SOC Lab Presentation

COMBINATIONAL LOGIC

Speaker

Group 3

Table Of Content

01. Parity Generator

```cpp
#include "parity_generator.h"



bool parity_generator(ap_uint<W> a) {
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_ctrl_none port=return

    bool parity = 0;


    for (int i = 0; i < W; i++) {
#pragma HLS UNROLL
        parity = parity ^ a[i];
    }

    return parity;
}

```

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


bool parity_generator_sw(ap_uint<W> a) {
    bool parity = 0;

    for (int i = 0; i < W; i++) {
        parity = parity ^ a[i];
    }

    return parity;
}

int main() {
    int status = 0;

    ap_uint<W> x;

    bool parity_hw;
    bool parity_sw;

    for (int i = 0; i < 32; i++) {
        x = (ap_uint<W>)i;
        parity_hw = parity_generator(x);
        parity_sw = parity_generator_sw(x);

        if (parity_sw != parity_hw)
        {
            status = -1;  // bitset = dec 2 unsigned W-bit binary number
            std::cout << "x = " << std::bitset<W>(x) << "  parity_hw = " << parity_hw << "  parity_sw = " << parity_sw <<std::endl;
            break;
        }
        std::cout << "x = " << std::bitset<W>(x) << "  parity_hw = " << parity_hw << "  parity_sw = " << parity_sw <<std::endl;
    }

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


    return status;
}
```

Top Function(Kernel)

```cpp
#include "parity_generator.h"



bool parity_generator(ap_uint<W> a) {
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_ctrl_none port=return

    bool parity = 0;


    for (int i = 0; i < W; i++) {
#pragma HLS UNROLL
        parity = parity ^ a[i];
    }



    return parity;
}
```

pure wire

Can't use when design is not "purely combinational"

data parallelism

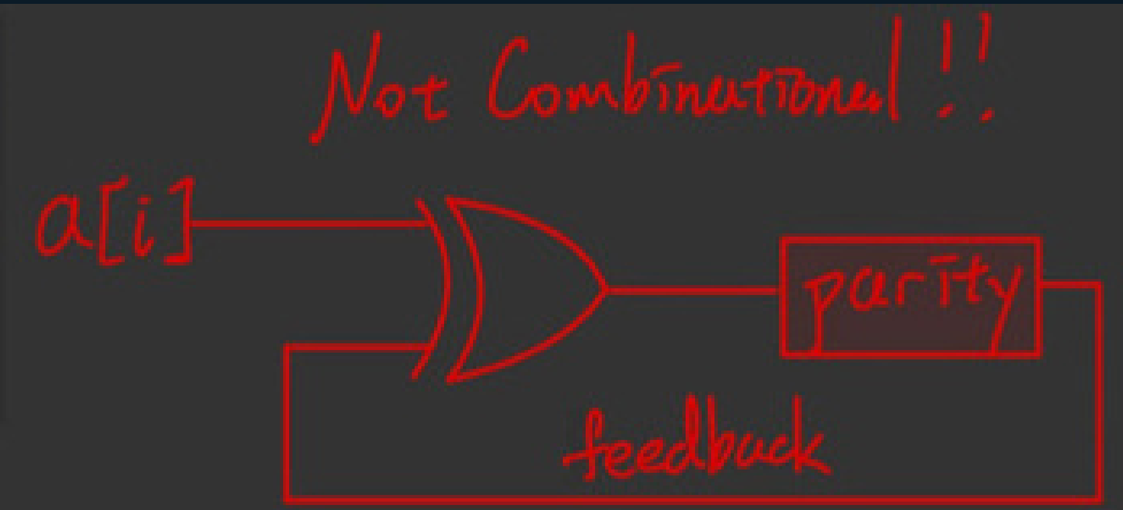
Block-level protocol:
ap_ctrl_none?
ap_ctrl_hs?
ap_ctrl_chain?

Combinational?
Sequential?

Without 'UNROLL'

Use only one hardware.
Needs a memory!

```
for (int i = 0; i < W; i++) {

    parity = parity ^ a[i];
}
```



A circuit with feedback incorporates some form of memory or state storage,
which allows it to consider past inputs or outputs when generating the current output.
These circuits are known as 'sequential' logic circuits.

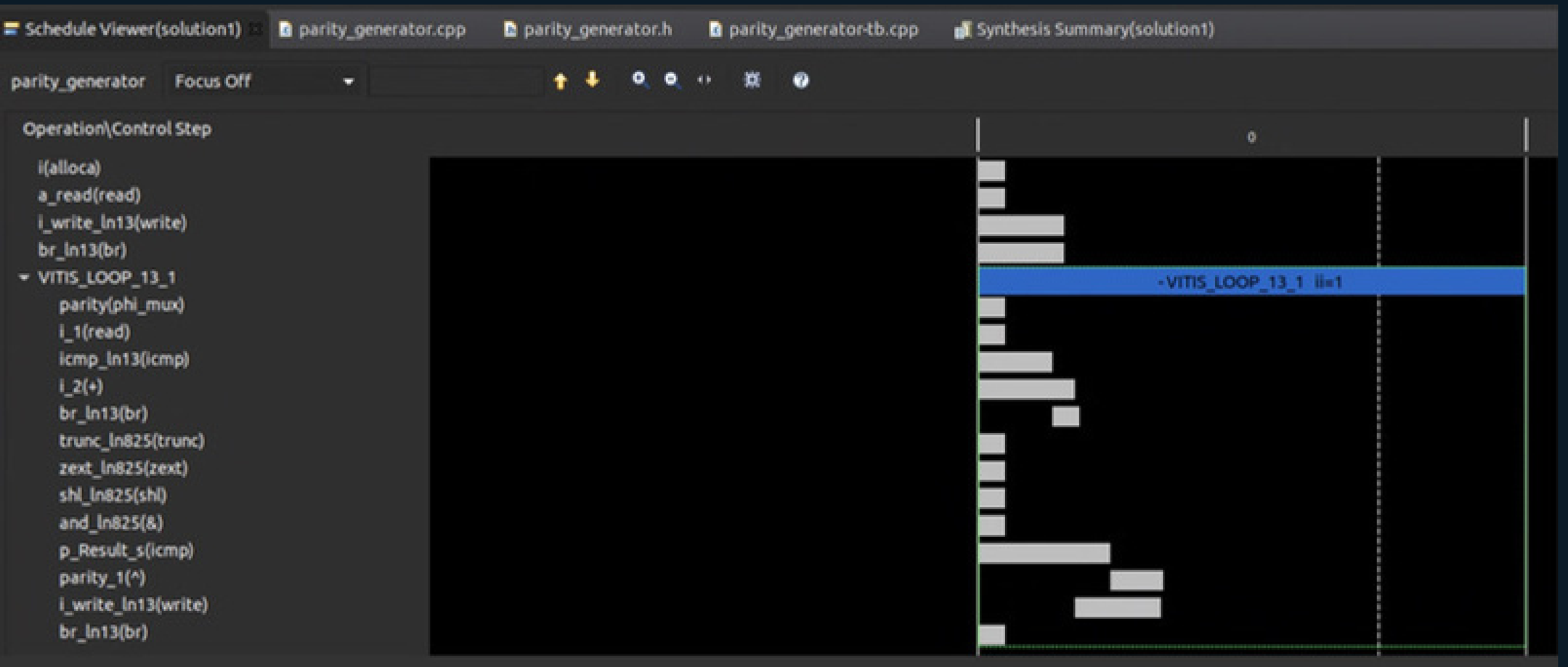
Without 'UNROLL'

Performance & Resource Estimates

Modules Loops

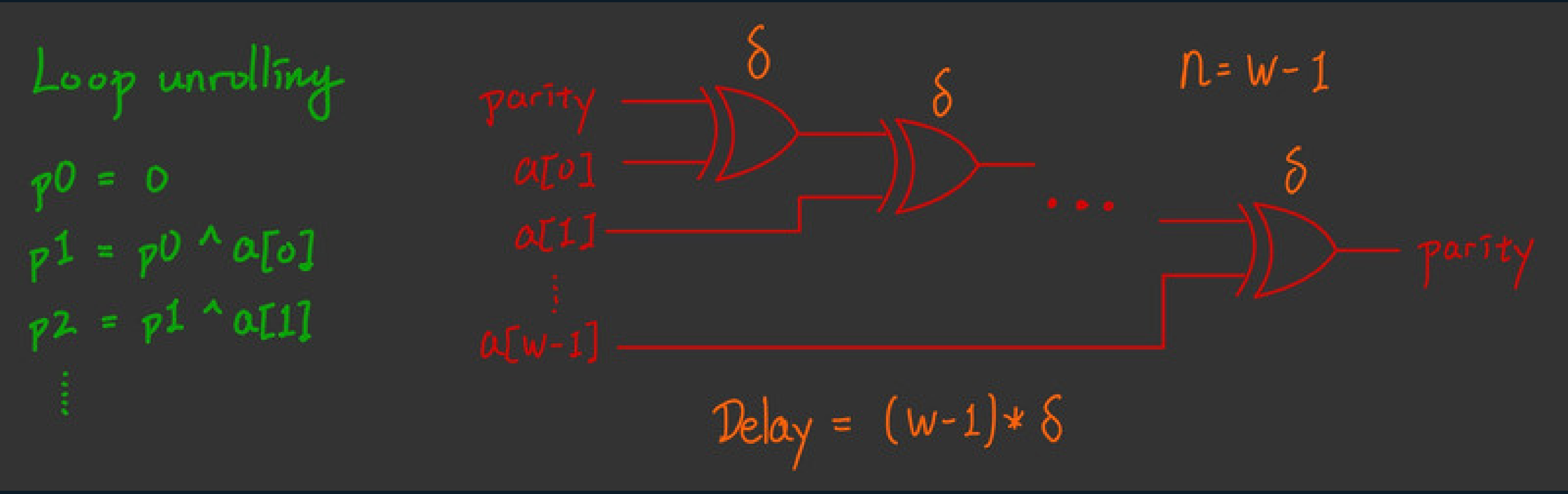
Sequential

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
parity_generator				-	18	180.000	-	19	-	no	0	0	8	125	0
VITIS_LOOP_13_1				-	16	160.000	1	1	16	yes	-	-	-	-	-



Loop Unrolling

Unroll the loop manually:



Unroll loops can create multiple independent operations rather than a single collection of operations.

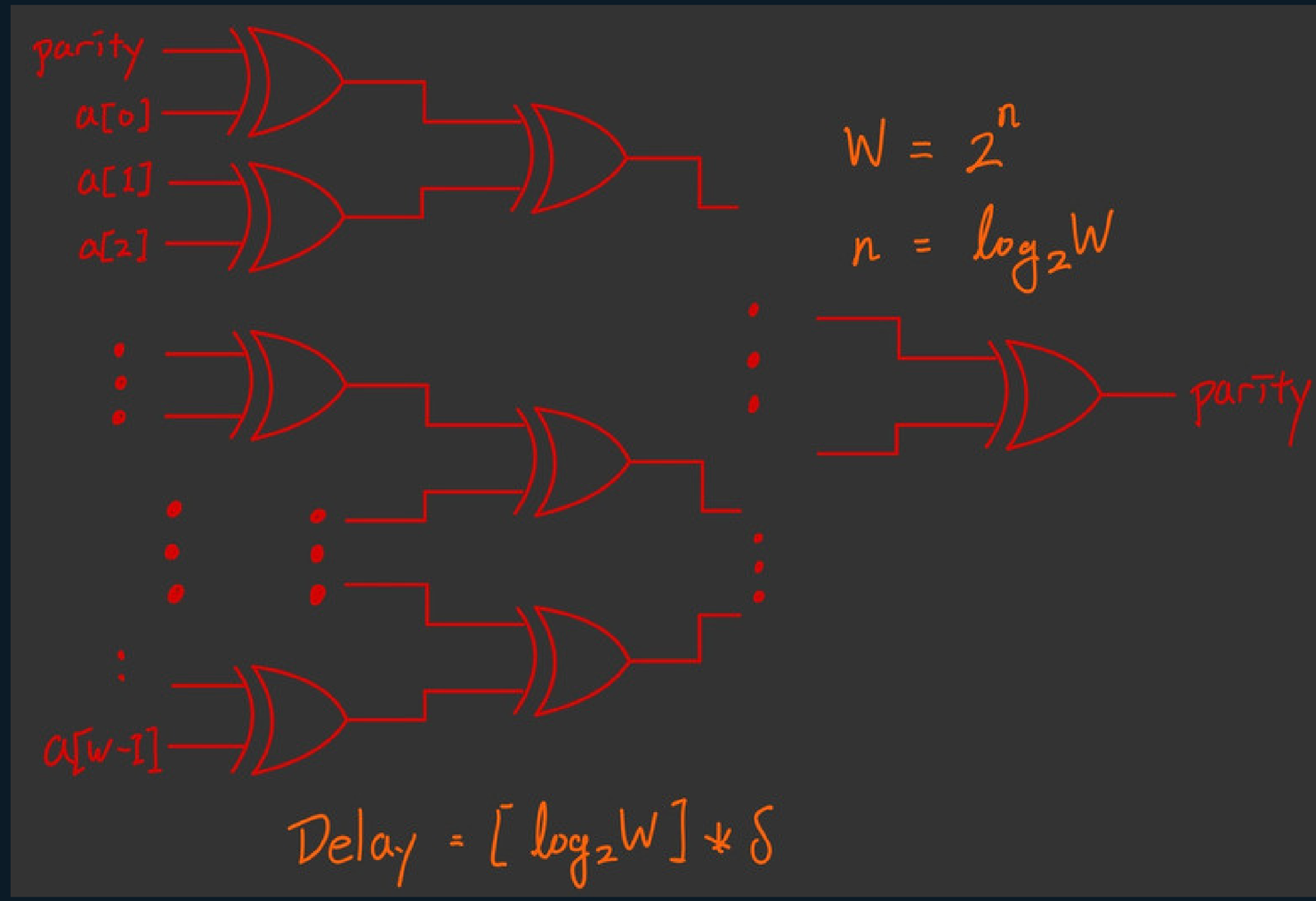
#pragma HLS UNROLL

