Preliminary Matlab Model of Low-pass Filter

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

1 IIR Filter Description

At first, this module will receive the sample signal from the oscillator. In this module, those 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 envelope module. The inputs are the sample signal *sta_FLT_in_DI* from the oscillator as well as the chip signals like 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 How to Create an Low-pass Filter

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

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

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

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

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

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}, \tag{3}$$

and

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2] - a_1 \cdot y[n-1] - a_2 \cdot y[n-2]. \tag{4}$$

For this simplified equation, it is easy to generate a Block Diagram. The following is 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:

$$\omega_0 = 2 \cdot \pi \cdot \frac{f_0}{F_S}. \tag{5}$$

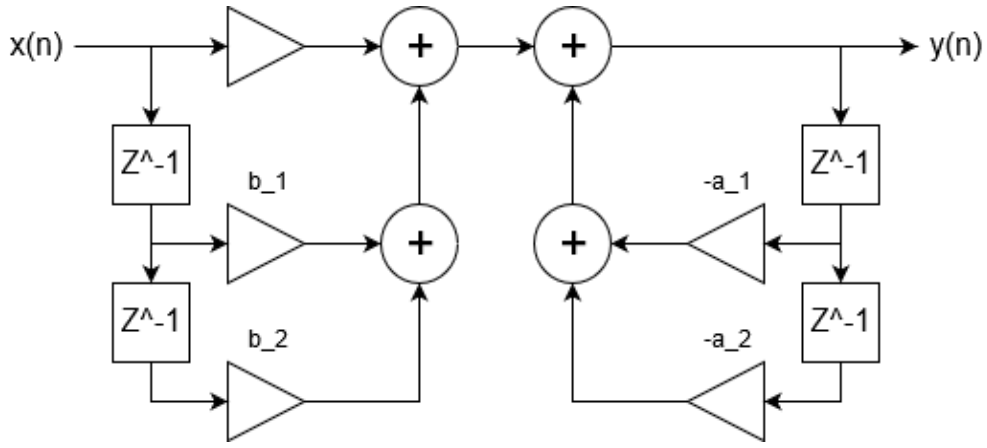

Figure 1: IIR Direct Form Structures.

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. (6):

$$\alpha = \frac{sin(\omega_0)}{2 \cdot Q}. \tag{6}$$

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:

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1}. \tag{7}$$

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.

$$b_0 = \frac{1 - cos(\omega_0)}{2}, \tag{8}$$

$$b_1 = 1 - cos(\omega_0), \tag{9}$$

$$b_2 = \frac{1 - cos(\omega_0)}{2}, \tag{10}$$

$$a_0 = 1 + \alpha, \tag{11}$$

$$a_1 = -2 \cdot cos(\omega_0), \tag{12}$$

$$a_2 = 1 - \alpha. \tag{13}$$

These calculated values can now be substituted into the direct form Eq. (2) and used to implement an LPF.

3 Specification Tables

In Table 1 we show the input parameters to the filter we also show the parameters gotten from memory in Table 2 and our output parameter in Table 3. Table 4 shows the different states of our block. The amount of bits for every parameter and every parameters quantization policy is explained in Table 6.

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 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_DI	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)

Table 3: Output Parameters

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

Table 4: Block States

State Variables	**Description**
$sta\_FLT\_OldIn\_D(1,2,3)$	These are 3 filter inputs. These will be used to generate the filter output.
$sta\_FLT\_OldSample\_D(1,2,3)$	These are 3 filter outputs. $sta\_FLT\_OldSample\_D(1,2)$ will be used to generate the filter output. $sta\_FLT\_OldSample\_D(3)$ will be the filter final output.

4 Matlab Implementation

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 function **FLT_init** 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. (8)−(13) to perform Eq. (2). Finally, the function outputs *sta_FLT_OldSample_D* which is the latest output in Eq. (2).

5 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 where tested and the Matlab implementation held up for all of them. In this report only two cases which shows 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. 2 and for $Q = 10; f_0 = 500$ is shown in Fig. 3.

6 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. 4. For Fig. 6 to 10, 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 an alternative is necessary.

6.1 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. 5. 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 5. 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.

