_rclk	1'h1
o_wfull	1'h0
o_rempty	1'h0
FIFO	32'd16 32'd17 32'...
[0]	32'd16
[1]	32'd17
[2]	32'd18
[3]	32'd3
[4]	32'd4
[5]	32'd5
[6]	32'd6
[7]	32'd7
[8]	32'd8
[9]	32'd9
[10]	32'd10
[11]	32'd11

Figure 12:wave case 3

Read the whole FIFO including the 3 extra elements	First the extra 3 elements in the buffer will be written then after that we will read the whole FIFO including them so now FIFO is empty

7.4 Case 4

7.4.1 Part 1

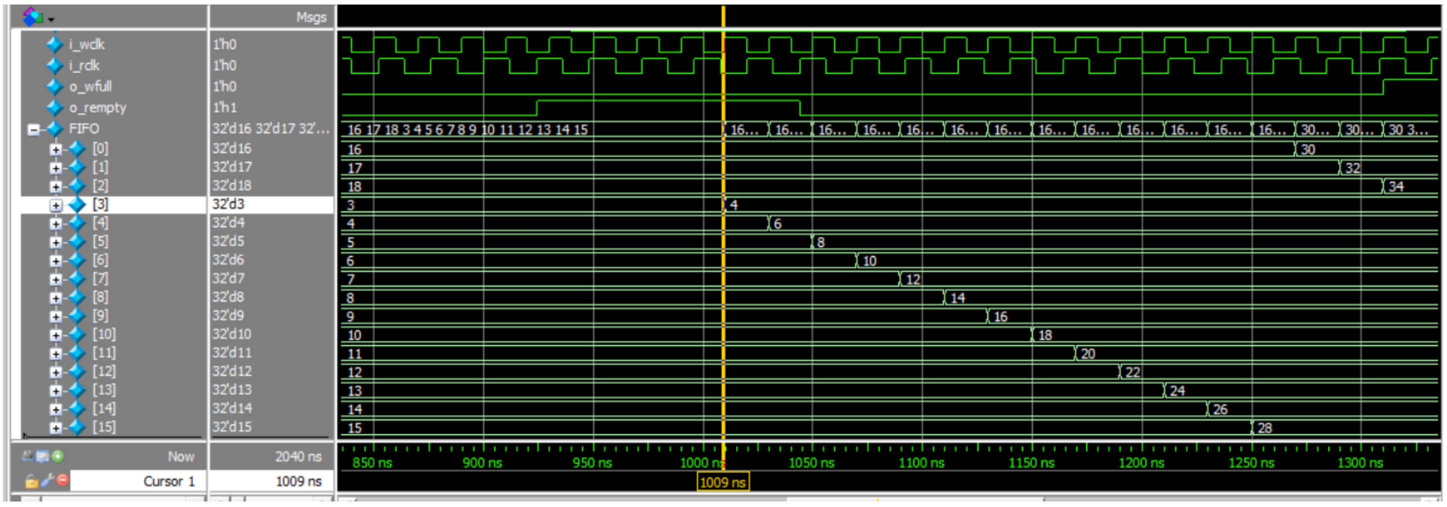


Figure 13:wave case 4 part1

