Report

Part 1 Filter Design

1. Get filter

The requirements are

(1) 100 tap FIR filter

(2) transition region of 0.2p~0.23p rad/sample

(3) stopband attenuation of at least 80dB

A low-pass FIR filter can be created using matlab command: firpm(N, f, a, dev);

Now we have a set of coefficients b_unquant and can analyze the frequency response using:

[H, w] = freqz(b_unquant, 1, 1024);

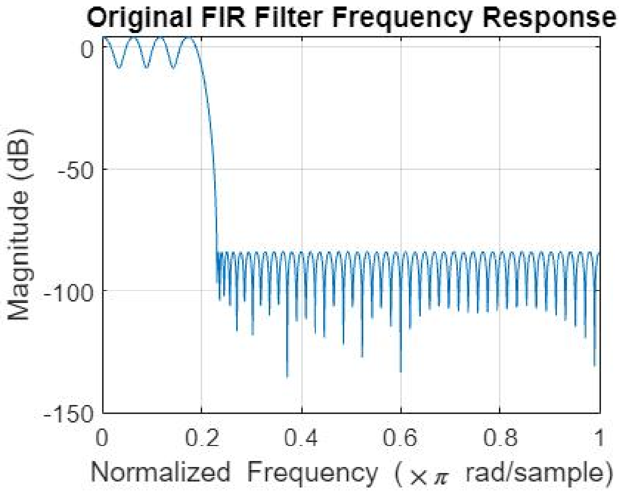


Figure 1: FIR Filter Freq Response

This filter meets the requirement of transition region of 0.2p~0.23p rad/sample and stopband attenuation of at least 80dB

The Passband Ripple is high (12.78 dB) but it cannot be further reduced given the constraint of 80dB attenuation while keeping 100 taps.

2. Quantitation

The quantitation effect is directly influenced by the data width chosen, when the data width is 16 bit, it causes ripple that breaks the -80dB line. When the data width is increased to 32bit, the frequency response align with the original filter (Figure 3).

All inputs and coefficients are quantized into 32-bit signed integer (Q31 form) and the product is defined in 62 bit (h * x >>> 31) The products are accumulated in 62 bits for safety and get truncated to 32-bit at the output. Saturation was used to guard overflow but it should not occur in the design.

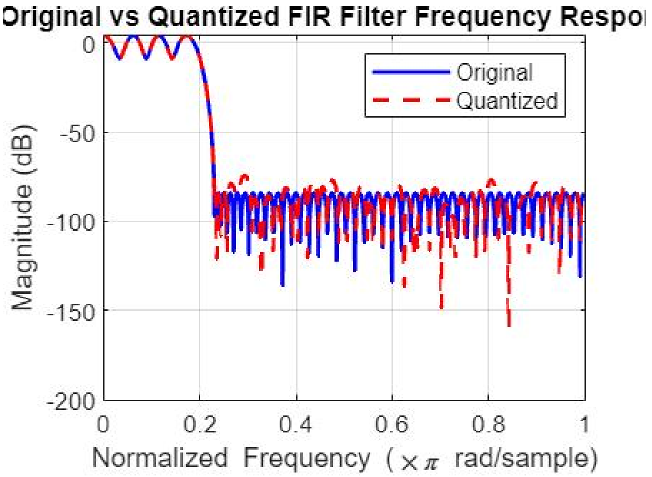


Figure 2: FIR Filter Freq Response Using 16-bit Quantitation

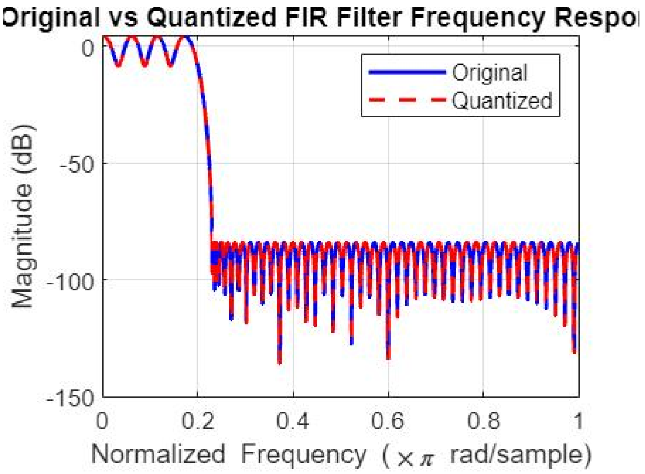


Figure 3: FIR Filter Freq Response Using 32-bit Quantitation

2. Create test vectors

We output the quantized (32-bit) x inputs and filter coefficients into a file. Then we used a matlab function to mimic the behaviors of the hardware, the output of this function should be the expected output from the hardware implementation. We can use these outputs to check the correctness of the RTL design.

Part 2 Implementation

1. We implements the filter in 4 different ways:

(1) Pipeline Only

The filter is implemented in a transposed form shown in Figure 4.

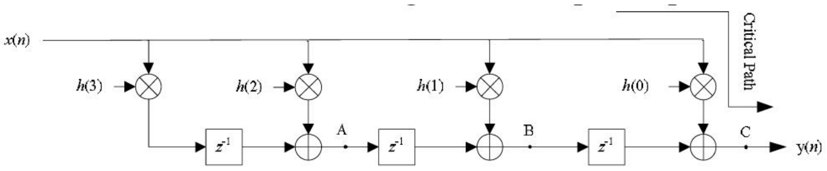


Figure 4: 4-tap Transposed FIR Filter [1]

(2) L=2 Parallel

(3) L=3 Parallel

(4) L=3 Parallel with Pipeline

Reference [1] shows a diagram for a L=4 parallel diagram which can be adapted to create L=2 parallel and L=3 parallel.

