# HLS-Assignment 9 PART-1

July 3, 2023

**Sampath Govardhan**
**FWC22071**

**VITIS-HLS**

## 1 Problem Statement

Problem Statemt

## 2 Header File

```
//header.h
#ifndef _HEADER_H_
#define _HEADER_H_

#include <hls_stream.h>
#include "ap_int.h"
using namespace std;

#define N 8       //length of input message
#define y 25      //length of divisor(parity)
#define x N+y-1   //length of crc (len of input+divisor-1)

typedef ap_uint<N> data;


void crc24a(hls::stream<data>& input, hls::stream<data>& output);

#endif
```

## 3 CRC bits Generator Code

```
//crc.cpp
#include "header.h"

void crc24a(hls::stream<data>& input, hls::stream<data>& output) {

#pragma HLS INTERFACE mode=axis register_mode=both port=input register
#pragma HLS INTERFACE mode=axis register_mode=both port=output register

        ap_uint<1> crc[x],oput[x];
    ap_uint<1> divisor[y] = {1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0,
    1, 1, 1, 1, 0, 1, 1};
    data o1,o2,o3,o4;

// Read input stream and do padding
    data d = input.read();
   loop1: for (int i = 0; i < x; i++) {
#pragma HLS UNROLL
        crc[i] = (i < N) ? d(i,i) : 0;
        oput[i] = (i < N) ? d(i,i) : 0;
    }
   ap_uint<1> last=input.read();

// Division is performed only when last is high
   loop2: for (int i = 0; i <= x - y; i++) {
#pragma HLS PIPELINE II=1
        if (crc[i] == 1 && last==1) {
          loop3:  for (int j = 0; j < y; j++) {
                  int k=i+j;
#pragma HLS UNROLL
                crc[k] = crc[k] ^ divisor[j];
            }
        }
    }

// Write the result to output stream c

   loop4:for (int i = 0; i < x; i++) {
#pragma HLS UNROLL
            oput[i] = crc[i] ^ oput[i];
        if (i < N) {
            o1(i, i) = oput[i];
        }else if (i < N * 2){
            o2(i % N, i % N) = oput[i];
```

```
        } else if (i < N * 3) {
            o3(i % N, i % N) = oput[i];
        } else {
            o4(i % N, i % N) = oput[i];
        }
    }

    output.write(o1);
    output.write(o2);
    output.write(o3);
    output.write(o4);
}
```

## 4 Test Bench Code

```
//crc_tb.cpp
#include "header.h"
#include <vector>
int main() {
    hls::stream<data> a,b;
    data w;
    ap_uint<1> last;

      w=0b00010110;
//msbtolsb

          /* ap_uint<1> dividend[8] = {0, 1, 1, 0, 1, 0, 0, 0};
//lsbtomsb
                for (int i = 0; i < 8; i++) {

                        w(i,i) = dividend[i];

                        }
              */
          last=1;
                a.write(w);
                a.write(last);
```

```
// Perform binary divison
    crc24a(a, b);

// Read the result from the output stream
    vector<ap_uint<1>> p;
    cout << "CRC generator output : ";
    while(!b.empty()){
          data d = b.read();
         for (int i = 0; i < N ; i++) {
                cout<< d(i,i);
                p.push_back(d(i,i));
          }
       }
       cout<<endl;

// Checking if output is valid or not
            bool flag=0;
            ap_uint<1> divisor[y] = {1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
            0, 1, 1, 1, 1, 1, 0, 1, 1};


       //Output is valid only when remainder divison of output with divisor is 0
            for (int i = 0; i <= x - y; i++) {
                if (p[i] == 1) {
                    for (int j = 0; j < y; j++) {
                        p[i + j] = p[i+j] ^ divisor[j];
                    }
                }
            }
            cout<<"CRC detector output :  ";
            for (int i = 0; i < 32; i++) {
                 cout<<p[i];
                if (p[i]==1){
                 flag=1;
                }
            }
             cout<<endl;
            if ( flag==0) {
                        cout << "!PASS!CRC Check at detector is Success" << endl;
                    }
            else {
                        cout << "!ERROR!CRC Check at detector has Failed" << endl;
                    }
    return 0;
}
```

