Project 1 — FIR Filter Design Report

author: Jason Yuan
repo: https://github.com/jas0xf/pp4fpgas-project1

Q1 — FIR11 Baseline

(a) Latency, II

METRIC	VALUE (CYCLES)
Latency	**19**
II	**20**

(b) Resourcce

RESOURCE	COUNT
BRAM_18K	**0**
DSP	**2**
LUT	**383**
FF	**733**

Q2 — Variable Bitwidths (FIR128)

(a) Design Variants

VARIANT	LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
Only `coef_t` changed	135	136	1.064	2	2	315	588
Only `data_t` changed	134	135	1.103	2	1	256	295
`coef_t` + `data_t`	134	135	1.103	1	1	256	263

(b) Minimum bitwidths without accuracy loss

```
#include "ap_int.h"
typedef ap_int<5>   coef_t;
typedef ap_int<17>  data_t;
typedef ap_int<16>  acc_t;
```

Q3 — Pipelining (FIR128)

(a) Baseline

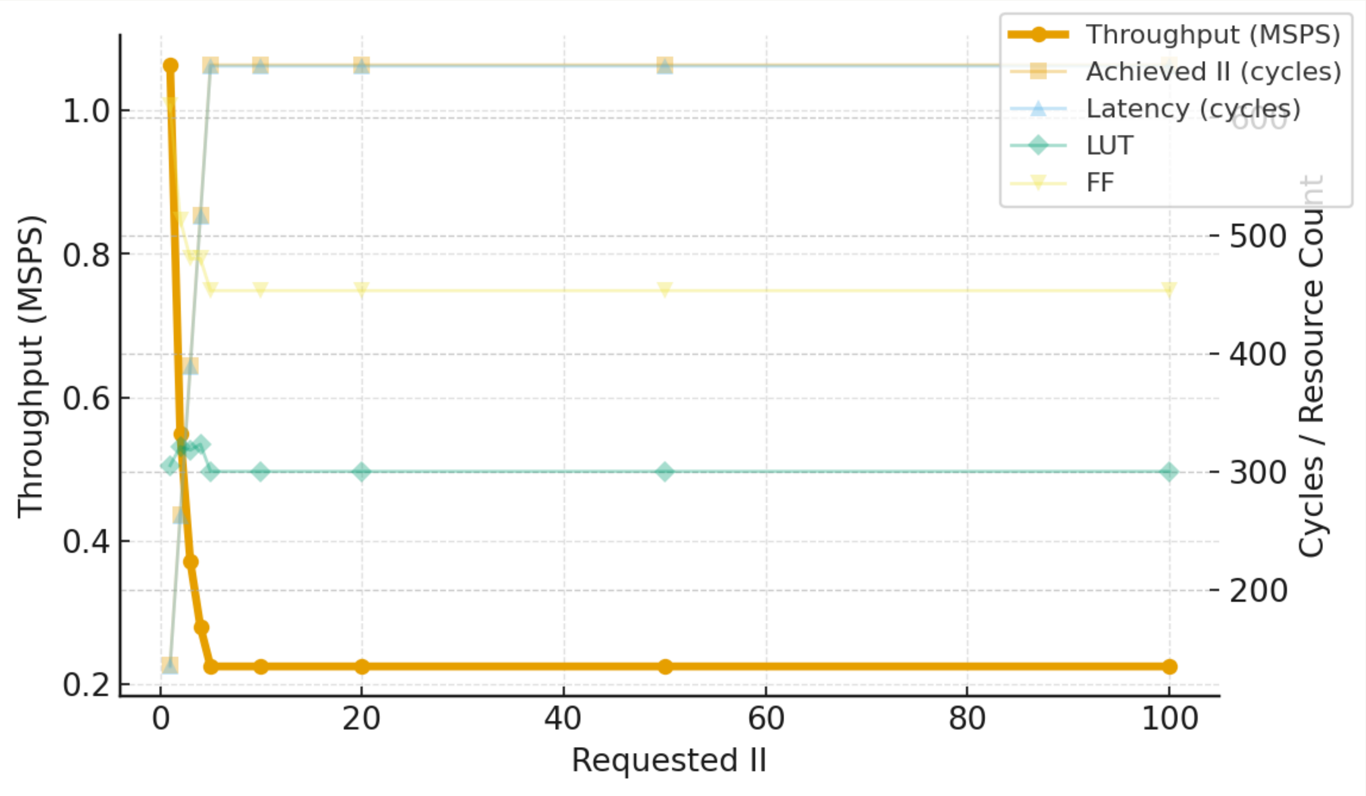
LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
135	136	1.064	3	2	305	610

(b) `#pragma HLS pipeline off`

LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
641	642	0.225	2	2	279	412

(c) Manual(Scripted) pipelining sweep (selected)

REQUESTED II	LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT (MSPS)	BRAM_18K	DSP	LUT	FF
1	135	136	1.064	3	2	305	610
2	262	263	0.550	2	2	321	513
3	389	390	0.371	2	2	318	480
4	516	517	0.280	2	2	323	480
5	643	644	0.225	2	2	300	453
10	643	644	0.225	2	2	300	453
20	643	644	0.225	2	2	300	453
50	643	644	0.225	2	2	300	453
100	643	644	0.225	2	2	300	453



(d) II = 5 (beyond this, throughput stays flat at the non-pipelined level).

