

Design a digital circuit using HLS for
CRC bits generator.

K.Vijaya Lakshmi

Duration : 15-06-23 to 28-06-23

Table of Contents

Table of Contents.....	2
Problem Statement.....	3
🎯 Requirements.....	3
👛 Considerations.....	3
🚚 Deliverables.....	3
Solution Approach.....	4
Code Implementation :.....	5
Test Procedures.....	8
For testing the output of CRC generator ,I used.....	8
HLS Simulation Results.....	8
Test_case 1.....	8
Test_case 2.....	9
Test_Case 3.....	10
RTL simulation.....	11
Block Diagram.....	11
Input image.....	11
Output Image.....	12
HLS Synthesis Report.....	13
Timing Analysis.....	13
Resource Utilization.....	14
Conclusion.....	15

Problem Statement

Design a digital circuit using RTL and HLS for a 5G NR CRC bits generator.

Requirements

- Refer to the 5.1 section in this [document](#) for the mathematical expressions of the CRC bits generation.
- The module should process 8 bits per clock cycle.
- The input and output buses should use the AXIS interface.
- Implement a pipelined design.
- The implementation should use as minimum resources as possible.

Considerations

- Implement the module only for CRC24A
- A last signal should be used to indicate the end of the input bit stream.
- The design implementation should be targeted on the ZCU111 FPGA board.

Deliverables

- Source code implementation of the design.
- Simulation results.
- Resource utilization report analysis.
- Timing report which contains the latency, interval, throughput and the Fmax of the circuit.
- Documentation detailing the design choices, and performance evaluation.

Solution Approach

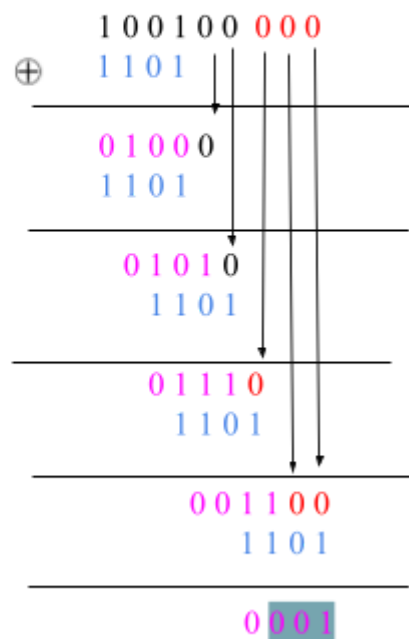
General Crc calculation

Example :

Input bits 1 0 0 1 0 0

Gen_Pol 1 1 0 1

Append 3 zeros after the last bit of input



Crc_bits : 0 0 1

Code Implementation :

I used case statements to implement the design .

Each state is explained below with an example .

State_1 :

- First take 8 bits from the input stream 1 0 0 1 1 0 1 0 and do xor operation .

Inputstream : {1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,}

↓
(LSB)
↓
t_last

Gen_Pol : 1 1 0 1

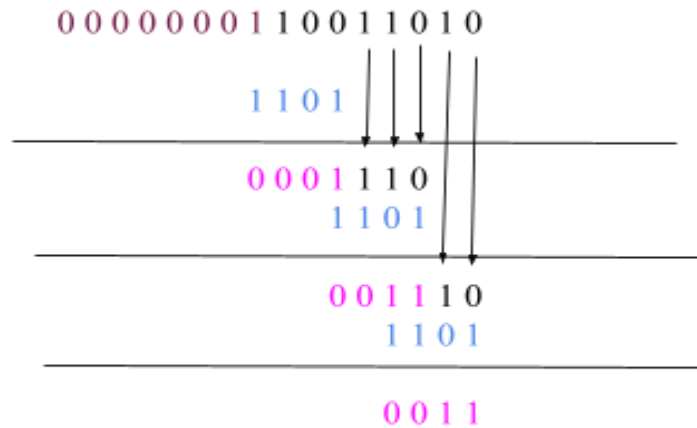
- First take 8 bits from the input stream 1 0 0 1 1 0 1 0 and do xor operation .

