# Design a digital circuit using HLS for CRC bits generator.

K.Vijaya Lakshmi

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

# Table of Contents

# 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

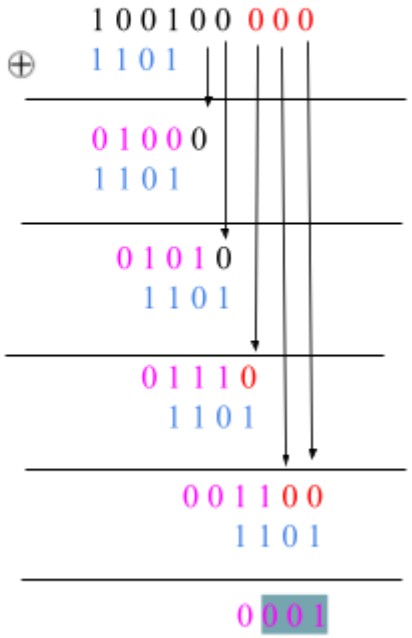
- Source code implementation of the design.
- Simulation results.
- Resource utilization report analysis.
- Timing report which contains the latency, interval, throughput and the Fmax of the circuit.
- Documentation detailing the design choices, and performance evaluation.

# 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 .

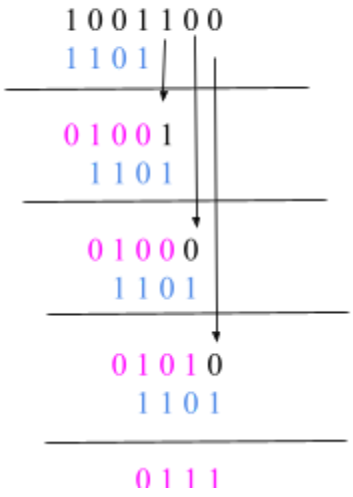
Inputstream : {1,0,0,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,}

(LSB )

t_last

Gen_Pol : 1 1 0 1

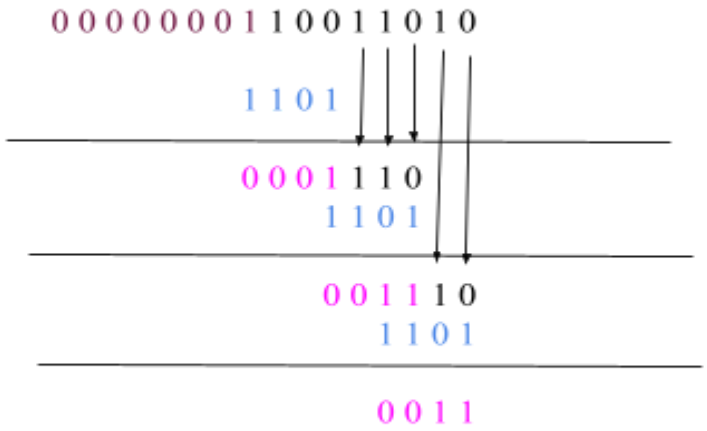
- First take 8 bits from the input stream 1 0 0 1 1 0 1 0 and do xor operation .



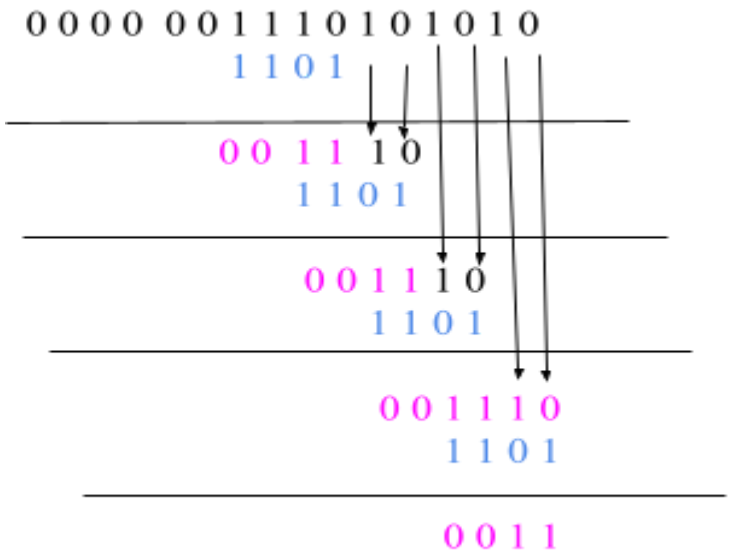
**State_2 :**

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

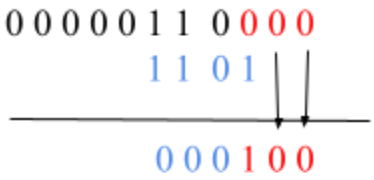
- Now we take another set of input to the MSB side




- 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 .



