128-TAP FIR FILTER and DESIGN OPTIMIZATIONS USING VIVADO HLS

Arunkumar Ravichandran(arravich)
Debosmit Majumder(dmajumde)
October 12th, 2017

Table of Contents

1. INTRODUCTION

This report deals with various design optimizations applied to a 128-tap FIR filter which include coding and algorithmic optimizations as well as design directives inserted using VIVADO HLS tool.

The following broad design variations and directives have been incorporated:

A. Variable Bitwidths

B. Pipelining

C. Removal of Conditional Statements

D. Loop Partitioning

E. Memory Partitioning

F. Loop Unrolling

The two most important metrics that have been observed while applying these design alternatives are,

1. Throughput
2. Resource Usage.

The optimizations have been applied both separately and in combination to reach a highly optimized architecture with respect to the above design metrics. At the end, all the architectures are compared to find out the Pareto optimal ones.

2. BASELINE FIR FILTER

The baseline design of the 128-Tap FIR Filter corresponds to "**fir128_baseline".**

It uses:

- "int" data types for input, output as well as the coefficients, i.e. 32 bits.
- 1 for-loop for both shifting of 'delay_line' and accumulator value calculation.
- Does not use memory partitioning or pipelining in its implementation.

The resource usage and throughput for this design is as follows. So, this filter design can compute ~ 0.2 million outputs per second.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192

3. FIR FILTER WITH VARIABLE BITWIDTHS

Variable bit-width design corresponds to "**fir128_optimized1".**

Question 1:

It uses:

- “int5” data type for the coefficients, “int8” for the inputs and “int16” for the outputs. It also uses “int9” for the value of “N” and “i”.
- decreasing the bit-width of each variable decreases the resource usage. Hence, less number of Flip-Flops and LUTs are used due to less usage of registers and expressions required for each computation.
- Latency also decreases due to decreased iteration latency in the loop. Now, we require an iteration latency of 4 instead of 5 previously.
- For coefficients, we can’t below “int5”; for input, we can’t go below “int8” and for output, we can’t go below “int16”, else accuracy will be lost and the results won’t match the golden output.
- The estimated clock timing also decreases due to less logic in computation.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
Variable Bitwidths	144	107	1	1	0.264

4. FIR FILTER WITH PIPELINED DESIGN

The pipelined design corresponds to “**fir128_optimized2”.**

It uses the design directive “#pragma HLS pipeline II=”.

Pipelining the design mainly increases the throughput at the cost of increasing the resources.

Question 2.

- the latency and the interval time between inputs decreases to the number of iterations of the loop i.e. it corresponds to the number of taps for the filter designed. This is because the loop is unrolled and all the operations inside the loop take place in parallel.
- However, the initiation interval (II) does not affect the design because of lack of sufficient memory ports without complete memory partitioning. The number of clock cycles taken remain at 128 (for the 128-tap filter). Hence, increasing the value to more than 128 does not make sense because the compiler optimizes to minimum possible interval. So, the largest value for II for an optimized design should be 128 or, the number of filter taps.
- However, with complete memory partitioning, increasing the Initiation Interval (II) in increments of 1 also increases the loop latency to the value of II. It thus increases interval time between outputs or decreases the throughput.
- So, pipelining should be used along with complete memory partitioning to have the maximum effect on design.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
Pipelining	5507	4736	20	1	0.89

5. FIR FILTER WITH REMOVED CONDITIONAL STATEMENTS

Conditional statements are resource expensive in nature. In this FIR-128 implementation, the if/else statement is present inside the for loop. Therefore, for every iteration of the loop, this if/else statement is going to be executed. Since the if x==0 is checking only one iteration(last), we can remove the if/else statement and modify the code in such a way that the function in the if case is executed after the end of for loop.

Question 3:

Once the if/else is statement is removed, the resource usage is significantly decreased. The number of LUTs and FFs as shown in table 1 have been decreased which indicates the reduction in the logical hardware created by the HLS tool to check if the condition has been satisfied.

We can also notice a significant improvement in the performance. The throughput performance increased a lot because the if x==0 check inside the for loop is avoided to be executed for 127 times.

Once the if/else statement is removed, we can notice that the loop can be optimized with parallelism. From the table 1, you will notice the significant improvement in performance.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
Remove conditional statements with pipeline	10860	9463	60	2	0.93
Without pipeline	123	71	1	1	0.53

6. FIR FILTER WITH LOOP PARTITIONING

Loop partitioning is nothing but dividing the for loop into two fundamental operations. By implementing this way, we can optimize each of the for loop.

Question 4

The loop is divided into two for loops: First for shifting data in the delay line and the 2nd for lop for calculating the output y.

The resource usage increased slightly and throughput performance is better than baseline 128-tap.

