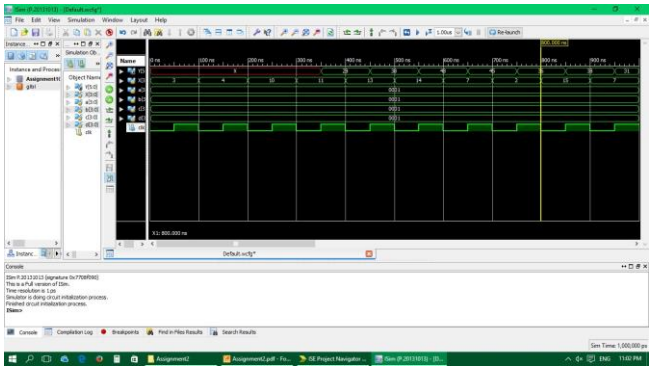


ADVANCED DIGITAL SYSTEM DESIGN

Assignment 5 Solutions

ABSTRACT

In this assignment our job is to develop a Given Micro-Architecture [Datapath & Controller] of the IPU which include Labelled and complete Datapath showing all data & control signal routings & connection, Correct & Complete Controller FSM, coding it in Verilog & providing simulation results in a PDF document, performing the simulation in MATLAB using the Fixed-Point Toolbox.

HAZOOR AHMAD
PHDEE17004

ZEESHAN HAIDER
MEE17001

Hazoor Ahmad PHDEE17004 Zeeshan Haider MSEE17001

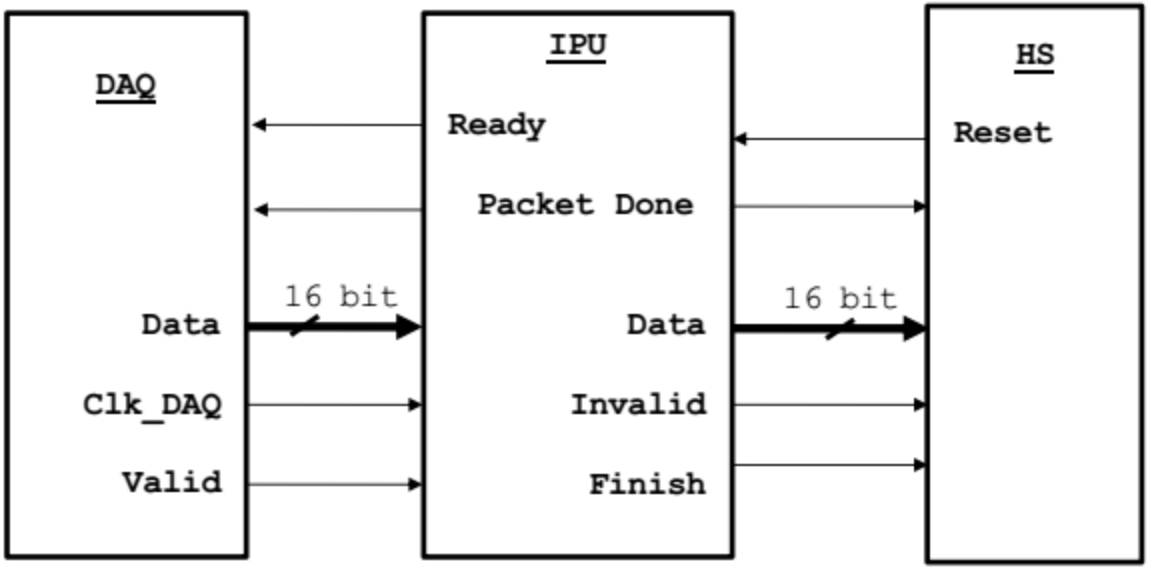
Advanced Digital System Design

Question Requirements:

- Provide the Micro-Architecture (Good to use PowerPoint/Visio to make your diagrams)
- Then code it in Verilog & provide your simulation results in a PDF document
- You also need to perform the simulation in MATLAB using the Fixed-Point Toolbox. The coefficients and x[n] shall be all in Q8.8 format (Shall be discussed in Class).
- The code for DAQ & HS is also required to be provided. DAQ shall use a ROM to initialize it with the same x[n] as for Matlab.
- Your output should match that of a corresponding simulation performed by yourselves in MATLAB. Make your own evaluation framework that allows you to test your code.
- Your MATLAB simulation should generate a x[n] data file for your DAQ. DAQ should import it using $readmemh. After your simulation your HS should store the results in a memory and you should dump the memory in a file. Then import it back in MATLAB to show that both the MATLAB and Xilinx results match.

To be submitted:

- Zipped Xilinx Project Folder with source codes & simulation
- PDF report containing
    - Micro-Architecture (Data-path + Complete FSM)
    - Your MATLAB vs Xilinx Simulation Results
- MATLAB code
- Fixed Point Simulation of the functionality using the MATLAB's Fixed-Point Toolbox



Design Specifications:

You have to design an IPU (Interfacing & Processing unit) that will provide an interface between a Data Acquisition Module (DAQ) and a Host System (HS). The IPU provides the functionality of applying the following processing on the received data

$$y[n] = a\,x[n-2] + b\,x[n-4] + c\,x[n-6]$$

The figure below shows the interfacing blocks at a very abstract level.

The DAQ provides a new 16-bit sample (x[n]) on every positive edge of its clock on its primary Data line (Data). The data is organized in the form of packets where each packet consists of TWELVE 16-bitconsecutive samples of x[n]. The DAQ also provides a “Valid” signal that defines the window in which the packets are valid. Only valid packet needs to be processed in the IPU. Non-valid packet/data will be discarded.

The required functionality of the IPU is explained below:

1. Whenever the HS wants to receive the data; or whenever the HS wants to reset the IPU; the HS gives a single cycle “reset” signal to IPU. This also initializes the IPU output to ZERO.
2. Starting from the next clock cycle; the IPU sends a “ready” signal to the DAQ. This informs the DAQ that the HS is ready to get the data from DAQ via IPU.
3. When the DAQ has valid data ready; it responds by sending a continuous stream of data in the form of packets.
4. A single cycle “Packet Done” signal is generated by IPU after processing of every packet.
5. A single cycle “Finish” signal is generated by IPU after a batch of 32 Packets is processed.
6. The IPC shall enter a wait state after the Finish signal & wait for ‘reset’ signal from HS to start the processing again.
7. In-case the input data from DAQ is in-valid; the IPU simply freezes all the computational blocks in their current state and sets the Invalid flag for the HS.

