Project 1: FIR Filter

Github link: [cse-237c-project]

Soumil Paranjpay
Navya Jain
Ishita Chawla
Divyang Wadhwani

Q1: FIR11 Baseline

a.

Latency (cycles)	Initialization Interval (cycles)
19	20

b.

BRAMs	DSPs	LUTs	FFs
0	2	383	733

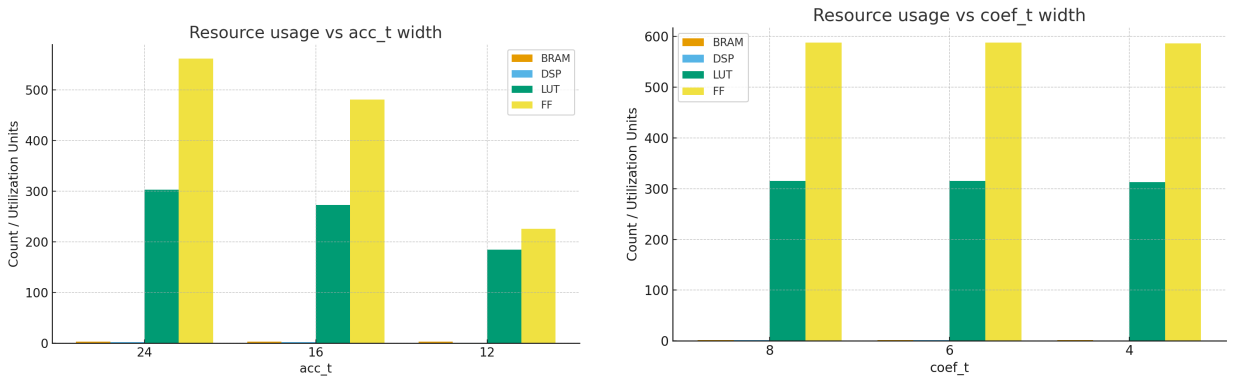
Q2: Variable Bit-Widths

a)

coef_t	latency	II	BRAM	DSP	LUT	FF
8	135	136	2	2	315	588
6	135	136	2	2	315	588
4	135	136	2	0	313	586

acc_t	latency	II	BRAM	DSP	LUT	FF
24	133	132	3	2	303	562
16	135	136	3	2	273	481
12	132	131	3	1	185	226

b) Minimum bitwidth we could use while matching the golden output were coef_t = 5 and acc_t = 16.



BRAM and DSP stayed almost the same across different bit widths.

Q3: Pipelining

a)

Case	latency	II	BRAM	DSP	LUT	FF	Throughput (MHz)
Baseline	135	136	3	2	383	733	1.063794

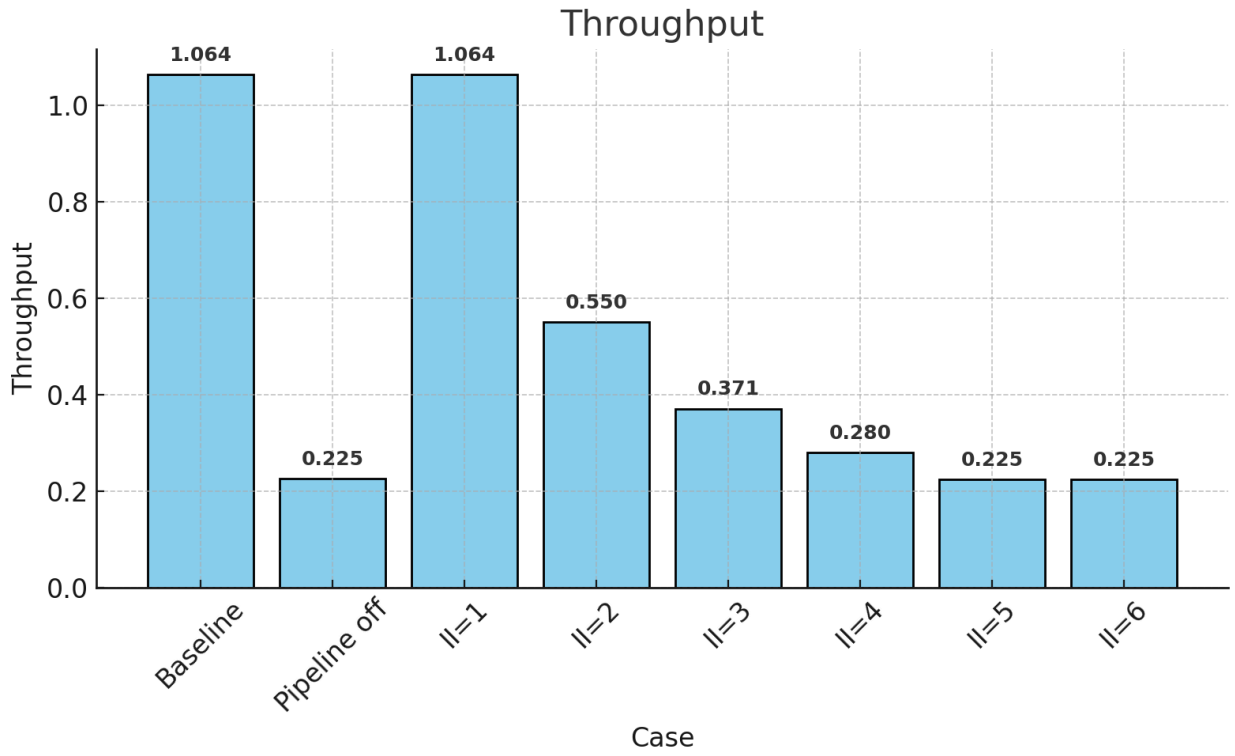
b)