- Now we reached last signal so append three zeros ( polynomial length is 4 so 4-1=3 zeros )




CRC bits : 1 0 0

**State_3 :**

- In this state I'm storing the last three bits into a temp variable and writing that to outputstream .
  Note : In each state after reading the input from the inputstream we are writing that to output . ( Because output = input + CRC bits ) .

  So now output is ,
  **Output** : 1 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0

**State_4 :**

- This is the reset state ,after this state it goes to state_1 .

**Code link :**
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

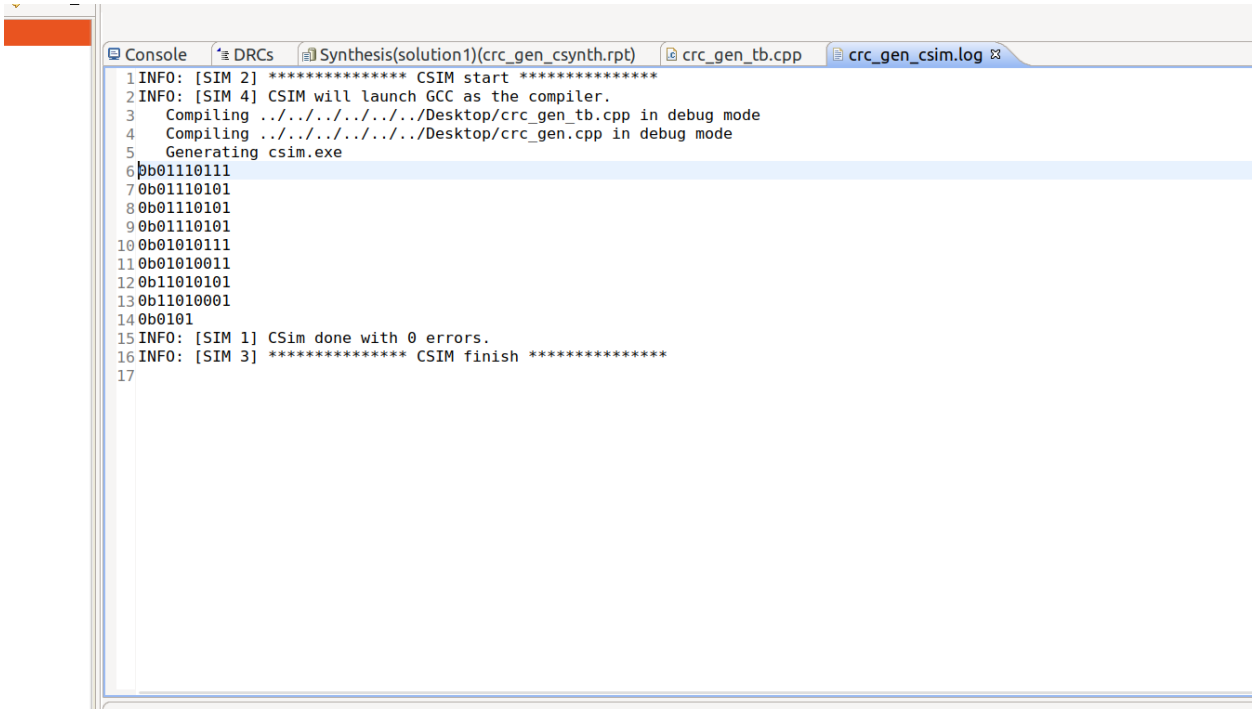
## HLS Simulation Results

## Test_case 1

Input :

