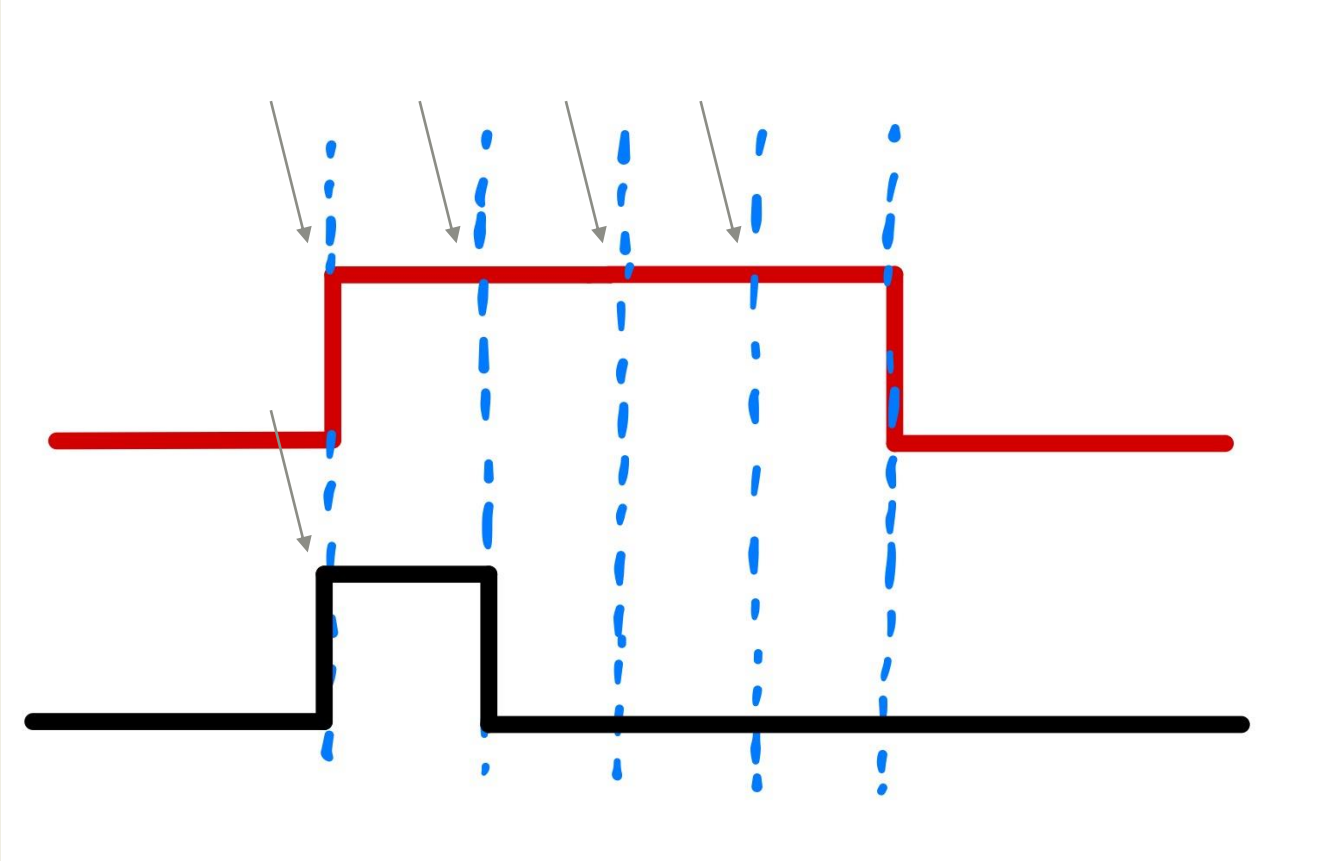
SEQUENTIAL LAB

Group5:
110060021曾偉博
110060035黃振寧
110061230張仕謙

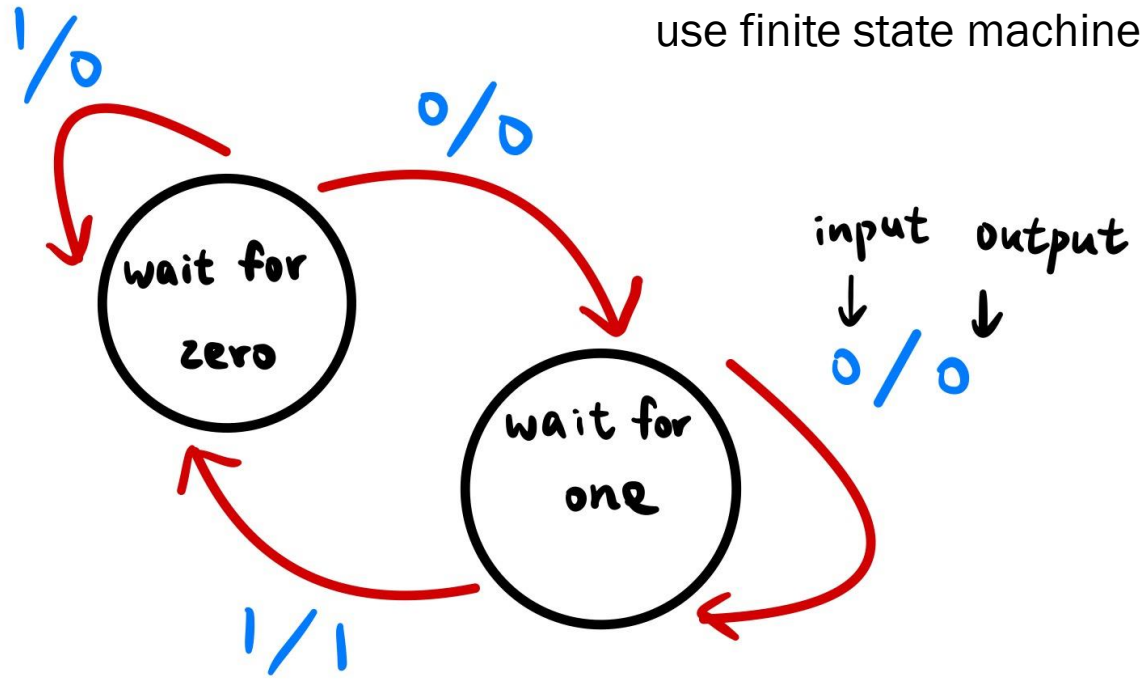
PULSE GENERATOR

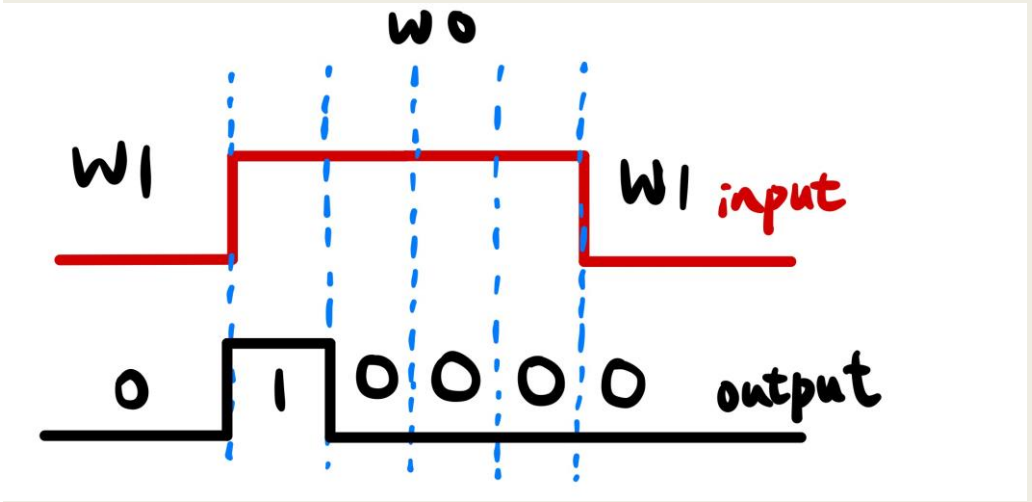
Why do we need a pulse generator?



How do we implement a pulse generator?

use finite state machine





HLS code

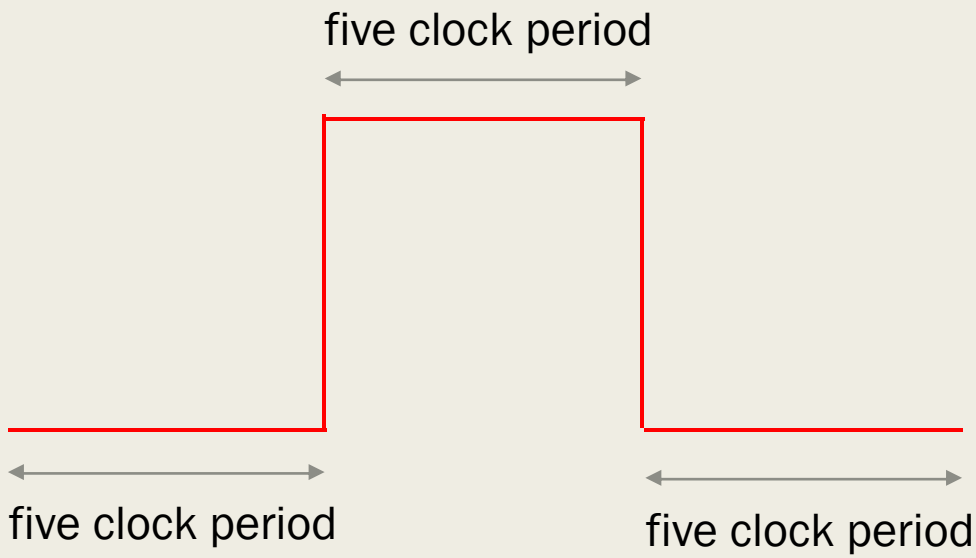
```
#include "pulse_generator.h"

typedef enum{w1, w0} pulse_gen_states_type;

void pulse_generator(bool input, bool &pulse) {
#pragma HLS INTERFACE ap_none port=input
#pragma HLS INTERFACE ap_none port=pulse
#pragma HLS INTERFACE ap_ctrl_none port=return


    static pulse_gen_states_type state = w1;


    pulse_gen_states_type next_state;
    bool next_pulse;
```

```
    switch(state) {
    case w1:
        if (input == 1) {
            next_state  = w0;
            next_pulse = 1;
        } else {
            next_state  = w1;
            next_pulse = 0;
        }
        break;
    case w0:
        if (input == 1) {
            next_state = w0;
            next_pulse = 0;
        } else {
            next_state = w1;
            next_pulse = 0;
        }
        break;
    default:

        break;
    }
```

```
    state = next_state;
    pulse = next_pulse;

}
```

HLS testbench

```
#include "pulse_generator-tb.h"
#include <iostream>



int main() {

    int status = 0;

    bool pulse;

    std::cout << " pulse = " ;
    for (int i = 0; i < 5; i++) {
         pulse_generator(0, pulse);
        std::cout << " " << pulse ;
    }

    for (int i = 0; i < 5; i++) {
        pulse_generator(1, pulse);
        std::cout << " " << pulse ;
    }

    for (int i = 0; i < 5; i++) {
        pulse_generator(0, pulse);
        std::cout << " " << pulse ;
    }

    std::cout << std::endl ;
    return status;
}
```



Result

pulse_generator.cpp | pulse_generator.h | pulse_generator-tb.cpp | pulse_generator-tb.h | pulse_generator_csim.log ×

```
1 INFO: [SIM 2] *************** CSIM start ***************
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3    Compiling ../../../../../lab_sequential/pulse_generator/pulse_generator-tb.cpp in debug mode
4    Compiling ../../../../../lab_sequential/pulse_generator/pulse_generator.cpp in debug mode
5    Generating csim.exe
6  pulse =  0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
7 INFO: [SIM 1] CSim done with 0 errors.
8 INFO: [SIM 3] *************** CSIM finish ***************
9
```

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	0.978 ns	2.70 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
0	0	0 ns	0 ns	1	1	no

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	2	-	-
Total	0	0	2	4	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	~0	~0	0

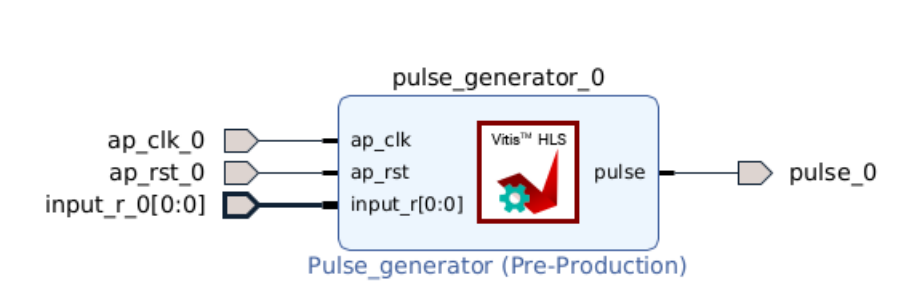
Cosimulation Report for 'pulse_generator'

General Information

Date: Tue 15 Aug 2023 07:42:05 AM EDT
Version: 2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)
Project: pulse generator vitis hls
Status: Pass

Solution: solution1 (Vivado IP Flow Target)
Product family: artix7
Target device: xc7a35t-cpg236-1

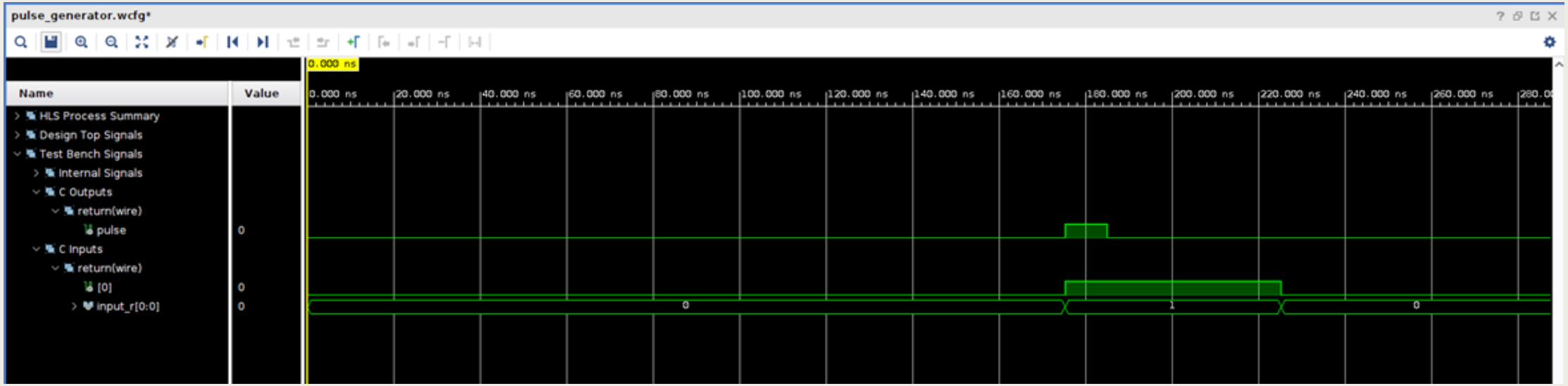
Block diagram



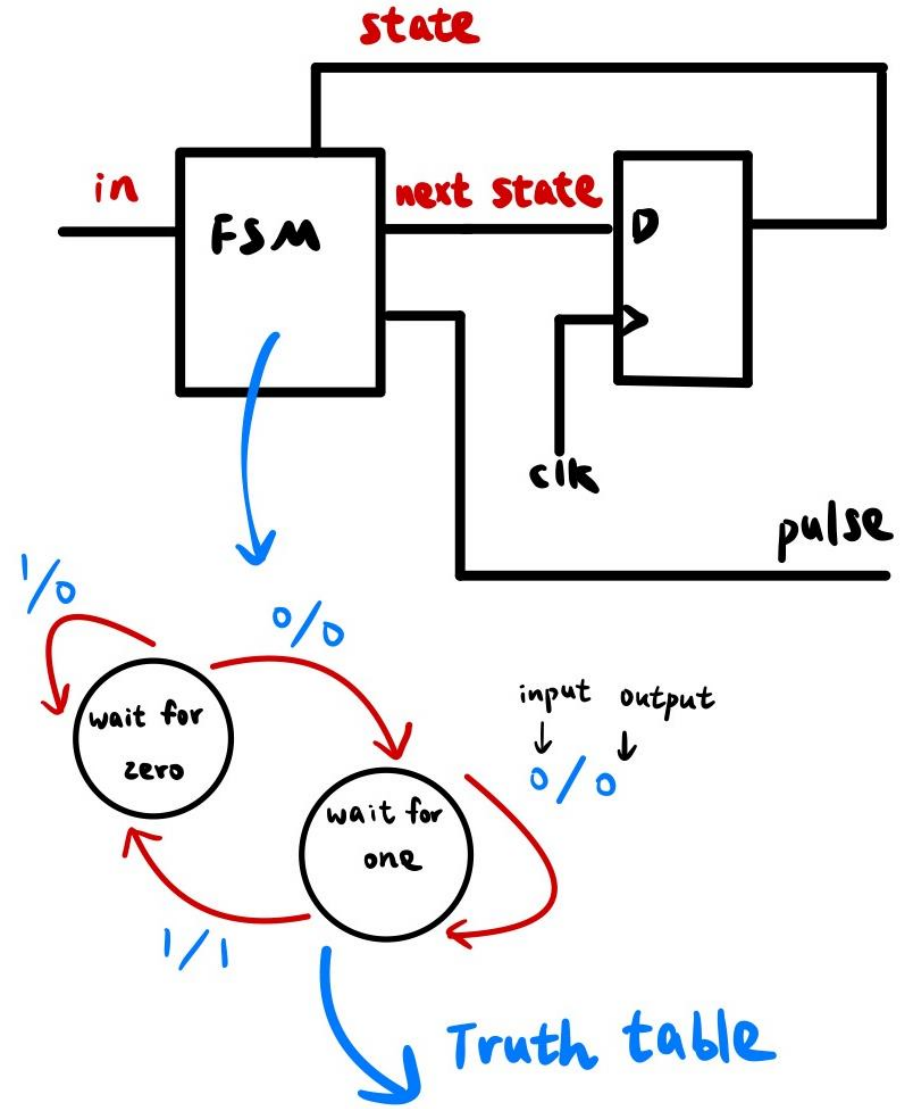
Tcl Console | Messages | Log | Reports | **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapse
synth_1 (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**	**8/15/2**	**00:00:**
impl_1	constrs_1	Not started																		
Out-of-Context Module Runs																				
design_1		Submodule Runs Complete																	8/15/2	00:00:
design_1_pul	design_1_pul	synth_design Complete!												1	1	0	0	0	8/15/2	00:00:

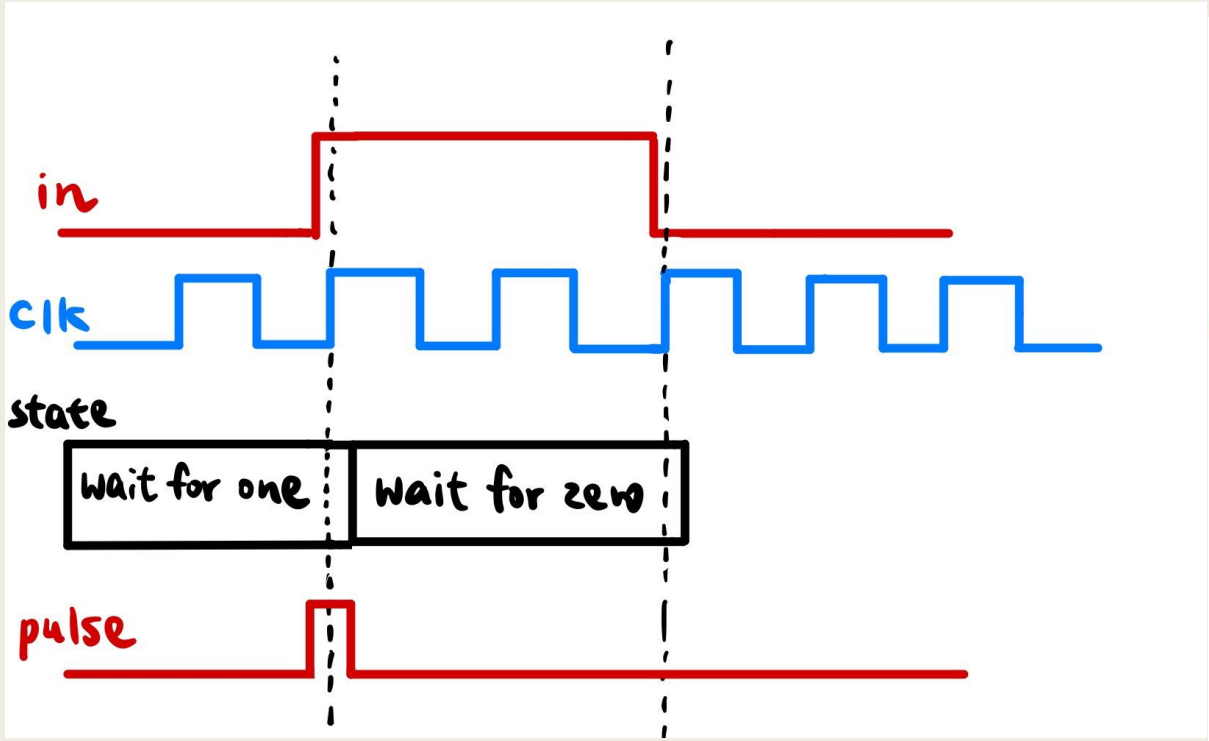
Waveform(HLS version)