Case	latency	II	BRAM	DSP	LUT	FF	Throughput (MHz)
Pipeline off	641	642	2	2	279	412	0.225352

c)

Case	latency	II	BRAM	DSP	LUT	FF	Throughput (MHz)
II=1	135	136	3	2	305	610	1.063793
II=2	262	263	2	2	321	513	0.550099
II=3	389	390	2	2	318	480	0.370964

II=4	516	517	2	2	323	480	0.279837
II=5	643	644	2	2	300	453	0.224652
II=6	643	644	2	2	300	453	0.224652



d) Initiation Interval (II) = 5 is the point where increasing II no longer makes sense. At this value, the throughput becomes the same as the pipeline off case and remains unchanged for II = 6. Beyond II = 5, there is no improvement in throughput, while latency and other resource metrics also show no benefit.

e) Based on our observation, Vitis HLS automatically pipelines the loop with a default initiation interval of 1. The Baseline and II=1 cases show identical latency, throughput, and resource usage. Therefore, even without setting the pragma, the tool effectively applies II=1 pipelining by default in this design.

Case	latency	II	BRAM	DSP	LUT	FF	Throughput (MHz)
Baseline	135	136	3	2	305	610	1.063794
II=1	135	136	3	2	305	610	1.063794

Q4: Removing Conditional Statements

A. With automatic pipelining:

Latency (cycles)	Initialization Interval (cycles)
134	135

BRAM	DSP	LUT	FF
2	2	316	418

Compared to the baseline, the version without conditional statements achieved a one-cycle reduction in both latency and initiation interval. This improvement is likely due to the simplified control logic resulting from the removal of branch operations.

B.
With pipelining disabled:

Latency (cycles)	Initialization Interval (cycles)
636	637

BRAM	DSP	LUT	FF
2	2	270	345

With manual pipelining turned off, the latency and II reduced by 5 cycles, a bigger improvement than the pipelined version. When pipelining is disabled, the design executes operations sequentially, so every extra control step or branch directly adds to the total latency. Removing conditional statements eliminates those control overheads entirely, which leads to a larger visible reduction in latency.

