FLT: Filter Block for SynTech

© 2019 Frans Fourie (fjf46@cornell.edu)
Haochuan Song (hs994@cornell.edu)

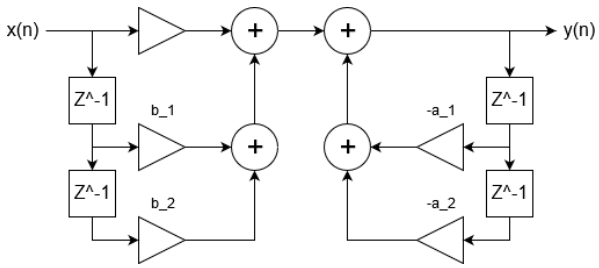


Figure 1: Low pass filter implemented by the block.

Abstract

The block designed in the following report receives the input signal from the oscillator, it also receives some input parameters that users can define such as the cut-off frequency and the quality factor. Using these inputs the block will implement a Low Pass Filter that removes unwanted harmonics from the signal received from the Oscillator block. The block will pass the filtered output on to the next block which is the Envelope block. A MATLAB Golden model was created in order to test the functionality of the block and used as the verification for the Verilog Code. After the Verilog code implementation was completed and perfectly matched the MATLAB model it was pushed by the Synopsis Design Compiler. After multiple optimizations where applied to the block acceptable results where obtained during synthesizing. The final block implementation now consists of a clock constraint of 120ns and a total cell area of 232402μm^2.

1 Introduction

In this report the design of the Syntech audio synthesizer filter module is documented. The filter module works as follow. First, this module will receive the sample signal from the oscillator. In this module, those signals are processed through the infinite-impulse-response (IIR) low-pass filter. The IIR filter aims to shape the harmonic structure of the oscillator. Thereby retaining the desired frequency components of the original signal and eliminating the unwanted harmonics. The new filtered signal will be passed on to the envelope module. The inputs are the sample signal $sta\_FLT\_in\_DI$ from the oscillator as well as the chip signals, such as the clock signal and reset signal. We also get values from memory which are the cut-off frequency, reciprocal of the quality factor, two divided by the sampling frequency and the scale down factor. These inputs are used to generate the weights $\{a_0, a_1, a_2, b_0, b_1, b_2\}$ (for a second-order filter) and output filtered signals $sta\_FLT\_Out\_DO$. The output is the processed signals from the IIR filter that will be passed to the envelope module.

2 MATLAB Golden Model

The MATLAB Golden model receives the input of the oscillator block and accurately simulates the flow and implementation of the Verilog Code. The MATLAB model is a fixed point model, to get the Golden model as close as possible to the Verilog implementation. Fig. 2 shows an example figure of how the MATLAB filter works to filter out the unwanted harmonic signals.

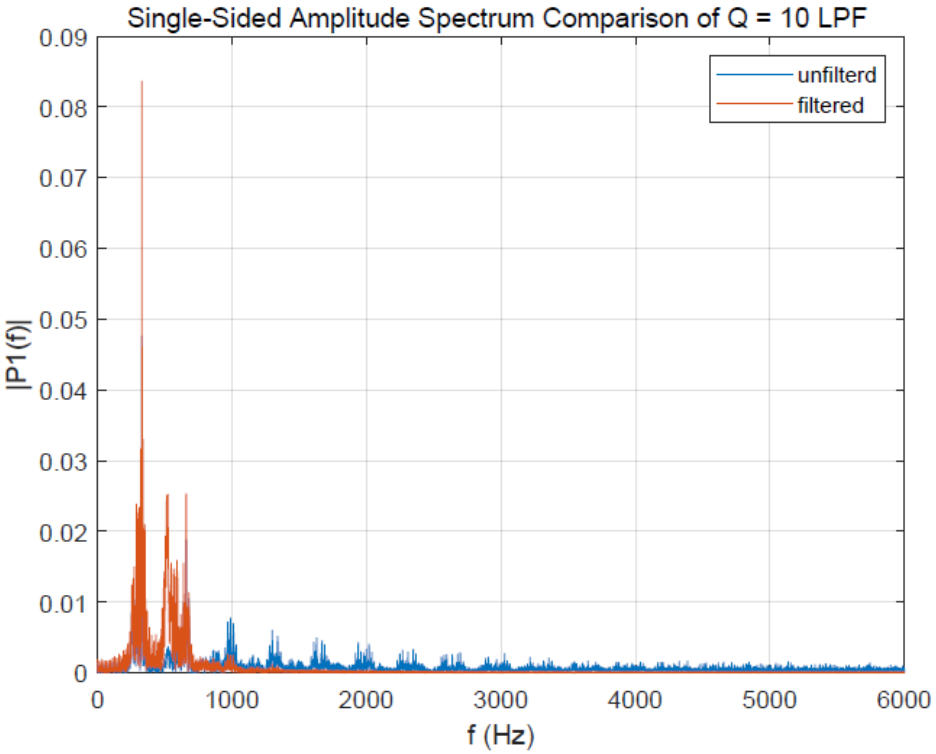


Figure 2: Functionality of FLT Block

2.1 Files

Listed below is the MATLAB files required for the Filter module with their functionality:

- `FLT_init.m` The filter initialization file gets all the required values from the memory as well as the user inputs and resets the input and output of the filter.
- `FLT.m` This is the main filter implementation. It starts off by declaring all the different quantization policies that will be used by the different wires and blocks in the Filter. The Filter then receives the input from the oscillator as well as receiving the initialization values from the *FLT_init.m* theses values are then all quantized. The program then proceeds to use the quantized values to calculate the filter parameters. The filter parameters are then fed to the main filter that calculates the filter output signal.

2.2 Parameters and States

In the fixed point MATLAB implementation, we have quantized the the IIR output with 24 bit precision and all the intermediate variables differently. In this block, the input data and output data are quantized with 24 bits and confined between $[-1, 1]$. The other parameters used within this block, including input

parameters and intermediate values, are quantized with 32 bits to guarantee the numerical accuracy. The MATLAB variable names, fixed-point-format, dynamic range and quantization policies are listed in Table 1.

Table 1: MATLAB input, internal, and output parameters including fixed-point format and unit.

MATLAB Code Name	**Fixed-point format**	**Dynamic Range**	**Unit**
alpha_D	{0,31,'s'}	$[-1,1)$	Ratio (Unitless)
Constant2_D	{3,28,'s'}	$[-8,8)$	Intermediate value (Unitless)
sin_out_D	{0,18,'s'}	$[-1,1)$	Intermediate value (Unitless)
cos_out_D	{0,18,'s'}	$[-1,1)$	Intermediate value (Unitless)
omega0_normalized_D	{0,15,'s'}	$[-1,1)$	Intermediate value (Unitless)
b0_D	{0,19,'s'}	$[-1,1)$	Parameter (Unitless)
b1_D	{1,18,'s'}	$[-2,2)$	Parameter (Unitless)
b2_D	{0,19,'s'}	$[-1,1)$	Parameter (Unitless)
a0_D	{1,18,'s'}	$[-2,2)$	Parameter (Unitless)
a0_inv_D	{0,19,'s'}	$[-1,1)$	Parameter (Unitless)
a1_minus_D	{2,29,'s'}	$[-4,4)$	Parameter (Unitless)
a2_minus_D	{0,31,'s'}	$[-1,1)$	Parameter (Unitless)
sta.FLT_OldSample0_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sta.FLT_OldSample1_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sta.FLT_OldSample2_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sta.FLT_OldIn1_D	{0,31,'s'}	$[-1,1]$	Intermediate value (Unitless)
sta.FLT_OldIn2_D	{0,31,'s'}	$[-1,1]$	Intermediate value (Unitless)
sta.FLT_OldIn3_D	{0,31,'s'}	$[-1,1]$	Intermediate value (Unitless)
par.FLT_RQ_D	{3,28,'s'}	$[-8,8)$	Ratio (Unitless)
par.FLT_f_0_D	{31,0,'s'}	$[-2147483648, 2147483648)$	Frequency (Hz)
par.FLT_RFS_norm_D	{0,31,'s'}	$[-1,1)$	Ratio (Unitless)
par.FLT_SD_D	{0,31,'s'}	$[-1,1)$	Ratio (Unitless)
sta.FLT_in_DI	{0,23,'s'}	$[-1,1)$	Sample value (Unitless)
sta.FLT_out_DO	{0,23,'s'}	$[-1,1)$	Sample value (Unitless)
xb0_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
xb1_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
xb2_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sumx2x1_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sumx0_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
sumy0_D	{1,30,'s'}	$[-2,2)$	Intermediate value (Unitless)
sumy2y1_D	{1,30,'s'}	$[-2,2)$	Intermediate value (Unitless)
sumy2y1_ls_D	{0,31,'s'}	$[-1,1)$	Intermediate value (Unitless)
ya1_D	{1,30,'s'}	$[-2,2)$	Intermediate value (Unitless)
ya2_D	{1,30,'s'}	$[-2,2)$	Intermediate value (Unitless)

The following is the list of the all the values stated above with each one's purpose and how it was calculated.

- `alpha_D` Calculated Ratio used in computing filter parameters.
  `(Constant2_D * sin_out_D)`

- `Constant2_D` Intermediate value used in the calculation of the filter parameters.
  `(par_FLT_RQ_D Right Shifted by 1 )`
- `sin_out_D` Intermediate value used in the calculation of the filter parameters.
  `(sin(omega0_normalized_D * π))`
- `cos_out_D` Intermediate value used in the calculation of the filter parameters.
  `(cos(omega0_normalized_D * π))`
- `omega0_normalized_D` Intermediate value used in the calculation of the filter parameters.
  `(par.FLT_RFS_norm_D * par.FLT_f0_D)`
- `a0_D` Intermediate value used in the calculation of the filter parameters.
  `(1*2^(alpha_frac_width) + alpha_D)`
- `b0_D` Filter parameter used in the main filter loop.
  `(b1_D with decimal shifted to the left 1 place)`
- `b1_D` Filter parameter used in the main filter loop.
  `(1*2^(cos_frac_width) - cos_out_D)`
- `b2_D` Filter parameter used in the main filter loop.
  `(b1_D with decimal shifted to the left 1 place)`
- `a0_inv_D` Filter parameter used in the main filter loop.
  `(1/a0_D)`
- `a1_minus_D` Filter parameter used in the main filter loop.
  `(cos_out_D with zero plugged after)`
- `a2_minus_D` Filter parameter used in the main filter loop.
  `(1*2^(alpha_frac_width) + alpha_D)`
- `xb0_D` Intermediate Filter value used in the calculation of the filter output.
  `b0_D * sta.FLT_OldIn0_D`
- `xb1_D` Intermediate Filter value used in the calculation of the filter output.
  `b1_D * sta.FLT_OldIn1_D`
- `xb2_D` Intermediate Filter value used in the calculation of the filter output.
  `b2_D * sta.FLT_OldIn2_D`
- `ya1_D` Intermediate Filter value used in the calculation of the filter output.
  `a1_minus_D * sta.FLT_OldSample1_D`
- `ya2_D` Intermediate Filter value used in the calculation of the filter output.
  `a2_minus_D * sta.FLT_OldSample2_D`
- `sumx2x1_D` Intermediate Filter value used in the calculation of the filter output.
  `xb1_D + xb2_D`
- `sumx0_D` Intermediate Filter value used in the calculation of the filter output.
  `xb0_D + sumx2x1_D`

- `sumy0_D` Intermediate Filter value used in the calculation of the filter output.
  `sumx0_D + sumy2y1_ls_D`
- `sumy2y1_D` Intermediate Filter value used in the calculation of the filter output.
  `ya1_D + ya2_D`
- `sumy2y1_ls_D` Intermediate Filter value used in the calculation of the filter output.
  `sumy2y1_D left shifted by 1`
- `par.FLT_RQ_D` This is the input gotten from the user. It is the inverse of the quality factor.
- `par.FLT_f_0_D` This is the input gotten from the user. It is the cut-off frequency of the filter.
- `par.FLT_RFS_norm_D` This is based on the SynTech module internal sampling frequency. It is the pseudo normalized sampling frequency.
- `par.FLT_SD_D` This is an internally calculated parameter. It is multiplied with the input of the FLT block, this is done to ensure the FLT block output is capped between (-1,1).
- `sta.FLT_OldSample(0-2)_D` This is an internal block state. The number after OldSample indicates how old the sample is. 0 indicates it is the current clock cycle calculated value, 1 means it is the previous clock cycles calculated value, 2 means it is 2 clock cycles ago calculated value. It is used in the calculation of filter output.
  `sta.FLT_In_DI * par.FLT_SD_D`
- `sta.FLT_OldIn(0-2)_D` This is an internal block state. The number after OldIn indicates how old the sample is. 0 indicates it is the current clock cycle calculated value, 1 means it is the previous clock cycles calculated value, 2 means it is 2 clock cycles ago calculated value. It is used in the calculation of filter output.
  `sumy0_D * a0_inv_D`
- `sta.FLT_In_DI` This is the input to the FLT block. It is received from the Oscillator block.
- `sta.FLT_out_D0` This is the output of the FLT block. It is sent to the Envelope block.

To scale the outputs of the LPF down to $[-1, 1]$, we utilize the conclusion from the conducted experiments. With a given Q, we can now predict the maximum output of the LPF I_inter_var. In hardware, we will use right shift to replace division. Thus, we will right shift the input by the calculated amount of bits that was calculated in Eq. 1:

$$n = \lceil \log_2(I\_inter\_var) \rceil. \tag{1}$$

To show the feasibility of this scheme, we set $f_0 = 500Hz$ and $Q = 10$. With this large Q, if we do not scale the results of the LPF, the outputs will be too large and can not be utilized to the next block. However, with the above method, it successfully scales the results to $[-1, 1]$ and we can see that the results in Fig. 2 are nearly equivalent to the floating-point results in Fig. 7.

In the table α is shown as being only between $[-1, 1)$. This is because in the program Q is currently fixed at 10 and thus $[-1, 1)$ is a sufficient range for α.

2.3 Detailed Functionality of MATLAB Code

We start our MATLAB implementation by declaring all our fixed point parameter variables and specifying the quantization policy that will be used for each fixed point variable. Our program receives the input signal from the oscillator block which is scaled using the scale down factor *par_FLT_SD_D* that is retrieved from memory. Both the input *sta_FLT_OldIn_D* and output *sta_FLT_OldSample_D* memory are also updated in this section. The filter parameters are then calculated along with the value for every path in the circuit between two blocks. This is then used to implement the filter and calculate the output signal that is sent to the next block. A detailed list of input and output parameters as well as all parameters gotten from memory can be found in the above Specification Tables section of the report. A detailed list of the quantization policies used can be found below in the quantization section of the report.

For a second-order IIR filter, we use the **FLT_init** function to initialize two 1×3 arrays in the state field including *sta_FLT_OldIn_D* and *sta_FLT_OldSample_D* that indicate 3 filter inputs and 3 outputs. We insert function **FLT** between **OSC** and **ENV** and change the inputs and outputs accordingly.

In function **FLT**, it first update state filed *sta.FLT* to simulate the data flow. It will then switch filter type at first where type 1 indicates the LPF. Default filter type is a wire which directly outputs the input from OSC. After that, it will calculate 6 weights for transfer function from Eq. (9)−(14) to perform Eq. (3). Finally, the function outputs *sta_FLT_OldSample_D* which is the latest output in Eq. (3).

2.3.1 Math to create a Low Pass Filter implemented in MATLAB

To create a filter, we start with the transfer function in the frequency domain:

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}. \tag{2}$$

The following is the direct form 1 implementation equation that is derived from the Eq. (2):

$$y[n] = (\frac{b_0}{a_0}) \cdot x[n] + (\frac{b_1}{a_0}) \cdot x[n-1] + (\frac{b_2}{a_0}) \cdot x[n-2] - (\frac{a_1}{a_0}) \cdot y[n-1] - (\frac{a_2}{a_0}) \cdot y[n-2]. \tag{3}$$

In a lot of applications, a_0 will be equal to 1. This would simplify the above equations to the following:

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}, \tag{4}$$

and

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2]. \tag{5}$$

For this simplified equation, it is easy to generate a Block Diagram. Fig. 1 depicts the Block diagram for the above equations.

Next we need to take the inputs the block receives to calculate the necessary parameters the block needs. First, we need to calculate ω_0. This will be done using the known sampling frequency F_S and the cut-off frequency f_0 and then using the following equation:

$$\omega_0 = 2 \cdot \pi \cdot \frac{f_0}{F_S}. \tag{6}$$

Following this we will need to calculate α. This will be done using the just computed ω_0 and the received quality factor Q and making use of Eq. (7):

$$\alpha = \frac{sin(\omega_0)}{2 \cdot Q}. \tag{7}$$

For the filter block, we are firstly going to focus on a Low Pass Filter (LPF), so we can now start looking at some low pass specific equations. The first equation is the normalized analog prototype equation for the LPF:

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1}. \tag{8}$$

The coefficients for the direct form 1 equation also needs to be calculated and they are calculated as follows using the parameters calculated above namely ω_0 and α.

$$b_0 = \frac{1 - cos(\omega_0)}{2}, \tag{9}$$

$$b_1 = 1 - cos(\omega_0), \tag{10}$$

$$b_2 = \frac{1 - cos(\omega_0)}{2}, \tag{11}$$

$$a_0 = 1 + \alpha, \tag{12}$$

$$a_1 = -2 \cdot cos(\omega_0), \tag{13}$$

$$a_2 = 1 - \alpha. \tag{14}$$

These calculated values can now be substituted into the direct form Eq. (3) and used to implement an LPF.

2.3.2 MATLAB Verification

In order to verify our MATLAB we simulated the block with different input parameters, comparing the floating point output to the fixed point output. Realistically the range in which the filter will operate is Q from 0.5 to 20 and f_0 from 500 to $50k$. A lot of combinations of these input values were tested and the MATLAB implementation held up for all of them. In this report only two cases, which show that the model holds up for the edge cases is shown. The graphs showing the comparison for the fixed point to floating point for $Q = 0.5; f_0 = 50k$ is shown in Fig. 3 and for $Q = 10; f_0 = 500$ is shown in Fig. 4.

2.3.3 Experiment 1

We experimented by changing the Q value of of our circuit and looking that our circuit functioned correctly for the different values of Q. The simulation was run and and the largest output signal for every Q between $[1, 100]$ in intervals of 1 was determined. We then did polynomial fitting to generate a curve to predict the largest outputs of the filter as Q changes. By then looking at the largest possible output for every value of Q, we could predict the dynamic range of the output and use this to determine our quantization policy. If Q is picked too large then the filter will incorrectly amplify the frequency components near the cut-off frequency. The prediction and curve fitting is shown below in Fig. 5. For Fig. 7 to 8, we show the single sided spectrum of our filters output for different values of Q. Making use of poly fit slowed down the circuit significantly and it would be hard to implement in hardware, therefore an alternative is necessary.