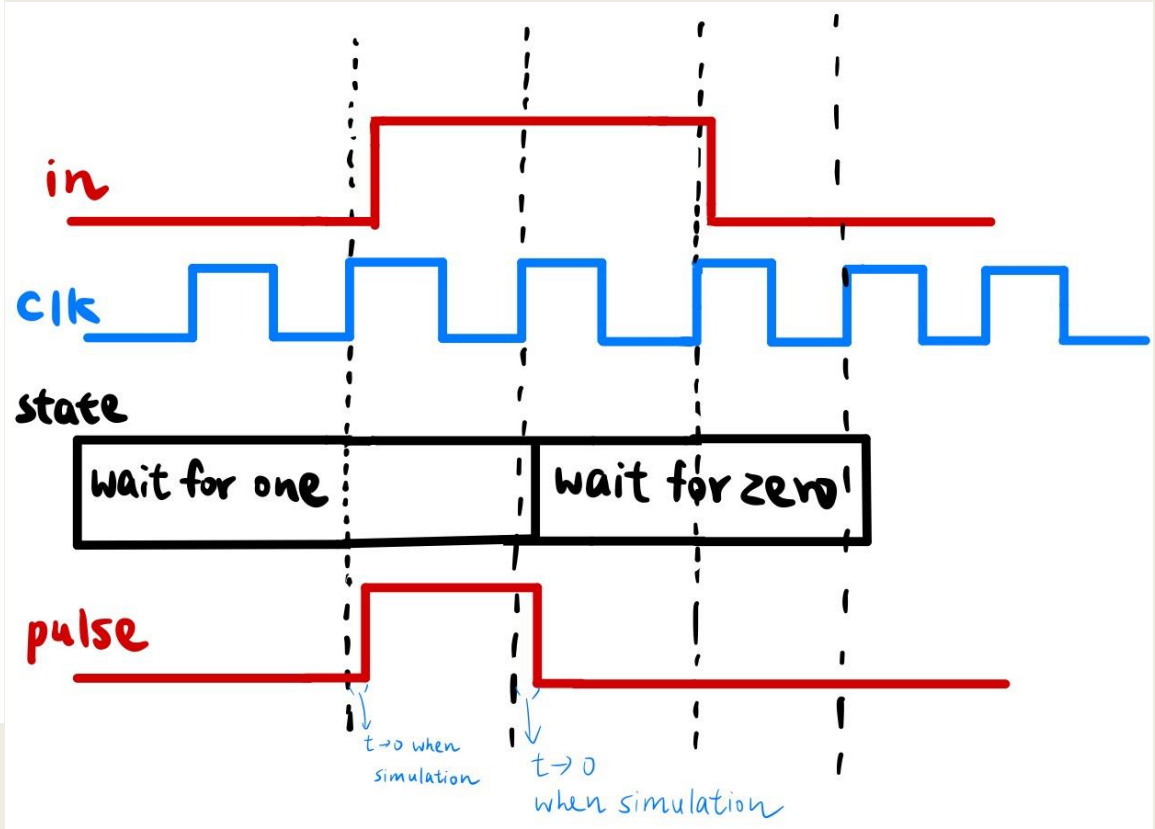


Logic structure



Timing waveform





Verilog code

FSM

```
`define wait_for_zero 1
`define wait_for_one 0

module pulse_generator(
    input in,
    input clk,
    input reset,
    output reg pulse
    );

reg state;
reg next_state;

always@* begin
    case(state)
        `wait_for_zero:
            if(in == 1'b1) begin
                next_state = state;
                pulse = 0;
            end
            else begin
                next_state = `wait_for_one;
                pulse = 0;
            end

        `wait_for_one:
            if(in == 1'b1) begin
                next_state = `wait_for_zero;
                pulse = 1;
            end
            else begin
                next_state = state;
                pulse = 0;
            end

        default:
        begin
            next_state = `wait_for_one;
            pulse = 1'bx;
        end
    endcase
end

    always@(posedge clk or posedge reset)
        if(reset)
            state <= `wait_for_one;
        else
            state <= next_state;

endmodule
```

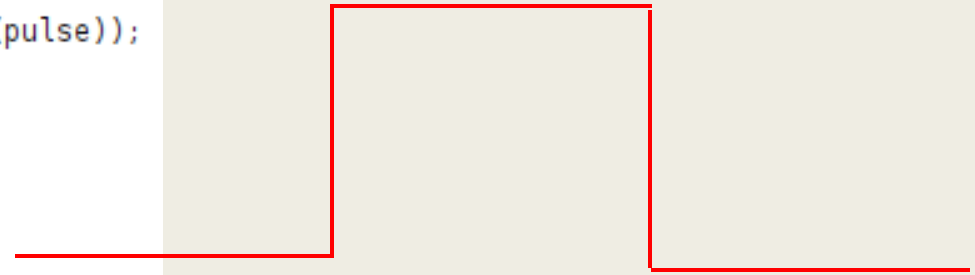
D flip flop

Verilog testbench

```
module pulse_generator_tb();
    reg in;
    reg clk;
    reg reset;
    wire pulse;
    pulse_generator U(.in(in),.clk(clk),.reset(reset),.pulse(pulse));

    initial begin
        clk = 0; reset = 1; in = 0;
        #125 reset = 0;
        #10
        @(posedge clk)
            in <= 1;
        #50 in = 0;
    end

    always #5 clk <= ~clk;

endmodule
```

If we use blocking assignment there, then the waveform will be wrong!

five clock period



Why?

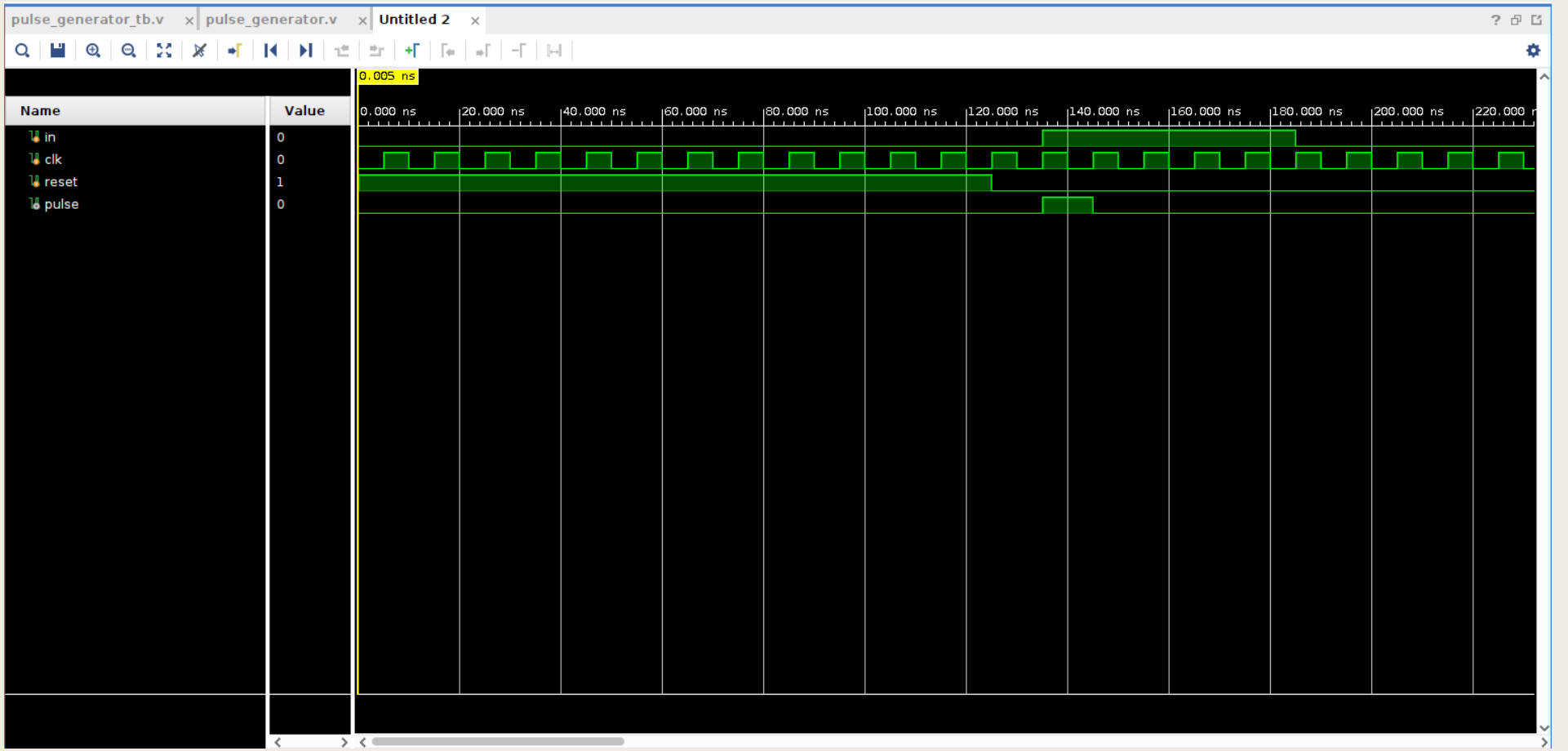
```
    reg state;
    reg next_state;

    always@* begin
        case(state)
            `wait_for_zero:
                if(in == 1'b1) begin
                    next_state = state;
                    pulse = 0;
                end
                else begin
                    next_state = `wait_for_one;
                    pulse = 0;
                end

            `wait_for_one:
                if(in == 1'b1) begin
                    next_state = `wait_for_zero;
                    pulse = 1;
                end
                else begin
                    next_state = state;
                    pulse = 0;
                end

            default:
            begin
                next_state = `wait_for_one;
                pulse = 1'bx;
            end
        endcase
    end
```

```
    always@(posedge clk or posedge reset)
        if(reset)
            state <= `wait_for_one;
        else
            state <= next_state;


endmodule
```

```
module pulse_generator_tb();
    reg in;
    reg clk;
    reg reset;
    wire pulse;
    pulse_generator U(.in(in),.clk(clk),.reset(reset),.pulse(pulse));

    initial begin
        clk = 0; reset = 1; in = 0;
        #125 reset = 0;
        #10
        @(posedge clk)
            in <= 1;
        #50 in = 0;
    end


    always #5 clk <= ~clk;

endmodule
```

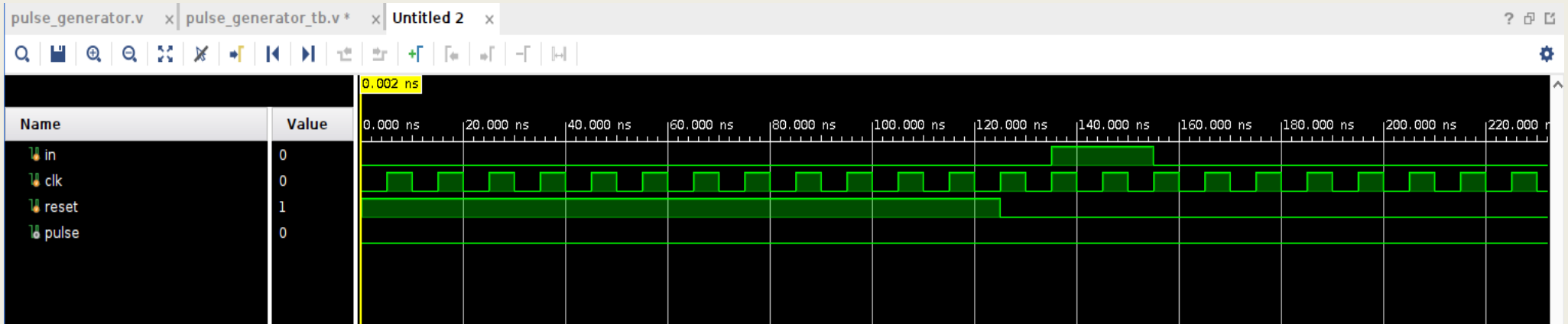
when 135ns:
1.RHS of clk <= ~clk;
2.clk change from 0 to 1
3-1.RHS of in <= 1;
3-2.RHS of state <= next_state
4-1.in become 1
4-2.state remain wait for one
5.pulse become 1

Waveform(Verilog version)



If we do in = 1 before positive edge ?

```
module pulse_generator_tb();
    reg in;
    reg clk;
    reg reset;
    wire pulse;
    pulse_generator U(.in(in),.clk(clk),.reset(reset),.pulse(pulse));

    initial begin

        clk = 0; reset = 1; in = 0;
        #125 reset = 0;
        #10 in = 1;
        #20 in = 0;
    end


    always #5 clk <= ~clk;

endmodule
```



Why?

```
    reg state;
    reg next_state;

    always@* begin
        case(state)
            `wait_for_zero:
                if(in == 1'b1) begin
                    next_state = state;
                    pulse = 0;
                end
                else begin
                    next_state = `wait_for_one;
                    pulse = 0;
                end

            `wait_for_one:
                if(in == 1'b1) begin
                    next_state = `wait_for_zero;
                    pulse = 1;
                end
                else begin
                    next_state = state;
                    pulse = 0;
                end

            default:
            begin
                next_state = `wait_for_one;
                pulse = 1'bx;
            end
        endcase
    end
```

```
    always@(posedge clk or posedge reset)
        if(reset)
            state <= `wait_for_one;
        else
            state <= next_state;


endmodule
```

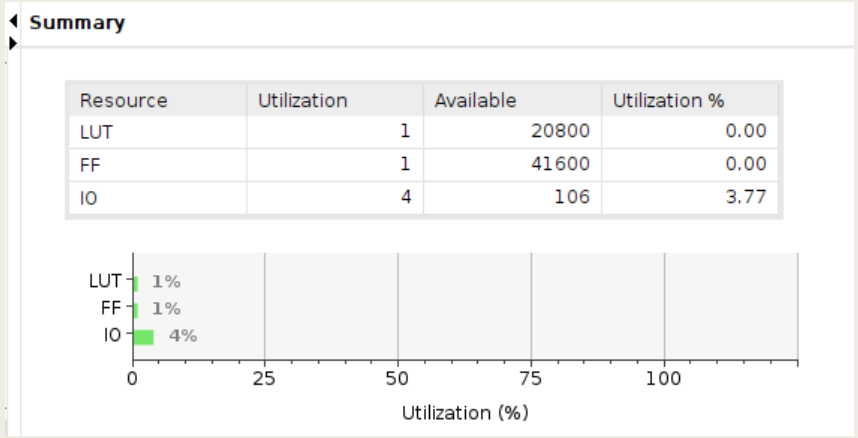
```
module pulse_generator_tb();
    reg in;
    reg clk;
    reg reset;
    wire pulse;
    pulse_generator U(.in(in),.clk(clk),.reset(reset),.pulse(pulse));

    initial begin

        clk = 0; reset = 1; in = 0;
        #125 reset = 0;
        #10 in = 1;
        #20 in = 0;
    end


    always #5 clk <= ~clk;

endmodule
```

When 135ns:

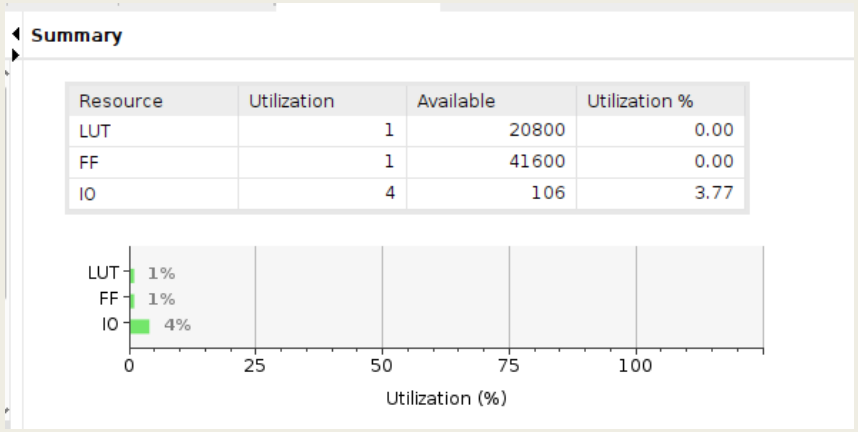
1. in = 1;
2. always@* block
3. next_state become `wait_for_zero
4. pulse become 1
5. RHS of clk <= ~clk decide
6. clk become 1
7. always@(posedge clk or posedge reset)
8. RHS of state <= next_state decide
9. state become `wait_for_zero
10. always@* block
11. next_state become `wait_for_zero
12. pulse become 0

Resource compare

HLS



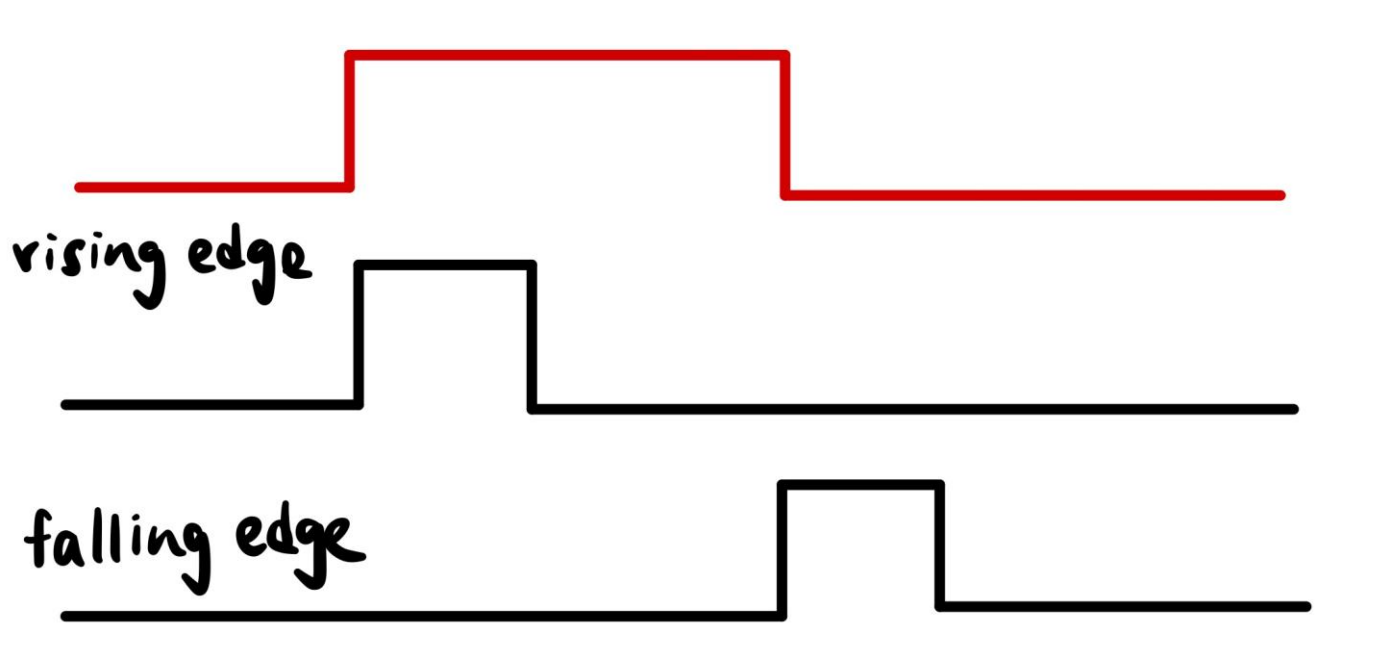
Verilog



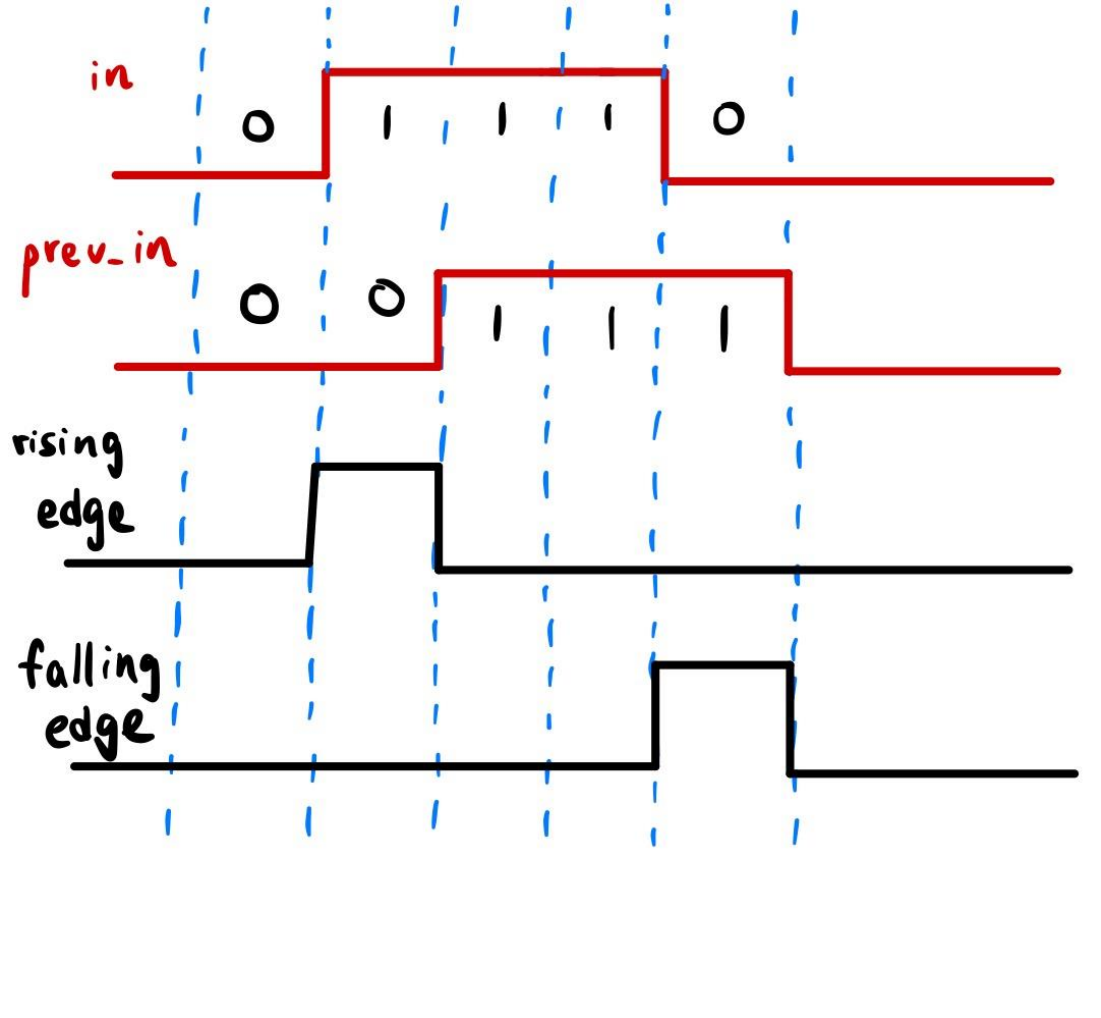
EDGE DETECTOR

What is edge detector?

similar to pulse generator



How do we implement an edge detector?

