CSE237C - Project1

Resource usage and Performance are summarized in tables I and III and are plotted in 1 and 2.

Question1 - Variable Bit-widths: Answered with reference to Optimized1 design.

a) The filter coefficients all lie between -16 to +15. So, it is enough to allocate 5 bits to each filter tap considering signed integer values. As the filter inputs in the input.dat file lie in the range of -128 to +127, 8 bit signed int array is enough to represent each input value. Finally, each of the multiplication operation results in 8+5 = 13 bit output and accumulation of 128 such 13 bit numbers gives an output with maximum bit depth of 13 + $\log_2 128 = 20$ bits. So, the variable to store the accumulated result is adjusted to 20 bit-depth in-order to maintain the accuracy.

b) Compared to the baseline implementation, the block ram usage decreased due to smaller bit-depth usage. Subsequently, the Flip flop and LUT usage also went down. The DSP usage remained to be at 1 unit which is the least amount of usage possible for any implementation. Overall this design has a little higher throughput of 0.19MHz compared to the baseline.

	BRAM_18k	DSP	FF	LUT
Baseline	2	1	332	247
Optimized1	1	1	78	171
Optimized2	1	1	136	223
Optimized3	1	1	116	236
Optimized4	1	12	1435	5422
Optimized5	0	12	1515	3437
Best	0	12	2255	3696

TABLE I: Resource usage for different designs

Question2 - Pipelining: Answered with reference to Optimized2 design which includes bit-width optimization in addition to loop pipelining with initiation interval(II) =2.

	II=2	II=3	II=4	II=5	II=6	II=7
Loop Latency	262	388	515	642	770	770

TABLE II: Comparison of pipelining performance for different Initiation interval

a) When II is set to 1, HLS tool automatically changed it to II = 2 because of the resource contention. The loop latency is tabulated in table II. Here the iteration latency is 6. As the II is increased, the loop latency starts monotonically increasing up to II=6 and saturates(stays constant) beyond II=6. This is because the iteration latency is 6 cycles and in-order to gain any advantage from pipelining, the initiation interval must be less than iteration latency. So, in general the saturation point in loop latency starts with Loop latency equals to iteration latency.

b) The design throughput has increase to 0.58 MHz due to pipelining but the flip flip and LUT usage increased when compared to optimized1 design. The DSP usage remained the same and has better utilization (i.e. accepting input data at a higher rate) due to pipelining.

	Latency # of cycles	Initiation interval # of cycles	Estimated timing(ns)	Throughput (MHz)
Baseline	897	898	6.912	0.16
Optimized1	769	770	6.508	0.19
Optimized2	261	262	6.508	0.58
Optimized3	134	135	6.508	1.13
Optimized4	119	120	7.075	1.17
Optimzed5	4	5	7.013	28.5
Best	4	1	7.013	142.5

TABLE III: Performance table

Question 3- Removing Conditional statements: Answered w.r.t optimized3 design. It is optimized for Bit-widths, Loop pipelining and code hoisting.

a) Removing the conditional statement within the loop increased the pipelining efficiency. The loop latency reduced by almost 50% and consequently the throughput also doubled compared to the previous design. We observe the reduction in loop latency due to the reduced Initiation interval for the loop from 2 to 1.

b) The design throughput has increase to 1.13 MHz. But the resource usage decreased slightly because the hardware to perform the conditional check need not be synthesized when the code is hoisted.

Question4 - Loop partitioning: Answered w.r.t optimized4 design. It is optimized for Bit-widths, Loop pipelining, code hoisting and loop partitioning.

a) Partitioning the loop may give opportunity to do loop unrolling. But here in this case, unrolling alone and didn't increase the design performance as evidenced by the throughput being almost the same as the previous design at 1.17 MHz. This happens because the variables that are stored in the block ram cannot be accessed all at once. So, even when the computation is parallelized, the design is bottlenecked by the memory access.

b) The hardware usage increased significantly because the loop is completely unrolled here. The DSPs, flip flops and LUTs increased by at least an order of magnitude because of parallelized hardware. Also, the hardware usage for complete unrolling both with and without loop partitioning looks exactly the same.