## 5 C simulation Output

```
INFO: [SIM 2] *************** CSIM start ***************
INFO: [SIM 4] CSIM will launch GCC as the compiler.
   Compiling ../../../../codes/crc_tb.cpp in debug mode
   Generating csim.exe
CRC generator output : 01101000101101001111001100000010
CRC detector output :  00000000000000000000000000000000
!PASS!CRC Check at detector is Success
INFO [HLS SIM]: The maximum depth reached by any hls::stream() instance in the
design is 4
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```

# 6 HLS Resource Consumption Report

crc.cpp header.h crc_tb.cpp crc24a_csim.log

**Utilization Estimates**

**Summary**

| Name | BRAM_18K | DSP | FF | LUT | URAM |
|---|---|---|---|---|---|
| DSP | - | - | - | - | - |
| Expression | - | - | 0 | 20 | - |
| FIFO | - | - | - | - | - |
| Instance | - | - | 38 | 768 | - |
| Memory | - | - | - | - | - |
| Multiplexer | - | - | - | 93 | - |
| Register | - | - | 50 | - | - |
| Total | 0 | 0 | 88 | 881 | 0 |
| Available | 624 | 1728 | 460800 | 230400 | 96 |
| Utilization (%) | 0 | 0 | ~0 | ~0 | 0 |

Figure 1: Resource Consumption

# 7 HLS Timing and Fmax Report

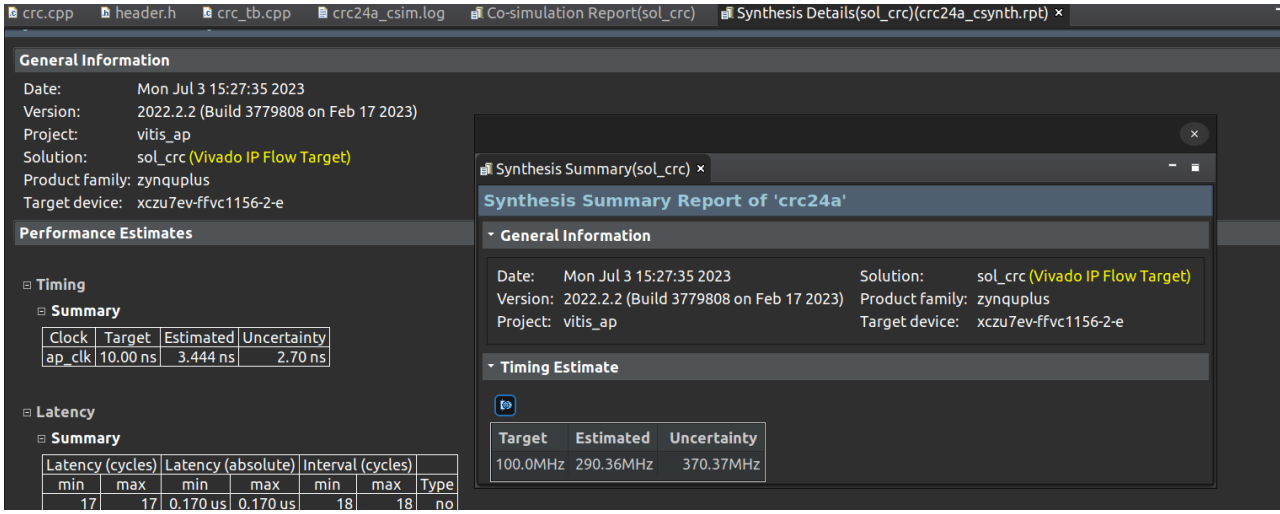



Figure 2: Timing and Fmax

# 8 CoSimulation Report

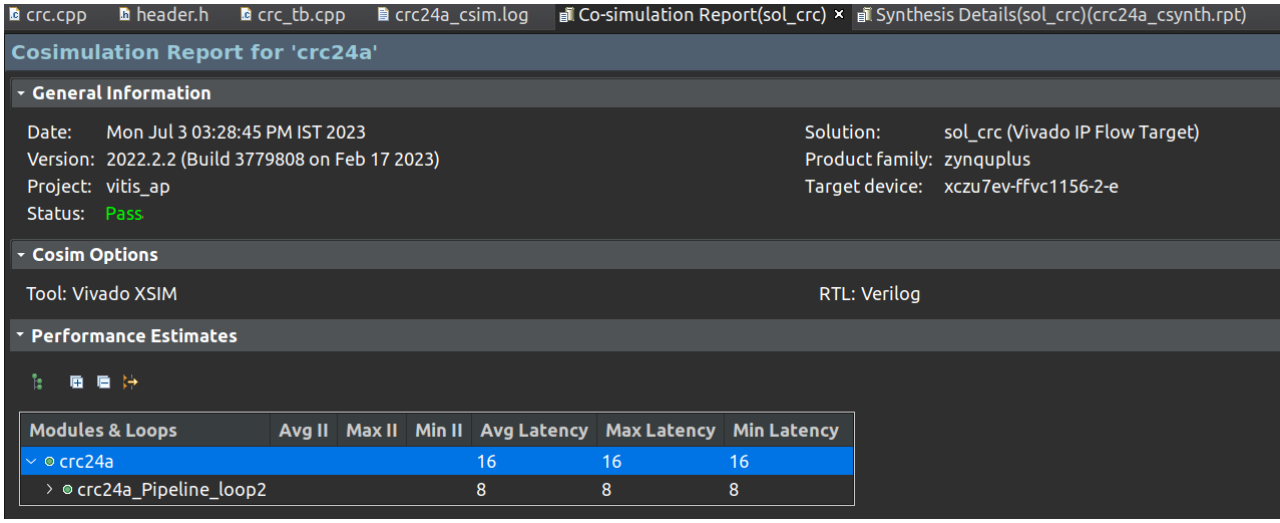



Figure 3: Cosimulation

# 9 Block Design

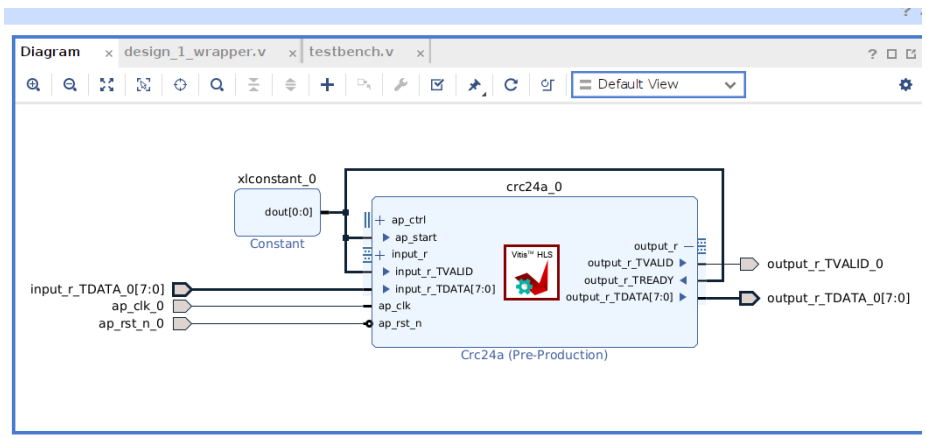


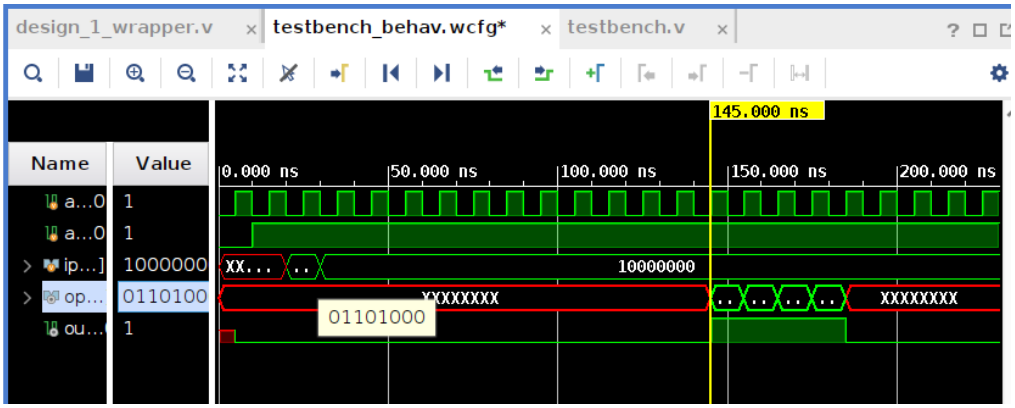

Figure 4: Block Diagram

# 10 Verilog Testbench

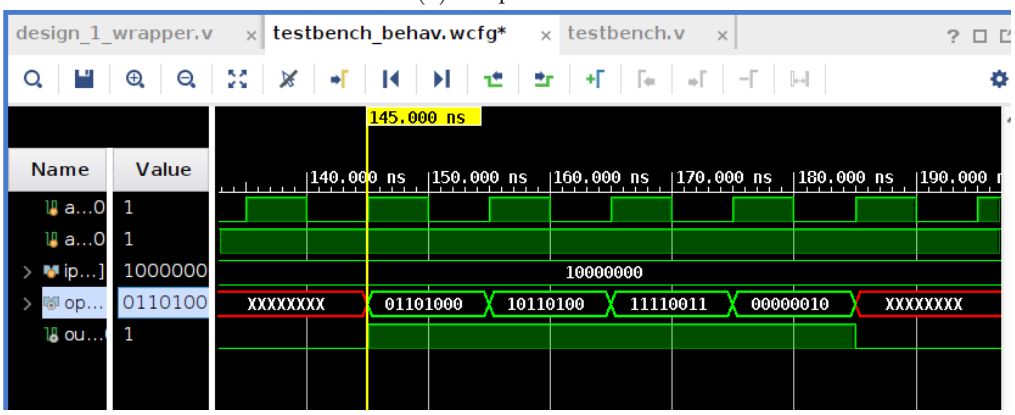
```
//testbench.v
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/26/2023 11:35:30 AM
// Design Name:
// Module Name: testbench
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module testbench();
        reg ap_clk_0;
        reg ap_rst_n_0;
        always #5 ap_clk_0=~ap_clk_0;