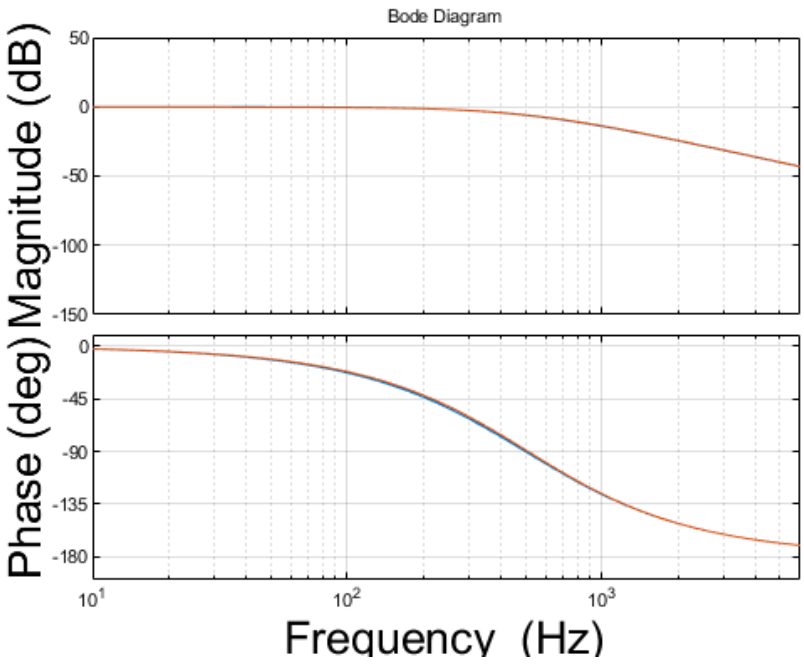


Figure 3: Float point to Fixed point comparison for $Q = 0.5$ and $f_0 = 50k$

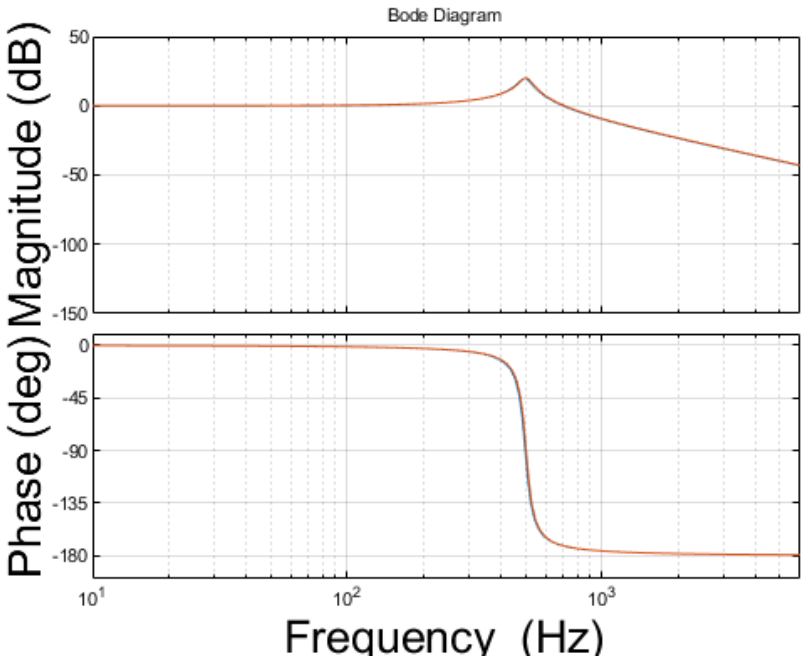


Figure 4: Float point to Fixed point comparison for $Q = 10$ and $f_0 = 500$

Original Shifting Strategy

By constructing and making use of a look up table, the need for poly fit can be eliminated. The look up table that will be used can be seen in Fig. 6. The lookup table was constructed by calculating log base 2 of the maximum output signal and then the answer was rounded up to get the amount of necessary bits. The

table showing what ranges of Q correspond to how many bits can be seen in Table 2. The only impact different inputs will have on this look up table is if the new inputs have a different amplitude range than $[0, 1]$. If the largest amplitude exceeds 1 then it will need to be specified in advance.

Scaling Down Strategy

In this deliverable, we choose $Q = 10$ and made use of an exact scaling down factor *par_FLT_SD_D* from Fig. 5. Then we could divide the input signal that is a 8 bit signal from OSC by *par_FLT_SD_D* to get a 32 bit value for computations inside our filter. Note that the main goal of this strategy is to fully utilize the dynamic range from $[-1, 1)$.

Table 2: Look up table for Q values

par.FLT.Q	$[0, 2)$	$[2, 6]$	$(6, 20]$	$(20, 66]$	$(66, 100]$
Bits to Right Shift by	0	1	2	3	4

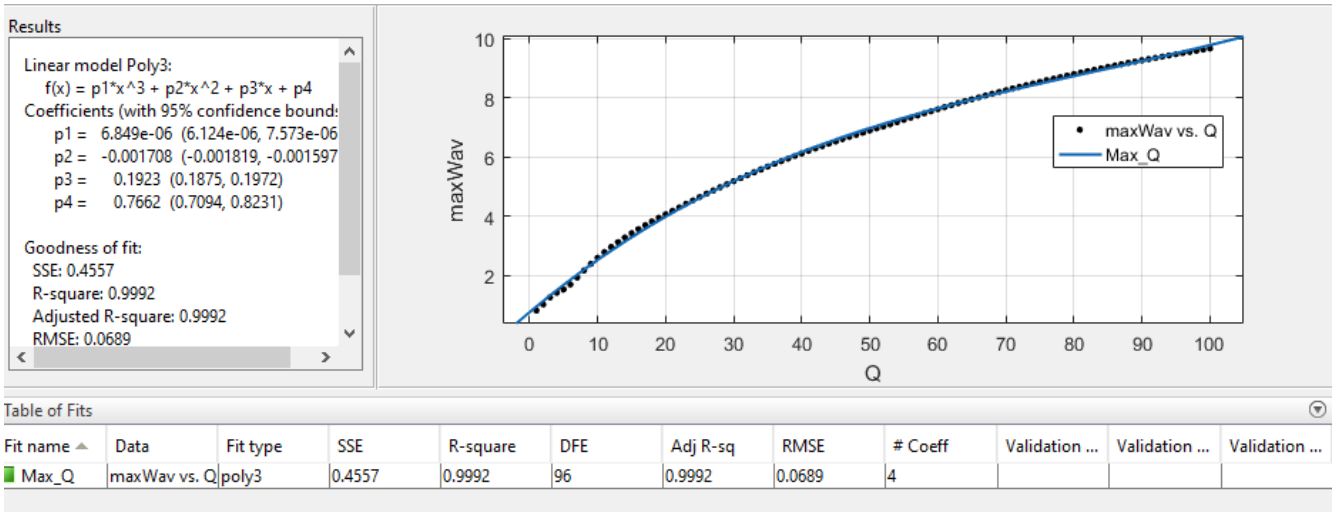


Figure 5: Predicted max output curve for different values of Q

2.3.4 Experiment 2

We experimented by changing the Q value of of our circuit and recalculating all the circuit parameters then creating a transfer function that was used to plot the bode plot of the filter response of the low pass filter. The below bode plot Fig. 9 shows the bode plots for different values of Q ranging from 0.5 to 30.5 in 1.5 increments. The bode plot shows the Magnitude (dB) versus Frequency (Hz) in the top graph and Phase (deg) versus Frequency (Hz) in the bottom graph.

2.3.5 Experiment 3

To optimise the block the least amount of bits needs to be used for the fixed point parameters while not letting the results deviate to much from the floating point results. In order to obtain the optimal quantization policies we tested different quantization policies comparing the floating point answer to the fixed point results. It was important to get the quantization of a_0 to a minimum as it is the input the divider which takes a lot of time, having the input as small as possible will significantly reduce this time. For a_0 we settled on a quantization policy of $\{1, 18, ' s'\}$ because when we changed the decimal bits to 17 or

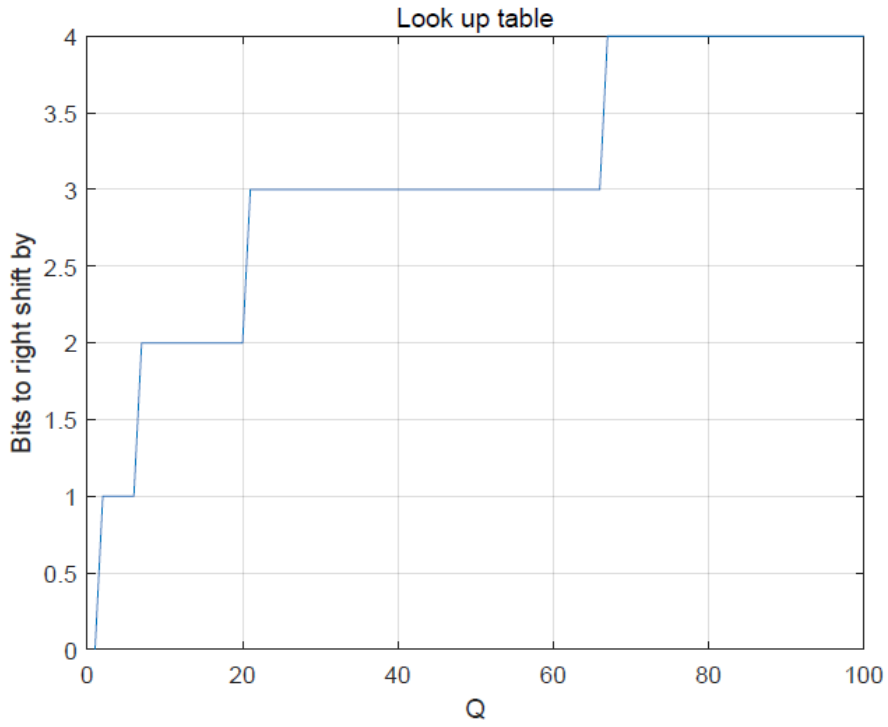


Figure 6: Predicted max output curve for different values of Q

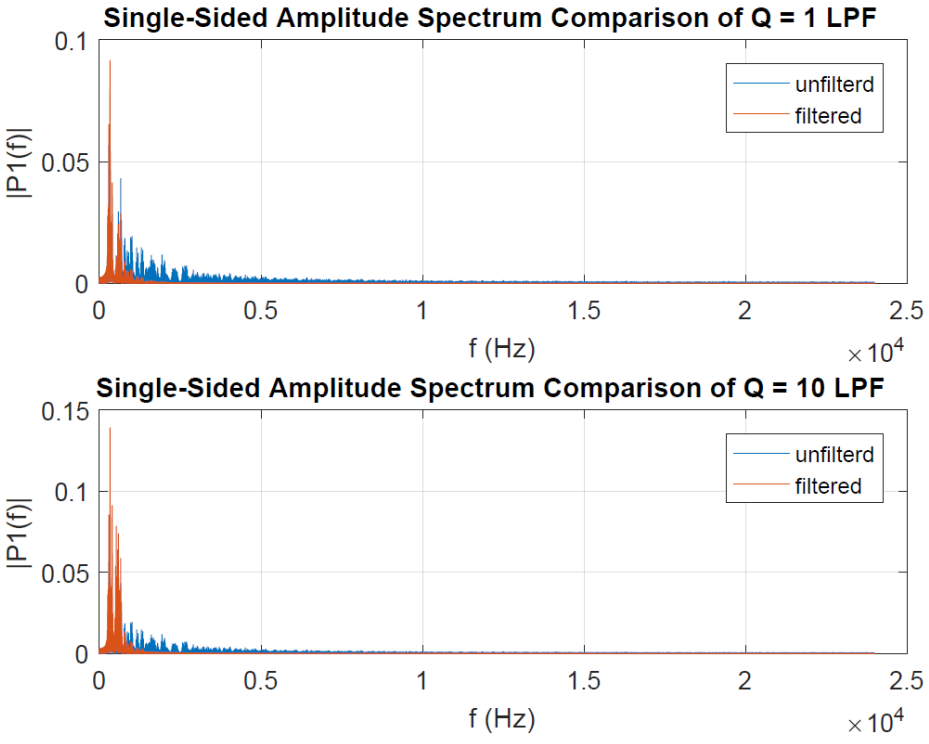


Figure 7: Output for $Q = 1$ and $Q = 10$

less we got significant differences between our fixed point and floating point results, this can be seen in

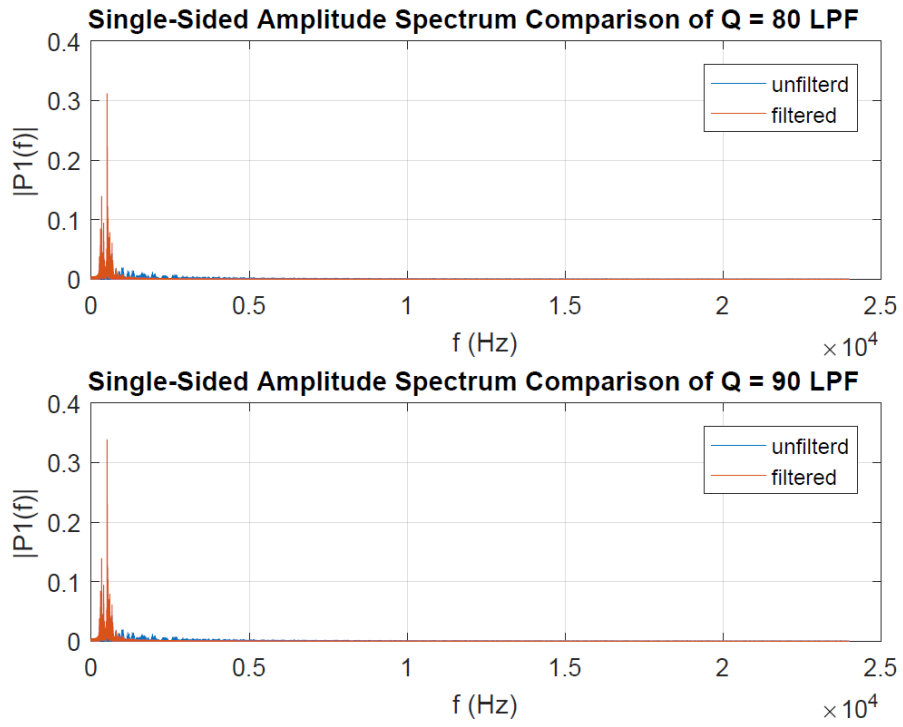


Figure 8: Output for $Q = 80$ and $Q = 90$

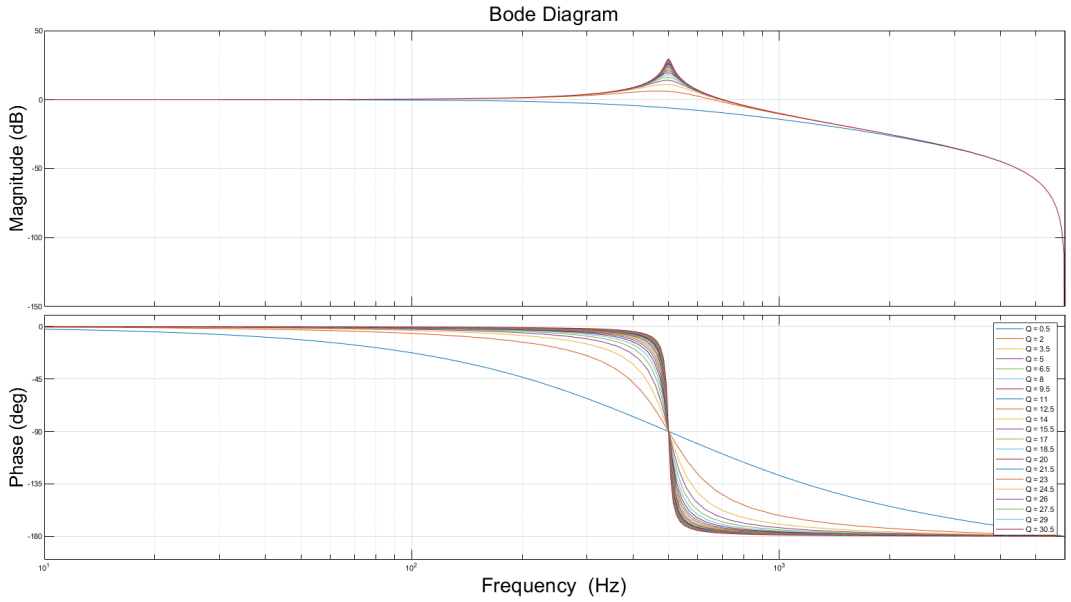


Figure 9: Bode plots for different Q values

Fig. 10 and Fig. 11. It was also important to reduce the size of the inputs to our cosine IP cores as they take a long time to compute. It was decided to use a quantization policy of $\{0, 15, ' s'\}$ for $omega0\_normalized\_D$ which is the input to the cosine IP cores. We also decided to use a quantization policy of $\{0, 18, ' s'\}$ for the outputs of the cosine IP cores since if we reduce the decimal bits we started losing a lot of precision as can

be seen in Fig. 12 and Fig. 13

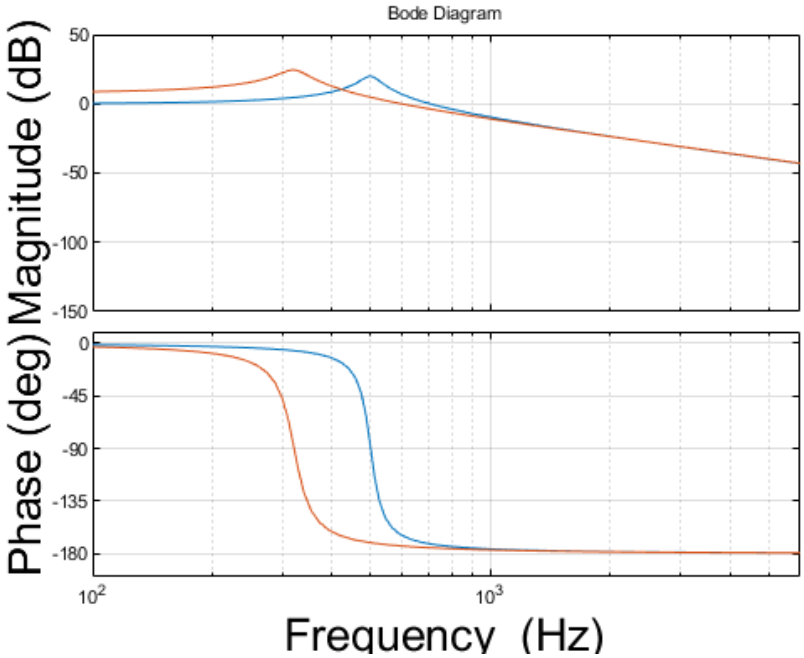


Figure 10: Bode plots comparing floating point to fixed point where a_o has only 16 decimal bits