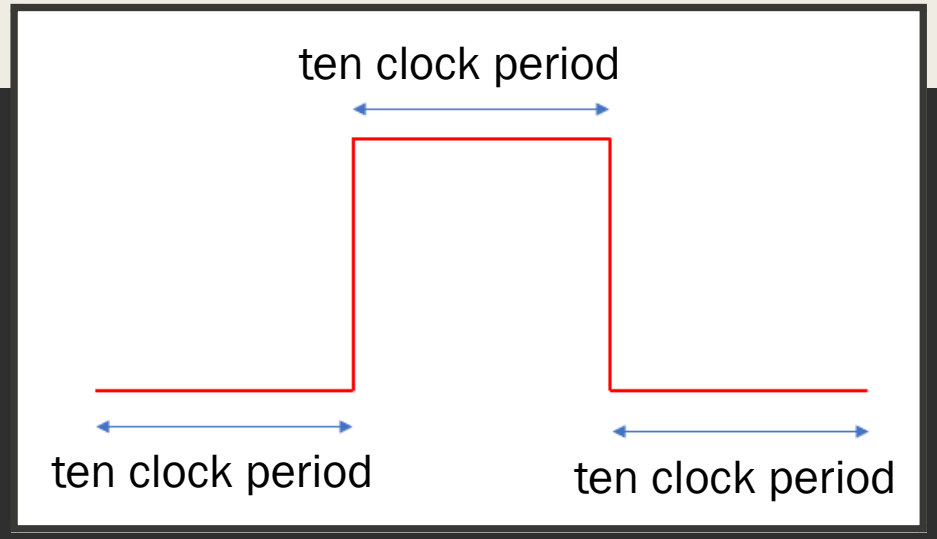


HLS code

```
#include "edge_detector.h"



void edge_detector(bool input_signal, bool &rising_edge, bool &faling_edge) {
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=input_signal
#pragma HLS INTERFACE ap_none port=rising_edge
#pragma HLS INTERFACE ap_none port=faling_edge



    static bool previous_input_signal = 0;



    if (previous_input_signal == 0 && input_signal == 1) {
        rising_edge = 1;
        faling_edge = 0;
    } else if (previous_input_signal == 1 && input_signal == 0) {
        rising_edge = 0;
        faling_edge = 1;
    } else {
        rising_edge = 0;
        faling_edge = 0;
    }

    previous_input_signal = input_signal;
}
```

HLS testbench

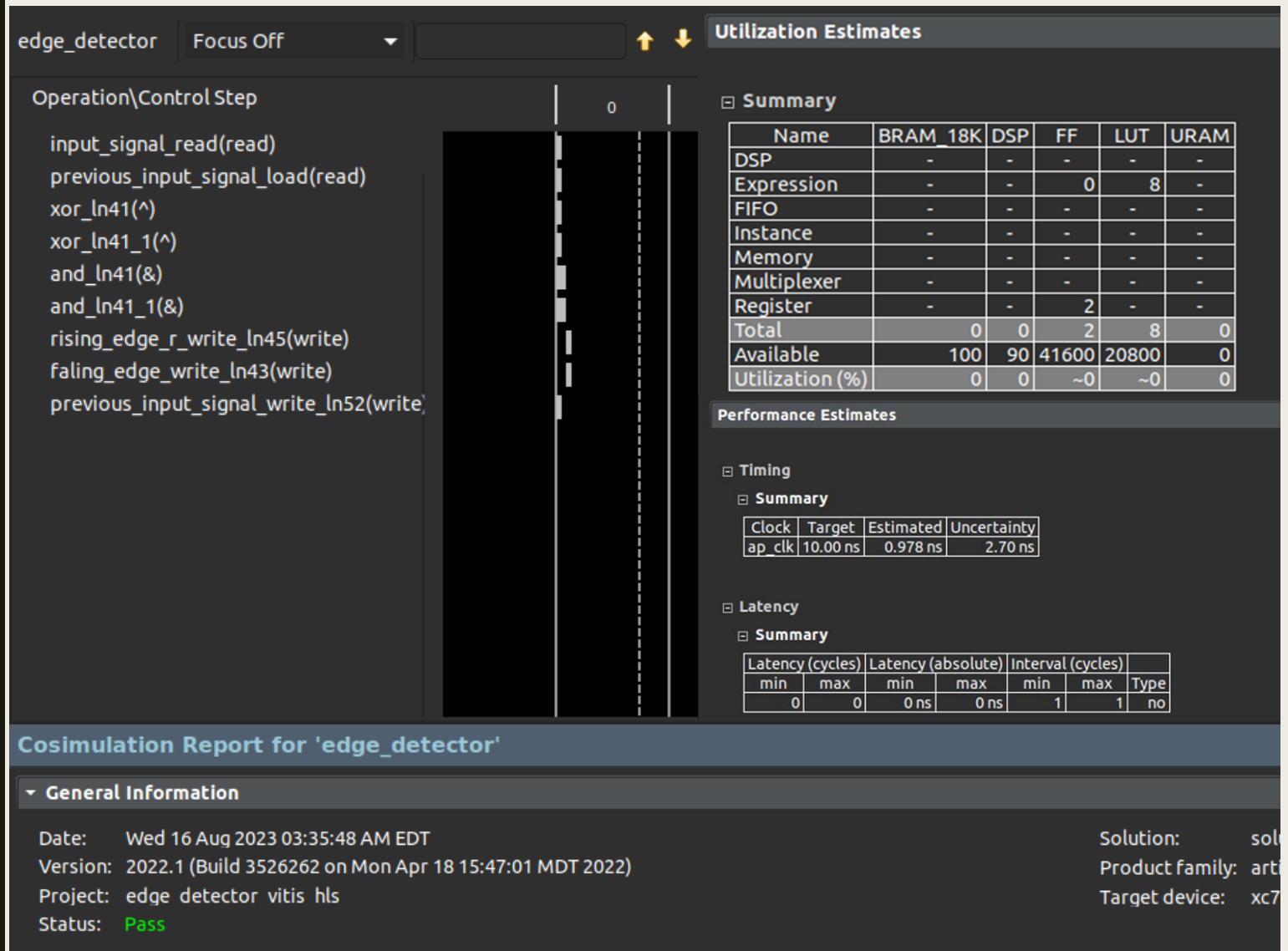
```cpp
#include "edge_detector-tb.h"
#include <iostream>



int main() {
    int status = 0;

    bool input_signal;
    bool rising_edge;
    bool faling_edge;

    for (int i = 0; i < 10; i++) {
        input_signal = 0;
        edge_detector(input_signal, rising_edge, faling_edge);
        std::cout << " input_signal = " << input_signal << " rising_edge = " << rising_edge <<  " faling_edge = " << faling_edge << std::endl;
    }

    for (int i = 0; i < 10; i++) {
        input_signal = 1;
        edge_detector(input_signal, rising_edge, faling_edge);
        std::cout << " input_signal = " << input_signal << " rising_edge = " << rising_edge <<  " faling_edge = " << faling_edge << std::endl;
    }


    for (int i = 0; i < 10; i++) {
        input_signal = 0;
        edge_detector(input_signal, rising_edge, faling_edge);
        std::cout << " input_signal = " << input_signal << " rising_edge = " << rising_edge <<  " faling_edge = " << faling_edge << std::endl;
    }



    return status;
}
```



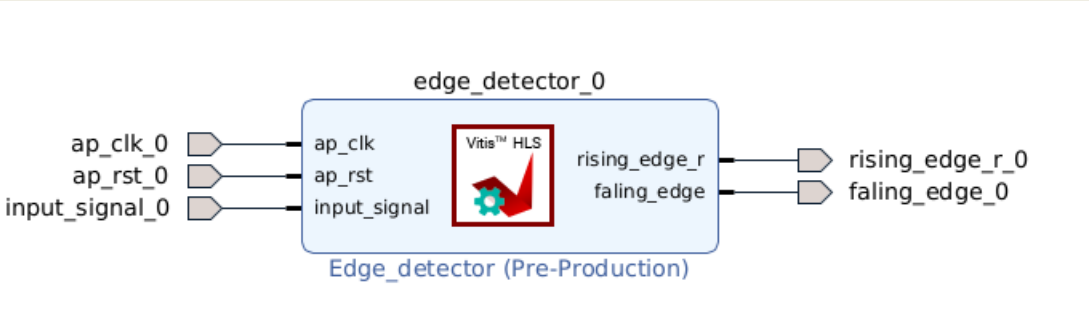
result—1

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../../lab_sequential/edge_detector/edge_detector-tb.cpp in debug mode
   Compiling ../../../../../lab_sequential/edge_detector/edge_detector.cpp in debug mode
   Generating csim.exe
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 1 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 1 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 1
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
 input_signal = 0 rising_edge = 0 faling_edge = 0
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************

```

result–2


edge_detector
Focus Off
Operation\Control Step
0
input_signal_read(read)
previous_input_signal_load(read)
xor_ln41(^)
xor_ln41_1(^)
and_ln41(&)
and_ln41_1(&)
rising_edge_r_write_ln45(write)
faling_edge_write_ln43(write)
previous_input_signal_write_ln52(write)
Utilization Estimates
Summary
Name | BRAM_18K | DSP | FF | LUT | URAM
DSP | - | - | - | - | -
Expression | - | - | 0 | 8 | -
FIFO | - | - | - | - | -
Instance | - | - | - | - | -
Memory | - | - | - | - | -
Multiplexer | - | - | - | - | -
Register | - | - | 2 | - | -
Total | 0 | 0 | 2 | 8 | 0
Available | 100 | 90 | 41600 | 20800 | 0
Utilization (%) | 0 | 0 | ~0 | ~0 | 0
Performance Estimates
Timing
Summary
Clock | Target | Estimated | Uncertainty
ap_clk | 10.00 ns | 0.978 ns | 2.70 ns
Latency
Summary
Latency (cycles) | Latency (absolute) | Interval (cycles)
min | max | min | max | min | max | Type
0 | 0 | 0 ns | 0 ns | 1 | 1 | no
Cosimulation Report for 'edge_detector'
General Information
Date: Wed 16 Aug 2023 03:35:48 AM EDT
Version: 2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)
Project: edge_detector_vitis_hls
Status: Pass
Solution:
Product family:
Target device:

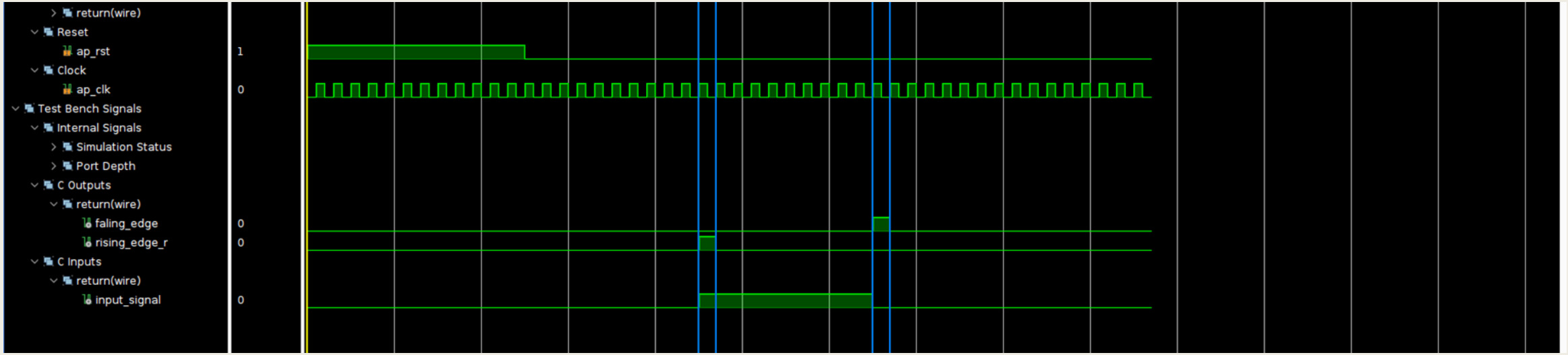
Block diagram



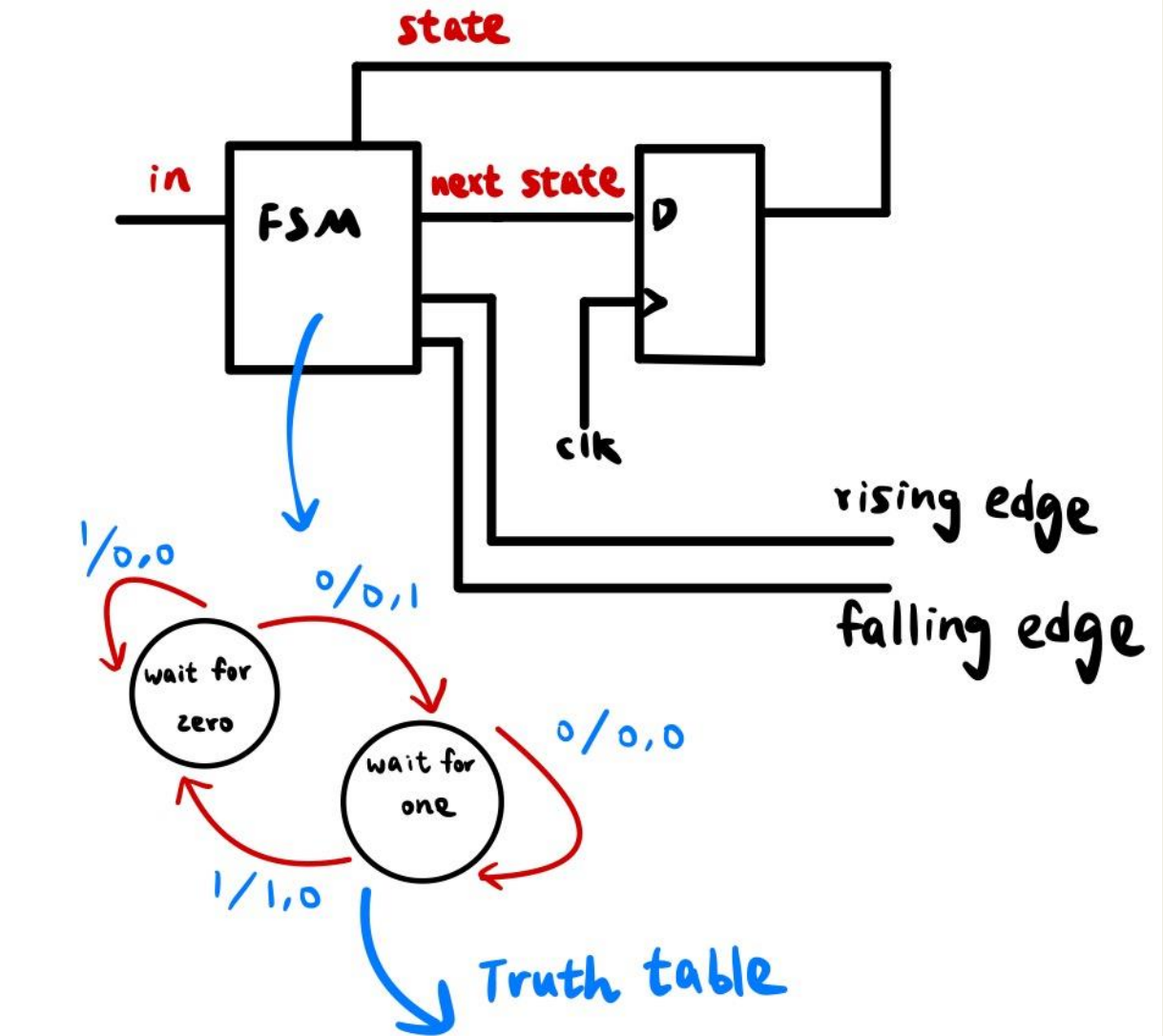
Tcl Console | Messages | Log | Reports | **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**	**8/16/2**	**00:00:**
impl_1	constrs_1	Not started																		
Out-of-Context Module Runs																				
design_1		Submodule Runs Complete																	8/16/2	00:00:
design_1_edc	design_1_edc	synth_design Complete!												1	1	0	0	0	8/16/2	00:00:

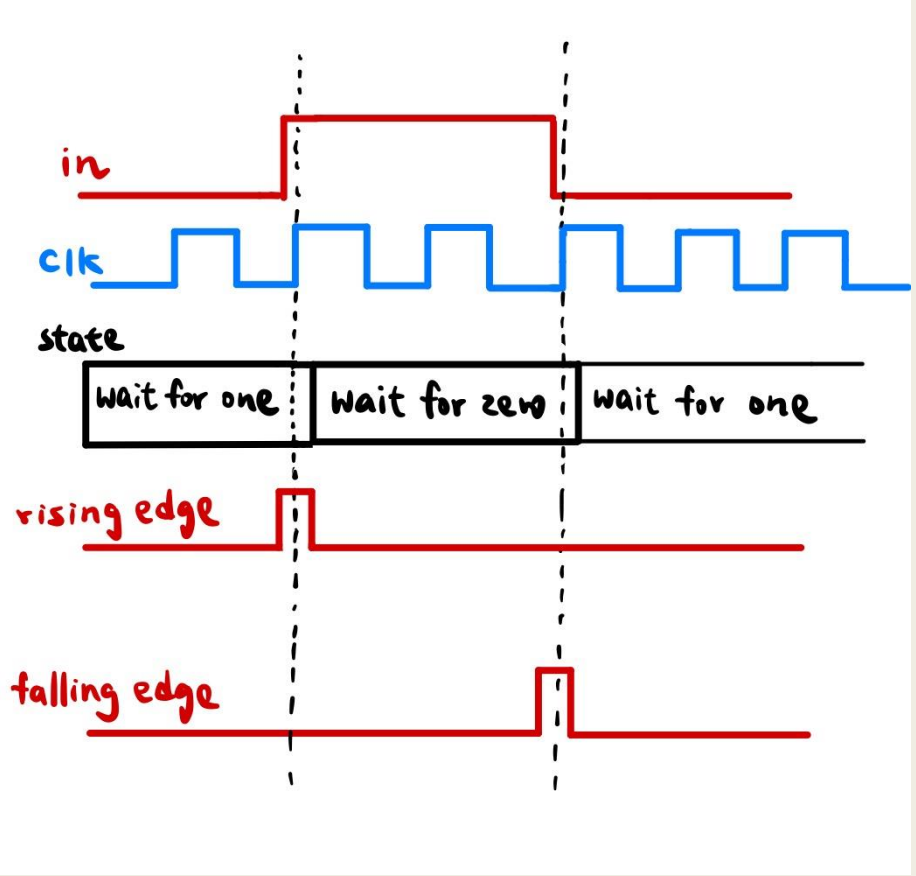
Waveform(HLS version)

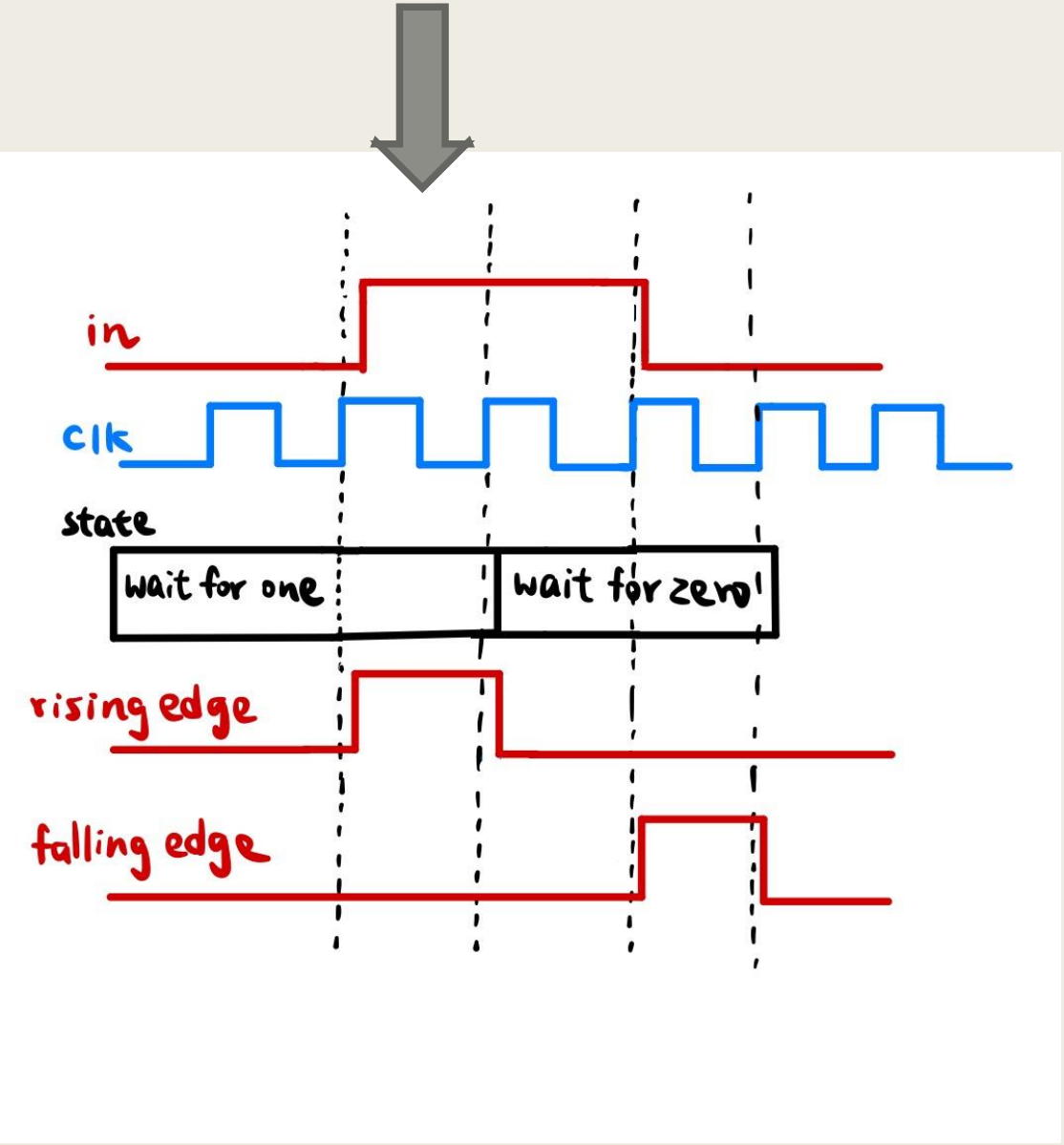


Logic structure



Timing waveform





Verilog code

FSM

