

Digital Implementation of FIR Filters

Abanoub Emad Hanna

Faculty of Engineering - Ain Shams University



Introduction

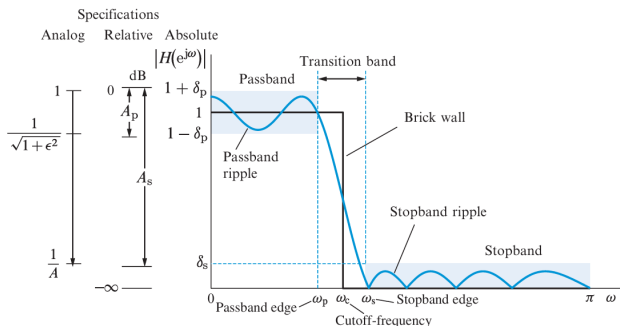
- A filter is an LTI system which is designed to pass a set of desired frequency components from a mixture of desired and undesired components.
- Finite impulse response (FIR) filters are:
 - 1 Always stable.
 - 2 No feedback \rightarrow no poles.
- For FIR systems the filter coefficients b_k are equal to the impulse response $h[k]$.

$$y[n] = \sum_{k=0}^M b_k x[n-k] = \sum_{k=0}^M h[k] x[n-k]$$

- I will focus on causal, linear phase Type I FIR filter.

Introduction

- Practical filters differ from ideal filters in several respects:
 - The passband responses are not perfectly flat.
 - The stopband responses cannot completely reject bands of frequencies.
 - The transition between passband and stopband regions takes place over a finite transition band.



Specifications

- Let's design a lowpass FIR filter with the following specifications (Example 10.2 in [1]):

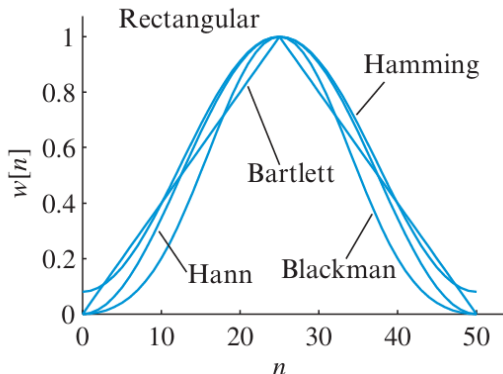
Parameter	Value
Passband Frequency (ω_p)	0.25π
Stopband Frequency (ω_s)	0.35π
Passband Ripple (A_p)	0.1 dB
Stopband Attenuation (A_s)	50 dB

- I will design using the window method and the equiripple optimum Chebyshev method and generate the graphs with the Python script.

Window Method

- The impulse response of the ideal filter is truncated by multiplying it by a window function and is given by

$$h[n] = h_d[n]w[n] = \frac{\sin(\omega_c(n - \alpha))}{\pi(n - \alpha)} w[n]$$



Standard Windows

- Determine δ_p and δ_s from A_p and A_s .

$$\delta_p = \frac{10^{\frac{A_p}{20}} - 1}{10^{\frac{A_p}{20}} + 1} = 5.756 \times 10^{-3} \quad \delta_s = \frac{1 + \delta_p}{10^{\frac{A_s}{20}}} = 3.18 \times 10^{-3}$$

- Determine the cutoff frequency ω_c .

$$\omega_c = \frac{\omega_p + \omega_s}{2} = 0.3\pi$$

- Determine the design parameters

$$A = -20 \log_{10} \min(\delta_p, \delta_s) = 49.95 \quad \Delta\omega = \omega_s - \omega_p = 0.1\pi$$

Standard Windows

- From the table, choose a window with the smallest stopband attenuation greater than $A \rightarrow$ **Hamming Window**.
- Determine L