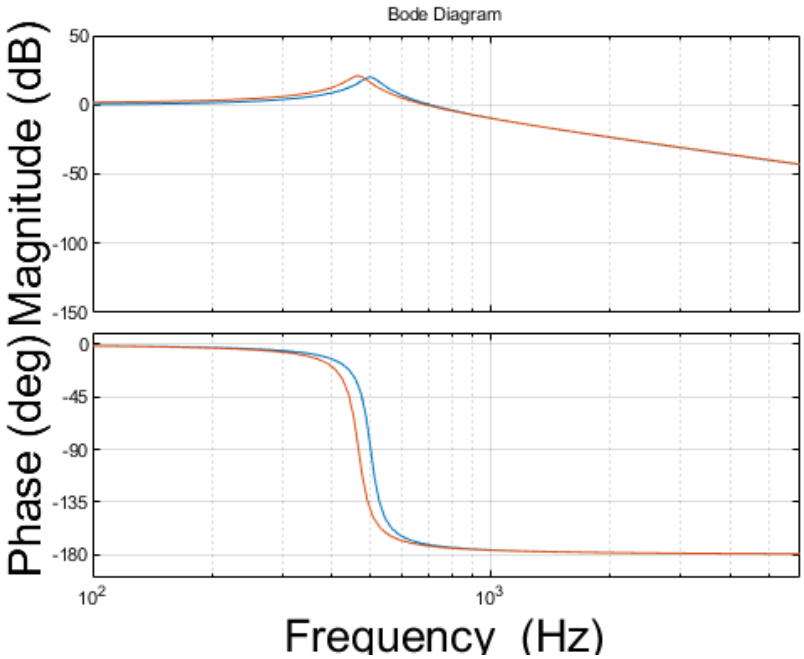


Figure 11: Bode plots comparing floating point to fixed point where a_o has only 17 decimal bits

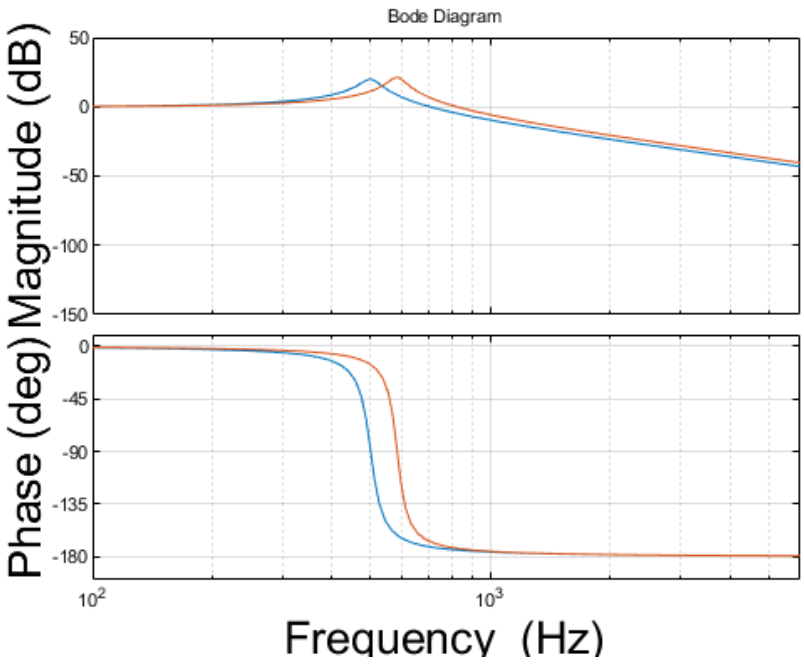


Figure 12: Bode plots comparing floating point to fixed point where cosine outputs have only 16 decimal bits

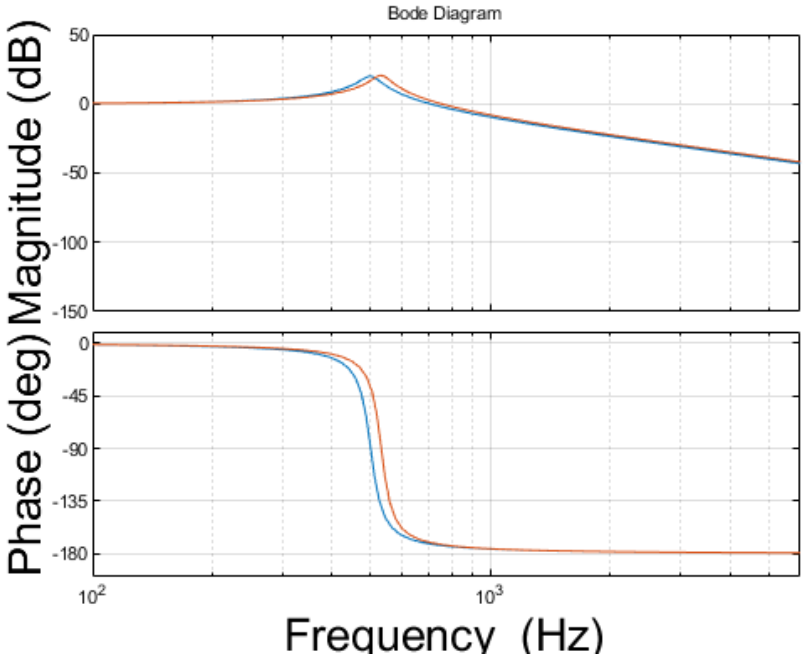


Figure 13: Bode plots comparing floating point to fixed point where cosine outputs have only 17 decimal bits

2.4 State of MATLAB Code

The MATLAB Golden model is completely working and matches the Verilog results. We verified the Matlab model by comparing the fixed point results to that of floating point results across the range of all the possible values and states the model could have.

3 Verilog Code

The verilog code is the Verilog implementation of the MATLAB golden model of the Filter block. The Verilog Code has the same functionality as the MATLAB golden model but in order to get the block to be of an acceptable size and acceptable speed some operations differ from those in the MATLAB model due to the difficulty of modeling these methods of operation in MATLAB. Some hardware specific operations that differ from the MATLAB model are the divider and cosine blocks implemented.

3.1 Files

The following is the list of all the Verilog code files created to create the FLT block:

- `FLT.v` This is the main implementation of the FLT block in Verilog. This file contains both the filter coefficient calculations as well as the main filter implementation.
- `FF.v` This is the Flip Flop Verilog file that is implemented in the main Verilog file.
- `DW_div_seq.v` This is the sequential divider Verilog file that is implemented in the main Verilog file.

3.2 Inputs and Outputs

For the Verilog implementation mostly the same input, internal and output parameters are used with the only differences being some slight naming changes to accommodate Verilog convention and coding rules as well as the few Verilog specific inputs such as the clock signal. All signals used in the Verilog implementation can be seen in Table 3 .

The following are the list of input and output signals used by the Verilog implementation:

- `CLK_CI` is the master clock signal.
- `RST_RBI` is the asynchronous reset signal; the module is reset if `RST_RBI=0`.
- `par\_In\_DI` the signal used to write parameters into the memory.
- `Addr\_DI` this is the memory address select signal.
- `WrEn\_SI` the memory write enable signal.
- `par.FLT_RQ_D` This is the input gotten from the user. It is the inverse of the quality factor.
- `par.FLT_f_0_D` This is the input gotten from the user. It is the cut-off frequency of the filter.
- `par.FLT_RFS_norm_D` This is based on the SynTech module internal sampling frequency. It is the pseudo normalized sampling frequency.
- `par.FLT_SD_D` This is an internally calculated parameter. It is multiplied with the input of the FLT block, this is done to ensure the FLT block output is capped between (-1,1).
- `sta.FLT_OldSample(0-2)_D` This is an internal block state. The number after OldSample indicates how old the sample is. 0 indicates it is the current clock cycle calculated value, 1 means it is the previous clock cycles calculated value, 2 means it is 2 clock cycles ago calculated value. It is used in the calculation of filter output.
  `sta.FLT_In_DI * par.FLT_SD_D`

Table 3: Verilog code inputs, outputs, and parameters including fixed-point format and unit.

MATLAB Code Name	**Fixed-point format**	**Dynamic Range**	**Unit**
$alpha\_D$	{0,31,'s'}	$[-1, 1)$	Ratio (Unitless)
$Constant2\_D$	{3,28,'s'}	$[-8, 8)$	Intermediate value (Unitless)
sin_out_D	{0,18,'s'}	$[-1, 1)$	Intermediate value (Unitless)
cos_out_D	{0,18,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$omega0\_normalized\_D$	{0,15,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$b0\_D$	{0,19,'s'}	$[-1, 1)$	Parameter (Unitless)
$b1\_D$	{1,18,'s'}	$[-2, 2)$	Parameter (Unitless)
$b2\_D$	{0,19,'s'}	$[-1, 1)$	Parameter (Unitless)
$a0\_D$	{1,18,'s'}	$[-2, 2)$	Parameter (Unitless)
$a0\_inv\_D$	{0,19,'s'}	$[-1, 1)$	Parameter (Unitless)
$a1\_minus\_D$	{2,29,'s'}	$[-4, 4)$	Parameter (Unitless)
$a2\_minus\_D$	{0,31,'s'}	$[-1, 1)$	Parameter (Unitless)
$sta_FLT\_OldSample0\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sta_FLT\_OldSample1\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sta_FLT\_OldSample2\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sta_FLT\_OldIn1\_D$	{0,31,'s'}	$[-1, 1]$	Intermediate value (Unitless)
$sta_FLT\_OldIn2\_D$	{0,31,'s'}	$[-1, 1]$	Intermediate value (Unitless)
$sta_FLT\_OldIn3\_D$	{0,31,'s'}	$[-1, 1]$	Intermediate value (Unitless)
$par_FLT\_RQ\_D$	{3,28,'s'}	$[-8, 8)$	Ratio (Unitless)
$par_FLT\_f_0\_D$	{31,0,'s'}	$[-2147483648, 2147483648)$	Frequency (Hz)
$par_FLT\_RFS\_norm\_D$	{0,31,'s'}	$[-1, 1)$	Ratio (Unitless)
$par_FLT\_SD\_D$	{0,31,'s'}	$[-1, 1)$	Ratio (Unitless)
$sta_FLT\_in\_DI$	{0,23,'s'}	$[-1, 1)$	Sample value (Unitless)
$sta_FLT\_out\_DO$	{0,23,'s'}	$[-1, 1)$	Sample value (Unitless)
$xb0\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$xb1\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$xb2\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sumx2x1\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sumx0\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$sumy0\_D$	{1,30,'s'}	$[-2, 2)$	Intermediate value (Unitless)
$sumy2y1\_D$	{1,30,'s'}	$[-2, 2)$	Intermediate value (Unitless)
$sumy2y1\_ls\_D$	{0,31,'s'}	$[-1, 1)$	Intermediate value (Unitless)
$ya1\_D$	{1,30,'s'}	$[-2, 2)$	Intermediate value (Unitless)
$ya2\_D$	{1,30,'s'}	$[-2, 2)$	Intermediate value (Unitless)
$WrEn\_SI$	{1,0,'u'}	$[0, 1]$	Memory write enable signal (Unitless)
$Addr\_DI$	{5,0,'u'}	$[0, 32]$	Memory select signal
par_In_DI	{32,0,'u'}	$[0, 4294967296)$	Parameter Memory input
CLK_CI	{1,0,'u'}	$[0, 1]$	Clock input signal
RST_RBI	{1,0,'u'}	$[0, 1]$	Reset input signal

- `sta.FLT_OldIn(0-2)_D` This is an internal block state. The number after OldIn indicates how old the sample is. 0 indicates it is the current clock cycle calculated value, 1 means it is the previous clock

cycles calculated value, 2 means it is 2 clock cycles ago calculated value. It is used in the calculation of filter output.
`sumy0_D * a0_inv_D`

- `sta.FLT_In_DI` This is the input to the FLT block. It is received from the Oscillator block.
- `sta.FLT_out_DO` This is the output of the FLT block. It is sent to the Envelope block.

For the variable naming issues, the only differences between those two platforms are the name of input/output flip flops. In Verilog codes, they are named with *sta_FLT_OldIn0-2_D* and *sta_FLT_OldOut0-2_D* where 0 is the newest value while in MATLAB codes, they are two 1×3 vectors namely *sta.FLT.OldIn_D* and *sta.FLT.OldSample_D* where the third element is the newest value. Also note that we have to create some intermediate wires for truncation after multiplication and division. Those variables do not exist in MATLAB codes since we directly use `RealMULT` and `RealDIV` functions. We have put the `~/verilog_source/src/FLT_wave.do` file in the zip file along with the testbench and I/O texts.

The input delay is 0.2ns to make sure the input still holds with input delay we set in the synthesis step. The output delay is 1.2ns in this step because the clock cycle is assumed to be 1ns here. In post-layout simulation, the output delay will be 150ns which contains 1 clock cycle (120ns) and 30ns extra propagation delay. To guarantee there will be enough time for the divider to produce a valid result, we insert 20 idle clock cycles in both the input and output files (just zeros in that period of time). Details can be found in Fig. 14 and Fig. 15. We highly recommend you to check the timing diagrams with our block diagram.

In the coefficients calculation circuit, we divide it into two separate parts. In general, all these 6 coefficients are synchronized by 4 flip flops due to the delay of the sequential divider. After the resetting the circuit, there are default parameters in the parameter memory so there will be several instantaneous results of $a0 \sim b2$ but they are not what we want. After the WrEN_SI goes low, we should have all these coefficients calculated right here if there is no flip flop inserted in these sequential logic, but for the sake of critical path, we do add some pipeline tricks here so at this specific time, there will only be some intermediate results like *alpha_D* and *cos_out_REG_D*. Note that these coefficients will not be valid until *a0_inv_D* is calculated.

Let's start from the calculation of $b0 \sim b2$. Due to the math relationship shown in Eq.9, these three variables (wires) in real circuits are connected to the same place and we just *remember* their different quantization strategies manually in our following design. They are synchronized by flip flop `FF_cos_out` which is enabled by the valid signal of the sequential divider. This flip flop was originally used for pipeline but since the the logic after this is pretty simple, we decide not to put an extra flip flop in our design.

Coefficient a_1 is also related to the circuits that calculates $b0 \sim b2$ so it's also synchronized by flip flop `FF_cos_out`.

To calculate $a0$ and $a2$, we need some tricks to eliminate the extremely long propagation delay. In the timing diagram, you can see that we extract *alpha_cal* out as a group because this intermediate variable is crucial to $a0$ and $a2$ (see block diagram and Eq. (14)). *a2_minus_REG_D* is calculated right after *alpha_D* but *a2_minus_D* has to wait for the synchronized signal. The divider used to calculate $a0$ is the biggest trouble in our design. We choose a sequential divider here considering the fact that these coefficients will not change as frequently as the data inputs. Hence, we decide to sacrifice the calculation delay to earn lower area and a shorter critical path. There are several things worth noting in the divider. We set the number of clock cycles to produce a valid result to 15 due to our test result. With all other default sets, it takes 15 clock cycles to produce a result. The enable signal here is stored in an asynchronized register and controlled by the *WrEn_SI* signal. Specifically, only when *WrEn_SI* is high, *a0_enable_S* is high. The divider is enabled in the next clock cycle after *a0_enable_S* turns high. To this end, at 5ns in Fig. 14,the

output of divider turns to *X* after being enabled, but it will only takes the last input during this enabled time window into calculation. Thus, the last input comes in at 8ns and the divider starts calculation at 9ns which produce the valid result at 24ns. Also at this time, *a0_enable_S* goes high so all these six coefficients are synchronized at 25ns.

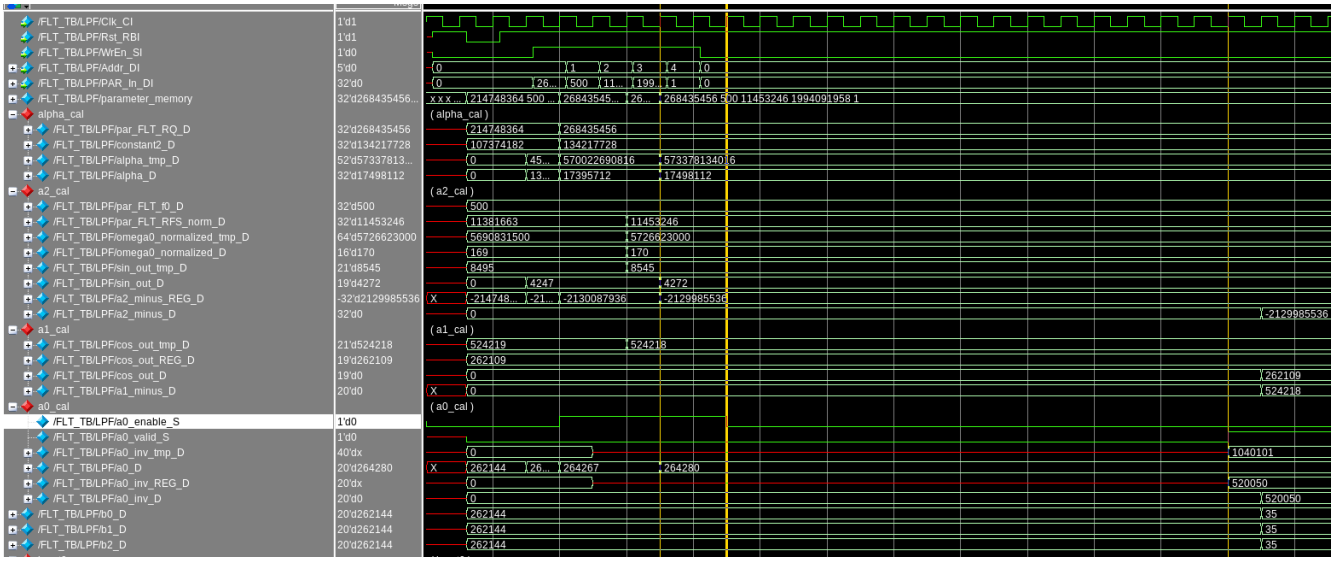


Figure 14: Timing diagram for coefficients calculation.

The calculation in the feed-back loop is very straightforward. To eliminate the input delay and synchronize the whole feed-back circuit, we put a flip flop after the input. To check whether the results match the MATLAB golden model, just simply check the output group in the Fig. 15 with *sta.FLT.OldSample_D*. The only things that may be confusing here are the sign extension after *sum_x0_D* and the left shift after *sum_y2y1_D*. Though they are both 32-bit, they have different number of fraction bits. We have to extend them to 33-bit and make two new wires that both have 31 fraction bits.