Question5 - Memory partitioning: Answered w.r.t optimized5 design. It is optimized for Bit-widths, Loop pipelining, code hoisting, loop partitioning + unrolling and memory partition.

a) When the tapped delay line is set to be completely partitioned, there is no more utilization of the BRAM blocks and Flip flop utilization increased slightly compared to the previous design. Interestingly, the LUT utilization reduced by approximately 50%. Overall, the design has throughput of 28.5MHz because memory access is no longer a bottleneck and unrolling of loops can be put to full advantage.

b) Instead of using only Flip flops to make the delay line, there are other options to split a delay line into multiple blocks that are kept under different BRAMs for simultaenous access. Following table IV summarizes the latency and resource usage for different kinds of array partition. Here the size column indicates the size of each block in cyclic or block allocation of memory. Increasing the size parameter for both block and cyclic partitions reduces the latency as it makes most parts of the memory simultaneously accessible. But if we try to achieve the latency of 4 cycles with these methods, it leads to very inefficient implementation.

	Size	BRAM_18k	DSP	FF	LUT	Latency
Block	4	4	12	1373	5068	46
	16	16	12	1226	5471	10
Cyclic	4	4	12	1294	5120	47
	16	16	12	1195	5520	11
Complete	-	0	12	1515	3437	4

TABLE IV: Resource and performance comparison of different memory partitions

Question6 - Best design: Answered w.r.t best design. It is optimized for Bit-widths, Loop pipelining, code hoisting, loop partitioning + unrolling, memory partition and function pipelining.

a) In the optimized5 design, we see that the fir block has a latency of 4 cycles which is the same as the initiation interval for the block. So, the block was able to accept one input sample every 4 cycles. By performing function level pipelining, we can divide the whole block into sub-blocks and store the intermediate values in registers which enables accepting one input sample every clock cycle. The design achieves 142.5MHz throughput.

b) For pipelining, intermediate registers are necessary and increases the usage of the Flip flops almost by 700 units compared to the optimized5 design.

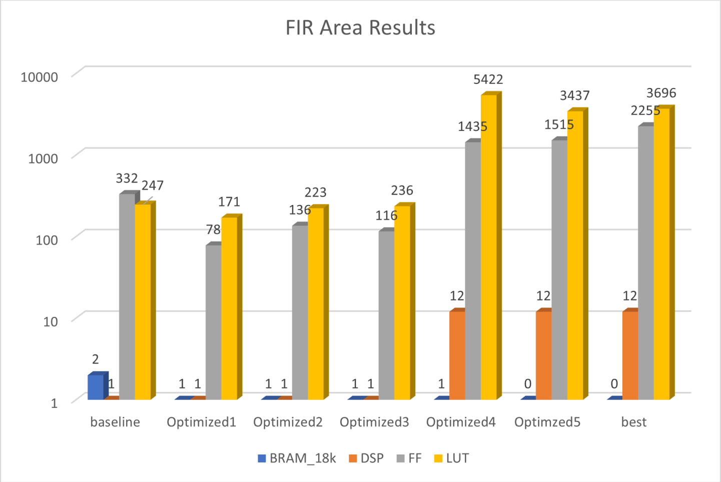

Fig. 1: Resource usage for different optimizations

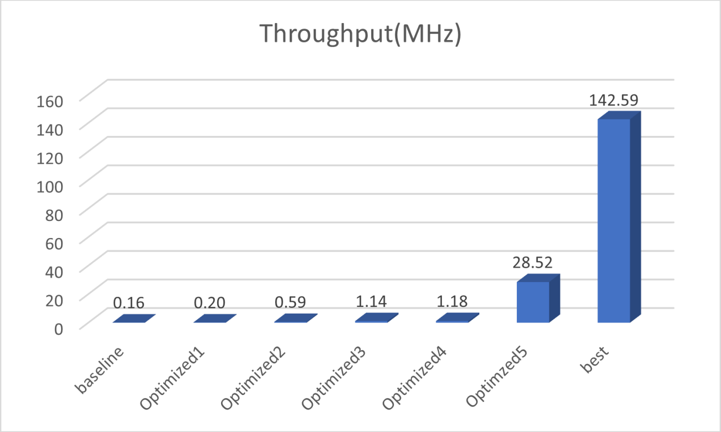

Fig. 2: Throughput for different optimizations