2. EXPERIMENTATION

CHAPTER 2.1: GOLDEN REFERANCE

Generating original Tx signal:

Initial setup gives a clear console, clears all the preset values of variables and closes all the figures.

```
% Initial Setup
clear; clc; close all;
```

A random pulse has to be generated to visualize an ideal working of the algorithm. The pulse of signal is designed to be of the length of 64 bauds and the pulse is names *theta.* All the bauds would hold the value between 0 and 1 as double [8 byte] floating point values. The signal *pulse* generates a complex double of *theta* with real and imaginary components.

```
% Create pulse to detect

rng('default');%random variable will be selected using default algorithm
PulseLen = 64;%64 bits will be the length of the pulse
theta = rand(PulseLen,1);%an array is generated with size 64x1 of values 0 to 1
pulse = exp(1i*2*pi*theta);%the values are converted into complex form
```

The total length of the pulse to be transmitted is of 5000 bauds which consists of the actual 64 bauds peak of the pulse. The *PulseLoc* variable is randomly generated to add the peak to the generated location. The entire array of *TxSignal* is given the value zero upon which the *pulse* signal is overlaid.

```
% Insert pulse to Tx signal

rng('shuffle');%random variable will be selected using default algorithm
TxLen = 5000;%total transmition length will be 5000 bits
PulseLoc = randi(TxLen-PulseLen*2);% defining where to insert the pulse

TxSignal = zeros(TxLen,1);% the rest 5000-
64 = 4936 bits should be zero while transmitting
```

```
TxSignal(PulseLoc:PulseLoc+PulseLen1) = pulse; %merging both pulse to
5000 bits chai
```

Generated TxSignal: (the peak attained in this diagram is subjected to change for every run).

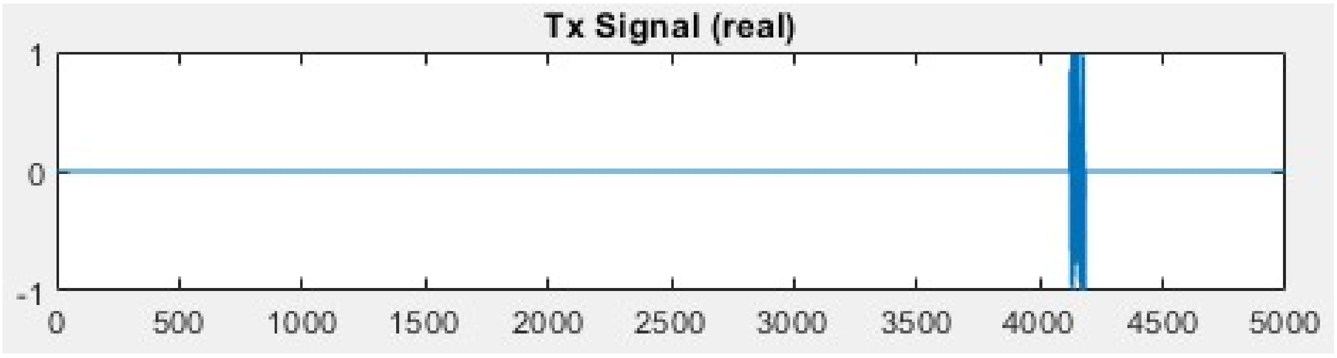


Fig 2.1- Original Tx Signal

The *TxSignal* attained so far is the signal that is required at the receiver end but to mimic the actual transmission of the signal AWGN is added to the signal and will be given a name *RxSignal.*

Adding noise to the original signal:

Noise signal holds a value random complex numbers which would be added onto the *TxSignal.* The *RxSignal* is scaled to fit the entire range between [-1, 1].

```
% Create Rx signal by adding noise

Noise = complex(randn(TxLen,1),randn(TxLen,1));%random complex numbers
RxSignal = TxSignal + Noise; % addding noise to signal

% Scale Rx signal to +/- one

scale1 = max([abs(real(RxSignal)); abs(imag(RxSignal))]); %converting it for conv
inience
RxSignal = RxSignal/scale1;%dividing it by max value of signal
```

Generated RxSignal :

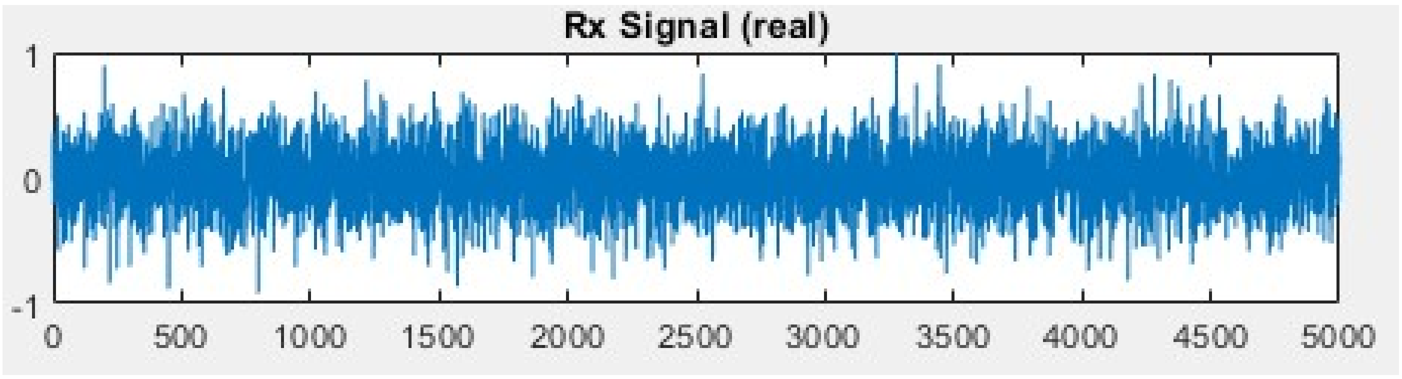


Fig 2.2- Noise added Rx Signal

Retrieving original signal:

A Correlation filter is used to retrieve back the algorithm. Where the conjugate of the inversed pulse is provided as *CorrFilter* signal. The task is to find the position of the pulse. The *FilterOut* Signal correlates the *RxSignal* against the matched filter to find the location of the signal, which also finds the peak value of the *pulse*.

```
% Create matched filter coefficients
CorrFilter = conj(flip(pulse))/PulseLen;%step 1 of corelation filter

% Correlate Rx signal against matched filter
FilterOut = filter(CorrFilter,1,RxSignal);%step 2 of corelation filter

% Find peak magnitude & location
[peak, location] = max(abs(FilterOut));%getting the value of peak at filter
output
```

The output after the filter.

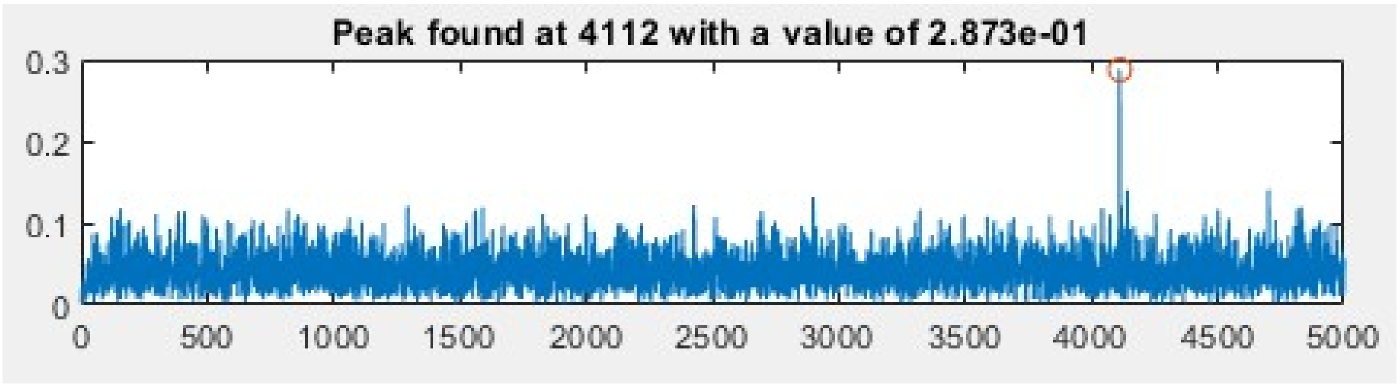


Fig 2.3- Reconstructed Rx Signal

The following commands are used to print the values and the graphs in the output screen.

```matlab
% Print results
figure(1)%creating a figure
subplot(311); plot(real(TxSignal)); title('Tx Signal (real)');%the signal without noise
subplot(312); plot(real(RxSignal)); title('Rx Signal (real)');%the signal with noise

t = 1:length(FilterOut);%length of signal
str = sprintf('Peak found at %d with a value of %.3d',location,peak);%print the value of peak
subplot(313); plot(t,abs(FilterOut),location,peak,'o'); title(str);%the filtered signal
```

The above used code is used as the golden reference to stimulate the same using a hardware friendly model using **simulink**.