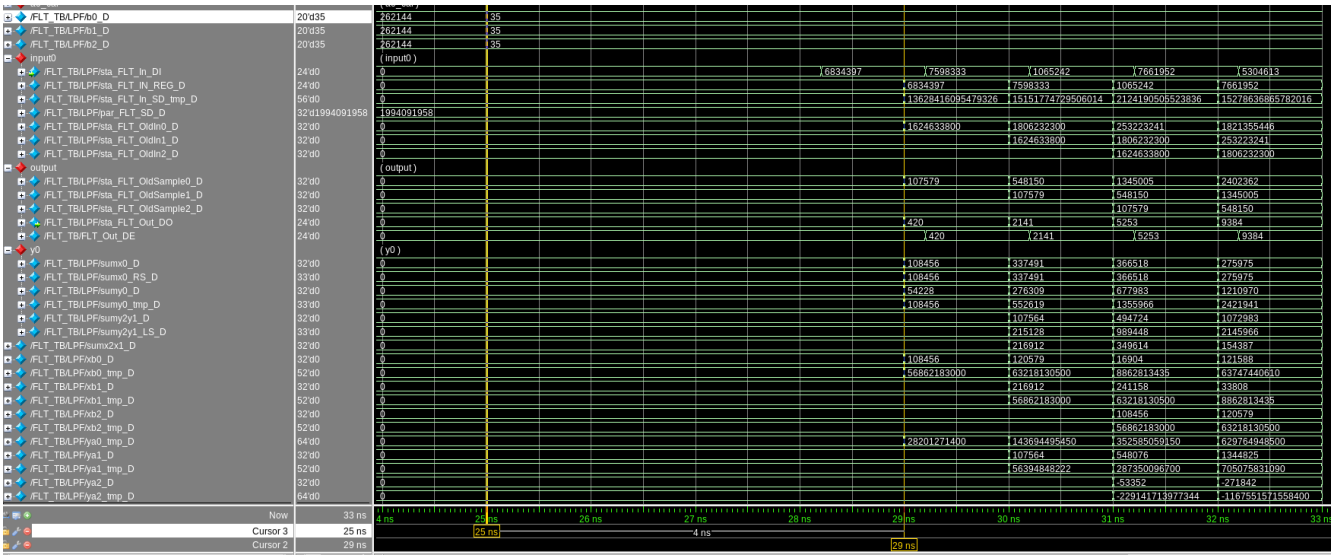


Figure 15: Timing diagram for feed-back loop calculation.

3.3 Block Diagrams and Functional Description

The following Fig 16 and Fig 17 is our hardware block diagram. Inputs are clearly shown in red circles and outputs are shown in green circles.

There are also a lot of constants used in the calculations. Outputs and inputs that have the same name/label are connected. These connections just weren't indicated with wire connections since, that would clutter the diagram and make it hard to read and understand. We make use of flip flops, adders, multipliers with and without constants, nots and memory. We also make use of IP cores in the form of a block that has cos and sin functionality which is used to compute the cosine and sine of ω_0. We have π in the block diagram but for the implementation we will approximate π and use the approximated constant instead of π.

In the Verilog Code the default values of the initialization parameters where also set in the Parameter Memory. The initialization parameters and there default values can be seen in the below Table 4.

Table 4: Table of default parameters

Parameter Memory Location	Parameter Name	Default Parameter Value
parameter_memory[0]	par_FLT_RQ_D	0.1
parameter_memory[1]	par_FLT_f0_D	500
parameter_memory[2]	par_FLT_RFS_norm_D	0.0053
parameter_memory[3]	par_FLT_SD_D	2
parameter_memory[4]	par_FLT_type_S	1

3.3.1 Coefficient calculations

In Fig. 17 we show how the filter parameters are calculated and what values are gotten from memory. There is also an implemented sequential divider IP core. From memory we get $par\_FLT\_RQ\_D$ (Reciprocal of Quality factor), $par\_FLT\_RFS\_norm\_D$ (2 divided by the Sampling frequency), $par\_FLT\_f0\_D$ (cut off frequency) and $par\_FLT\_SD\_D$ (scaling down factor). The diagram shows that we get CLK_CI, Rst_RBI, $WrEn\_SI$, $Addr\_DI$ and PAR_In_DI as inputs. This part of the block diagram has no outputs to external blocks only the parameter output to the main filter. All the values obtained from memory is shown the memory block.

For the Verilog the default initial parameters where set as $Q = 10$, $f0 = 500$, $fs = 12000$, and the following parameters $\frac{1}{Q}$, $f0$, $\frac{2}{fs} << 10$, and $ScalingDownFactor$ were stored in the memory. Note that the left shift to $par\_FLT\_RFS\_norm\_D$ and right shift for $\omega_0 = par\_FLT\_RFS\_norm\_D \times par\_FLT\_f0\_D$ compensate each other and will not cause extra calculation efforts. $par\_FLT\_RFS\_norm\_D$ is precalculated and stored in the memory. The parameters don't have actual decimal points in hardware. Thus the only thing left here is to manually remember the position of the decimal point for the parameters. The aim here is to fully utilize the dynamic range.

In the Verilog code the Block Diagram was recreated identically. For the $sin(\pi x)$ and $cos(\pi x)$ blocks we implemented the DW_sincos IP core. By changing the SIN_COS bit depending on the functionality required using 0 for sin and 1 for cos we can use this IP core for both sin and cos functionality. The DW_div_seq IP core was used in order to implement a sequential divider. The divider was set up to require 15 cycles to finish.

3.3.2 Main Filter Loop

In Fig. 16 we show the block diagram of the actual filter with its parameters in blue circles. The diagram shows that we get $sta\_FLT\_in\_DI$ as input. Lastly our only output is $sta\_FLT\_out\_DO$ that is outputted to

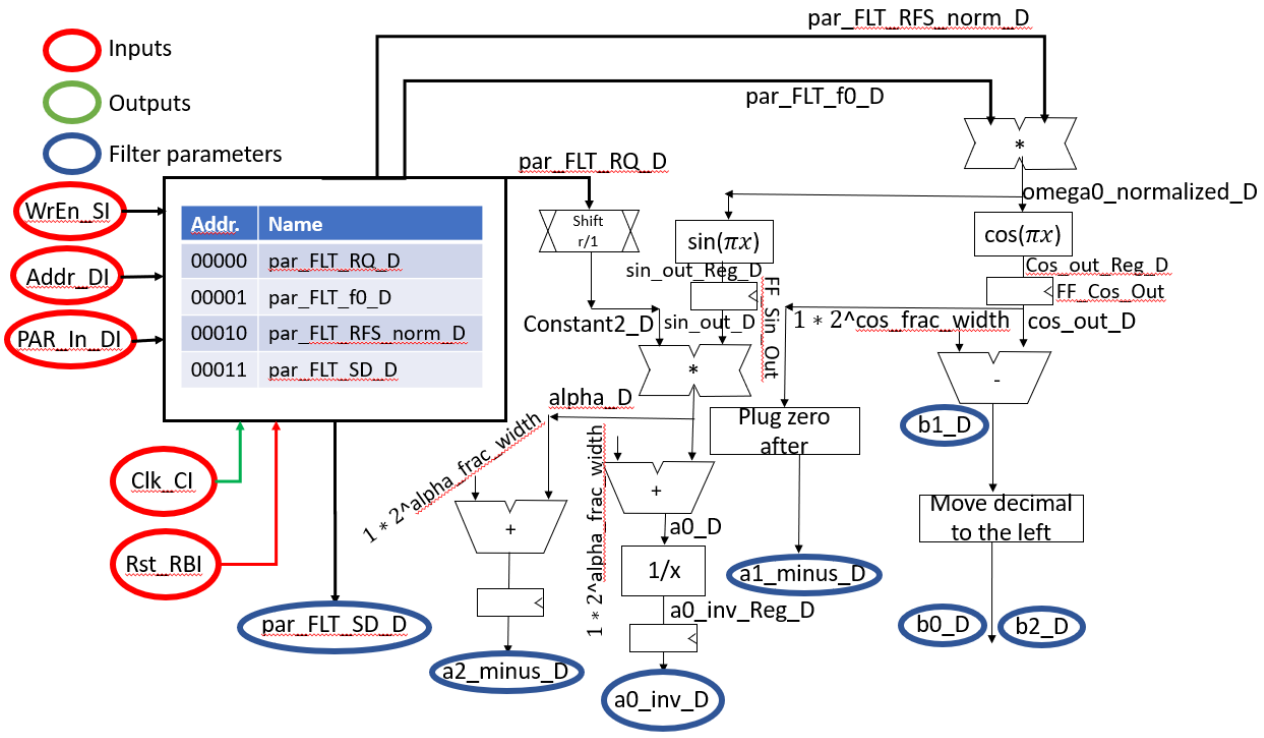


Figure 16: Detailed Block Diagram of memory and parameter calculation

the Envelope block.

par_FLT_f0_D is the 32-bit input and does not have an actual decimal point in hardware. Thus the only thing left here is to manually remember the position of the decimal point for the parameters. The aim here is to fully utilize the dynamic range.

3.4 Verification

A testbench was created using MATLAB containing a 1000 stimuli that was used to validate that the Verilog code matches the golden MATLAB model. Fig. 18 and Fig. 19 shows that the output of the Verilog code matches that of the Golden model and that all the values of the testbench matched that generated by the Verilog. Every parameter and every intermediate result has been verified against the Matlab Golden Model.

Four design tests have been done and passed. The Verilog implementation have been tested using the input from the oscillator and Random inputs. We tested 10 different Random input vectors and 5 different oscillator generated inputs. The Verilog implementation have also been tested using different values of Q ranging from $0.5 - 20$ and cut-off frequency values f_0 ranging from $500Hz - 50kHz$ as inputs. All the testes performed where passed and the implementation held up.

- *Oscillator input* tested using input from the oscillator block. Used the different oscillator tunes provided by the different teams for testing.
- *Random input* tested using random inputs uniformly distributed from $0 - 1$. These randomly generated inputs are to test for edge cases and abnormal behaviour.
- *Different Q values* tested using different quality factor values. See if quantization policies and operations work for Q values across the range of acceptable Q values.

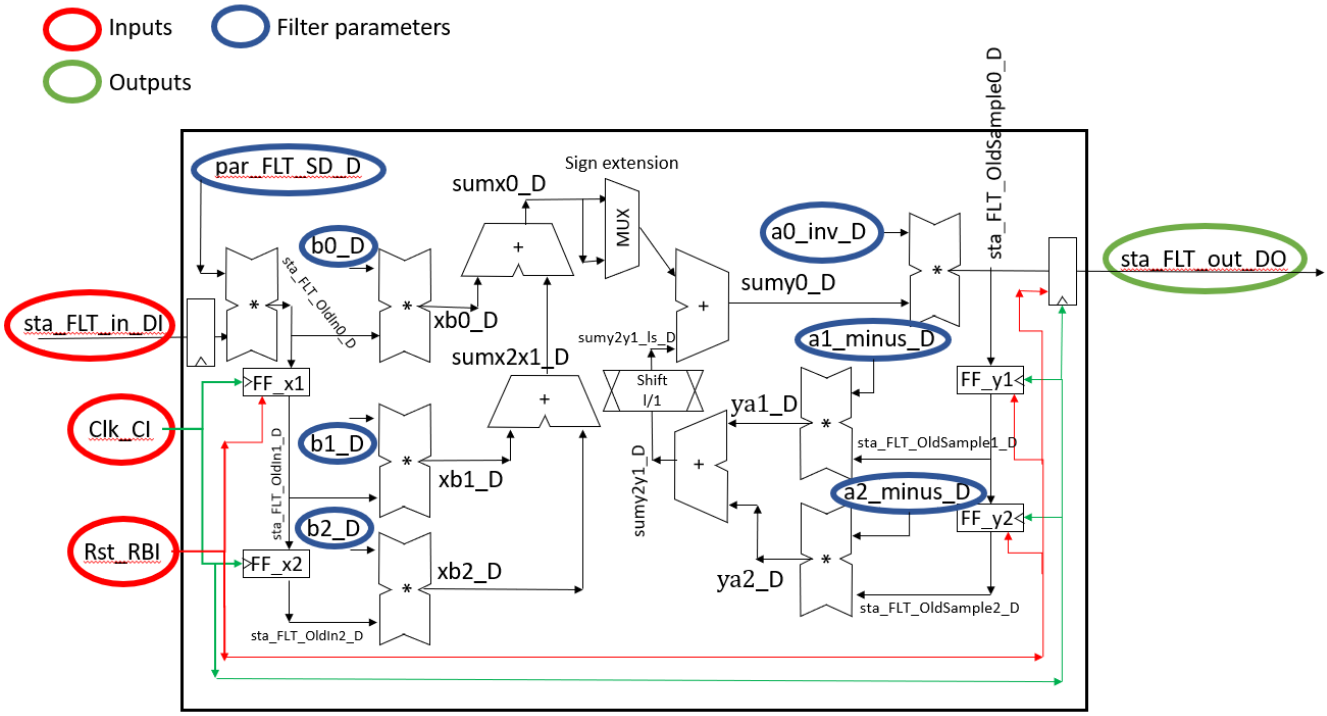


Figure 17: Detailed Block Diagram of main loop

```
# <<< :D All outputs match the expected results! :D >>>
# ** Note: $finish    : ../tb/FLT_TB.v(133)
#    Time: 1029200 ps  Iteration: 0  Instance: /FLT_TB
```

Figure 18: Output of Modelsim showing that the Verilog results matches that of the testbench

Figure 19: Showing that the output signal of the testbench and that of the Verilog matches up

- *Different f_0 values* tested using different cut-off frequencies. See if quantization policies and operations work for cut-off frequency values across the range of acceptable cut-off frequency values.
- *Different Sampling frequency values* tested using different module Sampling frequencies. See if quantization policies and operations work for different sampling frequency values across a range of expected values.

3.5 Synthesis Commands

The synthesis commands given in class where followed. The following is the synthesis script used.

```
1 ##########################################################################
2 set DESIGN_NAME "FLT"
3 #Set up number of cores
4 set_host_option -max_core $NUM_CORES
5
```

Table 5: Summary of Verification Tests.

Test	Brief description	# vectors
Oscillator input	Test using input from oscillator block	5
Random input	Test using randomly generated inputs	10
Different Q values	Test using different values of Q spanning functional range	21
Different f_0 values	Test using different values of f_0 spanning functional range	20
Different Sampling frequency values	Test using different realistic values	10

```
#Clean previous designs
remove_design -designs
sh rm -rf work/*


##########################################################################################

#Analyze design
#Verilog files
analyze -library work -format verilog { \
        ../src/FF.v \
        ../src/DW_div_seq.v\
        /opt/synopsys/M-2016.12/synopsys-dc-M-2016.12/dw/dw02/src_ver/DW_sincos.v\
        ../src/FLT.v \
}


#Elaborate design (with default parameters)
elaborate ${DESIGN_NAME} -library work

current_design ${DESIGN_NAME}

##########################################################################################

#Define constraints

set CLK_PERIOD  120.0

create_clock Clk_CI -period $CLK_PERIOD;
set_clock_transition 0.2 [get_clocks Clk_CI];
set_input_delay 0.2 -clock Clk_CI [remove_from_collection [all_inputs]

set_driving_cell -library saed90nm_typ -lib_cell INVX4 -pin ZN [all_inputs];
set_load 0.01 [all_outputs];


```

```
42 ################################################################################
43
44 #Compile design
45 compile_ultra -no_autoungroup
46
47
48 check_design
49
50 ################################################################################
51
52 #Reports
53 report_timing -max_paths 5 > ./reports/${DESIGN_NAME}_${CLK_PERIOD}ns_timing_max.rpt
54 report_timing -delay min -max_paths 5 >
55                 ./reports/${DESIGN_NAME}_${CLK_PERIOD}ns_timing_min.rpt
56 report_area -physical -hierarchy >
57                 ./reports/${DESIGN_NAME}_${CLK_PERIOD}ns_area.rpt
58 #report_power -net -cell -hierarchy >
59                 ./reports/${DESIGN_NAME}_${CLK_PERIOD}ns_power.rpt
60
61 ################################################################################
62
63 #Add dummy nets for unconnected pins /Not used in quentin nor mr wolf
64 define_name_rules verilog -add_dummy_nets
65 define_name_rules verilog -replacement_char x
66 define_name_rules verilog -case_insensitive
67
68 #Save the design database
69 write -format ddc -hierarchy -output
70                 ./outputs/${DESIGN_NAME}_${CLK_PERIOD}ns.ddc
71 #Change names for Verilog
72 change_names -rule verilog -hierarchy
73
74 write_sdc -nosplit ./outputs/${DESIGN_NAME}_${CLK_PERIOD}ns.sdc
75 write -format verilog -hierarchy -output
76                 ./outputs/${DESIGN_NAME}_${CLK_PERIOD}ns.v
77 write_parasitics -format distributed -output
78                 ./outputs/${DESIGN_NAME}_${CLK_PERIOD}ns.spef
79 write_sdf ./outputs/${DESIGN_NAME}_${CLK_PERIOD}ns.sdf
```

3.6 State of Verilog Code

The Verilog code matches the Matlab results 100% and have passed all tests.

4 Synthesis Results

The following are the synthesis results obtained when the Verilog code was pushed thru Synopsis Design compiler. It shows the AT diagram created in order to find the optimal clock speed to use for the final design. It then includes the synopsis reports generated when synthesizing the design using the selected clock constraint.

4.1 Block Optimization

In an attempt to optimize the block, the first thing is to try and reduce the size of the block. It was found that the IP core cosine block takes a lot of space and there are currently two of these blocks. Different possible solutions to shrink their impact on the size of the block was looked at. One possible solution is by making use of pipelines that will enable the use of one of these blocks for both sin and cos functionality. The second possible solution is to shrink the size of the input to these blocks and also reduce the amount of bits output needed which would shrink the necessary size of these blocks. The second solution was taken and by making use of Experiment 3, documented above, the minimum bits needed for the input and output of these blocks that did not effect the accuracy of the system to much was found and implemented. In this optimization step the input bits where reduced from 32 bits to 16 bits and the output from 32 bits to 19 bits, the precise quantizations can be seen in Table 1.

The divider was extremely large since it was a 64 bit divider that calculated the output every clock cycle. The first idea to optimize this was to look at other IP core dividers and seeing if implementing another divider such as a pipeline divider or sequential divide may reduce the area used by the divider. The second idea to optimize the size taken up by the divider was to reduce the amount of bits input to the divider. The optimization that was selected and implemented was a combination of both the ideas. By implementing a sequential divider that uses 15 clock cycles to generate an output the size was reduced greatly. Then by implementing a better quantization policy reducing the input bits of the divider from 32 to 20 bits and the output from 32 bits 20 bits the size taken by the divider was further reduced.

General quantization optimizing was also done reducing the amount of used bits where possible without sacrificing precision. This did not have such a significant impact on the size but with the above two major size optimizations the block was reduced to an acceptable size.

To optimize the critical path of the block and reduce unnecessary operations, a lot of flip flops were added. There were flip flops added after both the cosine IP cores to reduce the critical path. A flip flop was added after the divider to reduce the critical path and it is also necessary to control the output from the divider to the main feedback loop since the divider will only generate a new output every 15 clock cycles. Flip flops were also added at the end of every filter coefficient calculation in order to control the change of parameters to the main loop. This allows that when input parameters are changed that new filter coefficients are only passed to the filter once the divider is ready to pass its newly calculated coefficient as well.

These added flip flops reduced the critical path so that the critical path is no longer in the filter coefficient calculation part of the circuit. The critical path is still quite long but it is now found inside the main filter loop but it is low enough to be acceptable.

