Course Project: FIR Filter Design and Implementation
Due date: March 15th

Hisen Zhang <zhangz29@rpi.edu>

(1) Description of the use of Matlab for FIR filter design and the structure of your Verilog code

Using Matlab's Filter Designer tool, we were able to specify the desired filter characteristics and directly observe the implications on the filter's frequency response.

The process began with the definition of the filter's specifications as in Figure 1:
Filter Type: Low-pass
Method: Equiripple
Order: 100, to satisfy the design criteria.
Transition band: 0.2-0.23 pi

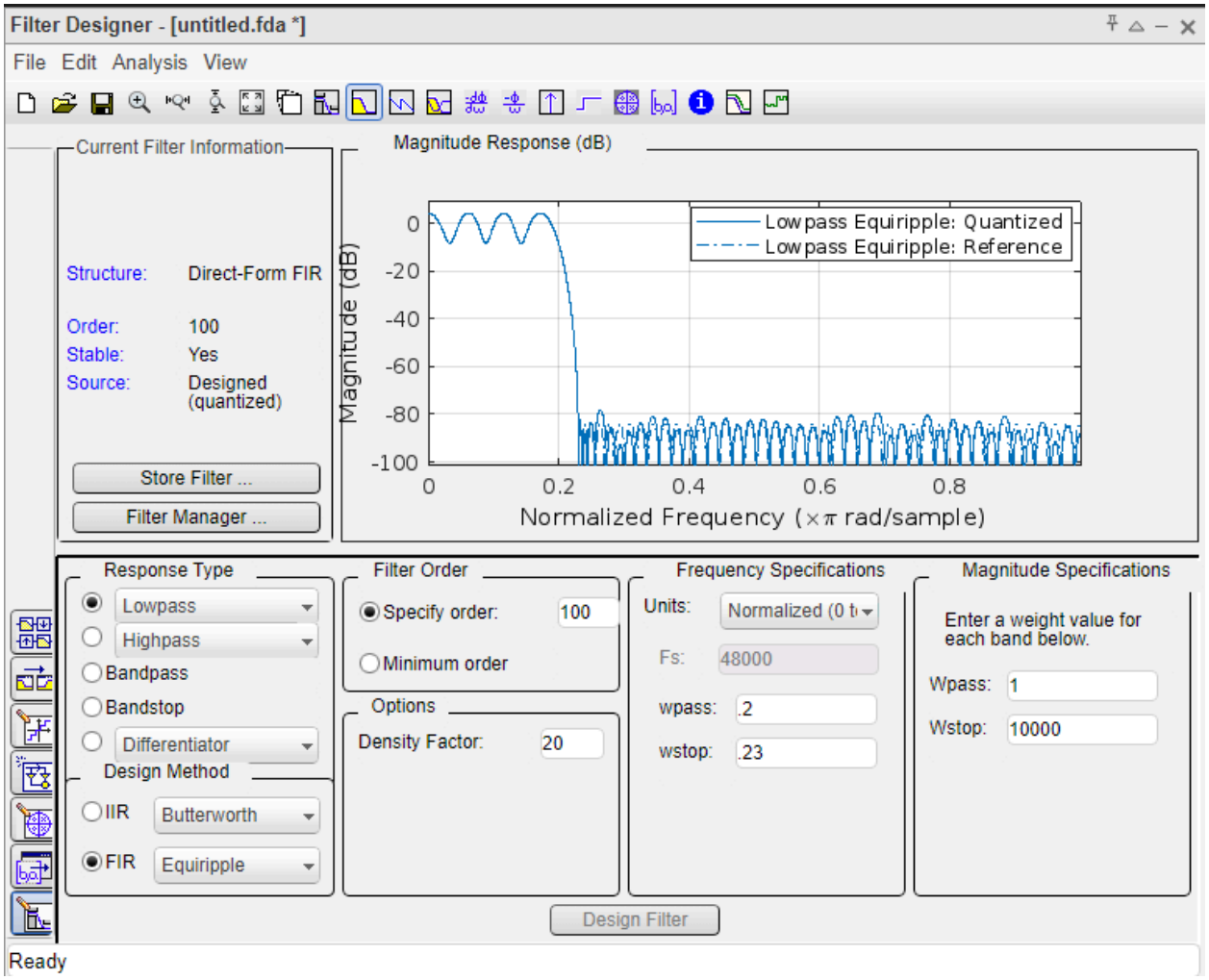

Figure 1 MATLAB Filter Designer

The magnitude specifications were adjusted to set the passband and stopband weights. A higher stopband weight relative to the passband weight ensures the desired stopband attenuation of at least 80dB. We set the passband weight (W_{pass}) to 1 and the stopband weight (W_{stop}) to 10,000, reflecting the significance of stopband attenuation in our design.