CHAPTER 2.2: HARDWARE FRIENDLY MODEL

Window Creation:

Measuring 5000 bauds can be done with ease by compiler at Matlab but to make it hardware applicable a window method is used where, a window of 11 bauds will be taken at a time and the middle baud (the 6th one) is checked to exceed the threshold value.

```
% Implementing the same above mentioned algorithm in a hardware friendly manner.

WindowLen = 11;%comparing the 5000 bits with 11 bits at a time serially
MidIdx = ceil(WindowLen/2);%gets the value of 6th bit in the series
threshold = 0.03;%only the peak would be more than this value.

%note: for convenience the threshold value was adjusted from experimental
       %trial and error method
```

Attaining a square root of a complex variable could increase the complexity of the circuit, hence *MagSqOut* signal is used such that the absolute value of the signal is squared hence making calculations easier.

```
% Compute magnitude squared to avoid sqrt operation
MagSqOut = abs(FilterOut).^2;
```

The Window of 5000 bauds is let to slide comparing the threshold value at each iteration. The *MidSample* and *CompareOut* confirms that the middle value satisfies the conditions only at the start of the actual pulse.

```
% Sliding window operation
for n = 1:length(FilterOut)-WindowLen %1 to 4989

% Compare each value in the window to the middle sample via subtraction
    DataBuff = MagSqOut(n:n+WindowLen-1);%stream of 11 bits
    MidSample = DataBuff(MidIdx);%bit 6 of the sample
    CompareOut = DataBuff - MidSample; % this is a vector


 % if all values in the result are negative and the middle sample is
    % greater than a threshold, it is a local max
    if all(CompareOut <= 0) && (MidSample > threshold)
        peak_2 = MidSample;
        location_2 = n + (MidIdx-1);%gives the peak location
    end
end
```

Fixed point insertion:

A fixed point is much feasible to use in a hardware model than a floating point. Hence tools from Fixed point designer toolbox is used to convert the values into fixed point sacrificing precision for bandwidth.

```
% Simulate model in fixed-point or floating-point
fxpt_mode = true;
if fxpt_mode  % fixed-point
    DT_input = fixdt(1,16,14);%fixdt(signed/unsigned',wordlength,fractionlengt
h)
    DT_filter = fixdt(1,18,15);
    DT_power = fixdt(1,18,11);
else  % floating-point
    DT_input = 'double';%64 bit floating point
    DT_filter = 'double';
    DT_power = 'double';
end
DT_coeff = fixdt(1,18); % coeff is treated as double if input is double
```

Converting column vector to row.

```
if iscolumn(CorrFilter)
    CorrFilter = transpose(CorrFilter); % need row vector for filter block
end
```

SimTime fixes the time for which the model will run. And slout instantiate the project.

```
%stimulation time to be length of the signal plus 30 time units
%30 time units= 10 in front end + 10 in back end + 10 buffer
SimTime = length(RxSignal) + WindowLen + 30;

% Simulate model
slout = sim('project_detect');
```

Two functions have been instantiated which will be called multiple times during simulation.

```
% Correlation filter output
FilterOutSL = getLogged(slout,'filter_out');
FilterValid = getLogged(slout,'filter_valid');
FilterOutSL = FilterOutSL(FilterValid);%signal through filter

%to compare and print logged signal values
compareData(real(FilterOut),real(FilterOutSL),{2 3 1},'ML vs SL correlator output (re)');
compareData(imag(FilterOut),imag(FilterOutSL),{2 3 2},'ML vs SL correlator output (im)');

% Magnitude squared output
MagSqSL = getLogged(slout,'mag_sq_out');
MagSqSL = MagSqSL(FilterValid);%magnitude square of filter

%to compare and print logged signal values
compareData(MagSqOut,MagSqSL,{2 3 3},'ML vs SL mag-squared output');

% Peak value
MidSampleSL = getLogged(slout,'mid_sample');
Detected = getLogged(slout,'detected');
PeakSL = double(MidSampleSL(Detected>0));
```

The output of the code is displayed in the figure using print commands.

```
fprintf('\nPeak location = %d, magnitude = %.3d using global max\n',location,peak);
fprintf('Peak location = %d, mag-squared = %.3d using local max\n',location_2,peak_2);
fprintf('Peak mag-squared from Simulink = %.3d, error = %.3d\n',PeakSL,abs(peak_2-
PeakSL));
```

Output:

- The red legend in the output is the measure of deviation of the simulink model's output to that of the Matlab's golden reference, which stays at zero indicating perfect working of the hardware model.
- Yellow legend is hidden in the graph as the blue legend overlaps the reference plot.
- The error is in terms of 10^{-6} is caused due to the fixed point conversion, which could be neglected.

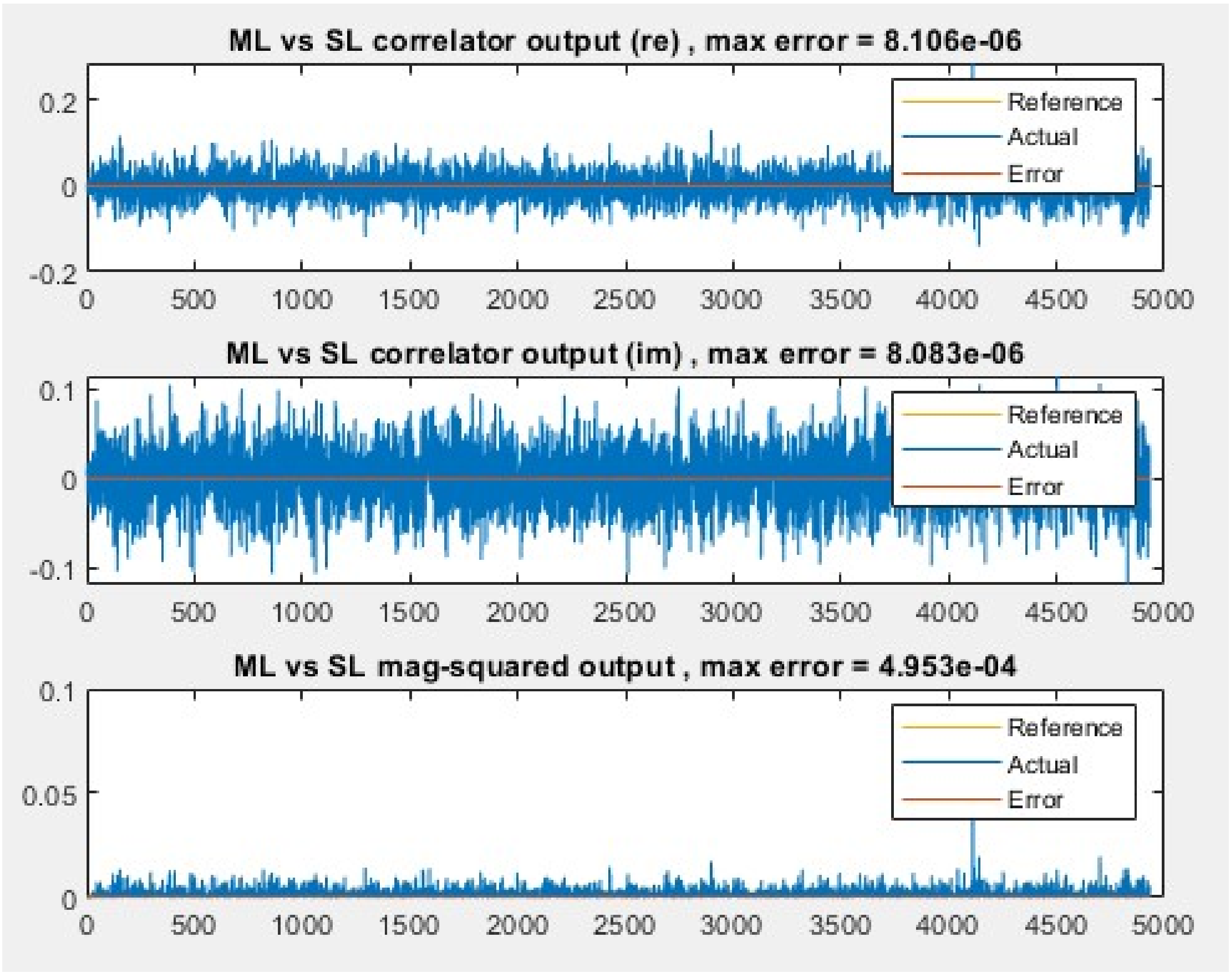
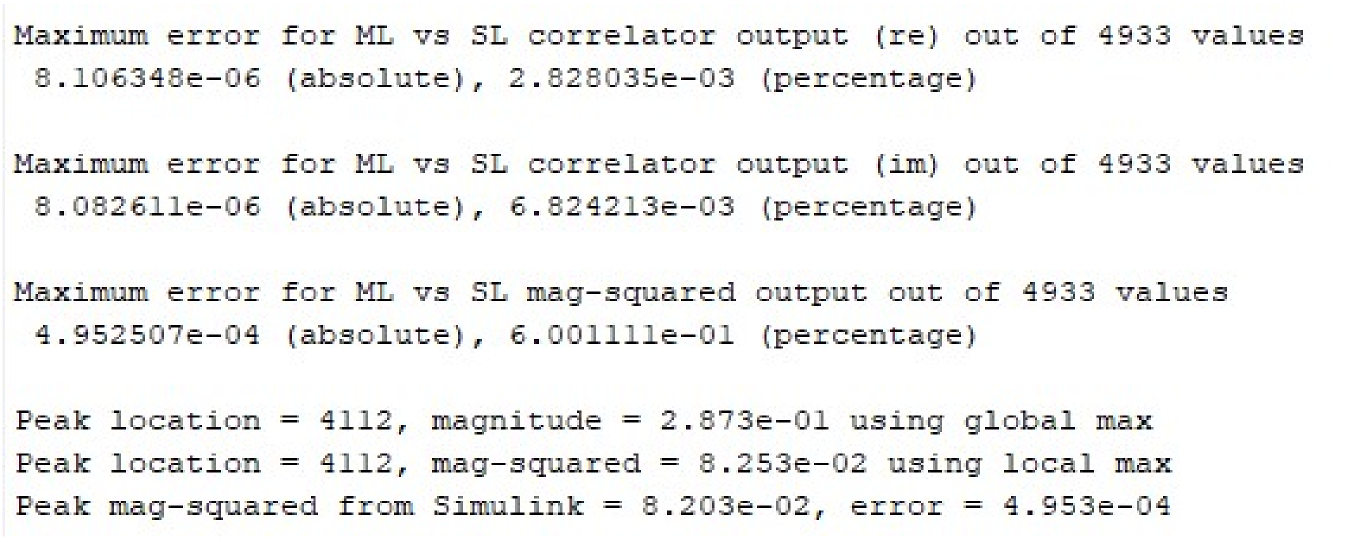


Fig 2.4- Rx and Tx Signals through hardware friendly code



```
Maximum error for ML vs SL correlator output (re) out of 4933 values
 8.106348e-06 (absolute), 2.828035e-03 (percentage)

Maximum error for ML vs SL correlator output (im) out of 4933 values
 8.082611e-06 (absolute), 6.824213e-03 (percentage)

Maximum error for ML vs SL mag-squared output out of 4933 values
 4.952507e-04 (absolute), 6.001111e-01 (percentage)

Peak location = 4112, magnitude = 2.873e-01 using global max
Peak location = 4112, mag-squared = 8.253e-02 using local max
Peak mag-squared from Simulink = 8.203e-02, error = 4.953e-04
```

Fig 2.5- Command Window Output

CHAPTER 2.3: FUNCTION BLOCKS

The two functions used in the above programs:

1. Getlogged:

Squeeze removes a dimension form the array. This function is used to log all the values in the simulink model.