I will write in the whole FIFO until it becomes full	The FIFO will be full first

7.4.2 Part2

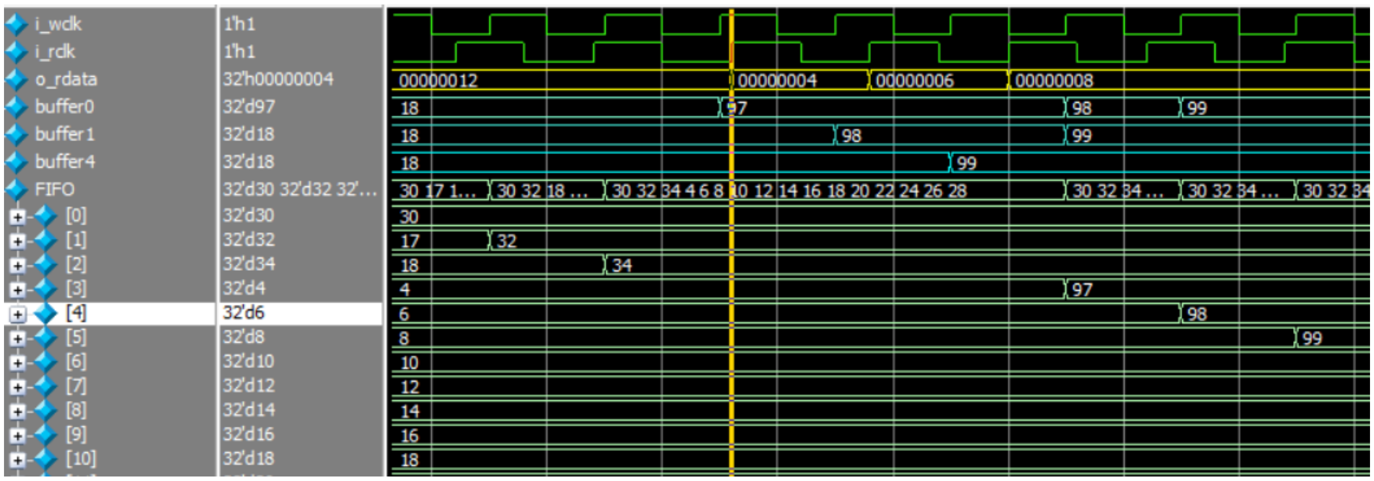


Figure 14:case 4 part2

Then I will try to read and write at the same time for 3 elements (cycles)	Then when I want to write again it will save the values in the three buffers and once, I read the 3 elements the buffered values will be transferred from the buffer to the FIFO

7.5 Case 5

7.5.1 Part1

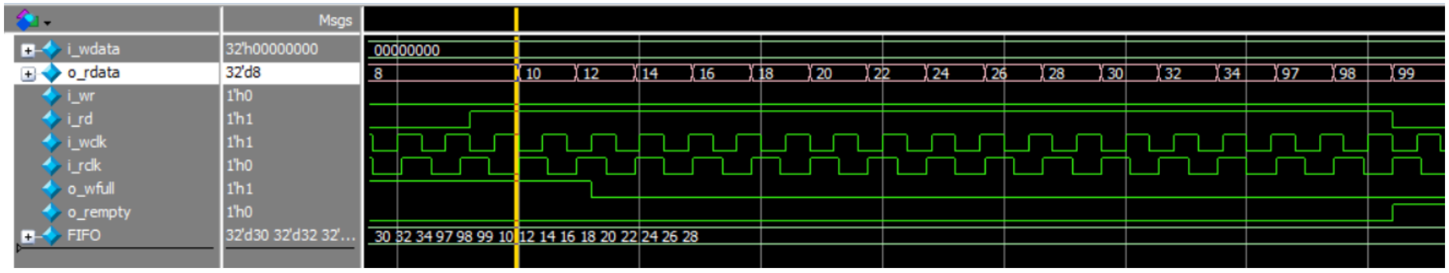
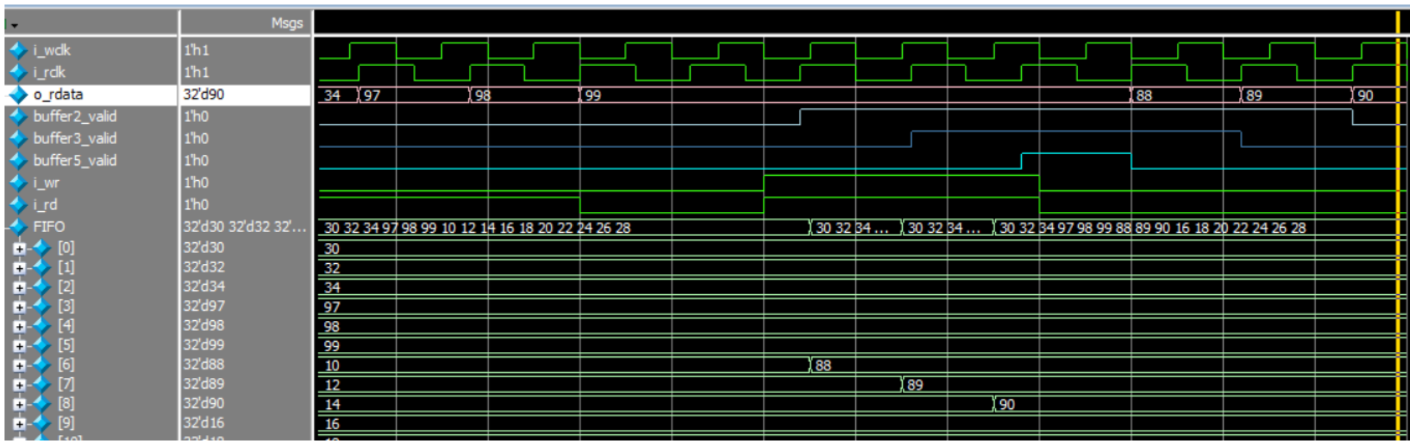