In addition to the signals shown in the figure; following functionalities are also required. There is also an internal variable Average Value that stores the average of all the y[n] till yet. There is an input signal to IPU named TEST and an output to the IPU

named TEST_Result. The functionality of these signals is as follows:

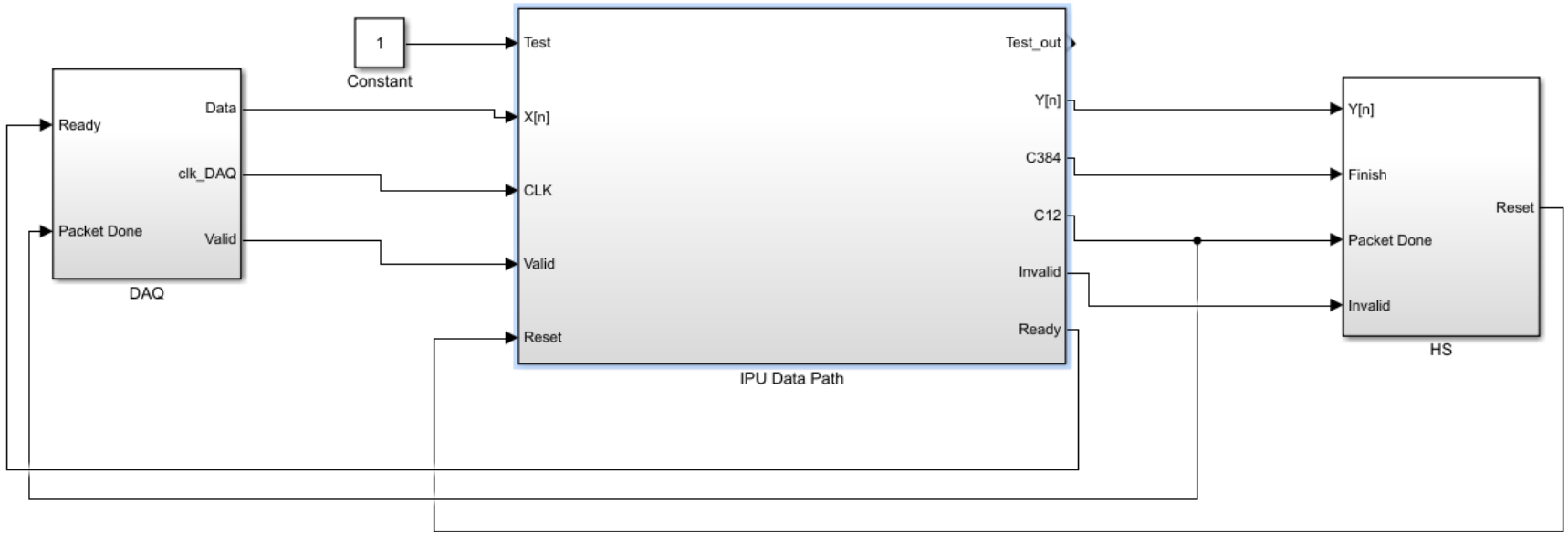
- If (TEST=1); value of TEST_Result is ONE if current y[n] is greater than average y[n]
- If (TEST=1); value of TEST_Result is ZERO if current y[n] is less than or equal to average y[n]
- If(TEST=0); value of TEST Result is ZERO.

Solution:

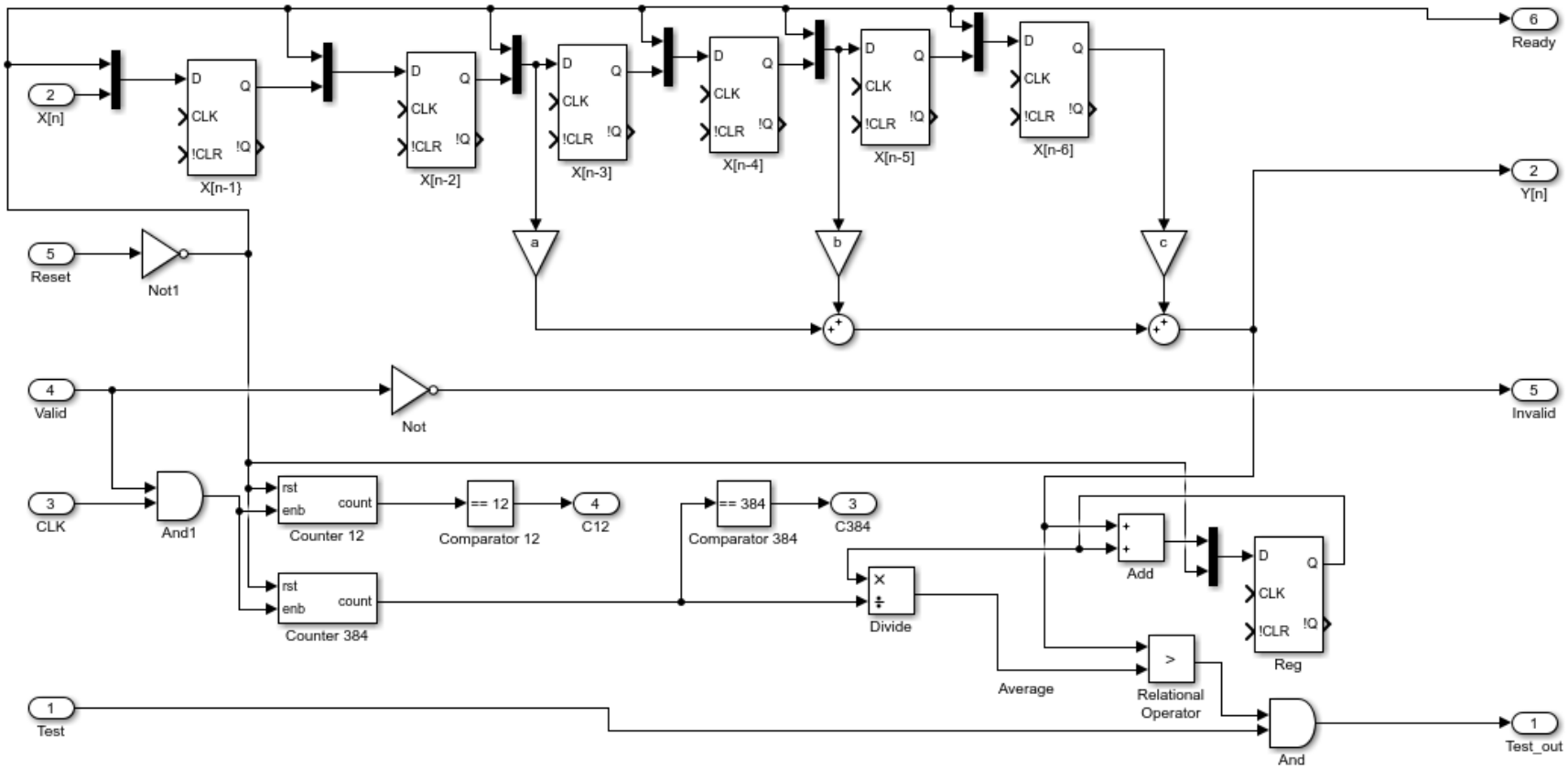
There are following four parts of solution:

1. Making a Labelled Datapath and Controller.
2. Making Complete Finite State Machine.
3. MATLAB code for Generation of Fixed Point Numbers for DAQ.
4. Xilinx Modules for DAQ, IPU and HS.
5. Macro Module and Testbench
6. MATLAB code for Simulation using Fixed Point Numbers.