(e) 1 is the default II setting (baseline equals to II=1)

Q4 — Removing Conditional Statements (FIR128)

(a) Auto-pipelined

CASE	LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
with condition (baseline)	**135**	**136**	**1.064**	**3**	**2**	**305**	**610**
without condition (code hoisted)	**134**	**135**	**1.072**	**2**	**2**	**316**	**418**

(b) Non-pipelined

CASE	LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
with condition	**641**	**642**	**0.225**	**2**	**2**	**279**	**412**
without condition (code hoisted)	**636**	**637**	**0.227**	**2**	**2**	**270**	**345**

Q5 — Loop Partitioning (FIR128)

(a) Partitioning Idea

Split the original loop into two loops:

1. tapped-delay-line (TDL) shift
2. multiply-accumulate (MAC). This fission exposes loop-level pipeline/unroll

opportunities.

(b) With vs. without loop partitioning (auto settings)

DESIGN	LATENCY (CYCLES)	II (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
baseline (no split)	**135**	**136**	**1.064**	**3**	**2**	**305**	**610**
partitioned	**267**	**268**	**0.540**	**3**	**2**	**343**	**400**

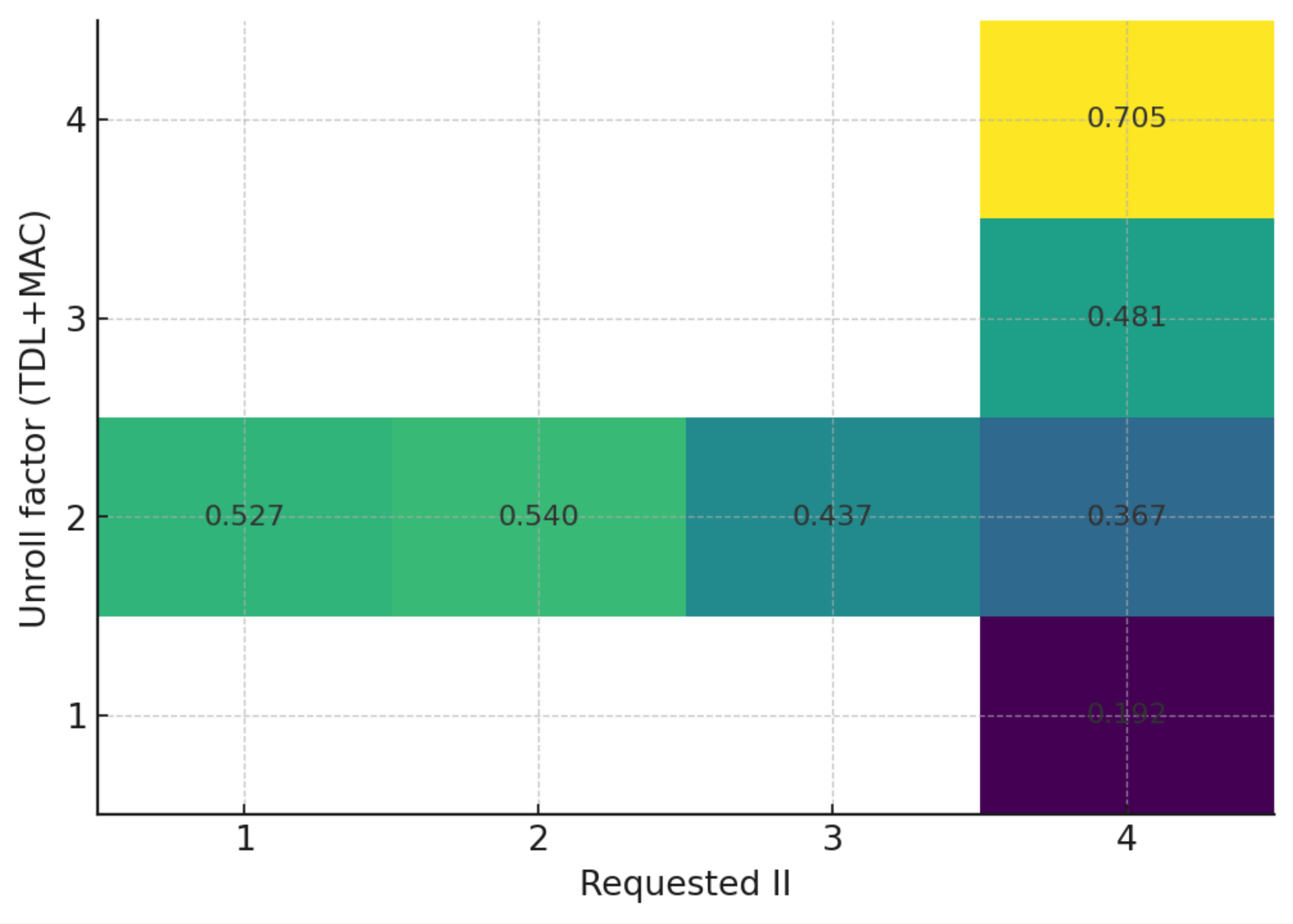
(c) With loop partitioning + unrolling (pipeline target II=1)