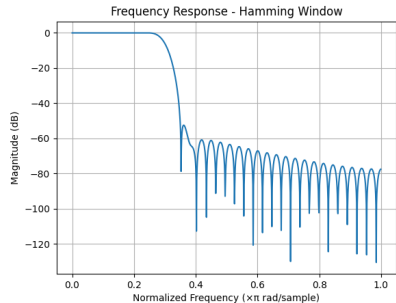
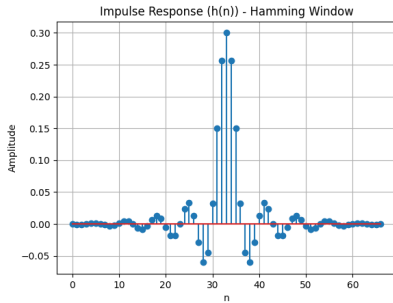
$$0.1\pi = \frac{6.6\pi}{L} \rightarrow L = 66$$

- Change Type II to I by increasing the length by 1, then determine M and α .

$$L = 66 + 1 = 67 \quad M = L - 1 = 66 \quad \alpha = M/2 = 33$$

Window name	Side lobe level (dB)	Approx. $\Delta\omega$	Exact $\Delta\omega$	$\delta_p \approx \delta_s$	A_p (dB)	A_s (dB)
Rectangular	-13	$4\pi/L$	$1.8\pi/L$	0.09	0.75	21
Bartlett	-25	$8\pi/L$	$6.1\pi/L$	0.05	0.45	26
Hann	-31	$8\pi/L$	$6.2\pi/L$	0.0063	0.055	44
Hamming	-41	$8\pi/L$	$6.6\pi/L$	0.0022	0.019	53
Blackman	-57	$12\pi/L$	$11\pi/L$	0.0002	0.0017	74

Standard Windows



Kaiser Window

- What makes the Kaiser window special is that it has a parameter β that can be adjusted to control the trade-off between the mainlobe width and the sidelobe level.
- From Kaiser's empirical formulas, the parameter β is given by

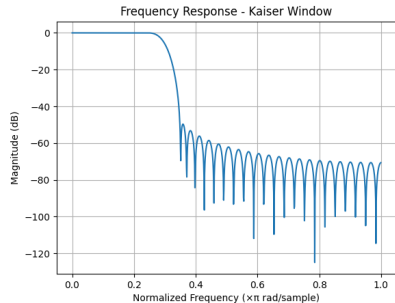
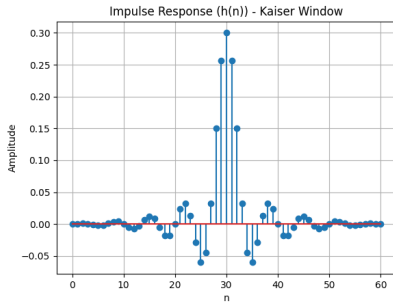
$$\beta = \begin{cases} 0.1102(A - 8.7) & \text{if } A > 50 \\ 0.5842(A - 21)^{0.4} + 0.07886(A - 21) & \text{if } 21 \leq A \leq 50 \\ 0 & \text{if } A < 21 \end{cases}$$

$$= 4.528$$

$$M = \frac{A - 8}{2.285\Delta\omega} = 59$$

- Change to Type I $\longrightarrow M = 60, L = 61, \alpha = 30$.

Kaiser Window



Chebyshev Approximation

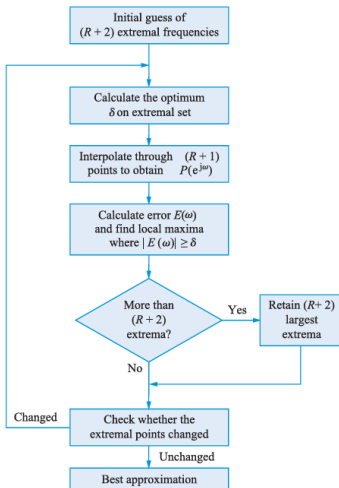
- This method minimizes the maximum error between the ideal and actual frequency responses, so we define:
 - $A_d(\omega)$: The desired frequency response.
 - $A(\omega)$: The actual frequency response.
 - $W(\omega)$: The weighting function.
 - $E(\omega)$: The error function.
- $\min(E(\omega))$ is desired

$$E(\omega) = W(\omega)[A_d(\omega) - A(\omega)]$$

- The Remez exchange algorithm is used to solve these kind of problems, and the Parks-McClellan algorithm is a specific application of the Remez exchange algorithm, tailored for designing FIR filters with linear phase.