1. Datapath and Controller:

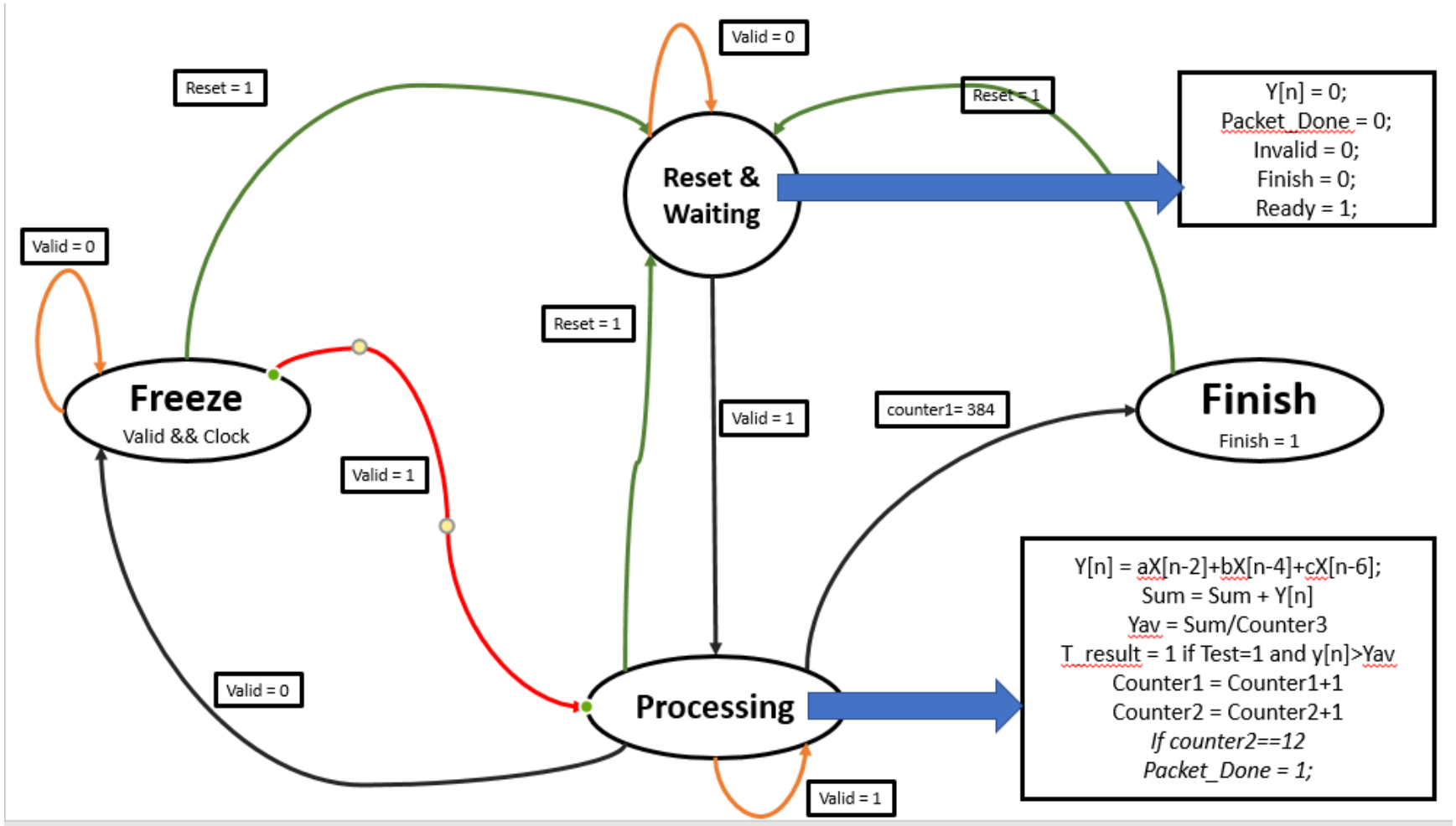


Abstract Level Datapath (Made in MATLAB Simulink)



Micro Architecture Datapath of IPU (Made in MATLAB Simulink)

2. Finite State Machine:



3. Fixed Point Number Generation:

Following is the MATLAB code for storing fixed point numbers in Q8.8 form:

```
x = [4.6875  1.1875  1.1250  1.4375  1.5000  0.1875  0.8125  4.5000  2.4375  2.8750  4.0625  0.3750  2.8125
     2.4375  3.1250  0.2500  4.0000  2.6250  2.3750  4.0625  3.6250  1.7500  2.8125  1.1250  4.6250  2.3125
     2.6875  2.6250  0.8750  0.6875  0       3.2500  2.3750  1.9375  3.8750  0.8750  5.0000  3.1875  2.1250
     1.1250  2.0625  1.3125  2.5000  4.4375  1.3750  0.5625  1.5000  1.1875  2.5625  0.3125  0.5000  3.8750
     0.5000  1.0625  1.0000  3.0625  2.0000  2.5625  4.3750  3.4375  4.2500  2.7500  2.1250  3.5000  3.2500
     0.3125  4.3125  3.0000  3.5000  1.5000  1.2500  3.1875  4.3750  4.0625  0.8125  2.0000  2.0000  3.3125
     4.1250  4.3125  4.1250  1.9375  4.7500  1.5000  0.5000  4.6250  4.2500  2.5000  2.0000  0.4375  0.8125
     3.8750  1.4375  4.0625  3.6250  2.3125  3.2500  4.5625  0.4375  1.8750  2.6875  0.5625  1.0625  2.1875
     1.0000  3.0625  4.0000  4.4375  1.4375  4.4375  0.1875  2.4375  1.9375  1.0000  2.1250  1.2500  2.5000
     4.5000  3.8750  3.5625  0.2500  0.4375  4.3125  0.1875  0.6250  4.3125  3.1250  3.0625  1.1250  1.8750
     3.4375  4.7500  4.3125  0.6875  2.9375  1.0625  4.9375  1.2500  2.3125  0.1250  4.9375  3.5000  2.3750
     3.3125  4.6250  3.8750  4.3125  0.7500  2.0000  3.4375  4.4375  2.8750  1.6250  1.1250  3.7500  1.6875
     4.9375  1.9375  1.0000  2.1250  0.3750  4.4375  1.8750  1.0000  3.5000  2.6875  1.7500  2.7500  3.8750
     0.1875  0.8125  4.9375  2.0625  2.8750  1.3125  2.5000  0.6250  4.8125  1.6250  1.3125  2.7500  1.9375
     2.1250  4.0000  0.4375  4.8125  1.4375  4.8125  3.8125  2.6875  2.7500  3.0000  1.4375  3.1250  3.5625
     2.4375  0.5625  3.2500  3.7500  1.8125  3.1250  0.2500  1.5625  3.1875  1.8125  0.9375  4.9375  2.3750
     2.9375  0.8125  1.0625  4.8750  3.6250  0.8125  3.6250  1.9375  2.8750  0.6875  0.1250  3.1875  3.6250
     0.4375  1.7500  1.2500  1.1875  4.3125  4.1250  3.5000  3.9375  4.6250  4.5625  2.2500  3.0000  4.6875
     3.3125  4.2500  1.6875  0.5000  1.7500  3.2500  2.3125  4.1875  4.3125  3.1875  1.1875  4.5625  2.5625
     0.2500  2.9375  1.0000  1.9375  4.8125  2.7500  2.9375  3.3750  0.8750  3.3125  4.3125  2.8750  4.5000
     2.8125  2.9375  2.5625  2.5000  4.7500  1.2500  1.6875  2.0625  0.8750  3.3750  2.6250  1.6875  1.0625
     3.5625  4.6250  4.7500  2.8750  1.0000  0.1875  0.8750  3.1875  1.1875  3.7500  4.5625  4.8125  4.3750
     2.4375  2.8750  2.0000  4.8750  3.8125  1.1875  2.0000  1.0625  3.7500  4.1875  4.8750  2.1875  0.4375
     3.0625  0.0625  2.9375  2.4375  3.0625  1.8125  4.6250  3.0625  1.0000  2.5625  2.9375  3.0000  2.3125];
     1.0625  4.0625  3.0625  2.0000  4.6250  3.1875  1.1250  3.3750  4.9375  0.7500  0.6250  3.6250
     3.2500  3.0625  3.5625  5.0000  3.0000  4.9375  1.8125  3.0000  3.5625  1.8750  4.6250  3.3750
     1.8750  2.3750  2.0000  1.3125  3.5000  1.0625  1.6250  1.7500  0.8750  4.1250  2.9375  1.0625
     0.5000  1.3125  4.3125  3.3125  3.6875  3.8125  0.4375  1.8125  4.3125  0.8750  4.4375  0.4375
     1.8750  1.3125  4.6250  4.8125  1.9375  4.4375  2.5625  0.8750  4.5625  1.6250  2.1250  1.3750
     1.3125  2.3750  3.7500  3.3750  1.2500  2.3750  4.1875  4.0000  4.8125  4.8125  3.0625  4.3125
```

```
x = fi(x,0,16,8);
h = fi([0 0.25 0 0.5 0 0.75]',0,16,8);
y = fi(conv(x,h),0,16,8);
memx = x.bin;
memh = h.bin;
memy = y.bin;
dlmwrite('memoryx.list',memx,'delimiter','');
dlmwrite('memoryh.list',memh,'delimiter','');
dlmwrite('memoryyMATLAB.list',memy,'delimiter','');
```