```
`define wait_for_zero 1
`define wait_for_one 0

module edge_detector(
    input in,
    input clk,
    input reset,
    output reg rising_edge,
    output reg falling_edge
    );

    reg state;
    reg next_state;
```

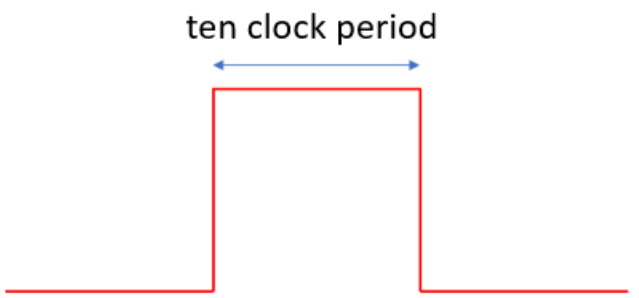
```
always@* begin
    case(state)
        `wait_for_zero:
            if(in == 1'b1) begin
                next_state = state;
                rising_edge = 0;
                falling_edge = 0;
            end
            else begin
                next_state = `wait_for_one;
                rising_edge = 0;
                falling_edge = 1;
            end

        `wait_for_one:
            if(in == 1'b1) begin
                next_state = `wait_for_zero;
                rising_edge = 1;
                falling_edge = 0;
            end
            else begin
                next_state = state;
                rising_edge = 0;
                falling_edge = 0;
            end

        default:
        begin
            next_state = `wait_for_one;
            rising_edge = 1'bx;
            falling_edge = 0;
        end
    endcase
end
```

```
    always@(posedge clk or posedge reset)
        if(reset)
            state <= `wait_for_one;
        else
            state <= next_state;


endmodule
```

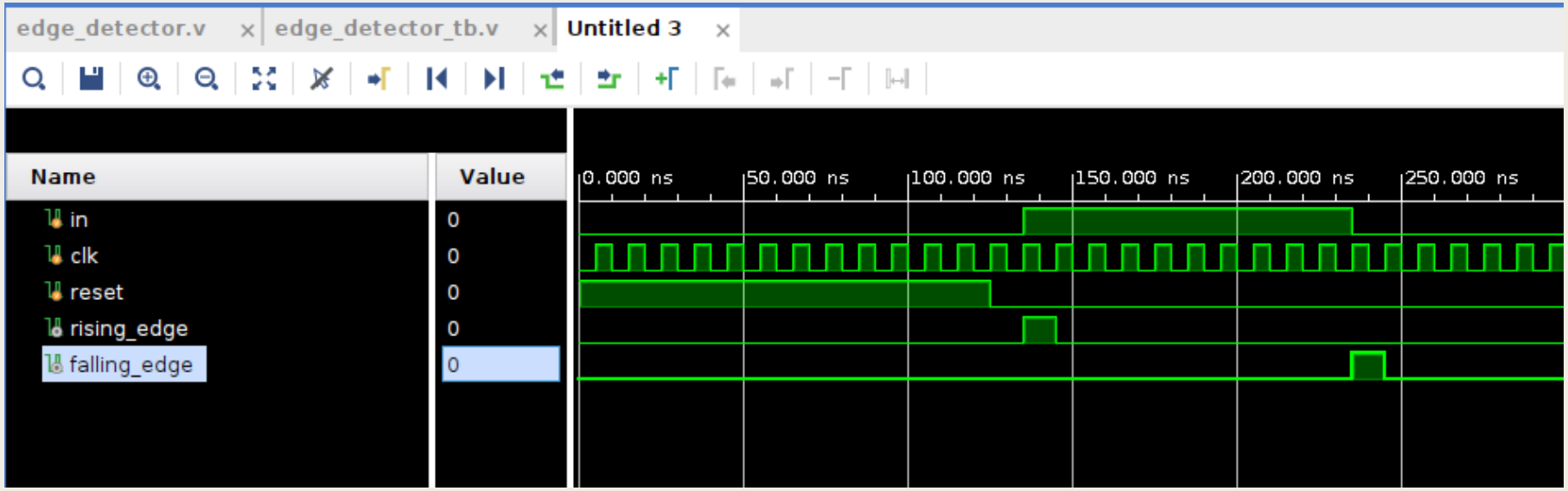
D flip flop

Verilog testbench

```
module edge_detector_tb();

    reg in;
    reg clk;
    reg reset;
    wire rising_edge;
    wire falling_edge;

    edge_detector U(.in(in),.clk(clk),.reset(reset),.rising_edge(rising_edge),.falling_edge(falling_edge));

    initial begin
        clk = 0; reset = 1; in = 0;
        #125 reset = 0;
        #10
        @(posedge clk)
            in <= 1;
        #100
        @(posedge clk)
            in <= 0;
    end

    always #5 clk <= ~clk;

endmodule
```



Waveform(Verilog version)

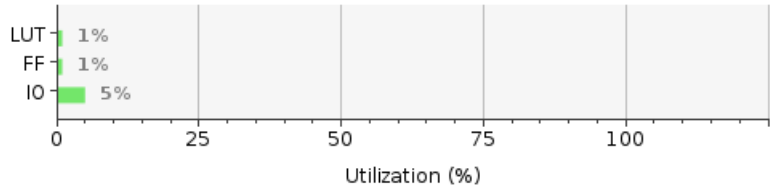


Resource compare

HLS

Summary

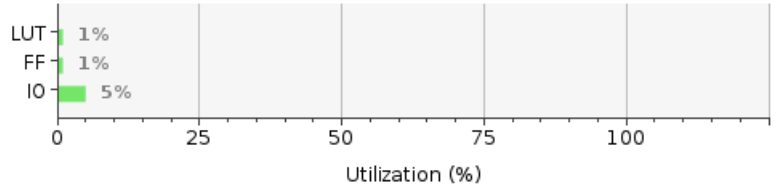
Resource	Utilization	Available	Utilization %
LUT	1	20800	0.00
FF	1	41600	0.00
IO	5	106	4.72



Verilog

Summary

Resource	Utilization	Available	Utilization %
LUT	1	20800	0.00
FF	1	41600	0.00
IO	5	106	4.72



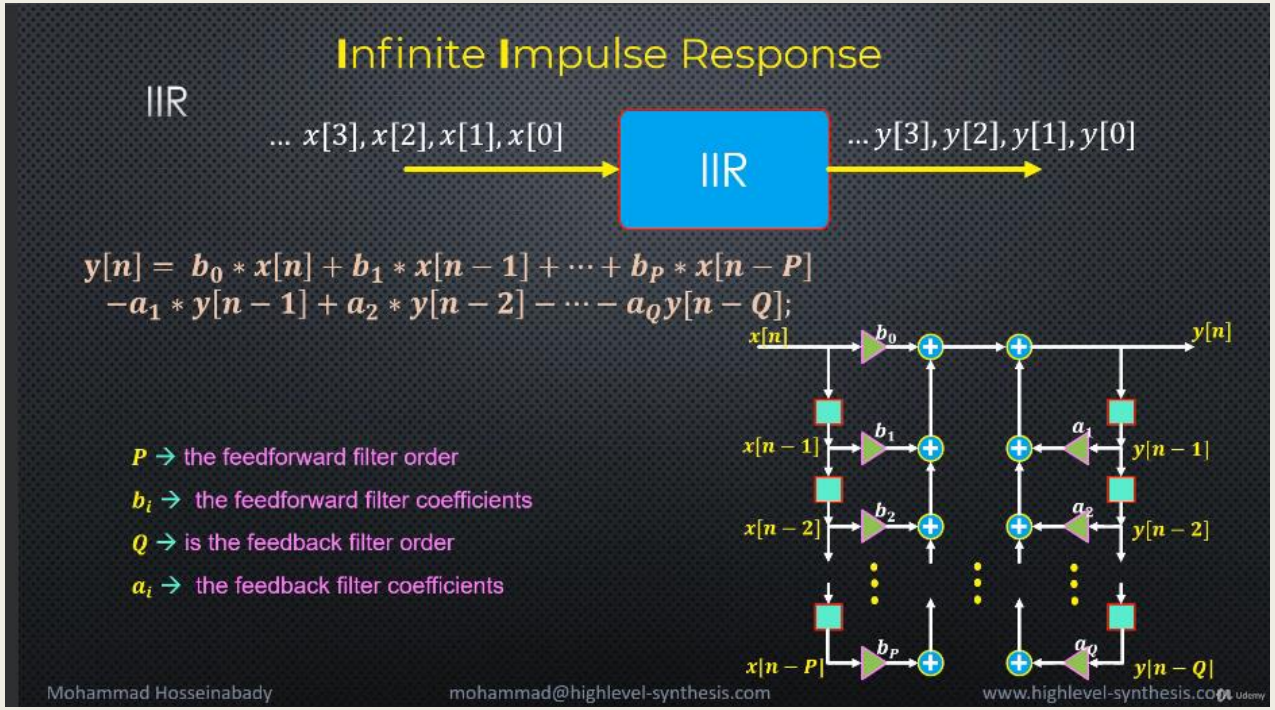
IIR

What is iir

infinite impulse response filter

Modify the frequency spectrum of the input signal through a certain computational relationship.

Ex. y[n] = b0*x[n] + b1*x[n-1] + b2*x[n-2] + …. – a1*y[n-1] – a2*y[n-2] - ……



HLS code

```
void iir(DATA_TYPE x, DATA_TYPE &y)
{
#pragma HLS INTERFACE ap_ctrl_hs port=return
#pragma HLS INTERFACE ap_none port=x
#pragma HLS INTERFACE ap_none port=y

//#pragma HLS PIPELINE
    static DATA_TYPE xn1 = 0;
    static DATA_TYPE xn2 = 0;

    static DATA_TYPE yn1 = 0;
    static DATA_TYPE yn2 = 0;

    DATA_TYPE  xn = x;
    DATA_TYPE  yn;

    yn = b0*xn+b1*xn1+b2*xn2-a1*yn1-a2*yn2;

    xn2 = xn1;
    xn1 = xn;

    yn2 = yn1;
    yn1 = yn;

    y = yn;

}
```

HLS testbench

```cpp
#include "iir-tb.h"


void iir_golden(DATA_TYPE *x, DATA_TYPE *y, int n) {


    y[0] = b0*x[0];
    y[1] = b0*x[1]+b1*x[0]-a1*y[0];

    for (int i = 2; i < n; i++) {
        y[i] = b0*x[i]+b1*x[i-1]+b2*x[i-2]-a1*y[i-1]-a2*y[i-2];
    }

}
```

```cpp
int main() {

    int status = 0;

    DATA_TYPE x[N] = {1, 2, 3, 4, 5, 6, 7, 8};
    DATA_TYPE y_hw[N] = {0};
    DATA_TYPE y_golden[N] = {0};

    for (int i = 0; i < N; i++) {
        iir(x[i], y_hw[i]);
        std::cout << "y_hw[" << i <<"] = " << y_hw[i] << std::endl;
    }

    iir_golden(x, y_golden, N);


    for (int i = 0; i < N; i++) {
        if (y_hw[i]  != y_golden[i]) {
            status = -1;
            std::cout << " Error at " << i
                      <<  " y_hw[" << i <<"] = " << y_hw[i]
                     <<  " y_golden[" << i <<"] = " << y_golden[i]
                                                    << std::endl;
            break;
        }
    }

    if(status == 0) {
        std::cout << "Test Passed! " << std::endl;
    } else {
        std::cout << "Test Failed! " << std::endl;
    }

    return status;