```
    for (int i = 0; i < W; i++) {
#pragma HLS UNROLL
        parity = parity ^ a[i];
    }
```

Transforms loops by creating multiple copies of the loop body in RTL design, which allows some or all loop iterations to occur in parallel.

Unroll the loop automatically:

Balanced Tree



No flip-flop, only LUT (Combinational)

Performance & Resource Estimates

Only LookUp Table

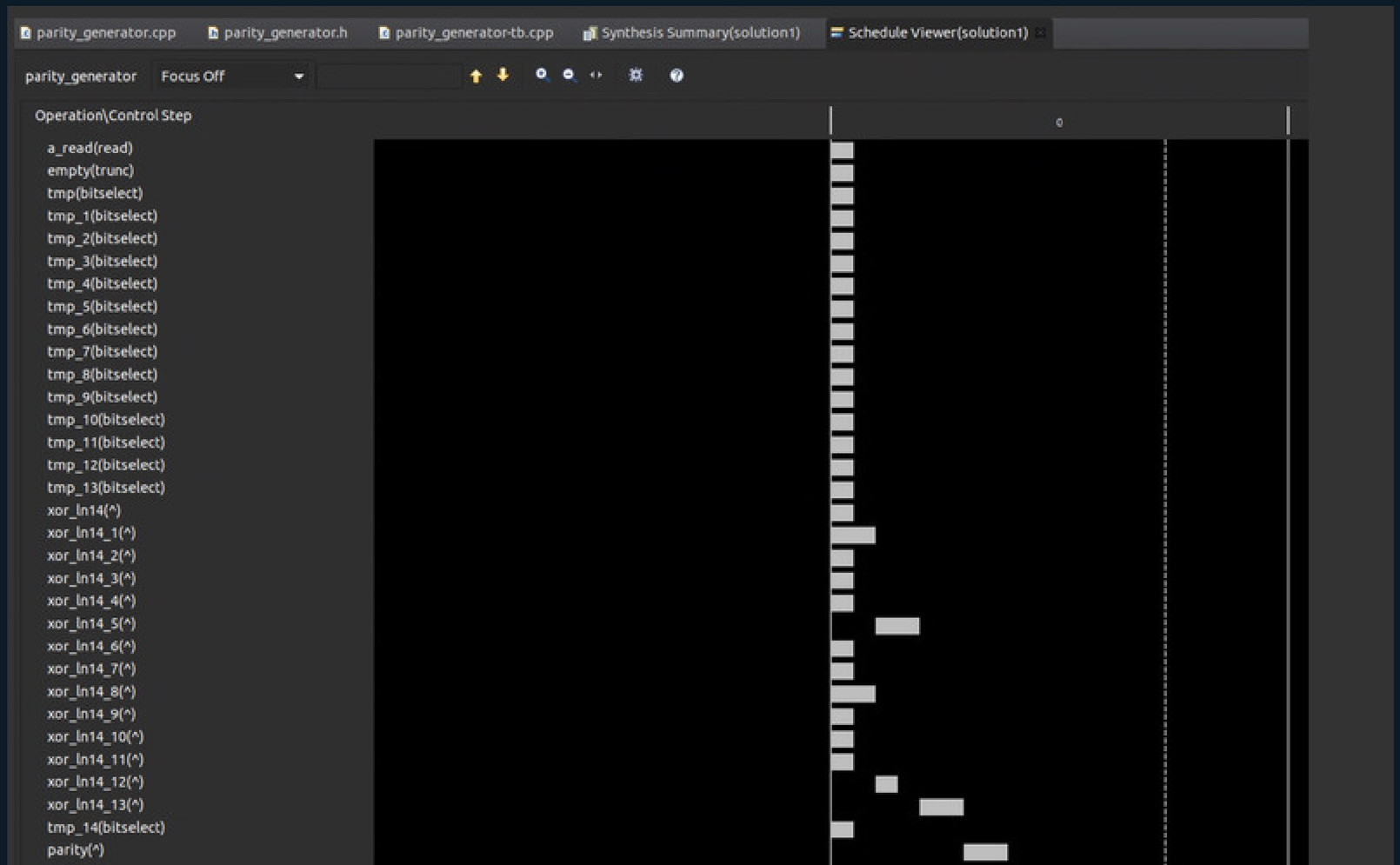
Modules ✓ Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
parity_generator				-	0	0.0	-	1	-	no	0	0	0	30	0

parity_generator.cpp | parity_generator.h | parity_generator-tb.cpp | Synthesis Summary(solution1) | Schedule Viewer(solution1)

parity_generator | Focus Off

Operation\Control Step



Summary

Name	BRAM_18K	DSP	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-		30	-
FIFO	-	-	-		-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	0	0	0	30	0
Available	280	220	106400	53200	0
Utilization (%)	0	0	0	~0	0

Parity Generator in Testbench

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

bool parity_generator_sw(ap_uint<W> a) {
    bool parity = 0;

    for (int i = 0; i < W; i++) {
        parity = parity ^ a[i];
    }

    return parity;
}
```

This means 'a' is a W-bit unsigned integer

Main Function in Testbench

```cpp
int main() {
    int status = 0;

    ap_uint<W> x;

    bool parity_hw;
    bool parity_sw;

    for (int i = 0; i < 65536; i++) {
        x = (ap_uint<W>)i;
        parity_hw = parity_generator(x);
        parity_sw = parity_generator_sw(x);

        if (parity_sw != parity_hw)
        {
            status = -1;
            std::cout << "x = " << std::bitset<W>(x) << "  parity_hw = " << parity_hw << "  parity_sw = " << parity_sw <<std::endl;
            break;
        }
        std::cout << "x = " << std::bitset<W>(x) << "  parity_hw = " << parity_hw << "  parity_sw = " << parity_sw <<std::endl;
    }

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


    return status;
}
```

W=16

From Kernel

From tb itself

W-bit binary number

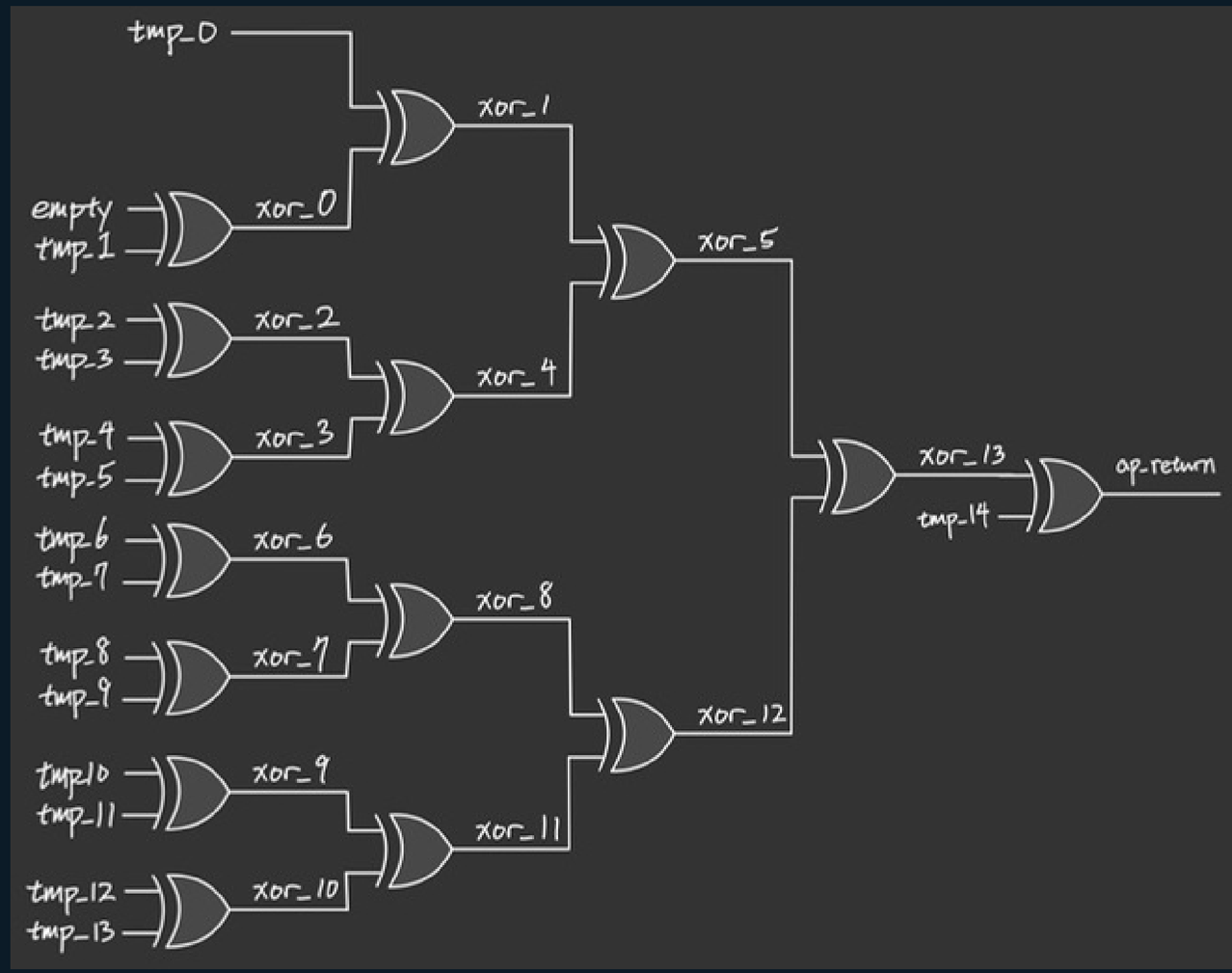
Result:

Generate even parity

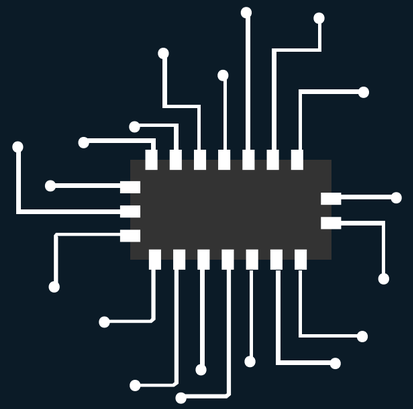
```
x = 0000000000000000  parity_hw = 0  parity_sw = 0
x = 0000000000000001  parity_hw = 1  parity_sw = 1
x = 0000000000000010  parity_hw = 1  parity_sw = 1
x = 0000000000000011  parity_hw = 0  parity_sw = 0
x = 0000000000000100  parity_hw = 1  parity_sw = 1
x = 0000000000000101  parity_hw = 0  parity_sw = 0
x = 0000000000000110  parity_hw = 0  parity_sw = 0
x = 0000000000000111  parity_hw = 1  parity_sw = 1
x = 0000000000001000  parity_hw = 1  parity_sw = 1
x = 0000000000001001  parity_hw = 0  parity_sw = 0
x = 0000000000001010  parity_hw = 0  parity_sw = 0
x = 0000000000001100  parity_hw = 0  parity_sw = 0
x = 0000000000001101  parity_hw = 1  parity_sw = 1
x = 0000000000001110  parity_hw = 1  parity_sw = 1
x = 0000000000001111  parity_hw = 0  parity_sw = 0
x = 0000000000010000  parity_hw = 1  parity_sw = 1
x = 0000000000010001  parity_hw = 0  parity_sw = 0
x = 0000000000010010  parity_hw = 0  parity_sw = 0
x = 0000000000010011  parity_hw = 1  parity_sw = 1
x = 0000000000010100  parity_hw = 0  parity_sw = 0
x = 0000000000010101  parity_hw = 1  parity_sw = 1
x = 0000000000010110  parity_hw = 1  parity_sw = 1
x = 0000000000010111  parity_hw = 0  parity_sw = 0
x = 0000000000011000  parity_hw = 0  parity_sw = 0
x = 0000000000011001  parity_hw = 1  parity_sw = 1
x = 0000000000011010  parity_hw = 1  parity_sw = 1
x = 0000000000011011  parity_hw = 0  parity_sw = 0
x = 0000000000011100  parity_hw = 1  parity_sw = 1
x = 0000000000011101  parity_hw = 0  parity_sw = 0
x = 0000000000011110  parity_hw = 0  parity_sw = 0
x = 0000000000011111  parity_hw = 1  parity_sw = 1
Test Successful!
```

.v File in HLS:

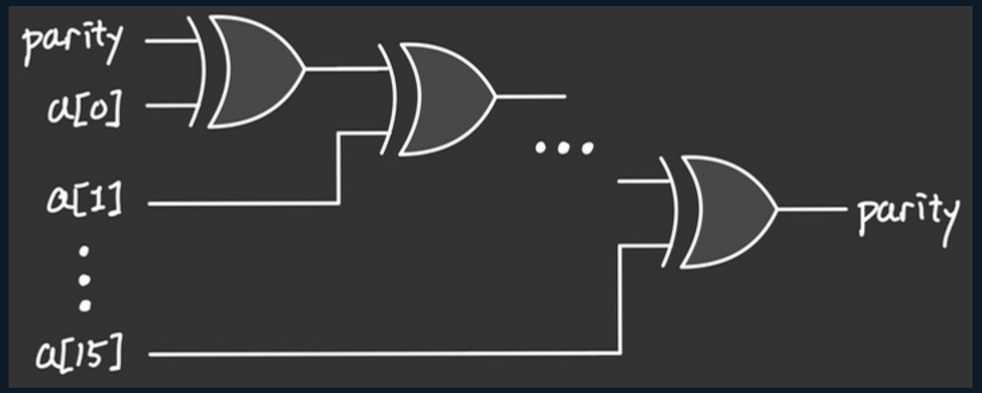
```
assign ap_return = (xor_ln15_13_fu_256_p2 ^ tmp_14_fu_262_p3);
assign xor_ln15_10_fu_238_p2 = (tmp_13_fu_170_p3 ^ tmp_12_fu_162_p3);

assign xor_ln15_11_fu_244_p2 = (xor_ln15_9_fu_232_p2 ^ xor_ln15_10_fu_238_p2);

assign xor_ln15_12_fu_250_p2 = (xor_ln15_8_fu_226_p2 ^ xor_ln15_11_fu_244_p2);

assign xor_ln15_13_fu_256_p2 = (xor_ln15_5_fu_208_p2 ^ xor_ln15_12_fu_250_p2);

assign xor_ln15_1_fu_184_p2 = (xor_ln15_fu_178_p2 ^ tmp_fu_66_p3);

assign xor_ln15_2_fu_190_p2 = (tmp_3_fu_90_p3 ^ tmp_2_fu_82_p3);

assign xor_ln15_3_fu_196_p2 = (tmp_5_fu_106_p3 ^ tmp_4_fu_98_p3);

assign xor_ln15_4_fu_202_p2 = (xor_ln15_3_fu_196_p2 ^ xor_ln15_2_fu_190_p2);

assign xor_ln15_5_fu_208_p2 = (xor_ln15_4_fu_202_p2 ^ xor_ln15_1_fu_184_p2);

assign xor_ln15_6_fu_214_p2 = (tmp_7_fu_122_p3 ^ tmp_6_fu_114_p3);

assign xor_ln15_7_fu_220_p2 = (tmp_9_fu_138_p3 ^ tmp_8_fu_130_p3);

assign xor_ln15_8_fu_226_p2 = (xor_ln15_7_fu_220_p2 ^ xor_ln15_6_fu_214_p2);

assign xor_ln15_9_fu_232_p2 = (tmp_11_fu_154_p3 ^ tmp_10_fu_146_p3);

assign xor_ln15_fu_178_p2 = (tmp_1_fu_74_p3 ^ empty_fu_62_p1);

endmodule //parity_generator
```



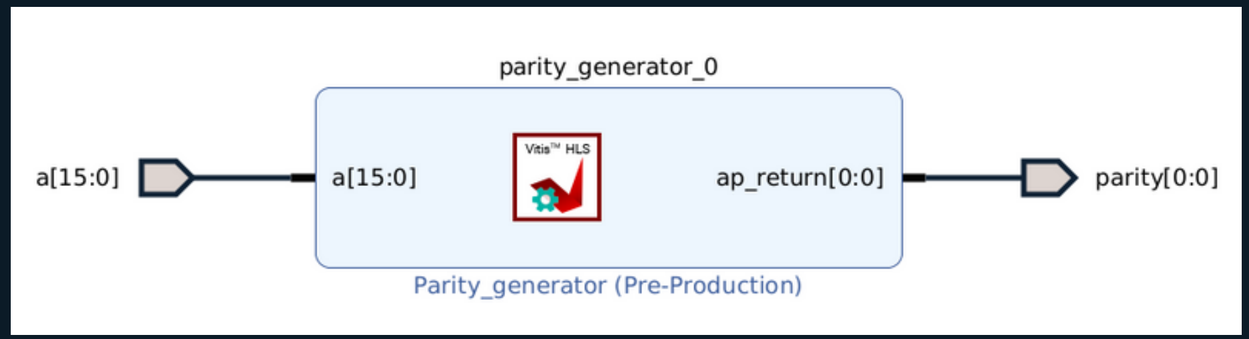
In Vivado:



Unroll manually




```
module parity_generator(
    input [15:0] a,
    output reg parity,
    output reg [16:0] even_parity
    );

    always @* begin
        parity = 1'b0 ^ a[0];
        parity = parity ^ a[1];
        parity = parity ^ a[2];
        parity = parity ^ a[3];
        parity = parity ^ a[4];
        parity = parity ^ a[5];
        parity = parity ^ a[6];
        parity = parity ^ a[7];
        parity = parity ^ a[8];
        parity = parity ^ a[9];
        parity = parity ^ a[10];
        parity = parity ^ a[11];
        parity = parity ^ a[12];
        parity = parity ^ a[13];
        parity = parity ^ a[14];
        parity = parity ^ a[15];
        even_parity = {parity, a};
    end

endmodule
```



The imported RTL IP.