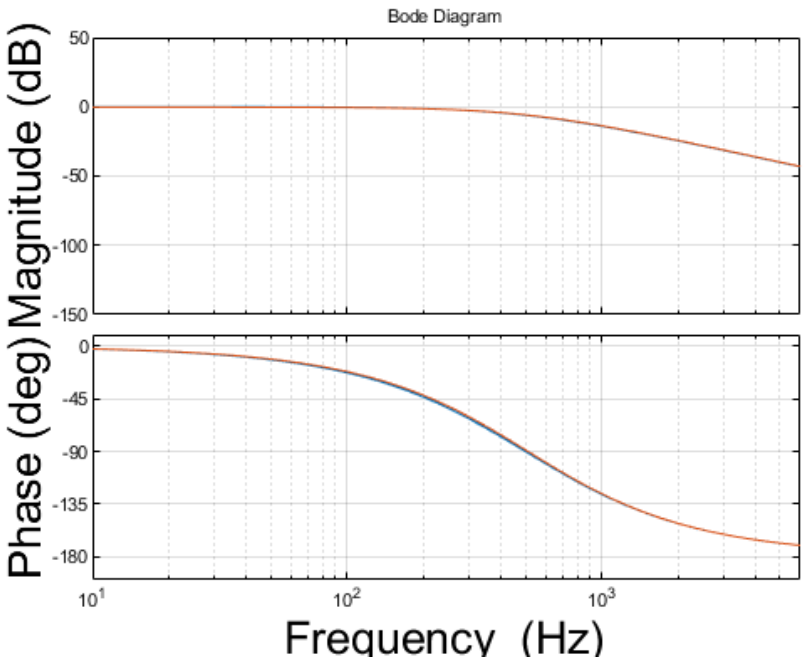

Figure 2: Float point to Fixed point comparison for $Q = 0.5$ and $f_0 = 50k$

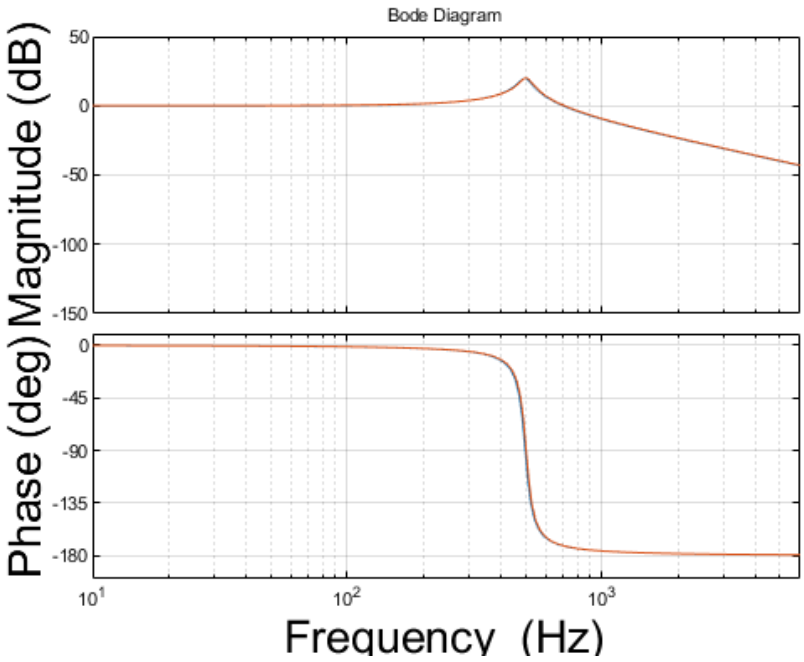

Figure 3: Float point to Fixed point comparison for $Q = 10$ and $f_0 = 500$

6.2 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. 4. 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 5: 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