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

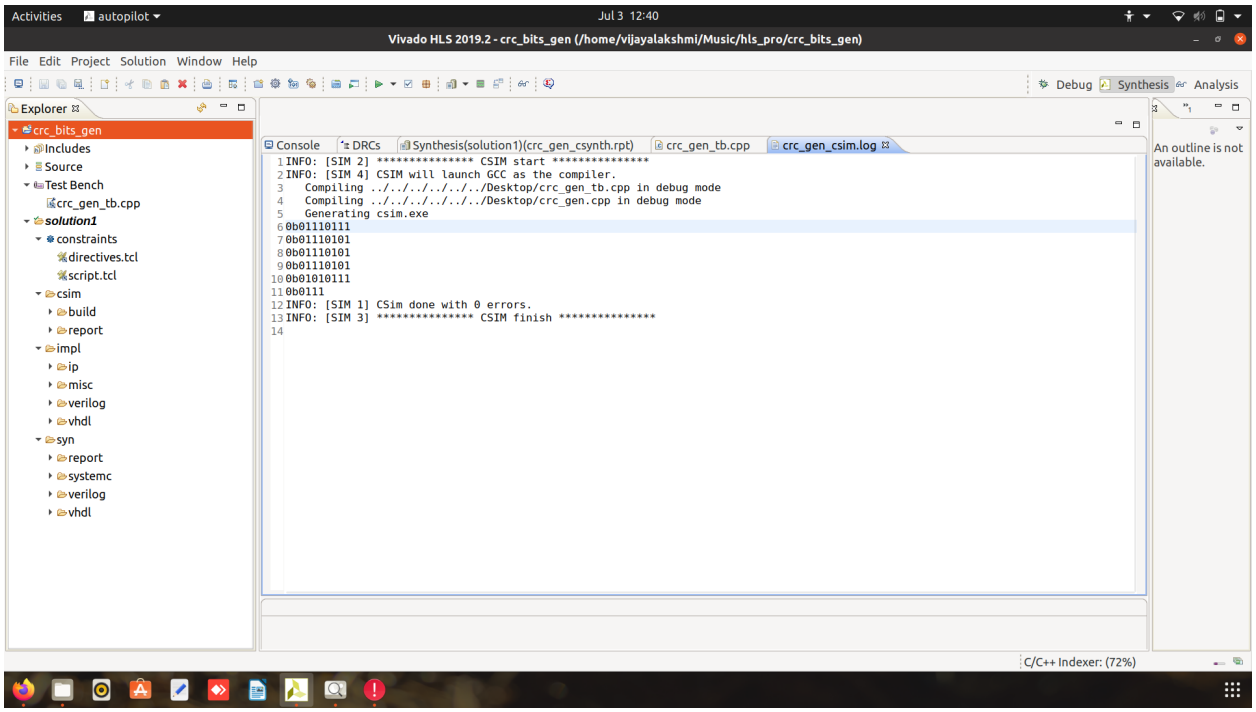
Output :



# Test_case 2

Input :
{1,1,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,0,1,1,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,0,1,0,1,0 };

Output :