In Vivado:

```
module parity_generator(
    input wire [15:0] a,
    output reg  [0:0] parity
    );

    integer i;

    always @* begin
        parity = 1'b0;
        for (i = 0; i < 16; i = i + 1) begin
            parity = parity ^ a[i];
        end
    end

endmodule
```

```
module parity_generator(
    input [15:0] a,
    output reg parity,
    output reg [16:0] even_parity
    );

    always @* begin
        parity = 1'b0 ^ a[0];
        parity = parity ^ a[1];
        parity = parity ^ a[2];
        parity = parity ^ a[3];
        parity = parity ^ a[4];
        parity = parity ^ a[5];
        parity = parity ^ a[6];
        parity = parity ^ a[7];
        parity = parity ^ a[8];
        parity = parity ^ a[9];
        parity = parity ^ a[10];
        parity = parity ^ a[11];
        parity = parity ^ a[12];
        parity = parity ^ a[13];
        parity = parity ^ a[14];
        parity = parity ^ a[15];
        even_parity = {parity, a};
    end

endmodule
```

Verilog unroll the loop to the RHS by default

In Vivado:

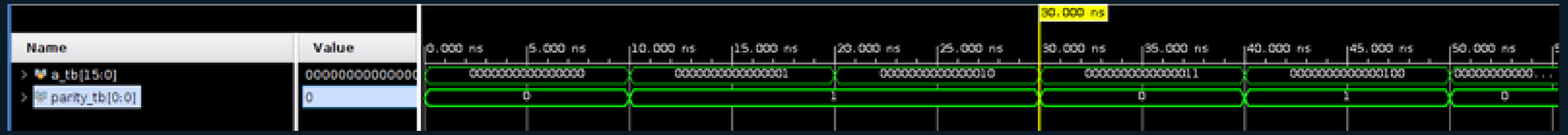
```
23      module parity_generator_tb();
24          reg [15:0] a_tb;
25          wire [0:0] parity_tb;
26
27          design_1_wrapper U0(.a(a_tb), .parity(parity_tb));
28
29          initial
30          begin
31          a_tb = 16'd0;
32          #10     a_tb = 16'd1;
33          #10     a_tb = 16'd2;
34          #10     a_tb = 16'd3;
35          #10     a_tb = 16'd4;
36          #10     a_tb = 16'd5;
37          #10     a_tb = 16'd6;
38          #10     a_tb = 16'd7;
39          #10     a_tb = 16'd8;
40          #10     a_tb = 16'd9;
41          #10     a_tb = 16'd10;
42          #10     a_tb = 16'd11;
43          #10     a_tb = 16'd12;
44          #10     a_tb = 16'd13;
45          #10     a_tb = 16'd14;
46          #10     a_tb = 16'd15;
47          #10     a_tb = 16'd16;
48          #10     a_tb = 16'd17;
49          #10     a_tb = 16'd18;
50          end
51
52
53      endmodule
```

Testbench for IP

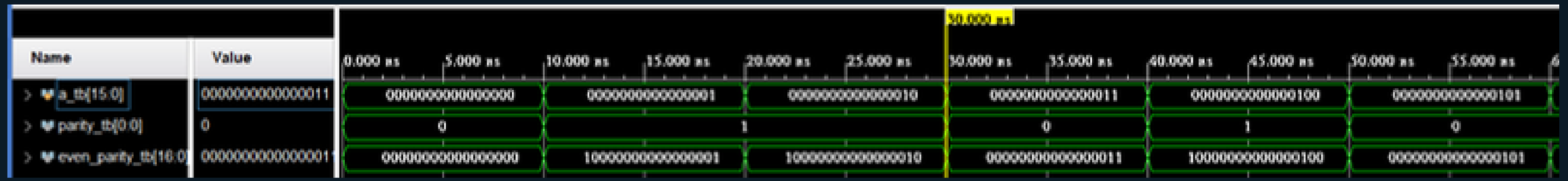
```
23      module parity_generator_tb();
24          reg [15:0] a_tb;
25          wire [0:0] parity_tb;
26          wire [16:0] even_parity_tb;
27          parity_generator U0(.a(a_tb), .parity(parity_tb), .even_parity(even_parity_tb));
28          initial
29          begin
30          a_tb = 16'd0;
31          #10     a_tb = 16'd1;
32          #10     a_tb = 16'd2;
33          #10     a_tb = 16'd3;
34          #10     a_tb = 16'd4;
35          #10     a_tb = 16'd5;
36          #10     a_tb = 16'd6;
37          #10     a_tb = 16'd7;
38          #10     a_tb = 16'd8;
39          #10     a_tb = 16'd9;
40          #10     a_tb = 16'd10;
41          #10     a_tb = 16'd11;
42          #10     a_tb = 16'd12;
43          #10     a_tb = 16'd13;
44          #10     a_tb = 16'd14;
45          #10     a_tb = 16'd15;
46          #10     a_tb = 16'd16;
47          end
48      endmodule
```

Testbench for verilog design

Testbench In Vivado:



the waveform of IP



the waveform of verilog design

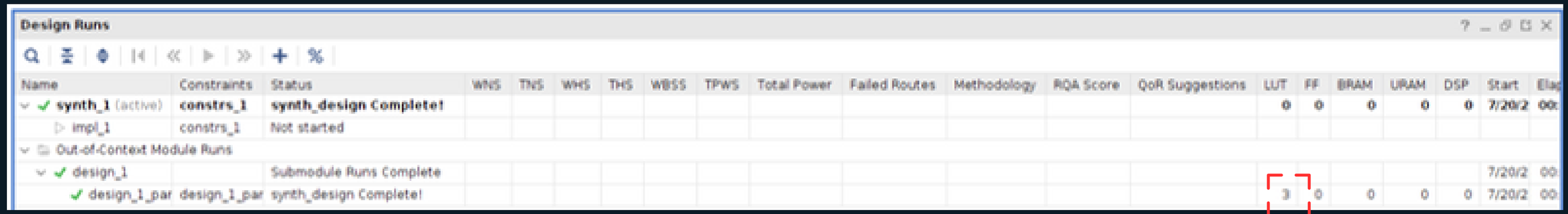
Another Version of Testbench:

```
22
23  module parity_generator_tb();
24      wire [0:0]  parity_hw;
25      reg  [0:0]  parity_sw;
26      reg [15:0]  a_hw;
27
28      design_1_wrapper U0 (.a(a_hw), .parity(parity_hw));
29
30      reg [15:0]  tmp = 16'd0;
31
32      integer i, j;
33
34      initial begin
35          for (j = 0; j < 65536; j = j + 1) begin
36              a_hw = tmp;
37              parity_generator_sw (tmp, parity_sw);
38              #5
39              if (parity_hw != parity_sw) begin
40                  $display("Error at %b, parity_sw = %b, parity_hw = %b", tmp, parity_sw, parity_hw);
41              end
42              else begin
43                  // $display("%b, parity_sw = %b, parity_hw = %b", tmp, parity_sw, parity_hw);
44              end
45              #5 tmp = tmp + 1'b1;
46          end
47      end
48
49      task parity_generator_sw;
50          input [15:0] a;
51          output [0:0] parity_sw;
52          begin
53              parity_sw = 1'b0;
54              for (i = 0; i < 16; i = i + 1) begin
55                  parity_sw = parity_sw ^ a[i];
56              end
57          end
58      endtask
59
60  endmodule
```

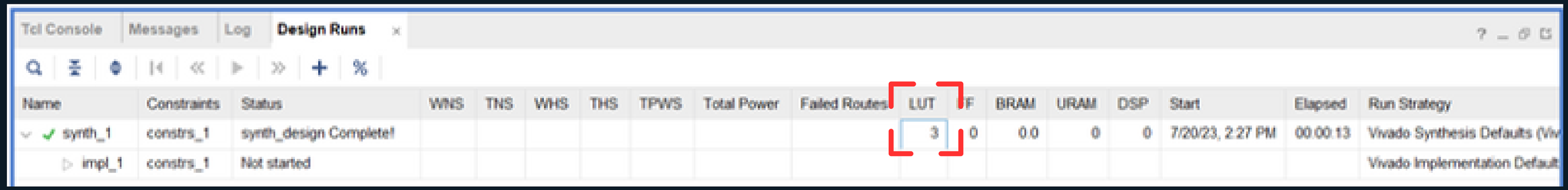
Waveform:


Name
Value
a_hw[15:0]
parity_hw[0:0]
parity_sw[0:0]
i[31:0]
j[31:0]
tmp[15:0]
90.000 ns
0.000 ns
5.000 ns
10.000 ns
15.000 ns
20.000 ns
25.000 ns
30.000 ns
35.000 ns
40.000 ns
45.000 ns
50.000 ns
55.000 ns
60.000 ns
65.000 ns
70.000 ns

In Vivado:



utilization of IP



utilization of verilog design

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	3	0	0	53200	<0.01
LUT as Logic	3	0	0	53200	<0.01
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

02. Leading One

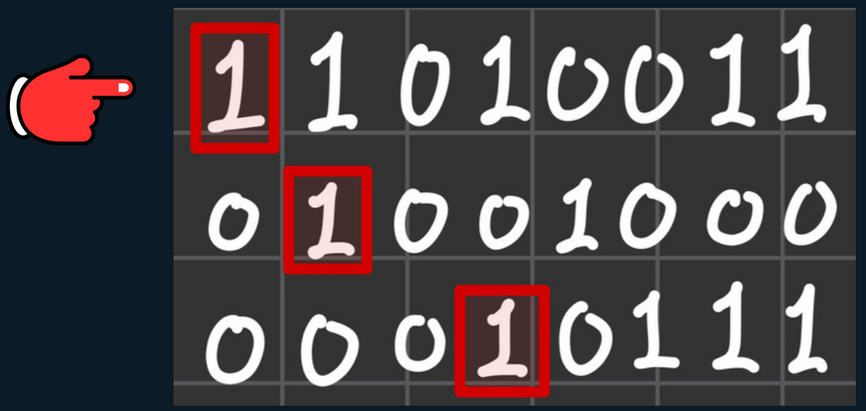
```cpp
#include "leading_one.h"

ap_int<5> leading_one(ap_uint<9> a) {
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_ctrl_none port=return



    ap_int<5> index;

    if (a[8] == 1) {
        index = 8;
    } else if (a[7] == 1) {
        index = 7;
    } else  if (a[6] == 1) {
        index = 6;
    } else  if (a[5] == 1) {
        index = 5;
    } else  if (a[4] == 1) {
        index = 4;
    } else  if (a[3] == 1) {
        index = 3;
    } else  if (a[2] == 1) {
        index = 2;
    } else  if (a[1] == 1) {
        index = 1;
    } else  if (a[0] == 1) {
        index = 0;
    } else {
        index = -1;
    }



    return index;
}
```

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


int find_leading_one_golden(int a) {
    int r;

    if (a == 0)
        r = -1;
    if (a > 0 && a < 2)
        r = 0;
    if (a > 1 && a < 4)
        r = 1;
    if (a > 3 && a < 8)
        r = 2;
    if (a > 7 && a < 16)
        r = 3;
    if (a > 15 && a < 32)
        r = 4;
    if (a > 31 && a < 64)
        r = 5;
    if (a > 63 && a < 128)
        r = 6;
    if (a > 127 && a < 256)
        r = 7;
    if (a > 255 && a < 512)
            r = 8;

    return r;
}
```

```cpp
int main() {
    int status = 0;

    for (int i = 0; i < 512; i++) {

        int        index_gold = find_leading_one_golden(i);
        ap_int<9> a = (ap_int<9>)i;
        ap_int<5> index = leading_one(a);

        if (index != index_gold) {
            status = -1;
            std::cout << "number = " << i << " index_gold = " << index_gold << " index = " << index << std::endl;
            std::cout << "number = " << a.to_string() << " leading one at " << index << std::endl;

        } else {                        // .to_string() : dec2bin
            std::cout << "number = " << a.to_string() << " leading one at " << index << std::endl;
        }

    }

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

    return status;
}
```

Kernel(Top Function)

```
#include "leading_one.h"

ap_int<5> leading_one(ap_uint<9> a) {
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_ctrl_none port=return



    ap_int<5> index;

    if (a[8] == 1) {
        index = 8;
    } else if (a[7] == 1) {
        index = 7;
    } else  if (a[6] == 1) {
        index = 6;
    } else  if (a[5] == 1) {
        index = 5;
    } else  if (a[4] == 1) {
        index = 4;
    } else  if (a[3] == 1) {
        index = 3;
    } else  if (a[2] == 1) {
        index = 2;
    } else  if (a[1] == 1) {
        index = 1;
    } else  if (a[0] == 1) {
        index = 0;
    } else {
        index = -1;
    }



    return index;
}
```



return the position of the first nonzero element in a row

Testbench

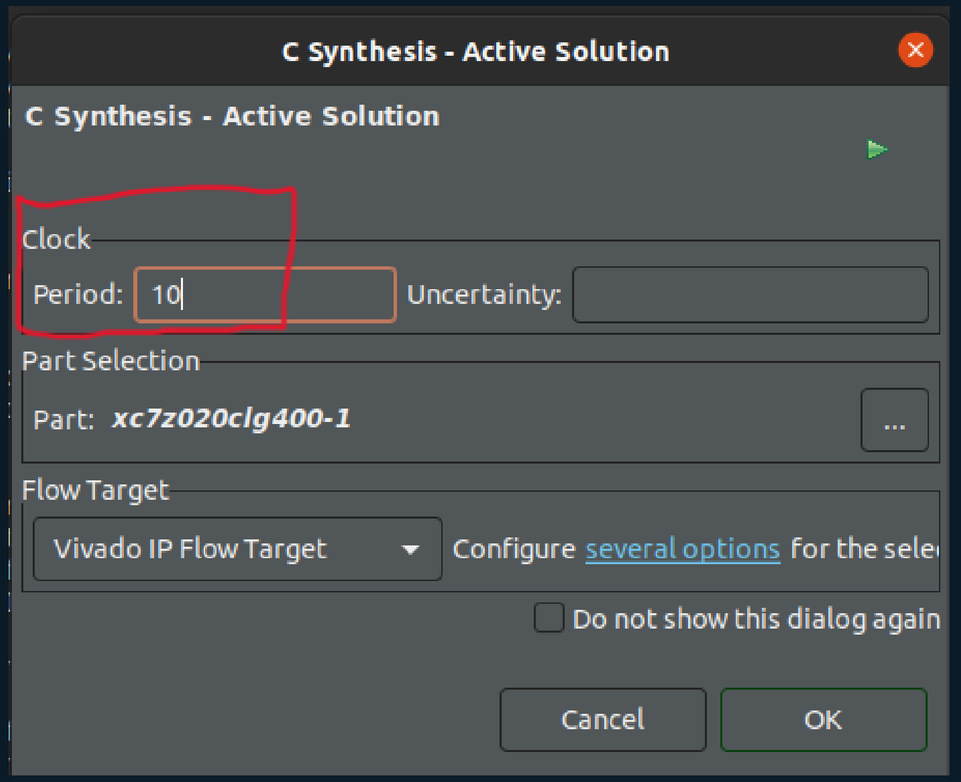
```cpp
int main() {
    int status = 0;

    for (int i = 0; i < 512; i++) {

        int        index_gold = find_leading_one_golden(i);
        ap_int<9> a = (ap_int<9>)i;
        ap_int<5> index = leading_one(a);

        if (index != index_gold) {
            status = -1;
            std::cout << "number = " << i << " index_gold = " << index_gold << " index = " << index << std::endl;
            std::cout << "number = " << a.to_string() << " leading one at " << index << std::endl;

        } else {                          // .to_string() : dec2bin
            std::cout << "number = " << a.to_string() << " leading one at " << index << std::endl;
        }