}
```

Result(10ns)

iir.cpp | iir.h | iir-tb.cpp | iir-tb.h | iir_csim.log ×

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../../lab_sequential/iir/iir-tb.cpp in debug mode
   Compiling ../../../../../lab_sequential/iir/iir.cpp in debug mode
   Generating csim.exe
y_hw[0] = 6
y_hw[1] = -11
y_hw[2] = 48
y_hw[3] = -128
y_hw[4] = 408
y_hw[5] = -1199
y_hw[6] = 3630
y_hw[7] = -10856
Test Passed!
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	231	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	17	-
Register	-	-	193	-	-
Total	0	0	193	248	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	~0	1	0

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	7.207 ns	2.70 ns

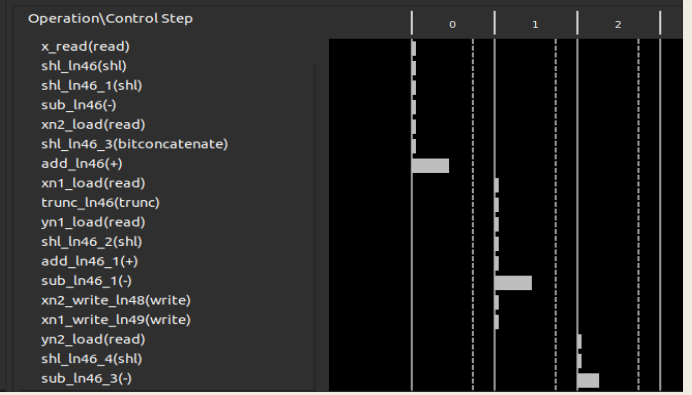
Latency

Summary

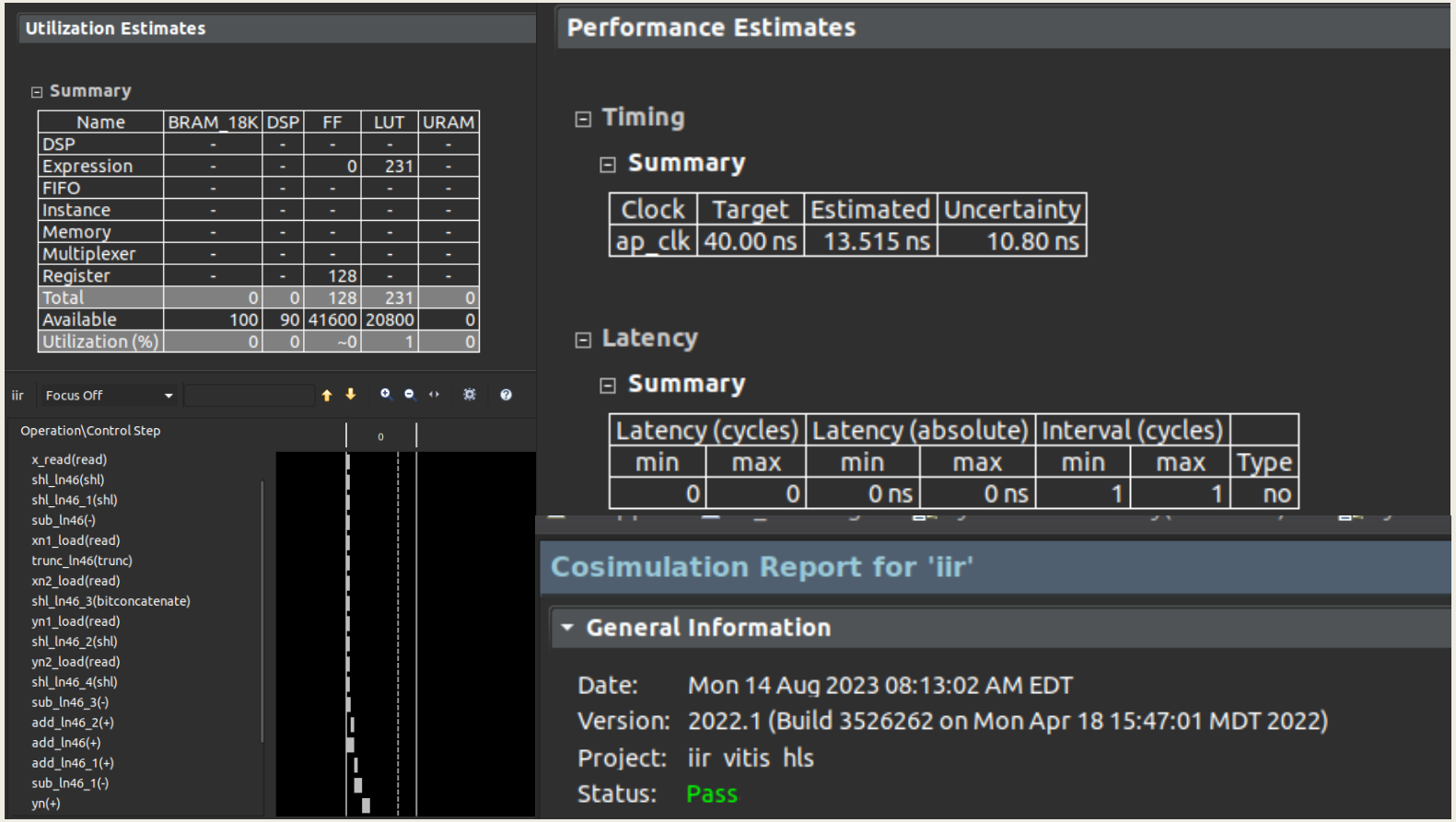
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
2	2	20.000 ns	20.000 ns	3	3	no

iir.cpp | iir_csim.log | Synthesis Summary(solution1) | Synthesis Details(solutio

iir | Focus Off



Result(40ns)



Utilization Estimates

Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	231	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	128	-	-
Total	0	0	128	231	0
Available	100	90	41600	20800	0
Utilization (%)	0	0	~0	1	0

iir Focus Off

Operation\Control Step

x_read(read)
shl_ln46(shl)
shl_ln46_1(shl)
sub_ln46(-)
xn1_load(read)
trunc_ln46(trunc)
xn2_load(read)
shl_ln46_3(bitconcatenate)
yn1_load(read)
shl_ln46_2(shl)
yn2_load(read)
shl_ln46_4(shl)
sub_ln46_3(-)
add_ln46_2(+)
add_ln46(+)
add_ln46_1(+)
sub_ln46_1(-)
yn(+)

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	40.00 ns	13.515 ns	10.80 ns

Latency

Summary

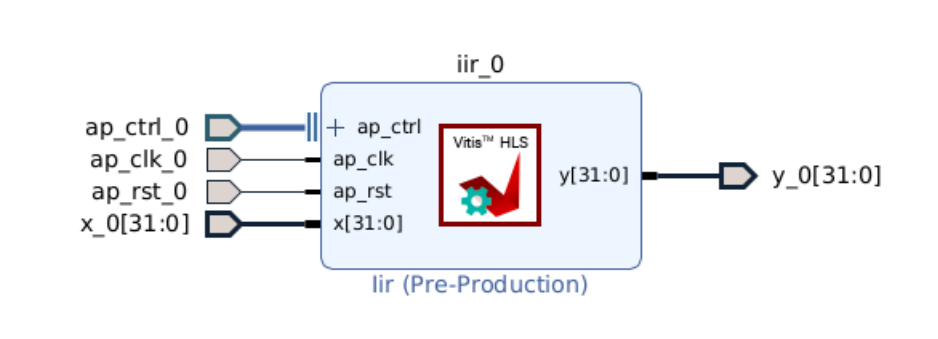
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
0	0	0 ns	0 ns	1	1	no

Cosimulation Report for 'iir'

General Information

Date: Mon 14 Aug 2023 08:13:02 AM EDT
Version: 2022.1 (Build 3526262 on Mon Apr 18 15:47:01 MDT 2022)
Project: iir vitis hls
Status: Pass

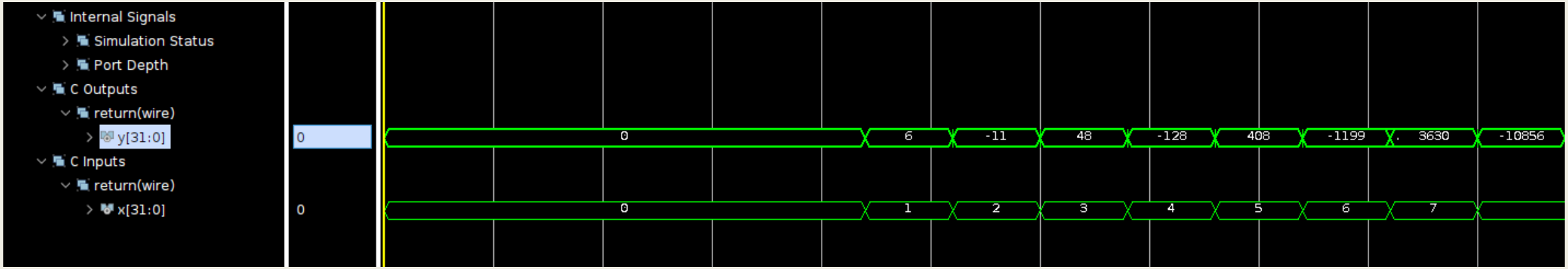
Block diagram



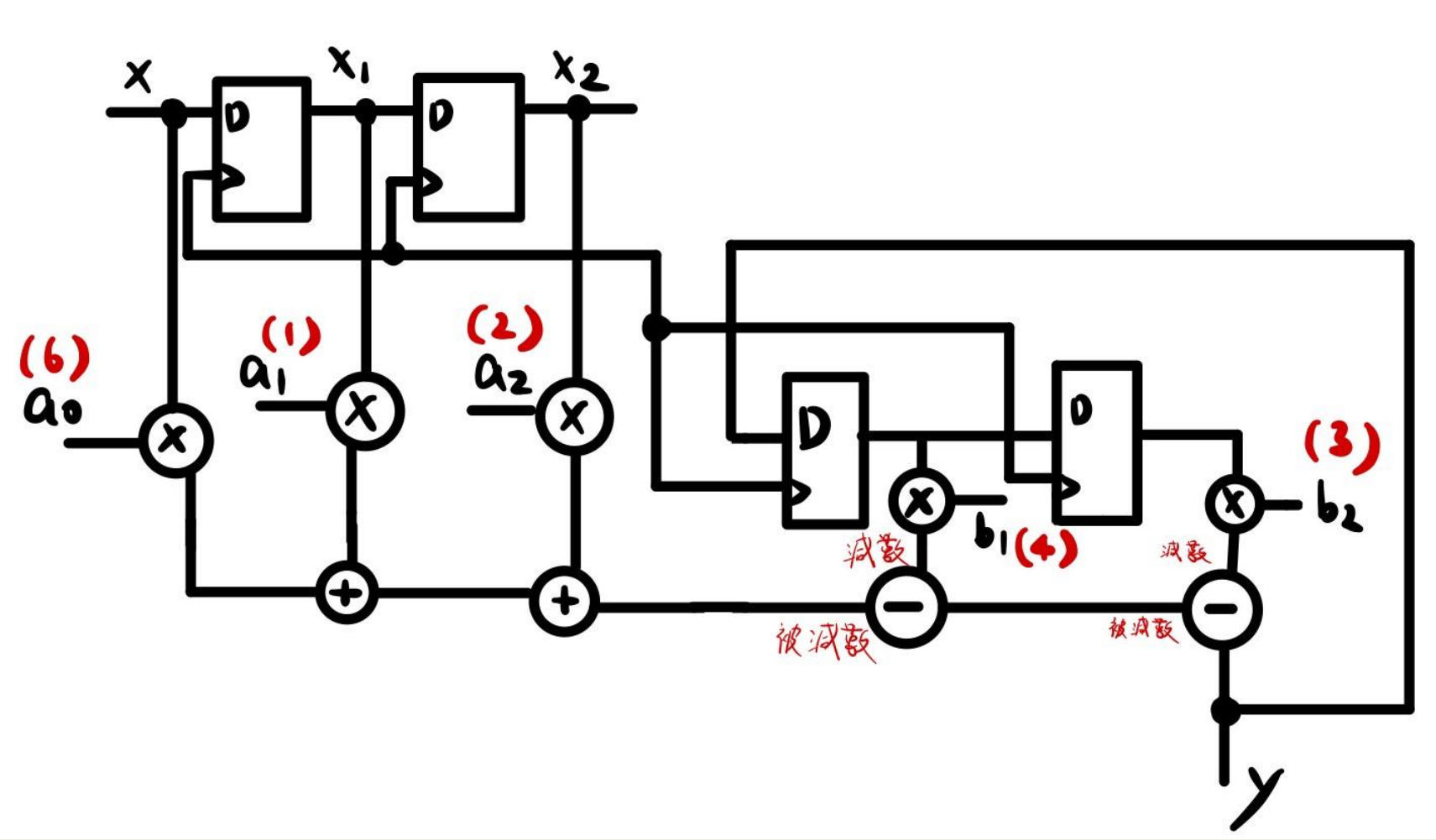
Tcl Console | Messages | Log | Reports | **Design Runs**

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**	**8/17/2**	**00:00:**
impl_1	constrs_1	Not started																		
Out-of-Context Module Runs																				
design_1		Submodule Runs Complete																	8/17/2	00:00:5
design_1_iir_(	design_1_iir_(	synth_design Complete!												152	127	0	0	0	8/17/2	00:00:5

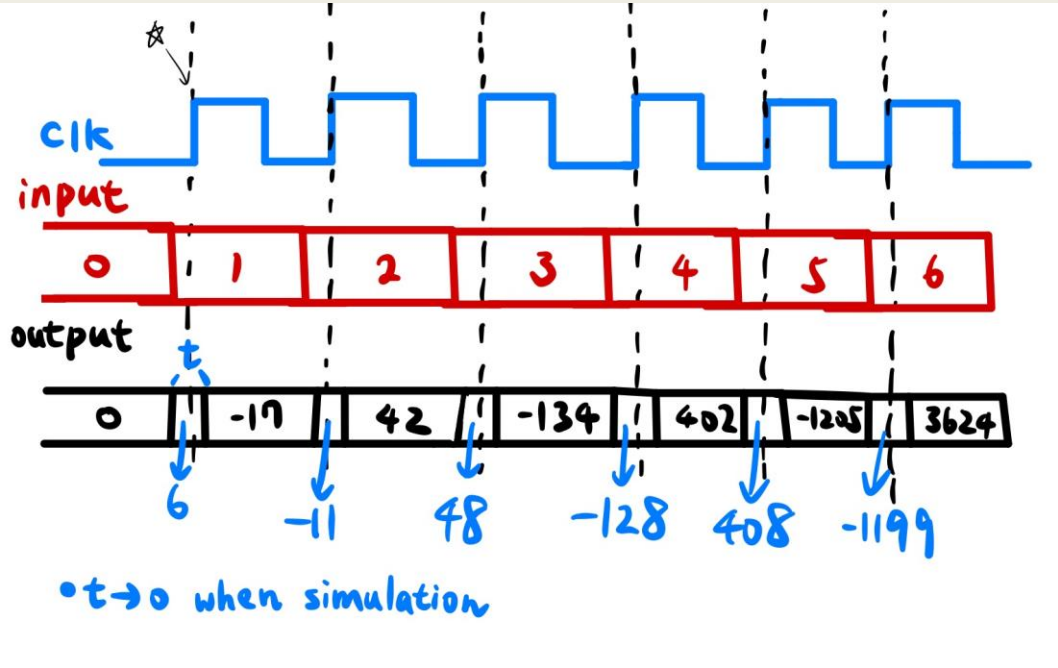
Waveform(HLS)

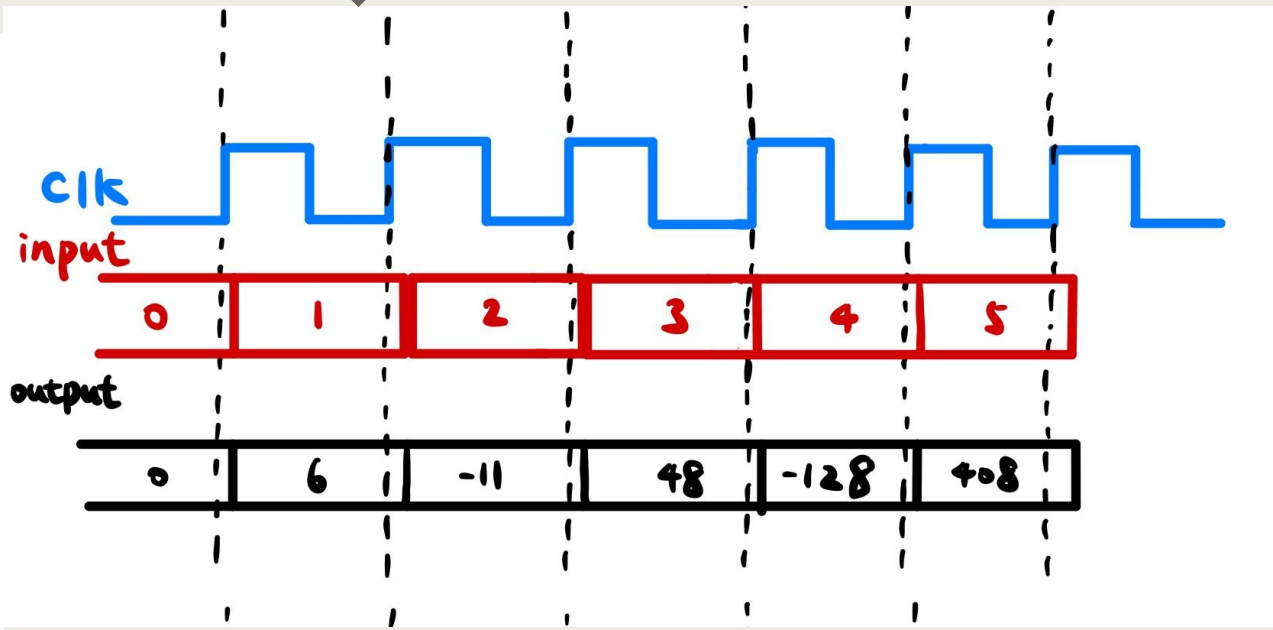


Logic structure

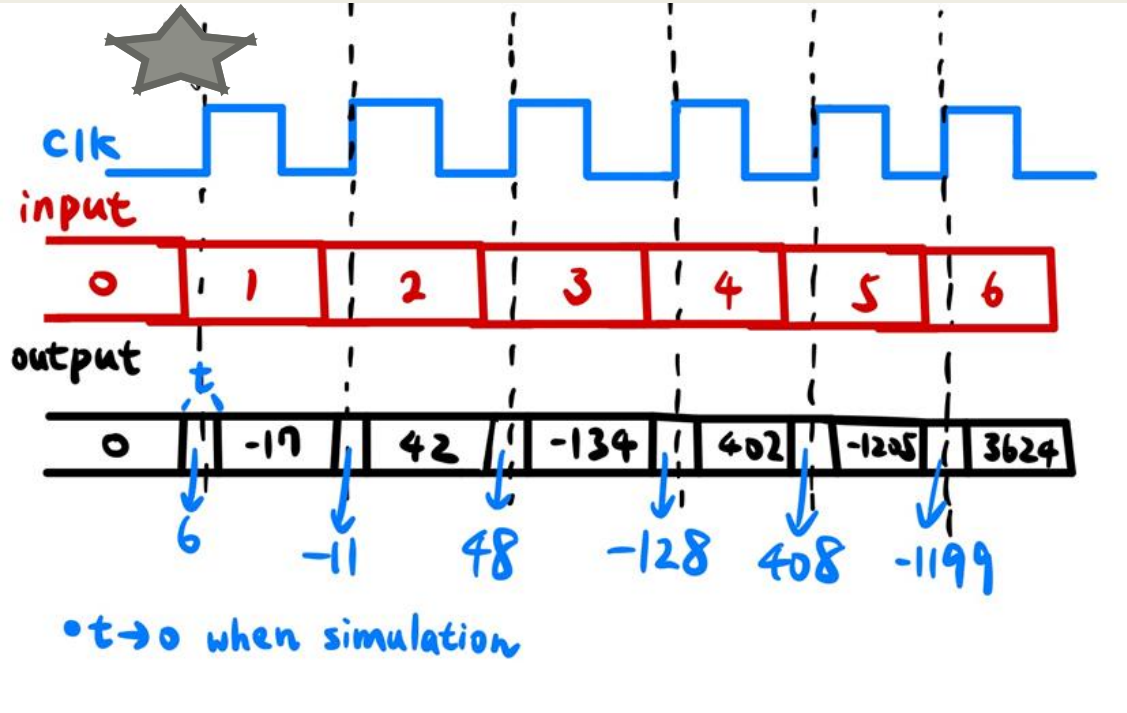


Timing waveform



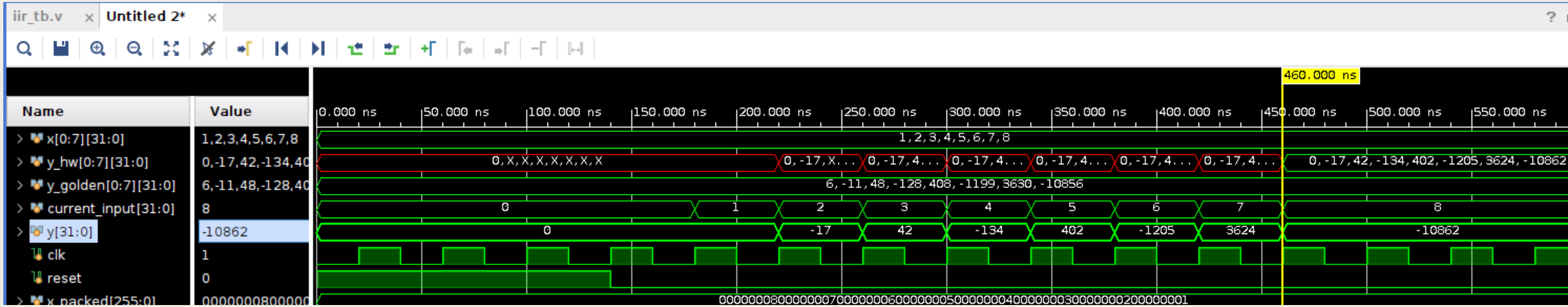


Why?



Star position:
1.before positive edge:
x = 1, x1 = 0, x2=0, y1=0, y2=0 → y = 6 (correct)
2.between positive edge and updating register:
x = 1, x1 = 0, x2=0, y1=0, y2=0 → y = 6 (correct)
3.after updating register:
x = 1, x1 = 1, x2=0, y1=6, y2=0 → y = -17 (wrong)

Wrong waveform



Verilog code

```
`define a1 32'sd4
`define a2 32'sd3
`define b0 32'sd6
`define b1 32'sd1
`define b2 32'sd2

module iir(
    input signed [31:0] x,
    input clk,
    input reset,
    output reg signed [31:0] y
    );

    reg signed [31:0] x1;
    reg signed [31:0] x2;
    reg signed [31:0] y1;
    reg signed [31:0] y2;
```

```
always@* begin
    y = `b0*x+`b1*x1+`b2*x2-`a1*y1-`a2*y2;
end
```

4 * 32 d flip flop

```
    always@(posedge clk or posedge reset)
        if(reset)
            x1 <= 32'sd0;
        else
            x1 <= x;

    always@(posedge clk or posedge reset)
        if(reset)
            x2 <= 32'sd0;
        else
            x2 <= x1;

    always@(posedge clk or posedge reset)
        if(reset)
            y1 <= 32'sd0;
        else
            y1 <= y;

    always@(posedge clk or posedge reset)
        if(reset)
            y2 <= 32'sd0;
        else
            y2 <= y1;

endmodule
```

Verilog testbench

```
`define a1 32'sd4
`define a2 32'sd3
`define b0 32'sd6
`define b1 32'sd1
`define b2 32'sd2

module iir_tb();

    reg signed [31:0] x [0:7];         //do int x[N] = {1, 2, 3, 4, 5, 6, 7, 8};
    reg signed [31:0] y_hw [0:7];      //do int y_hw[N] = {0};
    reg signed [31:0] y_golden [0:7]; //do DATA_TYPE y_golden[N] = {0};
    reg clk;
    reg reset;
    reg signed [31:0] current_input;
    reg [255:0] x_packed;
    reg [255:0] y_packed;
    wire signed [31:0] current_output;
    integer i;

    iir U(.x(current_input),.clk(clk),.reset(reset),.y(current_output));

  initial begin
      x[0] = 32'sd1; x[1] = 32'sd2; x[2] = 32'sd3; x[3] = 32'sd4;
      x[4] = 32'sd5; x[5] = 32'sd6; x[6] = 32'sd7; x[7] = 32'sd8;
      y_hw[0] = 32'sd0; y_golden[0] = 32'sd0;
      clk = 0; reset = 1; current_input = 0;

      // golden model
      for (i = 0; i < 8; i = i + 1) begin
          x_packed[i*32 +: 32] = x[i];
      end

      iir_golden(x_packed,y_packed);

      for (i = 0; i < 8; i = i + 1) begin
          y_golden[i] = y_packed[i*32 +: 32];
      end
      //
```

initialization

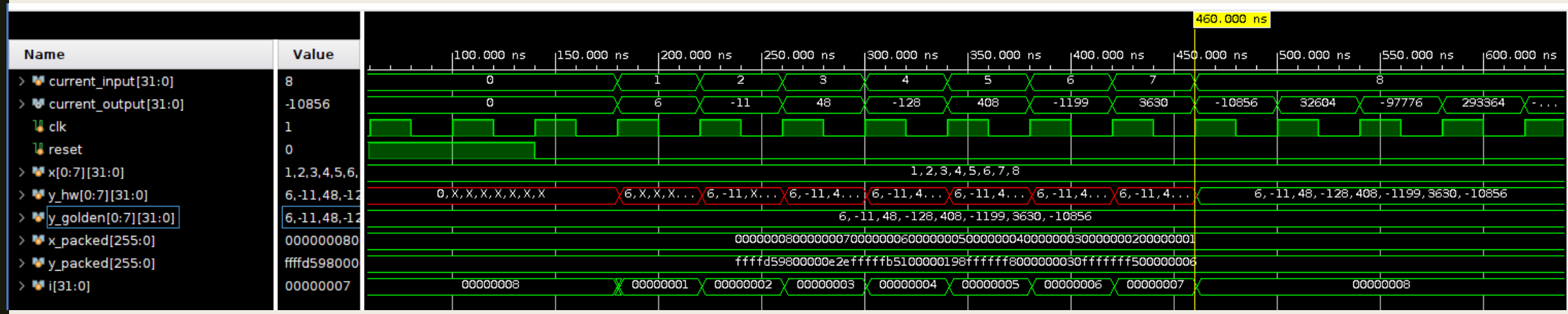
calculation of golden model

```
      #140 reset = 0;

      #40;

      for(i = 0; i < 8; i = i + 1) begin
          @(posedge clk)
              current_input <= x[i];
          #1 y_hw[i] = current_output;
          $display("y_hw[%0d] = %0d ", i, current_output);
      end

      //compare with golden model
      for(i = 0; i < 8; i = i + 1) begin
          if(y_hw[i] != y_golden[i])
              $display(" Error at %0d y_hw[%0d] = %0d y_golden[%0d] = %0d", i, i, y_hw[i], i, y_golden[i]);
      end
  end

  always #20 clk <= ~clk;
```

check

```
  task iir_golden;
      input [255:0] x_in;
      output reg [255:0] y_out;
      reg signed [31:0] x [0:7];
      reg signed [31:0] y [0:7];
      integer i;
      begin
          // Unpack the input values from the packed input
          for (i = 0; i < 8; i = i + 1) begin
              x[i] = x_in[i*32 +: 32];  // Slices out 32-bit segments from the packed input
          end

          y[0] = `b0 * x[0];
          y[1] = `b0 * x[1] + `b1 * x[0] - `a1 * y[0];

          for (i = 2; i < 8; i = i+1) begin
              y[i] = `b0 * x[i] + `b1 * x[i-1] + `b2 * x[i-2] - `a1 * y[i-1] - `a2 * y[i-2];
          end

          // Pack the internal array values into the packed output
          for (i = 0; i < 8; i = i + 1) begin
              y_out[i*32 +: 32] = y[i];  // Assigns 32-bit segments to the packed output
          end
      end
  endtask


endmodule
```

Waveform(Verilog)



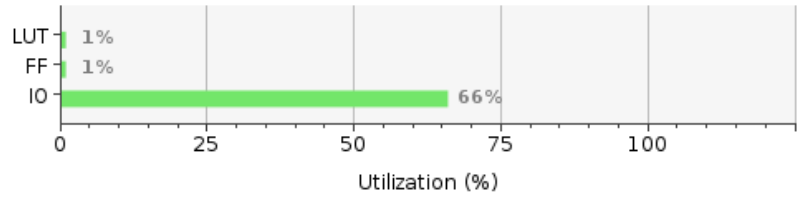

```
# run 1000ns
y_hw[0] = 6
y_hw[1] = -11
y_hw[2] = 48
y_hw[3] = -128
y_hw[4] = 408
y_hw[5] = -1199
y_hw[6] = 3630
y_hw[7] = -10856
INFO: [USF-XSim-96] XSim completed. Design snapshot 'iir_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

Resource compare

HLS

Summary

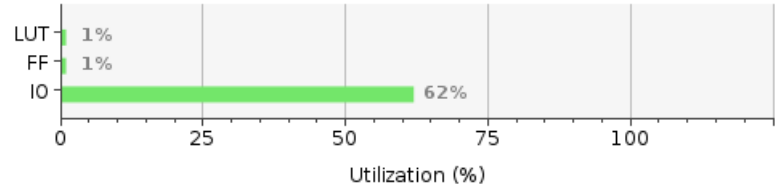
Resource	Utilization	Available	Utilization %
LUT	152	20800	0.73
FF	127	41600	0.31
IO	70	106	66.04



Verilog

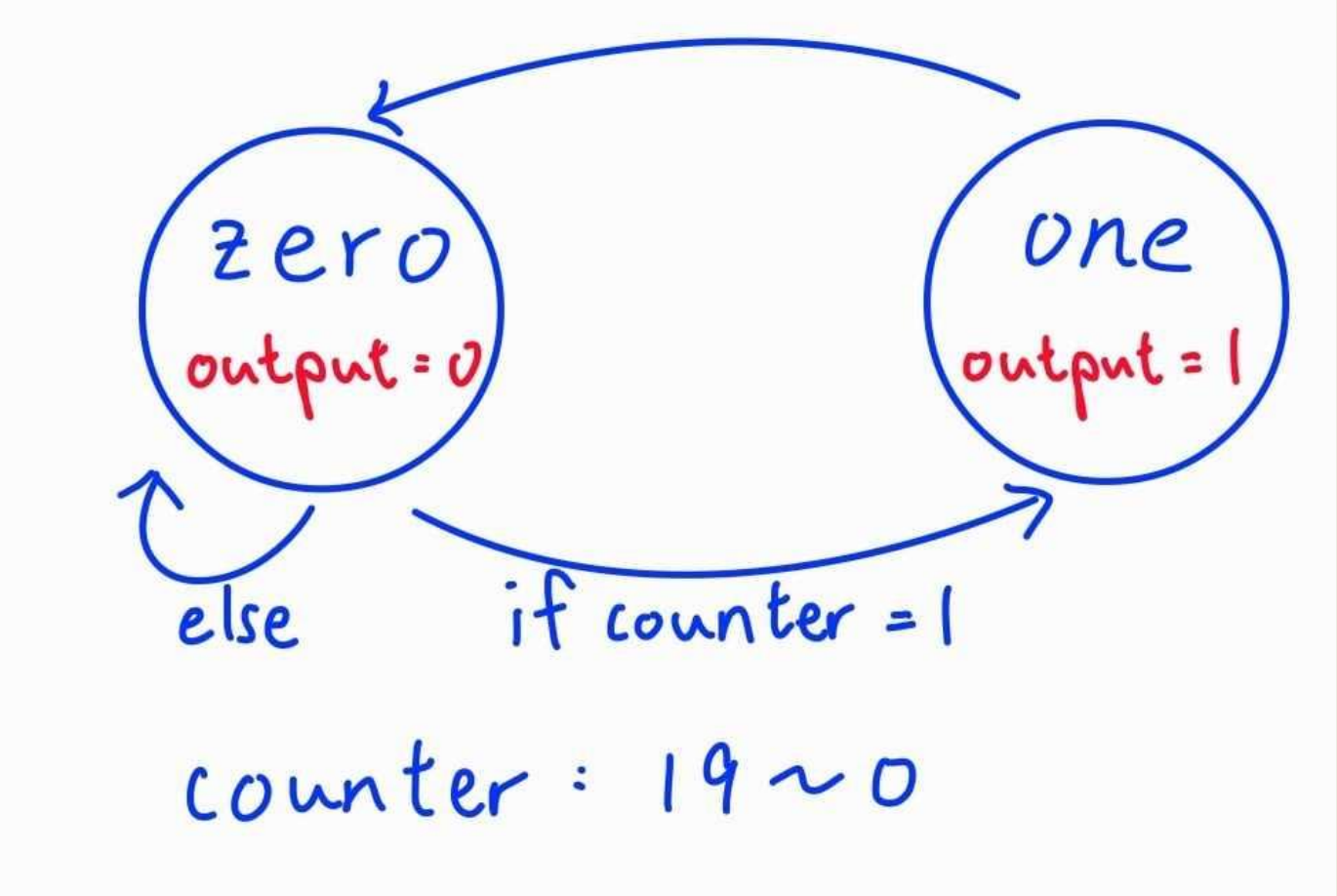
Summary

Resource	Utilization	Available	Utilization %
LUT	126	20800	0.61
FF	127	41600	0.31
IO	66	106	62.26