4. Xilinx Verilog Modules for DAQ, IPU, and HS:

Data Acquisition Module:

```
module DAQ(
input Ready,
input Packet_Done,
input clk,
output reg Valid,
output clk_DAQ,
output reg [15:0] Data_in);
     reg [15:0]memoryx[0:383];
     reg [8:0]counter;
initial
     begin

$readmemb("ExtraFiles/memoryx.list",memoryx);
     end
      assign clk_DAQ  = clk;
always @(posedge clk or posedge Ready)
     begin
     if(Ready)
          begin
             Valid = 1;
           Data_in=memoryx[counter];
             if(counter==384)
           begin
          counter<=0;
             end
             else
          counter<=counter+1;
           end
     else
           begin
          counter<=0;
                                  Valid = 0;
             end
     end
endmodule
```

Interfacing and Processing Unit:

```
module IPU(
input Reset,
input clk_in,
input Valid, Test,
input [15:0]X,
output reg Invalid,
output reg Test_Result,
output reg Finish,
output reg Ready,
output reg Packet_Done,
output [15:0]Y,
output [8:0]count,
output [3:0]count1
);
parameter Reset_Waiting_state=2'b00,
          processing_state=2'b01,
          Freeze_state=2'b10,
          Finish_state=2'b11;
reg [15:0]avg;
reg [8:0]counter;
reg [3:0]counter1;
reg [2:0] next_state;
reg [2:0] present_state;
assign count=counter;
assign count1=counter1;
always @(posedge clk_in or posedge Reset)
  begin
    if(Reset)
     begin
      present_state <= Reset_Waiting_state;
      counter<=1;
      counter1<=1;
    end
    else
     begin
      present_state<=next_state;
      if(Valid)
        begin
       if(counter==384)
         counter<=1;
       else
         counter<=counter+1;

       if(counter1==12)
         counter1<=1;
       else
         counter1<=counter1+1;
      end
      else
        begin
        counter1<=counter1;
        counter<=counter;
      end
   end
end
```

```
always@(present_state,Valid,Reset,counter,counter1,Finish,a
vg)
  begin
   case(present_state)

       Reset_Waiting_state:
       begin
       if(Valid)
           next_state=processing_state;
         else
           next_state=Reset_Waiting_state;
       end
       processing_state:
       begin
         if(Reset)
           next_state=Reset_Waiting_state;
         else begin
            if(!Valid)
           next_state=Freeze_state;
         else
            next_state=processing_state;
            if(counter==384)
           next_state=Finish_state;
           end
       end

         Freeze_state:
       begin
         if(Reset)
           next_state=Reset_Waiting_state;
         else
            begin
           if(!Valid)
             next_state=Freeze_state;
           else
             next_state=processing_state;
           end
       end
         Finish_state:
       begin
            if(Reset)
           next_state=Reset_Waiting_state;
            else
            next_state=Finish_state;
            end
         default:
       begin
         next_state=Reset_Waiting_state;
       end
    endcase
  end
FIR #(16,16,16,6)FinitImpulse(X,clk_in,Reset,Y,Valid);
always @(present_state,Y,avg,Finish,counter1,Test)
  begin
    case(present_state)
       Reset_Waiting_state:
      begin
         avg=0;
        Packet_Done= 0;
        Invalid = 1;
        Finish = 0;
        Ready = 1;
```

```
        Test_Result=0;
    end
     processing_state:
    begin
    if(Valid) begin
     Invalid=!Valid;
     avg=(Y+avg)>>1;
        if(counter1==12)
        Packet_Done=1;
     else
        Packet_Done=0;
        if(Test==0)
        Test_Result=0;
     else
        Test_Result=Test&(Y>avg);
        Ready = 1;
    end
     else begin
     Invalid=Valid;
     end
      end
      Freeze_state:
    begin
      Invalid=!Valid;
      if(Valid) begin
        Ready=1;
          end
      else begin
        Ready=0;
          end
    end
      Finish_state:
    begin
      Finish = 1;
      Ready = 0;
    end
  endcase
 end
endmodule
```