    }

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

    return status;
}
```

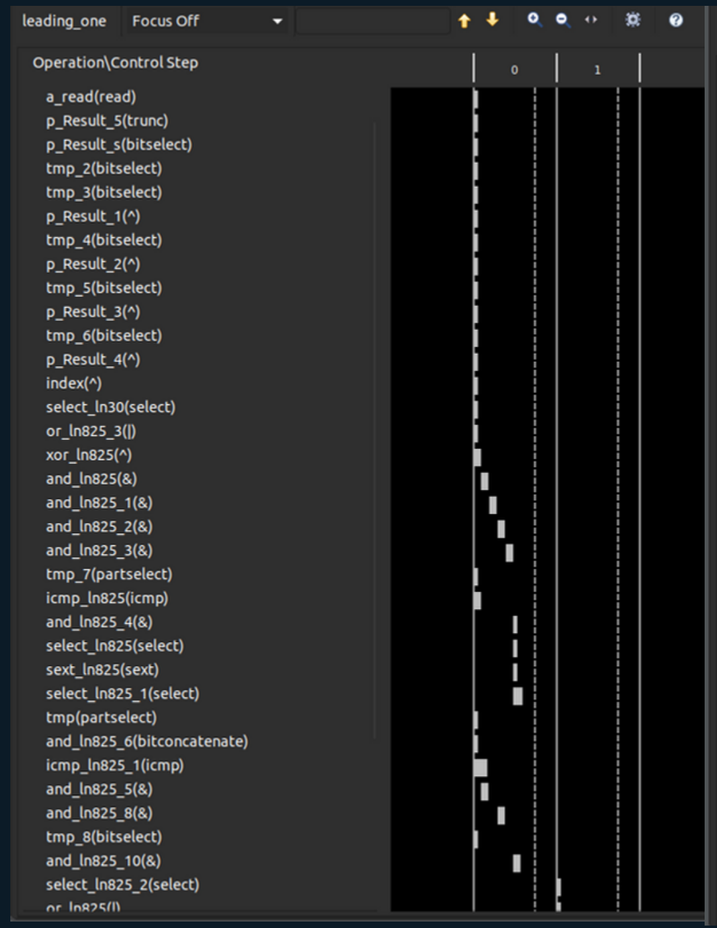
```cpp
#include "leading_one-tb.h"
#include <iostream>


int find_leading_one_golden(int a) {
    int r;

    if (a == 0)
        r = -1;
    if (a > 0 && a < 2)
        r = 0;
    if (a > 1 && a < 4)
        r = 1;
    if (a > 3 && a < 8)
        r = 2;
    if (a > 7 && a < 16)
        r = 3;
    if (a > 15 && a < 32)
        r = 4;
    if (a > 31 && a < 64)
        r = 5;
    if (a > 63 && a < 128)
        r = 6;
    if (a > 127 && a < 256)
        r = 7;
    if (a > 255 && a < 512)
            r = 8;

    return r;
}
```

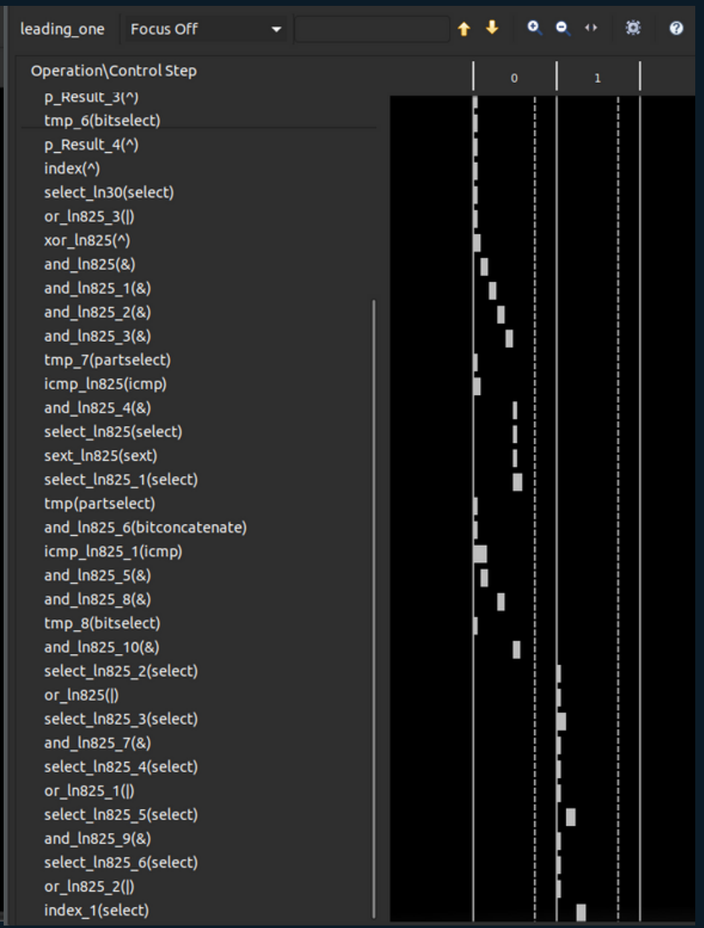
a.to_string() ----->

```
number = 0b0 leading one at -1
number = 0b01 leading one at 0
number = 0b010 leading one at 1
number = 0b011 leading one at 1
number = 0b0100 leading one at 2
number = 0b0101 leading one at 2
number = 0b0110 leading one at 2
number = 0b0111 leading one at 2
number = 0b01000 leading one at 3
number = 0b01001 leading one at 3
number = 0b01010 leading one at 3
number = 0b01011 leading one at 3
number = 0b01100 leading one at 3
number = 0b01101 leading one at 3
number = 0b01110 leading one at 3
number = 0b01111 leading one at 3
number = 0b010000 leading one at 4
number = 0b010001 leading one at 4
```

Synthesis Issue:

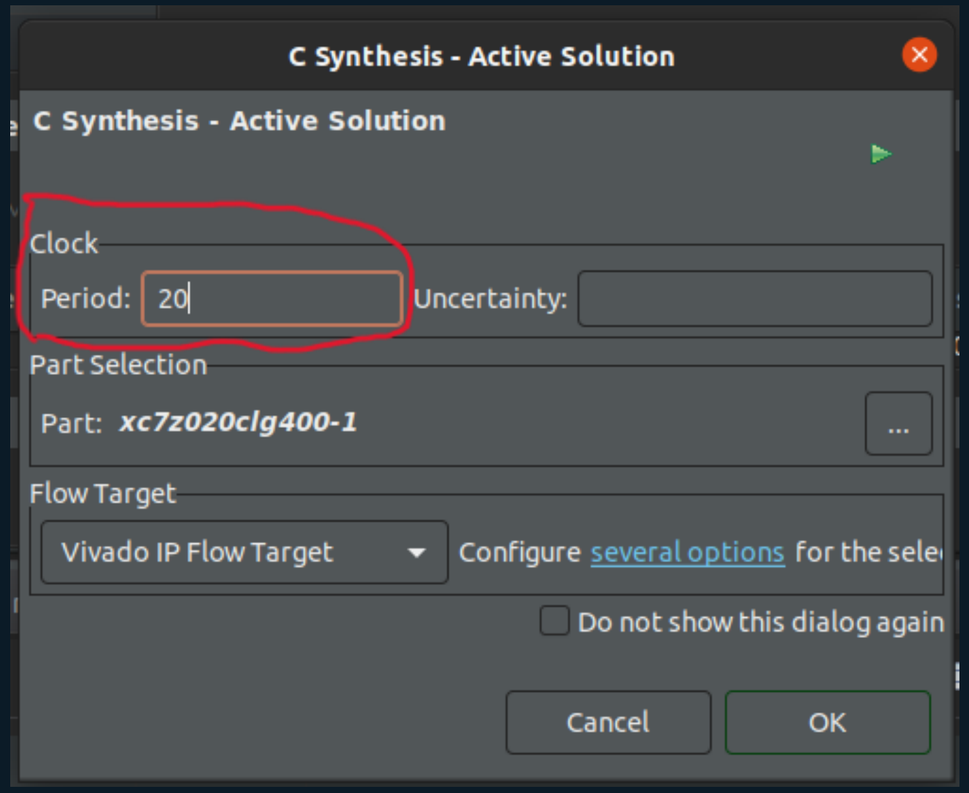


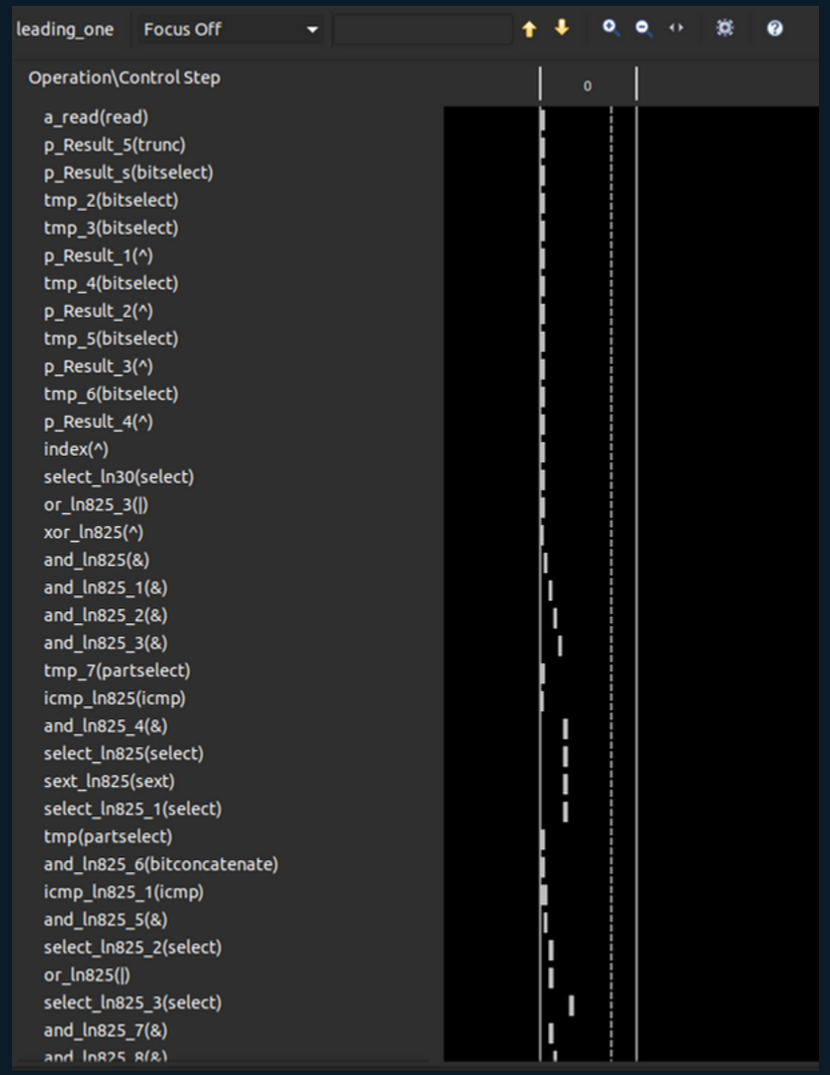




Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
leading_one				-	1	10.000	-	2	-	no	0		15	105	0

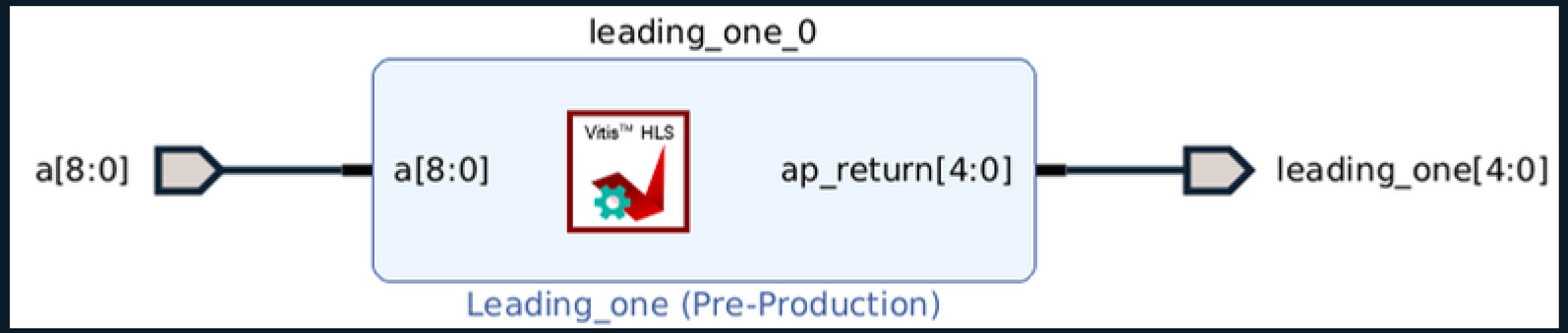
Solution:





Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
leading_one				-	0	0.0	-	1	-	no	0		0	91	0

In Vivado:



The imported RTL IP.

```
23  module leading_one(
24      input wire [8:0] a,
25      output reg [4:0] one
26      );
27      always @* begin
28          if (a[8] == 1'b1)
29              one = 5'd8;
30          else if (a[7] == 1'b1)
31              one = 5'd7;
32          else if (a[6] == 1'b1)
33              one = 5'd6;
34          else if (a[5] == 1'b1)
35              one = 5'd5;
36          else if (a[4] == 1'b1)
37              one = 5'd4;
38          else if (a[3] == 1'b1)
39              one = 5'd3;
40          else if (a[2] == 1'b1)
41              one = 5'd2;
42          else if (a[1] == 1'b1)
43              one = 5'd1;
44          else if (a[0] == 1'b1)
45              one = 5'd0;
46          else
47              one = 5'd0 - 1;
48      end
49  endmodule
```

Testbench In Vivado:

```
23  module leading_one_tb();
24      reg [8:0] a_tb;
25      wire [4:0] one_tb;
26
27      design_1_wrapper U0(.a(a_tb), .leading_one(one_tb));
28
29      initial
30      begin
31      a_tb = 9'd0;
32      #20     a_tb = 9'd1;
33      #20     a_tb = 9'd2;
34      #20     a_tb = 9'd4;
35      #20     a_tb = 9'd8;
36      #20     a_tb = 9'd16;
37      #20     a_tb = 9'd32;
38      #20     a_tb = 9'd64;
39      #20     a_tb = 9'd128;
40      #20     a_tb = 9'd256;
41      end
42
43  endmodule
```

Testbench for IP

```
23  module leading_one_tb();
24      wire [4:0] one_tb;
25      reg [8:0] a_tb;
26
27      leading_one U0(.one(one_tb), .a(a_tb));
28
29      initial
30      begin
31      a_tb = 9'd0;
32      #20     a_tb = 9'd1;
33      #20     a_tb = 9'd2;
34      #20     a_tb = 9'd4;
35      #20     a_tb = 9'd8;
36      #20     a_tb = 9'd16;
37      #20     a_tb = 9'd32;
38      #20     a_tb = 9'd64;
39      #20     a_tb = 9'd128;
40      #20     a_tb = 9'd256;
41      end
42  endmodule
```

Testbench for verilog design

In Vivado:

waveform of IP

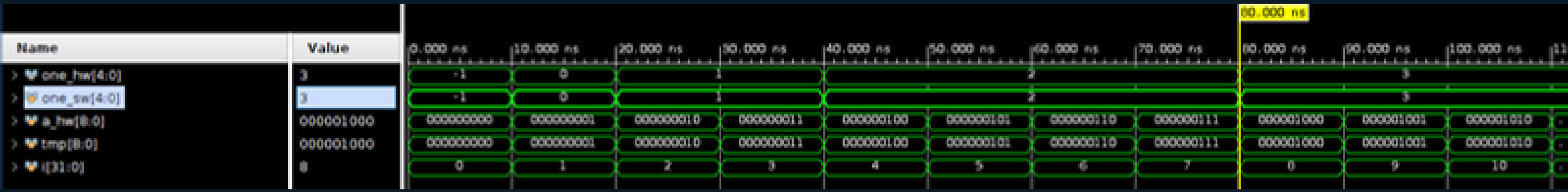
wavefrom of verilog design

Another Version of Testbench:

```
22
23      module leading_one_tb();
24          wire [4:0] one_hw;
25          reg  [4:0] one_sw;
26          reg  [8:0] a_hw;
27
28          design_1_wrapper U0 (.a(a_hw), .leading_one(one_hw));
29
30          reg [8:0] tmp = 9'd0;
31          integer i;
32
33          initial begin
34              for (i = 0; i < 512; i = i + 1) begin
35                  a_hw = tmp;
36                  leading_one_sw (tmp, one_sw);
37                  #5
38                  if (one_hw != one_sw) begin
39                      $display("Error at %b, leading_one_sw = %d, leading_one_hw = %d", tmp, one_sw, one_hw);
40                  end
41                  else begin
42                      $display("%b, leading_one_sw = %d, leading_one_hw = %d", tmp, one_sw, one_hw);
43                  end
44                  #5 tmp = tmp + 1'b1;
45              end
46          end
47
48          task leading_one_sw;
49              input [8:0] a;
50              output [4:0] leading_one_sw;
51              begin
52                  if (a[8] == 1'b1)
53                      leading_one_sw = 5'd8;
54                  else if (a[7] == 1'b1)
55                      leading_one_sw = 5'd7;
56                  else if (a[6] == 1'b1)
57                      leading_one_sw = 5'd6;
58                  else if (a[5] == 1'b1)
```