4.2 Synchronous Design

Fig. 20 shows that the synthesized design only makes use of Flip Flops exclusively and there are no Latches in the design.

```
Inferred memory devices in process
        in routine FF_DATA_WIDTH24 line 28 in file
                '../src/FF.v'.
===============================================================================
|    Register Name    |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
===============================================================================
|      Q_DO_reg       | Flip-flop |  24   |  Y  | N  | Y  | N  | N  | N  | N  |
===============================================================================
Presto compilation completed successfully.
```

Figure 20: Design report showing only Flip Flops used

4.3 AT-Tradeoff Plot

In Fig. 21 the logarithmic area time diagram of the FLT block is shown. In order to create this figure we ran synopsis with varying clock constraints ranging from 100*ns* to 170*ns*.

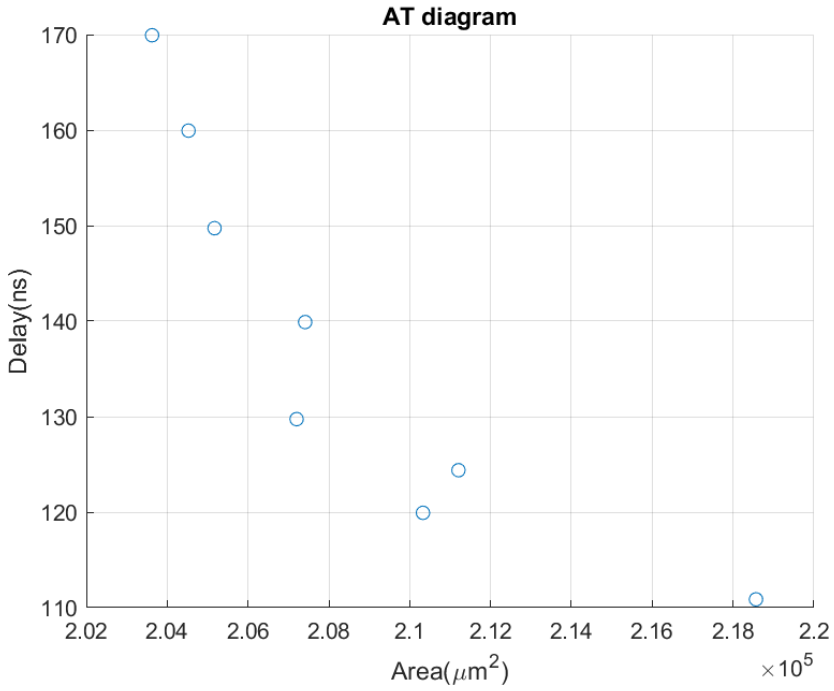


Figure 21: The AT diagram of the FLT block

From the diagram we choose the clock constraint as 120*ns* because it was the lowest clock constraint that had no violations.

4.4 Area, Delay, and Power Results

In the below tables the results when the clock constraint was set to 120*ns* is shown because it had no violations. For the maximum timing report of our design we obtained the following results that can be seen in Table 6.

Table 6: Table of maximum timing report for 120ns

Data required time	119.93
Data arrival time	-119.93
Slack (MET)	0.01

For the minimum timing report of our design we obtained the following results that can be seen in Table 7.

Table 7: Table of minimum timing report for 120ns

Data required time	0.01
Data arrival time	-0.35
Slack (MET)	0.33

For the are report of the design the following results shown in Table 8 where obtained.

Table 8: Table of area report for 120ns

Combinational	217556.583739
Noncombinational	14846.054619

Table 9 shows the summary of the Synthesis results obtained.

Table 9: Summary of Synthesis with Synopsis Design Compiler.

Metric	Result
Total cell area	232402.638358 μm^2
Total core area	217556.583739 μm^2
Clock frequency	8.333 MHz

4.5 State of Synthesis

The block has been successfully synthesized with reasonable results.

5 Backend: Place and Route

In this section, we show all the results we obtained from backend design with Cadence Innovus.

5.1 Floor-planning

In this design, we use 64 input pins and 24 output pin and the area is $490000\mu m^2$. The estimated density is 52.527% .

5.2 Area, Delay, and Power Results

The clock constraint we use for this design is 120ns and the final critical path is 49.718ns. Corresponding metrics can be found in Tbl. 10. Details can be found in the timing report files in the zip folder.

Table 10: Summary of Place and Route with Cadence Innovus.

Metric	Result
Total area	$49000\,\mu m^2$
Cell density	52.83 %
Clock frequency	20.11 MHz

As shown in the final timing report, the total negative slack is 0ns and the number of violating paths is also 0, thus there is no setup violation in our design. Similar proof can be found in the hold time hold timing report that our design does not have any hold violation either. The screenshot of these two reports are shown in Fig. 22 and Fig. 23.

The critical path from the report is shown in Fig. 24 and illustrated in Fig. 25.

5.3 Backend Screenshot

The screenshot of our final chip is shown in Fig. 26.

5.4 Design Rule Check (DRC)

Our design does not have and DRC violation. The proof can be found in Fig. 27.

5.5 Place and Route Commands

We follow the place and route commands given in class. We only changed the parameters listed in *TO DO* sections in `floorplan.tcl` and did not changed anything in `placement.tcl` and `routing.tcl`.

5.6 Post-Layout Simulation

The post-layout simulation results can be found in Fig. 28. The unexpected result at 3030ns are some glitches and the input and output then are still 0. Input comes in at 3600.2ns and the outputs of our post-layout circuit match the MATLAB golden model. Corresponding waveform is shown in Fig. 29.

We have run one regular tone provided in the `SynTech` folder and ten different random vectors generated by uniformly distributed random numbers `rand()`. We have also changed quality factor *par_FLT_RQ_D* at the 50th clock cycle and the divider works perfectly. All 6 parameters namely $a0 \sim b2$ are synchronized by the valid signal of the divider after changing *par_FLT_RQ_D*.

```
#####################################################################
#  Generated by:      Cadence Innovus 17.13-s098_1
#  OS:                Linux x86_64(Host ID vip-brg.ece.cornell.edu)
#  Generated on:      Sat Dec 14 20:47:39 2019
#  Design:            FLT
#  Command:           timeDesign -postRoute -expandedViews -outDir reports/08_timedesign_final
#####################################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):| 70.822  | 71.525  | 70.822  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|func_wc             | 70.822  | 71.525  | 70.822  |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|func_tc             | 106.679 | 106.916 | 106.679 |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|test_wc             | 430.822 | 431.525 | 430.822 |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+

+----------------+-----------------------------------+------------------+
|                |              Real                 |       Total      |
|    DRVs        +------------------+----------------+------------------|
|                |  Nr nets(terms)  |  Worst Vio     |  Nr nets(terms)  |
+----------------+------------------+----------------+------------------+
|   max_cap      |      0 (0)       |   0.000        |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000        |      0 (0)       |
|   max_fanout   |      0 (0)       |     0          |      0 (0)       |
|   max_length   |      0 (0)       |     0          |      0 (0)       |
+----------------+------------------+----------------+------------------+

Density: 52.830%
       (100.000% with Fillers)
------------------------------------------------------------
FLT_postRoute.summary.gz (END)
```

Figure 22: Setup timing reports.

5.7 State of Place and Route

The design placed and routed successfully and the post-layout simulation works perfectly.

```
#############################################################
#  Generated by:      Cadence Innovus 17.13-s098_1
#  OS:                Linux x86_64(Host ID vip-brg.ece.cornell.edu)
#  Generated on:      Sat Dec 14 20:47:48 2019
#  Design:            FLT
#  Command:           timeDesign -postRoute -hold -expandedViews -outDir reports/08_timedesign_final
#############################################################

------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.025  |  0.105  |  0.025  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|hold_bc             |  0.025  |  0.105  |  0.025  |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|hold_tc             |  0.049  |  0.202  |  0.049  |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+
|hold_wc             |  0.032  |  0.740  |  0.032  |
|                    |  0.000  |  0.000  |  0.000  |
|                    |    0    |    0    |    0    |
|                    |   948   |   437   |   638   |
+--------------------+---------+---------+---------+

Density: 52.830%
       (100.000% with Fillers)
------------------------------------------------------------
```

Figure 23: Hold timing reports.

```
#####################################################################
#  Generated by:      Cadence Innovus 17.13-s098_1
#  OS:                Linux x86_64(Host ID vip-brg.ece.cornell.edu)
#  Generated on:      Sat Dec 14 20:47:44 2019
#  Design:            FLT
#  Command:           timeDesign -postRoute -expandedViews -outDir reports/08_timedesign_final
#####################################################################
Path 1: MET Late External Delay Assertion
Endpoint:   sta_FLT_Out_DO[22]      (v) checked with  leading edge of 'Clk_clk'
Beginpoint: FF_FLT_IN/Q_DO_regx1x/Q (^) triggered by  leading edge of 'Clk_clk'
Path Groups: {default}
Analysis View: func_wc
Other End Arrival Time          0.000
- External Delay                0.000
+ Phase Shift                120.000
+ CPPR Adjustment              0.000
= Required Time               120.000
- Arrival Time                 49.178
= Slack Time                   70.822
     Clock Rise Edge                      0.000
     + Drive Adjustment                   0.043
     = Beginpoint Arrival Time            0.043
```

Figure 24: Critical path.

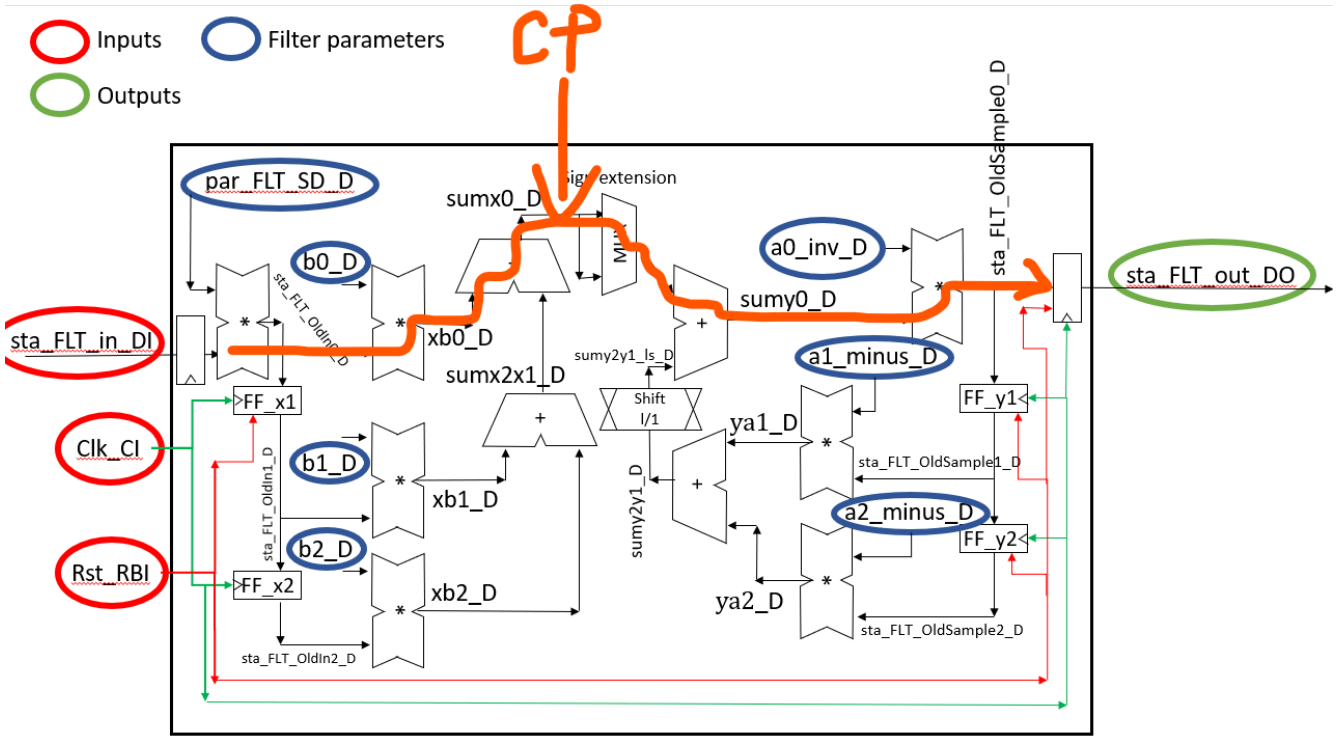


Figure 25: Critical path in block diagram.

References

[1] R. B. Johnson, "Cookbook formulae for audio equalizer biquad filter coefficients," `https://www.w3.org/2011/audio/audio-eq-cookbook.html`

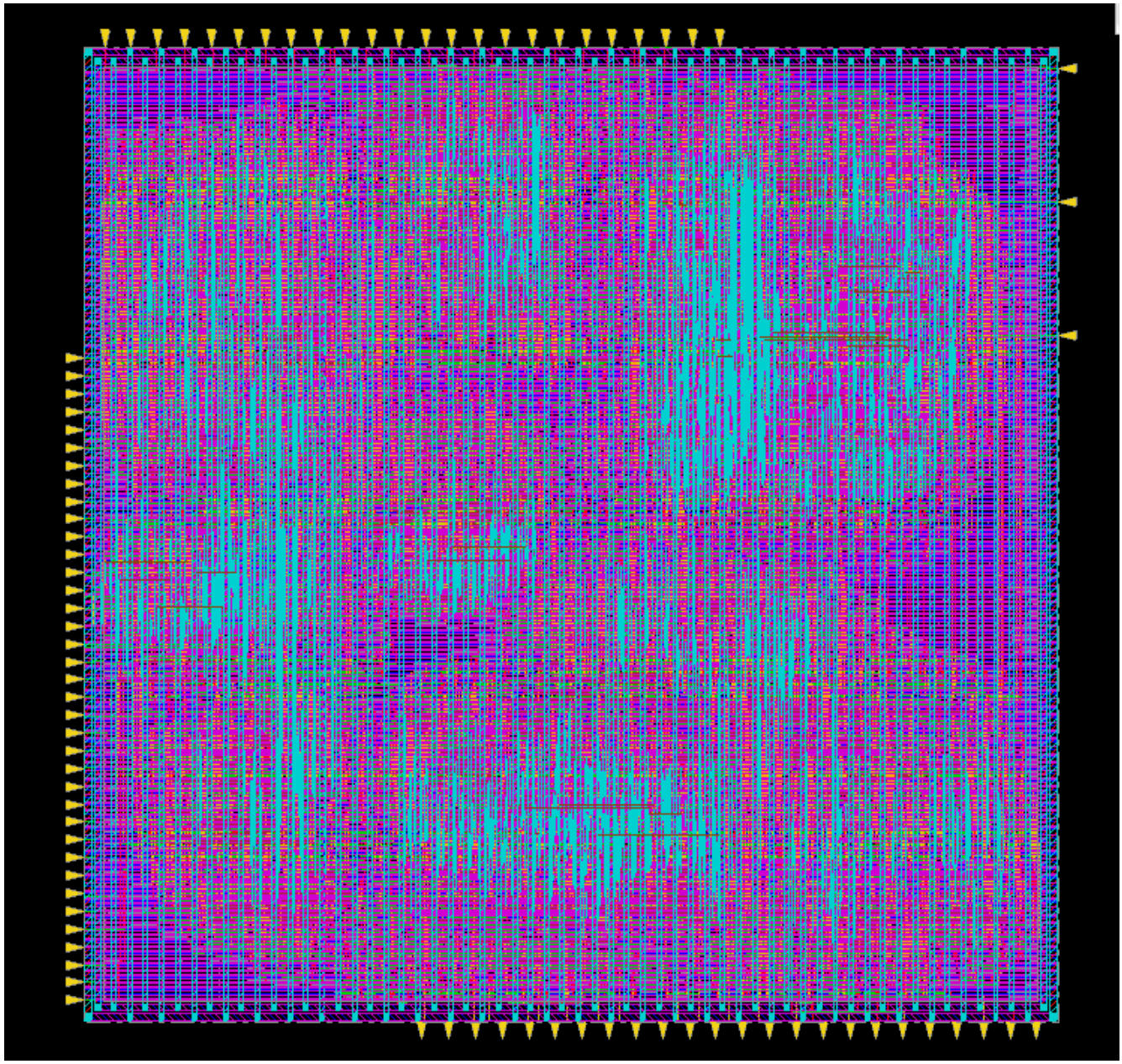


Figure 26: Final chip screenshot.

```
*** Starting Verify DRC (MEM: 2135.6) ***

 VERIFY DRC ..... Starting Verification
 VERIFY DRC ..... Initializing
 VERIFY DRC ..... Deleting Existing Violations
 VERIFY DRC ..... Creating Sub-Areas
 VERIFY DRC ..... Using new threading
VERIFY DRC ..... Sub-Area: {0.000 0.000 140.400 140.400} 1 of 25  Thread : 0
VERIFY DRC ..... Sub-Area: {561.600 0.000 699.840 140.400} 5 of 25  Thread : 1
VERIFY DRC ..... Sub-Area: {280.800 0.000 421.200 140.400} 3 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {0.000 280.800 140.400 421.200} 11 of 25  Thread : 2
VERIFY DRC ..... Sub-Area: {280.800 280.800 421.200 421.200} 13 of 25  Thread : 0
VERIFY DRC ..... Sub-Area: {561.600 140.400 699.840 280.800} 10 of 25  Thread : 1
VERIFY DRC ..... Sub-Area: {140.400 0.000 280.800 140.400} 2 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {0.000 421.200 140.400 561.600} 16 of 25  Thread : 2
VERIFY DRC ..... Sub-Area: {280.800 421.200 421.200 561.600} 18 of 25  Thread : 0
VERIFY DRC ..... Sub-Area: {561.600 280.800 699.840 421.200} 15 of 25  Thread : 2
VERIFY DRC ..... Sub-Area: {421.200 0.000 561.600 140.400} 4 of 25  Thread : 1
VERIFY DRC ..... Sub-Area: {0.000 140.400 140.400 280.800} 6 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {421.200 561.600 561.600 699.840} 24 of 25  Thread : 1
VERIFY DRC ..... Sub-Area: {140.400 140.400 280.800 280.800} 7 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {140.400 421.200 280.800 561.600} 17 of 25  Thread : 0
VERIFY DRC ..... Sub-Area: {421.200 280.800 561.600 421.200} 14 of 25  Thread : 2
VERIFY DRC ..... Sub-Area: {561.600 561.600 699.840 699.840} 25 of 25  Thread : 1
VERIFY DRC ..... Sub-Area: {0.000 561.600 140.400 699.840} 21 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {140.400 280.800 280.800 421.200} 12 of 25  Thread : 0
VERIFY DRC ..... Sub-Area: {421.200 140.400 561.600 280.800} 9 of 25  Thread : 2
VERIFY DRC ..... Sub-Area: {140.400 561.600 280.800 699.840} 22 of 25  Thread : 3
VERIFY DRC ..... Sub-Area: {421.200 421.200 561.600 561.600} 19 of 25  Thread : 1
VERIFY DRC ..... Thread : 0 finished.
VERIFY DRC ..... Sub-Area: {280.800 140.400 421.200 280.800} 8 of 25  Thread : 2
VERIFY DRC ..... Thread : 2 finished.
VERIFY DRC ..... Sub-Area: {561.600 421.200 699.840 561.600} 20 of 25  Thread : 1
VERIFY DRC ..... Thread : 1 finished.
VERIFY DRC ..... Sub-Area: {280.800 561.600 421.200 699.840} 23 of 25  Thread : 3
VERIFY DRC ..... Thread : 3 finished.

 Verification Complete : 0 Viols.

*** End Verify DRC (CPU: 0:00:07.3  ELAPSED TIME: 2.00  MEM: 38.0M) ***
```

Figure 27: DRC violation check.

```
VSIM 1> do wave.do
VSIM 2> run 3.6us
# [                3030] FLT_Out_DO :: Value 16777214 Expected        0
VSIM 3> run -a
# ** Note: $finish    : ../tb/FLT_POST_TB.v(109)
#    Time: 123510200 ps  Iteration: 0  Instance: /FLT_POST_TB
# 1
# Break in Module FLT_POST_TB at ../tb/FLT_POST_TB.v line 109
```

Figure 28: Results of post-layout simulation.