The Verilog code implements a parallel FIR filter module called filter with a 3-parallel architecture. The module takes a 16-bit input data sample (data_in) in Q15 format, processes it through three parallel paths, and produces a 35-bit output data sample (data_out) in Q32 format. The module is parameterized to allow flexibility in configuring the input, coefficient, and output widths, as well as the number of filter taps and parallel paths.

The filter coefficients are stored in the COEFFS parameter as an array of signed values in Q15 format. The module utilizes a generate block to instantiate three instances of the fixed_point_mac module, each representing a MAC (Multiply-Accumulate) unit for a subset of the filter taps. The input samples are distributed to the three parallel paths using an array of registers (path_data). The MAC units operate concurrently, processing different subsets of the filter taps in parallel, thereby exploiting parallelism to improve performance.

The module incorporates pipelining techniques to achieve higher throughput. The multiplication and accumulation operations within each MAC unit are pipelined, allowing multiple samples to be processed simultaneously. The input samples are distributed to the parallel paths in a pipelined manner using a sample counter. The outputs from the MAC units are combined in a pipelined fashion, with the final output being obtained by summing the results from all the MAC units when the sample counter reaches the last parallel path. The pipelining helps to reduce the overall latency and increase the throughput of the filter.

The designing script fir_filter.m and the coefficients are in “extra/” folder.

(2) Filter frequency response of the original (un-quantized) filter and quantized filter, comments/thoughts about the quantization effect, and anything you did to deal with overflow

Quantization is an essential process in implementing digital filters, transforming the theoretical floating-point coefficients derived from filter design tools like Matlab into a fixed-point format suitable for hardware deployment. This process is pivotal for ensuring that the filter can be effectively realized on platforms such as FPGAs or DSPs.

Quantization Parameters

Input Quantization:
Word Length: 16 bits
Fraction Length: 15 bits

Range: ±1

Output Quantization:
Word Length: 35 bits
Fraction Length: 32 bits
Range: ±4

Coefficient Quantization:
Word Length: 16 bits
Fraction Length: 17 bits
Range: ±0.25

The resulting magnitude response plot showed two curves (Figure 2): one for the quantized filter and one as a reference, which is the ideal filter response without quantization. The plot illustrated the filter's performance, with the quantized version displaying slight deviations from the ideal, primarily due to the effects of coefficient quantization. These deviations were within acceptable limits, indicating that the quantization did not significantly degrade the filter's performance.