Empty signal is also eliminated and it triggers an error while that happens. logsout, getElement are key triggers used to transfer the signal for one sub block to other.

```
%%Defining the functions used in the code above

%getLogged function
function signal_val = getLogged(simout_obj,signal_name)%function instantiation

logsout = simout_obj.logsout;%load logged signal

%checking for possible error in outputs
if isempty(logsout)
    error('No logged signal found. Make sure ''%s'' is logged in the model',..
.
        signal_name);
end

sig = logsout.getElement(signal_name);%read elements of logged signal

%checking for possible error in outputs
if isempty(sig)
    error('Signal ''%s'' not found. Make sure it is logged and named correctly
.',...
        signal_name);
end

signal_val = squeeze(sig.Values.Data);
%squeeze returns an array with the same elements as the input,...
%but with dimensions of length 1 removed.

end
```

2. Compare data:

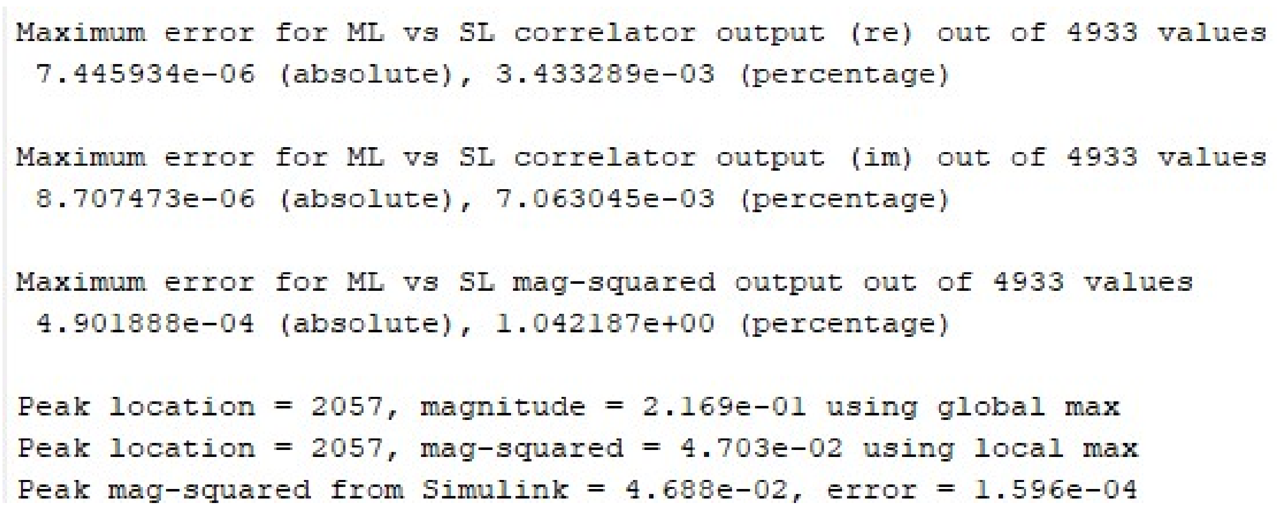
This function doesn’t affect the hardware model but is used to find the differences between both, hence would help us solve the bugs.

```
%compareData function
%function instantiation
function err_vec = compareData(reference,actual,figure_number,textstring)

% Vector input only
if ~isvector(reference) || ~isvector(actual)
    error('Input signals must be vector');%check for inputs to be vector
else
    if isrow(reference)
        reference = transpose(reference);%convert to column vector
    end
    if isrow(actual)
        actual = transpose(actual);%convert to column vector
    end
end

% Make signals same length if necessary
if length(reference) ~= length(actual)
%     warning(['Length of reference (%d) is not the same as actual signal (%d)
.'...
%         ' Truncating the longer input.'],length(reference),length(actual));

    %balance length of both the vectors to be equal
    len = 1:min(length(reference),length(actual));
    reference = reference(len);%length modification of reference
    actual = actual(len);%length modification of actual
end

% Turn complex into vector
if xor(isreal(reference),isreal(actual))%checking for nature of both signals
    error('Input signals are not both real or both complex');
elseif ~isreal(reference)
    ref_vec = double([real(reference) imag(reference)]);%covert to double
    act_vec = double([real(actual) imag(actual)]);%covert to double
    tag = {'(Real)','(Imag)'};
else
    ref_vec = double(reference);%covert to double
    act_vec = double(actual);%covert to double
    tag = {''};
end

% Configure figure
if iscell(figure_number)
    if size(ref_vec,2) > 1 % complex
        error('Cannot yet subplot multiple complex inputs');
    else
        figure(figure_number{1})%defines figure(2)
    end
else
    figure(figure_number)
end
c = get(groot,'defaultAxesColorOrder');%used at plotting

% Compute error
err_vec = ref_vec - act_vec;%error vector to be printed on output
max_err = max(abs(err_vec));%max error to be printed on output
max_ref = max(abs(ref_vec));%max reference to be printed on output
```

The calculated output is sent back to the main screen using print commands.

```
%output printed on command window
fprintf('\nMaximum error for %s out of %d values\n',textstring,length(actual))
;

for n = 1:size(ref_vec,2)

    %output printed on Command window
    fprintf('%s %d (absolute), %d (percentage)\n',tag{n},max_err(n),max_err(n)
/max_ref(n)*100);

    if iscell(figure_number)
        total_plot = figure_number{2};% equal to 3
        plot_num = figure_number{3};%figure number can be 1 to 3

    else
        total_plot = size(ref_vec,2);
        plot_num = n;
    end
    subplot(total_plot,1,plot_num)%subplot generation

    plot(ref_vec(:,n),'Color',c(3,:));%uses 1st color for reference

    hold on %plots 3 graphs over the same plot

    plot(act_vec(:,n),'Color',c(1,:));%uses 2nd color for actual
    plot(err_vec(:,n),'Color',c(2,:));%uses 3rd color for error

    legend('Reference','Actual','Error')%defining legend
    hold off

    title(sprintf('%s %s, max error = %.3d',textstring,tag{n},max_err(n)));%ti
tle
end
end
```

Command prompt output:



```
Maximum error for ML vs SL correlator output (re) out of 4933 values
 7.445934e-06 (absolute), 3.433289e-03 (percentage)

Maximum error for ML vs SL correlator output (im) out of 4933 values
 8.707473e-06 (absolute), 7.063045e-03 (percentage)

Maximum error for ML vs SL mag-squared output out of 4933 values
 4.901888e-04 (absolute), 1.042187e+00 (percentage)

Peak location = 2057, magnitude = 2.169e-01 using global max
Peak location = 2057, mag-squared = 4.703e-02 using local max
Peak mag-squared from Simulink = 4.688e-02, error = 1.596e-04
```

Fig 2.6- Command Window Output

CHAPTER 2.4: SIMULINK MODEL

Top model:

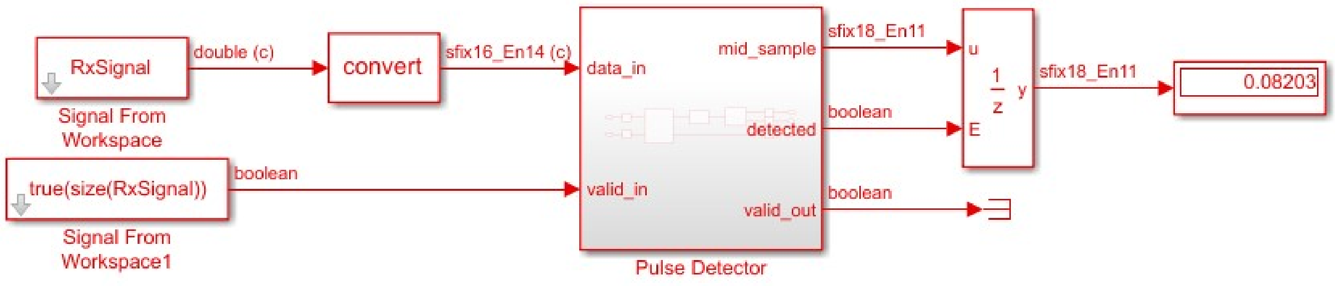


Fig 2.7- Block diagram of the top module

The *RxSignal* is given in parallel to valid in signal to trigger the incoming of received signal. The convert block converts the double precision floating point value to fixed point value. The output is displayed in the output block which is controlled by a unit delay block to avoid glitches.

Pulse Detector Top module:

This block will be the main module in the synthesis of RTL and other sub blocks will be instantiated from this top module.