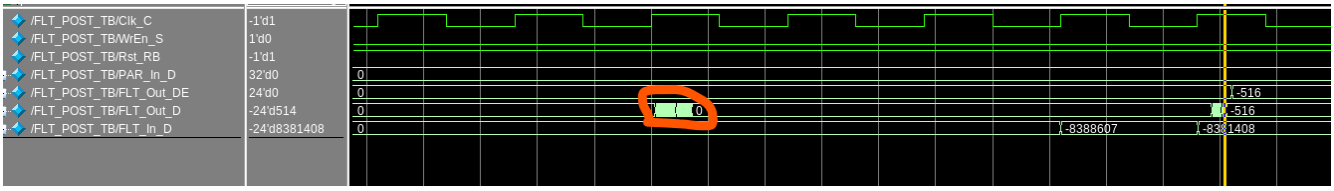


Figure 29: Waveform of post-layout simulation.

A Included Files in Zip-Folder

Provide a folder structure of all the files that you included (only folders, no files listed here). You can use the `tree` command or on a Mac there is also `find .` to quickly extract the tree structure. Here's a bad example:

```
.
+-- matlab_source
+-- verilog_source
    +-- IO_generator
    +-- modelsim
    +-- par
    ¦   +-- FLT_via_layer_VIA1.htmreports
    ¦   +-- FLT_via_layer_VIA2.htmreports
    ¦   +-- FLT_via_layer_VIA3.htmreports
    ¦   +-- FLT_via_layer_VIA4.htmreports
    ¦   +-- FLT_via_layer_VIA5.htmreports
    ¦   +-- FLT_via_layer_VIA6.htmreports
    ¦   +-- FLT_via_layer_VIA7.htmreports
    ¦   +-- FLT_via_layer_VIA8.htmreports
    ¦   +-- out
    ¦   +-- reports
    ¦   +-- save
    ¦   +-- scripts
    ¦   +-- src
    ¦   +-- timingReports
    +-- src
    ¦   +-- work
    +-- syn
    ¦   +-- alib-52
    ¦   +-- reports
    ¦   +-- scripts
    ¦   +-- work
    +-- tb

27 directories
```

B Design Review Slides

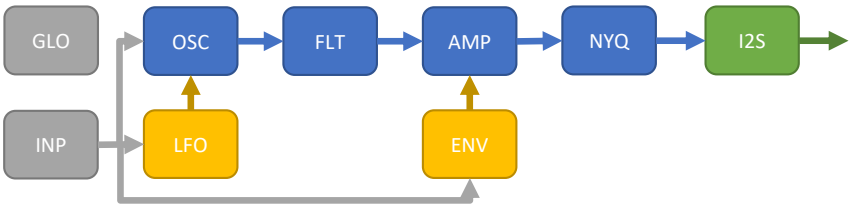
The following are the slides used for the design review presentation.

FLT Group

Hoachuan Song hs994

Frans Fourie fjf46

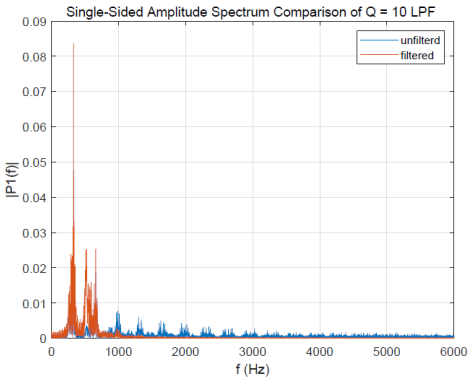
SynTech overview

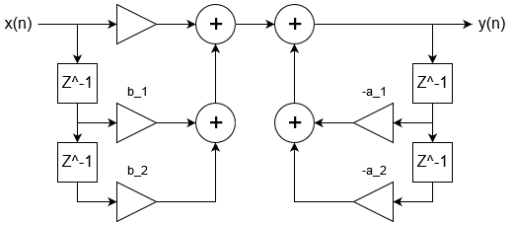


- We are doing the FLT Block; It is the second block receiving the signal from the Oscillator.
- In this module, the oscillator signals are processed through the IIR low-pass filter. The IIR filter aims to shape the harmonic structure of the oscillator. It will keep the desired frequency part of the original signal to be passed on to the next module.

Purpose of block (1)

- Purpose is to remove harmonics
- Calculate Parameters and then implement an IIR low-pass filter





Purpose of block (2)

- Input Parameters

- Output Parameters

Table 1: Input Parameters

Input Parameter	Description
sta_FLT_in_DI	This is the audio input signal we receive from the oscillator block that we will apply our filter to.
WrEn_SI	The signal that enables writing to the memory.
Addr_DI	This signal indicates to which address in the memory to write to.
PAR_In_DI	This is the parameter input to write into the memory.
Clk_CI	This is the chip clock input signal that will be used in our filter
Rst_RBI	This is the chip reset signal used to reset entire chip

Table 3: Output Parameters

Output Parameter	Description
sta_FLT_out_DO	This is the filtered signal our block passes to the next block.

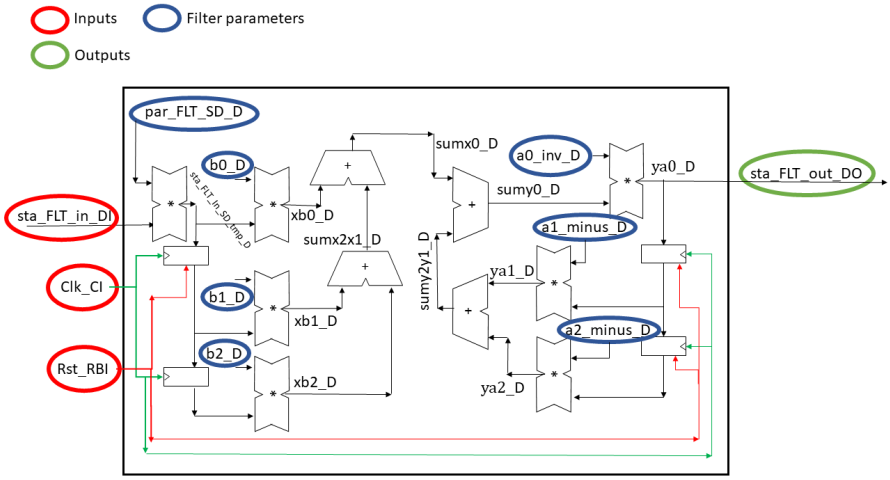
Purpose of block (3)

- Memory Parameters

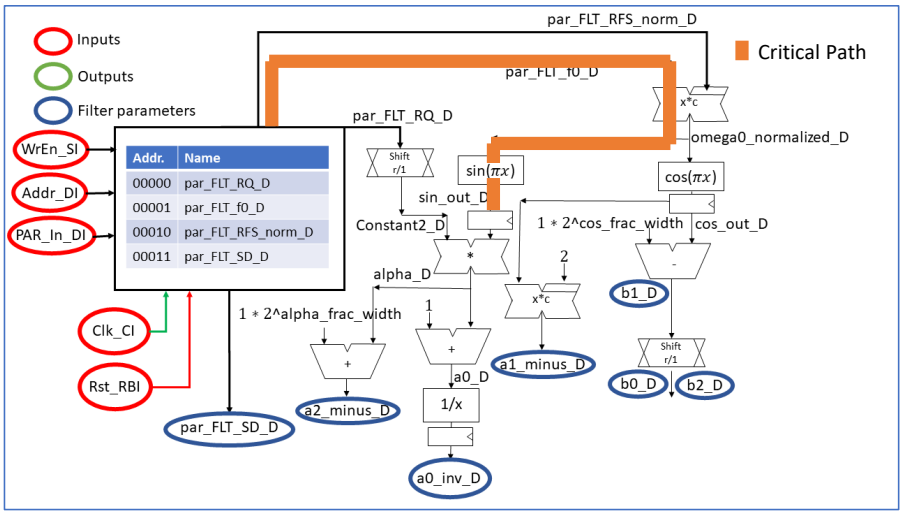
Table 2: Memory Parameters

Memory Parameter	Description
par_FLT_f0_D	This is the cut-off frequency of the filter. It can be set to a value that will be used as the filters cut-off frequency. The value for which the filters output is -3dB of the nominal pass-band value.
par_FLT_RQ_D	This is the reciprocal of the filters Quality factor. It can currently be set to any value but will likely be restrained to a certain range in the final design
par_RFS_D	Two divided by the internal sampling frequency without pi
par_FLT_SD_D	The scaling down factor that will be used on the input signal to the filter block
par_FLT_type_D1	This parameter can be used to specify what type of filter will be used for example a low-pass or high pass filter. When type=0 it is just a wire passing signals. (Will be used for future features if there is time)

Block diagram(1)

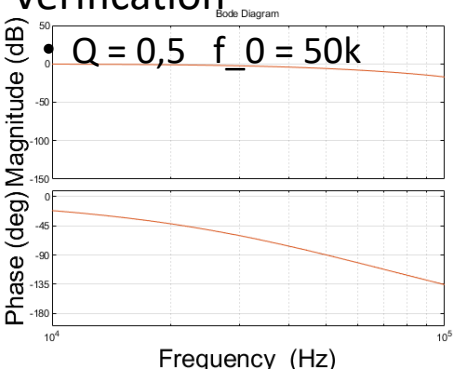


Block diagram(2)

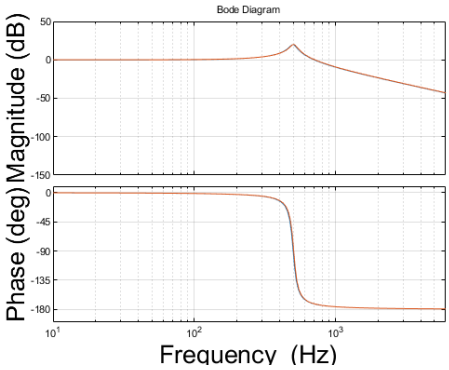


Design verification(1)

- Matlab Code Verification
  - Q = 0,5 f_0 = 50k



- Matlab Code Verification
  - Q = 10 f_0 = 500



Design verification(2)

- Functionality of Verilog Code
  - Verilog Code results are match expected results from Test Bench

```
# <<< :D All outputs match the expected results! :D >>>
# ** Note: $finish    : ../tb/FLT_TB.v(133)
#    Time: 1029200 ps  Iteration: 0  Instance: /FLT_TB
```

- Verilog Code verification
  - Every parameter and every intermediate result has been verified against the Matlab Golden Model
- Design Tests
  - 4 Types of tests have been done and passed
  - Input from Oscillator, Random Input, Changing Q value, Changing cut-off frequency value
  - Different values of Q ranging from 0,5 - 20 and different values for the cut-off frequency 500Hz – 50kHz
  - More Tests Needed

Design synthesis

- Report area

	Old FLT	16ns Sin Area	FLT
Combinational area:	444509.80	12235.16	222842.88
Buf/Inv area:	30911.38	658.02	10449.10
Noncombinational area:	8995.73	612.86	13847.96
Total cell area:	453505.53	12848.02	236690.84
Total area:	536887.79	13490.10	275997.69

Design synthesis

- Report delay

	Old FLT	16ns Sin Area	FLT
data arrival time:	379.92	15.29	109.90

- Performed optimizations
  - General Quantization optimization
  - Quantization optimization to reduce size of IP-Cores (sin, cos and divider)
  - Implemented a Sequential divider
  - Inserted Flip Flops

What is missing?

- Explain what is missing for your design
  - MATLAB = YES
  - Fixed-point optimization = YES
  - Verilog code = YES
  - Testbench = YES
  - Design synthesis = YES
  - Optimization of area = YES
  - Optimization of critical path = Work in Progress
  - Answer this: Can we take your Verilog code and integrate with the other group's code and it will just work = YES

Who was doing what?

- <Haochuan and FJ>: Wrote Matlab Code and verified
- <Haochuan and FJ>: Wrote Verilog Code and verified Using ModelSim
- <Haochuan>: Implementation of Sequential divider
- <FJ>: Block Diagram
- <Haochuan>: Generated stimuli and wrote testbench
- < Haochuan and FJ >: Used Synopsys design compiler to synthesize design

C MATLAB Code

The following is the main Matlab file used to simulate the FLT block.

```
% ========================================================================
% ECE 5746 - Simple Filter Model
% (c) 2019 hs994,fjf46@cornell.edu
%
% Author: Haochuan Song, Frans Fourie
% Last edited: 10/23/2019
% Project: SynTech
%
% ---Description-------------------------------------------------
% A second order IIR filter supports low-pass or direct pass
% ---I/O specifications-----------------------
% sta.FLT.In_DI                      input data
% sta.FLT.Out_DO                output data
%----parameter specifications-----------------
% par_FLT_RQ_D;                                 1/Q: reciprocal of quality factor of low-pass fil
% par_FLT_f0_D;                                 cut-off frequency, stored in parameter memory
% par_FLT_RFS_norm_D;                  2/Fs: Normalized reciprocal of Internal Sampling Frequency
% par_FLT_SD_D;                                 scaling down factor corresponding to quality fact
% par_FLT_type_S;                          control signal for different filter types; haven't used
%
% ========================================================================

function sta = FLT(par,sta)
%
% % example of first-order IIR filter
% % reads directly from OSC output state sta.OSC.Out_DO
% sta.FLT.Out_DO = (1-par.FLT.Alpha_DI)*sta.FLT.Out_DO + par.FLT.Alpha_DI*sta.OSC.Out_DO;

% specify all fixed-point parameters
%-------------------------------------------------------------------------
FixP_IIR_CAL = {0,31,'s'}; % {I,F,'s'} where 's' is signed
QType_IIR_CAL = 'SatTrc'; % we saturate and round

FixP_Q_CAL = {3,28,'s'}; % {I,F,'s'} where 's' is signed
QType_Q_CAL = 'SatTrc'; % we saturate and round

FixP_para_omega0 = {0,15,'s'};
QType_para_omega0 = 'SatTrc';

FixP_para_sincos = {0,18,'s'};

FixP_para_alpha = {0,31,'s'};
QType_para_alpha = 'SatTrc';
```

```
44
45 FixP_weight_b0 = {0,19,'s'};
46 QType_weight_b0 = 'SatTrc';
47
48 FixP_weight_b1 = {1,18,'s'};
49 QType_weight_b1 = 'SatTrc';
50
51 FixP_weight_a0 = {1,18,'s'};
52 QType_weight_a0 = 'SatTrc';
53
54 FixP_INV_CAL = {0,19,'s'};
55
56 FixP_weight_a1 = {2,29,'s'};
57 QType_weight_a1 = 'SatTrc';
58
59 FixP_weight_a2 = {0,31,'s'};
60 QType_weight_a2 = 'SatTrc';
61
62 FixP_in = {0,31,'s'}; % {I,F,'s'} where 's' is signed
63 QType_in = 'SatTrc'; % we saturate and round
64
65 FixP_out = {0,23,'s'}; % {I,F,'s'} where 's' is signed
66 QType_out = 'SatTrc'; % we saturate and round
67
68 FixP_inter_y = {1,30,'s'};
69 QType_inter_y = 'SatTrc';
70
71 FixP_inter_x = {0,31,'s'};
72 QType_inter_x = 'SatTrc';
73
74 FixP_add_inter = {1,31,'s'};
75 QType_add_inter = 'SatTrc';
76
77 % update FLT state
78 %-------------------------------------------------------------------------------------------
79 sta.FLT.In_DI = sta.OSC.Out_DO; % reads directly from OSC output state sta.OSC.Out_DO
80 par.FLT.SD_D = RealRESIZE(par.FLT.SD_D,FixP_IIR_CAL,QType_IIR_CAL);
81 sta.FLT.In_SD_tmp_D = par.FLT.SD_D*sta.FLT.In_DI;
82 sta.FLT.In_SD_tmp_D = RealRESIZE(sta.FLT.In_SD_tmp_D,{1,54,'s'},QType_in);
83 sta.FLT.OldIn_D(1:2) = sta.FLT.OldIn_D(2:3);            % update the input memory
84 sta.FLT.OldIn_D(3) = RealRESIZE(sta.FLT.In_SD_tmp_D,FixP_in,QType_in);
85 sta.FLT.OldSample_D(1:2) = sta.FLT.OldSample_D(2:3);    % update the output memory
86
87 % parameter calculation
88 % ------------------------------------------------------------------
89 switch par.FLT.type_S
```

```
    case 1 % low pass filter
        constant1 = RealRESIZE(par.FLT.RFS_norm_D,FixP_IIR_CAL,QType_IIR_CAL); %2/internal sampli
        omega0_normalized_D_fixed = RealMULT(2*par.FLT.f0_D/2^10,constant1,FixP_para_omega0, QTyp
        par.FLT.RQ_D = RealRESIZE(par.FLT.RQ_D,FixP_Q_CAL,QType_Q_CAL);
        constant2_D = RealMULT(0.5,par.FLT.RQ_D,FixP_Q_CAL,QType_Q_CAL);
        sin_out_D =  RealRESIZE(sin(pi*(omega0_normalized_D_fixed)), FixP_para_sincos,  QType_IIR
        alpha_D_fixed = RealMULT(constant2_D,sin_out_D, FixP_para_alpha, QType_para_alpha);
        cos_out_D =  RealRESIZE(cos(pi*(omega0_normalized_D_fixed)), FixP_para_sincos,  QType_IIR

        b1_D_fixed = RealSUB(1,cos_out_D , FixP_weight_b1, QType_weight_b1);
        b0_D_fixed = RealMULT(0.5,b1_D_fixed, FixP_weight_b0, QType_weight_b0);
        b2_D_fixed = b0_D_fixed;
        a0_D_fixed = RealADD(1,alpha_D_fixed, FixP_weight_a0, QType_weight_a0);
        a0_inv_D_fixed = RealRESIZE(1/a0_D_fixed, FixP_INV_CAL, QType_IIR_CAL);
        a1_minus_D_fixed = RealMULT(2, cos_out_D, FixP_weight_a1, QType_weight_a1);
        a2_minus_D_fixed = RealSUB(alpha_D_fixed, 1, FixP_weight_a2, QType_weight_a2);

        % feedback loop
        %-------------------------------------------------------------------------
        xb0_D = RealMULT(b0_D_fixed, sta.FLT.OldIn_D(3), FixP_inter_x, QType_inter_x);
        xb1_D = RealMULT(b1_D_fixed, sta.FLT.OldIn_D(2), FixP_inter_x, QType_inter_x);
        xb2_D = RealMULT(b2_D_fixed, sta.FLT.OldIn_D(1), FixP_inter_x, QType_inter_x);
        ya1_D = RealMULT(a1_minus_D_fixed, sta.FLT.OldSample_D(2), FixP_inter_y, QType_inter_y);
        ya2_D = RealMULT(a2_minus_D_fixed, sta.FLT.OldSample_D(1), FixP_inter_y, QType_inter_y);

        sumx2x1_D = RealADD(xb1_D, xb2_D, FixP_inter_x, QType_inter_x);
        sumx0_D = RealADD(sumx2x1_D, xb0_D, FixP_inter_x, QType_inter_x);
        sumy2y1_D = RealADD(ya2_D, ya1_D, FixP_inter_y, QType_inter_y);

        sumx0_RS_D = RealRESIZE(sumx0_D,FixP_add_inter ,QType_add_inter);
        sumy2y1_LS_D = RealRESIZE(sumy2y1_D,FixP_add_inter ,QType_add_inter);
        sumy0_tmp_D = RealADD(sumx0_RS_D, sumy2y1_LS_D, FixP_add_inter, QType_add_inter);
        sumy0_D =  RealRESIZE(sumy0_tmp_D, FixP_inter_y, QType_inter_y);
%         sumy0_D = RealADD(sumx0_D, sumy2y1_D, FixP_inter_y, QType_inter_y);
        sta.FLT.OldSample_D(3) = RealMULT(a0_inv_D_fixed, sumy0_D, FixP_IIR_CAL, QType_IIR_CAL);
        %-------------------------------------------------------------------------------------

        % current output of 2nd order LPF
        sta.FLT.Out_DO = RealRESIZE(sta.FLT.OldSample_D(3), FixP_out, QType_out);
    case 0
        sta.FLT.Out_DO = sta.FLT.In_DI;
end

end
```