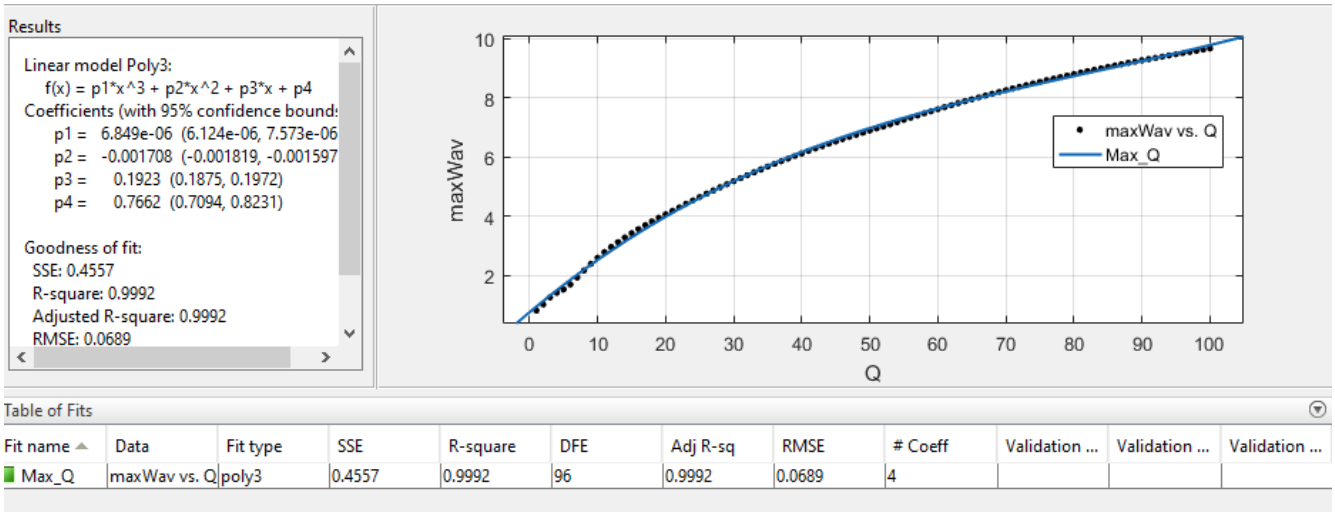

Figure 4: Predicted max output curve for different values of Q

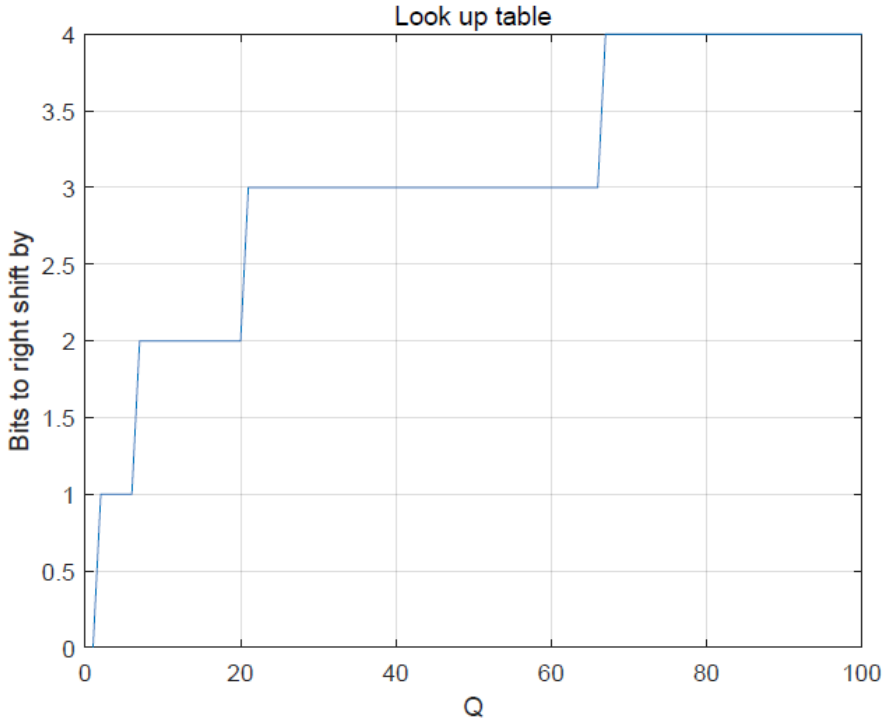

Figure 5: Predicted max output curve for different values of Q

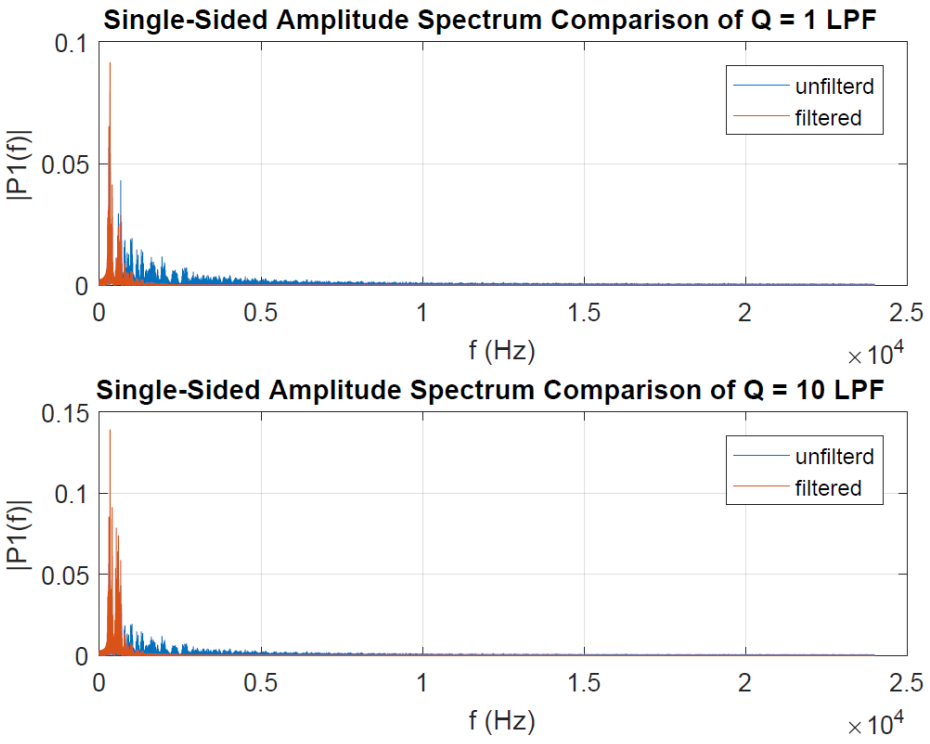

Figure 6: Output for $Q = 1$ and $Q = 10$

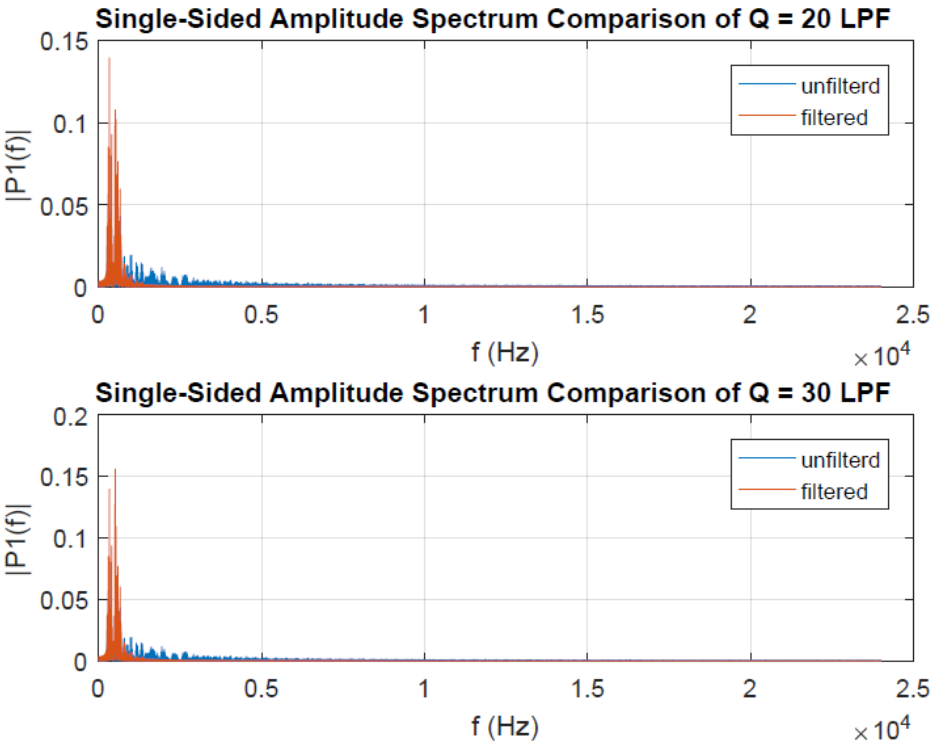

Figure 7: Output for $Q = 20$ and $Q = 30$

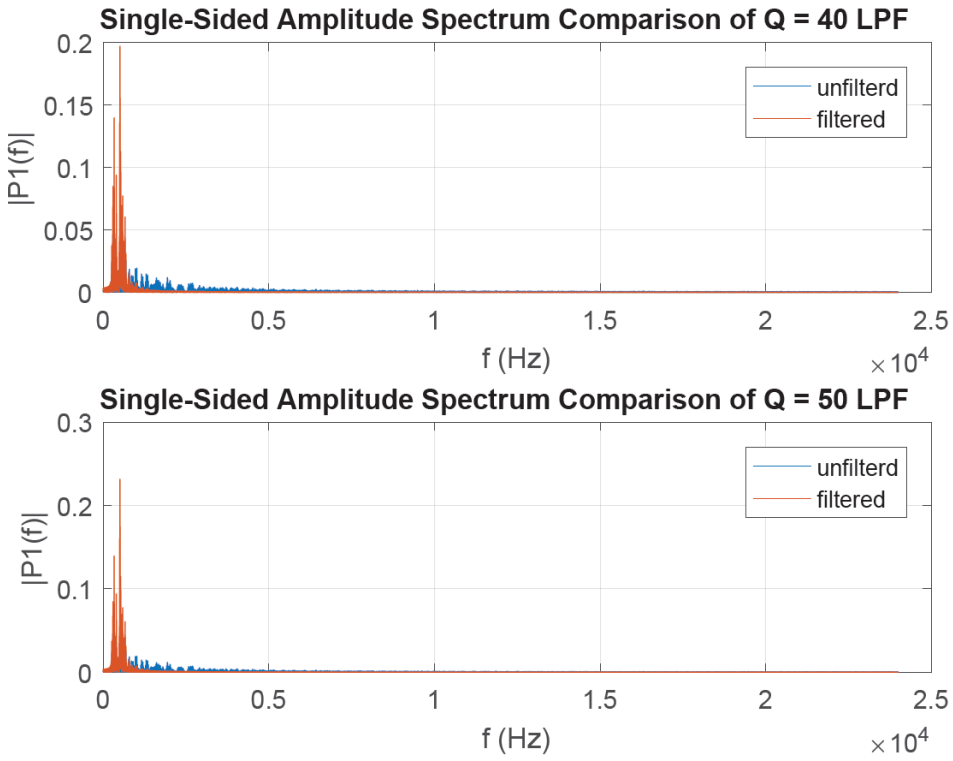