We used pipelining in each for loop and found the throughput performance increasing with the increment in resource usage.

The loop unrolling was also done. With the loop unrolling, we are just trying to reduce the number of iterations by performing the data manipulations in the for loop. Loop unrolling is a kind of parallelism where we execute a part of the loop in parallel.

As discussed in the FPGA programming book, we used the code available for loop unrolling and compared the results with the baseline.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
Loop partitioning	245	268	4	1	0.193
Loop partitioning with parallelism	10714	9197	60	2	0.93

7. FIR FILTER WITH MEMORY PARTITIONING

Question 5

Memory partitioning uses 3 types: complete, cyclic and block. Memory Partitioning should be applied to a pipelined design to achieve the best results with respect to throughput.

Complete Partitioning

The memory is completely partitioned into separate registers. So, pipelining the design uses the separate registers to compute operations in 1 cycle. Hence, FFs and LUTs usage are increased.

Cyclic Partition

This kind of partitioning creates smaller arrays by interleaving elements from the original array. In this case, memory array may be partitioned into "N" number of arrays depending on the loop partitioning factor. In the pipelined design, cyclic partitioning gives an increased throughput comparable to the complete partitioning of memory. Also, the resource usage increased significantly.

Block Partition

This kind of partitioning creates smaller arrays from consecutive blocks. In this case, the factor gives the number of memory blocks the array will be partitioned into. Block partitioning gives significant increase in throughput and also increases resource usage. Both the throughput and the resource usage are comparable to that of cyclic and complete partitioning.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
128 tap - Complete Memory P with pipe	4370	6455	20	0	101
128 tap - Complete Memory P wo Pipeline	1434	1096	1	0	0.53
128 tap - Block and Cyclic with pipeline give same performance	4370	6455	20	0	101