Q5: Loop Partitioning

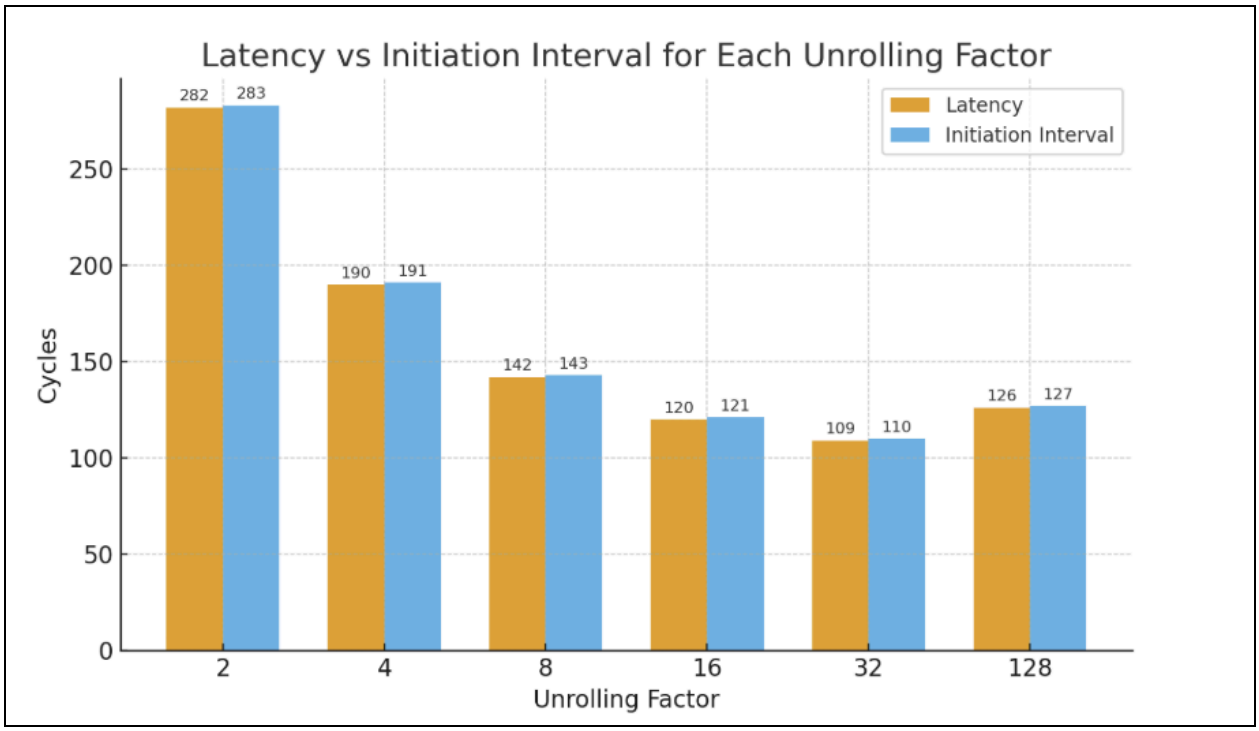
a. We have a combined shift-accumulate loop in the baseline code of our FIR128. This code, although beneficial from a sequential execution perspective can be further optimized in terms of hardware efficiency. Separating out the two core functionalities in the loop i.e. the shift register and the MAC operation can help us add refactoring to each of these hardware implementations separately, and consequently, efficiently. For example: we can implement loop unrolling and pipelining for each of the 2 separate loops.

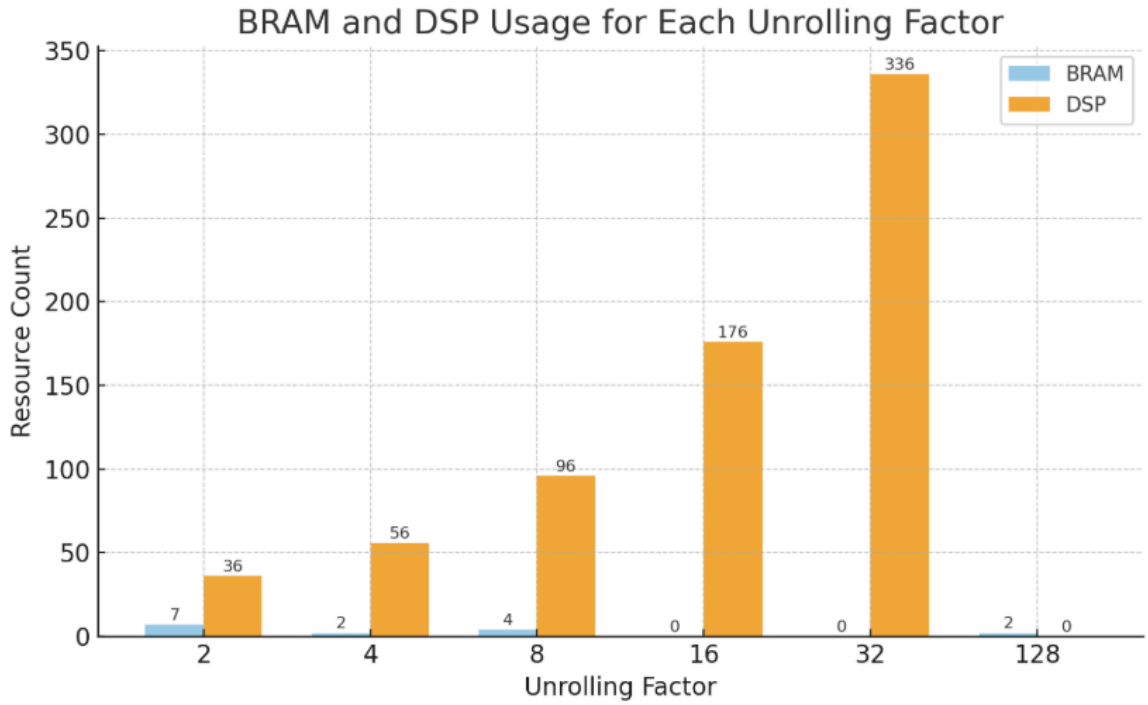
b. We can see an increase in both the latency and resource usage for the FIR implementation with partitioning and no further refactoring applied to each of the loops.