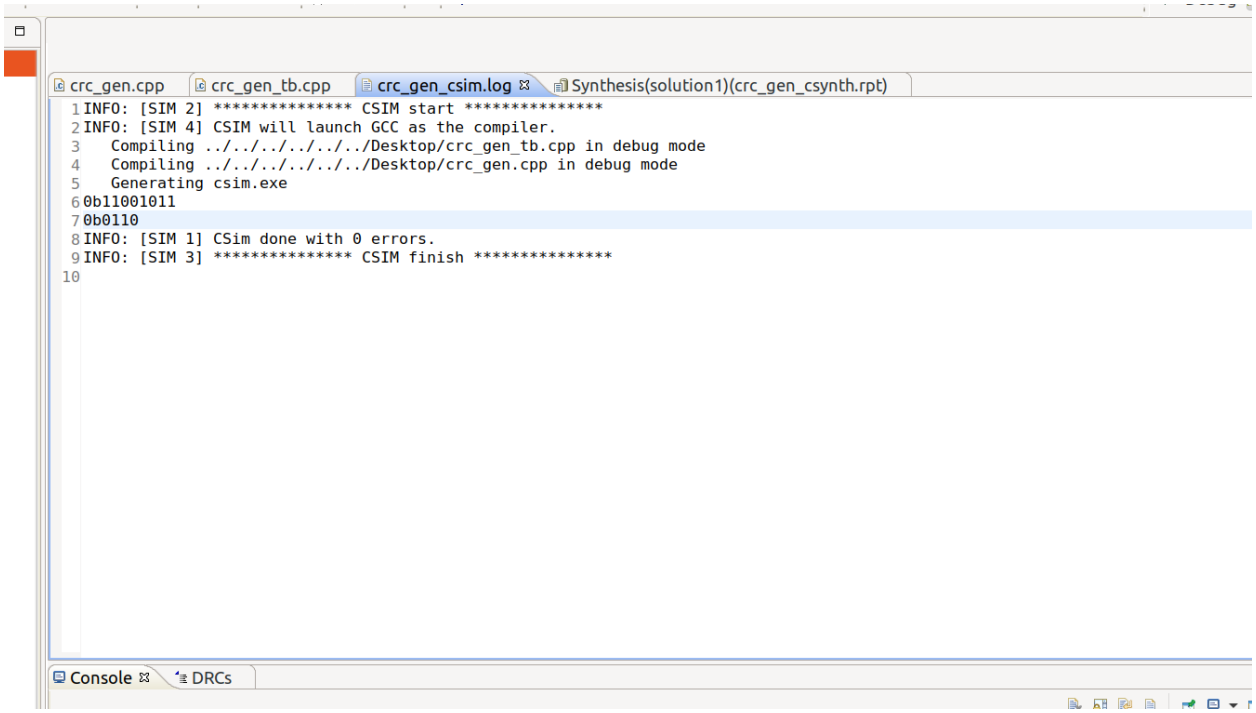


# Test_Case 3

Input :

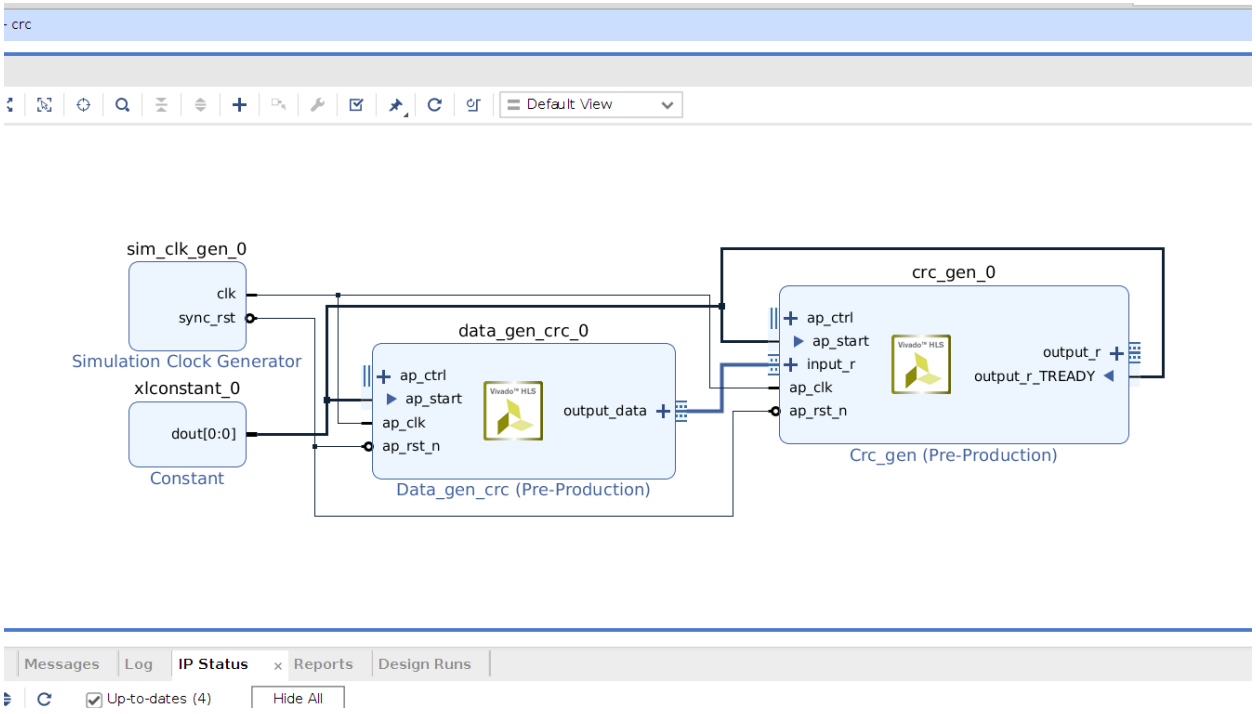
{1,1,0,1,0,0,1,1};