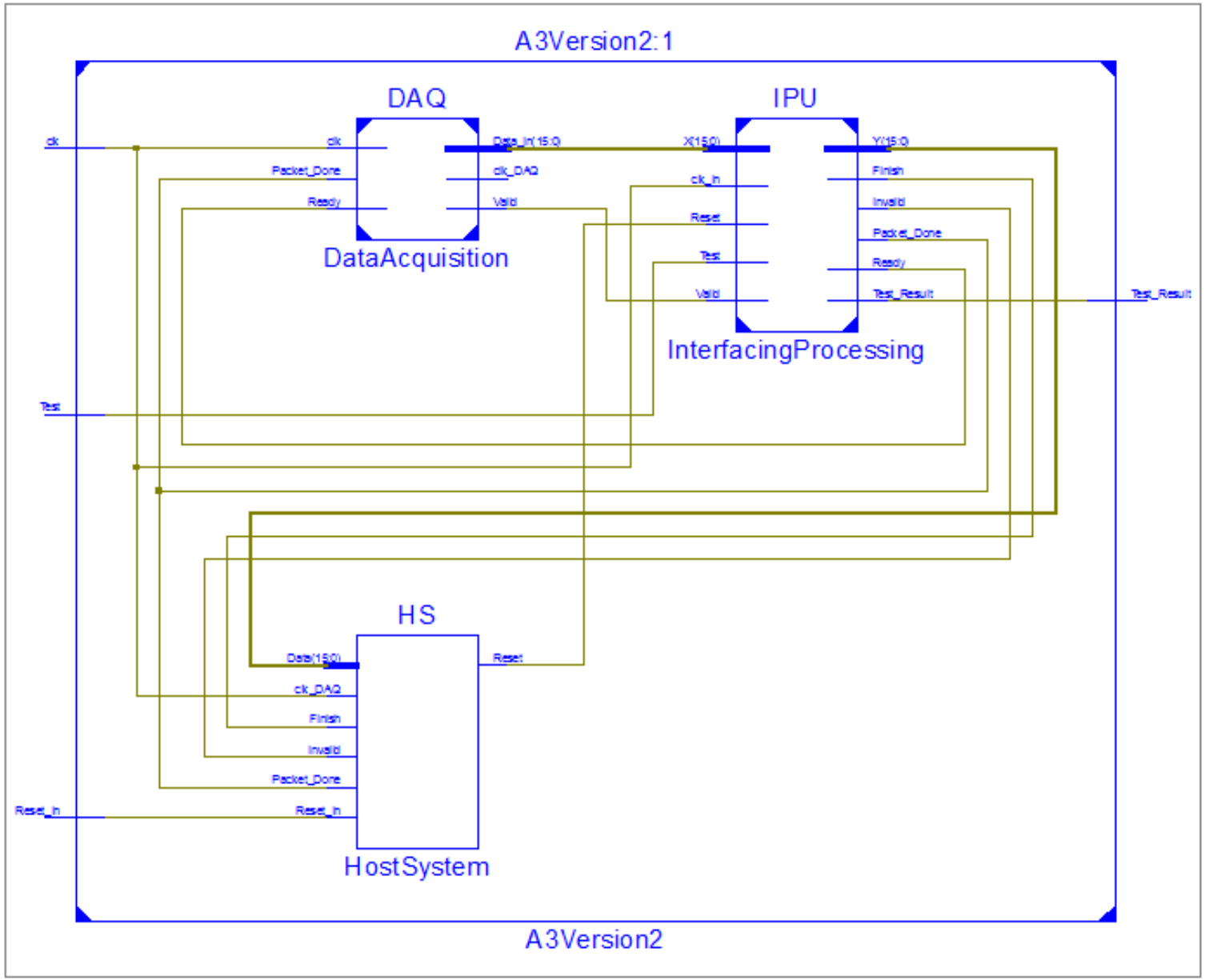
Host System:

```
module HS(
input Reset_in,
input Packet_Done,
input clk_DAQ,
input Invalid,
input Finish,
input [15:0]Data,out,
output Reset);
assign Reset = Reset_in;
assign out=Data;
integer file,k;
always @(posedge clk_DAQ or posedge Reset_in)
begin
    if(Reset_in) begin
     file = $fopen("ExtraFiles/memoryy.list","w");
    end else
```

```
    begin
     if(!Invalid)
       begin
       if(Finish)
          $fclose(file);
       else begin
          $fwrite(file,"%b\n",Data);
       end
        end
   end
end
endmodule
```

Finite Impulse Response Filter:

```
module FIR #(parameter IPL=16,parameter CEL=16,parameter
OPL=16,parameter IPD=6)(X,clk,Reset,Y,Valid);
input [(IPL-1):0]X;
input clk,Reset,Valid;
output [(OPL-1):0]Y;
reg [CEL-1:0]memoryh[0:IPD-1];
initial begin
$readmemb("ExtraFiles/memoryh.list",memoryh);
end
integer k,j;
reg [(IPL-1):0]Xn[0:(IPD-1)];
reg [31:0]Temp;
assign Y=Temp[23:8];
always @(posedge clk)
begin
  Temp=memoryh[0]*Xn[0];
  for (k=1; k <= IPD-1; k = k + 1)
    Temp= Temp + memoryh[k]*Xn[k];
end
always @(posedge clk or posedge Reset) begin
if(Reset) begin
for (j=(IPD-1); j>=0; j = j - 1)
Xn[j] <= 0;
end
else begin
if(Valid) begin
for (j=(IPD-1); j>=1; j = j - 1)
Xn[j] <= Xn[j-1];
Xn[0] <= X;
end
end end
endmodule
```


A3Version2:1
DAQ
IPU
clk
clk
Packet_Done
Ready
Data_in(15:0)
clk_DAQ
Valid
DataAcquisition
X(15:0)
clk_in
Reset
Test
Valid
Y(15:0)
Finish
Invalid
Packet_Done
Ready
Test_Result
InterfacingProcessing
Test_Result
Test
HS
Data(15:0)
clk_DAQ
Finish
Invalid
Packet_Done
Reset_in
Reset
HostSystem
Reset_in
A3Version2

5. Xilinx Verilog Macro Module and Testbench:

Macro Module:

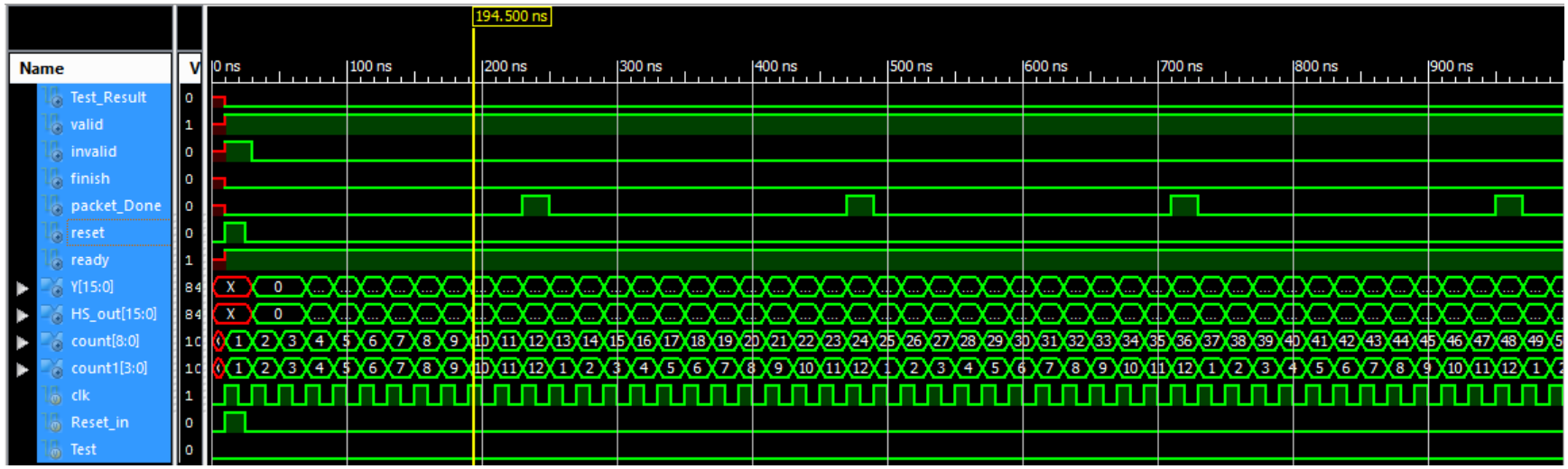
```
`timescale 1ns / 1ps
module XilinxAssignment3(clk, Reset_in, Test, Test_Result,valid, invalid, finish, packet_Done,reset,ready,Y,HS_out,count,count1);
input clk, Test, Reset_in;
output Test_Result,
valid, invalid, finish, packet_Done,reset,ready;
output [15:0]Y,HS_out;
output [8:0]count;
output [3:0]count1;
wire clk_DAQ;
wire [15:0] Data_in, Data;
wire Valid, Invalid, Finish, Packet_Done;
wire Reset,Ready;
assign valid=Valid;
assign invalid=Invalid;
assign finish=Finish;
assign packet_Done=Packet_Done;
assign reset=Reset;
assign ready=Ready;
assign Y=Data;
DAQ DataAcquisition(
```

```
.Ready(Ready),
.Packet_Done(Packet_Done),
.clk(clk),
.Valid(Valid),
.clk_DAQ(clk_DAQ),
.Data_in(Data_in));
IPU InterfacingProcessing(
.Reset(Reset),
.clk_in(clk),
.Valid(Valid), .Test(Test), .X(Data_in),
.Invalid(Invalid),
.Test_Result(Test_Result), .Finish(Finish),
.Ready(Ready), .Packet_Done(Packet_Done), .Y(Data),
.count(count),.count1(count1));
HS HostSystem(.Reset_in(Reset_in),
.Packet_Done(Packet_Done),
.clk_DAQ(clk), .Invalid(Invalid),
.Finish(Finish), .Data(Data), .Reset(Reset),
.out(HS_out));
endmodule
```

Testbench:

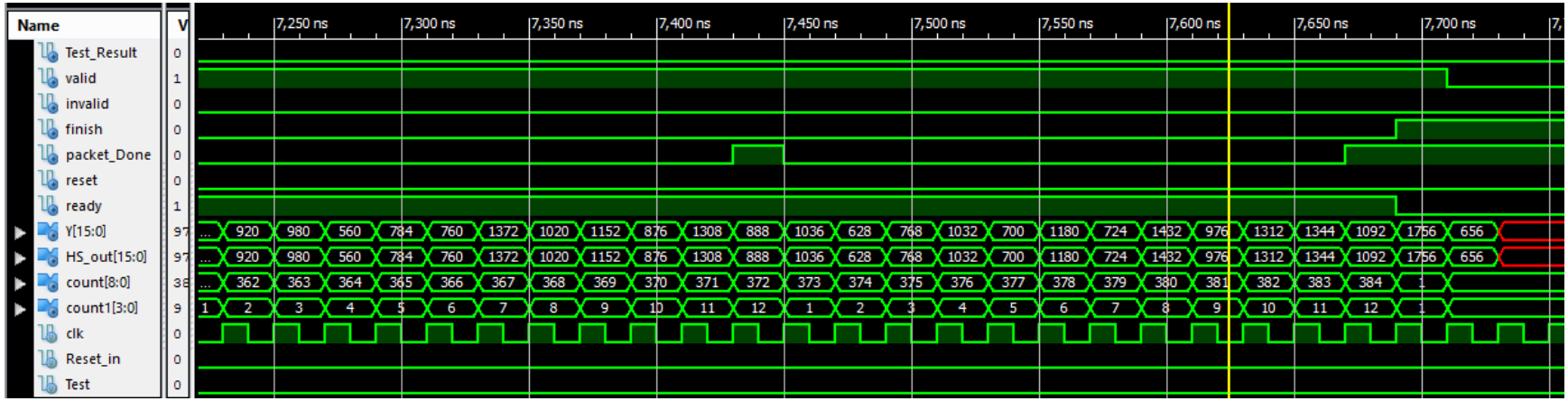
```
`timescale 1ns / 1ps
module TestBenchAssignment3;
    reg clk;
    reg Reset_in;
    reg Test;
    wire Test_Result;
    wire valid;
    wire invalid;
    wire finish;
    wire packet_Done;
    wire reset;
    wire ready;
```

```
wire [15:0] Y;
wire [15:0] HS_out;
wire [8:0] count;
wire [3:0] count1;
XilinxAssignment3 uut (
    .clk(clk),
    .Reset_in(Reset_in),
    .Test(Test),
    .Test_Result(Test_Result),
    .valid(valid),
    .invalid(invalid),
    .finish(finish),
    .packet_Done(packet_Done),
    .reset(reset),
    .ready(ready),
    .Y(Y),
    .HS_out(HS_out),
    .count(count),
    .count1(count1)
    );
initial begin
clk=0;
forever #10 clk=~clk;
end
   initial begin
      Reset_in = 0;
      Test = 0;
      #10;
     Reset_in = 1;
      #15;
      Reset_in = 0;
      #8000;
      Test = 1;
     Reset_in = 1;
      #15;
      Reset_in = 0;
    end
endmodule
```



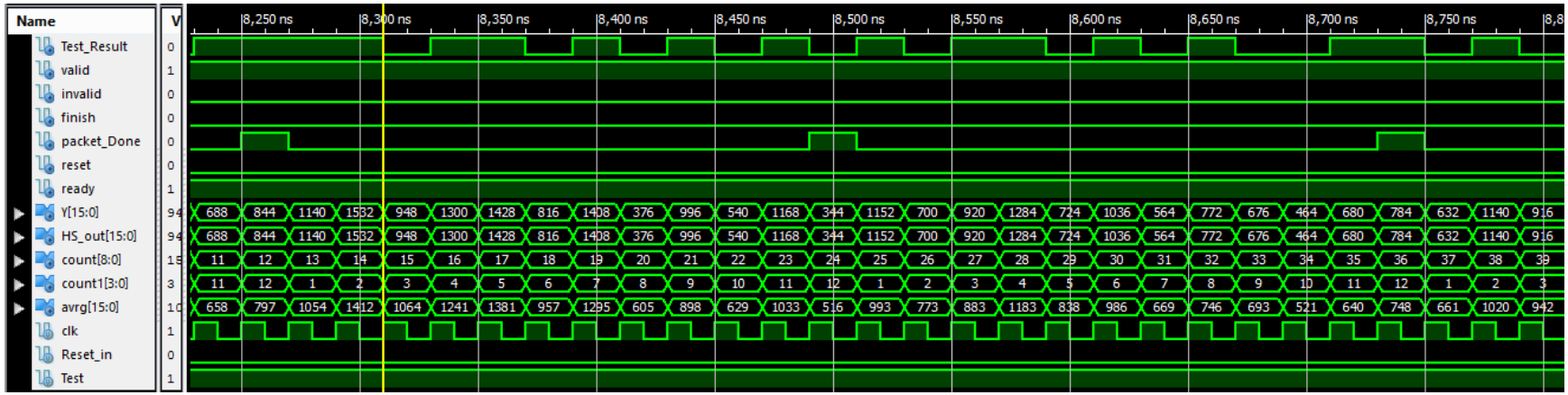
Packet Done after every 12 counts of data