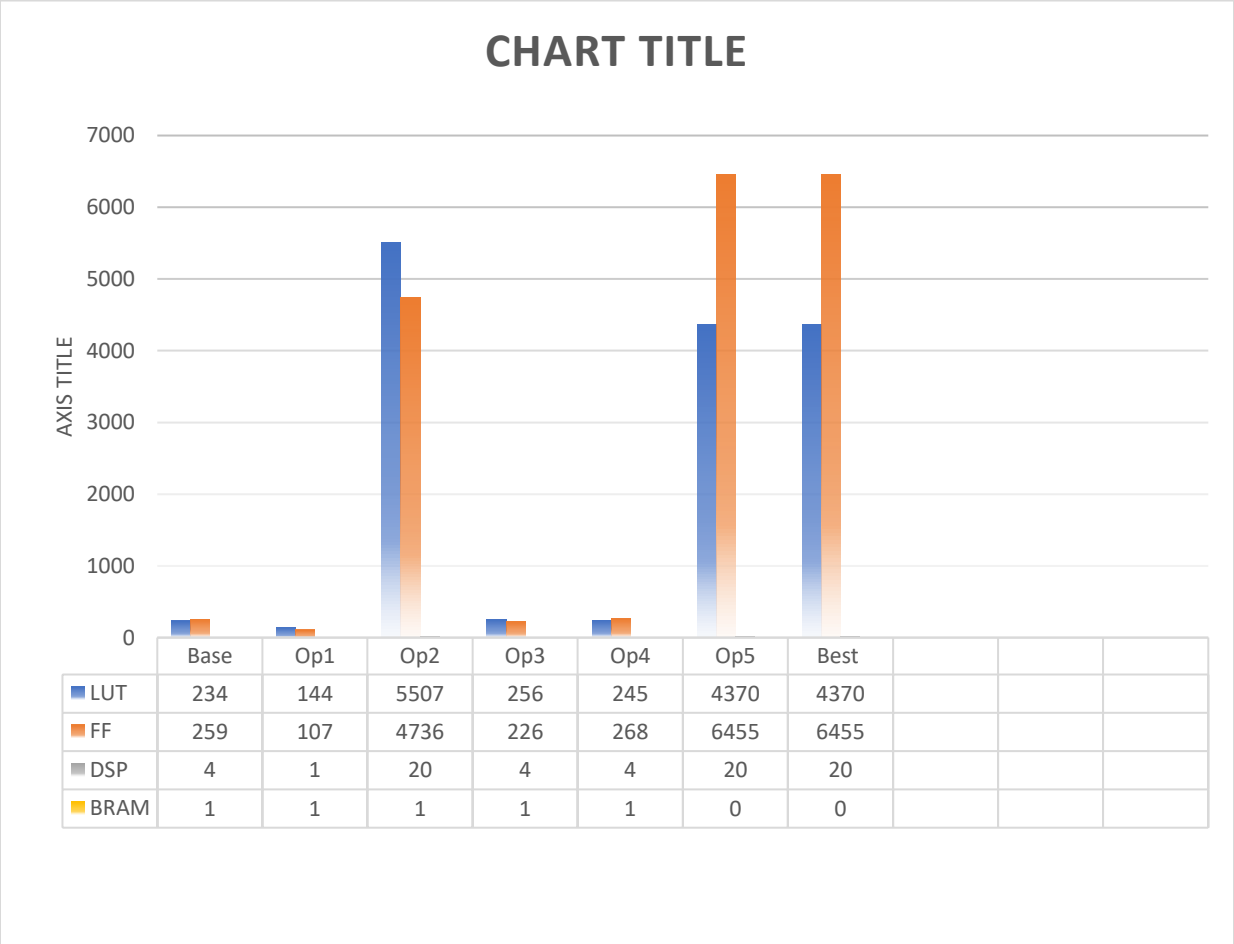
8. FIR FILTER WITH BEST DESGIN

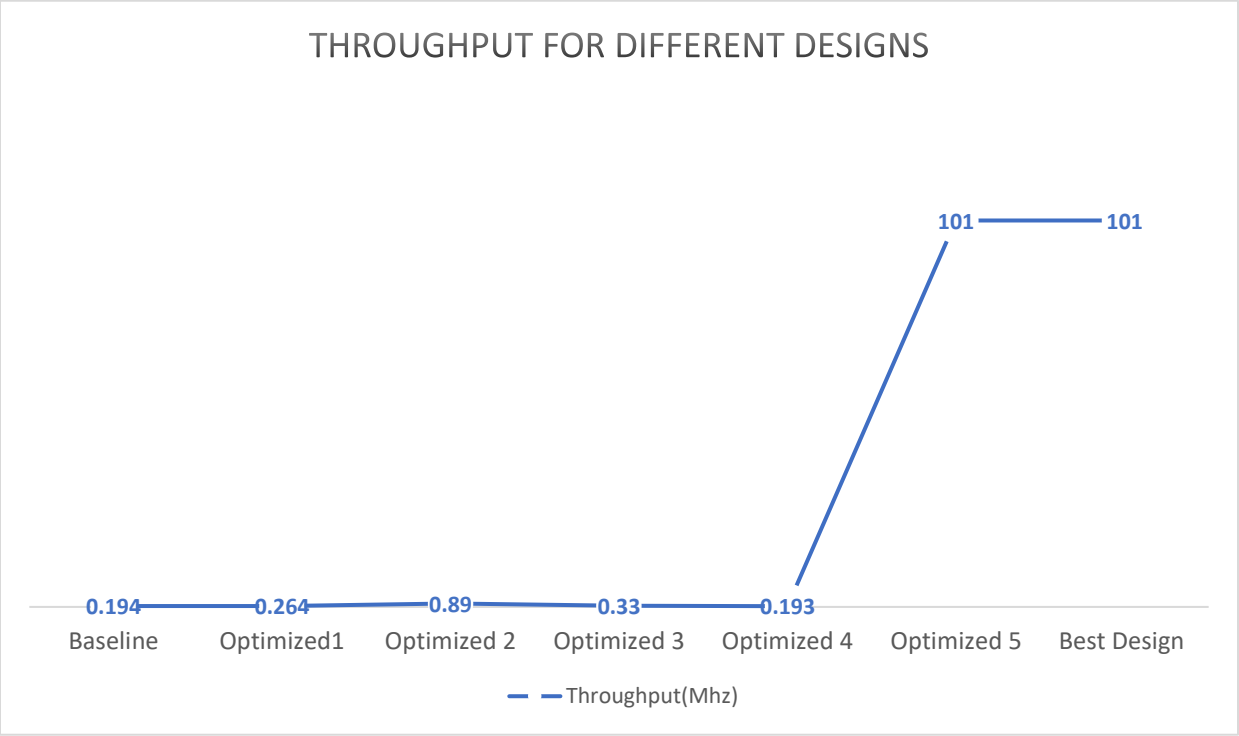
Question 6

We think the best design is one with high throughput and very less number of resource usage.

- As we can see that the resource usage is reducing with bit width modification, removing the conditional statements our design will implement that.
- As we notice from the throughput graph, we can see that the pipelining with memory partitioning is improving the throughput significantly. Hence decision was made to incorporate this. Loop partitioning is avoided as it increases the resource usage but doesn't significantly improve the throughput performance.

	FFs	LUT	DSP	BRAM	Throughput(MHz)
Baseline	234	259	4	1	0.192
Best Design	4370	6455	20	0	101Mhz




THROUGHPUT FOR DIFFERENT DESIGNS
0.194
0.264
0.89
0.33
0.193
101
101
Baseline
Optimized1
Optimized 2
Optimized 3
Optimized 4
Optimized 5
Best Design
Throughput(Mhz)