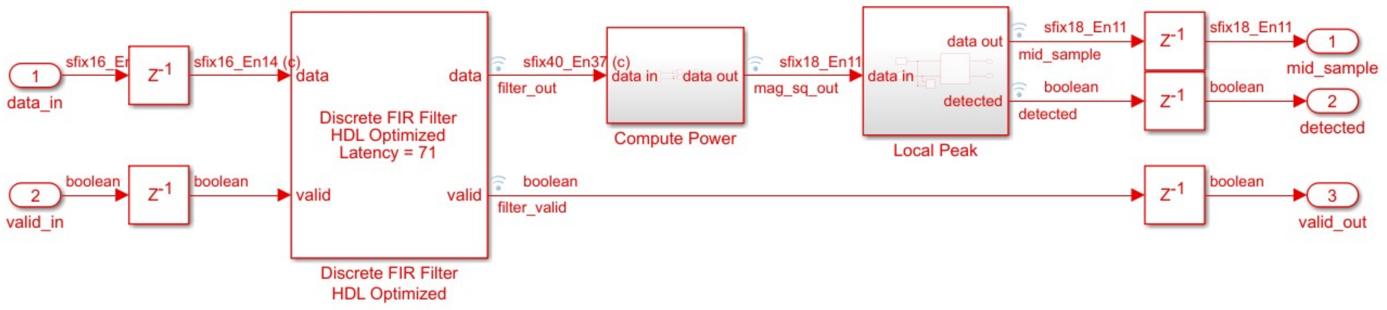


Fig 2.8- Block diagram of the pulse detector module

The signal is fed into the discrete FIR filter which then sends out the filter out signal. The filter_out signal is logged into Compute power sub block.

Compute Power sub module:

The incoming fixed point value is converted into less precision value of the same and is squared in a hardware friendly manner. The result is then logged to the next sub block.

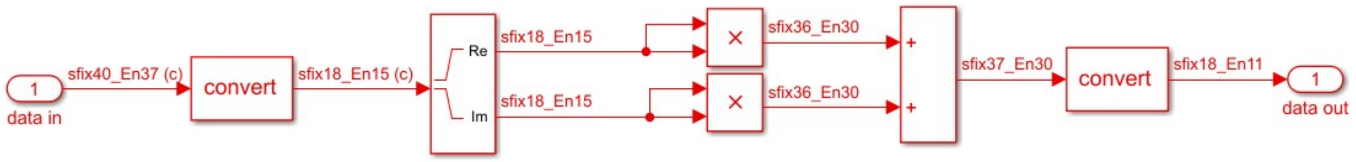


Fig 2.9- Block diagram of compute power sub module

Local Peak sub module:

11 values are collected using the delay block which is connected to same DT block which checks if the connected inputs are of same data type. If not it forces all the input to be of same type.

The fcn block holds a matlab code, which checks for the peak, if detected will send out the data in data out port and sets the detected signal to be high. If not the window is updated to receive the next bit contnuing till detection of the peak.

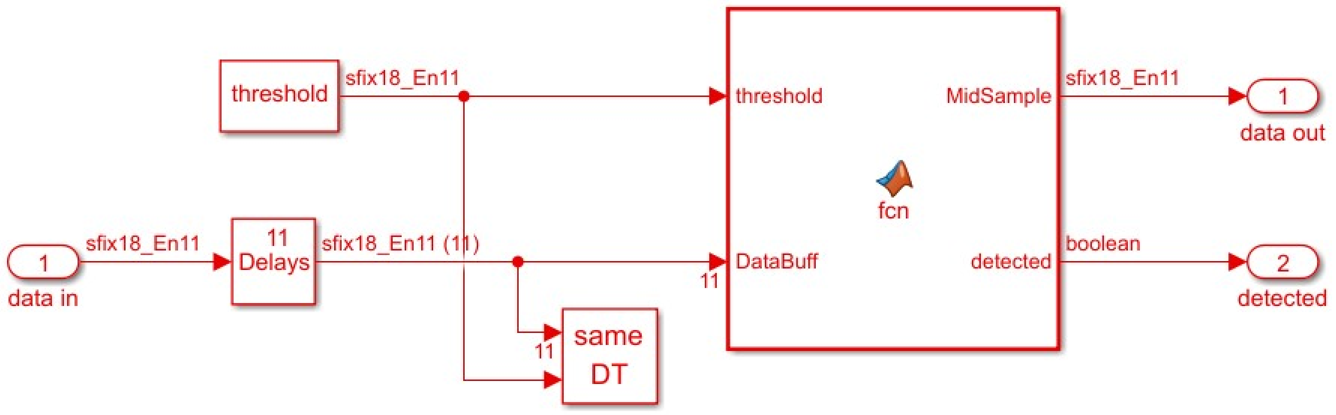


Fig 2.10- Block diagram of local peak module

fcn code:

```matlab
% Hardware friendly implementation of peak finder
%
% Function inputs:
% * WindowLen - non-tunable parameter defined under Simulink->Edit Data
% * threhold  - input port (connected to constant)
% * DataBuff  - input port (buffering done using Simulink block)
%
% Function outputs:
% * "detected" is set when MidSample is local max

function [MidSample,detected] = fcn(threshold, DataBuff, WindowLen)
%#codegen


MidIdx = ceil(WindowLen/2);

% Compare each value in the window to the middle sample via subtraction
MidSample = DataBuff(MidIdx);
CompareOut = DataBuff - MidSample; % this is a vector

% if all values in the result are negative and the middle sample is
% greater than a threshold, it is a local max
if all(CompareOut <= 0) && (MidSample > threshold)
    detected = true;
else
    detected = false;
end
```

This mimics the part of window code in Matlab but has few keywords converted to hardware friendly Boolean terms such as true and false.

CHAPTER 2.5: HDL CODER

HDL Coder:

The DUT compatibility is checked using HDL coder, checker not only ensures the blocks used in the subsystem is real compatible but also ensures the settings, ports and configurations of these blocks do not generate inefficient hardware. The HDL model checker includes options for native floating-point and industry standard checks.

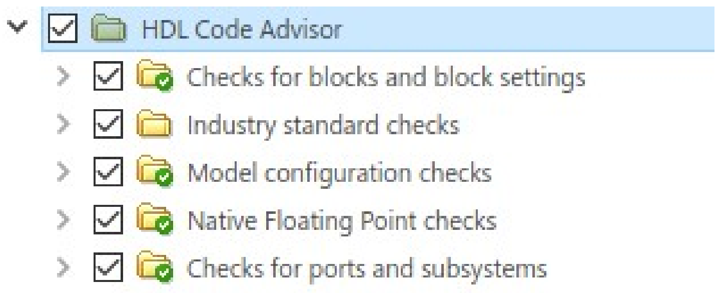
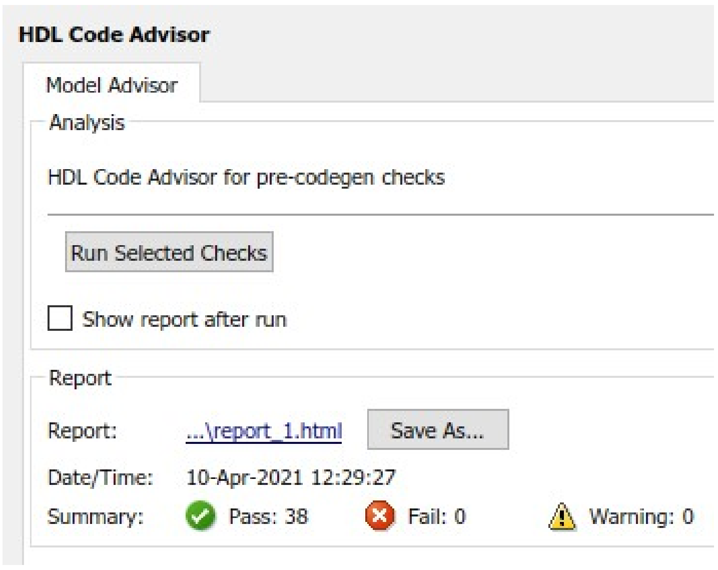

HDL Code Advisor
Checks for blocks and block settings
Industry standard checks
Model configuration checks
Native Floating Point checks
Checks for ports and subsystems

Fig 2.11- HDL Coder Compatibility check



HDL Code Advisor for pre-codegen checks

Report: ...\report_1.html

Date/Time: 10-Apr-2021 12:29:27

Summary: Pass: 38 Fail: 0 Warning: 0

Fig 2.12- HDL Code Advisor Output

Once the checks are successful, the HDL workflow advisor has to be run to initiate generation of HDL code. After generation of HDL code, Quartus prime is used to visualize the resources and RTL design.