Parks-McClellan Algorithm

- Read more about the algorithm in section 10.6.3 in [1].



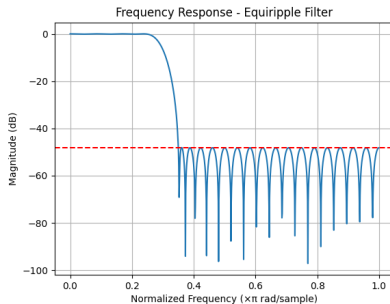
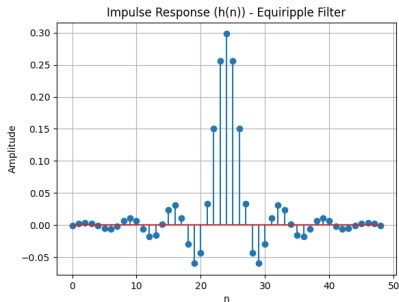
Parks-McClellan Algorithm

- We have to obtain M to use the algorithm. Kaiser introduced a formula to estimate M .

$$M = \frac{-20 \log_{10}(\delta_p \delta_s)^{0.5} - 13}{2.324 \Delta \omega} = 48$$

- We then check if it achieves the desired specifications. If not, we increase M and repeat the process.
- Using Remez function in Python, the specifications are met, so $L = 49$ and $\alpha = 24$.

Equiripple Filter



- Notice how the ripples in the passband and stopband are equal.

Comparison

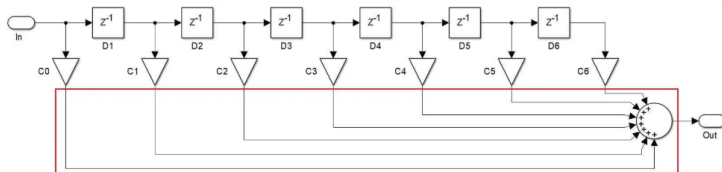
- The window method is easier to implement and understand, but the equiripple method provides smaller number of taps for the same specifications.

Method	Length
Window (Hamming)	67
Window (Kaiser)	61
Equiripple	49

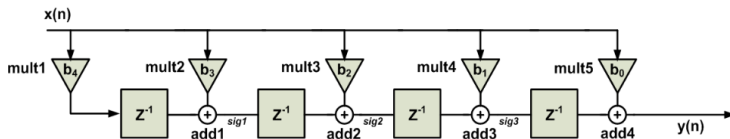
- The Python code calculates the filter coefficients and edit the System Verilog Code directly as well as creates the .coe file.

Architectures

- **Direct Form** architecture is the simplest and most intuitive, but the highlighted path limits the maximum clock frequency.



- **Transposed Form** architecture achieves higher clock frequencies.



Folding

- Due to the symmetry of linear phase filter coefficients ($b_k = b_{M-k}$), we can use half the number of multipliers.
- Also, we only need to store almost half the number of coefficients.
- For example:

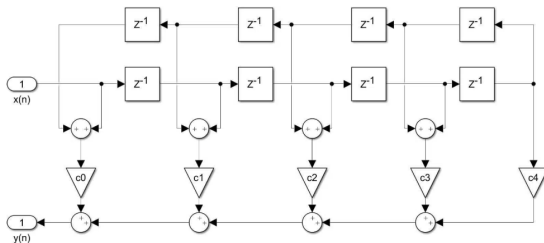
$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + b_3x[n-3] + b_4x[n-4]$$

\Downarrow

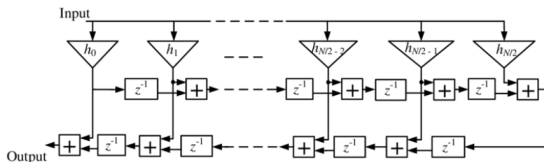
$$y[n] = b_0(x[n] + x[n-4]) + b_1(x[n-1] + x[n-3]) + b_2x[n-2]$$

Folding

- Direct Form



- Transposed Form



Code

- Fixed Coefficients:
 - Coefficients are fixed, so they can be optimized reducing area and pins.
 - Suitable for some audio applications.
- Reloadable coefficients:
 - Coefficients can be changed during runtime, so they are stored in a memory block, where we can write to it.
 - Suitable for audio equalizers.
- Use the compiler directives to switch between the two modes.