1 0 0 1 1 0 0	
1 1 0 1	
0 1 0 0 1	
1 1 0 1	
0 1 0 0 0	
1 1 0 1	
0 1 0 1 0	
1 1 0 1	
0 1 1 1	

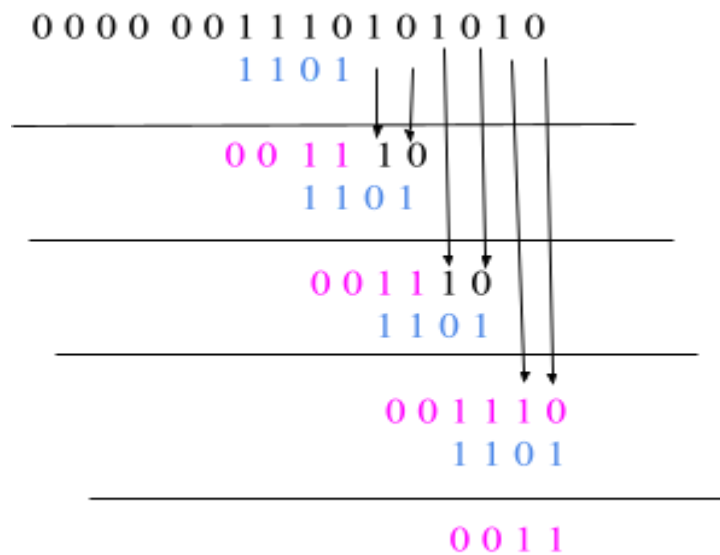
State_2 :

In this case we can expect a new set of data and we do the xor operation .

- Now we take another set of input to the MSB side



- We still have some input bits left in the input stream , so read the data and append to MSB side and the previous result is shifted to LSB .



-

https://github.com/vijayakannamaneni/HLS_Assignments/blob/main/crc_gen/src/crc_gen.cpp