Output :



# RTL simulation

## Block Diagram




## Input image

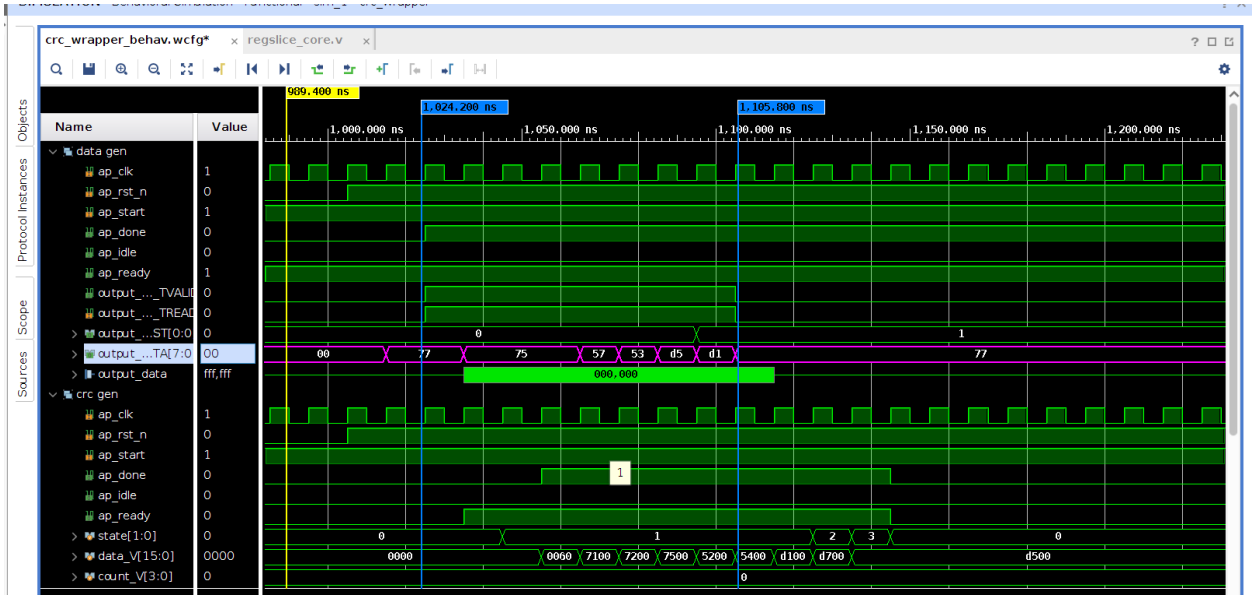



*Figure : Input _image (between two