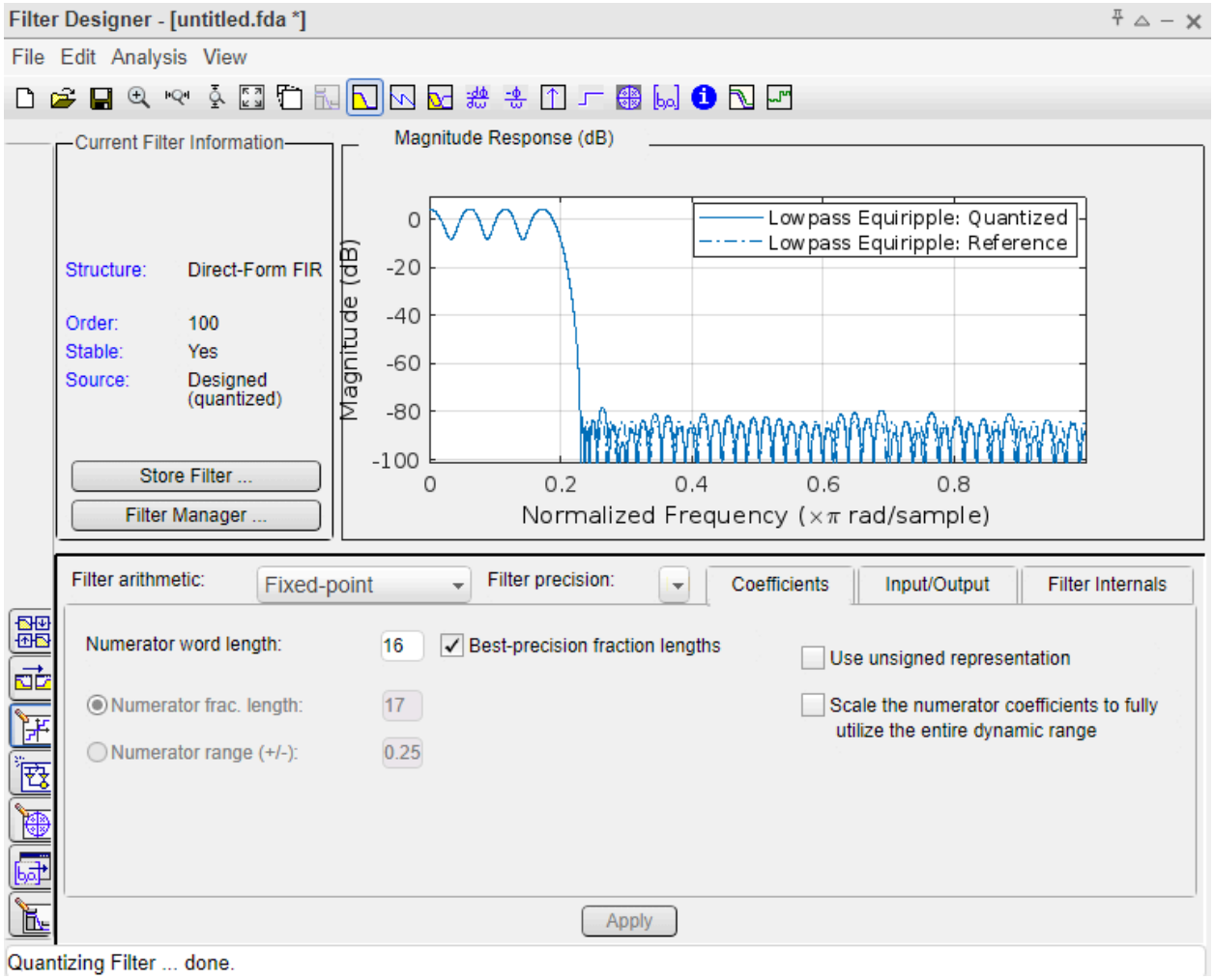

Figure 2 Filter Quantization

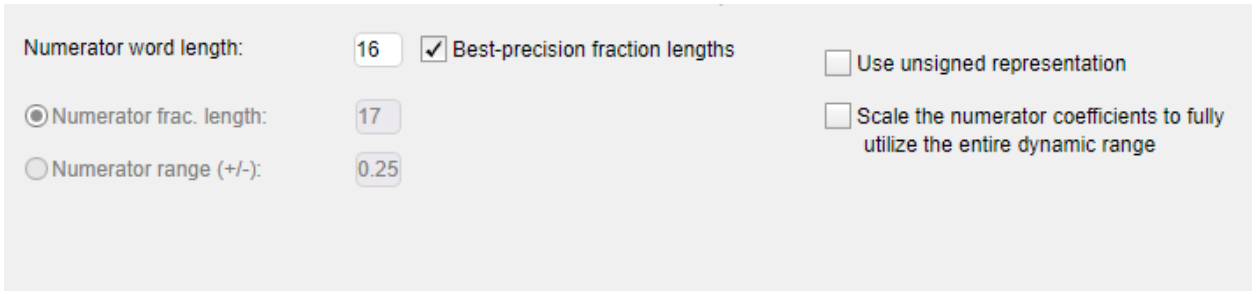

Figure 3 Coefficients Quantization

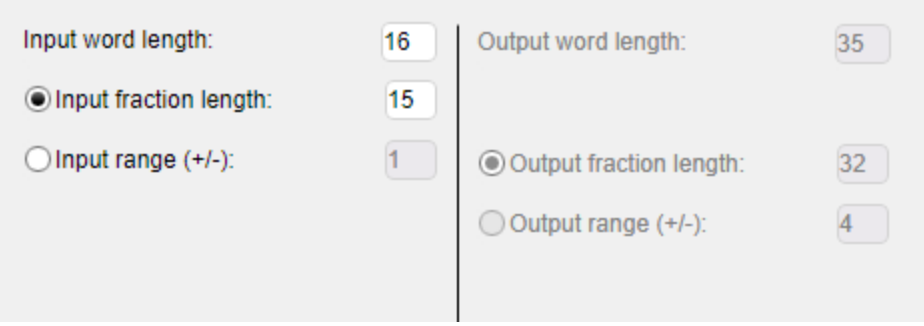

Figure 4 I/O Quantization

The frequency response of the quantized FIR filter:

Passband: These variations are within an acceptable range and do not significantly affect the overall response of the filter.

Stopband: The essential characteristic of stopband attenuation, as specified to be at least 80dB, remains intact, suggesting that the quantization process does not adversely affect the filter's ability to attenuate unwanted frequencies. However, there is a bit more fluctuation in the stop band than in the pass band.

Overflow Management:

The output word length and fraction length are chosen to be significantly larger than those for the input and coefficients. This ensures that the accumulation of products does not result in overflow, a critical consideration in filter implementation. Still, the multiplication result has a width of INPUT_WIDTH + COEFF_WIDTH, which is 32 bits in this case (16 + 16). The resulting format after multiplication is Q30 (15 + 15 = 30 fraction bits).

To align the multiplication result with the Q30 format of the accumulator, we sign-extend it to the accumulator width (35 bits) and then select the 30 most significant bits (excluding the sign bit) to be added to the accumulator. The sign extension is performed by replicating the sign bit of the multiplication result 5 times (35 - 32 = 3, plus 2 additional bits for proper sign extension). Then, the 30 most significant bits of the multiplication result are selected, starting from the bit position INPUT_WIDTH+COEFF_WIDTH-2 down to INPUT_FRAC_WIDTH+COEFF_FRAC_WIDTH-30.

(3) Architecture of your pipelined and/or parallelized FIR filter

The code files are in “src/” folder.

The Verilog code implements a pipelined and parallelized FIR filter architecture with 3 parallel paths (L=3) to enhance performance (Figure 5). The filter takes in 16-bit input samples (data_in) and produces 35-bit output samples (data_out).