Test Procedures

For testing the output of CRC generator ,I used

- HLS Simulation using testbench
- RTL simulation
- Matlab

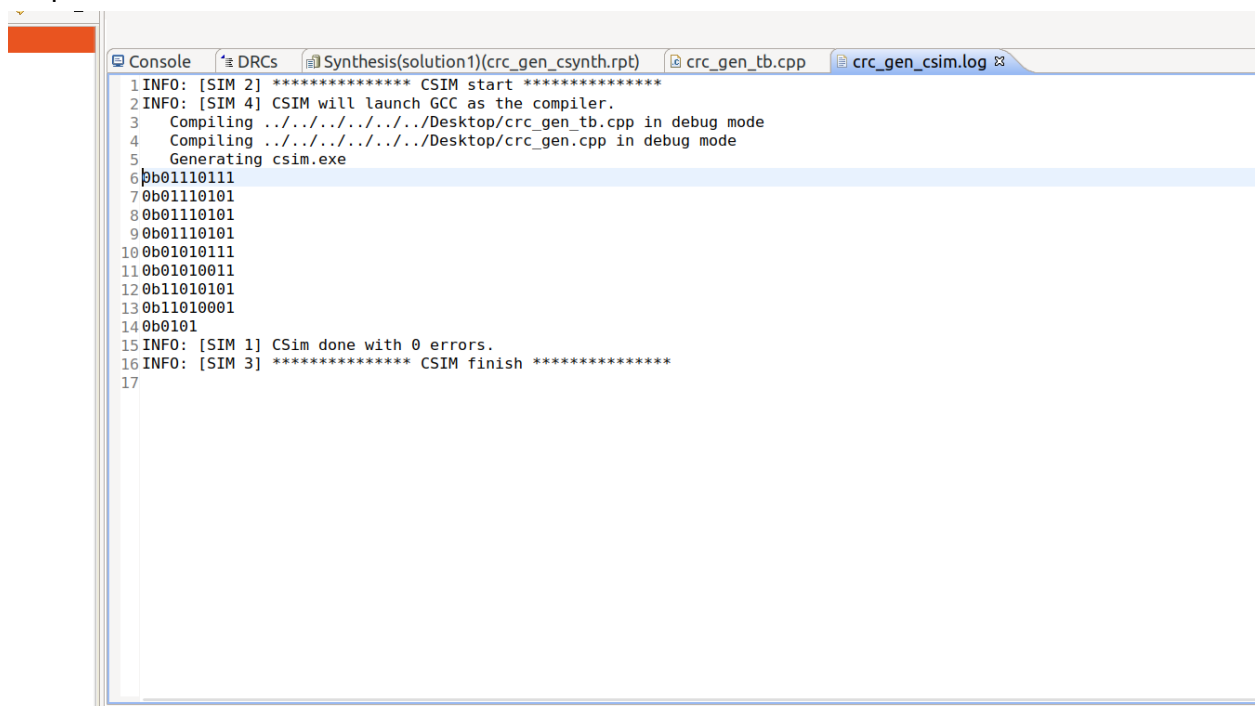
HLS Simulation Results

Test_case 1

Input :

{1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,1,0,1,0,1,0,1,0,1,0,1,1,1,0,0,0,1,0,1,1 }

Output :



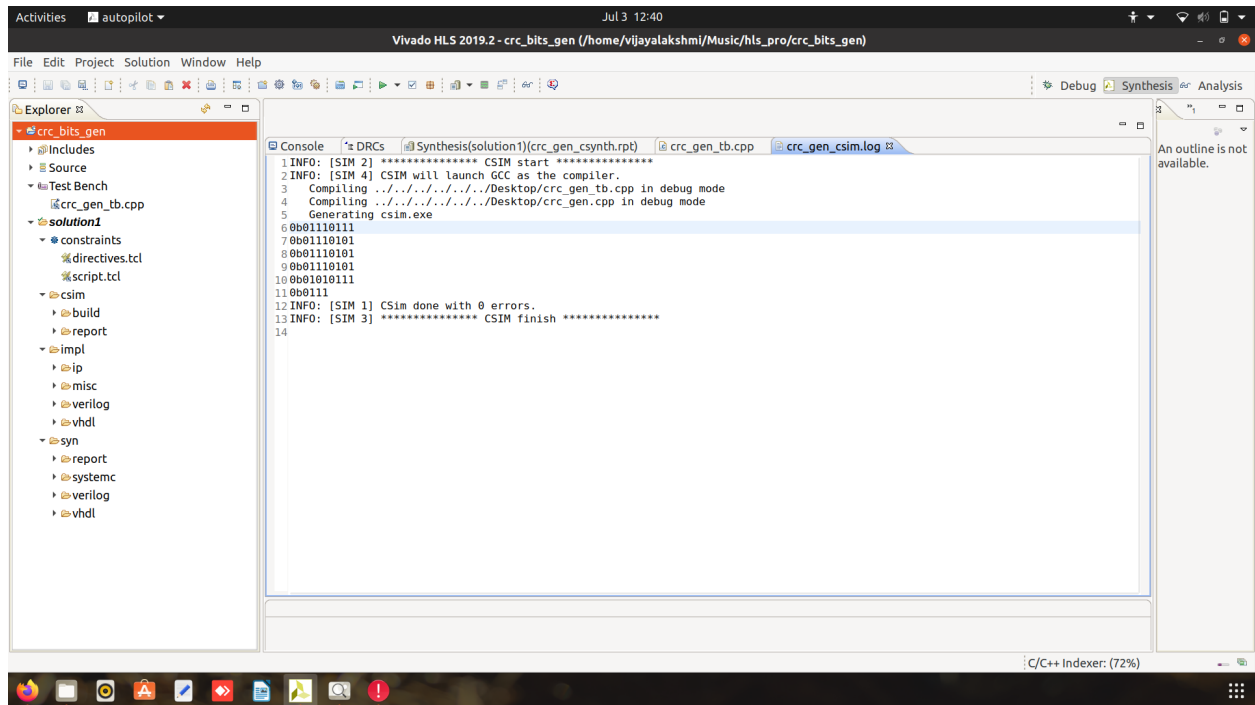
```
1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3   Compiling ../../../../../../Desktop/crc_gen_tb.cpp in debug mode
4   Compiling ../../../../../../Desktop/crc_gen.cpp in debug mode
5   Generating csim.exe
6 6b0111011
7 70b011101
8 80b011101
9 90b011101
10 10b010101
11 11b010101
12 12b110101
13 13b1101001
14 14b0101
15 INFO: [SIM 1] CSim done with 0 errors.
16 INFO: [SIM 3] ***** CSIM finish *****
17
```