Test Result Being LOW at all values of (not Y>avrg) or Test=0



Finishing after 384 counts of data and then waiting for Reset

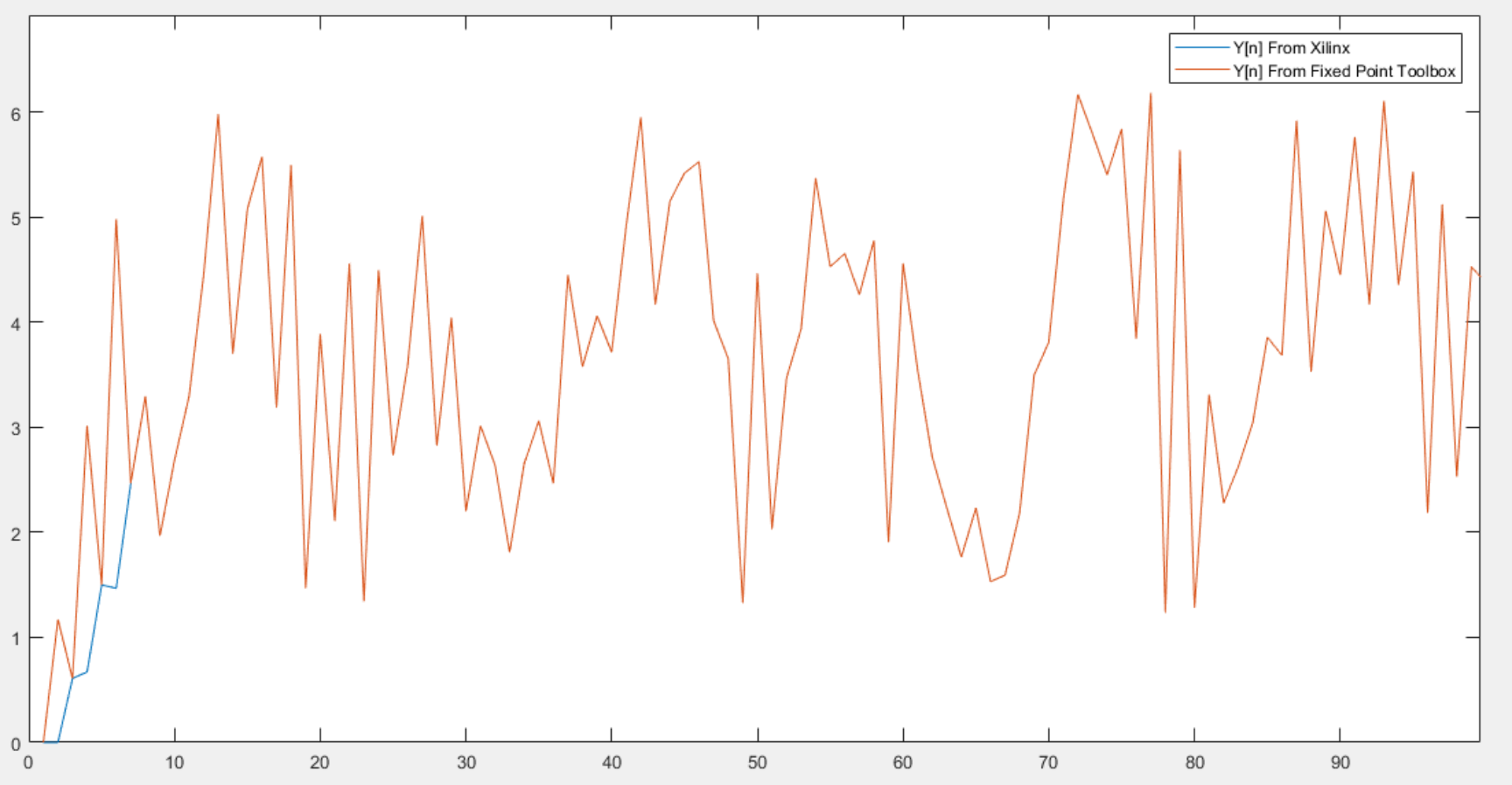
Test Result Being LOW at all values of (not Y>avrg) or Test=0



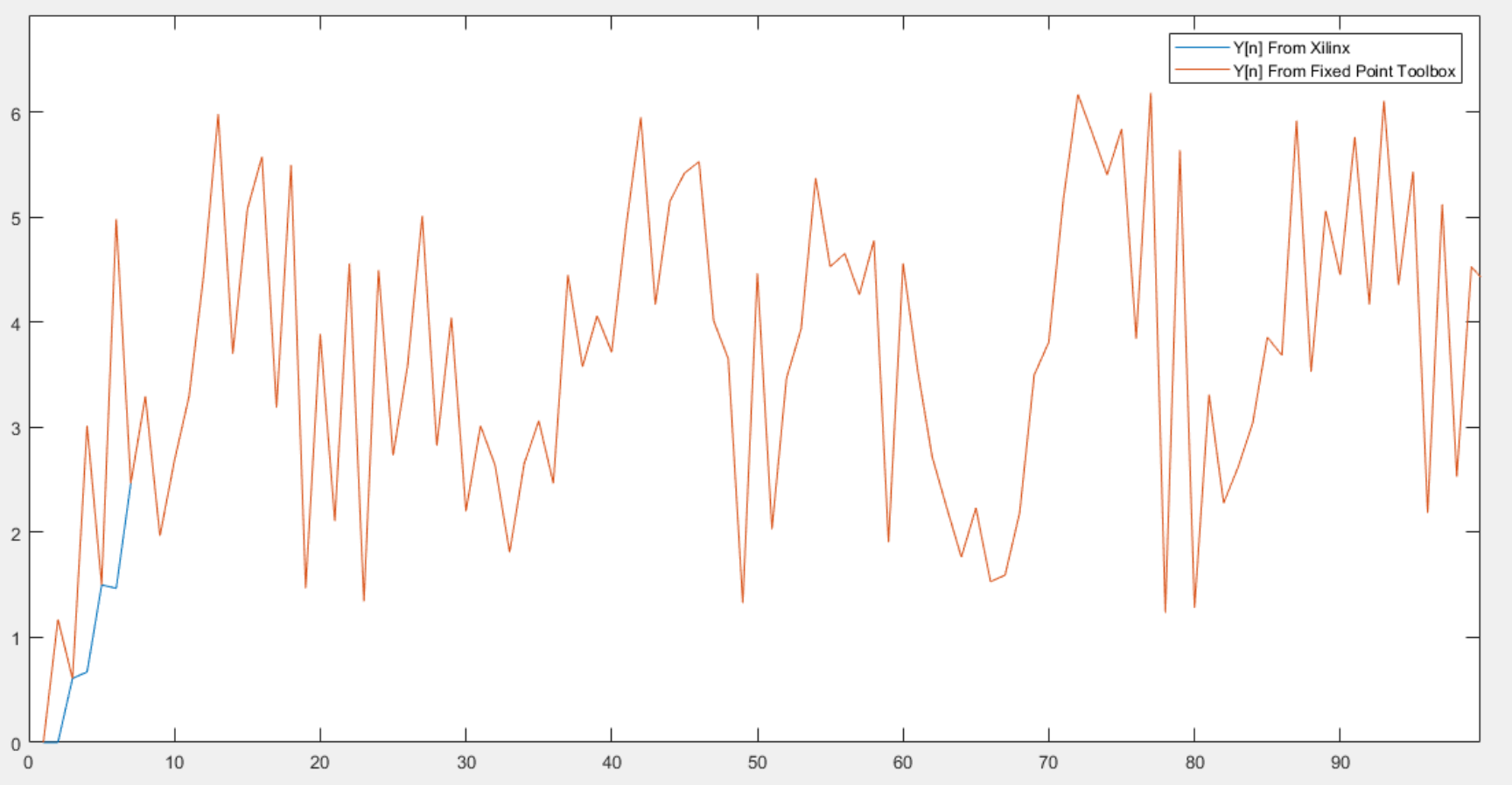
Test Result Being HIGH at all values of Y>avrg and Test=1