3. RESULTS AND OUTPUTS

CHAPTER 3.1: RTL DESIGN OF MODULES

Pulse Detector module:

a)

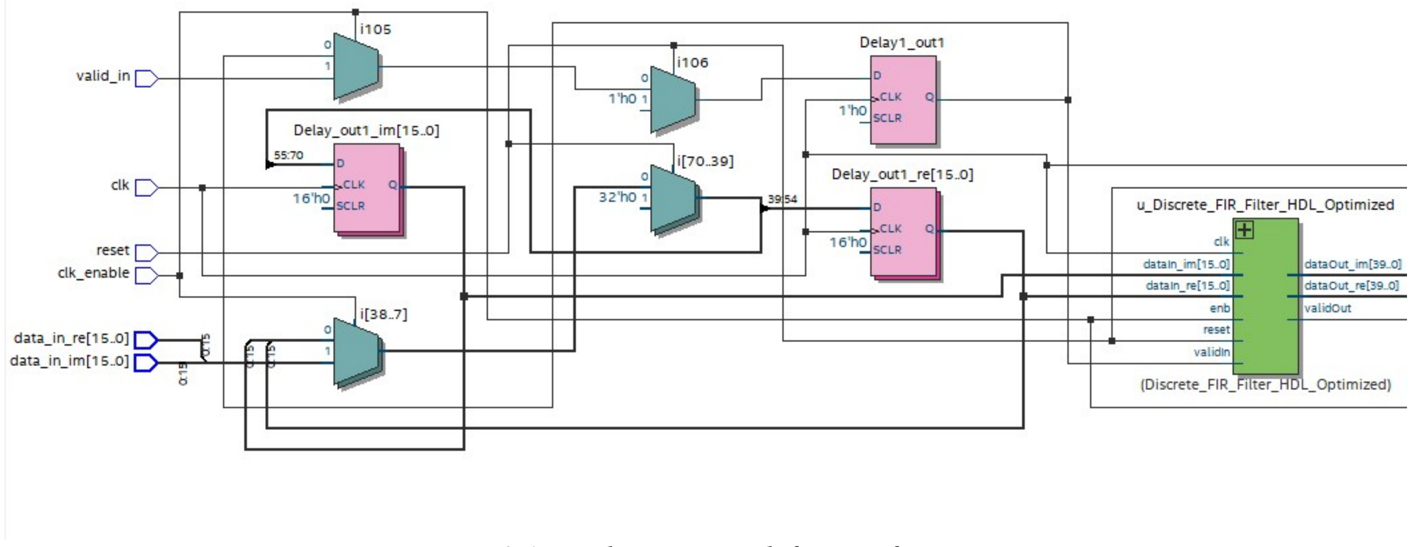


Fig 3.1 – Pulse Detector left part of RTL

b)

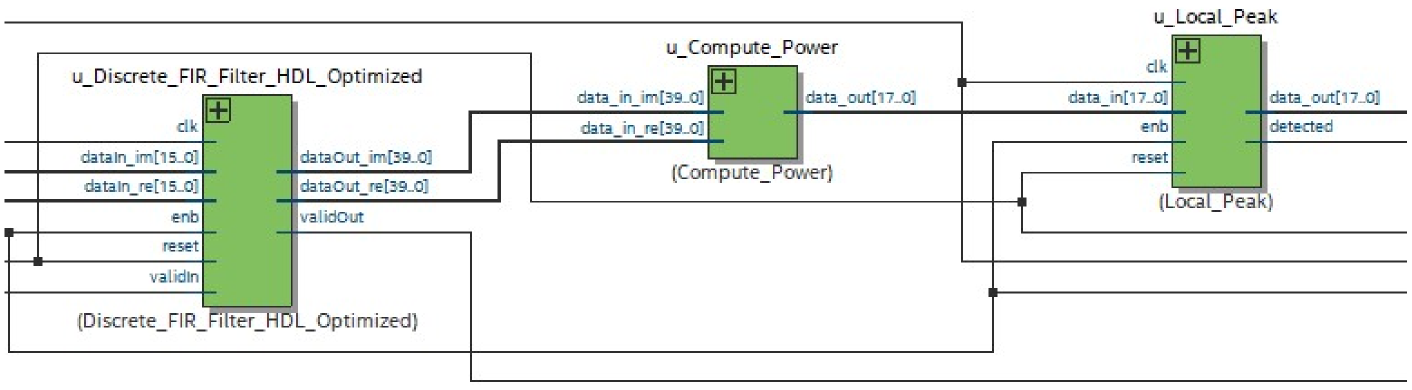


Fig 3.2 – Pulse Detector middle part of RTL

c)

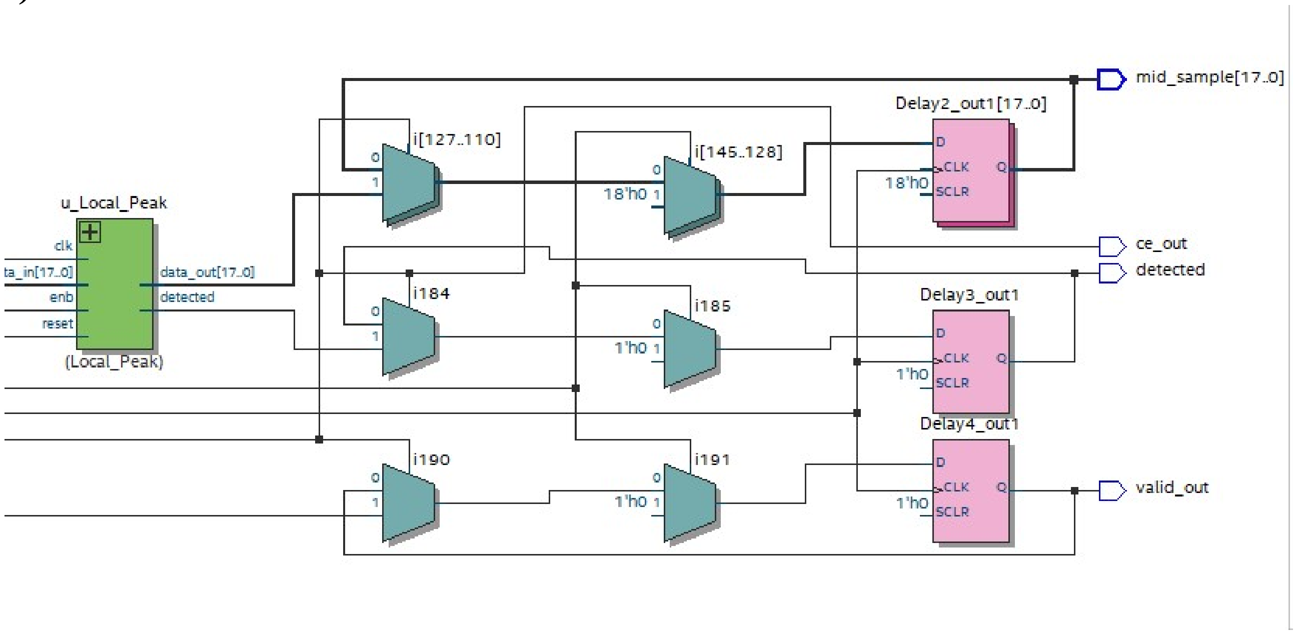


Fig 3.3 – Pulse Detector right part of RTL

FIR filter sub module:

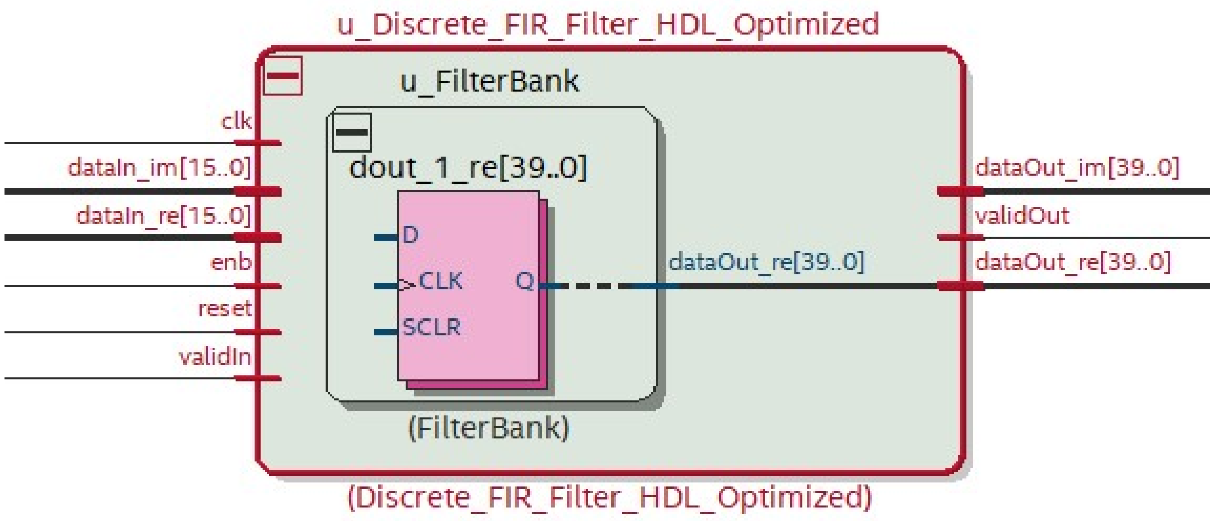


Fig 3.4 –RTL Design of Filter sub module

Compute Power sub module:

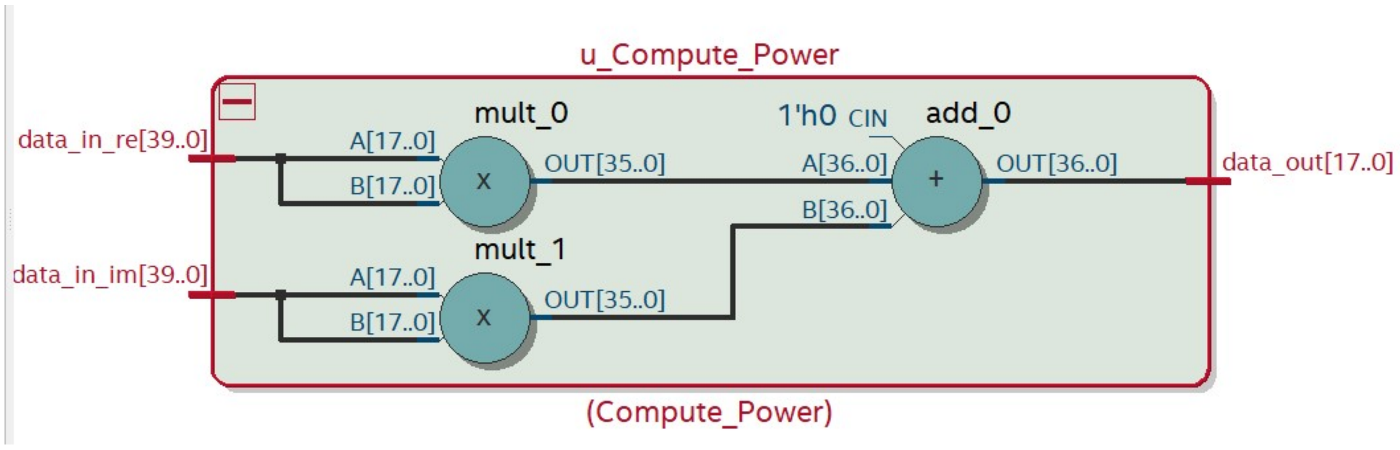


Fig 3.5 –RTL Design of Compute Power sub module

Local peak sub module:

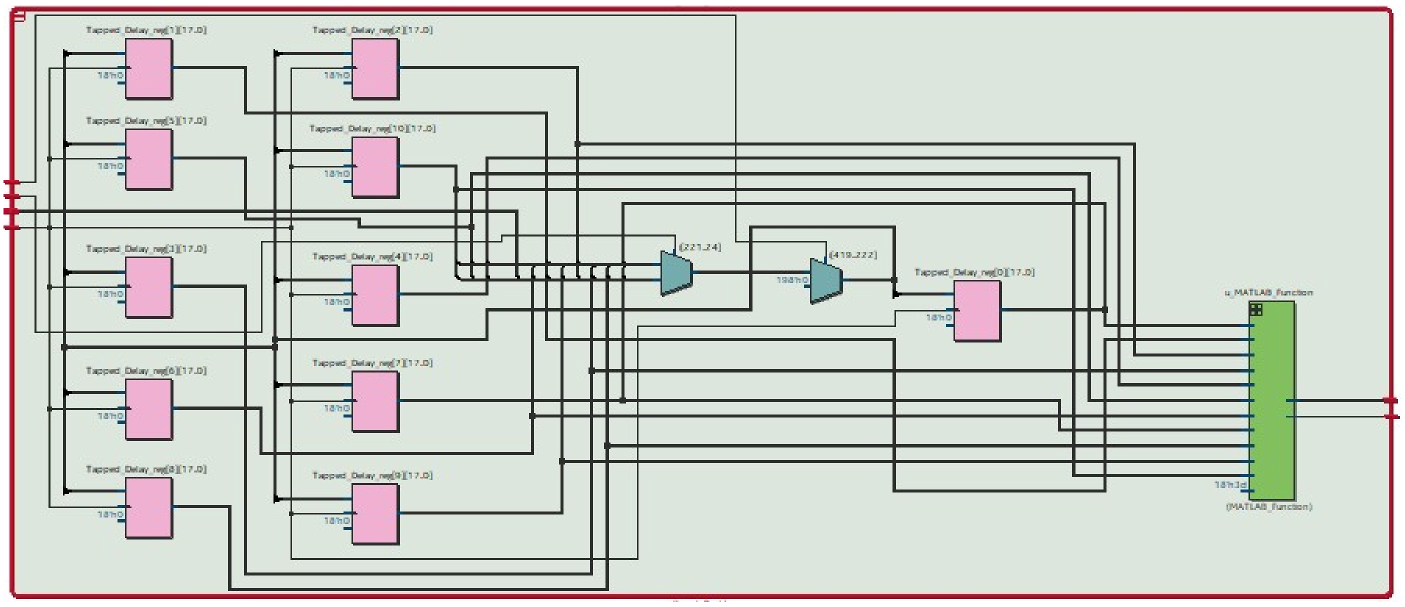

Tapped_Delay_reg[1][17..0]
Tapped_Delay_reg[2][17..0]
Tapped_Delay_reg[5][17..0]
Tapped_Delay_reg[10][17..0]
Tapped_Delay_reg[3][17..0]
Tapped_Delay_reg[4][17..0]
Tapped_Delay_reg[6][17..0]
Tapped_Delay_reg[7][17..0]
Tapped_Delay_reg[8][17..0]
Tapped_Delay_reg[9][17..0]
Tapped_Delay_reg[0][17..0]
18'h0
[221..24]
[419..222]
198'h0
u_MATLAB_function
18'h3d
(MATLAB_function)

Fig 3.6 –RTL Design of Local Peak sub module

MATLAB function sub module:

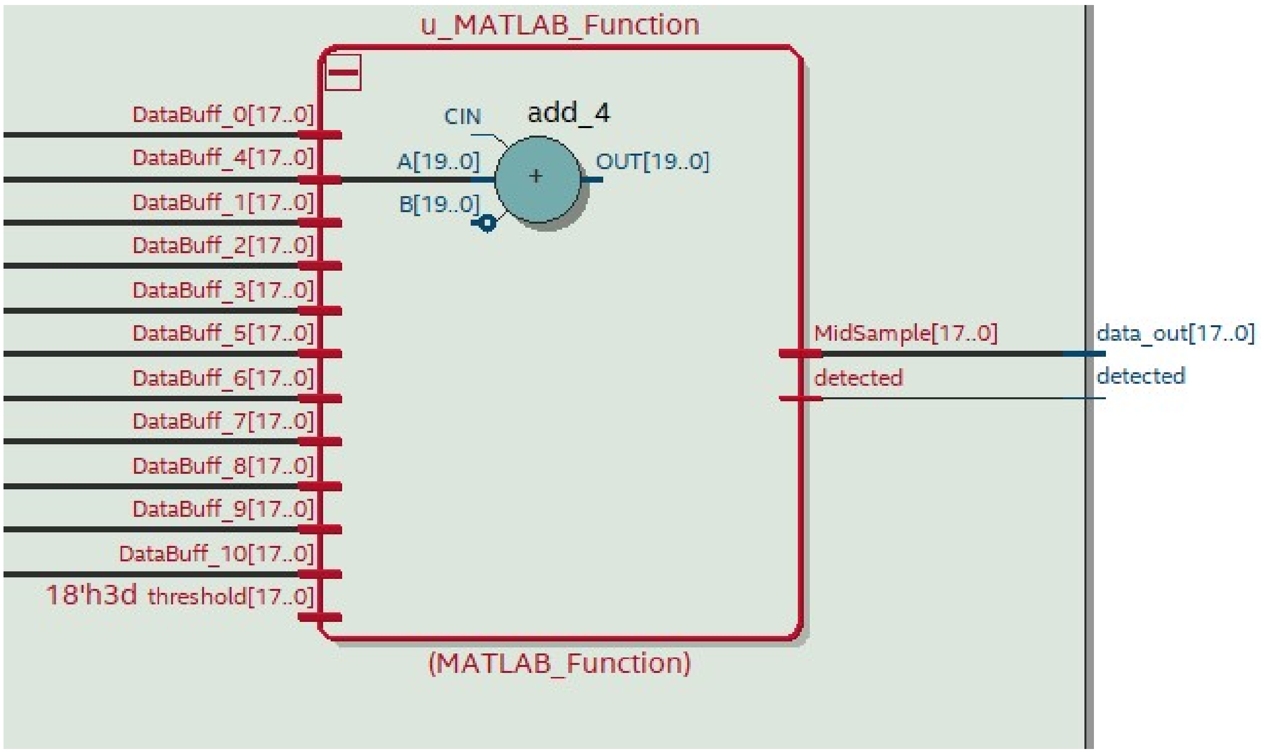
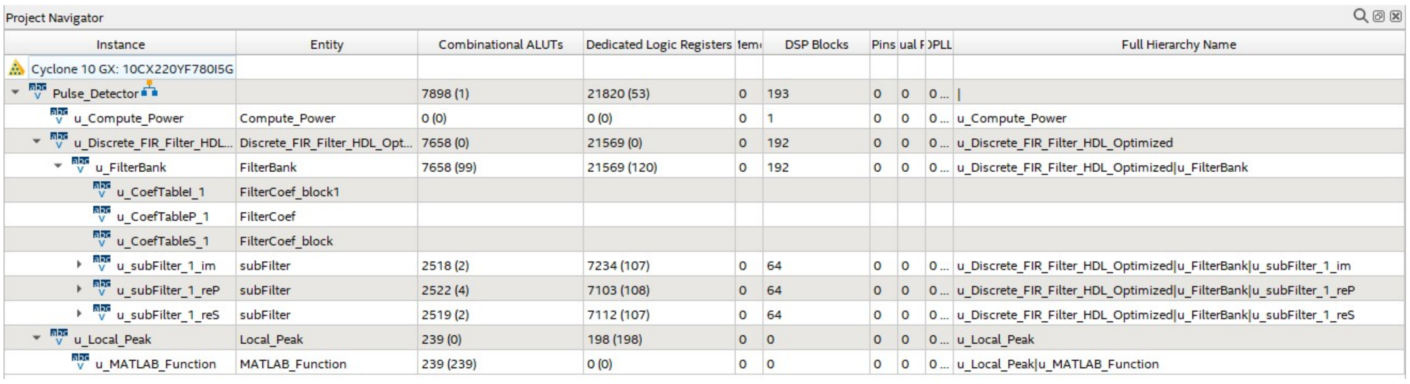


Fig 3.7 –RTL Design of MATLAB function sub module

Modules hierarchy and resource utilization:


Project Navigator

Instance	Entity	Combinational ALUTs	Dedicated Logic Registers	1em	DSP Blocks	Pins	ual F	)PLL	Full Hierarchy Name
Cyclone 10 GX: 10CX220YF780I5G									
Pulse_Detector		7898 (1)	21820 (53)	0	193	0	0	0 ...	|
u_Compute_Power	Compute_Power	0 (0)	0 (0)	0	1	0	0	0 ...	u_Compute_Power
u_Discrete_FIR_Filter_HDL...	Discrete_FIR_Filter_HDL_Opt...	7658 (0)	21569 (0)	0	192	0	0	0 ...	u_Discrete_FIR_Filter_HDL_Optimized
u_FilterBank	FilterBank	7658 (99)	21569 (120)	0	192	0	0	0 ...	u_Discrete_FIR_Filter_HDL_Optimized|u_FilterBank
u_CoefTableI_1	FilterCoef_block1								
u_CoefTableP_1	FilterCoef								
u_CoefTableS_1	FilterCoef_block								
u_subFilter_1_im	subFilter	2518 (2)	7234 (107)	0	64	0	0	0 ...	u_Discrete_FIR_Filter_HDL_Optimized|u_FilterBank|u_subFilter_1_im
u_subFilter_1_reP	subFilter	2522 (4)	7103 (108)	0	64	0	0	0 ...	u_Discrete_FIR_Filter_HDL_Optimized|u_FilterBank|u_subFilter_1_reP
u_subFilter_1_reS	subFilter	2519 (2)	7112 (107)	0	64	0	0	0 ...	u_Discrete_FIR_Filter_HDL_Optimized|u_FilterBank|u_subFilter_1_reS
u_Local_Peak	Local_Peak	239 (0)	198 (198)	0	0	0	0	0 ...	u_Local_Peak
u_MATLAB_Function	MATLAB_Function	239 (239)	0 (0)	0	0	0	0	0 ...	u_Local_Peak|u_MATLAB_Function