Figure 15:wave case5 part1

I will read the whole FIFO until it becomes empty	The FIFO will be empty first

7.5.2 Part2



	Msgs
i_wclk	1'h1
i_rclk	1'h1
o_rdata	32'd90
buffer2_valid	1'h0
buffer3_valid	1'h0
buffer5_valid	1'h0
i_wr	1'h0
i_rd	1'h0
FIFO	32'd30 32'd32 32'...
[0]	32'd30
[1]	32'd32
[2]	32'd34
[3]	32'd97
[4]	32'd98
[5]	32'd99
[6]	32'd88
[7]	32'd89
[8]	32'd90
[9]	32'd16

Figure 16:wave case5 part2

then I will try to write and read at the same time for 3 elements (cycles)	then when I want to read again it will rise the 3 buffers flags and once, I write the 3 elements will be transferred from the FIFO to the output

8 Vivado

8.1 Elaborated design for 8-bit width

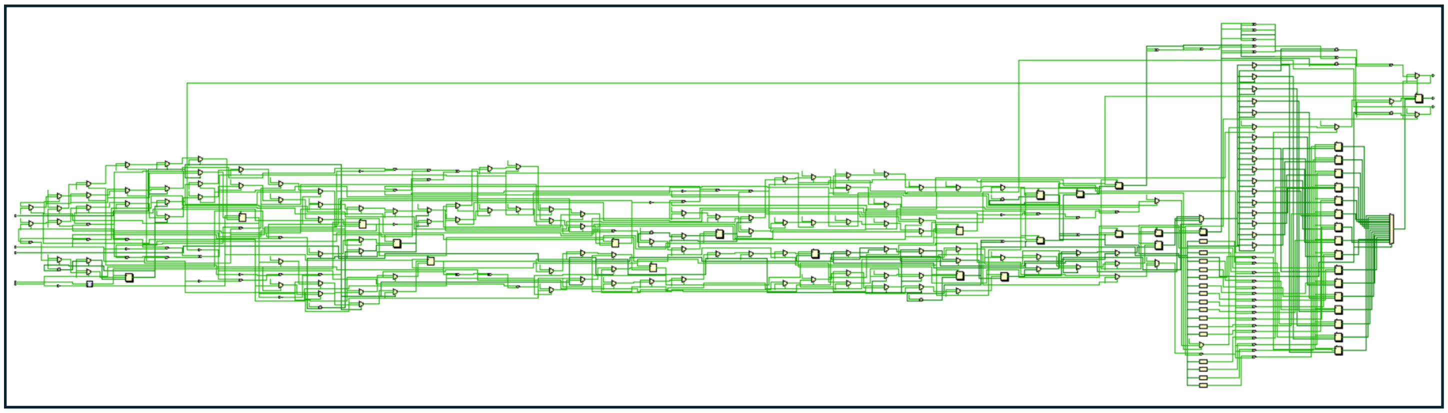


Figure 17:elaborated design for 8-bits

8.2 Elaborated design for 32-bit width

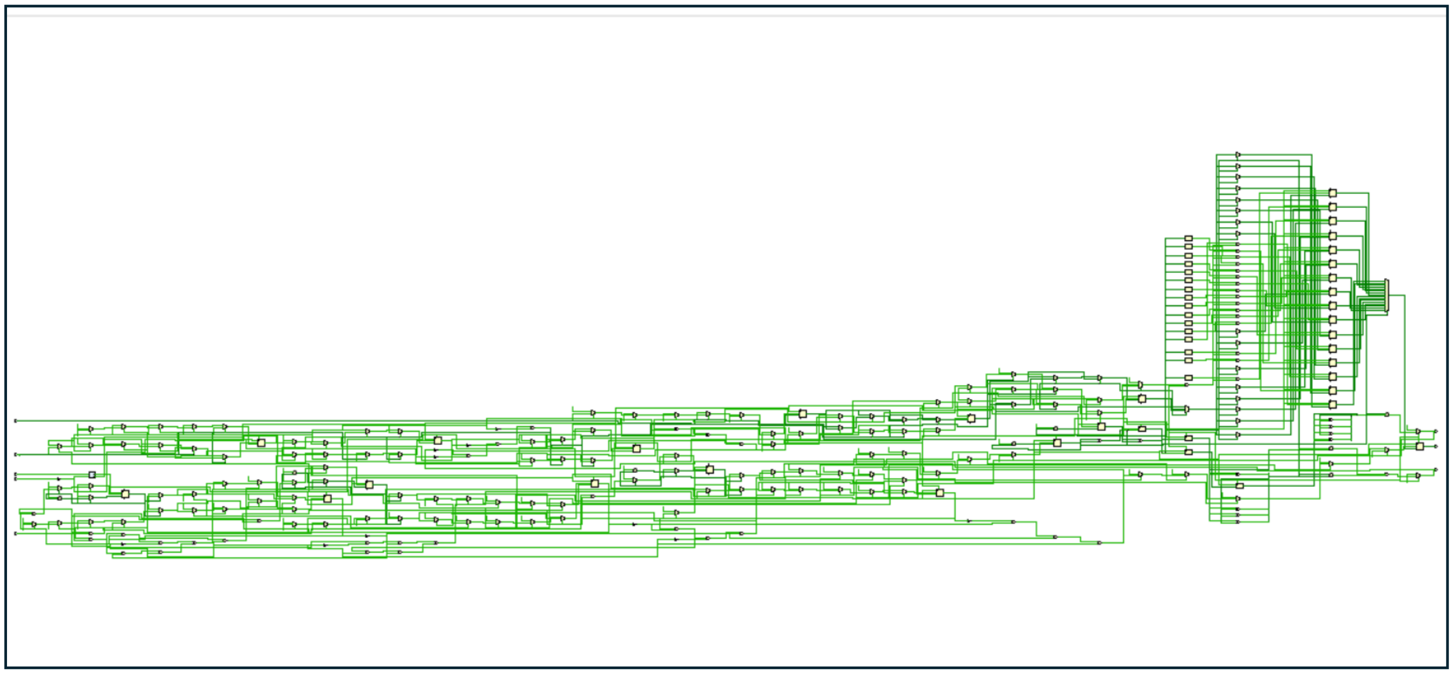