        reg [7:0] ip;
        wire [7:0] op;
        wire output_r_TVALID_0;

        initial begin
        ap_clk_0=0;ap_rst_n_0=0;
        #10
        ap_rst_n_0=1;
        #10
        ip=8'b00010110;//ascii "h"
        #10
        ip=8'b00000001;
        #200
        $finish;
        end
      design_1_wrapper uut(.ap_clk_0(ap_clk_0), .ap_rst_n_0(ap_rst_n_0),
      .input_r_TDATA_0(ip),.output_r_TDATA_0(op),
      .output_r_TVALID_0(output_r_TVALID_0));


endmodule
```

# 11 Output Waveform




(a) Output of IP




(b) Zoomed format of above figure

## 12 Design Choices

```
1. Initially I had designed the code with input message stream of unknown length
up to a maximum of 1024,whenever input stream reads a value of 1, until then all
values are stored in a array and then binary division takes place.(code is
uploaded in github named dummy.cpp is in codes folder along with all other codes

2. This code compiles successfully, but while synthesis it gives an error
"unsupported memory access on variable 'vla211' which is (or contains) an array
with unknown size at compile time".

3. Since hls does not support dynamic memory allocation and arrays should be of
fixed length so Initally I took input message length as a fixed variable with 8
bits and continued with the problem accordingly.