Test_case 2

Input :

{1,1,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,0,1,0,1,0 };

Output :

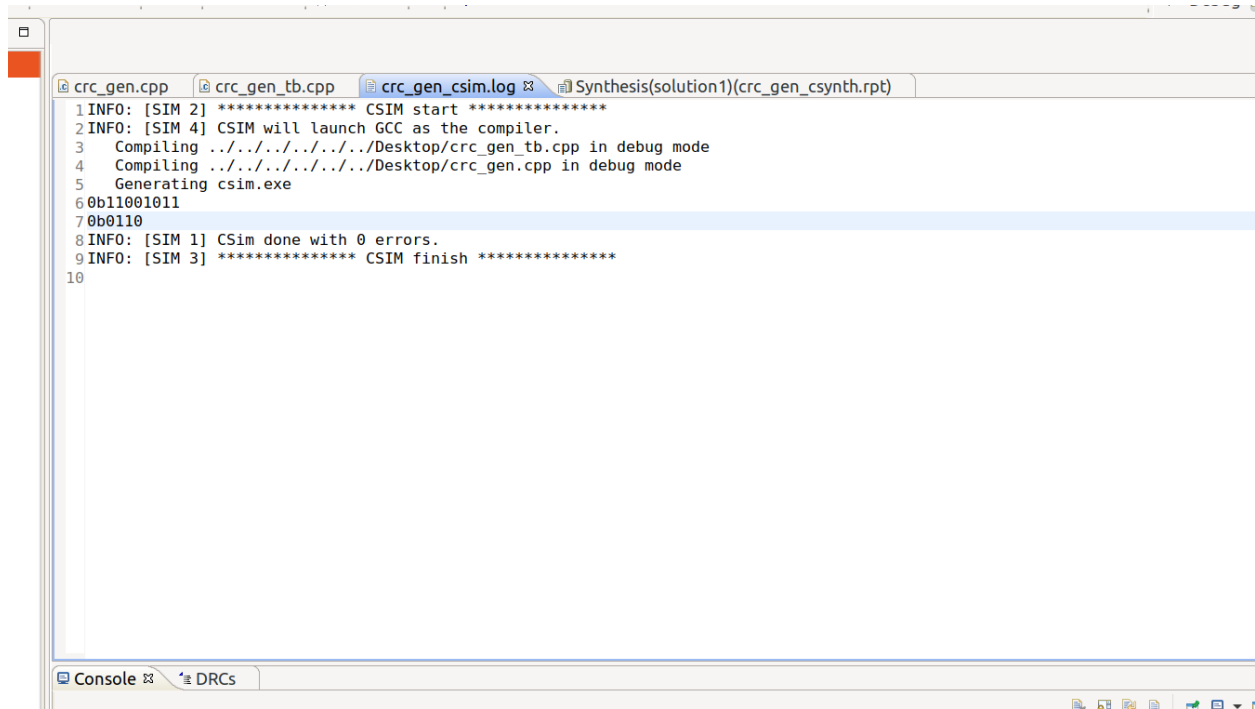


Test_Case 3

Input :

{1,1,0,1,0,0,1,1};

Output :