```
59                      leading_one_sw = 5'd5;
60                  else if (a[4] == 1'b1)
61                      leading_one_sw = 5'd4;
62                  else if (a[3] == 1'b1)
63                      leading_one_sw = 5'd3;
64                  else if (a[2] == 1'b1)
65                      leading_one_sw = 5'd2;
66                  else if (a[1] == 1'b1)
67                      leading_one_sw = 5'd1;
68                  else if (a[0] == 1'b1)
69                      leading_one_sw = 5'd0;
70                  else
71                      leading_one_sw = 5'd0 - 1;
72              end
73          endtask
74
75      endmodule
76
```

Waveform:


Name
Value
one_hw[4:0]
3
one_sw[4:0]
3
a_hw[8:0]
000001000
tmp[8:0]
000001000
i[31:0]
8
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
-1
0
1
2
3
000000000
000000001
000000010
000000011
000000100
000000101
000000110
000000111
000001000
000001001
000001010
0
1
2
3
4
5
6
7
8
9
10

In Vivado:

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**	**7/20/2**	**00:00:**
impl_1	constrs_1	Not started																		
Out-of-Context Module Runs																				
design_1		Submodule Runs Complete																	7/20/2	00:01:
design_1_lea	design_1_lea	synth_design Complete!												7	0	0	0	0	7/20/2	00:01:

utilization of IP

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	synth_design Complete!								7	0	0.0	0	0	7/20/2	00:00:14	Vivado Synthesis Defaults (Vivado Sy
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Viva

utilization of verilog design

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	7	0	0	53200	0.01
LUT as Logic	7	0	0	53200	0.01
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

03.Integer Division&Modulus

```
const long int n = 1234101;


int divbyconstant(int a) {
#pragma HLS INLINE off

    return a / n;
}

void integer_division_modulus(int a, int &r) {
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=a
#pragma HLS INTERFACE ap_none port=r

    //r = a%n;

    //r = a - n*(a/n);

    r = a-n*divbyconstant(a);
}
```

Why we need an inline off here?

Why not use "%"?

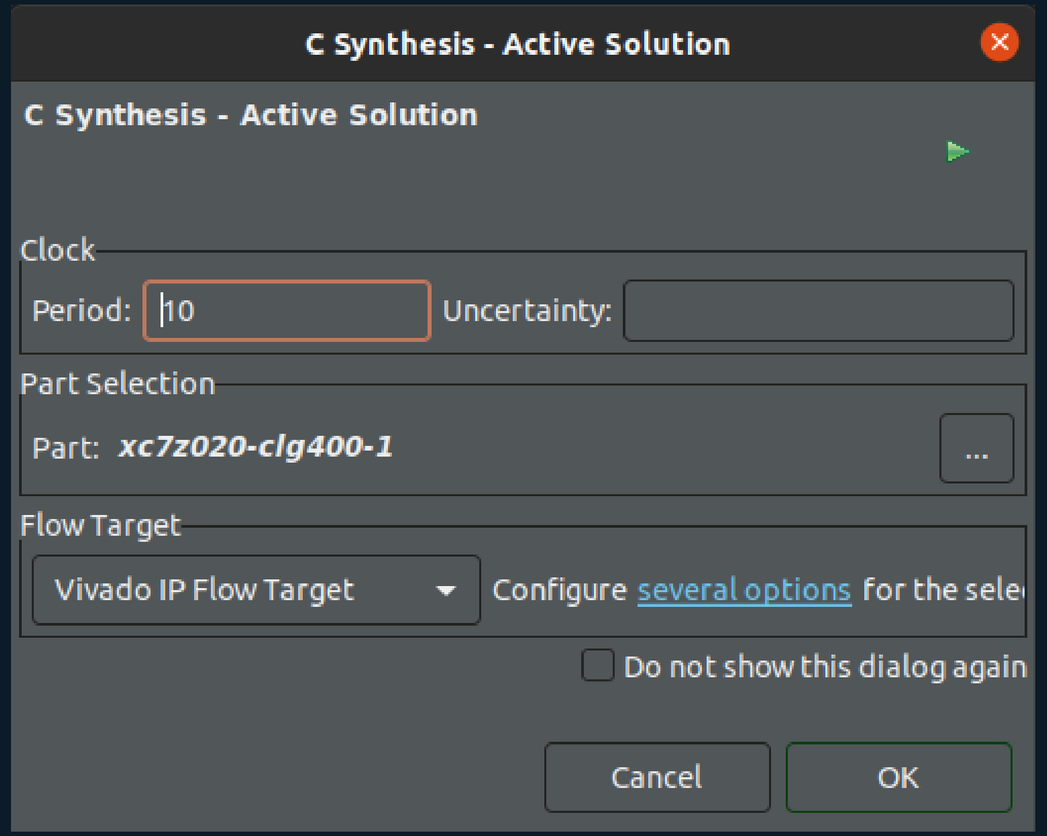
Why not just use "%" ?

Testbench

```cpp
#include <iostream>
#include "integer_division_modulus_tb.h"

int main() {
    for (long int i = 1234099; i < 1234120; i++) {
        int a = i, b;
        int &r = b;
        integer_division_modulus(a, r);
        std::cout << "r = " << b << std::endl;
    }
}
```

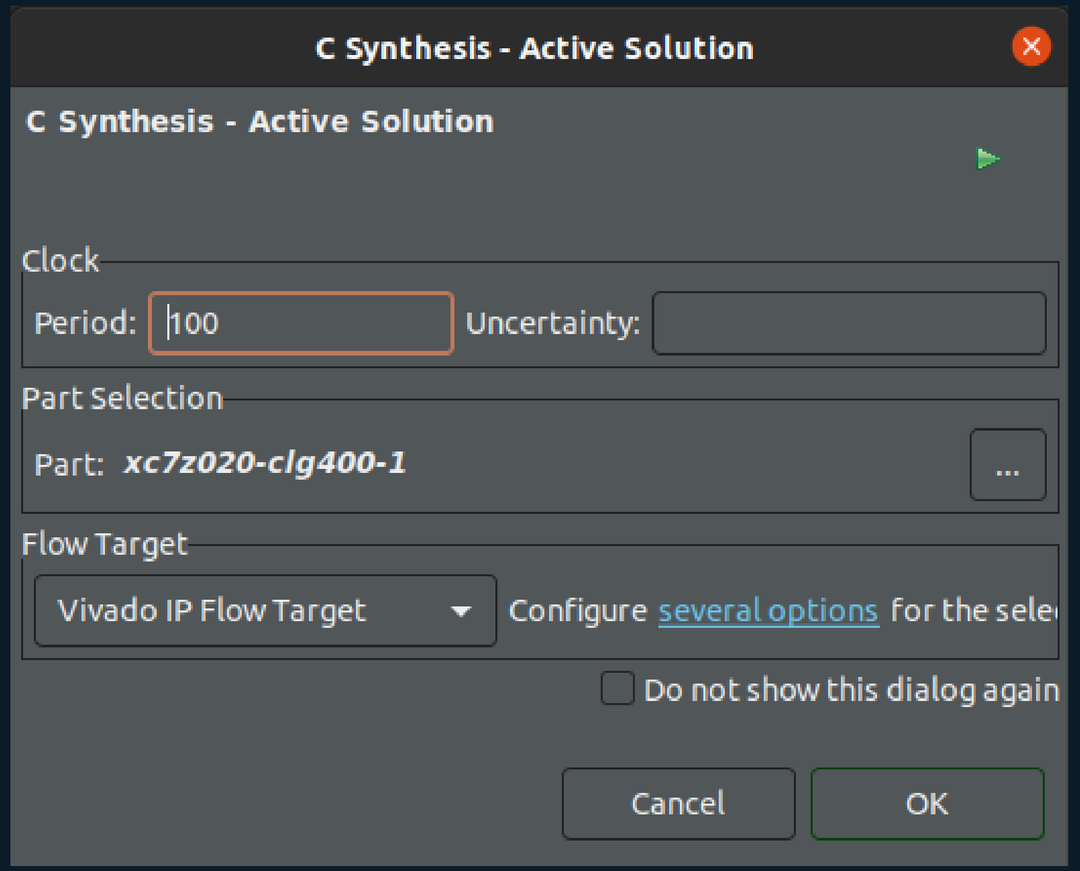
```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../integer_division_modulus_tb.cpp in debug mode
   Generating csim.exe
r = 1234099
r = 1234100
r = 0
r = 1
r = 2
r = 3
r = 4
r = 5
r = 6
r = 7
r = 8
r = 9
r = 10
r = 11
r = 12
r = 13
r = 14
r = 15
r = 16
r = 17
r = 18
INFO: [SIM 1] CSim done with 0 errors.
```

Synthesis Issue:



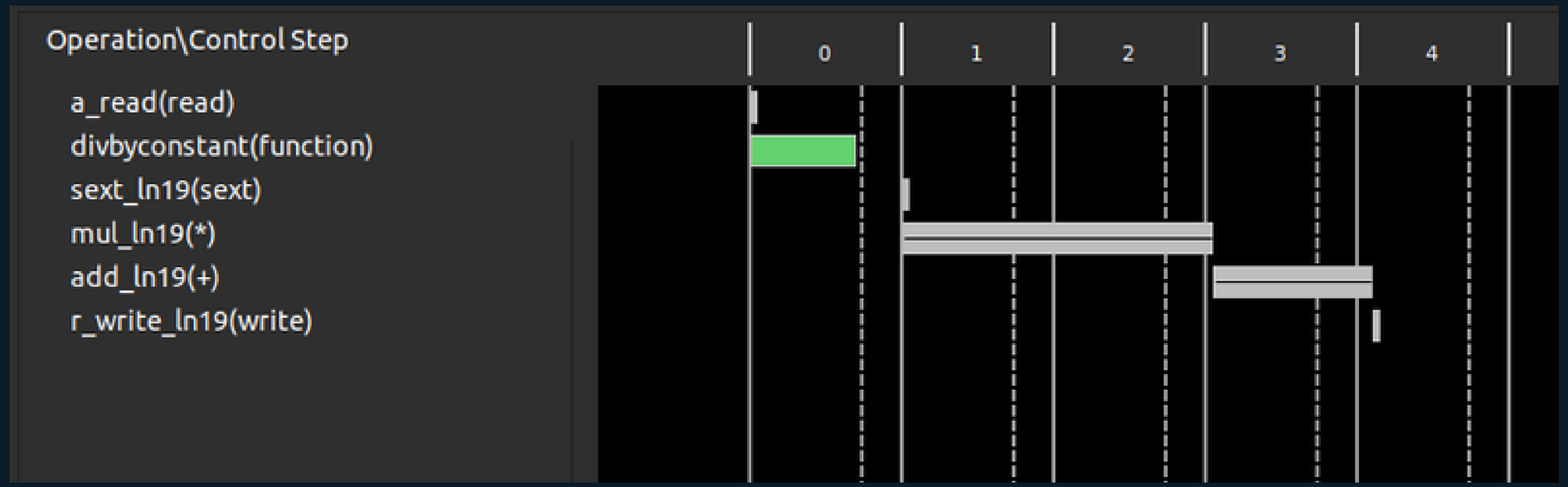
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
integer_division_modulus				-	6	60.000	-	7	-	no	0	4	260	213	0
divbyconstant				-	2	20.000	-	2	-	no	0	3	254	182	0

Synthesis Issue:



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
integer_division_modulus				-	3	300.000	-	4	-	no	0	4	4	155	0
divbyconstant				-	0	0.0	-	0	-	no	0	3	0	130	0

Synthesis Issue:



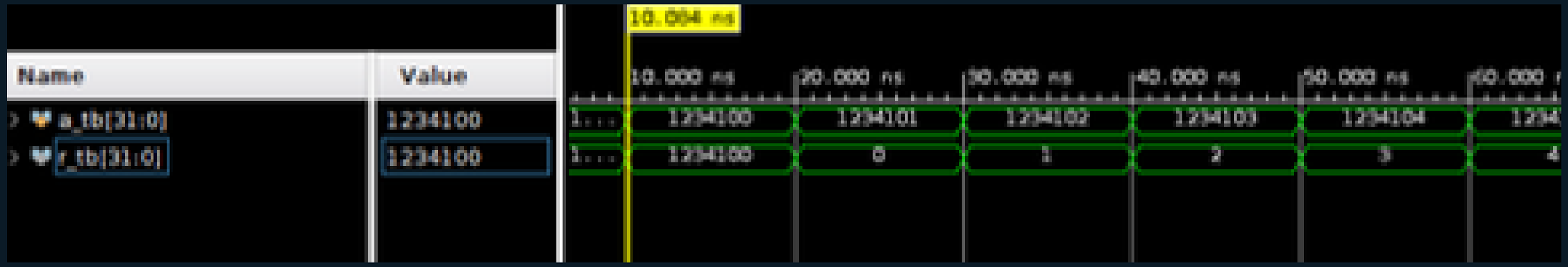
In vivado:

```
module golden_integer_division_modulus(
    input [31:0]a,
    output reg [31:0] r
    );
    //internal function
    function [31:0] divbyconstant;
        input [31:0] a;
        parameter n = 1234101;
        divbyconstant = a / n;
    endfunction

    //main function
    always @(a)
    begin
        r = a -(1234101 * divbyconstant(a));
    end
endmodule
```

In vivado: Testbench

```
module integer_division_modulus_tb();
    reg [31:0] a_tb;
    wire [31:0] r_tb;

    integer_division_modulus u0(.a(a_tb), .r(r_tb));

    initial
    begin
    #10 a_tb = 32'd1234099;
    #10 a_tb = 32'd1234100;
    #10 a_tb = 32'd1234101;
    #10 a_tb = 32'd1234102;
    #10 a_tb = 32'd1234103;
    #10 a_tb = 32'd1234104;
    #10 a_tb = 32'd1234105;
    #10 a_tb = 32'd1234106;
    #10 a_tb = 32'd1234107;
    #10 a_tb = 32'd1234108;
    #10 a_tb = 32'd1234109;
    #10 a_tb = 32'd1234110;
    end
endmodule
```

In vivado: waveform



In vivado: Synthesis Issue

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	279	0	0	53200	0.52
LUT as Logic	279	0	0	53200	0.52
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

In vivado: Testbench(another way)

```
module integer_division_modulus_tb();
    reg clk;
    reg rst;
    reg [31:0] a_tb;
    wire [31:0] r_sw_tb;
    wire [31:0] r_hw_tb;
    integer i;

    design_1_wrapper U0 (.ap_clk_0(clk),
                         .ap_rst_0(rst),
                         .a_0(a_tb),
                         .r_0(r_hw_tb));
    golden_integer_division_modulus A0(.a(a_tb),
                                .r(r_sw_tb));

    initial begin
        clk = 1'b1;
        forever begin
            #1 clk = ~clk;
        end
    end

    initial begin
        rst = 1'b1;
        a_tb = 32'd1234101;

        #5 @(posedge clk) rst = 1'b0;

        for (i = 0; i < 200; i = i + 1) begin
            #5
            if (r_sw_tb != r_hw_tb) begin
                $display("Error when a = %d, r = %d in software, r = %d in hardware", a_tb, r_sw_tb, r_hw_tb);
            end
            else begin
                $display("a = %d, r = %d in software, r = %d in hardware", a_tb, r_sw_tb, r_hw_tb);
            end
            #5
            a_tb = a_tb + 1'b1;
        end
    end
endmodule
```