4. By using above aspects in point3 I designed the cyclic redundancy check
generator module in HLS.
```

# HLS-Assignment 9 PART-2

July 4, 2023

**VIVADO-VERILOG**

## 1 Problem Statement

Problem Statemt

## 2 CRC bits Generator Code

```
//crcaxis.v
`timescale 1ns / 1ps

module axis_reg #(
        parameter integer DW_IN = 8,
        parameter integer DW_OUT = 32
    )
    (

        input wire clk,
        input wire reset_n,
        input wire [DW_IN - 1:0] s_tdata,
        input wire s_tvalid,
        output wire s_tready,
        output wire [DW_IN - 1:0] m_tdata,
        output wire m_tvalid,
        input wire m_tready
    );

    reg m_tvalid_i;
    reg [0:24] divisor = 25'b1100001100100110011111011;
    reg [0:31] crc_reg,crc_own;
    reg [1:0] cycle_counter;
```

```
    reg [7:0] oup;
    integer i, j;

    always @(posedge clk) begin
        if(!reset_n) begin
            m_tvalid_i <= 0;
            crc_reg <= 0;
            crc_own<=0;
            cycle_counter<=0;
        end else if(s_tready && s_tdata!=8'b00000001) begin
             crc_reg  = {s_tdata,{24{1'b0}}};

    for (i = 0; i <=7; i = i + 1) begin
      if (crc_reg[i] == 1) begin
        for (j = 0; j < 25; j = j + 1) begin
          crc_reg[i + j] = crc_reg[i + j] ^ divisor[j];
        end
      end
    end
    end
    crc_own = {s_tdata,crc_reg[8:31]};
    oup=crc_own[7+(8*cycle_counter) -:8];
    cycle_counter = cycle_counter + 1;

    end

    assign m_tdata = oup;
    assign m_tvalid = m_tdata?1:0;
    assign s_tready = m_tready || !m_tvalid;
endmodule

endmodule
```

## 3 Test Bench Code

```
//axistb.v
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/27/2023 10:47:14 AM
// Design Name:
// Module Name: axistb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module axistb(

    );
    reg clk;
    reg reset_n;
    reg [7:0] s_tdata;
    reg s_tvalid;
    wire s_tready;
    wire [7:0] m_tdata;
    wire m_tvalid;
    reg m_tready;

    // Instantiating
   axis_reg #(
        .DW_IN(8),
        .DW_OUT(32)
    ) dut (
        .clk(clk),
        .reset_n(reset_n),
        .s_tdata(s_tdata),
```

```
        .s_tvalid(s_tvalid),
        .s_tready(s_tready),
        .m_tdata(m_tdata),
        .m_tvalid(m_tvalid),
        .m_tready(m_tready)
    );
/*   design_1_wrapper uut
 (.clk_0(clk),
  .m_0_tdata(m_tdata),
  .m_0_tvalid(m_tvalid),
  .reset_n_0(reset_n),
  .s_0_tdata(s_tdata),
  .s_0_tready(s_tready),
  .s_0_tvalid(s_tvalid),
  .m_0_tready(m_tready));
  */
  // Clock generation
  always #5 clk = ~clk;

  initial begin
      // Initialize inputs
      clk = 0;
      reset_n = 1;
      s_tdata = 8'h00;
      s_tvalid = 0;


      // Apply reset
      reset_n <= 0;
      #10;
      reset_n <= 1;
      m_tready<=1;