SINGLE CYCLE REGULAR PULSES

HLS Design



HLS code

```
#define PULSE_PERIOD 20

typedef enum{zero, one} states_type;
void single_cycle_regular_pulses(bool &periodic_pulses) {
#pragma HLS INTERFACE ap_none port=periodic_pulses
#pragma HLS INTERFACE ap_ctrl_none port=return

    static states_type state = zero;
    static unsigned int counter = PULSE_PERIOD-1;

    states_type  next_state;
    unsigned int next_counter;

    bool periodic_pulses_local;

    switch(state) {
    case zero:
        if (counter == 1) {
            next_counter            = counter-1;
            next_state              = one;
        } else {
            next_counter            = counter-1;
            next_state              = zero;
        }
        periodic_pulses_local = 0;
        break;
    case one:
        next_counter            = PULSE_PERIOD-1;
        next_state              = zero;
        periodic_pulses_local = 1;
        break;
    default:
        break;
    }

    state = next_state;
    counter = next_counter;
    periodic_pulses = periodic_pulses_local;
}
```

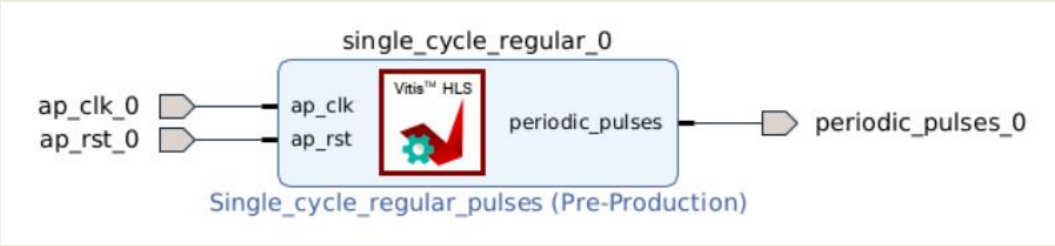
```cpp
int main() {

    int status = 0;

    bool periodic_pulses_local;
    std::cout <<  std::endl;
    std::cout <<  std::endl;

    for (int i = 0; i < 100; i++) {
        single_cycle_regular_pulses(periodic_pulses_local);
        std::cout << periodic_pulses_local ;
    }

    std::cout <<  std::endl;
    std::cout <<  std::endl;
    std::cout <<  std::endl;
    return status;
}
```

HLS testbench

```
00000010000000000000000010000000000000000010000000000000000010000000000000000010000000000000
```

Synthesis Result & Waveform

Synthesis Result



Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects	Source File
1	ap_clk_0	10.000	0.000	5.000	☑	[get_ports ap_clk_0]	<unsaved constraints>

Setup

Worst Negative Slack (WNS):	6.859 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	65

Hold

Worst Hold Slack (WHS):	0.316 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	65

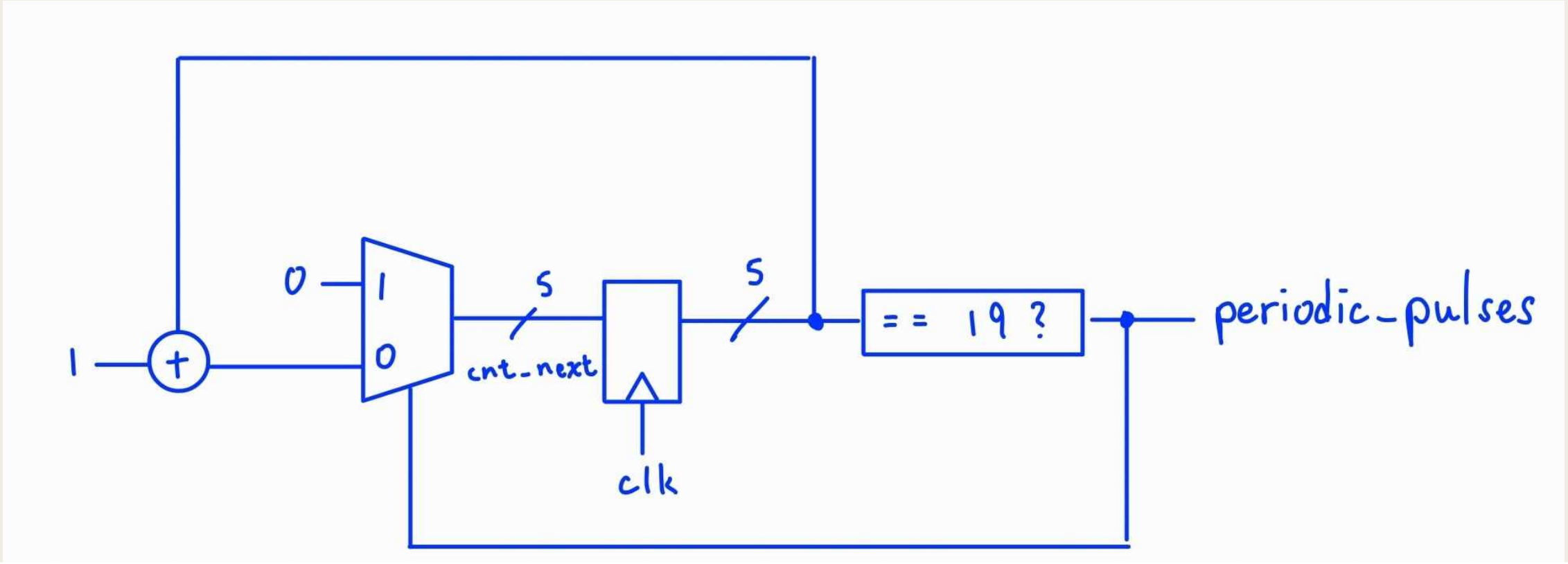
Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	34

All user specified timing constraints are met.

Resource	Utilization	Available	Utilization %
LUT	39	20800	0.19
FF	33	41600	0.08
IO	2	106	1.89
BUFG	1	32	3.13

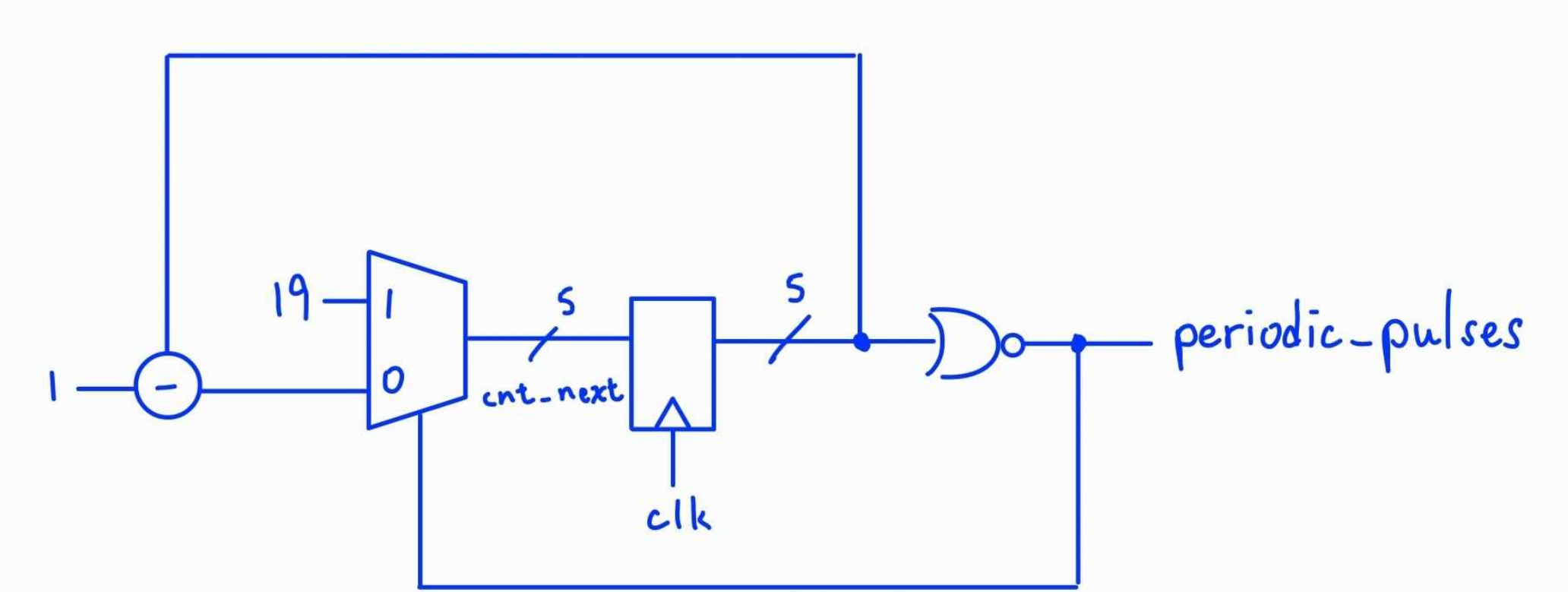
Verilog design



Cnt	0	1	…	18	19	0	1	…	18	19	0	1	2
output	0	0	…	0	1	0	0	…	0	1	0	0	0

Optimized design

Optimized or not ? (Both be implemented by LUT ?)



Cnt	19	18	…	1	0	19	18	…	1	0	19	18	17
output	0	0	…	0	1	0	0	…	0	1	0	0	0

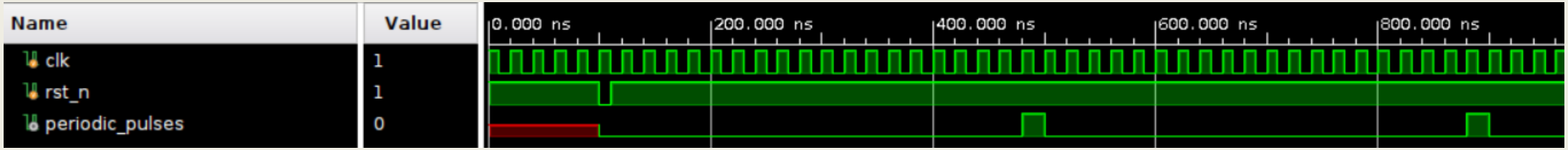
Verilog code

```
module single_cycle_regular_pulses(clk, rst_n, cnt, periodic_pulses);
    input clk;
    input rst_n;
    output reg[4:0] cnt;
    output reg periodic_pulses;

    reg [4:0] cnt_next;

    //combinational logic
    always@(*)
    begin
        periodic_pulses = ~|cnt;
        if (periodic_pulses)
        begin
            cnt_next = 19;
        end
        else
        begin
            cnt_next = cnt-1;
        end
    end

    //sequential logic
    always@(posedge clk or negedge rst_n)
    begin
        if (!rst_n)
        begin
            cnt <= 19;
        end
        else
            cnt <= cnt_next;
    end
endmodule
```

Verilog testbench

```
module single_cycle_regular_pulses_tb();

    reg clk;
    reg rst_n;
    wire[4:0] cnt;
    wire periodic_pulses;

    single_cycle_regular_pulses single_cycle_regular_pulses_0(.clk(clk), .rst_n(rst_n),
    .cnt(cnt), .periodic_pulses(periodic_pulses));

    always #10 clk = ~clk;

    initial
    begin
        clk       = 1;
        rst_n     = 1;
        #100 rst_n = 0;
        #10 rst_n = 1;
        #1000 $finish;
    end
endmodule
```

Simulation result



Synthesis result

Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects	Source File
1	clk	10.000	0.000	5.000	☐	[get_ports clk]	single_cycle_regular_pulses.xdc

Setup

Worst Negative Slack (WNS):	7.956 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

Hold

Worst Hold Slack (WHS):	0.325 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	6

All user specified timing constraints are met.

Resource	Estimation	Available	Utilization %
LUT	4	20800	0.02
FF	5	41600	0.01
IO	3	106	2.83
BUFG	1	32	3.13

Comparison

Utilization

HLS

Resource	Utilization	Available	Utilization %
LUT	39	20800	0.19
FF	33	41600	0.08
IO	2	106	1.89
BUFG	1	32	3.13

verilog

Resource	Estimation	Available	Utilization %
LUT	4	20800	0.02
FF	5	41600	0.01
IO	3	106	2.83
BUFG	1	32	3.13

Timing

HLS

Setup

Worst Negative Slack (WNS):	6.859 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	65

Hold

Worst Hold Slack (WHS):	0.316 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	65

Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	34

All user specified timing constraints are met.

verilog

Setup

Worst Negative Slack (WNS):	8.236 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

Hold

Worst Hold Slack (WHS):	0.146 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	5

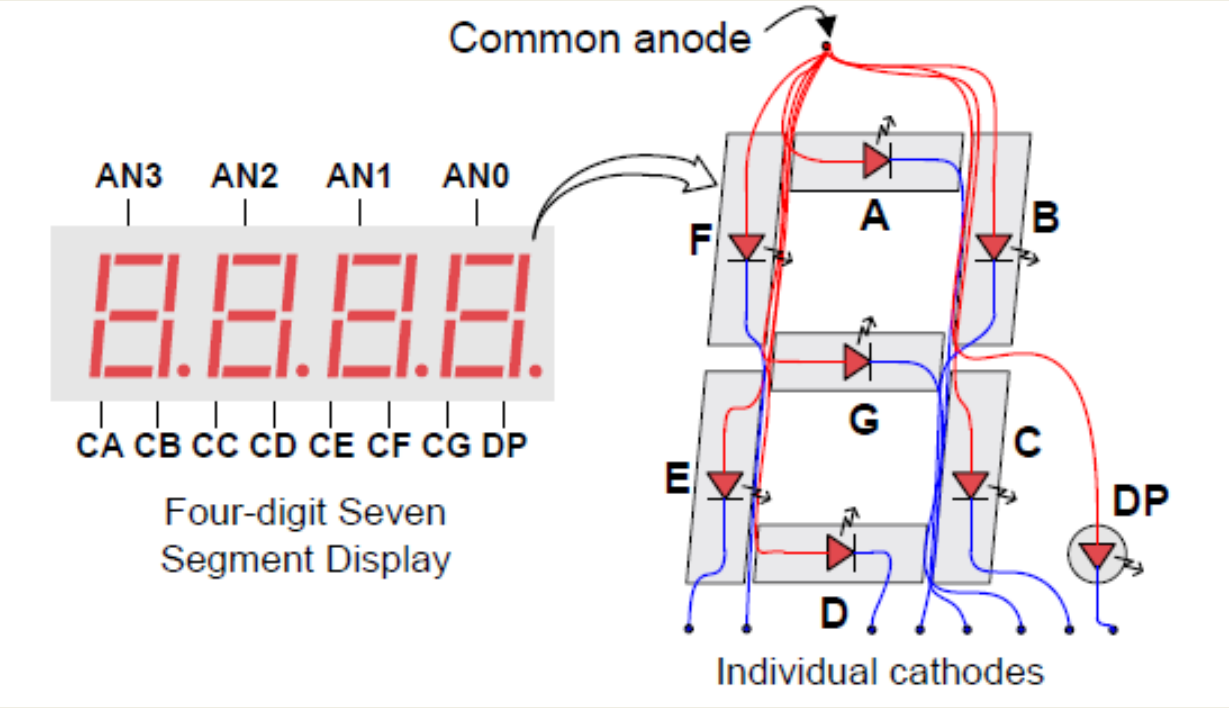
Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	6

All user specified timing constraints are met.

ONE DIGIT BCD COUNTER

SSD control signal



HLS code

```
#pragma HLS INTERFACE ap_none port=seven_segment_enable
#pragma HLS INTERFACE ap_none port=seven_segment_data
#pragma HLS INTERFACE ap_none port=pulse
#pragma HLS INTERFACE ap_ctrl_hs port=return


    static ap_uint<4> counter_state = 0;

    if (pulse == 1){
        if (counter_state == 9)
            counter_state = 0;
        else
            counter_state++;
    }

    seven_segment_data = seven_segment_code[counter_state];
    seven_segment_enable = 0b1110;
```

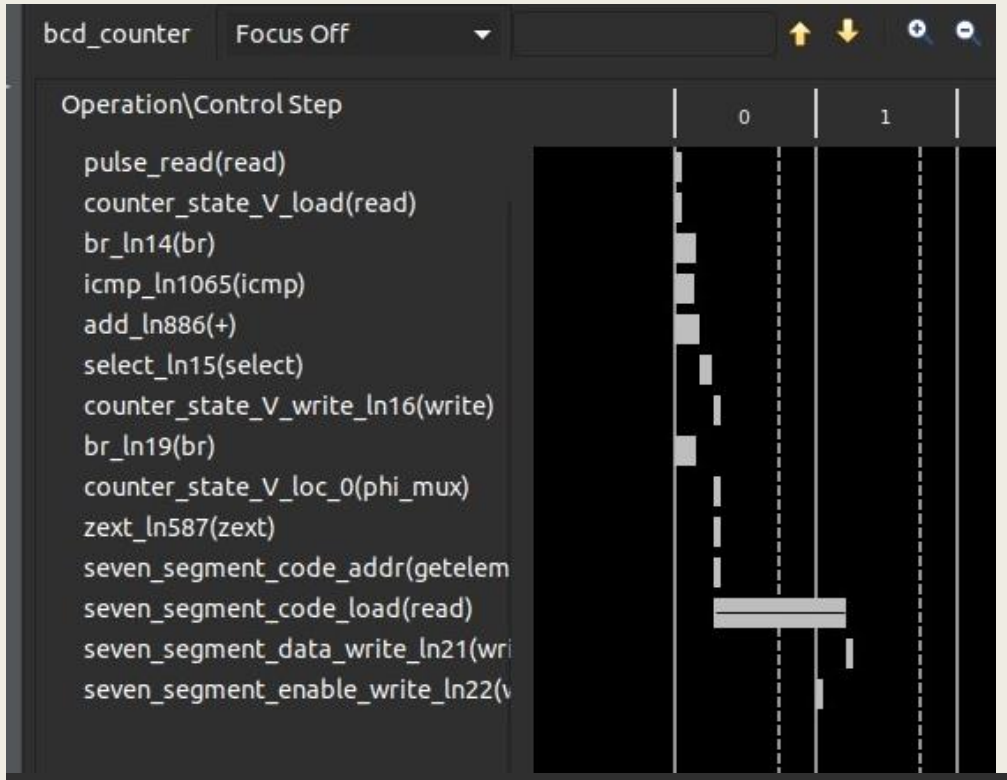
```
const unsigned int seven_segment_code[10] = {
        0b11000000,  // 0
        0b11111001,  // 1
        0b10100100,  // 2
        0b10110000,  // 3
        0b10011001,  // 4
        0b10010010,  // 5
        0b10000010,  // 6
        0b11111000,  // 7
        0b10000000,  // 8
        0b10010000,  // 9
};
```

```
bool pulse;
ap_uint<8> seven_segment_data_hw;
ap_uint<4> seven_segment_enable_hw;

ap_uint<8> seven_segment_data_sw;
ap_uint<4> seven_segment_enable_sw;


for (int i = 0; i < 12; i++) {
    bcd_counter(1, seven_segment_data_hw, seven_segment_enable_hw);
}


if (status == 0) {
    std::cout << "Test Passed!" << std::endl;
} else {
    std::cout << "Test Failed!" << std::endl;
}
```

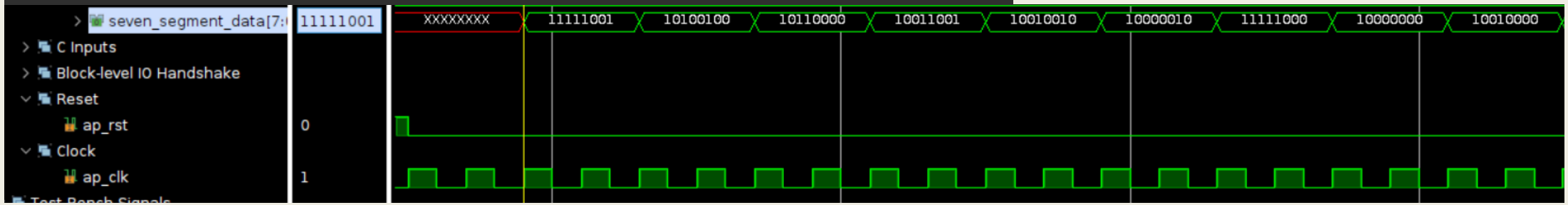
HLS testbench

```
00000010000000000000000010000000000000000010000000000000000010000000000000000010000000000000
```