Fig 3.8 –Resource utilization details

CHAPTER 3.2: VERILOG CODES

Pulse detector:

```
`timescale 1 ns / 1 ns

module Pulse_Detector
          (clk,
           reset,
           clk_enable,
           data_in_re,
           data_in_im,
           valid_in,
           ce_out,
           mid_sample,
           detected,
           valid_out);


  input   clk;
  input   reset;
  input   clk_enable;
  input   signed [15:0] data_in_re;  // sfix16_En14
  input   signed [15:0] data_in_im;  // sfix16_En14
  input   valid_in;
  output  ce_out;
  output  signed [17:0] mid_sample;  // sfix18_En11
  output  detected;
  output  valid_out;


  wire enb;
  reg signed [15:0] Delay_out1_re;  // sfix16_En14
  reg signed [15:0] Delay_out1_im;  // sfix16_En14
  reg  Delay1_out1;
  wire signed [39:0] filter_out_re;  // sfix40_En37
  wire signed [39:0] filter_out_im;  // sfix40_En37
  wire filter_valid;
  wire signed [17:0] mag_sq_out;  // sfix18_En11
  wire signed [17:0] mid_sample_1;  // sfix18_En11
  wire detected_1;
  reg signed [17:0] Delay2_out1;  // sfix18_En11
  reg  Delay3_out1;
  reg  Delay4_out1;


  assign enb = clk_enable;

  always @(posedge clk)
    begin : Delay_process
      if (reset == 1'b1) begin
        Delay_out1_re <= 16'sb0000000000000000;
        Delay_out1_im <= 16'sb0000000000000000;
      end
      else begin
        if (enb) begin
          Delay_out1_re <= data_in_re;
          Delay_out1_im <= data_in_im;
        end
      end
    end



  always @(posedge clk)
    begin : Delay1_process
      if (reset == 1'b1) begin
        Delay1_out1 <= 1'b0;
      end
      else begin
        if (enb) begin
          Delay1_out1 <= valid_in;
        end
      end
    end



  Discrete_FIR_Filter_HDL_Optimized u_Discrete_FIR_Filter_HDL_Optimized (.clk(clk),
                                                                         .reset(reset),
                                                                         .enb(clk_enable),
                                                                         .dataIn_re(Delay_out1_re),  // sfix1
6_En14
                                                                         .dataIn_im(Delay_out1_im),  // sfix1
6_En14
                                                                         .validIn(Delay1_out1),
                                                                         .dataOut_re(filter_out_re),  // sfix
40_En37
                                                                         .dataOut_im(filter_out_im),  // sfix
40_En37
                                                                         .validOut(filter_valid)
                                                                         );

  Compute_Power u_Compute_Power (.data_in_re(filter_out_re),  // sfix40_En37
                                 .data_in_im(filter_out_im),  // sfix40_En37
                                 .data_out(mag_sq_out)  // sfix18_En11
                                 );

  Local_Peak u_Local_Peak (.clk(clk),
                           .reset(reset),
                           .enb(clk_enable),
                           .data_in(mag_sq_out),  // sfix18_En11
                           .data_out(mid_sample_1),  // sfix18_En11
                           .detected(detected_1)
                           );

  always @(posedge clk)
    begin : Delay2_process
      if (reset == 1'b1) begin
```

```
        Delay2_out1 <= 18'sb000000000000000000;
      end
      else begin
        if (enb) begin
          Delay2_out1 <= mid_sample_1;
        end
      end
    end



  assign mid_sample = Delay2_out1;

  always @(posedge clk)
    begin : Delay3_process
      if (reset == 1'b1) begin
        Delay3_out1 <= 1'b0;
      end
      else begin
        if (enb) begin
          Delay3_out1 <= detected_1;
        end
      end
    end



  assign detected = Delay3_out1;

  always @(posedge clk)
    begin : Delay4_process
      if (reset == 1'b1) begin
        Delay4_out1 <= 1'b0;
      end
      else begin
        if (enb) begin
          Delay4_out1 <= filter_valid;
        end
      end
    end



  assign valid_out = Delay4_out1;

  assign ce_out = clk_enable;

endmodule  // Pulse_Detector
```

FIR filter:

```
`timescale 1 ns / 1 ns

module Discrete_FIR_Filter_HDL_Optimized
          (clk,
           reset,
           enb,
           dataIn_re,
           dataIn_im,
           validIn,
           dataOut_re,
           dataOut_im,
           validOut);


  input   clk;
  input   reset;
  input   enb;
  input   signed [15:0] dataIn_re;  // sfix16_En14
  input   signed [15:0] dataIn_im;  // sfix16_En14
  input   validIn;
  output  signed [39:0] dataOut_re;  // sfix40_En37
  output  signed [39:0] dataOut_im;  // sfix40_En37
  output  validOut;




  FilterBank u_FilterBank (.clk(clk),
                           .reset(reset),
                           .enb(enb),
                           .dataIn_re(dataIn_re),  // sfix16_En14
                           .dataIn_im(dataIn_im),  // sfix16_En14
                           .validIn(validIn),
                           .dataOut_re(dataOut_re),  // sfix40_En37
                           .dataOut_im(dataOut_im),  // sfix40_En37
                           .validOut(validOut)
                           );

endmodule  // Discrete_FIR_Filter_HDL_Optimized
```

The submodule Filterbank used is attached in the https://github.com/raja-aadhithan/RTL-design-for-MATLAB-model/tree/main/verilog_syntesis page, which consists of further submodules namely : subFilter, FilterCoef_block, FilterCoef_block1.

Compute power:

```verilog
`timescale 1 ns / 1 ns

module Compute_Power
          (data_in_re,
           data_in_im,
           data_out);


  input   signed [39:0] data_in_re;  // sfix40_En37
  input   signed [39:0] data_in_im;  // sfix40_En37
  output  signed [17:0] data_out;  // sfix18_En11


  wire signed [17:0] Data_Type_Conversion1_out1_re;  // sfix18_En15
  wire signed [17:0] Data_Type_Conversion1_out1_im;  // sfix18_En15
  wire signed [35:0] Product_out1;  // sfix36_En30
  wire signed [35:0] Product1_out1;  // sfix36_En30
  wire signed [36:0] Add_add_cast;  // sfix37_En30
  wire signed [36:0] Add_add_cast_1;  // sfix37_En30
  wire signed [36:0] Add_out1;  // sfix37_En30
  wire signed [17:0] Data_Type_Conversion_out1;  // sfix18_En11


  assign Data_Type_Conversion1_out1_re = data_in_re[39:22];
  assign Data_Type_Conversion1_out1_im = data_in_im[39:22];



  assign Product_out1 = Data_Type_Conversion1_out1_re * Data_Type_Conversion1_out1_re;



  assign Product1_out1 = Data_Type_Conversion1_out1_im * Data_Type_Conversion1_out1_im;



  assign Add_add_cast = {Product_out1[35], Product_out1};
  assign Add_add_cast_1 = {Product1_out1[35], Product1_out1};
  assign Add_out1 = Add_add_cast + Add_add_cast_1;


  assign Data_Type_Conversion_out1 = Add_out1[36:19];



  assign data_out = Data_Type_Conversion_out1;

endmodule  // Compute_Power
```