      // Send data and wait for it to be accepted
      s_tdata <= 8'b01101000;
      #10
      s_tdata <= 8'b00000001;
      #10
      s_tdata <=8'dx;

      s_tvalid <= 1;
      #10;
      s_tvalid <= 0;
      #20
      m_tready<=0;
```

```
        #5
        // Finish simulation
        $finish;
    end

endmodule
```

# 4 Output Waveform

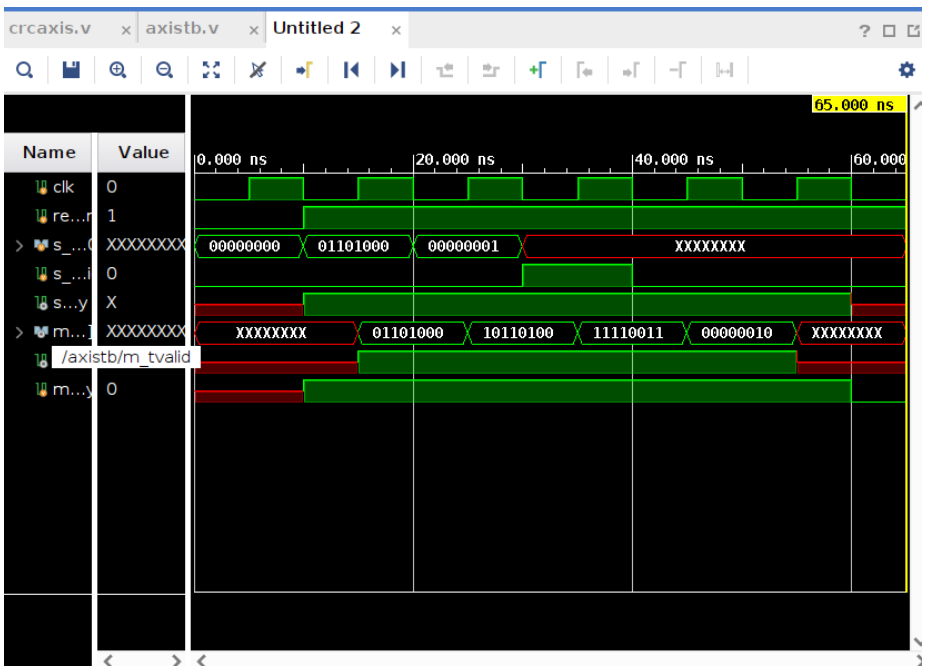



Figure 1: Output of RTL Testbench

# 5 Block Design

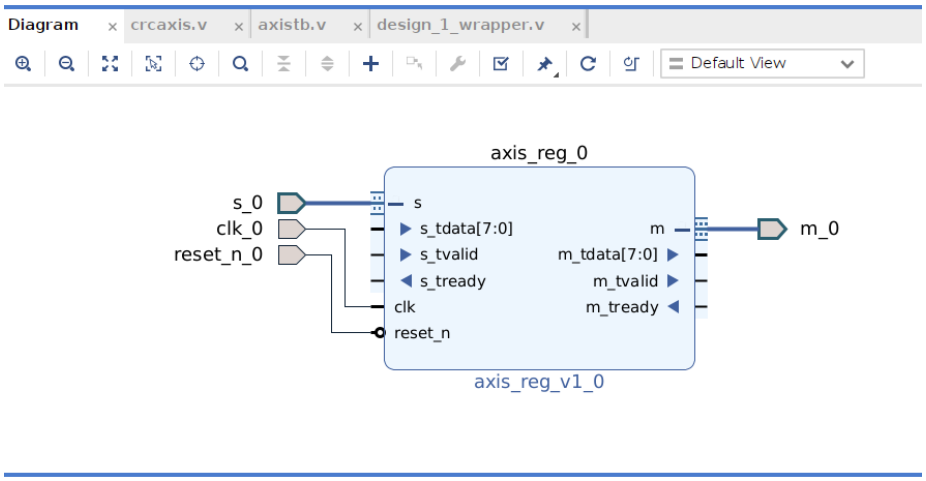



Figure 2: Block Diagram

# 6 Verilog Testbench

```
//axistb.v
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/27/2023 10:47:14 AM
// Design Name:
// Module Name: axistb
```

```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////


module axistb(

    );
    reg clk;
    reg reset_n;
    reg [7:0] s_tdata;
    reg s_tvalid;
    wire s_tready;
    wire [7:0] m_tdata;
    wire m_tvalid;
    reg m_tready;