Code

- Full code on GitHub ([link](#))

```
49
50  always_ff @(posedge clk or negedge rst) begin
51      if (!rst) begin
52          for (i = 0; i < TAPS; i = i + 1) begin
53              acc[i] <= 0;
54          end
55          y <= 0;
56      end
57      else begin
58          if (fir_en) begin
59              acc[0] <= x * COEFFFS[0];
60              for (i = 1; i < TAPS; i = i + 1) begin
61                  if (i == TAPS-1) begin
62                      y <= acc[i-1] + (x * COEFFFS[0]);
63                  end
64                  else if (i < SYM_TAPS)
65                      acc[i] <= acc[i-1] + (x * COEFFFS[i]);
66                  else
67                      acc[i] <= acc[i-1] + (x * COEFFFS[TAPS-i-1]);
68                  end
69              end
70          end
71      end
72  end
```

CORDIC

- We can store sampled sin wave values in a text file and read them in the testbench or use CORDIC IP to generate the values.
- Coordinate Rotation Digital Computer (CORDIC) uses an input phase angle to calculate the sine and cosine of the angle.
- The output frequency is given by:

$$F_{out} = \frac{2\pi M F_{clk_{cordic}}}{2^{N-1}}$$

- where
 - M : Phase accumulator value.
 - $F_{clk_{cordic}}$: CORDIC clock frequency.
 - N : Number of bits in the phase accumulator.

Test Strategy

- The FIR frequency is set to 30.72MHz and the CORDIC frequency is set to 100MHz.
- $F_c = 0.15F_s = 4.608 \text{ MHz}$.
- I have created 3 test cases by adding sin waves with different frequencies.
 - 0.5 MHz + 1 MHz \rightarrow Both pass.
 - 1 MHz + 8 MHz \rightarrow Only the 1 MHz pass.
 - 6 MHz + 7 MHz \rightarrow Both attenuated.

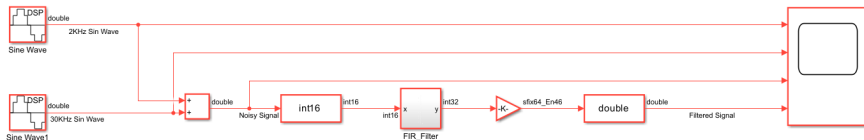
Waveforms

- The output of the equiripple FIR filter with 49 taps.

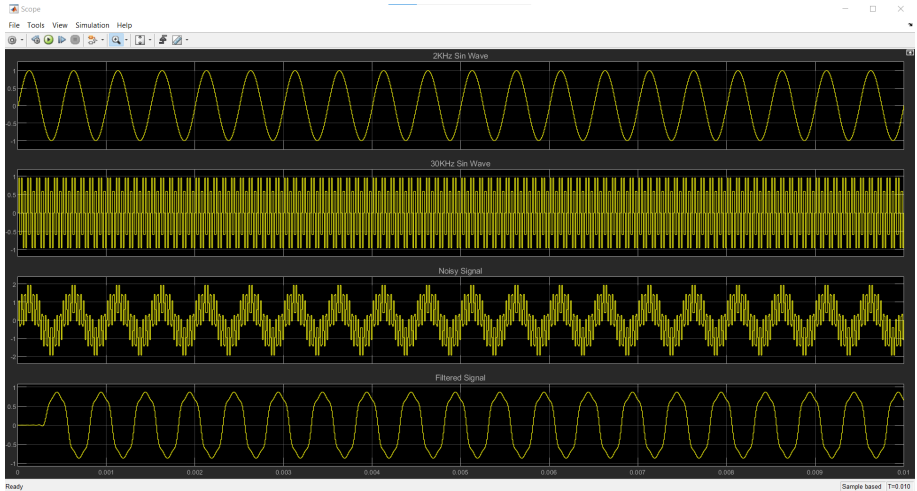


Simulink Model

- "importhdl" is used to import the FIR filter HDL code into Simulink, but System Verilog is not supported, so we will make the Python script generate the Verilog code.
- Testing Strategy:
 - $F_s = 100 \text{ KHz}$.
 - $\omega_c = 0.3\pi \rightarrow f_c = 0.15F_s = 15 \text{ KHz}$
 - Input is $2 \text{ KHz} + 30 \text{ KHz} \rightarrow$ only 2 KHz pass.