Figure 8: Output for $Q = 40$ and $Q = 50$

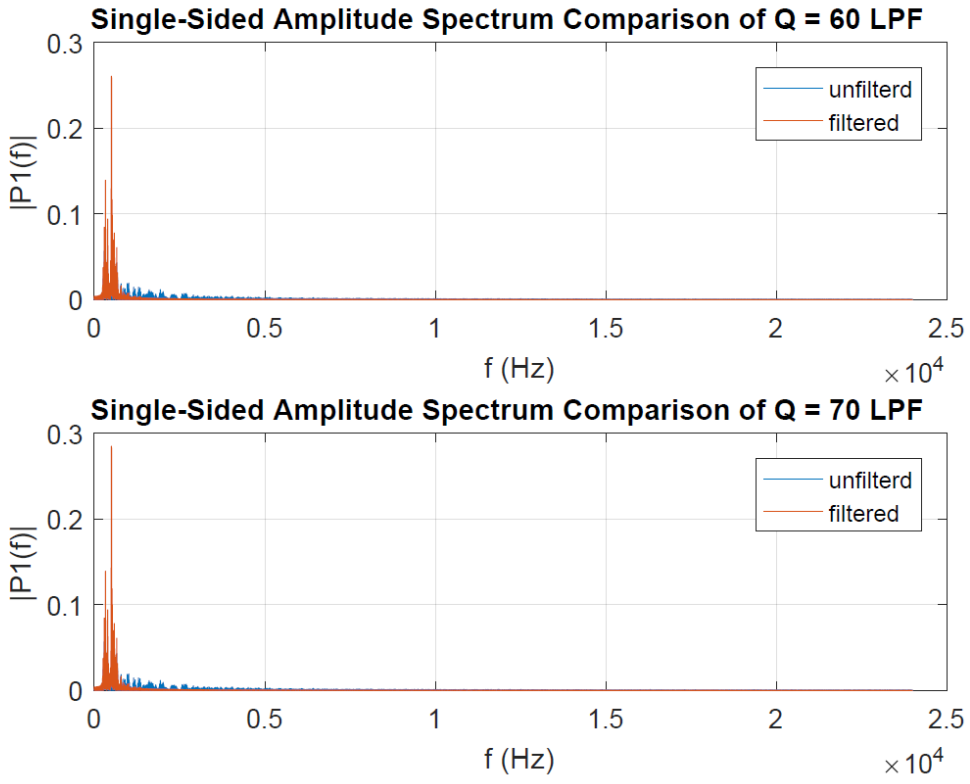

Figure 9: Output for $Q = 60$ and $Q = 70$

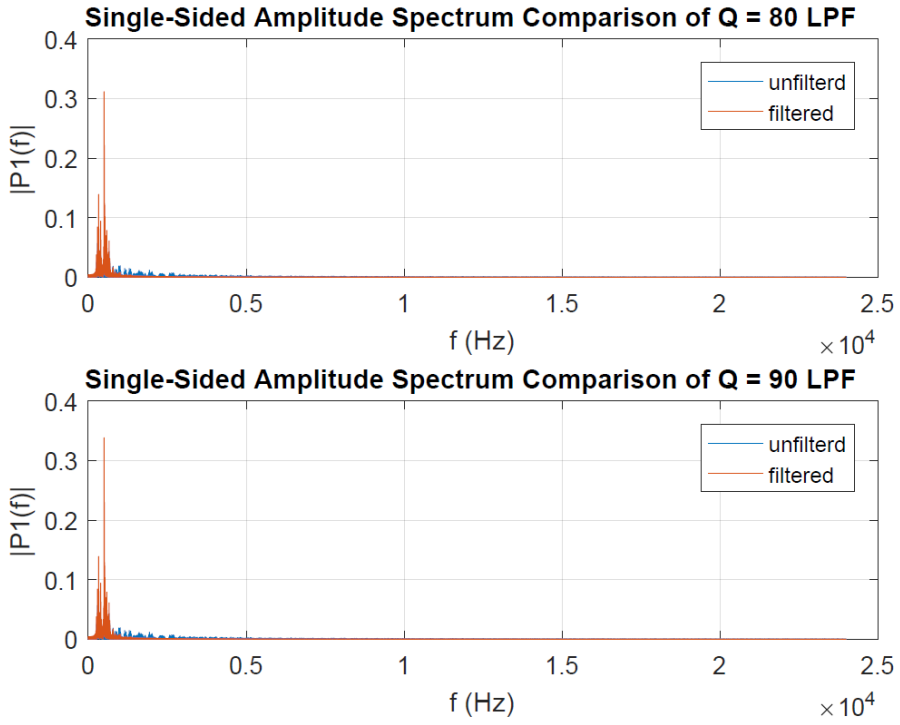

Figure 10: Output for $Q = 80$ and $Q = 90$

7 Experiment 2

We experimented by changing the Q value of of our circuit and re calculating 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. 11 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.