Figure 18:elaborated design for 32-bits

8.3 Synthesis for 8-bits

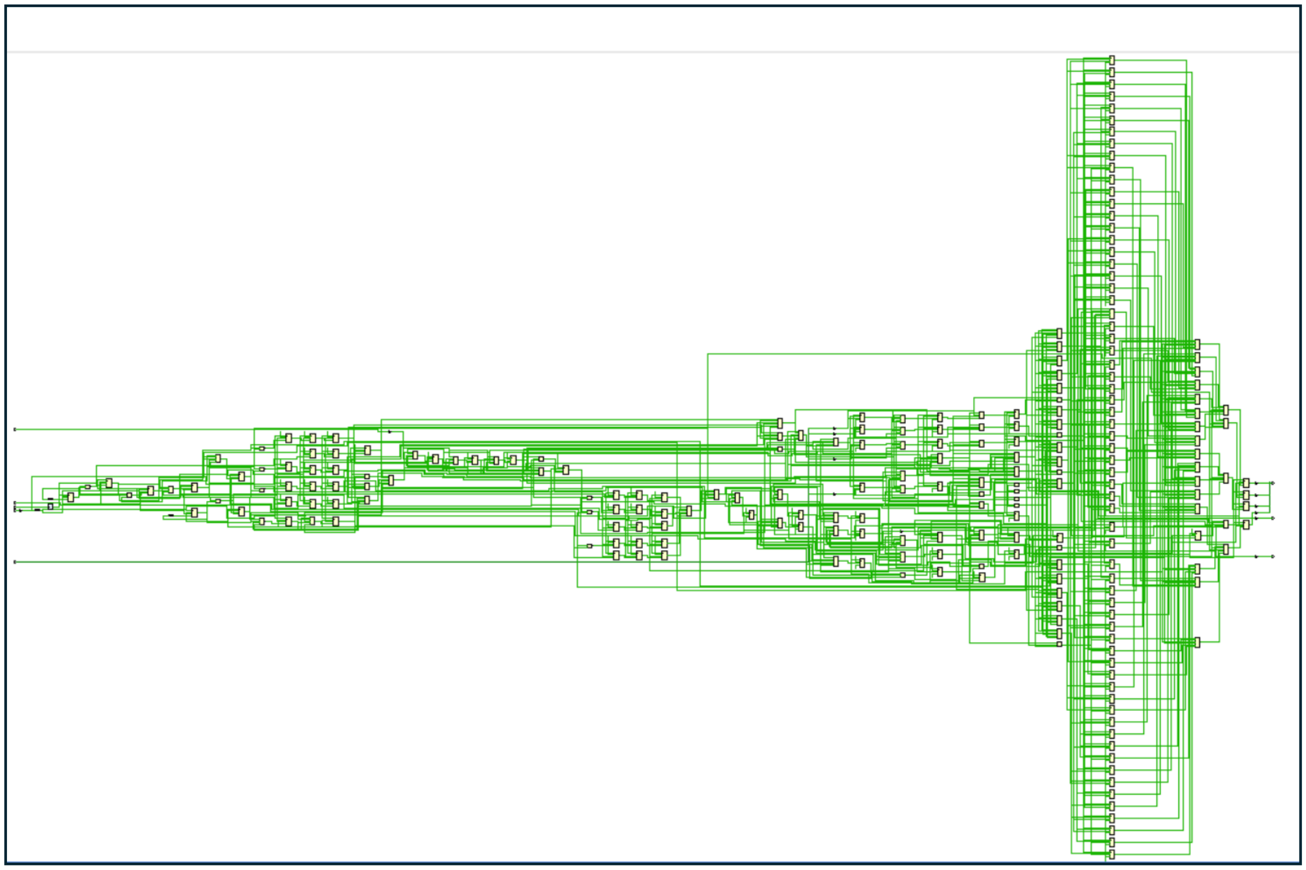


Figure 19:synthesized design for 8-bits

8.4 Synthesis for 32-bits

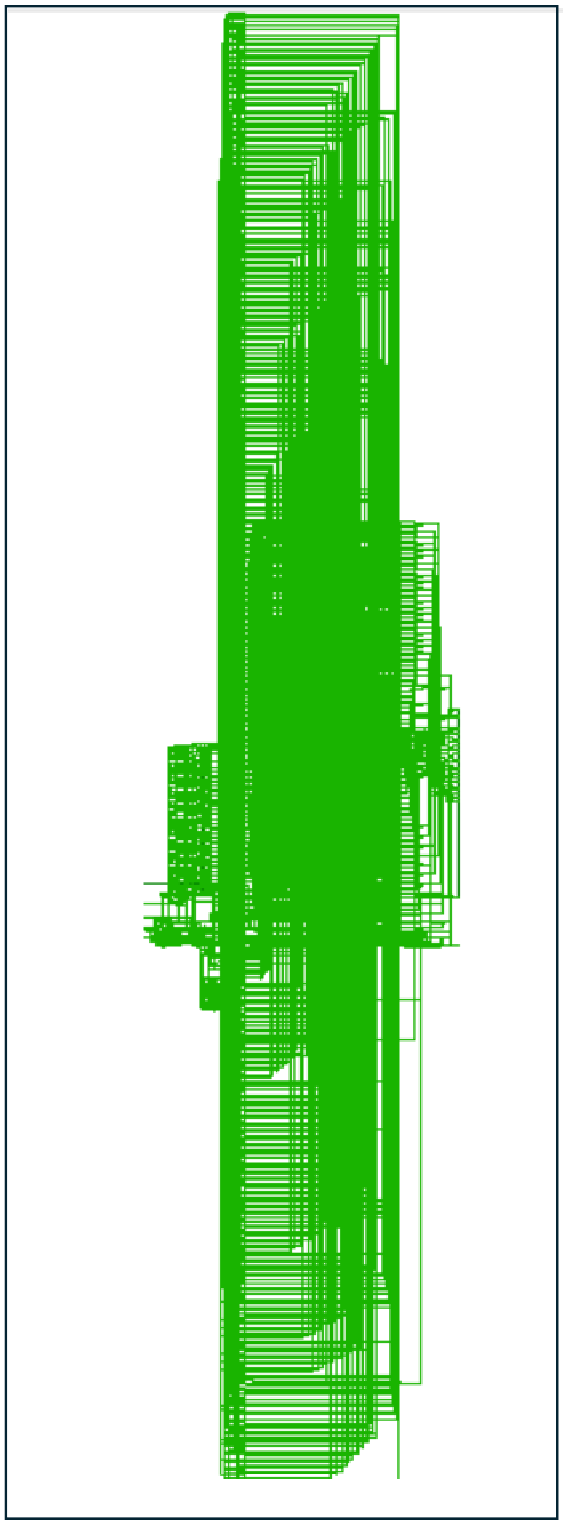


Figure 20: synthesized design for 32-bits

8.5 Implementation for 8-bits

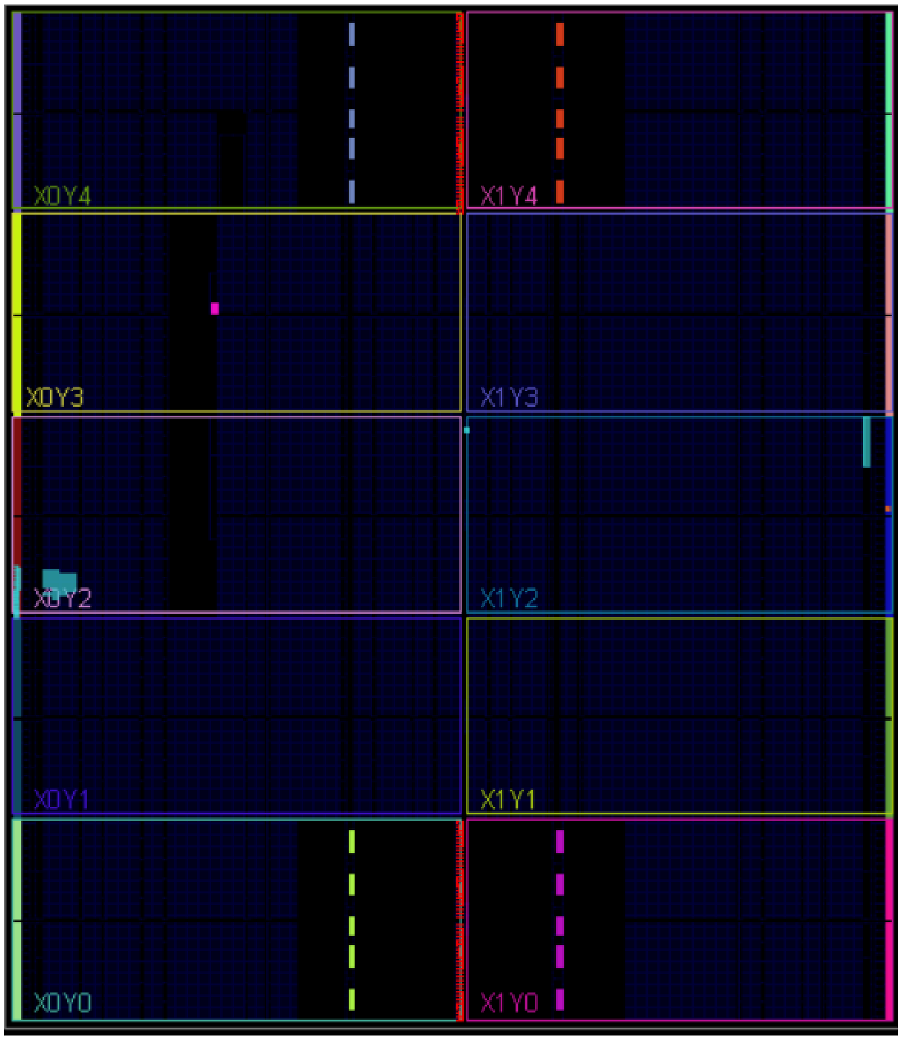


Figure 21:implemented design for 8 bits

8.6 Implementation for 32-bits

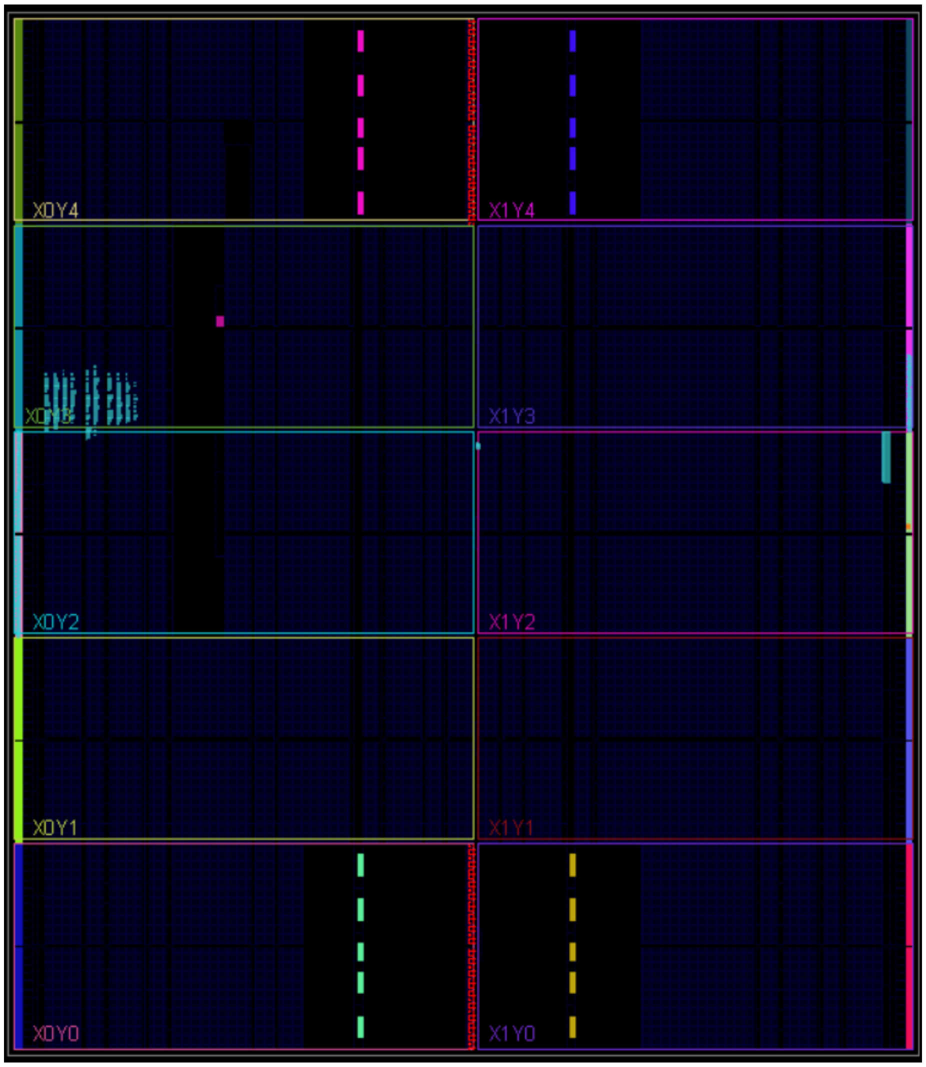