```
a =    1234101, r =          0 in software, r =          0 in hardware
a =    1234102, r =          1 in software, r =          1 in hardware
a =    1234103, r =          2 in software, r =          2 in hardware
a =    1234104, r =          3 in software, r =          3 in hardware
a =    1234105, r =          4 in software, r =          4 in hardware
a =    1234106, r =          5 in software, r =          5 in hardware
a =    1234107, r =          6 in software, r =          6 in hardware
a =    1234108, r =          7 in software, r =          7 in hardware
a =    1234109, r =          8 in software, r =          8 in hardware
a =    1234110, r =          9 in software, r =          9 in hardware
a =    1234111, r =         10 in software, r =         10 in hardware
a =    1234112, r =         11 in software, r =         11 in hardware
a =    1234113, r =         12 in software, r =         12 in hardware
a =    1234114, r =         13 in software, r =         13 in hardware
a =    1234115, r =         14 in software, r =         14 in hardware
a =    1234116, r =         15 in software, r =         15 in hardware
a =    1234117, r =         16 in software, r =         16 in hardware
a =    1234118, r =         17 in software, r =         17 in hardware
a =    1234119, r =         18 in software, r =         18 in hardware
a =    1234120, r =         19 in software, r =         19 in hardware
a =    1234121, r =         20 in software, r =         20 in hardware
a =    1234122, r =         21 in software, r =         21 in hardware
a =    1234123, r =         22 in software, r =         22 in hardware
a =    1234124, r =         23 in software, r =         23 in hardware
a =    1234125, r =         24 in software, r =         24 in hardware
a =    1234126, r =         25 in software, r =         25 in hardware
a =    1234127, r =         26 in software, r =         26 in hardware
a =    1234128, r =         27 in software, r =         27 in hardware
a =    1234129, r =         28 in software, r =         28 in hardware
a =    1234130, r =         29 in software, r =         29 in hardware
a =    1234131, r =         30 in software, r =         30 in hardware
a =    1234132, r =         31 in software, r =         31 in hardware
a =    1234133, r =         32 in software, r =         32 in hardware
a =    1234134, r =         33 in software, r =         33 in hardware
a =    1234135, r =         34 in software, r =         34 in hardware
a =    1234136, r =         35 in software, r =         35 in hardware
a =    1234137, r =         36 in software, r =         36 in hardware
a =    1234138, r =         37 in software, r =         37 in hardware
a =    1234139, r =         38 in software, r =         38 in hardware
a =    1234140, r =         39 in software, r =         39 in hardware
a =    1234141, r =         40 in software, r =         40 in hardware
a =    1234142, r =         41 in software, r =         41 in hardware
a =    1234143, r =         42 in software, r =         42 in hardware
a =    1234144, r =         43 in software, r =         43 in hardware
a =    1234145, r =         44 in software, r =         44 in hardware
a =    1234146, r =         45 in software, r =         45 in hardware
a =    1234147, r =         46 in software, r =         46 in hardware
a =    1234148, r =         47 in software, r =         47 in hardware
a =    1234149, r =         48 in software, r =         48 in hardware
a =    1234150, r =         49 in software, r =         49 in hardware
a =    1234151, r =         50 in software, r =         50 in hardware
```

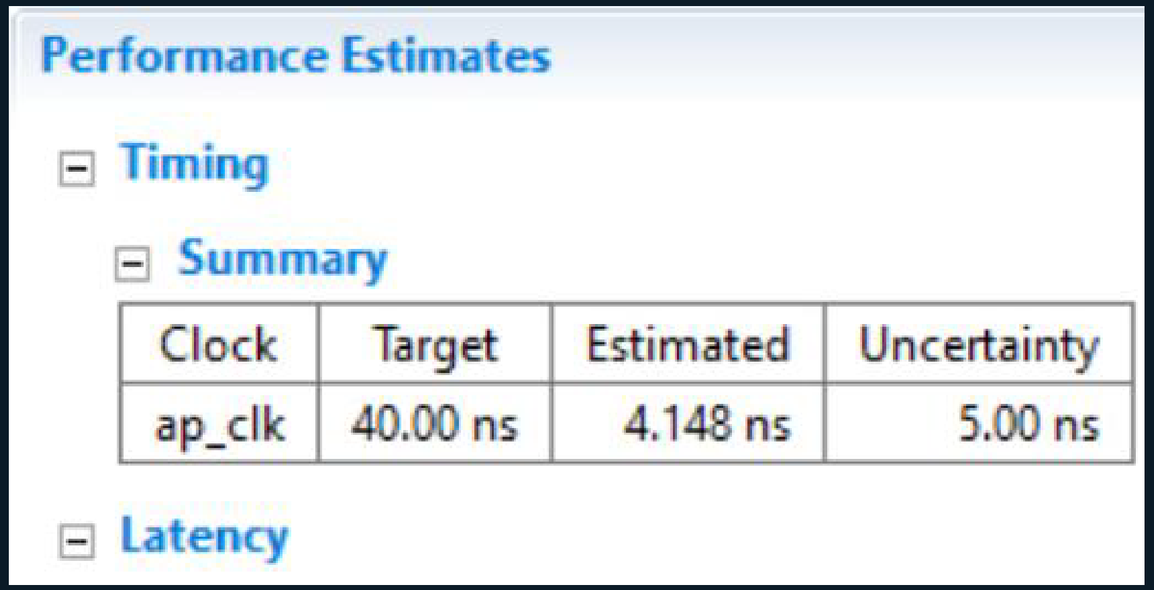
In vivado: Testbench(another way)

```
a =    1234101, r =          0 in software, r =          0 in hardware   a =    1234101, r =          0 in software, r =          0 in hard
a =    1234102, r =          1 in software, r =          1 in hardware   a =    1234102, r =          1 in software, r =          1 in hard
a =    1234103, r =          2 in software, r =          2 in hardware   a =    1234103, r =          2 in software, r =          2 in hard
a =    1234104, r =          3 in software, r =          3 in hardware   a =    1234104, r =          3 in software, r =          3 in hard
a =    1234105, r =          4 in software, r =          4 in hardware   a =    1234105, r =          4 in software, r =          4 in hard
a =    1234106, r =          5 in software, r =          5 in hardware   a =    1234106, r =          5 in software, r =          5 in hard
a =    1234107, r =          6 in software, r =          6 in hardware   a =    1234107, r =          6 in software, r =          6 in hard
a =    1234108, r =          7 in software, r =          7 in hardware   a =    1234108, r =          7 in software, r =          7 in hard
a =    1234109, r =          8 in software, r =          8 in hardware   a =    1234109, r =          8 in software, r =          8 in hard
a =    1234110, r =          9 in software, r =          9 in hardware   a =    1234110, r =          9 in software, r =          9 in hard
a =    1234111, r =         10 in software, r =         10 in hardware   a =    1234111, r =         10 in software, r =         10 in hard
a =    1234112, r =         11 in software, r =         11 in hardware   a =    1234112, r =         11 in software, r =         11 in hard
a =    1234113, r =         12 in software, r =         12 in hardware   a =    1234113, r =         12 in software, r =         12 in hard
a =    1234114, r =         13 in software, r =         13 in hardware   a =    1234114, r =         13 in software, r =         13 in hard
a =    1234115, r =         14 in software, r =         14 in hardware   a =    1234115, r =         14 in software, r =         14 in hard
a =    1234116, r =         15 in software, r =         15 in hardware   a =    1234116, r =         15 in software, r =         15 in hard
a =    1234117, r =         16 in software, r =         16 in hardware   a =    1234117, r =         16 in software, r =         16 in hard
a =    1234118, r =         17 in software, r =         17 in hardware   a =    1234118, r =         17 in software, r =         17 in hard
a =    1234119, r =         18 in software, r =         18 in hardware   a =    1234119, r =         18 in software, r =         18 in hard
a =    1234120, r =         19 in software, r =         19 in hardware   a =    1234120, r =         19 in software, r =         19 in hard
a =    1234121, r =         20 in software, r =         20 in hardware   a =    1234121, r =         20 in software, r =         20 in hard
a =    1234122, r =         21 in software, r =         21 in hardware   a =    1234122, r =         21 in software, r =         21 in hard
a =    1234123, r =         22 in software, r =         22 in hardware   a =    1234123, r =         22 in software, r =         22 in hard
a =    1234124, r =         23 in software, r =         23 in hardware   a =    1234124, r =         23 in software, r =         23 in hard
a =    1234125, r =         24 in software, r =         24 in hardware   a =    1234125, r =         24 in software, r =         24 in hard
a =    1234126, r =         25 in software, r =         25 in hardware   a =    1234126, r =         25 in software, r =         25 in hard
a =    1234127, r =         26 in software, r =         26 in hardware   a =    1234127, r =         26 in software, r =         26 in hard
```

In vivado: Synthesis Issue

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**
impl_1	constrs_1	Not started																
Out-of-Context Module Runs																		
design_1		Submodule Runs Complete																
design_1_inte	design_1_int	synth_design Complete!												136	0	0	0	5

why not use "%"



timing of a single stage

The resulted hardware was not fully combinational and required multiple stage to complete.

why not use "%"

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.148 ns	0.62 ns

Latency

Summary

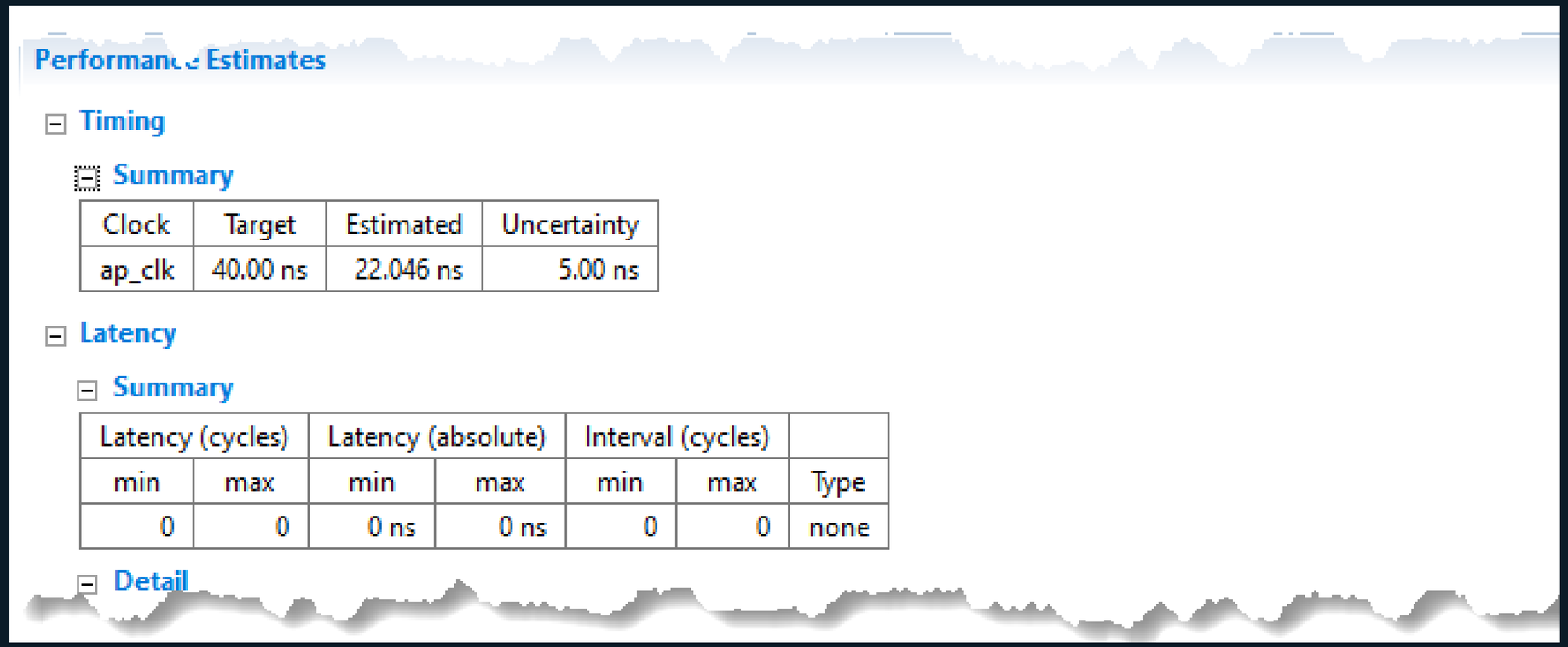
Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
35	35	0.175 us	0.175 us	35	35	none

Detail

whole design latency

The resulted hardware was not fully combinational and required multiple stage to complete.

why not use "%"



Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	40.00 ns	22.046 ns	5.00 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
0	0	0 ns	0 ns	0	0	none

Detail

compare with combinational one

04. Binary-to-BCD(Division)

```
void get_digit(uint14 &a, uint4 &digit) {
    digit = a-10*(a/10);
    a = a/10;
}
```

```
void binary2bcd_div(uint14 in_binary, uint16 *packed_bcd) {
#pragma HLS INTERFACE ap_none port=in_binary
// #pragma HLS INTERFACE ap_none port=unpacked_bcd
#pragma HLS INTERFACE ap_none port=packed_bcd
#pragma HLS INTERFACE ap_ctrl_none port=return


    uint14 a = in_binary;

    uint4 digit_1;
    uint4 digit_2;
    uint4 digit_3;
    uint4 digit_4;


    get_digit (a, digit_1);
    get_digit (a, digit_2);
    get_digit (a, digit_3);
    get_digit (a, digit_4);

    *packed_bcd   = (digit_4, digit_3, digit_2,  digit_1);

}
```

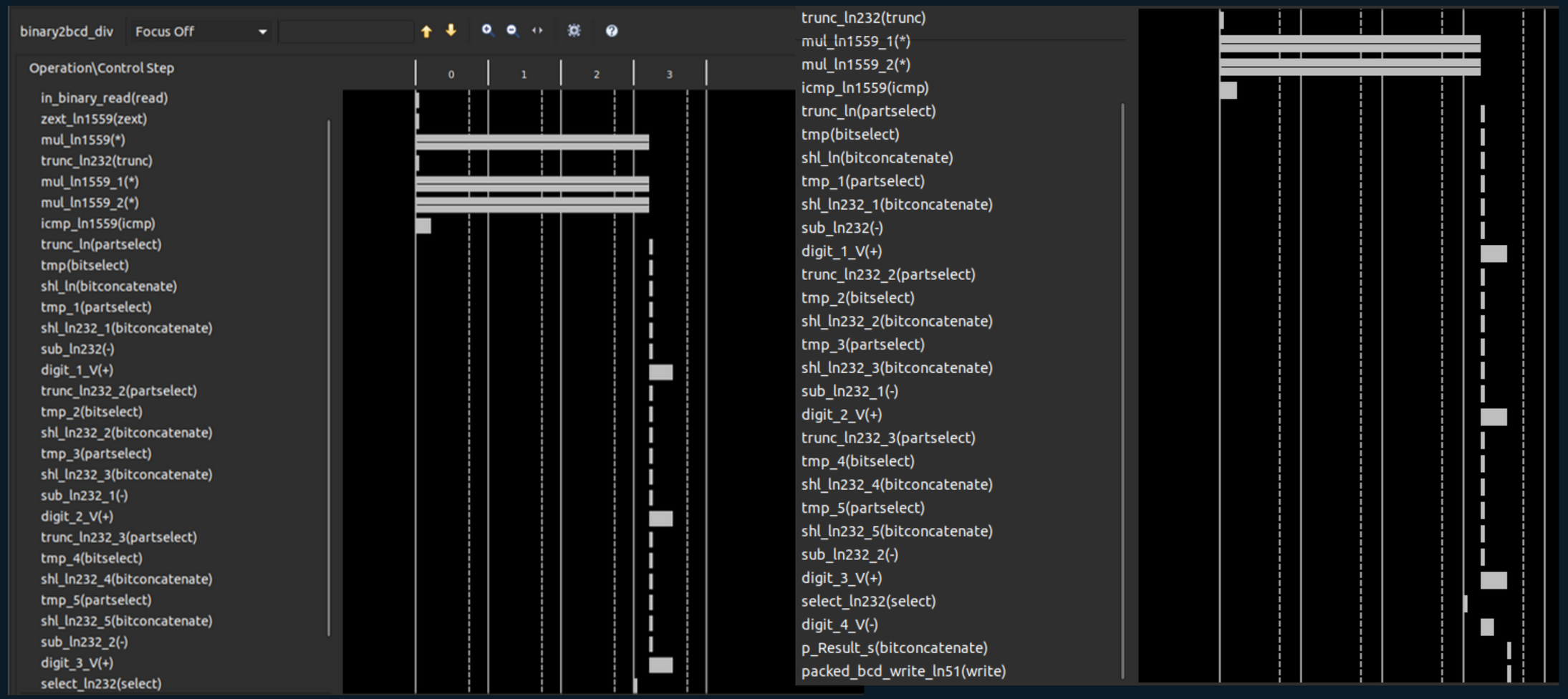
Testbench: compare result

```cpp
int main() {
    int status = 0;

    uint14 in_binary = 9999;

    uint16 packed_bcd_hw;
    uint16 packed_bcd_sw;

    for (int i = 0; i < 9999; i++) {
        in_binary = (uint14)i;
        binary2bcd_div(in_binary, &packed_bcd_hw);
        binary2bcd_software(in_binary, &packed_bcd_sw);

        if (packed_bcd_hw !=packed_bcd_sw) {
            status = -1;
            std::cout << "Error at " << i << "  in_binary = " << in_binary.to_string()
                      << " packed_bcd_hw = " << packed_bcd_hw.to_string()
                      << " packed_bcd_sw = " << packed_bcd_sw.to_string()
                      << std::endl;
            break;
        }
    }