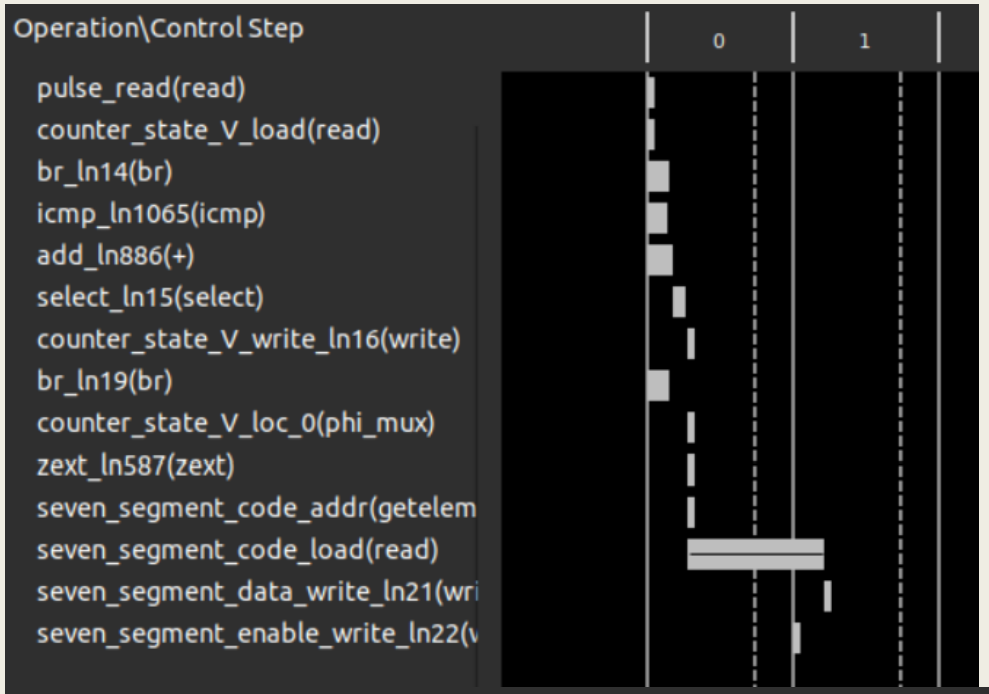
Synthesis Result & Waveform



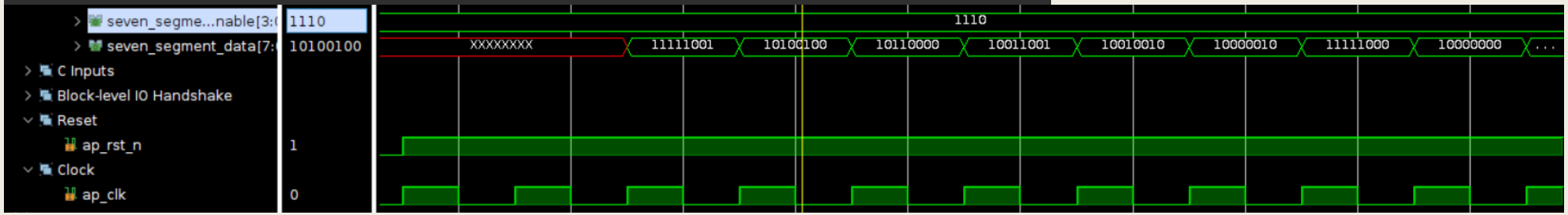
Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
bcd_counter	2	2	2	1	1	1



Pipelined



Modules & Loops	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
bcd_counter	1	1	1	1	1	1



Utilization

Resource	Utilization	Available	Utilization %
LUT	13	20800	0.06
FF	13	41600	0.03
IO	19	106	17.92

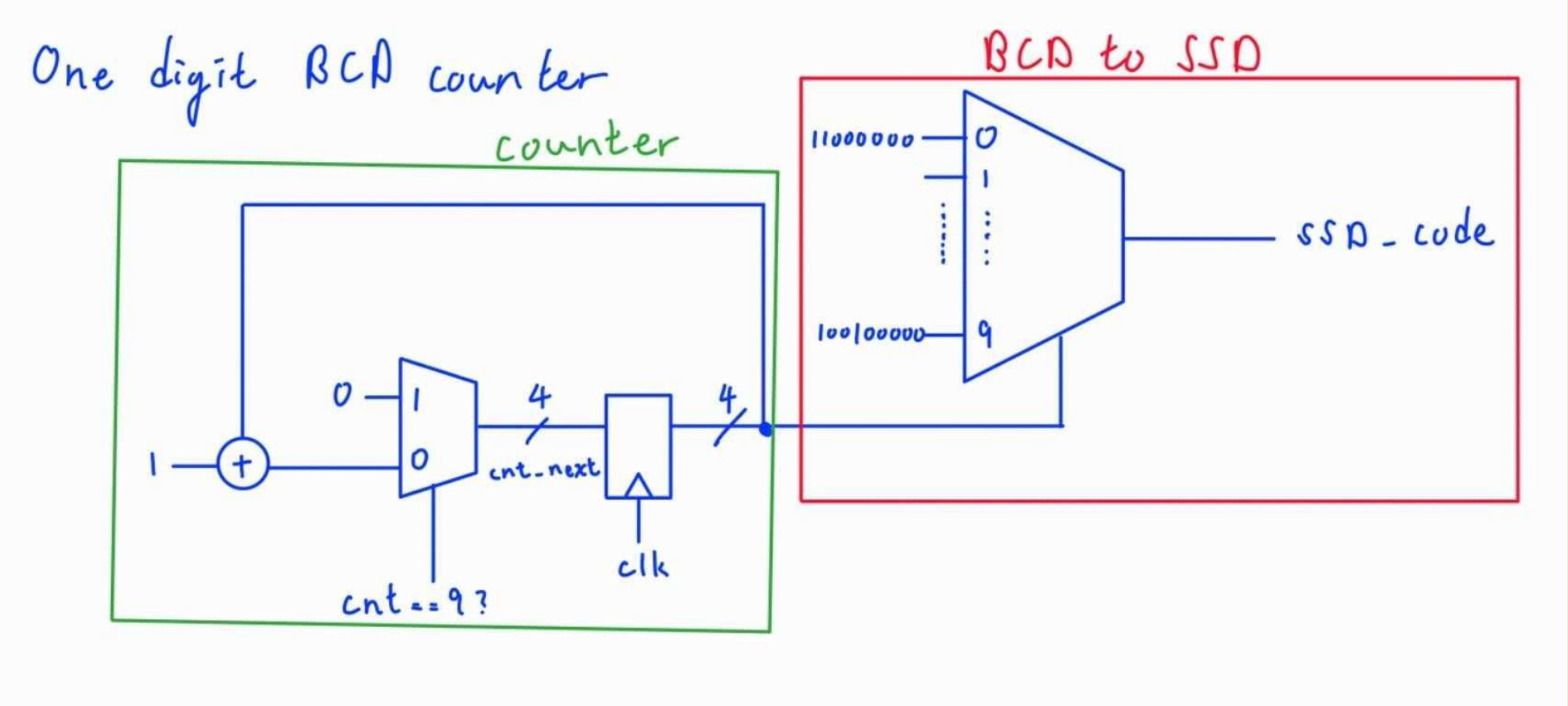
Timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6.917 ns	Worst Hold Slack (WHS):	0.255 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	27	Total Number of Endpoints:	27	Total Number of Endpoints:	13

All user specified timing constraints are met.

Verilog Design



Verilog code

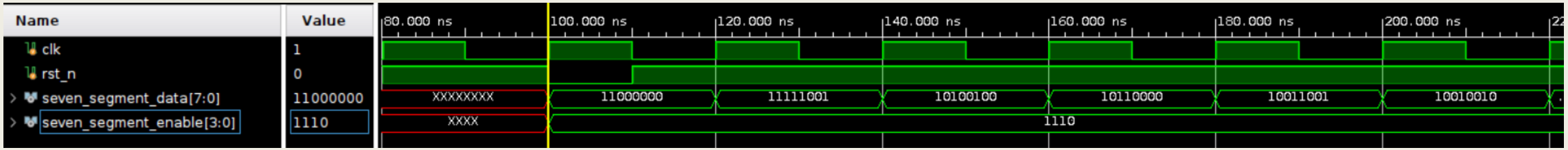
```
always@(*)
begin
    //sevensegment enabler
    seven_segment_enable = 4'b1110;
    //bcd to sevensegment decoder
    case(bcd_counter)
        4'd0: seven_segment_data = 8'b11000000;
        4'd1: seven_segment_data = 8'b11111001;
        4'd2: seven_segment_data = 8'b10100100;
        4'd3: seven_segment_data = 8'b10110000;
        4'd4: seven_segment_data = 8'b10011001;
        4'd5: seven_segment_data = 8'b10010010;
        4'd6: seven_segment_data = 8'b10000010;
        4'd7: seven_segment_data = 8'b11111000;
        4'd8: seven_segment_data = 8'b10000000;
        4'd9: seven_segment_data = 8'b10010000;
        //This shold not happen
        default: seven_segment_data = 8'b11111111;
    endcase
```

```
    //counter
    if(bcd_counter == 4'd9)
        bcd_counter_next = 4'd0;
    else
        bcd_counter_next = bcd_counter + 4'd1;
end

//sequential logic
always@(posedge pulse or negedge rst_n)
begin
    if(!rst_n)
        bcd_counter <= 4'd0;
    else
        bcd_counter <= bcd_counter_next;
end
```

Verilog testbench

```
module bcd_counter_multicycle_tb();
    reg clk;
    reg rst_n;
    wire [7:0] seven_segment_data;
    wire [3:0] seven_segment_enable;

    bcd_counter_multicycle bcd_counter_multicycle_0(.rst_n(rst_n), .pulse(clk),
    .seven_segment_data(seven_segment_data), .seven_segment_enable(seven_segment_enable));

    always #10 clk = ~clk;
    initial
    begin
        clk       = 1;
        rst_n     = 1;
        #100 rst_n = 0;
        #10 rst_n = 1;
        #1000 $finish;
    end
endmodule
```

Simulation result



Timing

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.613 ns	Worst Hold Slack (WHS):	0.261 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	4	Total Number of Endpoints:	4	Total Number of Endpoints:	5

All user specified timing constraints are met.

Synthesis result

Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
FF	4	41600	0.01
IO	14	106	13.21
BUFG	1	32	3.13

Comparison

Utilization

HLS

Resource	Utilization	Available	Utilization %
LUT	13	20800	0.06
FF	13	41600	0.03
IO	19	106	17.92

verilog

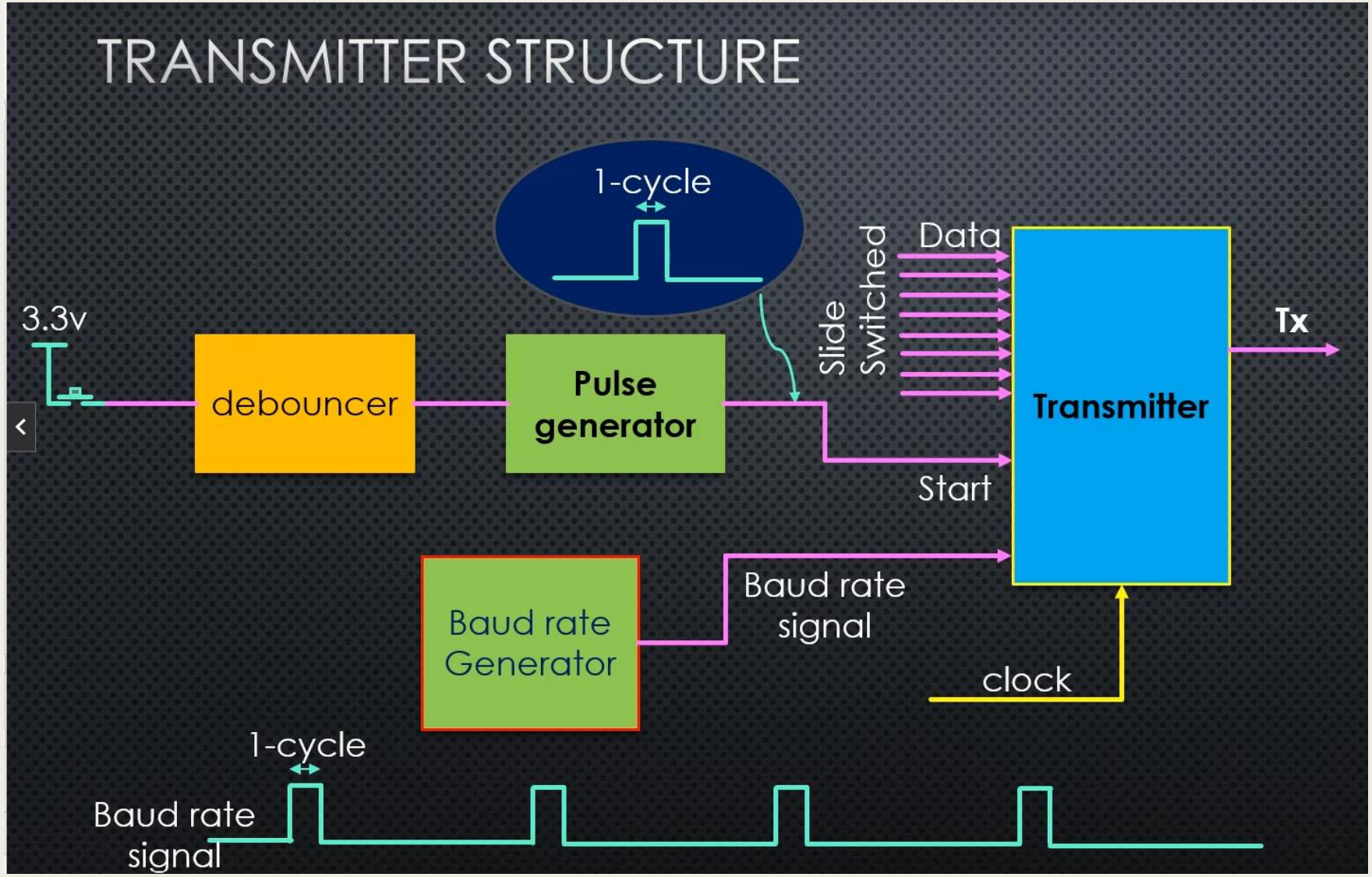
Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
FF	4	41600	0.01
IO	14	106	13.21
BUFG	1	32	3.13

Timing	
HLS (pipelined)	**Design Timing Summary** **Setup** Worst Negative Slack (WNS): 6.917 ns Total Negative Slack (TNS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 27 **Hold** Worst Hold Slack (WHS): 0.255 ns Total Hold Slack (THS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 27 **Pulse Width** Worst Pulse Width Slack (WPWS): 4.500 ns Total Pulse Width Negative Slack (TPWS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 13 **All user specified timing constraints are met.**
verilog	**Setup** Worst Negative Slack (WNS): 8.613 ns Total Negative Slack (TNS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 4 **Hold** Worst Hold Slack (WHS): 0.261 ns Total Hold Slack (THS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 4 **Pulse Width** Worst Pulse Width Slack (WPWS): 4.500 ns Total Pulse Width Negative Slack (TPWS): 0.000 ns Number of Failing Endpoints: 0 Total Number of Endpoints: 5 **All user specified timing constraints are met.**

UART

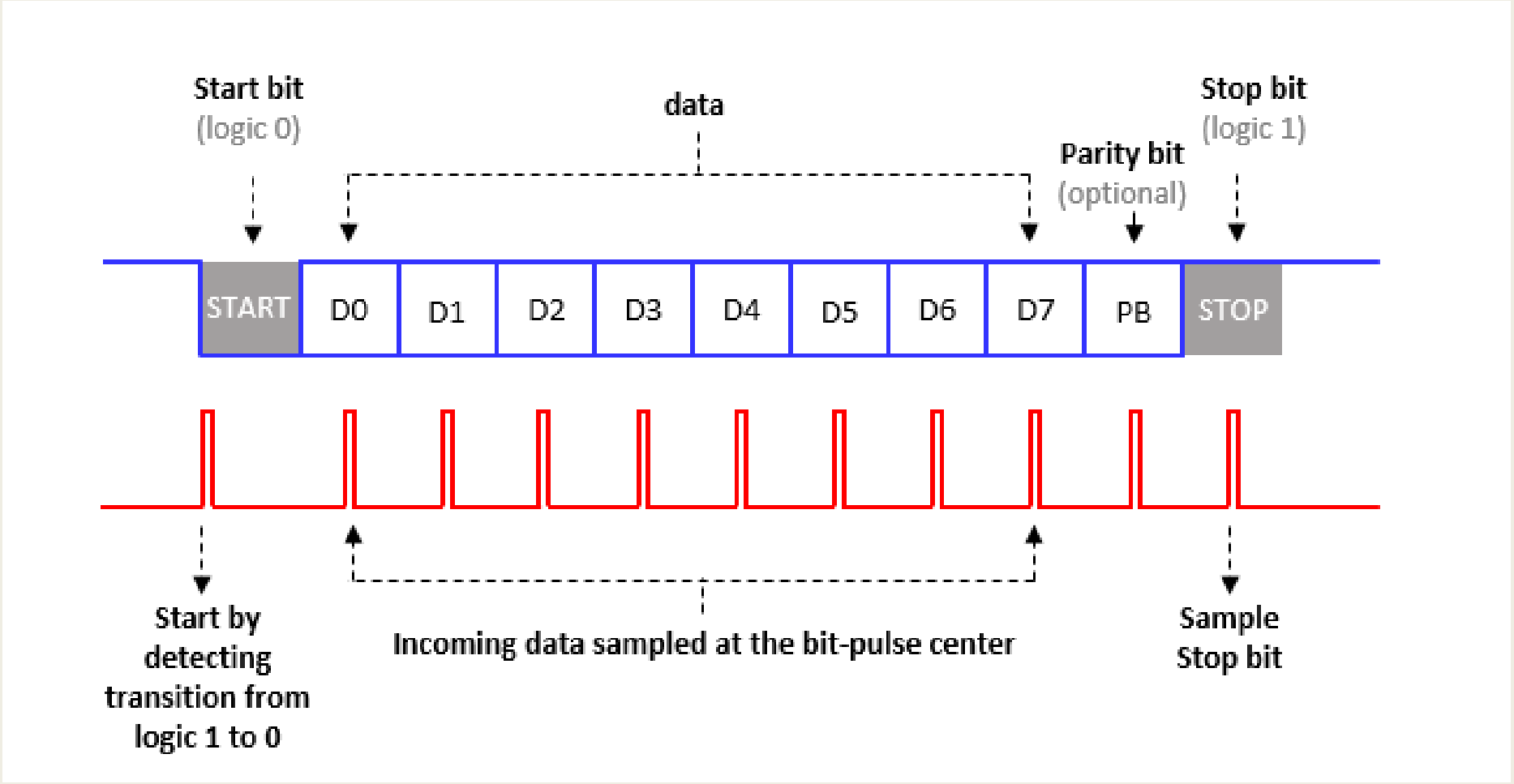
Universal Asynchronous Receiver Transmitter

UART TRANSMITTER

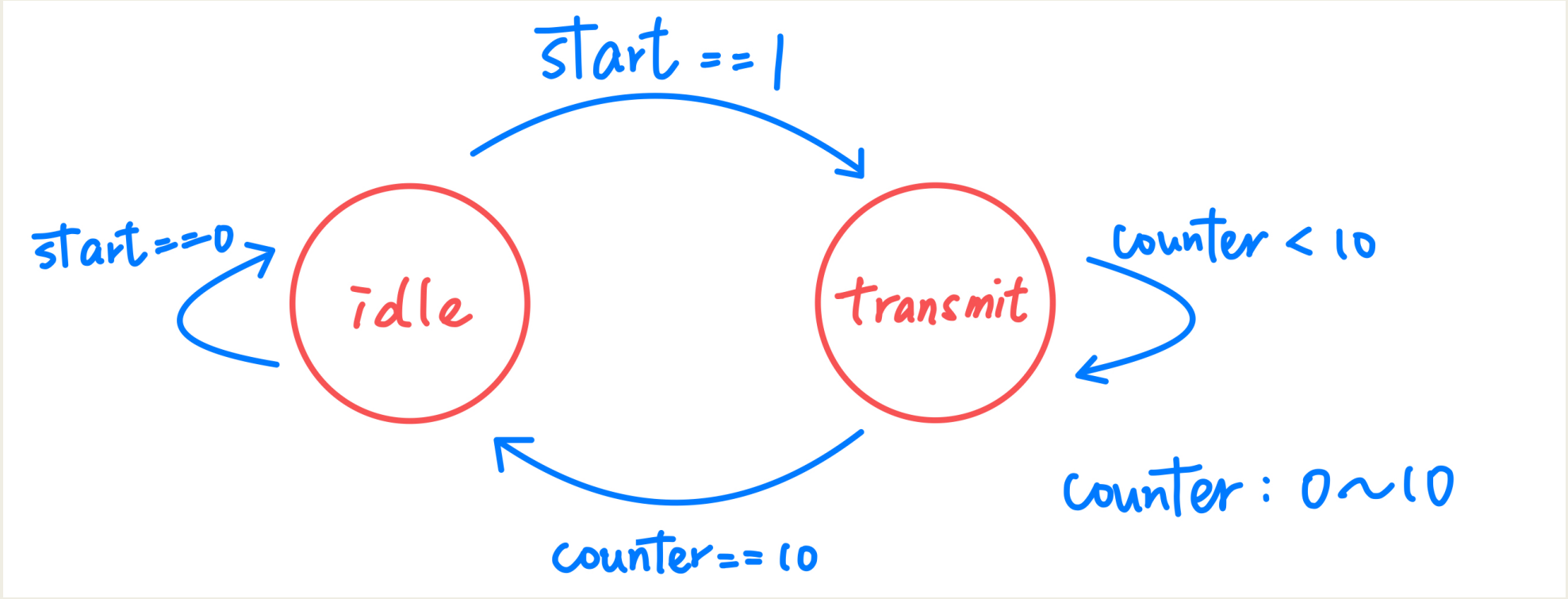

TRANSMITTER STRUCTURE
1-cycle
3.3v
debouncer
Pulse generator
Slide Switched
Data
Transmitter
Tx
Start
Baud rate Generator
Baud rate signal
clock
1-cycle
Baud rate signal

Baud Rate Generator

- In both transmitter and receiver
- The rate should be the same in transmitter and receiver
- In the project, we take 9600(1/s) as example
- Coding is implemented by the single cycle regular pulse generator
- The number of counter is 10416 = (1 / 9600s) / 10ns
- [10ns is the clock period of FPGA]