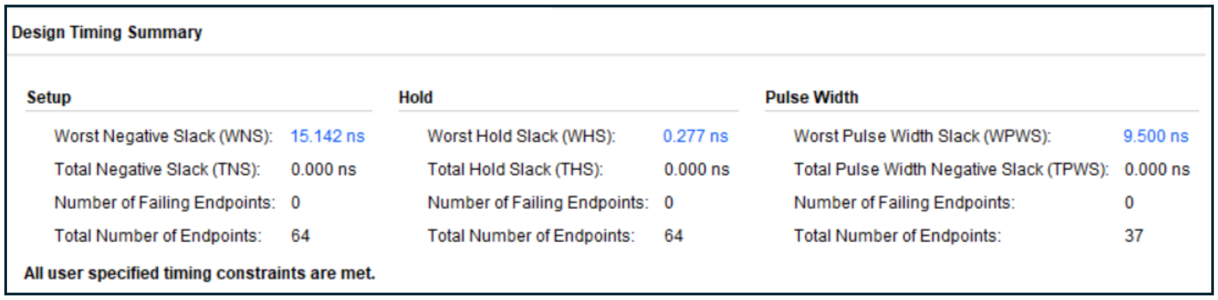


Figure 22:implemented design for 32 bits

8.7 Timing report at 50MHZ,45MHZ for 8 bits



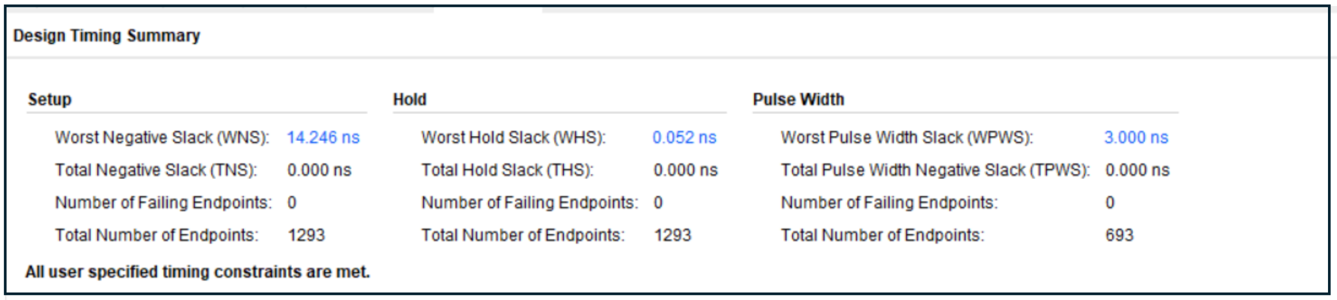
Design Timing Summary

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	15.142 ns	Worst Hold Slack (WHS):	0.277 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	64	Total Number of Endpoints:	64	Total Number of Endpoints:	37