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

    return status;
}
```

Testbench: software

```cpp
#include "binary2bcd_div-tb.h"
#include <iostream>

void binary2bcd_software(uint14 in_binary, uint16 *packed_bcd) {

    int packed_bcd_tmp;
    int in_binary_tmp = in_binary;

    if (in_binary == 100) {
        std::cout << std::endl;
    }

    int r = in_binary_tmp%10;
    packed_bcd_tmp = r;
    in_binary_tmp = in_binary_tmp/10;

    r = in_binary_tmp%10;
    packed_bcd_tmp = r*0b10000 + packed_bcd_tmp;
    in_binary_tmp = in_binary_tmp/10;

    r = in_binary_tmp%10;
    packed_bcd_tmp = r*0b100000000 + packed_bcd_tmp;
    in_binary_tmp = in_binary_tmp/10;

    r = in_binary_tmp%10;
    packed_bcd_tmp = r*0b1000000000000 + packed_bcd_tmp;

    *packed_bcd = packed_bcd_tmp;
}
```

Synthesis Issue(20ns):



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
binary2bcd_div				-	3	60.000	-	4	-	no	0	3	9	96	0

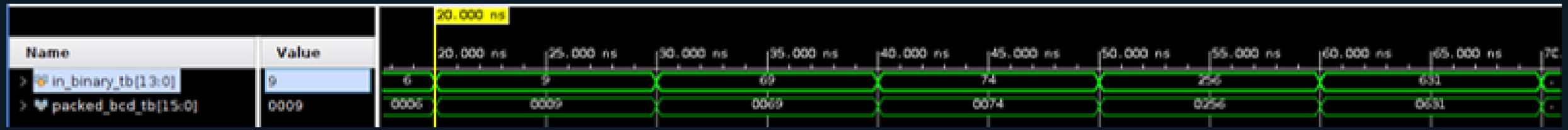
In vivado:

```
module binary2bcd_div(
    input wire [13:0] in_binary,
    output reg [15:0] packed_bcd
    );
    reg [13:0] a;
    reg [3:0] d1, d2, d3, d4;
    // function
    function [3:0]digit;
        input [13:0] a;
        digit = a - 10 * (a / 10);
    endfunction

    // main function
    always @* begin

        a = in_binary;
        d1 = digit(a);
        a = a/10;
        d2 = digit(a);
        a = a/10;
        d3 = digit(a);
        a = a/10;
        d4 = digit(a);
        packed_bcd = {d4, d3, d2, d1};
    end
endmodule
```

In vivado: Testbench

```
module binary2bcd_div_tb();
    reg [13:0] in_binary_tb;
    wire [15:0] packed_bcd_tb;

    binary2bcd_div u0(.in_binary(in_binary_tb), .packed_bcd(packed_bcd_tb));
    initial
    begin
    #10 in_binary_tb = 14'd6;
    #10 in_binary_tb = 14'd9;
    #10 in_binary_tb = 14'd69;
    #10 in_binary_tb = 14'd74;
    #10 in_binary_tb = 14'd256;
    #10 in_binary_tb = 14'd631;
    #10 in_binary_tb = 14'd2420;
    #10 in_binary_tb = 14'd6959;
    end

endmodule
```

In vivado: waveform



In vivado: Synthesis Issue

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	43	0	0	53200	0.08
LUT as Logic	43	0	0	53200	0.08
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

In vivado: Testbench(another way)

```
module binary2bcd_div_tb();
    reg clk;
    reg rst;
    reg [13:0] in_binary_tb;
    wire [15:0] packed_bcd_sw_tb;
    wire [15:0] packed_bcd_hw_tb;
    integer i;

    design_1_wrapper U0 (.ap_clk_0(clk),
                         .ap_rst_0(rst),
                         .in_binary_0(in_binary_tb),
                         .packed_bcd_0(packed_bcd_hw_tb));
    golden_binary2bcd_div A0 (.in_binary(in_binary_tb),
                        .packed_bcd(packed_bcd_sw_tb));

    initial begin
        clk = 1'b1;
        forever begin
            #1 clk = ~clk;
        end
    end

    initial begin
        rst = 1'b0;
        in_binary_tb = 14'd0;

        #5 @(posedge clk) rst = 1'b1;

        for (i = 0; i < 100; i = i + 1) begin
            #10
            if (packed_bcd_hw_tb != packed_bcd_sw_tb) begin
                $display("Error at %d, sw = %b, hw = %b", in_binary_tb, packed_bcd_sw_tb, packed_bcd_hw_tb);
            end
            else begin
                $display("Correct, sw = %b, hw = %b", packed_bcd_sw_tb, packed_bcd_hw_tb);
            end
            #5
            in_binary_tb = in_binary_tb + 1;
        end
    end
endmodule
```

```
Correct, sw = 0000000000000000, hw = 0000000000000000
Correct, sw = 0000000000000001, hw = 0000000000000001
Correct, sw = 0000000000000010, hw = 0000000000000010
Correct, sw = 0000000000000011, hw = 0000000000000011
Correct, sw = 0000000000000100, hw = 0000000000000100
Correct, sw = 0000000000000101, hw = 0000000000000101
Correct, sw = 0000000000000110, hw = 0000000000000110
Correct, sw = 0000000000000111, hw = 0000000000000111
Correct, sw = 0000000000001000, hw = 0000000000001000
Correct, sw = 0000000000001001, hw = 0000000000001001
Correct, sw = 0000000000010000, hw = 0000000000010000
Correct, sw = 0000000000010001, hw = 0000000000010001
Correct, sw = 0000000000010010, hw = 0000000000010010
Correct, sw = 0000000000010011, hw = 0000000000010011
Correct, sw = 0000000000010100, hw = 0000000000010100
Correct, sw = 0000000000010101, hw = 0000000000010101
Correct, sw = 0000000000010110, hw = 0000000000010110
Correct, sw = 0000000000010111, hw = 0000000000010111
Correct, sw = 0000000000011000, hw = 0000000000011000
Correct, sw = 0000000000011001, hw = 0000000000011001
Correct, sw = 0000000000100000, hw = 0000000000100000
Correct, sw = 0000000000100001, hw = 0000000000100001
Correct, sw = 0000000000100010, hw = 0000000000100010
Correct, sw = 0000000000100011, hw = 0000000000100011
Correct, sw = 0000000000100100, hw = 0000000000100100
Correct, sw = 0000000000100101, hw = 0000000000100101
Correct, sw = 0000000000100110, hw = 0000000000100110
Correct, sw = 0000000000100111, hw = 0000000000100111
Correct, sw = 0000000000101000, hw = 0000000000101000
Correct, sw = 0000000000101001, hw = 0000000000101001
Correct, sw = 0000000000110000, hw = 0000000000110000
Correct, sw = 0000000000110001, hw = 0000000000110001
Correct, sw = 0000000000110010, hw = 0000000000110010
Correct, sw = 0000000000110011, hw = 0000000000110011
Correct, sw = 0000000000110100, hw = 0000000000110100
Correct, sw = 0000000000110101, hw = 0000000000110101
Correct, sw = 0000000000110110, hw = 0000000000110110
Correct, sw = 0000000000110111, hw = 0000000000110111
Correct, sw = 0000000000111000, hw = 0000000000111000
Correct, sw = 0000000000111001, hw = 0000000000111001
```

In vivado: Synthesis Issue

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP
✓ **synth_1** (active)	**constrs_1**	**synth_design Complete!**												**0**	**0**	**0**	**0**	**0**
▷ impl_1	constrs_1	Not started																
Out-of-Context Module Runs																		
✓ design_1		Submodule Runs Complete																
✓ design_1_bin	design_1_bin	synth_design Complete!												11	9	0	0	3

05. Binary 2 BCD (Double Dabble)

243 in BCD representation:

Hundreds	Tens	Ones	Original
0010	0100	0011	11110011

Algorithm

The scratch space is initialized to all zeros.
The algorithm iterates n times.
On each iteration, any BCD digit which is at least 5 (0101 in binary) is incremented by 3 (0011); then the entire scratch space is left-shifted one bit.

Why add 3 when >= 5 ?

The increment ensures that a value of 5,
incremented and left-shifted, becomes 16 (10000),
thus correctly "carrying" into the next BCD digit.

Example

```
0000 0000 0000   11110011   Initialization
0000 0000 0001   11100110   Shift
0000 0000 0011   11001100   Shift
0000 0000 0111   10011000   Shift
0000 0000 1010   10011000   Add 3 to ONES, since it was 7
0000 0001 0101   00110000   Shift
0000 0001 1000   00110000   Add 3 to ONES, since it was 5
0000 0011 0000   01100000   Shift
0000 0110 0000   11000000   Shift
0000 1001 0000   11000000   Add 3 to TENS, since it was 6
0001 0010 0001   10000000   Shift
0010 0100 0011   00000000   Shift
   2    4    3
       BCD
```

Kernel (top function)

```
#include "binary2bcd_double_dabble.h"



void double_dabble(uint16 &scratch_pad) {
#pragma HLS INLINE

    scratch_pad = scratch_pad << 1;
    if (scratch_pad(11, 8) > 4)
        scratch_pad(11, 8) = scratch_pad(11, 8) + 3;

}



void binary2bcd_double_dabble(uint8 in_binary, uint16 *unpacked_bcd, uint8 *packed_bcd) {
#pragma HLS INTERFACE ap_none port=in_binary
#pragma HLS INTERFACE ap_none port=unpacked_bcd
#pragma HLS INTERFACE ap_none port=packed_bcd
#pragma HLS INTERFACE ap_ctrl_none port=return

    uint16 scratch_pad = in_binary;
    uint4  zero_4      = 0b0000;


    double_dabble(scratch_pad);
    double_dabble(scratch_pad);
    double_dabble(scratch_pad);
    double_dabble(scratch_pad);

    double_dabble(scratch_pad);
    double_dabble(scratch_pad);
    double_dabble(scratch_pad);
    scratch_pad = scratch_pad << 1;


    *packed_bcd   = scratch_pad(15, 8);
    *unpacked_bcd = (zero_4, scratch_pad(15, 12), zero_4, scratch_pad(11, 8));

}
```

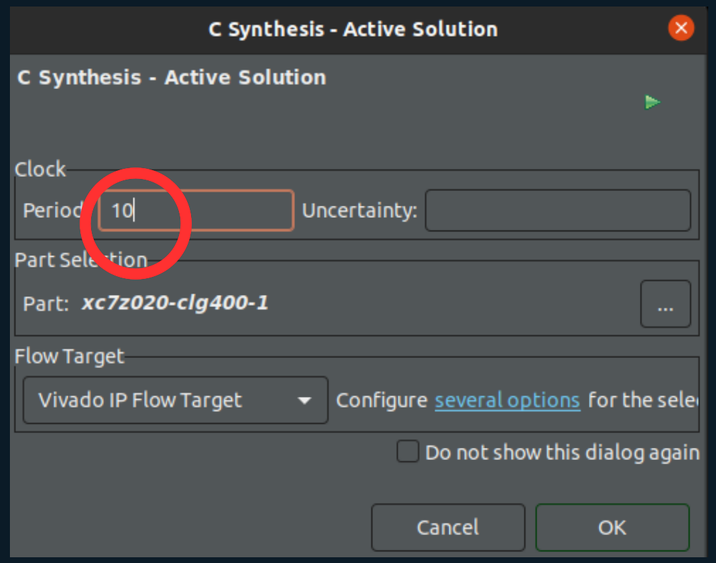
Which do the exact algorithm just mentioned!

Testbench

```cpp
#include "binary2bcd_double_dabble-tb.h"
#include <iostream>



int main() {
    int status = 0;

    uint8  in_binary = 47;
    uint16 unpacked_bcd;
    uint8 packed_bcd;



    binary2bcd_double_dabble(in_binary, &unpacked_bcd, &packed_bcd);


    std::cout << " in_binary = " << in_binary.to_string() << " unpacked_bcd = " << unpacked_bcd.to_string() << " packed_bcd = " << packed_bcd.to_string() << std::endl;