blue markers highlighted in magenta color )*

# Output Image

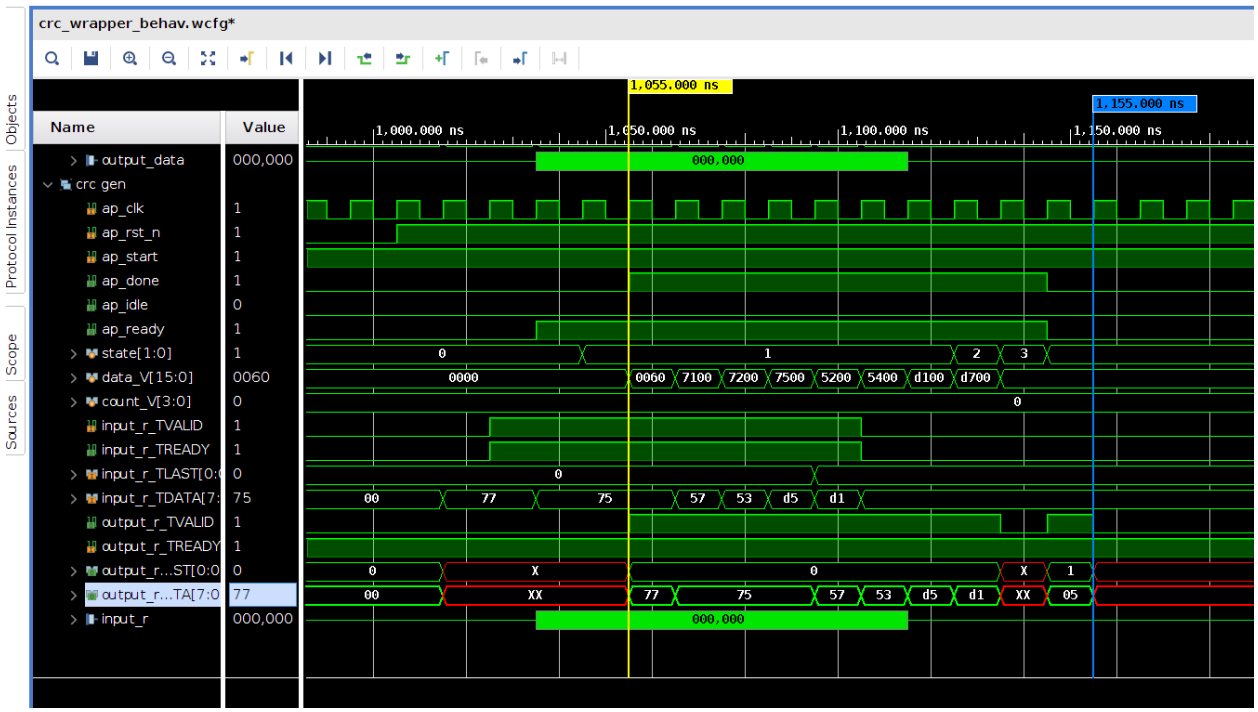
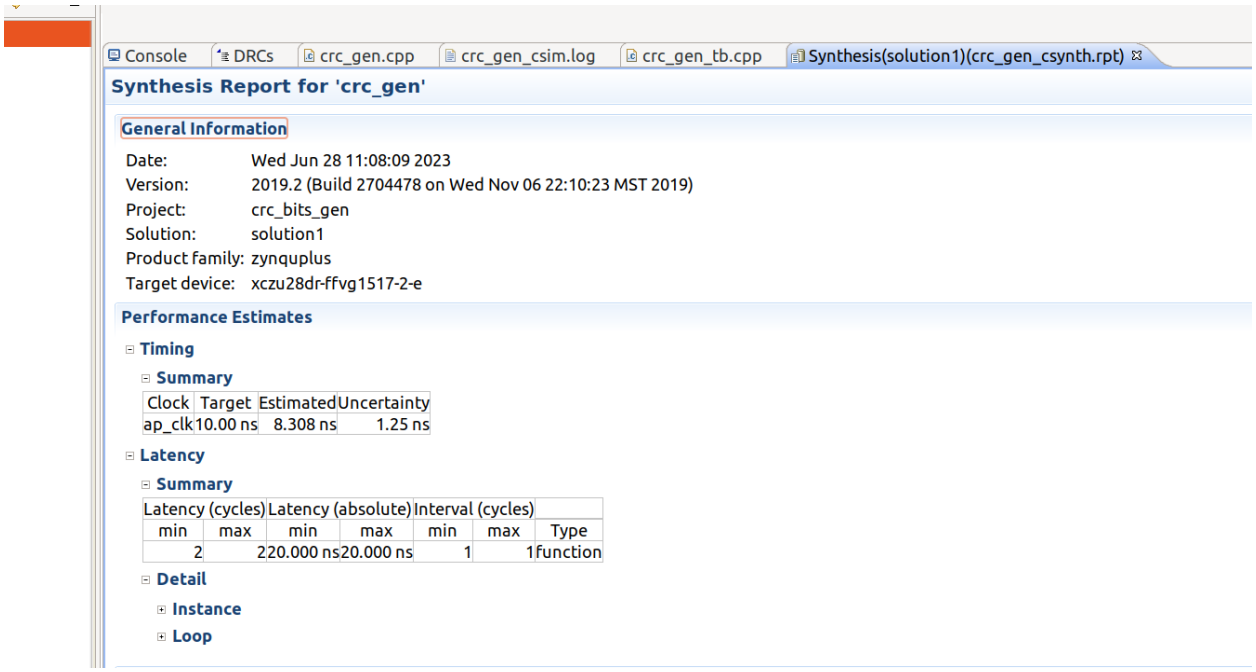