Waveforms



Synthesis

- The RTL is compatible with both ASIC and FPGA implementation flows.
- We will use FPGA, as it typically provides dedicated DSP blocks optimized for multiply-accumulate (MAC) operations.

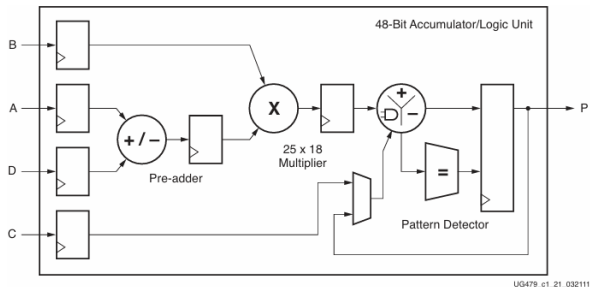
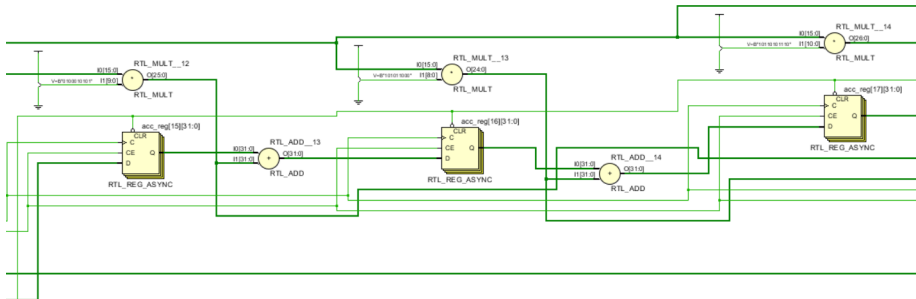


Figure: 7 Series DSP48E1 Slice

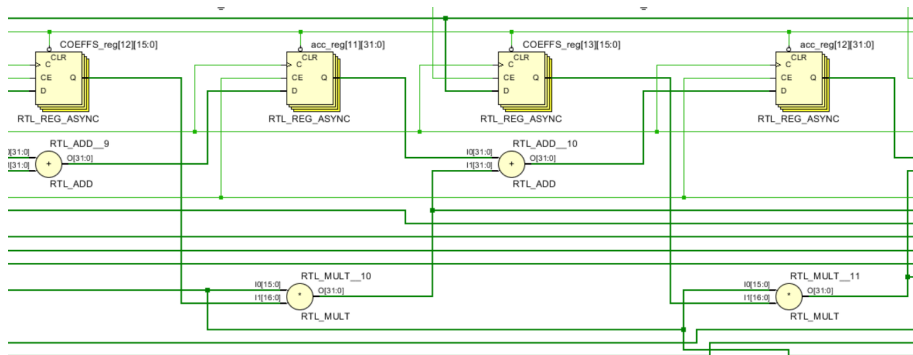
Elaboration

- For fixed coefficients, no registers are needed for the coefficients.



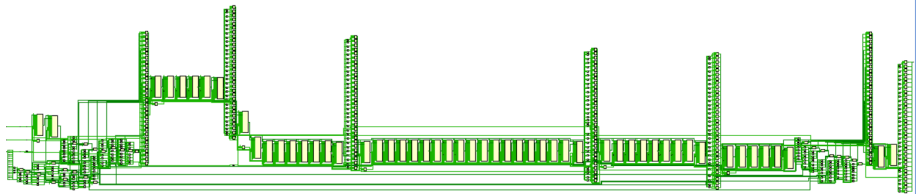
Elaboration

- For reloadable coefficients, the coefficients are stored in flip-flops.



Synthesis

- I will continue with the fixed coefficients design.