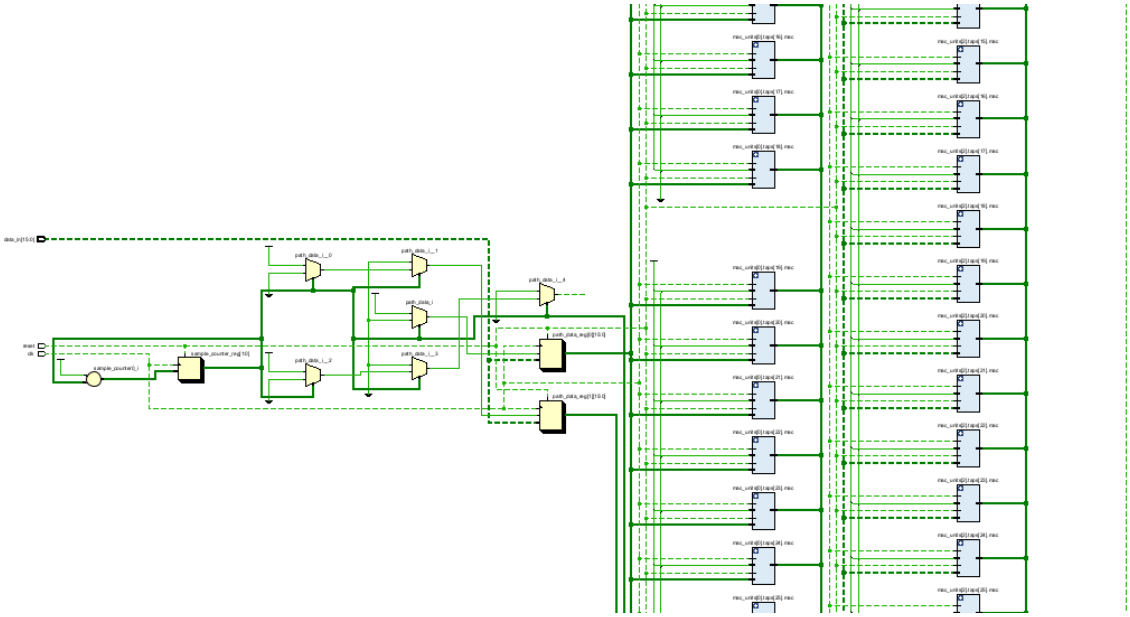

Figure 5 RTL Schematics (Partial)

The architecture leverages both parallelism and pipelining techniques to achieve high throughput. Parallelism is exploited by distributing the input samples across 3 parallel paths, allowing concurrent processing of different sample subsets. The input samples are distributed to the parallel paths in a round-robin manner using a 2-bit sample counter (sample_counter). Multiplexers are used to select the appropriate input for each path based on the sample counter value, and the selected inputs are registered in dedicated path_data registers for each parallel path.

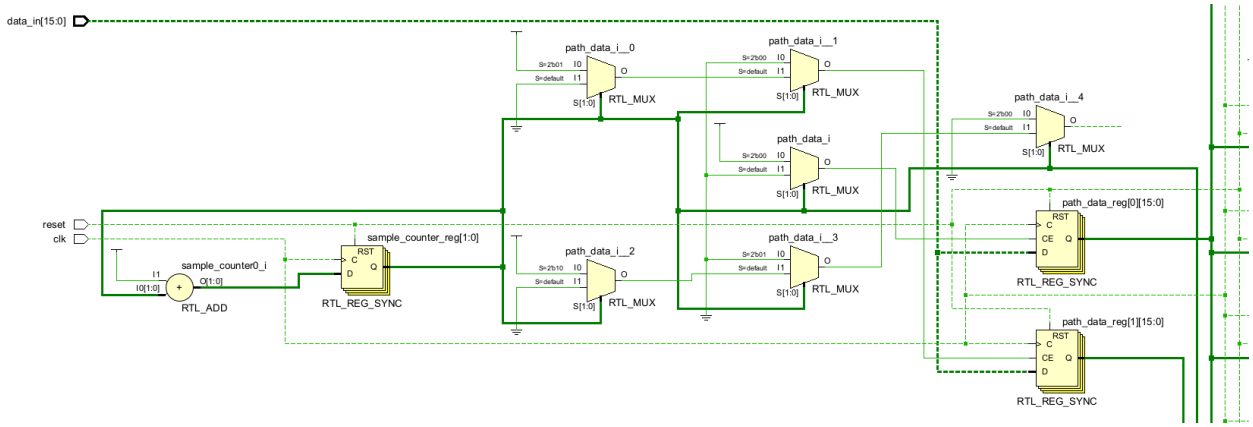

Figure 6 Sample Counter and Mux

Each parallel path contains 33 multiply-accumulate (MAC) units, which are instantiated using a generate block in Verilog. The MAC units are implemented in a separate module called

fixed_point_mac, which performs the multiplication and accumulation operations in a pipelined manner. The coefficients for each MAC unit are stored as parameters, allowing for easy customization of the filter coefficients. The MAC units operate concurrently on different subsets of the filter taps, further exploiting parallelism within each path.

Pipelining is employed within each MAC unit to improve throughput. The multiplication and accumulation operations are divided into multiple pipeline stages, enabling multiple samples to be processed simultaneously in different stages. This pipelining technique helps to reduce the overall latency and increase the throughput of the filter.