All user specified timing constraints are met.

Figure 23:time report for 8 bits

8.8 Timing report at 50MHZ,45MHZ for 8 bits



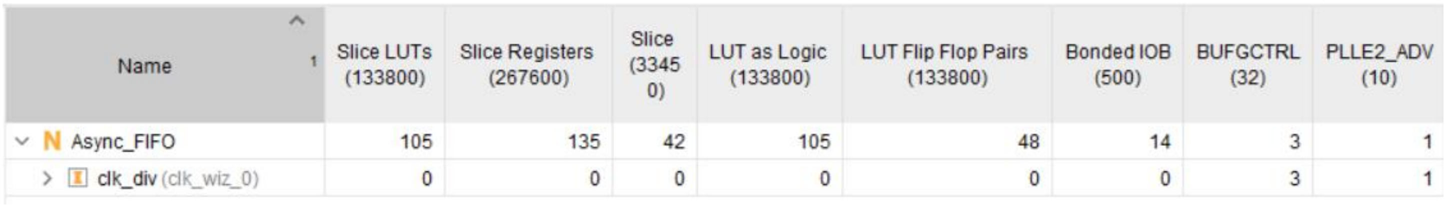
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	14.246 ns	Worst Hold Slack (WHS):	0.052 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1293	Total Number of Endpoints:	1293	Total Number of Endpoints:	693

All user specified timing constraints are met.