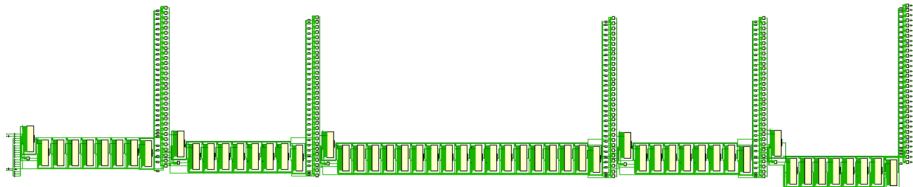


Name	1	Slice LUTs (303600)	Slice Registers (607200)	DSP s (280 0)	Bonded IOB (600)	BUFGCTRL (32)
N FIR_Filter		180	230	53	51	1

Figure: Kaiser Window Filter (61 taps)

Synthesis

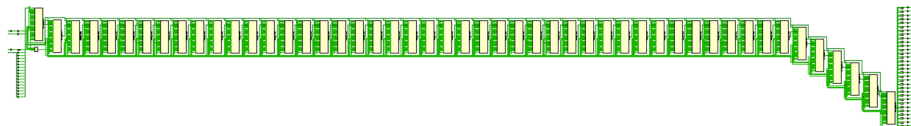
- We can use attributes to further optimize the design.
- We will use (* use_dsp48 = "yes" *) to improve DSP slices utilization.



Name	1	Slice LUTs (303600)	Slice Registers (607200)	DSP s (280 0)	Bonded IOB (600)	BUFGCTRL (32)
N FIR_Filter		81	165	55	51	1

Synthesis

- For equiripple filter, the tool was able to automatically utilize the DSP slices without the need for attributes.



Name	1	Slice LUTs (303600)	Slice Registers (607200)	DSP s (280 0)	Bonded IOB (600)	BUFGCTRL (32)
N FIR_Filter		17	1	49	51	1

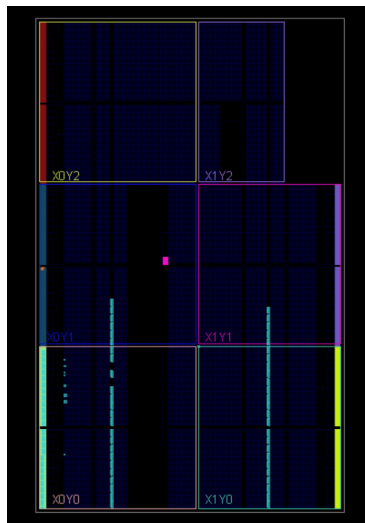
Figure: Equiripple Filter (49 taps)

Constraints

- All the previous results were obtained using XC7VX485TFFG1157-1, but I will continue with Boolean XC7S50CSGA324-1 and 30.72 MHz clock.

```
1 create_clock -period 32.552 -name clk [get_ports clk]
2 set_property -dict {PACKAGE_PIN F14 IOSTANDARD LVCMOS33} [get_ports {clk}]
3
4 # Set Bank 0 voltage
5 set_property CFGBVS VCCO [current_design]
6 set_property CONFIG_VOLTAGE 3.3 [current_design]
7
8 # Input Delays
9 set_input_delay -clock clk -max 2 [get_ports rst]
10 set_input_delay -clock clk -min 0 [get_ports rst]
11 set_input_delay -clock clk -max 2 [get_ports x]
12 set_input_delay -clock clk -min 0 [get_ports x]
13
14 # Output Delays
15 set_output_delay -clock clk -max 2 [get_ports y]
16 set_output_delay -clock clk -min 0 [get_ports y]
```


Device



Reports

Name	Slice LUTs (32600)	Slice Registers (65200)	Slice (8150)	LUT as Logic (32600)	DSPs (120)	Bonded IOB (210)	BUFGCTRL (32)
N FIR_Filter	17	1	8	17	49	50	1