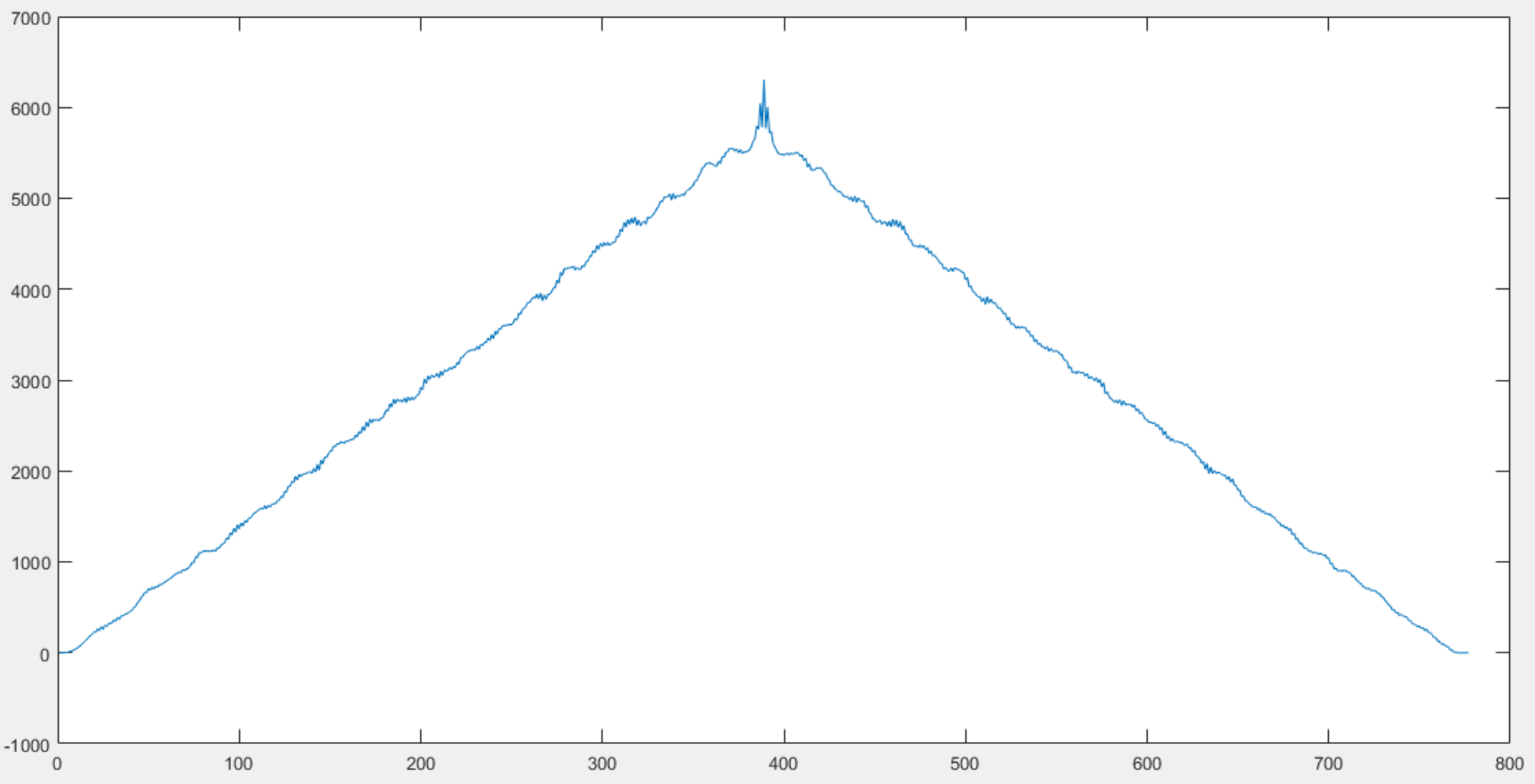
6. MATLAB code for Simulation using Fixed Point Numbers:

```
memy1 = bin2dec(num2str(dlmread('memoryy.list')))/2^8;
y1 = double(fi(memy1,0,16,8));

memy2 = bin2dec(num2str(dlmread('memoryyMATLAB.list')))/2^8;
y2 = double(fi(memy2,0,16,8));

plot(y1);
hold
plot(y2);
legend('Y[n] From Xilinx','Y[n] From Fixed Point Toolbox')
cross      = xcorr(y1,y2);
cross_coeff = max(crosscorr(y1,y2));
RMS   = sqrt(mse(y1,y2));
figure,plot(cross);
```



Simulation Plots Between Y[n] obtained from Xilinx and from Fixed Point Toolbox


Y[n] From Xilinx
Y[n] From Fixed Point Toolbox
0
1
2
3
4
5
6
0
10
20
30
40
50
60
70
80
90



Cross correlation between both input and output

CHECKING OF SIMILARITY BETWEEN TWO WAVEFORMS DEPENDS UPON THE FOLLOWING PARAMETERS WHICH ARE ROOT MEAN SQUARE ERROR, AND CO-EFFICIENT OF CROSS CORRELATION

MSE is calculated from MATLAB mse() function and then we get RMS value by taking square root which is given by:

```
>> RMS


RMS =


   4.0555
```

Both waveforms are highly correlated as they give very high coefficient of cross correlation:

```
>> cross_coeff


cross_coeff =


   0.9617
```