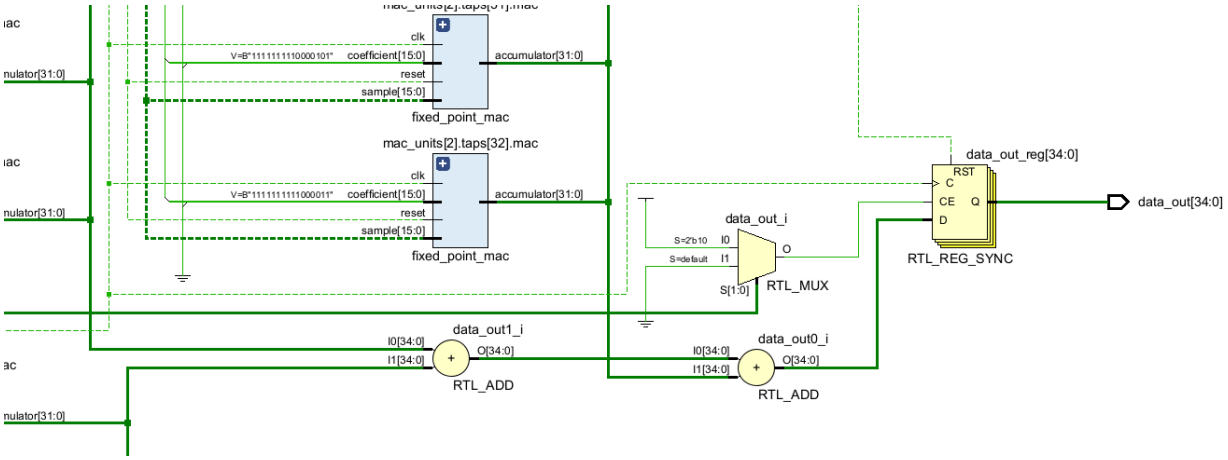

Figure 7 Output & Pipelining

The outputs from the MAC units in each parallel path are accumulated using adders. The accumulated results from the 3 paths are then summed together using a final adder stage, which combines the partial results from each path to produce the final output of the filter (Figure 7). The final sum is registered in the data_out_reg register to synchronize the filter output with the clock and reduce the critical path delay.

(4) Detailed hardware implementation results (e.g., area, clock frequency, power estimation)

Timing and Clock Frequency:

As shown in Figure 8, the filter design achieves a consistent clock frequency across all paths, ranging from 8.932 MHz to 8.943 MHz. The slight variations in frequency can be attributed to differences in routing delays. The design maintains a safe clock skew of 0 ns and a clock uncertainty of 0 ns, ensuring reliable timing performance. The clock delay is reported as 5.411 ns for most paths, with a few paths having a slightly lower delay of 5.405 ns. The logic delay ranges from 3.527 ns to 3.532 ns, indicating a well-balanced design. The net delay is consistent at 3.529 ns for all paths.

Name	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Uncertainty	Bounding Box Size	Clock Region Distance	Clock Relationship	Clock Delay Group	Logic Levels	Routes
Path 1	inf	8.943	5.411	3.532	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 2	inf	8.943	5.411	3.532	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 3	inf	8.94	5.411	3.529	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 4	inf	8.938	5.411	3.527	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 5	inf	8.938	5.411	3.527	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 6	inf	8.937	5.405	3.532	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 7	inf	8.937	5.405	3.532	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 8	inf	8.936	5.411	3.525	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 9	inf	8.934	5.405	3.529	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA
Path 10	inf	8.932	5.405	3.527	0	∞	0	0% x 0%	NA	Safely Timed	Same Clock	14	NA

Figure 8 Timing Among Different Signal Paths

Power Estimation:

Figure 9 presents the power estimation results for the FIR filter design. The total on-chip power consumption is estimated to be 93.927 W, which is within the design power budget. The dynamic power contributes 99% of the total power, while the device static power accounts for the remaining 1%. The design power budget margin is not specified, suggesting that further power optimization techniques could be explored if needed.

The dynamic power breakdown reveals that the DSP components consume the largest portion at 40%, followed by logic at 28%, I/O at 8%, and signals at 24%. This distribution indicates that the DSP units, which perform the multiplication and accumulation operations, are the most power-intensive components in the design. The logic power consumption can be attributed to the control logic, multiplexers, and registers used in the design. The I/O power is relatively low at 8%, suggesting efficient input/output handling. The signal power of 24% represents the power consumed by the interconnects and routing resources.