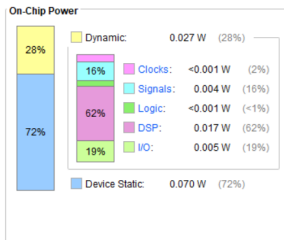
Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 18.156 ns	Worst Hold Slack (WHS): 0.074 ns	Worst Pulse Width Slack (WPWS): 15.776 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 3903	Total Number of Endpoints: 3903	Total Number of Endpoints: 51

All user specified timing constraints are met.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

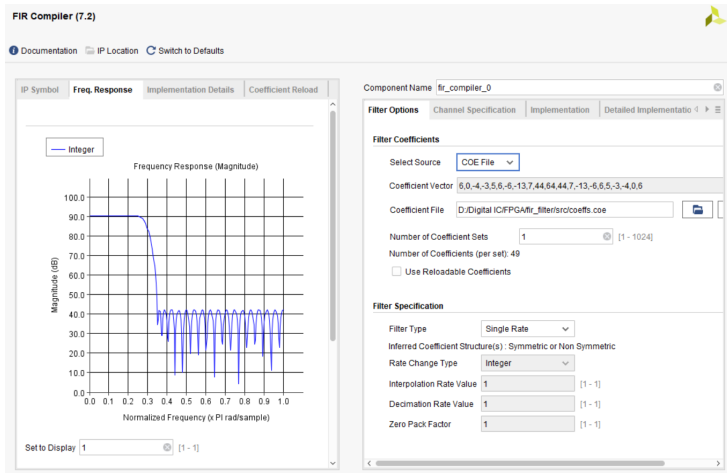
Total On-Chip Power: 0.097 W
Design Power Budget: Not Specified
Power Budget Margin: N/A
Junction Temperature: 25.5°C
Thermal Margin: 59.5°C (12.0 W)
Effective θ_{JA} : 4.9°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity



Optmization

- We can further optimize the design by using Vivado's FIR Compiler IP.






Advantages

- A major advantage of using the IP is the ability to configure both the input sampling frequency and the clock frequency.
- By setting a low input sampling frequency while maintaining a high clock frequency, the design can optimize resource utilization.
- This allows a single DSP slice to perform multiple MAC operations before the next input sample arrives, effectively reducing the total number of DSP slices required.
- let's test with a clock frequency of 100 MHz and a sampling frequency of 44.1 KHz & 100 MHz.

Comparison

Hardware Oversampling Specification

Select Format	Frequency Specification 
Sample Period (Clock Cycles)	1 [1.0 - 1.0E7]
Input Sampling Frequency (MHz)	100  [1.0E-6 - 189952.0]
Clock Frequency (MHz)	100  [0.390625 - 742.0]

Clock cycles per input:	1
Clock cycles per output:	1
Number of parallel inputs	1
Number of parallel outputs	1

Name	Slice LUTs (303600)	Slice Registers (607200)	DSPs (2800)	Bonded IOB (600)
> N fir_top	777	1268	25	52

Comparison

Hardware Oversampling Specification

Select Format	Frequency Specification	▼
Sample Period (Clock Cycles)	1	[1.0 - 1.0E7]
Input Sampling Frequency (MHz)	0.0441	ⓧ [1.0E-6 - 189952.0]
Clock Frequency (MHz)	100.0	ⓧ [1.72265625E-4 - 742.0]

Clock cycles per input:	2267
Clock cycles per output:	2267
Number of parallel inputs	1
Number of parallel outputs	1

Name	1	Slice LUTs (303600)	Slice Registers (607200)	DSPs (2800)	Bonded IOB (600)
> N fir_top		104	170	1	52

Comparison

- Note that using more DSP slices instead of LUTs does not necessarily mean a better design especially for fixed coefficients.
- DSP slices might be needed by more critical parts of the design so you might have to save them if the FPGA is resource constrained.

References

- [1] Dimitris Manolakis, Vinay Ingle, Stephen Kogon, "Applied Digital Signal Processing", Cambridge University Press, 2011.
- [2] Dr. Michael Ibrahim, "Digital Signal Processing", Lecture Notes ([link](#)).
- [3] Verification using CORDIC ([YouTube Video](#)).
- [4] Yaseen Salah's FIR Implementation ([link](#)).
- [5] Equiripple Filters ([YouTube Video](#))