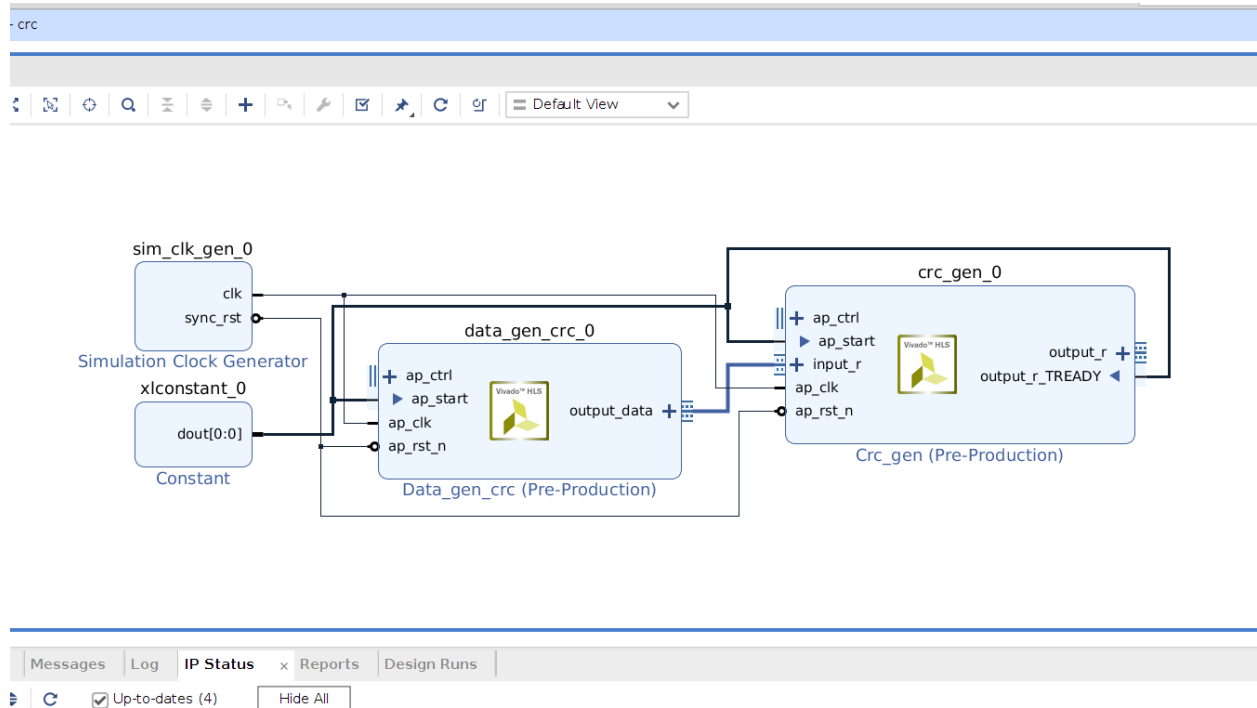
The screenshot shows a software interface with a log window. The log window has four tabs: 'crc_gen.cpp', 'crc_gen_tb.cpp', 'crc_gen_csim.log', and 'Synthesis(solution1)(crc_gen_csynth.rpt)'. The 'crc_gen_csim.log' tab is active, displaying the following text:

```
1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3   Compiling ../../../../../../Desktop/crc_gen_tb.cpp in debug mode
4   Compiling ../../../../../../Desktop/crc_gen.cpp in debug mode
5   Generating csim.exe
6 0b11001011
7 0b0110
8 INFO: [SIM 1] CSim done with 0 errors.
9 INFO: [SIM 3] ***** CSIM finish *****
10
```

At the bottom of the interface, there is a 'Console' tab and a 'DRCs' tab. The 'Console' tab is active, and the 'DRCs' tab is inactive.