    return status;
}
```

Synthesis

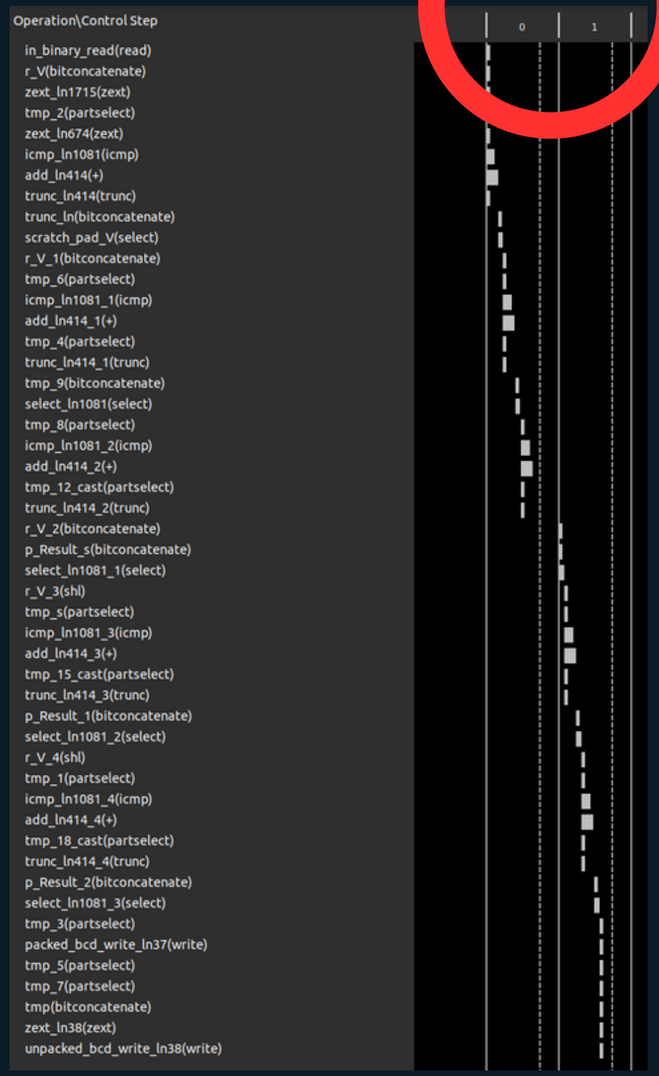
Problem: Why there are FFs?



The clock period is too short.
It takes more than 1 clock cycle to finish the task.
Therefore, there are flip-flops used

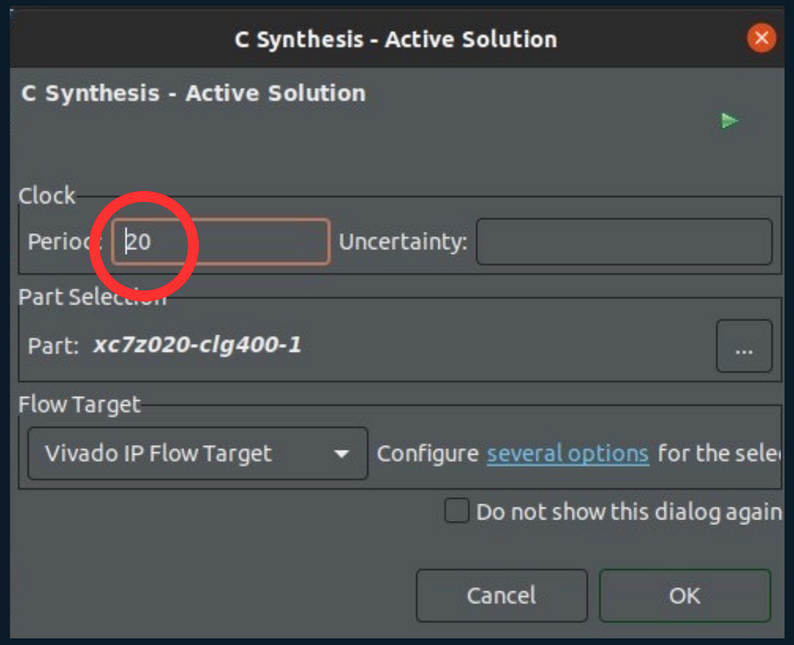
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
binary2bcd_double_dabble				-	1	10.000	-	2	-	no	0	0	21	96	0

Schedule view



Synthesis

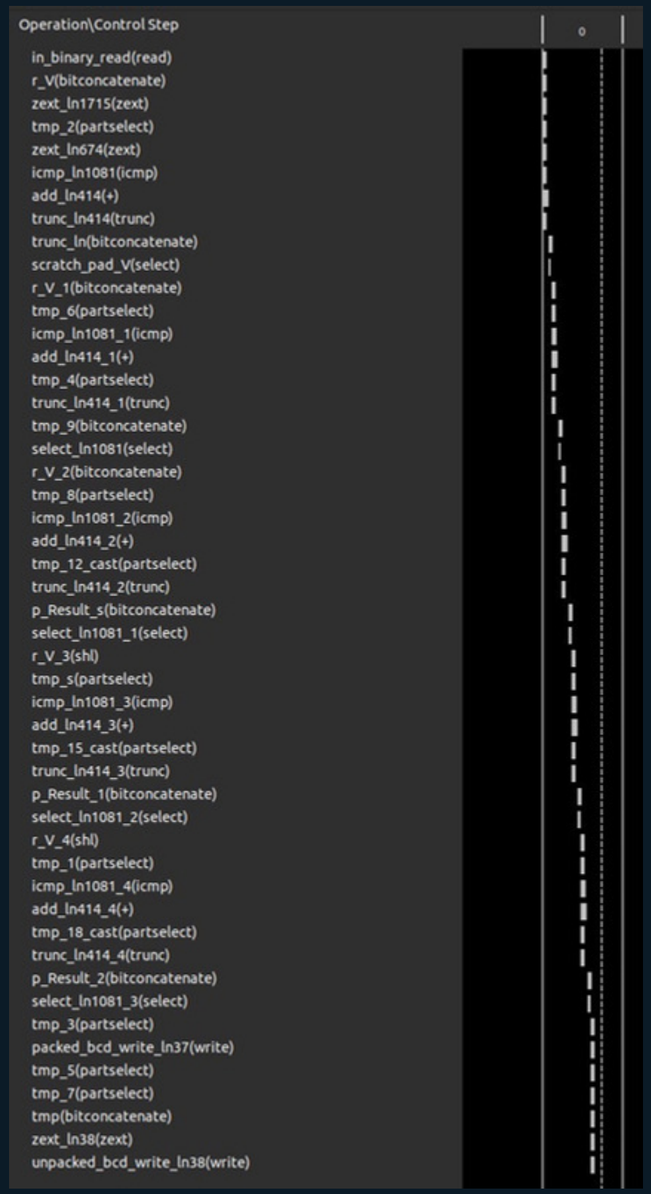
Solution: Increase the clock period



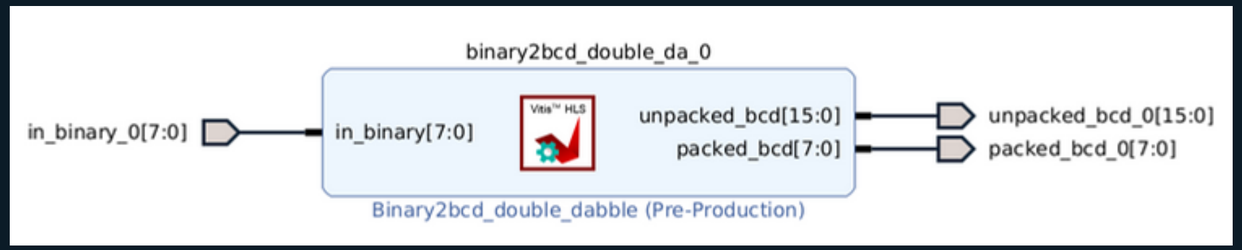
Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
binary2bcd_double_dabble				-	0	0.0	-	1	-	no	0	0	0	182	0

Only combinational logics!

Schedule view



Vivado



The imported RTL IP from HLS

verilog code (manual)

```
module double_dabble(
    input [7:0] in_binary,
    output wire [15:0] unpacked_bcd,
    output wire [7:0] packed_bcd
    );

    reg [15:0] shift_0;
    reg [15:0] temp;
    reg  [7:0] add3check;

    always @* begin
        shift_0 = {8'd0, in_binary};
    // 1st shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 2nd shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 3rd shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 4th shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 5th shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 6th shift
        temp = shift_0 * 2'd2;
        add3check= temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 7th shift
        temp = shift_0 * 2'd2;
        add3check = temp[15:8];
        if(add3check[3:0] > 3'd4) begin
            add3check = add3check + 3'd3;
        end
        shift_0[15:8] = add3check;
        shift_0[7:0] = temp[7:0];
    // 8th shift
        shift_0 = shift_0 * 2'd2;
    end

    assign packed_bcd = shift_0[15:8];
    assign unpacked_bcd[15:12] = 4'b0;
    assign unpacked_bcd[11:8] = shift_0[15:12];
    assign unpacked_bcd[7:4] = 4'b0;
    assign unpacked_bcd[3:0] = shift_0[11:8];

    endmodule
```

one iteration

```
// 1st shift
    temp = shift_0 * 2'd2;
    add3check = temp[15:8];
    if(add3check[3:0] > 3'd4) begin
        add3check = add3check + 3'd3;
    end
    shift_0[15:8] = add3check;
    shift_0[7:0] = temp[7:0];
```

Vivado - testbench

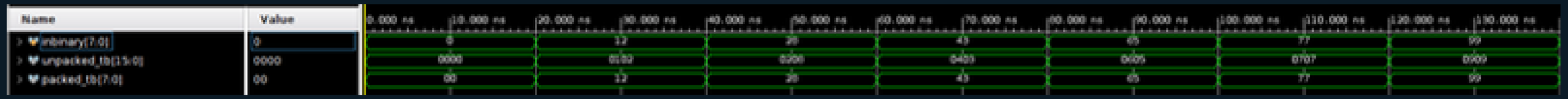
```
module double_dabble_tb( );
    reg [7:0] inbinary;
    wire [15:0] unpacked_tb;
    wire [7:0] packed_tb;

    design_1_wrapper U0 (.in_binary_0(inbinary), .packed_bcd_0(packed_tb), .unpacked_bcd_0(unpacked_tb));

    initial
    begin
    #0      inbinary = 8'd0;
    #20     inbinary = 8'd12;
    #20     inbinary = 8'd28;
    #20     inbinary = 8'd43;
    #20     inbinary = 8'd65;
    #20     inbinary = 8'd77;
    #20     inbinary = 8'd99;
    #20     inbinary = 8'd0;
    end

endmodule
```

Testbench for IP

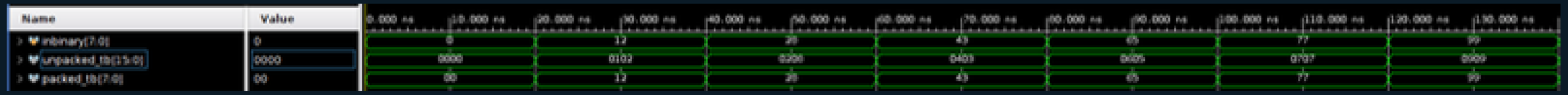
```
module double_dabble_tb( );
    reg [7:0] inbinary;
    wire [15:0] unpacked_tb;
    wire [7:0] packed_tb;

    double_dabble U0 (.in_binary(inbinary), .unpacked_bcd(unpacked_tb), .packed_bcd(packed_tb));

    initial
    begin
    #0          inbinary = 8'd0;
    #20         inbinary = 8'd12;
    #20         inbinary = 8'd28;
    #20         inbinary = 8'd43;
    #20         inbinary = 8'd65;
    #20         inbinary = 8'd77;
    #20         inbinary = 8'd99;
    #20         inbinary = 8'd0;
    end

endmodule
```

Testbench for
manual verilog design

Vivado - waveform



wave form of IP



wave form of manual verilog design

The same!

Vivado - testbench for all cases

Testbench for IP and manual verilog design

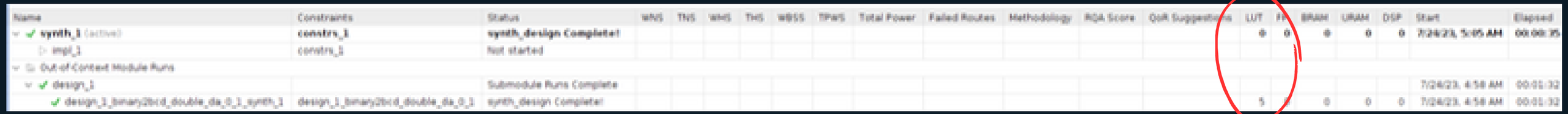
```
module double_dabble_tb( );
    reg [7:0] inbinary;
    wire [15:0] unpacked_hw;
    wire [7:0] packed_hw;
    reg [15:0] unpacked_sw;
    reg [7:0] packed_sw;

    double_dabble UO (.in_binary(inbinary), .unpacked_bcd(unpacked_hw), .packed_bcd(packed_hw));

    reg [15:0] temp = 16'd0;

    integer i, j;

    initial begin
        for ( i = 0; i < 100; i = i + 1) begin
            inbinay = temp;
            double_dadabble_sw(temp, unpacked_sw, packed_sw);
            #5
            if (unpacked_hw != unpacked_sw) begin
                $display("Error at %b, unpacked_bcd = %b, unpacked_sw = %b", temp, unpacked_hw, unpacked_sw);
                $display("Error at %b, packed_bcd = %b, packed_sw = %b", temp, unpacked_hw, unpacked_sw);
            end
            else begin
                $display("%b, unpacked_bcd = %b, unpacked_sw = %b", temp, unpacked_hw, unpacked_sw);
                $display("%b, packed_bcd = %b, packed_sw = %b", temp, unpacked_hw, unpacked_sw);
            end
            #5 temp = temp + 1'b1;
        end
    end


    task double_dabble_sw;
        input [7:0] a;
        output [15:0] unpacked_sw;
        output [7:0] packed_sw;
        reg [15:0] shift;
        reg [15:0] tmp;
        reg  [7:0] add3check;
        begin
            unpacked_sw = 16'b0;
            packed_sw = 8'd0;
            shift = {8'd0, a};
            for (j = 0; j < 8; j = j + 1) begin
                temp = shift * 2'd2;
                add3check = tmp[15:0];
                if(add3check[3:0] > 4) begin
                    add3check = add3check + 3'd3;
                end
                shift[15:8] = add3check;
                shift[7:0] = tmp[7:0];
            end
            shift = shift * 2'd2;
            packed_sw = shift[15:8];
            unpacked_sw[15:12] = 4'd0;
            unpacked_sw[11:8] = shift[15:12];
            unpacked_sw[7:4] = 4'd0;
            unpacked_sw[3:0] = shift[11:8];
        end
    endtask
    endmodule
```

Vivado simulation

```
correct 00000000, unpacked_bcd = 0000000000000000, unpacked_sw = 0000000000000000
correct 00000000, packed_bcd = 00000000, packed_sw = 00000000
correct 00000001, unpacked_bcd = 0000000000000001, unpacked_sw = 0000000000000001
correct 00000001, packed_bcd = 00000001, packed_sw = 00000001
correct 00000010, unpacked_bcd = 0000000000000010, unpacked_sw = 0000000000000010
correct 00000010, packed_bcd = 00000010, packed_sw = 00000010
correct 00000011, unpacked_bcd = 0000000000000011, unpacked_sw = 0000000000000011
correct 00000011, packed_bcd = 00000011, packed_sw = 00000011
correct 00000100, unpacked_bcd = 0000000000000100, unpacked_sw = 0000000000000100
correct 00000100, packed_bcd = 00000100, packed_sw = 00000100
correct 00000101, unpacked_bcd = 0000000000000101, unpacked_sw = 0000000000000101
correct 00000101, packed_bcd = 00000101, packed_sw = 00000101
correct 00000110, unpacked_bcd = 0000000000000110, unpacked_sw = 0000000000000110
correct 00000110, packed_bcd = 00000110, packed_sw = 00000110
correct 00000111, unpacked_bcd = 0000000000000111, unpacked_sw = 0000000000000111
correct 00000111, packed_bcd = 00000111, packed_sw = 00000111
correct 00001000, unpacked_bcd = 0000000000001000, unpacked_sw = 0000000000001000
correct 00001000, packed_bcd = 00001000, packed_sw = 00001000
correct 00001001, unpacked_bcd = 0000000000001001, unpacked_sw = 0000000000001001
correct 00001001, packed_bcd = 00001001, packed_sw = 00001001
correct 00001010, unpacked_bcd = 0000000100000000, unpacked_sw = 0000000100000000
correct 00001010, packed_bcd = 00010000, packed_sw = 00010000
correct 00001011, unpacked_bcd = 0000000100000001, unpacked_sw = 0000000100000001
correct 00001011, packed_bcd = 00010001, packed_sw = 00010001
correct 00001100, unpacked_bcd = 0000000100000010, unpacked_sw = 0000000100000010
correct 00001100, packed_bcd = 00010010, packed_sw = 00010010
correct 00001101, unpacked_bcd = 0000000100000011, unpacked_sw = 0000000100000011
correct 00001101, packed_bcd = 00010011, packed_sw = 00010011
correct 00001110, unpacked_bcd = 0000000100000100, unpacked_sw = 0000000100000100
correct 00001110, packed_bcd = 00010100, packed_sw = 00010100
```

The results of the hardware and software same!

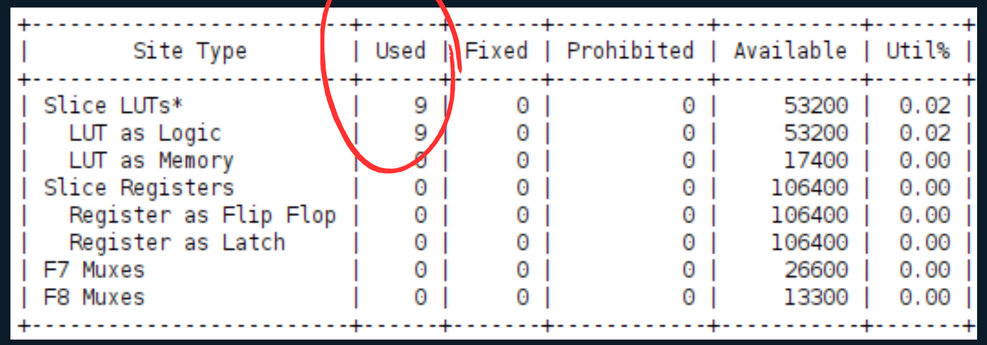
Vivado - Resources



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	7/24/23, 5:05 AM	00:00:35
impl_1	constrs_1	Not started																		
Out-of-Context Module Runs																				
design_1		Submodule Runs Complete																	7/24/23, 4:58 AM	00:01:32
design_1_binary2bcd_double_da_0_1_synth_1	design_1_binary2bcd_double_da_0_1	synth_design Complete!												5	0	0	0	0	7/24/23, 4:58 AM	00:01:32

resources of IP

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	constrs_1	synth_design Complete!												9	0	0	0	0	7/23/2	00:00:29
impl_1	constrs_1	Not started																		



Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	9	0	0	53200	0.02
LUT as Logic	9	0	0	53200	0.02
LUT as Memory	0	0	0	17400	0.00
Slice Registers	0	0	0	106400	0.00
Register as Flip Flop	0	0	0	106400	0.00
Register as Latch	0	0	0	106400	0.00
F7 Muxes	0	0	0	26600	0.00
F8 Muxes	0	0	0	13300	0.00

resources of verilog design

Thank You!