8 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 quintization policies we tested different quintization 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 and 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 less we got significatn differences between our fixed point and floating point results, this can be seen in Fig. 12 and Fig. 13. 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. 14 and Fig. 15

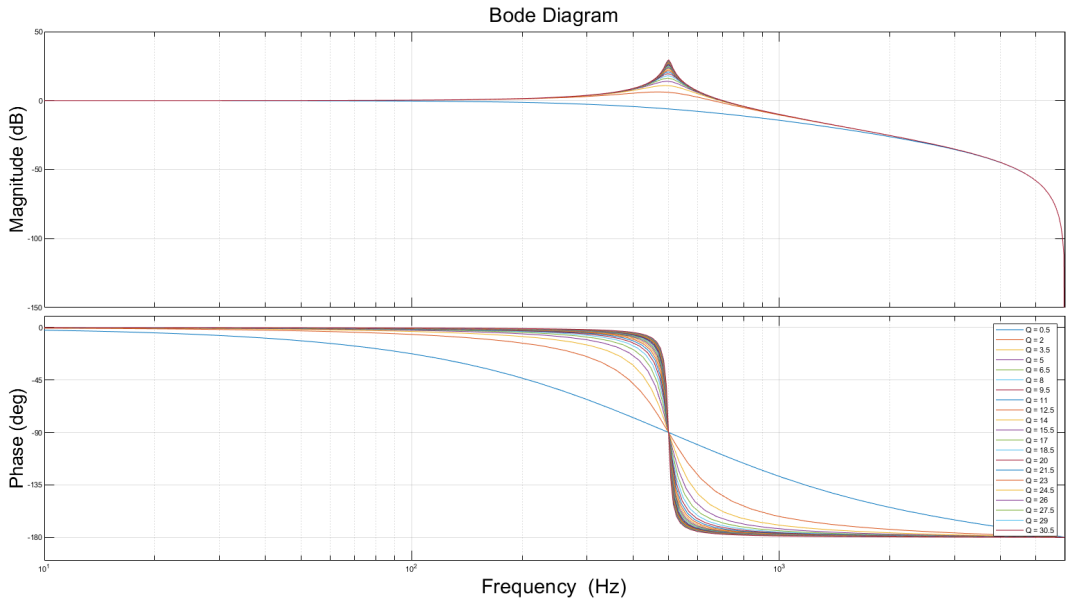

Figure 11: Bode plots for different Q values

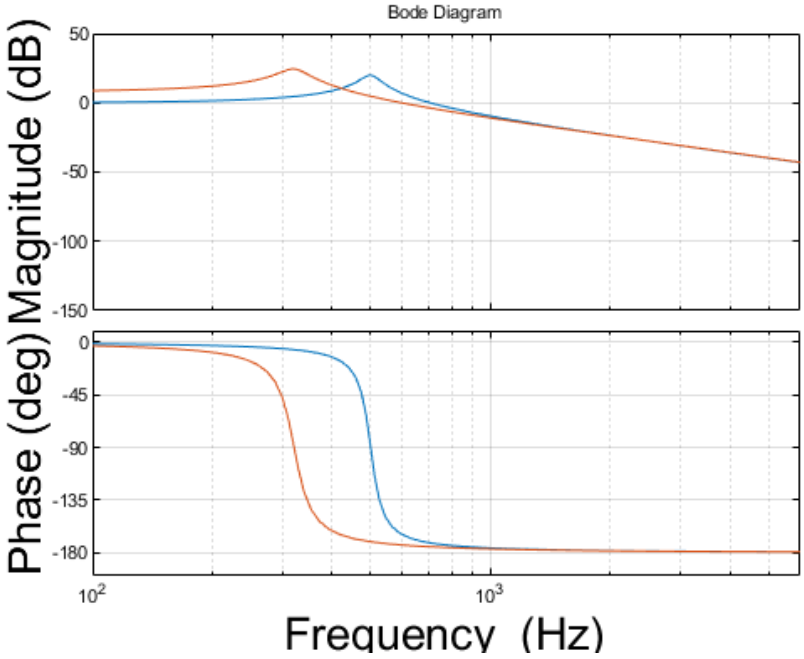

Figure 12: Bode plots comparing floating point to fixed point where a_o has only 16 decimal bits

9 Quantization

In this deliverable, 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 parameters, are quantized with 32 bits to guarantee the numerical accuracy. Here, we assume the minimum

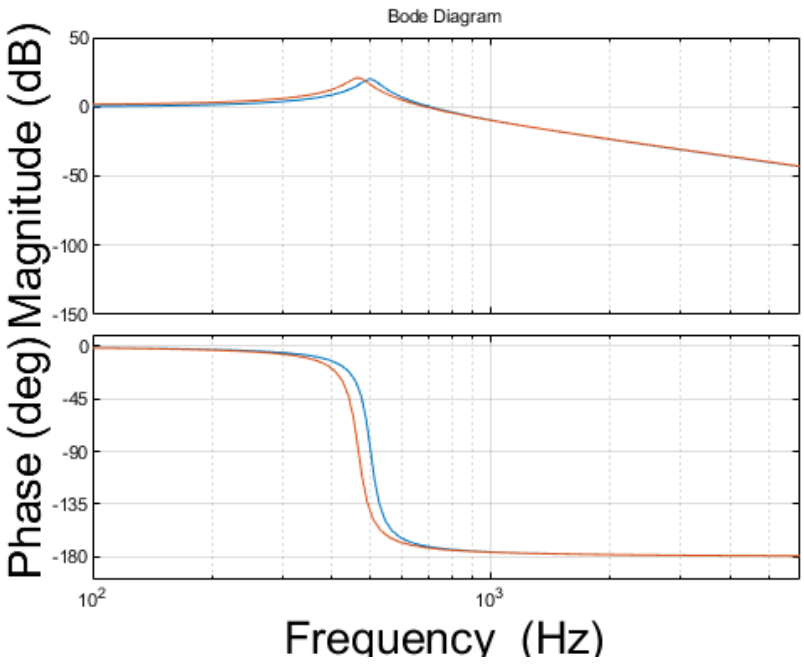

Figure 13: Bode plots comparing floating point to fixed point where a_o has only 17 decimal bits