RTL simulation

Block Diagram



Input image

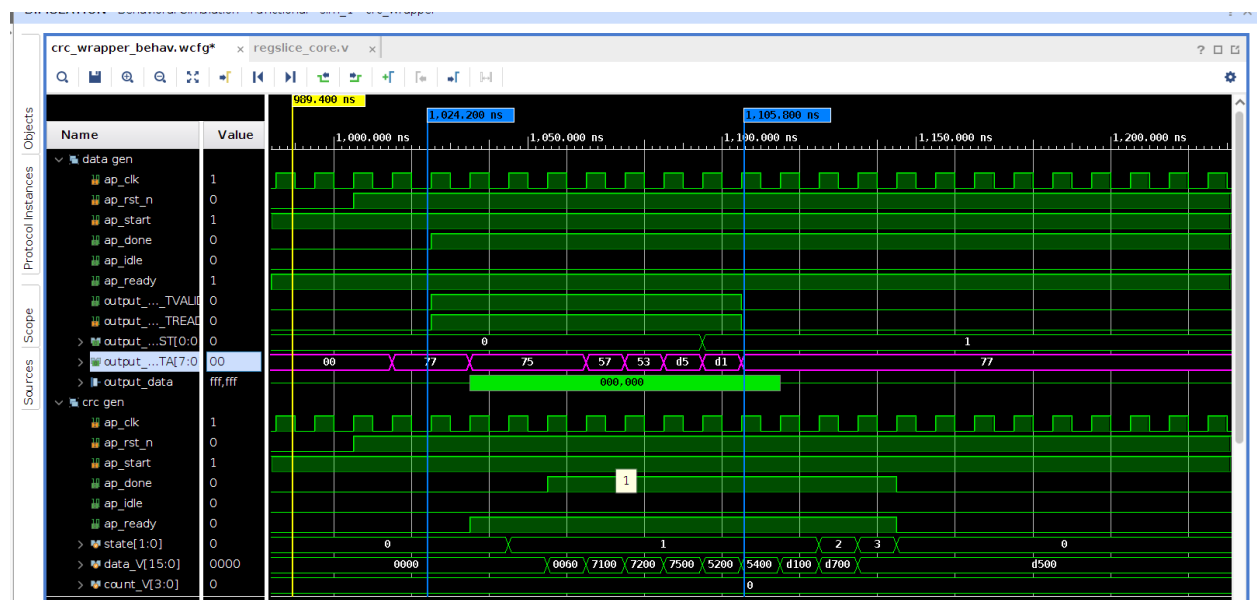


Figure : Input_image (between two blue markers highlighted in magenta color)

Output Image

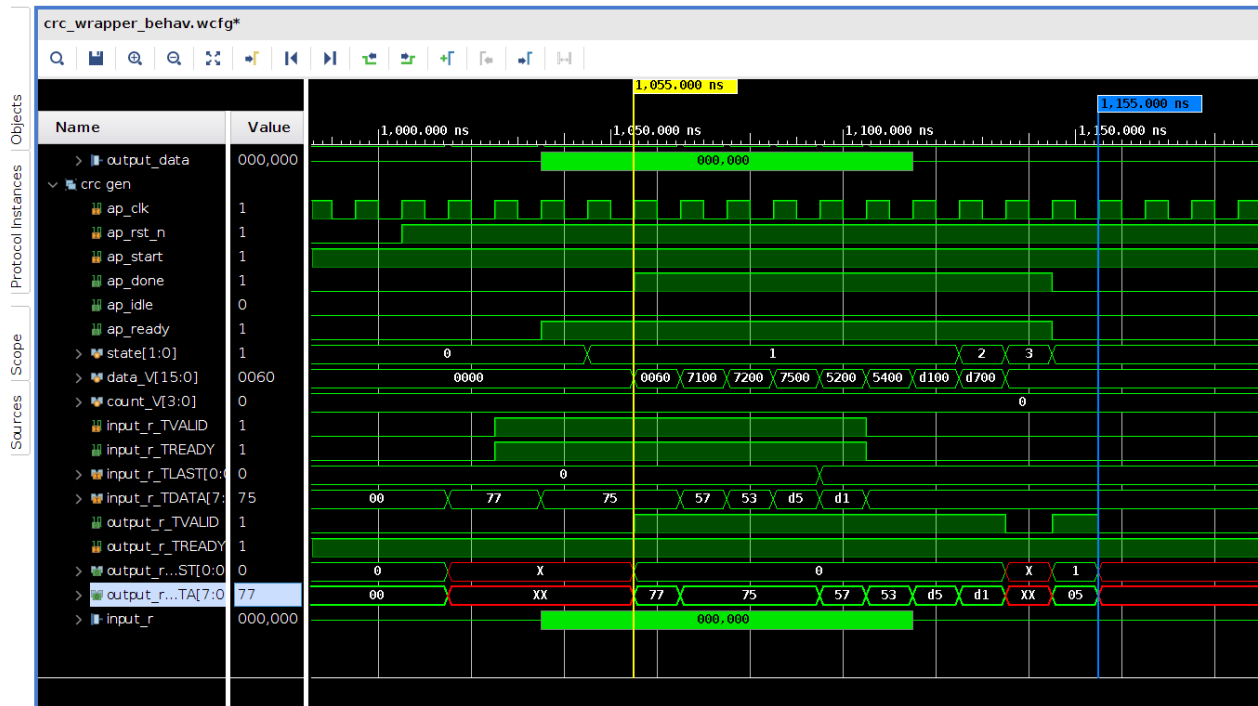
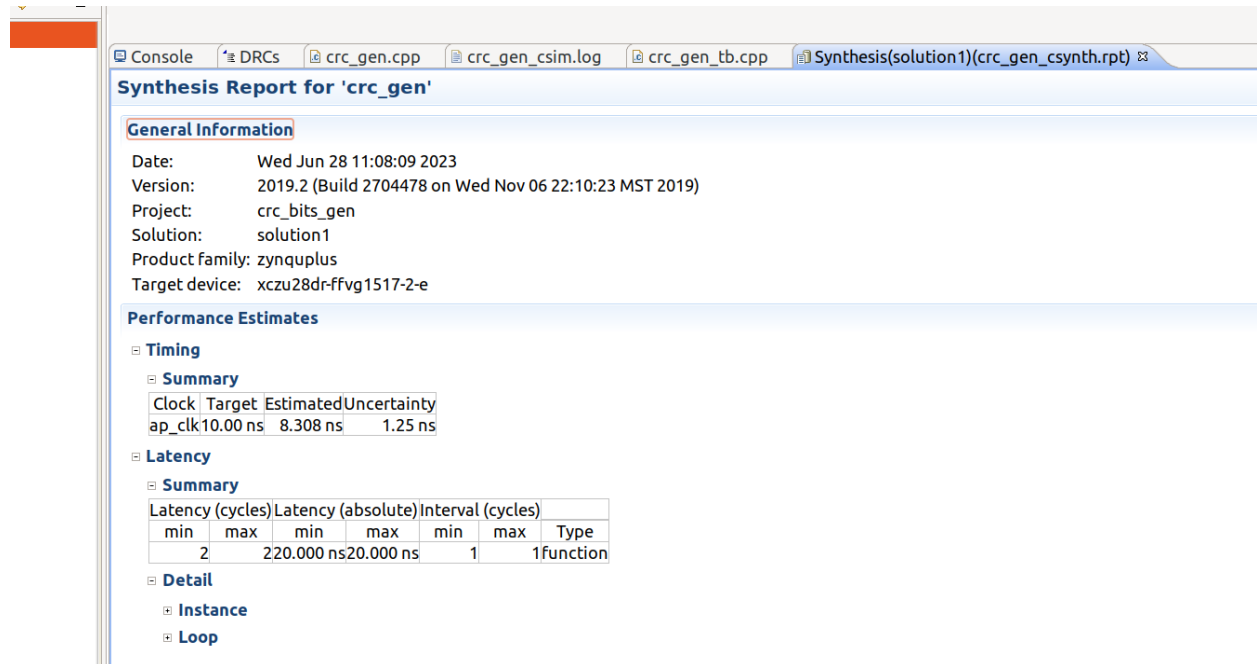