Figure 24:time report for 32 bits

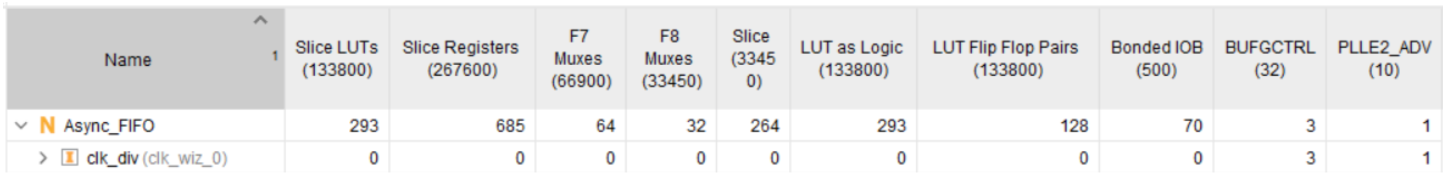
8.9 Utilization report for 8 bits



Name	Slice LUTs (133800)	Slice Registers (267600)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)	PLLE2_ADV (10)
Async_FIFO	105	135	42	105	48	14	3	1
clk_div (clk_wiz_0)	0	0	0	0	0	0	3	1

Figure 25:Utilization report for 8 bits

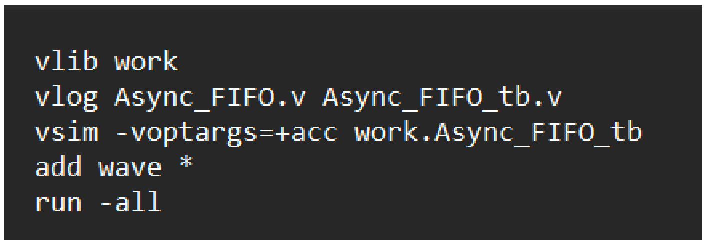
8.10 Utilization report for 32 bits



Name	Slice LUTs (133800)	Slice Registers (267600)	F7 Muxes (66900)	F8 Muxes (33450)	Slice (33450)	LUT as Logic (133800)	LUT Flip Flop Pairs (133800)	Bonded IOB (500)	BUFGCTRL (32)	PLLE2_ADV (10)
Async_FIFO	293	685	64	32	264	293	128	70	3	1
clk_div (clk_wiz_0)	0	0	0	0	0	0	0	0	3	1

Figure 26:Utilization report for 32 bits

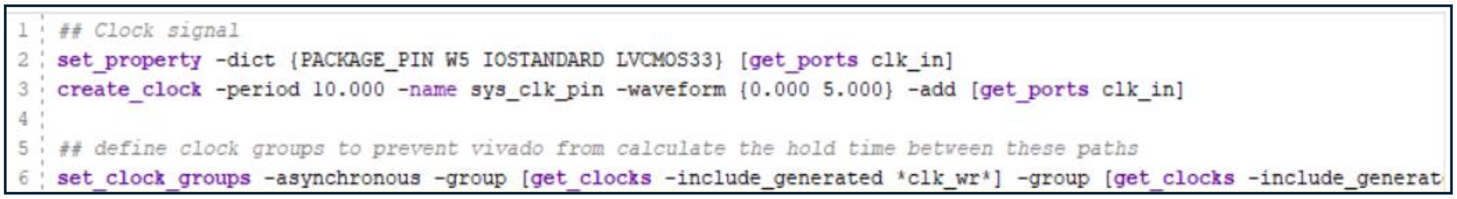
8.11 do file



```
vlib work
vlog Async_FIFO.v Async_FIFO_tb.v
vsim -voptargs=+acc work.Async_FIFO_tb
add wave *
run -all
```

Figure 27:do file

8.12 time constraint file



```
## Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports clk_in]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk_in]

## define clock groups to prevent vivado from calculate the hold time between these paths
set_clock_groups -asynchronous -group [get_clocks -include_generated *clk_wr*] -group [get_clocks -include_generat
```

Figure 28:constraint file