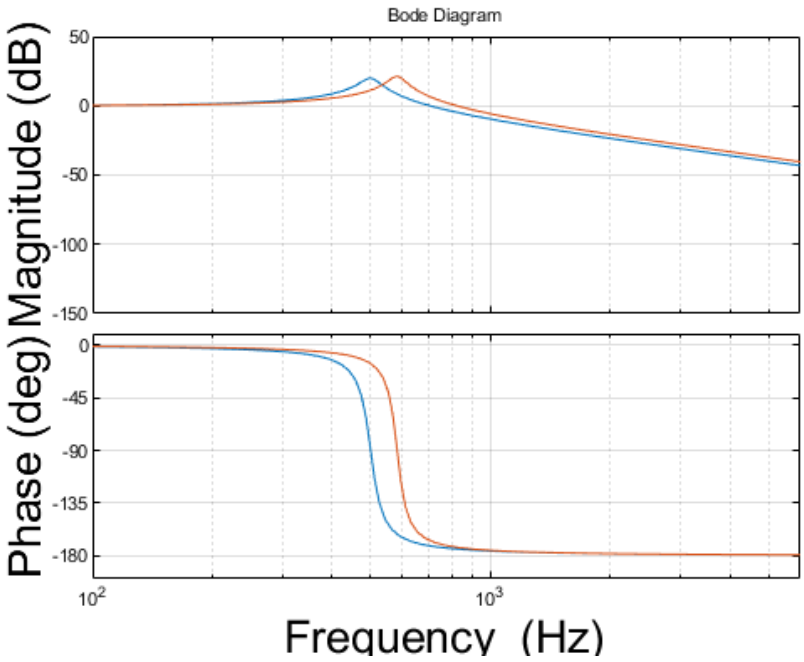

Figure 14: Bode plots comparing floating point to fixed point where cosine outputs have only 16 decimal bits

value of Q is 0.1 because small Qs will not make a reasonable LPF and that's why α lies in $[-5, 5]$. The quantization policies are listed in Table 6.

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

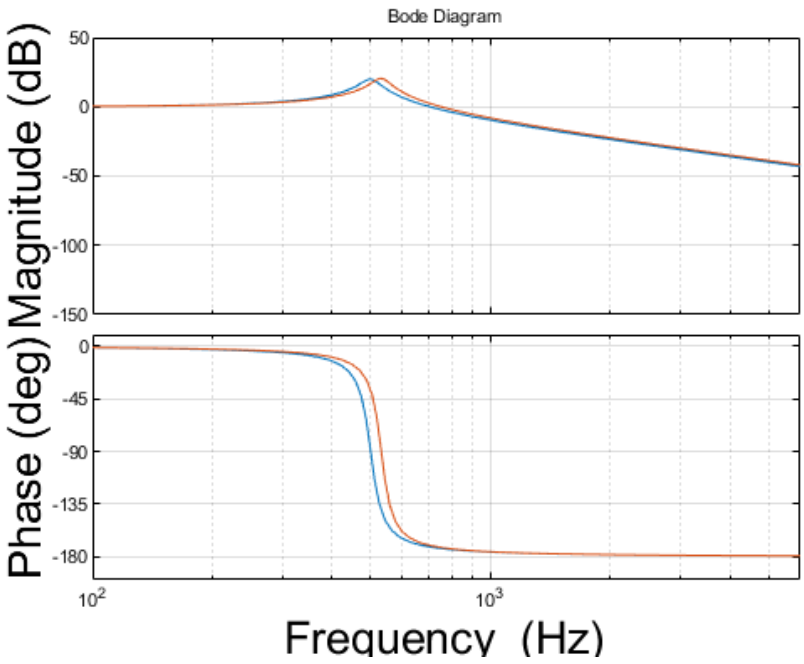

Figure 15: Bode plots comparing floating point to fixed point where cosine outputs have only 17 decimal bits

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. 14:

$$n = \lceil \log_2(I\_inter\_var) \rceil. \tag{14}$$

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. 16 are nearly equivalent to the floating-point results in Fig. 6.

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.

10 Block Diagram

The following Fig 17 and Fig 18 is our hardware block diagram. The diagram shows that we get *sta_FLT_in_DI*, *CLK_CI*, *Rst_RBI*, *WrEn_SI*, *Addr_DI* and *PAR_In_DI* as inputs. 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). Lastly our only output is *sta_FLT_out_DO*. Inputs are clearly shown in red circles and outputs are shown in green circles. All the values gotten from memory is shown in the memory block.

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

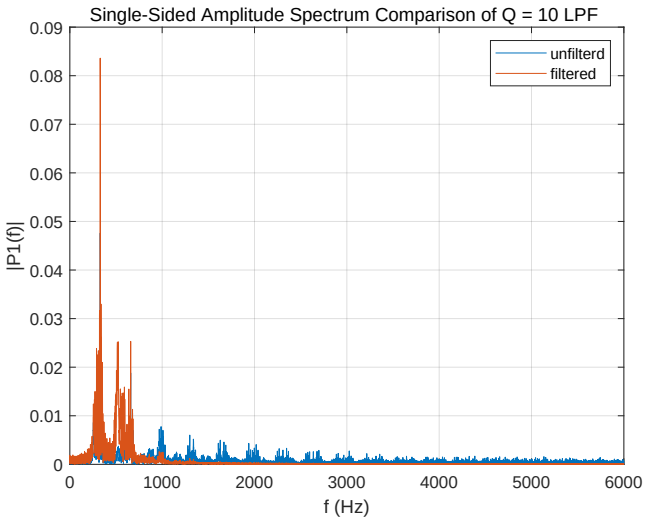

Figure 16: Amplitude spectrum of fixed points waveform.

the block diagram but for the implementation we will approximate π and use the approximated constant instead of π. In Fig. 17 we show the block diagram of the actual filter with its parameters in blue circles. In Fig. 18 we show how the filter parameters are calculated and what values are gotten from memory. There is also an implemented sequential divider IP core.

11 Verilog Implementation

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 $ScalingDownFactor$ where 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 pre-calculated and stored in the memory; $par\_FLT\_f0\_D$ 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 ω_0. 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.

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. 19 and Fig. 20 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

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 7.