Figure : Output image (from yellow to blue marker showing output # (highlighted Left))

HLS Synthesis Report

Timing Analysis



The screenshot displays the Xilinx IDE interface with the 'Synthesis Report for 'crc_gen'' open. The report is divided into sections: General Information, Performance Estimates, Timing, Latency, and Detail. The Timing section shows a summary table for clock targets, and the Latency section shows a summary table for latency and interval.

Synthesis Report for 'crc_gen'

General Information

Date: Wed Jun 28 11:08:09 2023
Version: 2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)
Project: crc_bits_gen
Solution: solution1
Product family: zynqplus
Target device: xczu28dr-ffvg1517-2-e

Performance Estimates

Timing

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00 ns	8.308 ns	1.25 ns

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		Type
min	max	min	max	min	max	
2	220.000	ns	20.000 ns	1	1	function

Detail

Instance

Loop

Resource Utilization

ConsoleDRCscrc_gen.cppcrc_gen_csim.logcrc_gen_tb.cppSynthesis(solution1)(crc_gen_csynth.rpt)

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18KDSP48E	FF	LUT	URAM
DSP	-	-	-	-
Expression	-	-	0	2154
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	240
Register	-	-	152	-
Total	0	0	152	2394
Available	2160	4272850560	425280	80
Utilization (%)	0	0	~0	~0

Detail

Instance

DSP48E

Memory

FIFO

Expression

Multiplexer

Register

Conclusion

- The module is processing 8 bits per clock cycle .
- The input and output buses are using the AXIS interface .
- The design is pipelined .