The following is the Matlab initialization file.

```
% ========================================================================
% ECE 5746 - Simple Filter Model (INITIALIZATION)
% (c) 2019 hs994,fjf46@cornell.edu
%
% Author: Haochuan Song, Frans Fourie
% Last edited: 10/23/2019
% Project: SynTech
%
% ---Description----------------------------
% Initialize a second order IIR filter
% ---I/O specifications-----------------------
% sta.FLT.In_DI                       input data
% sta.FLT.Out_DO                 output data
%----parameter specifications-----------------
% par_FLT_RQ_D;                                 1/Q: reciprocal of quality factor of low-pass fil
% par_FLT_f0_D;                                 cut-off frequency, stored in parameter memory
% par_FLT_RFS_norm_D;                  2/Fs: Normalized reciprocal of Internal Sampling Frequency
% par_FLT_SD_D;                                 scaling down factor corresponding to quality fact
% par_FLT_type_S;                        control signal for different filter types; haven't used
%
% ========================================================================

function [par,sta] = FLT_init(par,sta)

% FLT parameters
par.FLT.f0_D = 500;   %cut-off frequency of low-pass filter
Q = 1;              % quality factor of low-pass filter
par.FLT.RQ_D = 1/Q; % Reciprocal of Q, which is going to be used in the FLT.m
%-------------------------------------------------------------------------------
% 2/Reciprocal of Internal Sampling Frequency, which is going to be used in the FLT.m
% 2e10 here is to fully utilize the dynamic range, otherwise it will suffer
% a lot from qunatization error bc par.GLO.FSInt_DI is too large
par.FLT.RFS_norm_D = (2/par.GLO.FSInt_DI)*2^10;
%-------------------------------------------------------------------------------
par.FLT.type_S = 1; %type 1 indicates low pass filter
par.FLT.SD_D = 1/(1+(Q/13)); % scaling down factor; stored in the memory


% all state variables used by this block must be initialized
sta.FLT.In_DI = 0;
sta.FLT.OldIn_D = zeros(1,3);         % size = [1 3] input memory
sta.FLT.Out_DO = 0;
sta.FLT.OldSample_D = zeros(1,3); % keep old output sample, size = [1 3] output memory

```

```
end
```

The following is the Matlab file used to generate test values.

```
% ------------------------
% TestAdd.m
% Generates the stimuli and expected-output files for FLT
% Author: Hcoahuan Song (hs994@cornell.edu), Oscar Castaneda (oc66@cornell.edu)
% ------------------------

%Parameters
clear
clc

simname = 'FLT_IO_rnd2.mat';
mode = 'RND';

load('InpPara.mat')
load(simname)
trials = 5;
n_bits = 31;
input_file = ['..\tb\',mode,'_FLT_in.txt'];
output_file = ['..\tb\',mode,'_FLT_out.txt'];
% inter_file = '..\tb\FLT_inter.txt';

%Open files
fin = fopen(input_file,'w');
fout = fopen(output_file,'w');
% finter = fopen(inter_file,'w');

%Apply reset
fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
fprintf(fout,'%s \n','x');
fprintf(fin,'%d %d %d %d %d\n',[0,0,0,0,0]);
fprintf(fout,'%s \n','x');
fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
fprintf(fout,'%s \n','x');

%Feed in parameters
%  Rst_RB, WrEn_S, Addr_D, PAR_In_D, FLT_In_D
fprintf(fin,'%d %d %d %10.0f %d\n',[1,1,0,par.FLT.RQ_D*2^28,0]);  % 1/Q
fprintf(fout,'%d\n',0);
fprintf(fin,'%d %d %d %d %d\n',[1,1,1,par.FLT.f0_D,0]); % f0
fprintf(fout,'%d\n',0);
fprintf(fin,'%d %d %d %10.0f %d\n',[1,1,2,constant1*2^n_bits,0]); % 2/Fs*2^10
fprintf(fout,'%d\n',0);
```

```
fprintf(fin,'%d %d %d %10.0f %d\n',[1,1,3,par.FLT.SD_D*2^n_bits,0]); % scaling down factor; also
fprintf(fout,'%d\n',0);
fprintf(fin,'%d %d %d %d %d\n',[1,1,4,par.FLT.type_S,0]); % type selection;also input;
fprintf(fout,'%d\n',0);

for i = 1:20
    fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]); % input(2)
    fprintf(fout,'%d\n',0); % output(2)
end

for i = 1:length(FLT_DI)
    fprintf(fin,'%d %d %d %d %7.0f\n',[1,0,0,0,FLT_DI(i)*2^23]); % input(2)
    fprintf(fout,'%d\n',FLT_DO(i)*2^23); % output(2)
end

% fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,FLT_DI(3)*2^23]); % input(3)
% fprintf(fout,'%d\n',FLT_DO(3)*2^23); % output(3)

% % Reset to 0
% fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
% fprintf(fout,'%s \n','x');
% fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
% fprintf(fout,'%s \n','x');
% fprintf(fin,'%d %d %d %d %d\n',[0,0,0,0,0]);
% fprintf(fout,'%s \n','x');
% fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
% fprintf(fout,'%s \n','x');
% fprintf(fin,'%d %d %d %d %d\n',[1,0,0,0,0]);
% fprintf(fout,'%s \n','x');

% Intermidiate results
% fprintf(finter,'%10.0f %10.0f %10.0f %10.0f\n',[constant2_D*2^31, alpha_D_fixed*2^31, a2_minus_


% %Random cases
% for i=1:trials
%   %Generate inputs
%   A = randi(2^n_bits)-1;
%   B = randi(2^n_bits)-1;
%   Cin = randi(2)-1;
%   %Generate expected output
%   FullSum = A+B+Cin;
%   Sum = mod(FullSum+(i==2^2*n_bits),2^n_bits);
%   Cout = floor((FullSum+(i==2^2*n_bits))/(2^n_bits));
%   %Write to files
%   fprintf(fin,'%d %d %d %d %d\n',[1,1,A,B,Cin]);
```

```
89 %   fprintf(fout,'%d %d\n',[Sum,Cout]);
90 % end
91
92 %Close files
93 fclose(fin);
94 fclose(fout);
95 % fclose(finter);
```

D Verilog Code

The following is the main Verilog code.

```
//================================================================================
//ECE 5746 - Simple Filter Model
//(c) 2019 hs994,fjf46@cornell.edu
//
//Author: Haochuan Song, Frans Fourie
//Last edited: 12/05/2019
//Project: SynTech
//
//---Description-------------------------------------------------
//A second order IIR filter supports low-pass or direct pass
//---I/O Description---------------------------------------------
// sta_FLT_In_DI            [24bit]    input data =  sta.OSC.Out_DO
// sta_FLT_Out_DO           [24bit]    output data
// ================================================================================

/* ------------------------------------------------------------------------------
// Module ports declaration
// ------------------------------------------------------------------------------ */
module FLT
#(
  // ---- The following parameters are the same for all blocks
          parameter ADDR_WIDTH   = 5,  // number of entries in the parameter memory is equal to 2
    parameter MEM_WIDTH    = 32,  // word length of each memory entry to store parameters
    // ---- the following parameters are FLT block specific
    parameter IN_WIDTH    = 24,   // width of FLT inputs
    parameter OUT_WIDTH   = 24,   // width of FLT outputs
    parameter COF_WIDTH   = 32, // width of FLT coefficients
          parameter COSIN_WIDTH = 19
)
(
  // ---- The following signals are the same for all blocks (no touchy!)
  input                                 Clk_CI,     // Clock signal
  input                                 Rst_RBI,    // Asynchronous active low reset
  input                                 WrEn_SI,    // Active high write enable
  input         [ADDR_WIDTH-1:0]        Addr_DI,    // Address of the parameter in the memory
  input         [MEM_WIDTH-1:0]         PAR_In_DI,  // Parameter input (written from external int
  // ---- The following signals are FLT block specific
  input  signed [IN_WIDTH-1:0]           sta_FLT_In_DI,  // Input to the block (block can read fr
  output signed [OUT_WIDTH-1:0]          sta_FLT_Out_DO // Output of the block (state). Use signe
);

/* ------------------------------------------------------------------------------
```

```
// Common parameters for all blocks (do not change)
// ------------------------------------------------------------------------------ */

localparam MEM_DEPTH = (1 << ADDR_WIDTH);   // You can have at most 2^ADD_WIDTH parameters per bl
integer i;                                  // temporary variable
// ---- Defines block parameter memory that can be written from the outside of your module
// ---- Creates a register array (memory) to hold the parameter of the block
reg  [MEM_WIDTH-1:0] parameter_memory [0:4];

// ------------------------------------------------------------------------------
// Internal parameters and signals/variables declaration in FLT
// Define the internal wires, regs and all other variables of your module
// ------------------------------------------------------------------------------ */

// ---- Assign a wire with descriptive names to each of these memory entries that hold a paramete
localparam  RQ_frac_WIDTH                = 31;
localparam  alpha_frac_WIDTH         = 31;
localparam        cos_frac_WIDTH                 = 30;
localparam  a0_frac_WIDTH                = 31;
localparam        RFS_shift_WIDTH        = 10;
localparam  DIV_WIDTH           = 20;

wire signed [COF_WIDTH-1:0] par_FLT_RQ_D;              // 1/Q: reciprocal of quality factor of l
wire signed [COF_WIDTH-1:0] par_FLT_f0_D;              // cut-off frequency, stored in parameter
wire signed [COF_WIDTH-1:0] par_FLT_RFS_norm_D;   // 2/Fs: Normalized reciprocal of Internal Samp
wire signed [COF_WIDTH-1:0] par_FLT_SD_D;              // scaling down factor corresponding to q
wire                        [COF_WIDTH-1:0] par_FLT_type_S;       // control signal for different f

assign  par_FLT_RQ_D = parameter_memory[0];       // par_FLT_RQ_D is the name of the first entry
assign  par_FLT_f0_D = parameter_memory[1];
assign  par_FLT_RFS_norm_D = parameter_memory[2];
assign  par_FLT_SD_D = parameter_memory[3];
assign  par_FLT_type_S = parameter_memory[4];

// ---- FLT block specific wires and registers
// -- Wires in combinational logic
wire signed [COSIN_WIDTH-1:0] sin_out_D;                              // output of sine block
wire signed [COF_WIDTH-1:0]   constant2_D;                      // 1/(2Q); right shift 1bit to p
wire signed [COF_WIDTH-1:0]   alpha_D;                                // alpha, intermediate par
wire signed [COF_WIDTH-1:0]   a2_minus_D;                             // -a2
wire signed [DIV_WIDTH-1:0]   a0_D;                                         // a0
wire signed [DIV_WIDTH-1:0]   a0_inv_D;                               // 1/a0
wire signed [16-1:0]          omega0_normalized_D;         // sin(pi*omega), called nomalized
wire signed        [COSIN_WIDTH-1:0]        cos_out_D;                                          // out
wire signed [COSIN_WIDTH:0] a1_minus_D;                                     // -a1
wire signed [COSIN_WIDTH:0] b1_D;                                               // b1
```

```
89  wire signed        [COSIN_WIDTH:0] b0_D;                                                    // b0
90  wire signed [COSIN_WIDTH:0] b2_D;                                                  // b2 = b0
91
92  wire signed [2*COF_WIDTH-1:0] omega0_normalized_tmp_D;       // 64bit temporary result for omega
93  wire signed [COSIN_WIDTH+1:0]        sin_out_tmp_D;                                // 21bit t
94  wire signed [COSIN_WIDTH+1:0]        cos_out_tmp_D;                                // 21bit t
95  wire signed [COF_WIDTH+COSIN_WIDTH:0]   alpha_tmp_D;    // 64bit temporary result for alpha
96  wire signed [2*DIV_WIDTH-1:0]   a0_inv_tmp_D;                              // 64bit temporary resu
97  wire signed [COF_WIDTH-1:0]          a0_rem_D;                                          // r
98  wire                                                                    a0_divby0_D;
99
100 // -- Wires in sequencial logic
101 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldIn0_D;                           // FF
102 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldIn1_D;                           // FF
103 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldIn2_D;                           // FF
104 wire signed        [COF_WIDTH-1:0]               xb0_D;
105 wire signed        [COF_WIDTH-1:0]               xb1_D;
106 wire signed        [COF_WIDTH-1:0]               xb2_D;
107 wire signed        [COF_WIDTH-1:0]               sumx2x1_D;
108 wire signed [COF_WIDTH-1:0]               sumx0_D;
109 wire signed [COF_WIDTH-1:0]               sumy2y1_D;
110 wire signed [COF_WIDTH-1:0]               sumy0_D;
111 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldSample0_D;          // FFy0,32b
112 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldSample1_D;               // FFy1,32b
113 wire signed        [COF_WIDTH-1:0]         sta_FLT_OldSample2_D;               // FFy2,32b
114 wire signed        [COF_WIDTH-1:0]               ya1_D;
115 wire signed        [COF_WIDTH-1:0]               ya2_D;
116
117 wire signed [IN_WIDTH-1:0]                              sta_FLT_IN_REG_D;        // to align th
118 wire signed [IN_WIDTH+COF_WIDTH-1:0]          sta_FLT_In_SD_tmp_D;// 24+32 = 56bit temporary resu
119 wire signed [COF_WIDTH+COSIN_WIDTH:0]        xb0_tmp_D;                          // 52bit temporary
120 wire signed [COF_WIDTH+COSIN_WIDTH:0]        xb1_tmp_D;                          // 52bit temporary
121 wire signed [COF_WIDTH+COSIN_WIDTH:0]        xb2_tmp_D;                          // 52bit temporary
122 wire signed [2*COF_WIDTH-1:0]                      ya0_tmp_D;                          // 64bit t
123 wire signed [COF_WIDTH+COSIN_WIDTH:0]  ya1_tmp_D;                          // 52bit temporary resul
124 wire signed [2*COF_WIDTH-1:0]                      ya2_tmp_D;                          // 64bit t
125 wire signed [COF_WIDTH:0]                          sumy0_tmp_D;               // 33bit tem
126
127 reg a0_enable_S;   // when to start sequintial divider
128 wire a0_valid_S;    // when the division is finished
129 wire signed [COSIN_WIDTH-1:0] cos_out_REG_D;
130 wire signed [COF_WIDTH-1:0]   a2_minus_REG_D;           // -a2_REG
131
132 /* -----------------------------------------------------------------------------
133 // Register Transfer Level (RTL) description of parameter memory (do not change)
134 // ----------------------------------------------------------------------------- */
```

```
135
136 // ---- Description of the parameter memory (same for all blocks)
137 always @(posedge Clk_CI or negedge Rst_RBI)
138 begin                                 // begin the memory control block
139   if (!Rst_RBI) begin  // Reset all the parameters upon Rst_RBI being deasserted low
140      parameter_memory[0] = 32'b0000_1100_1100_1100_1100_1100_1100_1100;
141          parameter_memory[1] = 500;
142          parameter_memory[2] = 32'b0000_0000_1010_1101_1010_1011_1001_1111;
143          parameter_memory[3] = 4;
144          parameter_memory[4] = 1;
145   end else if (WrEn_SI) begin
146     parameter_memory[Addr_DI] = PAR_In_DI; // Write parameter to memory if WrEn_SI is asserted hi
147   end
148 end                                           // end the memory control block
149
150 /* -----------------------------------------------------------------------------
151 // Register Transfer Level (RTL) description of FLT specific code
152 // ----------------------------------------------------------------------------- */
153
154 // ---------------------------------------------------------------------------------------------
155 // ******************Coefficients calculation Begin*****************************************
156 // ---------------------------------------------------------------------------------------------
157 assign omega0_normalized_tmp_D = par_FLT_f0_D * par_FLT_RFS_norm_D;
158 assign omega0_normalized_D = {omega0_normalized_tmp_D[63],omega0_normalized_tmp_D[39:25]};
159 /* Instance of DW_sincos-------------------------------------------------
160 sin_out_D = sin (pi * omega0_normalized_D)
161 ------------------------------------------------------------------------------*/
162 // calculate a1_minus_D, b1_D, b2_D
163 // calculate -a1
164 DW_sincos #(
165         .A_width(16),
166         .WAVE_width(COSIN_WIDTH+2),  // 21b = {s,1,19} 1 sign bit, 1 integer bit, 18 fraction bit
167         .arch(0),
168         .err_range(1))
169 FLT_cos (
170         .A(omega0_normalized_D),
171         .SIN_COS(1'b1), // 0 for sine 1 for cosine
172         .WAVE(cos_out_tmp_D) // {s,1,19}
173 );
174 assign cos_out_REG_D = {cos_out_tmp_D[20],cos_out_tmp_D[18:1]}; // 19b = {s,0,18}
175 // pipelining, make a1_minus_D, b0_D, b1_D, b2_D output with a0_inv_D at the same time
176 FF #(
177   .DATA_WIDTH ( COSIN_WIDTH )
178 )
179 FF_cos_OUT
180 (
```

```
  .Clk_CI  ( Clk_CI         ),
  .Rst_RBI ( Rst_RBI        ),
  .WrEn_SI ( a0_valid_S     ),
  .D_DI    ( cos_out_REG_D  ),
  .Q_DO    ( cos_out_D      )
);
assign a1_minus_D = {cos_out_D[18:0],1'b0};         // 20b = {s,1,18}

// calculate b1_D
assign b1_D = 20'b0_1_00_0000_0000_0000_0000 - {1'b0,cos_out_D};         // 20b: {s,1,18} - {s,1,
// calculate b0_D, b2_D
assign b0_D = b1_D;                // b0 = b1/2: move the decimal to the left; 20b:{s,0,19}
assign b2_D = b1_D;                // b2 = b0 {s,0,19}


// calculate a2_minus_D, a0_inv_D
DW_sincos #(
        .A_width(16),
        .WAVE_width(COSIN_WIDTH+2),  // 21b = {s,1,19} 1 sign bit, 1 integer bit, 18 fraction bit
        .arch(0),
        .err_range(1))
FLT_sin (
        .A(omega0_normalized_D),
        .SIN_COS(1'b0), // 0 for sine 1 for cosine
        .WAVE(sin_out_tmp_D) // {s,1,19}
);
wire signed [COSIN_WIDTH-1:0] sin_out_REG_D;
assign sin_out_REG_D = {sin_out_tmp_D[20],sin_out_tmp_D[18:1]}; // {s,0,18}
FF #(
  .DATA_WIDTH ( COSIN_WIDTH )
)
FF_sin_OUT
(
  .Clk_CI  ( Clk_CI         ),
  .Rst_RBI ( Rst_RBI        ),
  .WrEn_SI ( 1'b1           ),
  .D_DI    ( sin_out_REG_D  ),
  .Q_DO    ( sin_out_D      )
);

// this is for par_FLT_RQ = 0.1---------------------------------------
// calculate -a2
assign constant2_D = par_FLT_RQ_D >>> 1;                 // right shift 1 bit: (1/Q)/2, {s,0,31}
assign alpha_tmp_D = constant2_D * sin_out_D;            // 51bit: alpha = (sin(omega0)/2Q):  [s,
assign alpha_D = {alpha_tmp_D[50],alpha_tmp_D[45:15]};   // 32bit {s,0,31}  // truancate to 32bit
//assign a2_minus_D = alpha_D + (1<<alpha_frac_WIDTH);   // -a2 = alpha - 1 = alpha + (-1) is in
```

```
227 assign a2_minus_REG_D = {1'b1,alpha_D[COF_WIDTH-2:0]};       // same operation as alpha_D + (1<<3
228 // pipelining, make a2_minus_D output with a0_inv_D at the same time
229 FF #(
230   .DATA_WIDTH ( COF_WIDTH )
231 )
232 FF_a2_minus
233 (
234   .Clk_CI  ( Clk_CI         ),
235   .Rst_RBI ( Rst_RBI        ),
236   .WrEn_SI ( a0_valid_S     ),
237   .D_DI    ( a2_minus_REG_D  ),
238   .Q_DO    ( a2_minus_D      )
239 );
240
241 // calculate 1/a0 = a0_inv
242 assign a0_D = {1'b0,1'b1,alpha_D[COF_WIDTH-2:COF_WIDTH-19]};                    //20b
243 //-----------------------------------------------------------------------
244 wire signed [DIV_WIDTH-1:0] a0_inv_REG_D;
245 assign a0_inv_REG_D = {a0_inv_tmp_D[39],a0_inv_tmp_D[19:1]};      // truancate to 20bit = {sign b
246
247 always @(posedge Clk_CI or negedge Rst_RBI)
248 begin
249   if(~Rst_RBI)
250     a0_enable_S <= 1'b0;
251   else if (WrEn_SI == 1)
252     a0_enable_S <= 1'b1;
253   else
254     a0_enable_S <= 1'b0;
255 end
256 DW_div_seq #(
257   .a_width(DIV_WIDTH*2),
258   .b_width(DIV_WIDTH),
259   .tc_mode(1),
260   .num_cyc(15)      // 15 clks to finish the divider
261 )
262 FLT_div_a0_inv (
263   .clk(Clk_CI),
264   .rst_n(Rst_RBI),
265   .hold(1'b0), // always not hold
266   .start(a0_enable_S),
267   .a(40'b0100_0000_0000_0000_0000_0000_0000_0000_0000_0000),
268   .b(a0_D),
269   .complete(a0_valid_S),
270   .divide_by_0(),
271   .quotient(a0_inv_tmp_D),
272   .remainder()
```

```
);

FF #(
  .DATA_WIDTH ( DIV_WIDTH )
)
FF_DIV_OUT
(
  .Clk_CI  ( Clk_CI           ),
  .Rst_RBI ( Rst_RBI          ),
  .WrEn_SI ( a0_valid_S       ),
  .D_DI    ( a0_inv_REG_D     ),
  .Q_DO    ( a0_inv_D   )
);
// ------------------------------------------------------------------------------------
// ******************Coefficients calculation Done!*************************************
// ------------------------------------------------------------------------------------



// ------------------------------------------------------------------------------------
// ******************Sequencial logic calculation Begin!*********************************
// sclae input down and calculate xb0---------------
FF #(
  .DATA_WIDTH ( IN_WIDTH )
)
FF_FLT_IN
(
  .Clk_CI  ( Clk_CI                    ),
  .Rst_RBI ( Rst_RBI                   ),
  .WrEn_SI ( 1'b1                      ),
  .D_DI    ( sta_FLT_In_DI          ),
  .Q_DO    ( sta_FLT_IN_REG_D          )
);
assign sta_FLT_In_SD_tmp_D = sta_FLT_IN_REG_D * par_FLT_SD_D;                // {s,0,23}*{s,0,31}
assign sta_FLT_OldIn0_D = {sta_FLT_In_SD_tmp_D[IN_WIDTH+COF_WIDTH-1],sta_FLT_In_SD_tmp_D[IN_WIDTH
assign xb0_tmp_D = sta_FLT_OldIn0_D * b0_D;                                                  // {s
assign xb0_D = {xb0_tmp_D[51],xb0_tmp_D[49:19]};                                            // {s,0,3

// calculate xb1-----------------------------------------
FF #(
  .DATA_WIDTH ( COF_WIDTH )
)
FF_x1
(
  .Clk_CI  ( Clk_CI                    ),
  .Rst_RBI ( Rst_RBI                   ),
```

```
319   .WrEn_SI ( 1'b1                          ),
320   .D_DI    ( sta_FLT_OldIn0_D          ),
321   .Q_DO    ( sta_FLT_OldIn1_D          )
322 );
323 assign xb1_tmp_D = b1_D * sta_FLT_OldIn1_D;                                    // {s,1,18}*{s
324 assign xb1_D = {xb1_tmp_D[51],xb1_tmp_D[48:18]};                          // {s,0,31}
325
326 // calculate xb2-----------------------------------------
327 FF #(
328   .DATA_WIDTH ( COF_WIDTH )
329 )
330 FF_x2
331 (
332   .Clk_CI  ( Clk_CI                        ),
333   .Rst_RBI ( Rst_RBI                       ),
334   .WrEn_SI ( 1'b1                          ),
335   .D_DI    ( sta_FLT_OldIn1_D          ),
336   .Q_DO    ( sta_FLT_OldIn2_D          )
337 );
338 assign xb2_tmp_D = b2_D * sta_FLT_OldIn2_D;                                    // {s,0,19}*{s
339 assign xb2_D = {xb2_tmp_D[51],xb2_tmp_D[49:19]};        // {s,0,31}
340
341 // calculate sum of input loop---------------------------------------
342 assign sumx2x1_D = xb1_D + xb2_D;                                                // {s,0,
343 assign sumx0_D = sumx2x1_D + xb0_D;                                                // {s,
344
345 // calculate sum of output loop---------------------------------------
346 //assign sumy0_tmp_D = (sumx0_D>>>1) + (sumy2y1_D<<1);                    // {s,1,31}={s,1,31}+{s,
347 wire signed [COF_WIDTH:0] sumx0_RS_D;                                    // sumx0:{s,0,31} -> sum
348 wire signed [COF_WIDTH:0] sumy2y1_LS_D;                                  // sumy2y2_D:{s,1,30} ->
349 assign sumx0_RS_D = sumx0_D[COF_WIDTH-1]?{1'b1,sumx0_D}:{1'b0,sumx0_D};  // sign extension to mak
350 assign sumy2y1_LS_D = sumy2y1_D<<1;
351 assign sumy0_tmp_D = sumx0_RS_D + sumy2y1_LS_D;
352
353 assign sumy0_D = sumy0_tmp_D[COF_WIDTH:1];
354 assign ya0_tmp_D = a0_inv_D * sumy0_D;
355 assign sta_FLT_OldSample0_D = {ya0_tmp_D[51],ya0_tmp_D[48:18]};                // {s,0,31(60:30)
356 assign sta_FLT_Out_DO = sta_FLT_OldSample0_D[31:8];
357
358 // calculate ya1-----------------------------------------
359 FF #(
360   .DATA_WIDTH ( COF_WIDTH )
361 )
362 FF_y1
363 (
364   .Clk_CI  ( Clk_CI                                ),
```

```
  .Rst_RBI ( Rst_RBI                              ),
  .WrEn_SI ( 1'b1                                 ),
  .D_DI    ( sta_FLT_OldSample0_D          ),
  .Q_DO    ( sta_FLT_OldSample1_D          )
);
assign ya1_tmp_D = a1_minus_D * sta_FLT_OldSample1_D;                    // {s,2,49} = {s,1,1
assign ya1_D = {ya1_tmp_D[51],ya1_tmp_D[49:19]};                             // {s,1,30(60:30

// calculate ya2-----------------------------------------
FF #(
  .DATA_WIDTH ( COF_WIDTH )
)
FF_y2
(
  .Clk_CI  ( Clk_CI                               ),
  .Rst_RBI ( Rst_RBI                              ),
  .WrEn_SI ( 1'b1                                 ),
  .D_DI    ( sta_FLT_OldSample1_D          ),
  .Q_DO    ( sta_FLT_OldSample2_D          )
);
assign ya2_tmp_D = a2_minus_D * sta_FLT_OldSample2_D;                    // {s,1,62} = {s,0,3
assign ya2_D = ya2_tmp_D[63:32];

// calculate sum of output loop----------------------------
assign sumy2y1_D = ya2_D + ya1_D;


// ---------------------------------------------------------------------------------------

// ---------------------------------------------------------------------------------------
// ********************Sequencial logic calculation Done!*************************************
// ---------------------------------------------------------------------------------------

/* ---------------------------------------------------------------------------
// Done!
// --------------------------------------------------------------------------- */

endmodule
```