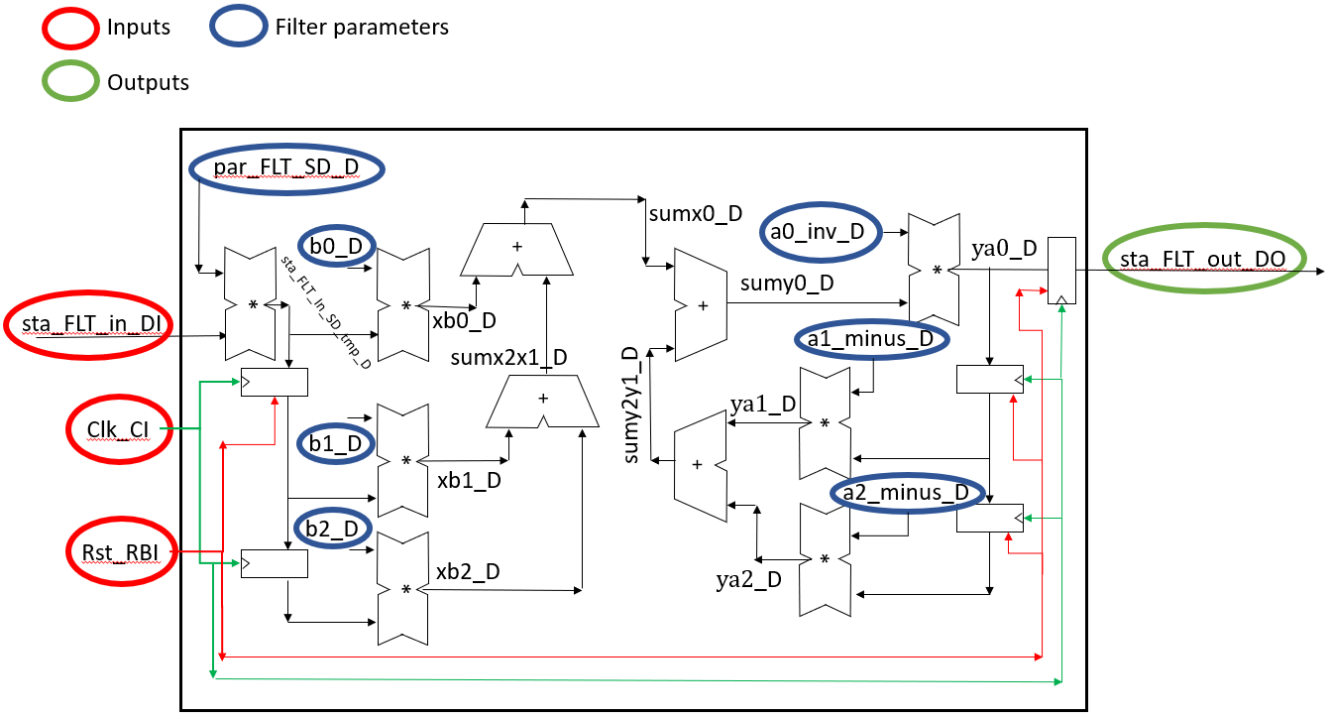

Figure 17: Detailed Block Diagram of main loop

11.1 Design Tests

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

12 Synopsis Results

For the area report of our design we got the following results that can be seen in Table 8.

For the Synopsis simulation we used a clock periods ranging from $100ns$ to $170ns$. In the below tables the results when the clock constraint was set to $120ns$ is shown because it had no violations. For the maximum timing report of our design we got the following results that can be seen in Table 9.

For the minimum timing report of our design we got the following results that can be seen in Table 10.

For the are report of the design the following results shown in Table 11 where gotten.

13 Area Time Diagram

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

14 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 shrinking there 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.

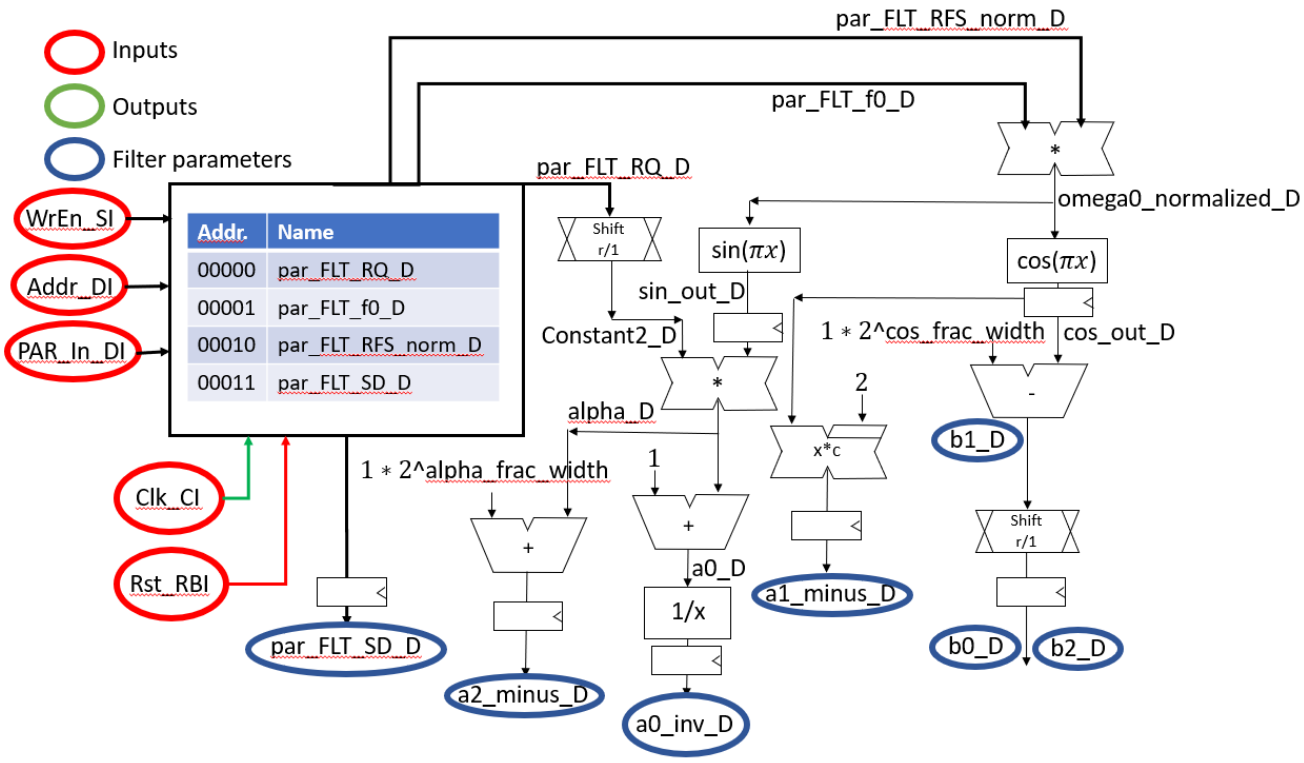

Figure 18: Detailed Block Diagram of memory and parameter calculation

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

Figure 19: Output of Modelsim showing that the Verilog results matches that of the testbench

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 that is talked about 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 6.