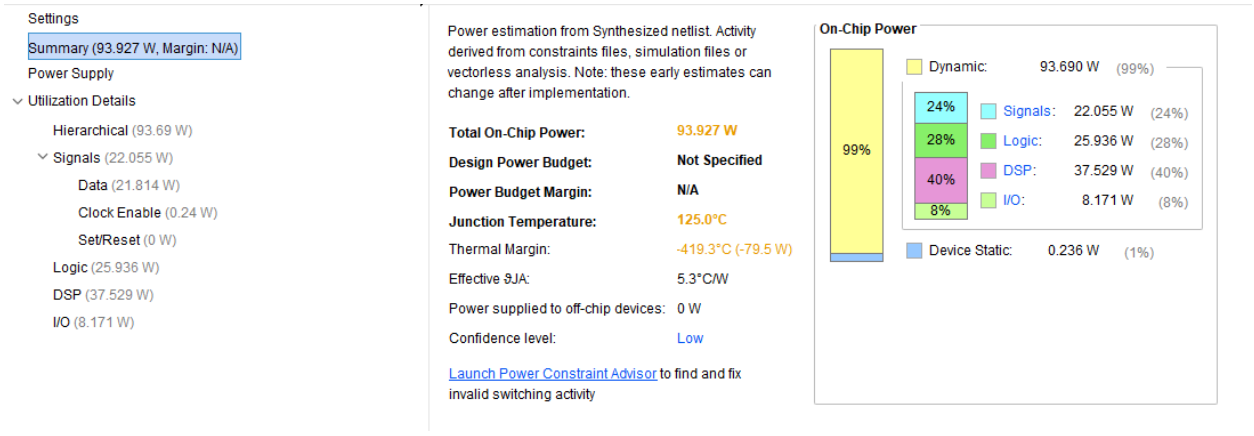

Figure 9 Power Estimate

Chip Area:

The FIR filter design utilizes 322 cells and 53 I/O ports, as reported by the synthesis results. These resources are mapped to the specific FPGA architecture, which consists of configurable logic blocks (CLBs) and I/O blocks (IOBs). The number of cells represents the combinational and sequential logic elements used in the design, while the I/O ports correspond to the input and output signals.

The exact chip area in terms of FPGA-specific blocks depends on the target FPGA device and the specific mapping and placement of the design. The synthesis tool optimizes the design to fit within the available resources of the target FPGA, considering factors such as lookup tables (LUTs), flip-flops (FFs), and routing resources.

(5) Further analysis and conclusion

Summary:

The pipelined and parallelized FIR filter design implemented in this project demonstrates the effective utilization of hardware optimization techniques to achieve high performance and efficiency. By leveraging parallelism and pipelining, the filter can process multiple samples concurrently, resulting in improved throughput compared to a sequential implementation.

The use of 3 parallel paths allows for the distribution of the workload, enabling the filter to handle a higher input data rate. The pipelining within each MAC unit further enhances the throughput by overlapping the execution of multiple operations. This combination of parallelism and pipelining significantly reduces the overall latency and increases the processing speed of the filter.

The hardware implementation results highlight the resource utilization and performance characteristics of the design. The consistent timing performance across all paths ensures reliable operation, while the power estimation results provide insights into the power distribution and thermal characteristics of the design. The chip area utilization, as reported by the synthesis results, demonstrates the efficient use of FPGA resources.

Further optimization and analysis:

The power estimation results indicate that the DSP components consume a significant portion of the dynamic power. Investigating alternative arithmetic implementations or exploring power optimization techniques specific to DSP blocks could potentially reduce power consumption.

Additionally, the quantization process introduces some deviations from the ideal filter response, particularly in the stopband region. While these deviations are within acceptable limits, further analysis and fine-tuning of the quantization parameters could be conducted to minimize the impact on the filter's performance.

The filter design could also be extended to support different filter configurations and parameters. Incorporating parameterization and flexibility in the Verilog code would allow for easy adaptation to various filter specifications and requirements.

Furthermore, the design could be optimized for specific FPGA architectures, taking advantage of device-specific features and resources. This could involve exploring different synthesis and implementation strategies, as well as utilizing vendor-specific IP cores or libraries.

Overall, the FIR filter design and implementation project demonstrates the successful application of hardware optimization techniques and provides a solid framework for further research and development in the field of digital signal processing.