The following is the testbench Verilog code.

```
/*
 * ECE 5746 - Simple Filter Block Testbench
 * Project: SynTech
 * FLT_TB.v
 *
```

```
 * Author: Haochuan Song
 * Last updated: Oct 26, 2019
 * (c) 2019 hs994,fjf46@cornell.edu
 *
 */

`timescale 1ns / 1ps

module FLT_TB();

 localparam CLK_PERIOD = 1.0; // Clock period in ns
 localparam IN_DELAY   = 0.2; // Delay after clock edge that testbench signals take to reach DUT
 localparam OUT_DELAY  = 1.2; // Delay after clock edge that DUT outputs take to change
 localparam Inter_DELAY = 7;
 localparam ADDR_WIDTH = 5;   // Number of bits for address
 localparam MEM_WIDTH = 32;   // Number of bits for memory words
 localparam IN_WIDTH = 24;    // Number of bits for module inputs
 localparam OUT_WIDTH = 24;   // Number of bits for module outputs
 localparam COF_WIDTH = 32;   // Number of bits for FLT coefficients
 localparam CAL_WIDTH = 19;   // Number of bits for FLT intermediate results

 localparam CLK_PERIOD_HALF = CLK_PERIOD/2;

 integer fileIn;
 integer recIn;
 integer fileOut;
 integer recOut;
 integer fileInter;
 integer recInter;
 integer error;

 ////////////////////
 //DUT instantiation
 ////////////////////

 reg                   Clk_C;
 reg                   Rst_RB;
 reg                   WrEn_S;
 reg  [ADDR_WIDTH-1:0] Addr_D;
 reg  [MEM_WIDTH-1:0]  PAR_In_D;
 reg  [IN_WIDTH-1:0]   FLT_In_D;
 wire [OUT_WIDTH-1:0]  FLT_Out_D;
 reg  [OUT_WIDTH-1:0]  FLT_Out_DE;

 FLT #(
   .ADDR_WIDTH( ADDR_WIDTH ),
```

```
52    .MEM_WIDTH ( MEM_WIDTH  ),
53    .IN_WIDTH  ( IN_WIDTH   ),
54    .OUT_WIDTH ( OUT_WIDTH  ),
55    .COF_WIDTH ( COF_WIDTH  ),
56    .COSIN_WIDTH ( CAL_WIDTH  )
57  )
58  LPF (
59    .Clk_CI           ( Clk_C     ),
60    .Rst_RBI          ( Rst_RB    ),
61    .WrEn_SI          ( WrEn_S    ),
62    .Addr_DI          ( Addr_D    ),
63    .PAR_In_DI        ( PAR_In_D  ),
64    .sta_FLT_In_DI    ( FLT_In_D  ),
65    .sta_FLT_Out_DO   ( FLT_Out_D )
66  );
67
68  //Clock generation
69  initial begin
70    Clk_C = 1'b1;
71    forever
72      #CLK_PERIOD_HALF Clk_C=~Clk_C;
73  end
74
75  //Stimuli application
76  initial begin
77    //Wait for the input delay
78    #(IN_DELAY) begin end
79    //Prepare stimuli file
80    fileIn = $fopen("../tb/RND_FLT_in.txt","r");
81    //Read file on a per cycle basis
82    while(!$feof(fileIn)) begin
83      recIn = $fscanf(fileIn, "%d %d %d %d %d\n", Rst_RB, WrEn_S, Addr_D, PAR_In_D, FLT_In_D);
84          #CLK_PERIOD begin end
85    end
86    //Close file
87    $fclose(fileIn);
88    //$finish;
89  end
90
91   //Output comparison
92  initial begin
93    //Initialize the error counter
94    error = 0;
95    //Wait for the output delay
96    #(OUT_DELAY) begin end
97    //Prepare expected output file
```

```
    fileOut = $fopen("../tb/RND_FLT_out.txt","r");
    //Read file on a per cycle basis
    while(!$feof(fileOut)) begin
      recOut = $fscanf(fileOut, "%d\n", FLT_Out_DE);
      //For each signal, we compare the expected output with the one obtained
      //EXA_Out_DO
      if(&FLT_Out_DE !== 1'bX) begin
        if(FLT_Out_D !== FLT_Out_DE) begin
          $display("[", $time, "] FLT_Out_DO :: Value %d Expected %d",FLT_Out_D,FLT_Out_DE);
          error = error + 1;
        end
      end
      #CLK_PERIOD begin end
    end
    //Close file
    $fclose(fileOut);
    //Finish simulation
    if(error == 0) $display("<<< :D All outputs match the expected results! :D >>>");

    $finish;
  end

endmodule
```

```
/*
 * ECE 5746 - Simple Filter Block Testbench
 * Project: SynTech
 * FLT_TB.v
 *
 * Author: Haochuan Song
 * Last updated: Oct 26, 2019
 * (c) 2019 hs994,fjf46@cornell.edu
 *
 */

`timescale 1ns / 1ps

module FLT_POST_TB();

 localparam CLK_PERIOD = 120; // Clock period in ns
 localparam IN_DELAY   = 0.2; // Delay after clock edge that testbench signals take to reach DUT
 localparam OUT_DELAY  = 150.2; //Propagation takes around 30ns; Delay after clock edge that DUT

 localparam ADDR_WIDTH = 5;   // Number of bits for address
 localparam MEM_WIDTH = 32;   // Number of bits for memory words
 localparam IN_WIDTH = 24;    // Number of bits for module inputs
```

```
localparam OUT_WIDTH = 24;   // Number of bits for module outputs
localparam COF_WIDTH = 32;   // Number of bits for FLT coefficients
localparam CAL_WIDTH = 19;   // Number of bits for FLT intermediate results

localparam CLK_PERIOD_HALF = CLK_PERIOD/2;

integer fileIn;
integer recIn;
integer fileOut;
integer recOut;
integer fileInter;
integer recInter;
integer error;

////////////////////
//DUT instantiation
////////////////////

reg                     Clk_C;
reg                     Rst_RB;
reg                     WrEn_S;
reg  [ADDR_WIDTH-1:0]   Addr_D;
reg  [MEM_WIDTH-1:0]    PAR_In_D;
reg  [IN_WIDTH-1:0]     FLT_In_D;
wire [OUT_WIDTH-1:0]    FLT_Out_D;
reg  [OUT_WIDTH-1:0]    FLT_Out_DE;

FLT LPF (
  .Clk_CI           ( Clk_C     ),
  .Rst_RBI          ( Rst_RB    ),
  .WrEn_SI          ( WrEn_S    ),
  .Addr_DI          ( Addr_D    ),
  .PAR_In_DI        ( PAR_In_D  ),
  .sta_FLT_In_DI    ( FLT_In_D  ),
  .sta_FLT_Out_DO   ( FLT_Out_D )
);

//Clock generation
initial begin
  Clk_C = 1'b1;
  forever
    #CLK_PERIOD_HALF Clk_C=~Clk_C;
end

//Stimuli application
initial begin
```

```
    //Wait for the input delay
    #(IN_DELAY) begin end
    //Prepare stimuli file
    fileIn = $fopen("../tb/RND_FLT_in.txt","r");
    //Read file on a per cycle basis
    while(!$feof(fileIn)) begin
      recIn = $fscanf(fileIn, "%d %d %d %d %d\n", Rst_RB, WrEn_S, Addr_D, PAR_In_D, FLT_In_D);
          #CLK_PERIOD begin end
    end
    //Close file
    $fclose(fileIn);
    //$finish;
  end

   //Output comparison
  initial begin
    //Initialize the error counter
    error = 0;
    //Wait for the output delay
    #(OUT_DELAY) begin end
    //Prepare expected output file
    fileOut = $fopen("../tb/RND_FLT_out.txt","r");
    //Read file on a per cycle basis
    while(!$feof(fileOut)) begin
      recOut = $fscanf(fileOut, "%d\n", FLT_Out_DE);
      //For each signal, we compare the expected output with the one obtained
      //EXA_Out_DO
      if(&FLT_Out_DE !== 1'bX) begin
        if(FLT_Out_D !== FLT_Out_DE) begin
          $display("[", $time, "] FLT_Out_DO :: Value %d Expected %d",FLT_Out_D,FLT_Out_DE);
          error = error + 1;
        end
      end
      #CLK_PERIOD begin end
    end
    //Close file
    $fclose(fileOut);
    //Finish simulation
    if(error == 0) $display("<<< :D All outputs match the expected results! :D >>>");

    $finish;
  end

endmodule
```