The divider was extremely large 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 the divider takes. The second idea to optimize the size the divider takes 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

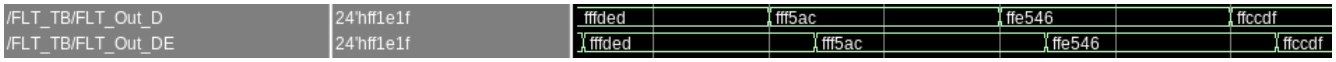

Figure 20: Showing that the output signal of the testbench and that of the Verilog matches up

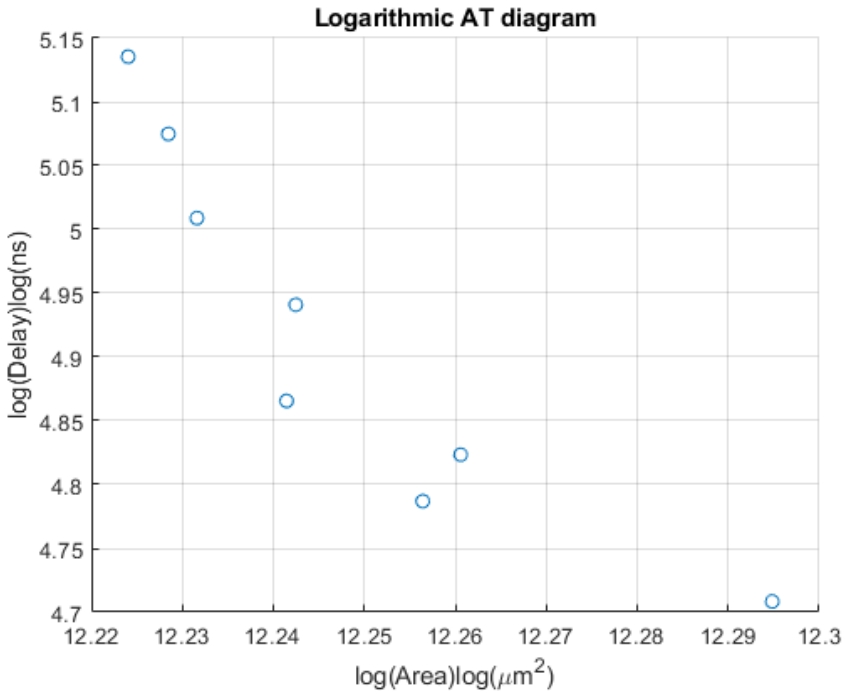

Figure 21: logarithmic AT diagram

two major size optimizations the size of the block is acceptable.

To optimize the critical path of the block and reduce unnecessary operations a lot of flip flops where added. There where 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 where 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.

References

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

Table 6: Table of quantization policies

	{**I,F,sign**}	**Quantization Type**	**Dynamic Range**
α_D	{0,31,'s'}	SatTrc	$[-1, 1)$
$Constant2\_D$	{3,28,'s'}	SatTrc	$[-8, 8)$
sin_out_D	{0,18,'s'}	SatTrc	$[-1, 1)$
cos_out_D	{0,18,'s'}	SatTrc	$[-1, 1)$
$omega0\_normalized\_D$	{0,15,'s'}	SatTrc	$[-1, 1)$
b_0_D	{0,19,'s'}	SatTrc	$[-1, 1)$
b_1_D	{1,18,'s'}	SatTrc	$[-2, 2)$
b_2_D	{0,19,'s'}	SatTrc	$[-1, 1)$
a_0_D	{1,18,'s'}	SatTrc	$[-2, 2)$
$a_0\_inv\_D$	{0,19,'s'}	SatTrc	$[-1, 1)$
$a_1\_minus\_D$	{2,29,'s'}	SatTrc	$[-4, 4)$
$a_2\_minus\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sta\_FLT\_OldSample\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sta\_FLT\_OldIn\_D$	{0,31,'s'}	SatTrc	$[-1, 1]$
$par\_FLT\_RQ\_D$	{3,28,'s'}	SatTrc	$[-8, 8)$
$par\_FLT\_f_0\_D$	{31,0,'s'}	SatTrc	$[-2147483648, 2147483648)$
$par\_FLT\_RFS\_norm\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$par\_FLT\_SD\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sta\_FLT\_in\_DI$	{0,23,'s'}	SatTrc	$[-1, 1)$
$sta\_FLT\_out\_DO$	{0,23,'s'}	SatTrc	$[-1, 1)$
$sta\_FLT\_in\_SD\_tmp\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$xb0\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$xb1\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$xb2\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sumx2x1\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sumx0\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$sumy0\_D$	{1,30,'s'}	SatTrc	$[-2, 2)$
$sumy2y1\_D$	{1,30,'s'}	SatTrc	$[-2, 2)$
$ya0\_D$	{0,31,'s'}	SatTrc	$[-1, 1)$
$ya1\_D$	{1,30,'s'}	SatTrc	$[-2, 2)$
$ya2\_D$	{1,30,'s'}	SatTrc	$[-2, 2)$

Table 7: 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

Table 8: Table of Area report

Combinational area	444509.801505
Buf/Inv area	30911.386371
Noncombinational area	8995.737743
Net Interconnect area	83382.257924
Total cell area	453505.539248
Total area	536887.797172

Table 9: Table of maximum timing report for 120ns

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

Table 10: Table of maximum timing report for 120ns

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

Table 11: Table of area report for 120ns

Combinational	210328.4748
Noncombinational	14848.8194