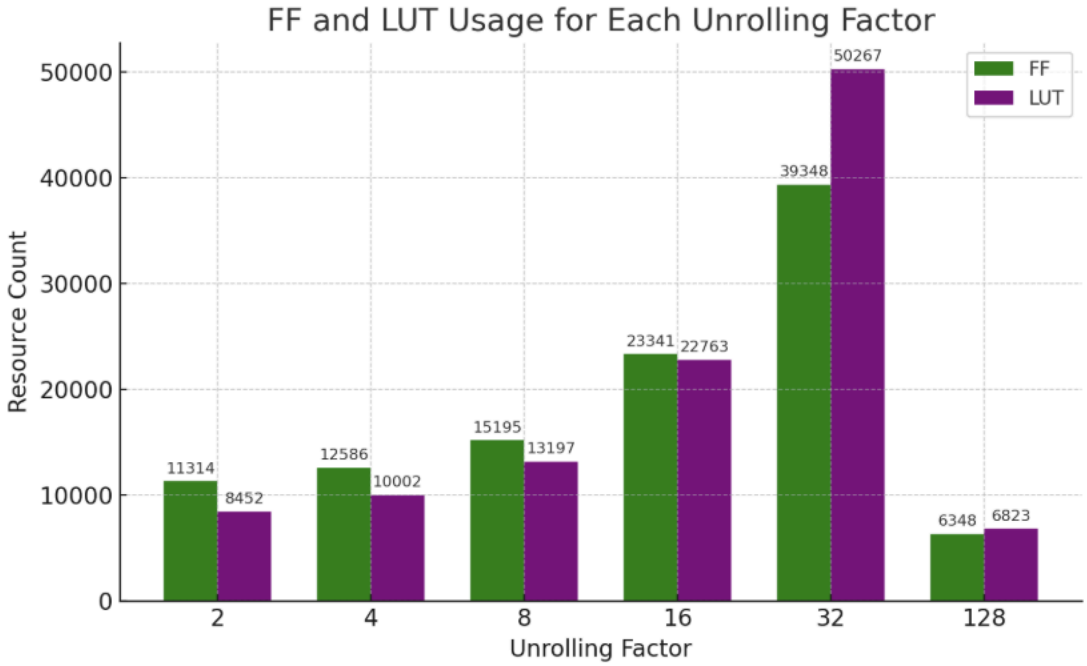
Partitioning	Latency	Initiation Interval	Resource Usage			
			BRAM	LUT	FF	DSP
Yes	267	268	3	343	400	2
No	135	136	3	305	610	3

c. For the given architecture of FIR128, we applied loop partitioning as mentioned above. Subsequently, we applied loop unrolling for each of the loops for each of the factors shown in the table below.
We could see that the latency initially rise for low unrolling factors and seem to stabilize around the high unrolling factors.

Unrolling factor	Latency	Initiation Interval	Resource Usage			
			BRAM	LUT	FF	DSP
2	282	283	7	8452	11314	36
4	190	191	2	10002	12586	56
8	142	143	4	13197	15195	96
16	120	121	0	22763	23341	176
32	109	110	0	50267	39348	336
128	126	127	2	6823	6348	0


Latency vs Initiation Interval for Each Unrolling Factor
Latency
Initiation Interval
Cycles
Unrolling Factor
0
50
100
150
200
250
2
4
8
16
32
128
282
283
190
191
142
143
120
121
109
110
126
127


BRAM and DSP Usage for Each Unrolling Factor
BRAM
DSP
Resource Count
Unrolling Factor
0
50
100
150
200
250
300
350
2
4
8
16
32
128
7
36
2
56
4
96
0
176
0
336
2
0



d. For the given architecture of FIR128 there seems to be some overlap between loop unrolling and pipelining. Loop unrolling leads to a separate hardware sequence for each of the unrolled loops, whereas loop pipelining requires additional hardware corresponding to the largest cycle consuming component. (the multipliers). Yes, we can apply unrolling and pipelining simultaneously to save the additional hardware required and gain maximum throughput.