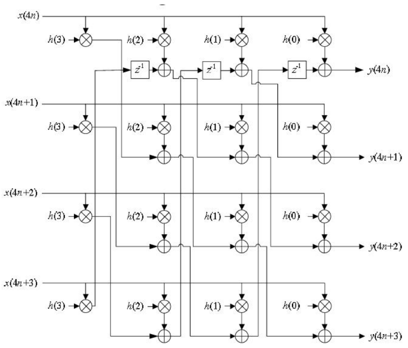


Figure 5: L=4 parallel transposed FIR Filter

2. Implementation Results

Since Vivado requires a complete setup, including port mapping, to run implementation, we will use a dummy top wrapper and assign some arbitrary values to input and coefficients. Then force Vivado not erase them using (* DONT_TOUCH = "true" *)

We measure the performance of these different implementation in the following areas:

(1) The total number of cycles for the filter to process 90 samples in the testbench (simulation).

(2) Worst Negative Slack (WNS) in ns.

(3) The hardware resources used (LUT and FF)

(4) Total on Chip Power (W)

Filter Implem	Total cycles	WNS	LUT	FF	Total on Chip Power(W)
Pipeline only	91	5.442	1643	96	0.123
L=2 only	47	4.5	1626	96	0.123
L=3 only	32	4.646	1610	96	0.123
L=3 & Pipeline	32	5.523	1651	297	0.142

Part 3 Conclusion

1. Effect of Transposed Form

The transposed structure needs a smaller number of registers.

When Adding pipeline registers to the the transposed form, there is no extra input-output latency introduced But a M cycle latency (where M is the number of stages) will be added if using the direct form.

But because the input needs to broadcast to all of the multipliers in the transposed form, it may cause fanout problem during routing while the direct form does not experience this issue because it does not need to propagates the signal across a long distance.

2. Parallel and Pipelining

From the results the pipeling does lead to a lower WNS(1ns less) which indicates a higher achievable clock frequency. But the difference is not obvious. Also the parallelism does not cause a higher resource use, which is unexpected. Some possible explanations may be

(1) Implementation tool optimized the data flow.
(2) The tool may places the coefficients in BRAM
(3) True parallelism was not actually implemented.

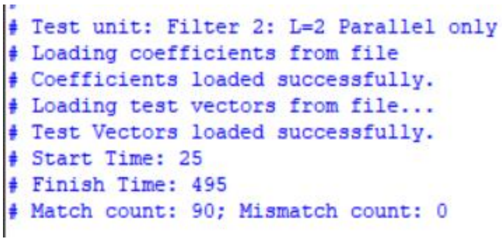
But the most likely reason in my opinion is that the multiplication and addition are implemented using LUTs, and they remain within single LUT's capacity. Additional parallelism (L=2, L=3) may not require more LUTs. The critical path may be similar which can also explain the small difference in WNS. Another clue is that in post-synthesis estimation the parallelism caused significantly more resources, which may indicates a big difference between the architecture in the HDL and the actual implementation.

Reference

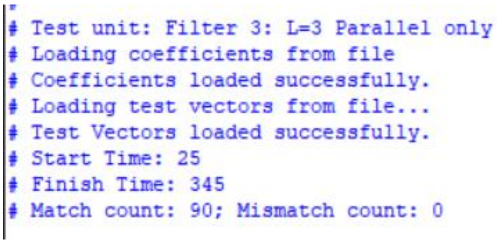
[1] B. Hou, Y. Yao, and M. Qin, "Design and FPGA Implementation of High-speed Parallel FIR Filters," Advances in computer science research, Jan. 2015, doi:
https://doi.org/10.2991/icmra-15.2015.189.

Appendix A

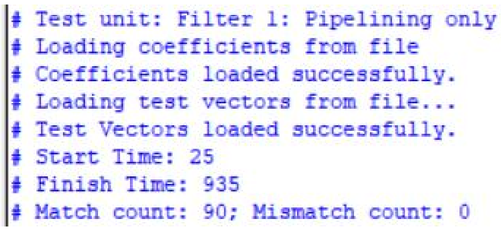
Simulation Results



```
# Test unit: Filter 2: L=2 Parallel only
# Loading coefficients from file
# Coefficients loaded successfully.
# Loading test vectors from file...
# Test Vectors loaded successfully.
# Start Time: 25
# Finish Time: 495
# Match count: 90; Mismatch count: 0
```



```
# Test unit: Filter 3: L=3 Parallel only
# Loading coefficients from file
# Coefficients loaded successfully.
# Loading test vectors from file...
# Test Vectors loaded successfully.
# Start Time: 25
# Finish Time: 345
# Match count: 90; Mismatch count: 0
```



```
# Test unit: Filter 1: Pipelining only
# Loading coefficients from file
# Coefficients loaded successfully.
# Loading test vectors from file...
# Test Vectors loaded successfully.
# Start Time: 25
# Finish Time: 935
# Match count: 90; Mismatch count: 0
```