    // Instantiating
 /*  axis_reg #(
        .DW_IN(8),
        .DW_OUT(32)
    ) dut (
        .clk(clk),
        .reset_n(reset_n),
        .s_tdata(s_tdata),
        .s_tvalid(s_tvalid),
        .s_tready(s_tready),
        .m_tdata(m_tdata),
        .m_tvalid(m_tvalid),
        .m_tready(m_tready)
    );*/
     design_1_wrapper uut
   (.clk_0(clk),
    .m_0_tdata(m_tdata),
    .m_0_tvalid(m_tvalid),
    .reset_n_0(reset_n),
    .s_0_tdata(s_tdata),
```

```
    .s_0_tready(s_tready),
    .s_0_tvalid(s_tvalid),
    .m_0_tready(m_tready));

    // Clock generation
    always #5 clk = ~clk;

    initial begin
        // Initialize inputs
        clk = 0;
        reset_n = 1;
        s_tdata = 8'h00;
        s_tvalid = 0;


        // Apply reset
        reset_n <= 0;
        #10;
        reset_n <= 1;
        m_tready<=1;


        // Send data and wait for it to be accepted
        s_tdata <= 8'b01101000;
        #10
        s_tdata <= 8'b00000001;
        #10
        s_tdata <=8'dx;

        s_tvalid <= 1;
        #10;
        s_tvalid <= 0;
        #20
        m_tready<=0;
        #5
        // Finish simulation
        $finish;
    end

endmodule
```

# 7 Output Waveform

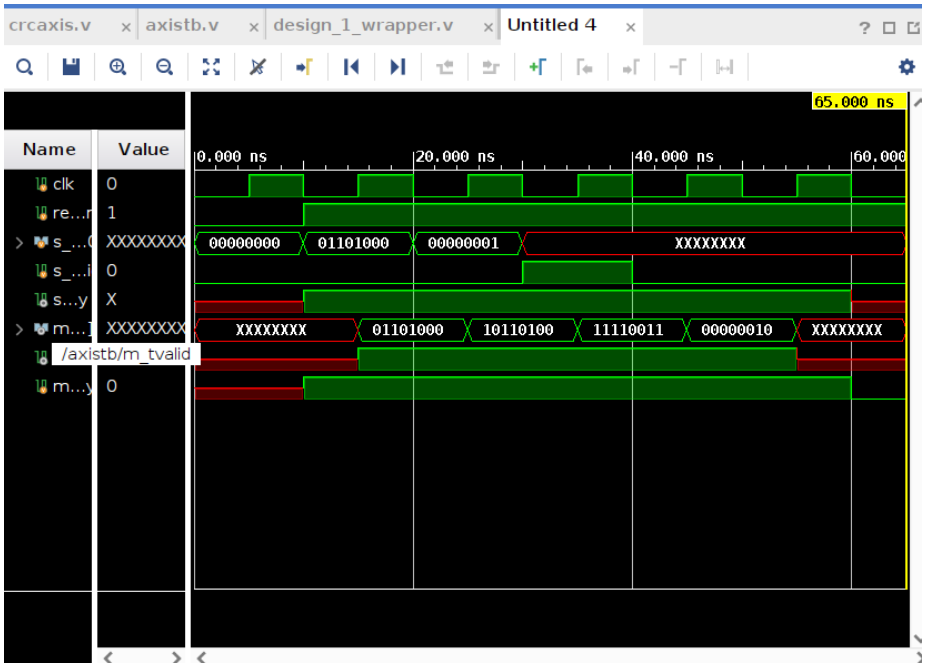



Figure 3: Output of IP Testbench

# 8 Design Choices

1. With same design Choices as PART–1, I designed PART–2 by defining all I/O buses manually.

# 9 Matlab Reference


Figure 4: Matlab Reference

# 10 Conclusion

The Output of CRC IP in both PART–1 and Part–2 is matching with Output of reference Matlab code and also using this floating Point Converter Online : https://www.h-schmidt.net/FloatConverter/IEEE754.html

**GITHUB :** https://github.com/dk-425/Training.git