Q6: Memory Partitioning

a. We kept the loop unrolling factor as 2 in order to gauge the effect of different array partitioning schemes on the Latency/II and resource usage.

This is because an unrolling pragma without a factor would mean a single register read/write implementation for each of the elements of the array.

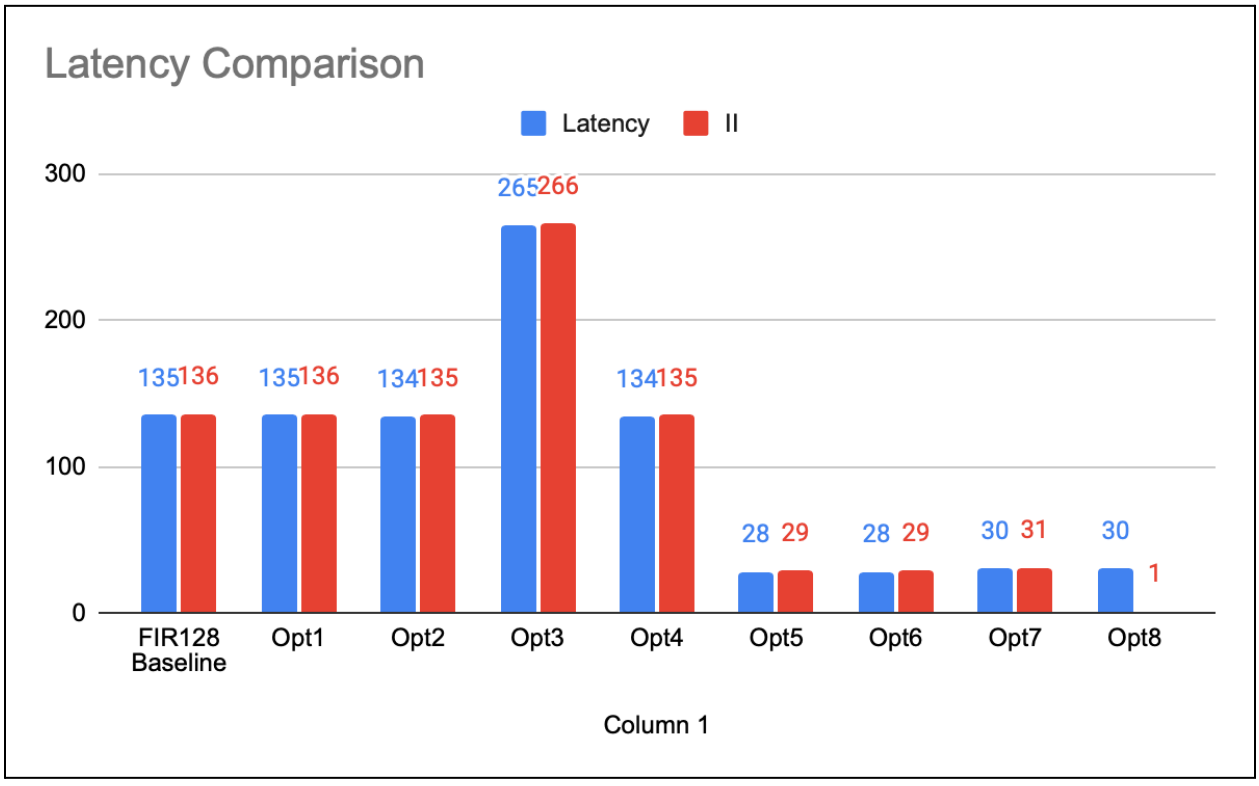
As can be seen, we get the maximum performance out of the Complete partitioning scheme but it also needs the maximum resources for it to be implemented as it needs a single register for each element when compared with block and cyclic where we would constraint our read/write ports and subsequently the throughput (II)