*Figure : Output image (from yellow to blue marker showing output # (highlighted Left))*

# HLS Synthesis Report

## Timing Analysis



Console | DRCs | crc_gen.cpp | crc_gen_csim.log | crc_gen_tb.cpp | Synthesis(solution1)(crc_gen_csynth.rpt)

**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: zynquplus
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**

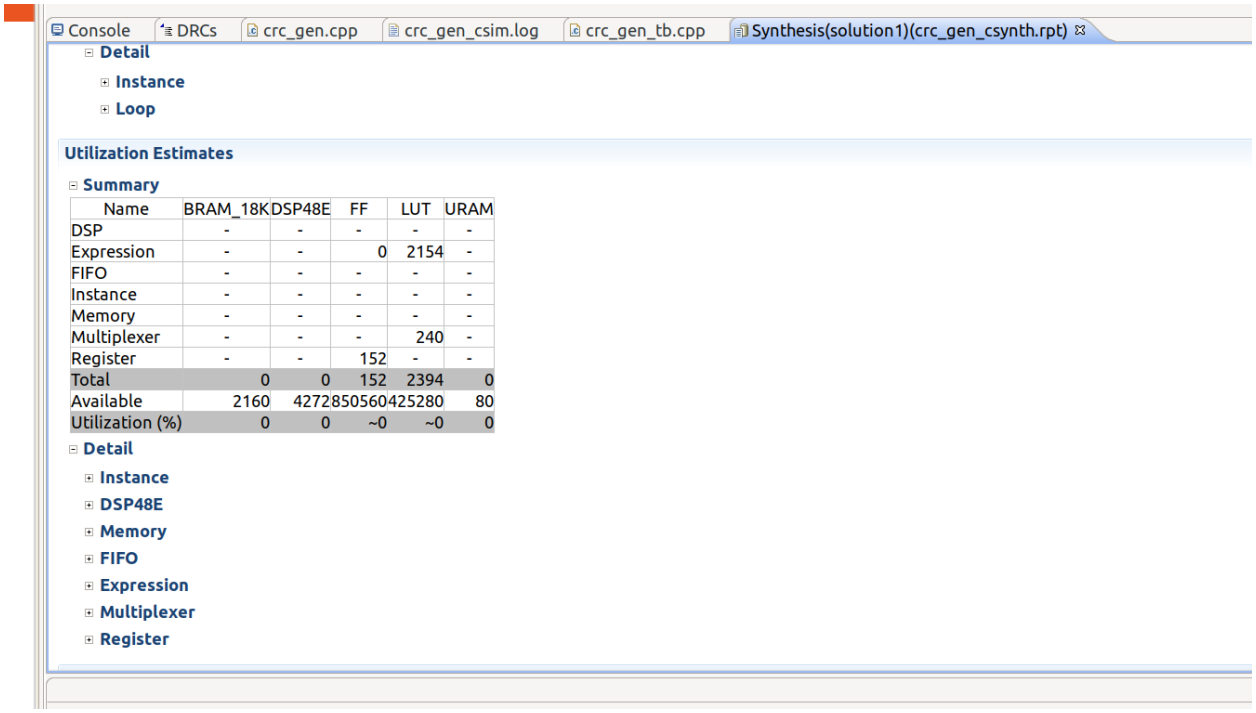
| Latency (cycles) | | Latency (absolute) | | Interval (cycles) | | |
|---|---|---|---|---|---|---|
| min | max | min | max | min | max | Type |
| 2 | 2 | 20.000 ns | 20.000 ns | 1 | 1 | function |

**Detail**

**Instance**

**Loop**

# Resource Utilization



# Conclusion

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