Local Peak:

```verilog
`timescale 1 ns / 1 ns

module Local_Peak
          (clk,
           reset,
           enb,
           data_in,
           data_out,
           detected);


  input   clk;
  input   reset;
  input   enb;
  input   signed [17:0] data_in;  // sfix18_En11
  output  signed [17:0] data_out;  // sfix18_En11
  output  detected;


  wire signed [17:0] Constant_out1;  // sfix18_En11
  reg signed [17:0] Tapped_Delay_reg [0:10];  // sfix18 [11]
  wire signed [17:0] Tapped_Delay_reg_next [0:10];  // sfix18_En11 [11]
  wire signed [17:0] Tapped_Delay_out1 [0:10];  // sfix18_En11 [11]
  wire signed [17:0] MidSample;  // sfix18_En11


  assign Constant_out1 = 18'sb000000000000111101;



  always @(posedge clk)
    begin : Tapped_Delay_process
      if (reset == 1'b1) begin
        Tapped_Delay_reg[0] <= 18'sb000000000000000000;
        Tapped_Delay_reg[1] <= 18'sb000000000000000000;
        Tapped_Delay_reg[2] <= 18'sb000000000000000000;
        Tapped_Delay_reg[3] <= 18'sb000000000000000000;
        Tapped_Delay_reg[4] <= 18'sb000000000000000000;
        Tapped_Delay_reg[5] <= 18'sb000000000000000000;
        Tapped_Delay_reg[6] <= 18'sb000000000000000000;
        Tapped_Delay_reg[7] <= 18'sb000000000000000000;
        Tapped_Delay_reg[8] <= 18'sb000000000000000000;
        Tapped_Delay_reg[9] <= 18'sb000000000000000000;
        Tapped_Delay_reg[10] <= 18'sb000000000000000000;
      end
      else begin
        if (enb) begin
          Tapped_Delay_reg[0] <= Tapped_Delay_reg_next[0];
          Tapped_Delay_reg[1] <= Tapped_Delay_reg_next[1];
          Tapped_Delay_reg[2] <= Tapped_Delay_reg_next[2];
          Tapped_Delay_reg[3] <= Tapped_Delay_reg_next[3];
          Tapped_Delay_reg[4] <= Tapped_Delay_reg_next[4];
          Tapped_Delay_reg[5] <= Tapped_Delay_reg_next[5];
          Tapped_Delay_reg[6] <= Tapped_Delay_reg_next[6];
          Tapped_Delay_reg[7] <= Tapped_Delay_reg_next[7];
          Tapped_Delay_reg[8] <= Tapped_Delay_reg_next[8];
          Tapped_Delay_reg[9] <= Tapped_Delay_reg_next[9];
          Tapped_Delay_reg[10] <= Tapped_Delay_reg_next[10];
        end
      end
    end

  assign Tapped_Delay_out1[0] = Tapped_Delay_reg[0];
  assign Tapped_Delay_out1[1] = Tapped_Delay_reg[1];
  assign Tapped_Delay_out1[2] = Tapped_Delay_reg[2];
  assign Tapped_Delay_out1[3] = Tapped_Delay_reg[3];
  assign Tapped_Delay_out1[4] = Tapped_Delay_reg[4];
  assign Tapped_Delay_out1[5] = Tapped_Delay_reg[5];
  assign Tapped_Delay_out1[6] = Tapped_Delay_reg[6];
  assign Tapped_Delay_out1[7] = Tapped_Delay_reg[7];
  assign Tapped_Delay_out1[8] = Tapped_Delay_reg[8];
  assign Tapped_Delay_out1[9] = Tapped_Delay_reg[9];
  assign Tapped_Delay_out1[10] = Tapped_Delay_reg[10];
  assign Tapped_Delay_reg_next[0] = Tapped_Delay_reg[1];
  assign Tapped_Delay_reg_next[1] = Tapped_Delay_reg[2];
  assign Tapped_Delay_reg_next[2] = Tapped_Delay_reg[3];
  assign Tapped_Delay_reg_next[3] = Tapped_Delay_reg[4];
  assign Tapped_Delay_reg_next[4] = Tapped_Delay_reg[5];
  assign Tapped_Delay_reg_next[5] = Tapped_Delay_reg[6];
  assign Tapped_Delay_reg_next[6] = Tapped_Delay_reg[7];
  assign Tapped_Delay_reg_next[7] = Tapped_Delay_reg[8];
  assign Tapped_Delay_reg_next[8] = Tapped_Delay_reg[9];
  assign Tapped_Delay_reg_next[9] = Tapped_Delay_reg[10];
  assign Tapped_Delay_reg_next[10] = data_in;



  MATLAB_Function u_MATLAB_Function (.threshold(Constant_out1),  // sfix18_En11
                                     .DataBuff_0(Tapped_Delay_out1[0]),  // sfix18_En11
                                     .DataBuff_1(Tapped_Delay_out1[1]),  // sfix18_En11
                                     .DataBuff_2(Tapped_Delay_out1[2]),  // sfix18_En11
                                     .DataBuff_3(Tapped_Delay_out1[3]),  // sfix18_En11
                                     .DataBuff_4(Tapped_Delay_out1[4]),  // sfix18_En11
                                     .DataBuff_5(Tapped_Delay_out1[5]),  // sfix18_En11
                                     .DataBuff_6(Tapped_Delay_out1[6]),  // sfix18_En11
                                     .DataBuff_7(Tapped_Delay_out1[7]),  // sfix18_En11
                                     .DataBuff_8(Tapped_Delay_out1[8]),  // sfix18_En11
                                     .DataBuff_9(Tapped_Delay_out1[9]),  // sfix18_En11
                                     .DataBuff_10(Tapped_Delay_out1[10]),  // sfix18_En11
                                     .MidSample(MidSample),  // sfix18_En11
                                     .detected(detected)
                                     );

  assign data_out = MidSample;

endmodule  // Local_Peak
```

4. CONCLUSION

In this project I have analyzed the strength of MATLAB and Simulink and created the Simulink model of the pulse detection algorithm. Introduced design architecture options that takes a control over speed and area trade-offs and converted the Simulink design to fixed point.

Have generate and synthesize the optimized HDL code. Report generation are done in the HDL code generation. Have visualized and identified the pre and post routing timing information and highlight the critical parts in the model. This analysis used the annotate model which synthesis result option and enables the HDL coder to display the DUT with more accurate critical path timing with the steps in the workflow advisor. Have successfully generated and synthesized the optimized HDL code for the pulse detection algorithm implemented in MATLAB.

5. EDA TOOLS USED

SCRIPTING LANGUAGE: MATLAB
TOOL USED: MATLAB R2020b

SIMULATOR: MODEL SIM (Intel FPGA Started Edition)

SYNTHESIZER: QUARTUS **(**QUARTUS PRIME PRO 20.4)

IDE (BLOCK DIAGRAM): SIMULINK

TOOLBOX USED:

- FIXED POINT DESIGNER
- HDL CODER
- SIGNAL PROCESSING
- DSP SYSTEM

6. FUTURE SCOPE

Current state: The project covers the First half of Front end design of the Chip. Attached below is the work flow of a ASIC/FPGA chip from specification to production.

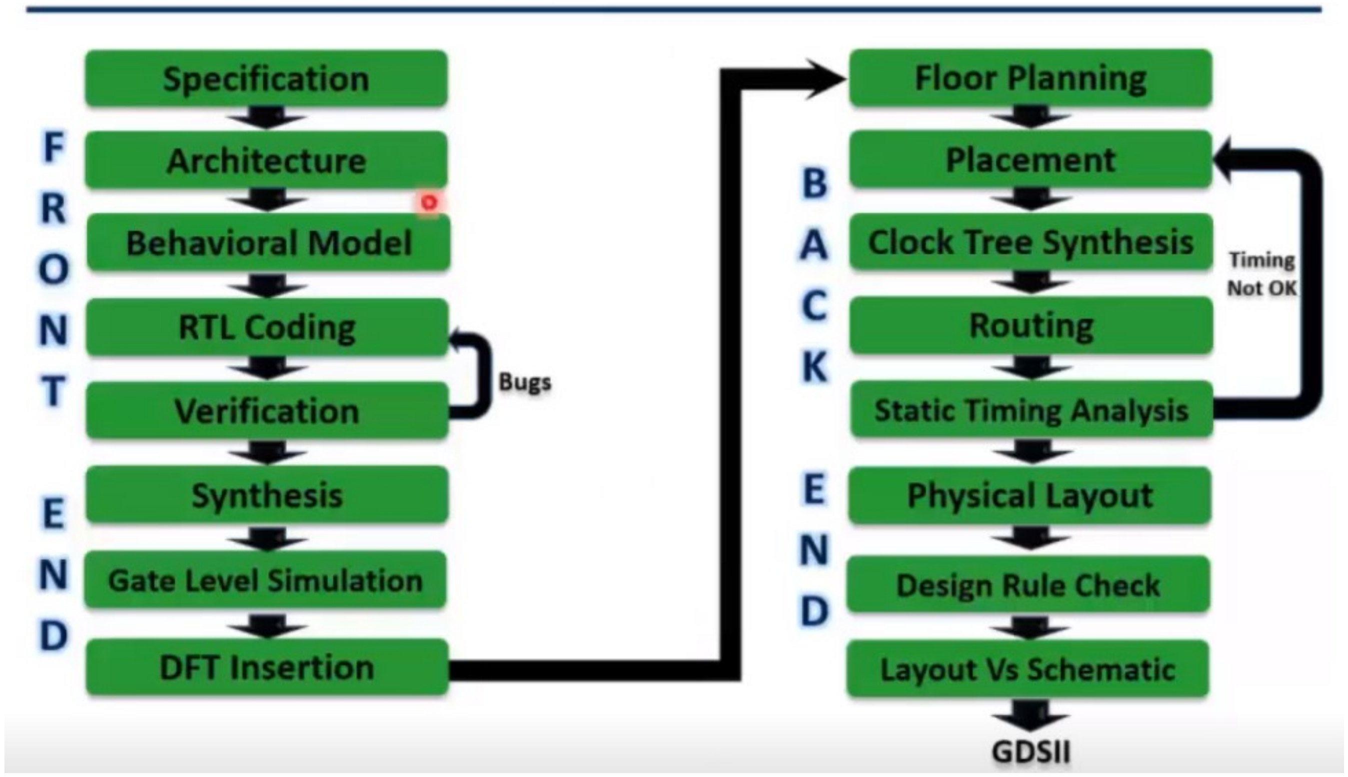


Fig 6.1 – VLSI Design Flow

Possible future scopes:

- Using System Verilog and methodologies like UVM the verification can be done.
- After which synthesis can be done once STA is done.
- On completion of the above steps, the design can be pushed on to the back end.
- On completion of the 8 stages of back end, the GDSII file is generated which can be sent to the foundry.

REFERENCES

- https://www.mathworks.com/matlabcentral/fileexchange
- https://www.mathworks.com/help/fixedpoint/ug/view-fixed-point-number-circles.html
- http://msdl.cs.mcgill.ca/people/mosterman/presentations/date07/tutorial.pdf
- https://www.microsemi.com/document-portal/doc_view/131619-modelsim-user
- https://www.mathworks.com/help/hdlcoder/ug/system-design-with-hdl-code-generation-from-matlab-and-simulink.html
- https://www.techsource-asia.com/generating-hdl-code-from-simulink
- https://www.matlabexpo.com/content/dam/mathworks/mathworks-dot-com/images/events/matlabexpo/in/2017/accelerating-fpga-asic-design-verification.pdf