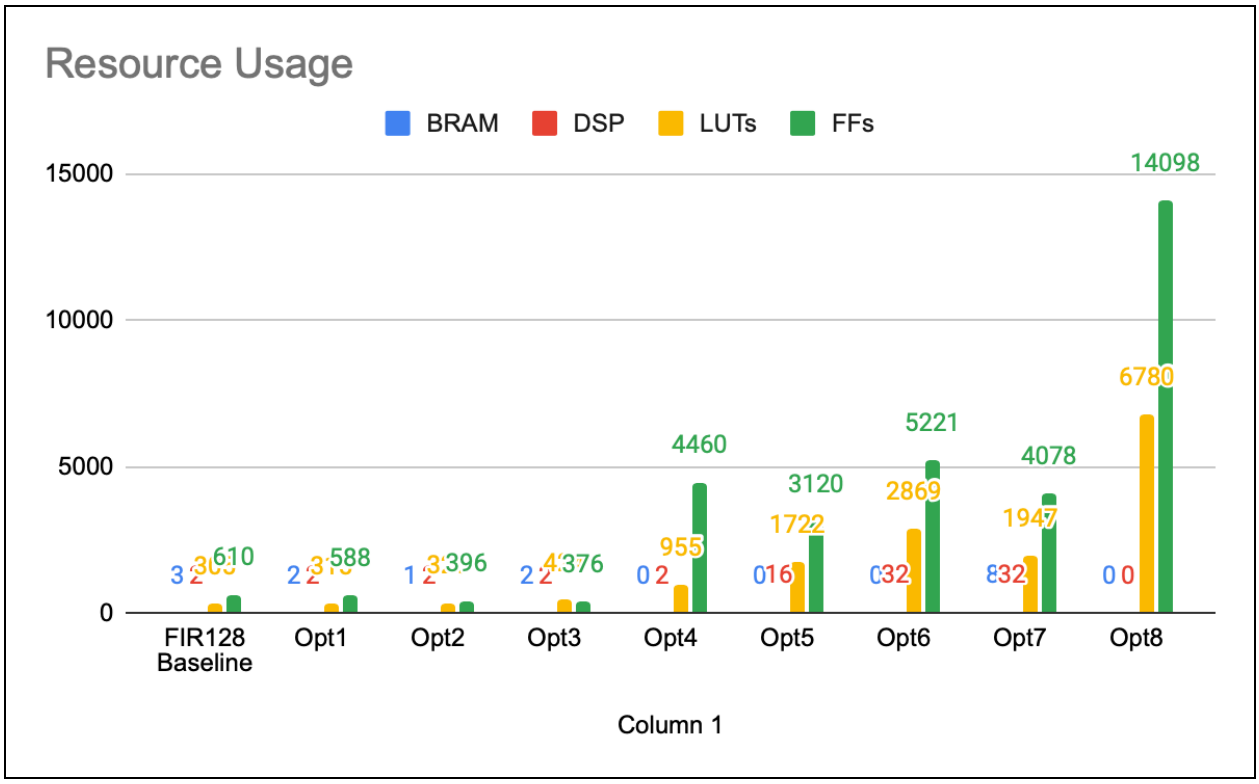
Array Partitioning	Latency	Initiation Interval	Resource Usage			
			BRAM	LUT	FF	DSP
Block	206	207	4	1212	769	4
Cyclic	140	141	4	502	624	4
Complete	139	140	0	6591	4806	4

b. Loop unrolling alone was already done in Q5 without array partitioning (with unrolling factor of 2) with a latency of 282 and II of 283. (So, array partitioning definitely bumps the throughput)

Q7: Best Design

Design Space Exploration





A. It is clearly visible that there is a tradeoff between latency and resource usage. Optimization 7 has maximum throughput (one per cycle) while having enormously high resource usage.

To achieve this result, loop unrolling and array partitioning were applied to enable MAC operations across all filter taps. The computation was fully pipelined to accept a new input every cycle, maximizing throughput. These optimizations eliminate loop dependencies and exploit spatial parallelism at the cost of higher resource utilization.

Latency (cycles)	Initialization Interval (cycles)	Throughput (MHz)
30	1	**100**

BRAM	DSP	LUT	FF
0	0	6780	14098

B. This implementation fully unrolls the computation and partitions the shift register, instantiating separate multipliers, adders, and registers for all N taps. This enables parallel processing of all filter coefficients in a single cycle, significantly improving throughput. However, it also leads to a substantial increase in resource utilization

compared to the sequential baseline, which reuses the same hardware across multiple cycles.