Start bit
(logic 0)
data
Parity bit
(optional)
Stop bit
(logic 1)
START
D0
D1
D2
D3
D4
D5
D6
D7
PB
STOP
Start by detecting transition from logic 1 to 0
Incoming data sampled at the bit-pulse center
Sample Stop bit

Design


start == 1
start == 0
idle
transmit
counter < 10
counter == 10
counter : 0~10

Verilog code

```
`define idle 1'b0
`define transmit 1'b1

module uart_transmitter(
    input clk,
    input rst_n,
    input baud_rate_signal,
    input [7:0]data,
    input start,
    output reg uart_tx
);

    reg state;
    reg next_state;
    reg [3:0]bit_counter;
    reg [3:0]next_bit_counter;
    reg uart_tx_local;
    reg [9:0]d ;
```

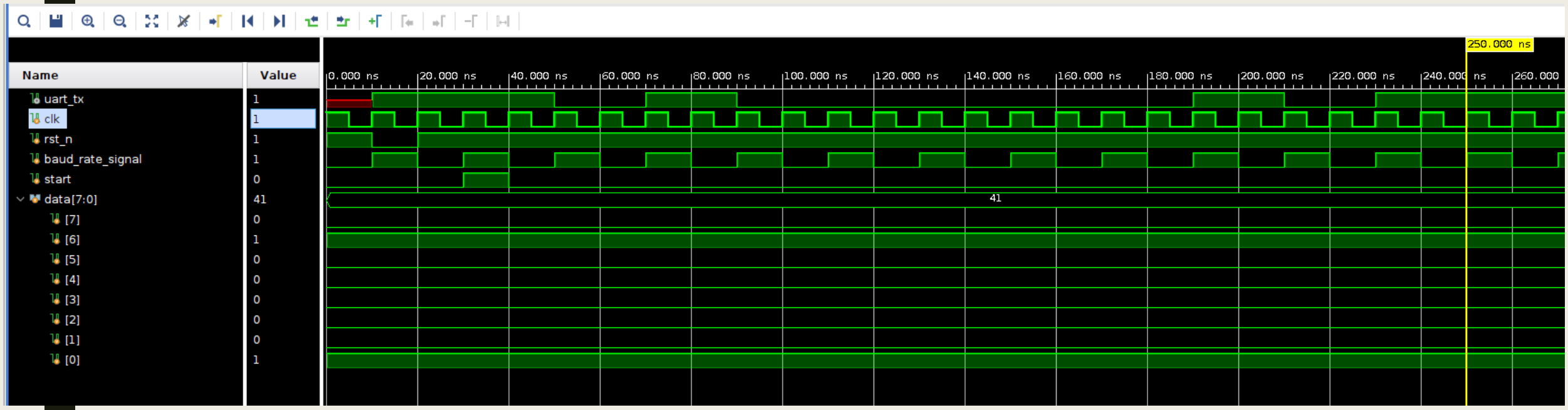
```
    always@(posedge clk or negedge rst_n)
        if(~rst_n) begin
            state <= `idle;
            bit_counter <= 0;
            uart_tx <= 1;
        end
        else begin
            state <= next_state;
            bit_counter <= next_bit_counter;
            uart_tx <= uart_tx_local;
        end

endmodule
```

```
    always@* begin
        d = {1'b1, data, 1'b0};

        case(state)
            `idle:
                if(start == 1'b1) begin
                    next_state = `transmit;
                    uart_tx_local = 1'b1;
                    next_bit_counter = 0;
                end
                else begin
                    next_state = `idle;
                    uart_tx_local = 1'b1;
                    next_bit_counter = 0;
                end
            `transmit:
                if(baud_rate_signal == 1'b1) begin
                    if(bit_counter == 4'd10) begin
                        next_state = `idle;
                        uart_tx_local = 1'b1;
                        next_bit_counter = 0;
                    end
                    else begin
                        next_state = `transmit;
                        uart_tx_local = d[bit_counter];
                        next_bit_counter = bit_counter + 1;
                    end
                end
                else begin
                    if(bit_counter == 1'b0)
                        uart_tx_local = 1;
                    else
                        uart_tx_local = d[bit_counter - 1 ];
                    next_state = `transmit;
                    next_bit_counter = bit_counter;
                end
            default: begin
                next_state = `idle;
                next_bit_counter = 1'b0;
            end
            endcase
    end
```

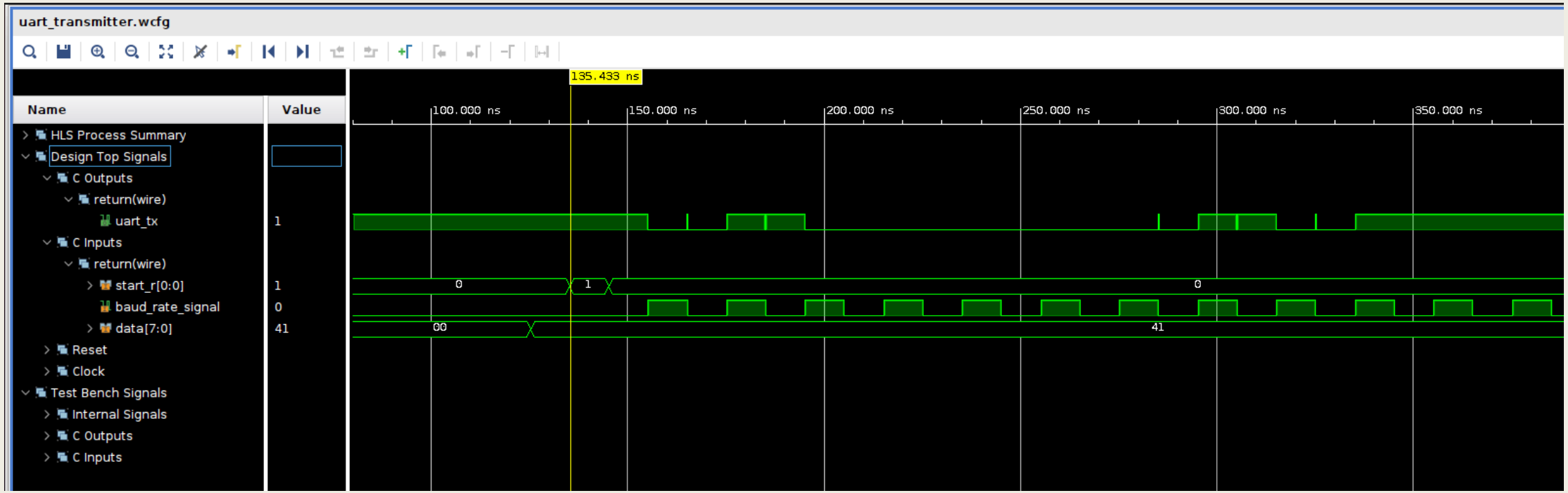
Verilog testbench

```
module uart_transmitter_tb(

    );

    wire uart_tx;
    reg clk, rst_n, baud_rate_signal, start;
    reg [7:0]data;

    uart_transmitter U(.clk(clk), .uart_tx(uart_tx), .data(data),
                       .baud_rate_signal(baud_rate_signal), .start(start));

    initial begin
        clk = 1;
        rst_n = 1;
        data = 8'b01000001;
        start = 0;
        baud_rate_signal = 0;
        #10 rst_n = 0;
        #10 rst_n = 1;
        #10 start = 1;
        #10 start = 0;
    end

    always #5 clk <= ~clk;
    always #10 baud_rate_signal <= ~baud_rate_signal;

endmodule
```

Simulation waveform(verilog)



Simulation waveform(HLS)



Utilization

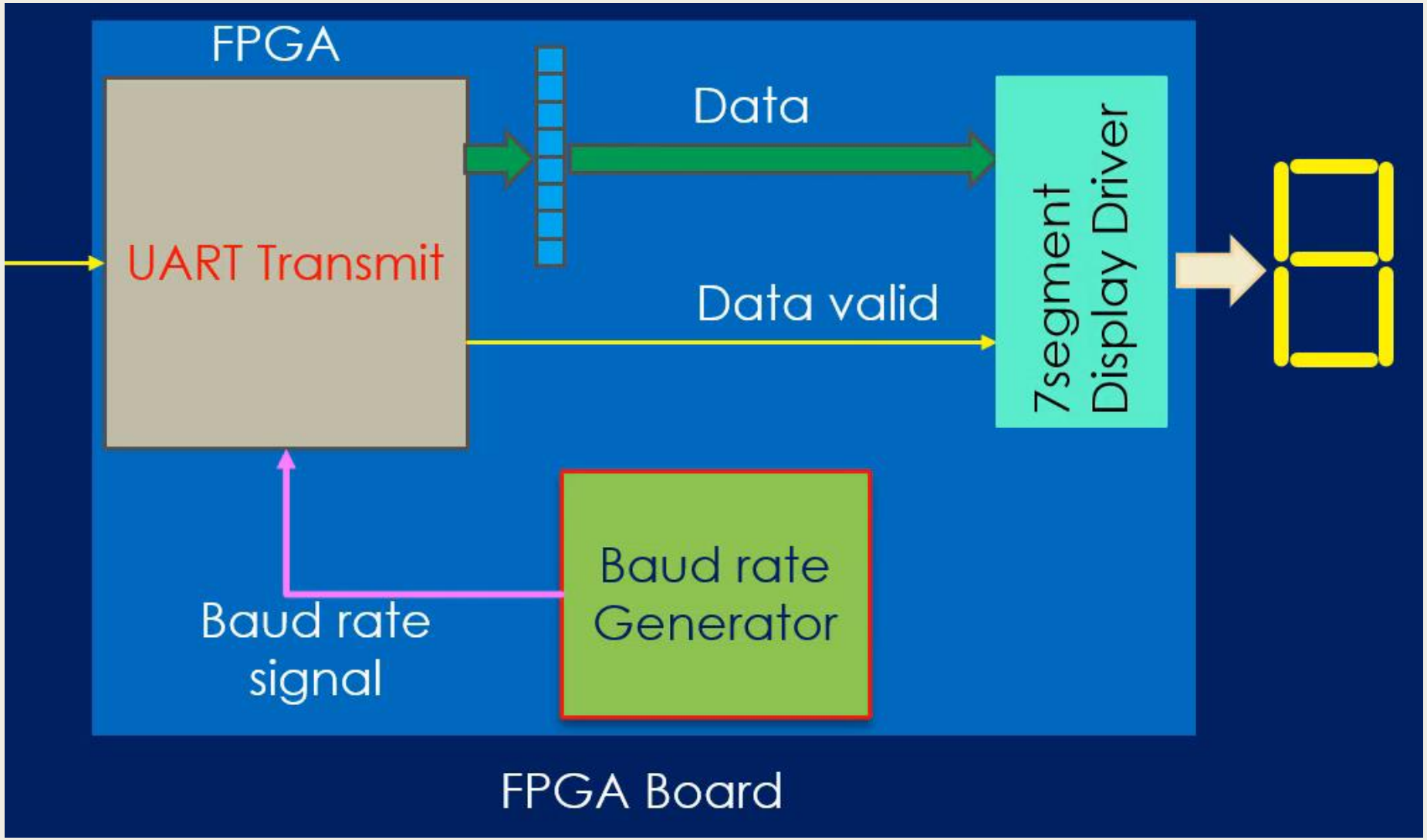
Resource	Utilization	Available	Utilization %
LUT	22	20800	0.11
FF	33	41600	0.08
IO	13	106	12.26

HLS

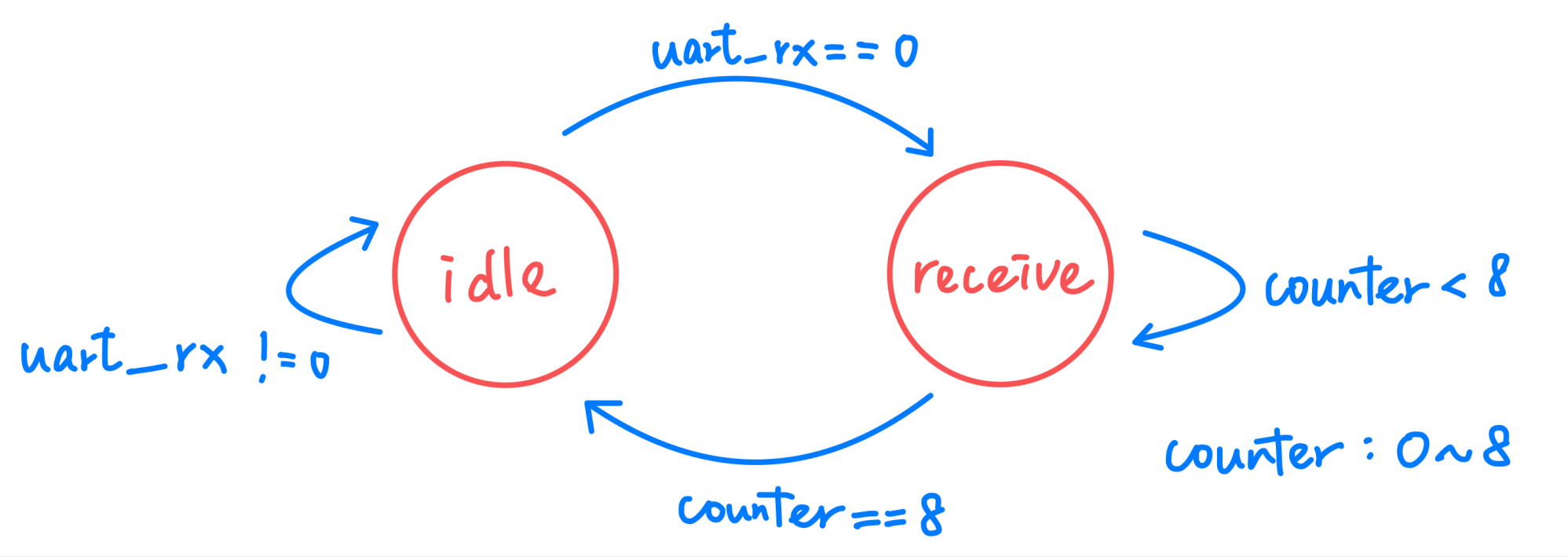
Resource	Estimation	Available	Utilization %
LUT	12	20800	0.06
FF	6	41600	0.01
IO	13	106	12.26
BUFG	1	32	3.13

verilog

UART RECEIVER


FPGA
UART Transmit
Data
Data valid
7segment
Display Driver
Baud rate
signal
Baud rate
Generator
FPGA Board

Design


uart_rx == 0
idle
receive
uart_rx != 0
counter < 8
counter : 0~8
counter == 8

Verilog code

```
`define idle 1'b0
`define receive 1'b1

module uart_receiver(
    input clk,
    input rst_n,
    input uart_rx,
    input baud_rate_signal,
    output reg [7:0]data,
    output reg valid_data
    );

    reg state, next_state;
    reg [3:0]bit_counter;
    reg [3:0]next_bit_counter;
    reg valid_data_local;
    reg [7:0]d;
```

```
    always@(posedge clk or negedge rst_n)
        if(~rst_n) begin
            state <= `idle;
            bit_counter <= 0;
            valid_data <= 0;
            data <= 0;
        end
        else begin
            state <= next_state;
            bit_counter <= next_bit_counter;
            valid_data <= valid_data_local;
            data <= d;
        end

endmodule
```

Verilog code(con')

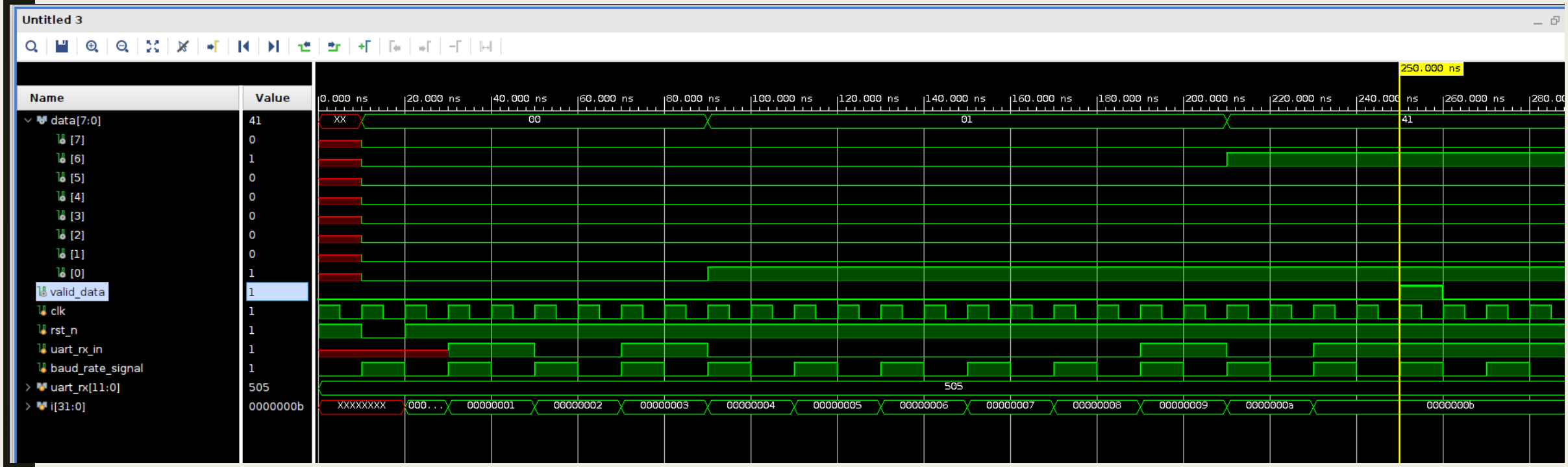
```
always@* begin
    case(state)
        `idle: begin
            if(baud_rate_signal == 1) begin
                if(uart_rx == 0)
                    next_state = `receive;
                else
                    next_state = `idle;
            end
            else begin
                next_state = `idle;
            end
            valid_data_local = 0;
            next_bit_counter = 0;
            d = data;
        end
```

```
        `receive: begin
            if(baud_rate_signal == 1) begin
                if(bit_counter == 8 && uart_rx == 1) begin
                    valid_data_local = 1;
                    next_state = `idle;
                    next_bit_counter = 0;
                    d = data;
                end
                else if(bit_counter == 8 && uart_rx == 0) begin
                    valid_data_local = 0;
                    next_state = `idle;
                    next_bit_counter = 0;
                    d = data;
                end
                else begin
                    valid_data_local = 0;
                    next_state = `receive;
                    d[bit_counter] = uart_rx;
                    next_bit_counter = bit_counter + 1;
                end
            end
            else begin
                valid_data_local = 0;
                next_state = `receive;
                next_bit_counter = bit_counter;
                d = data;
            end
        end
        default: begin
            valid_data_local = 0;
            next_bit_counter = 0;
            d[bit_counter] = 0;
            next_state = `idle;
        end
    endcase
    end
```

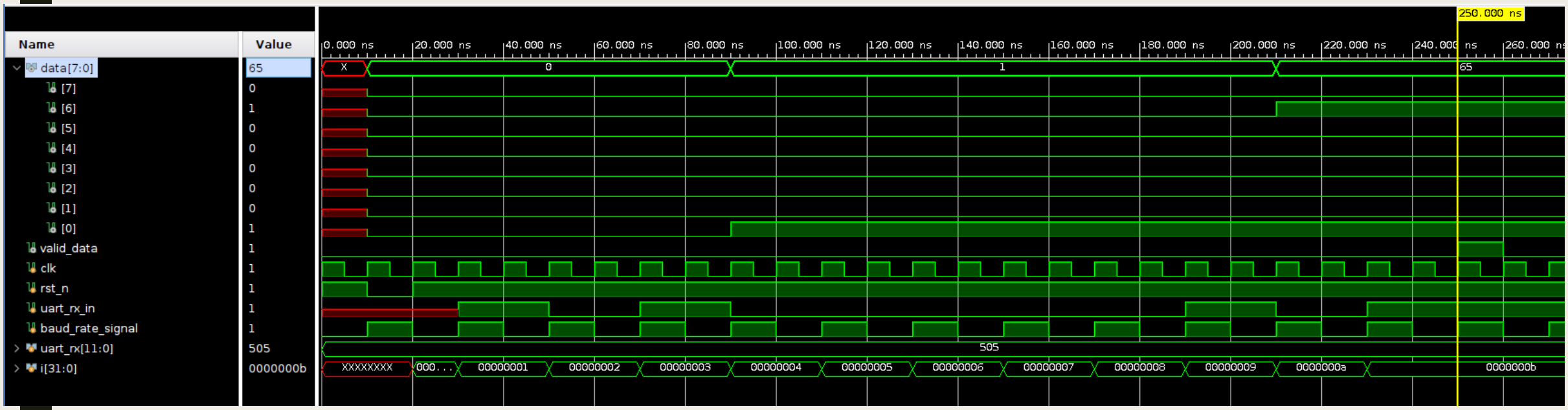
Verilog testbench

```
module uart_receiver_tb(

    );

    wire [7:0]data;
    wire valid_data;
    reg clk, rst_n, uart_rx_in, baud_rate_signal;
    reg [11:0]uart_rx = 11'b10100000101;
    integer i;

    uart_receiver U(.clk(clk), .rst_n(rst_n), .uart_rx(uart_rx_in),
                    .baud_rate_signal(baud_rate_signal), .data(data), .valid_data(valid_data));

    initial begin
    clk = 1; rst_n = 1; baud_rate_signal = 0;
    #10 rst_n = 0;
    #10 rst_n = 1;

    for(i = 0; i < 11; i = i + 1) begin
        @(posedge baud_rate_signal)
            uart_rx_in <= uart_rx[i];
    end

    end

    always #5 clk <= ~clk;
    always #10 baud_rate_signal <= ~baud_rate_signal;
endmodule
```

Simulation waveform(verilog)



Simulation waveform(HLS)



Utilization

Resource	Utilization	Available	Utilization %
LUT	35	20800	0.17
FF	41	41600	0.10
IO	13	106	12.26

HLS

Resource	Estimation	Available	Utilization %
LUT	11	20800	0.05
FF	14	41600	0.03
IO	13	106	12.26
BUFG	1	32	3.13

verilog