TDL UNROLL	MAC UNROLL	UNROLL FACTOR	II (CYCLES)	LATENCY (CYCLES)	THROUGHPUT(MSPS)	BRAM_18K	DSP	LUT	FF
No	No	–	**268**	**267**	**0.540**	**3**	**2**	**343**	**400**
Yes	No	–	**263**	**262**	**0.550**	**3**	**2**	**2,808**	**4,538**
No	Yes	–	**199**	**198**	**0.726**	**2**	**0**	**4,908**	**2,758**
Yes	Yes	–	**127**	**126**	**1.137**	**2**	**0**	**6,823**	**6,348**
Yes	Yes	**4**	**159**	**158**	**0.729**	**2**	**88**	**10,583**	**14,325**
Yes	Yes	**8**	**127**	**126**	**0.9126**	**4**	**160**	**14,738**	**18,551**
Yes	Yes	**16**	**112**	**111**	**1.035**	**0**	**304**	**25,728**	**27,974**
Yes	Yes	**30**	**104**	**103**	**0.931**	**0**	**556**	**52,306**	**53,971**
Yes	Yes	**32**	**105**	**104**	**1.104**	**0**	**592**	**53,184**	**42,568**

(d) — Loop Unrolling × Pipelining

Unrolling duplicates hardware inside an iteration; pipelining overlaps iterations. Used together, unrolling shortens loop-carried spans and exposes parallel work, allowing the pipeline to achieve a **smaller II** and thus higher throughput (≈ *f*clk / II), at the cost of area.

Throughput vs. Unroll × Pipeline (heatmap)



Evidence (TDL+MAC both unrolled)

UNROLL FACTOR	REQUESTED II	II (CYCLES)	THROUGHPUT (MSPS)	DSP	LUT	FF
1	4	776	0.192	1	283	61
2	1	220	**0.527**	50	8,732	11,547
2	2	268	**0.540**	2	488	444
2	3	331	0.437	2	391	129
2	4	394	0.367	2	405	130
3	4	310	0.481	3	480	186
4	4	205	0.705	2	736	599

Conclusion. Increasing the **unroll factor** (at fixed Req II=4) reduces achieved II (776→205) and raises throughput (0.192→0.705 MSPS), with rising LUT/FF (and modest DSP here). Tightening the **pipeline target** (at fixed unroll=2) improves throughput (Req II 4→1: 0.367→0.527 MSPS) when the tool can lower achieved II, but can demand **much more area** (e.g., DSP 2→50 at Req II=1). Thus, **yes**—unrolling and pipelining together clearly benefit performance, bounded by resource budgets and timing closure.

> ***Note on `II=1`. To meet `II=1` the tool often packs more work into each pipeline stage or duplicates hardware to break dependencies, which*** *lengthens the critical path** and* ***increases the estimated clock period*** *(slower Fclk). That's why, in our results, the `unroll=2, req II=1` build achieved a slightly lower II but a* ***worse clock*** *than `req II=2`, yielding* ***lower net throughput*** *despite the more aggressive II target.*

Q6 — Memory Partitioning (FIR128)

(a) Partition options (latency, II, resources)

PARTITION	PART. FACTOR	UNROLL	II (CYCLES)	LATENCY (CYCLES)	EST CLK (NS)	THROUGHPUT (MSPS)	BRAM_18K	DSP	LUT	FF
block	16	16	153	152	7.040	0.928	0	32	12,523	8,329
complete	–	1	266	265	6.912	0.544	0	2	4,023	8,468
cyclic	16	16	31	30	6.923	**4.660**	0	32	2,798	5,172
cyclic	32	1	269	268	6.912	0.538	0	2	2,291	3,615
cyclic	32	32	24	23	6.923	**6.019**	0	60	5,338	10,221
cyclic	64	64	21	20	6.923	**6.878**	0	88	10,455	20,005
cyclic	128	128	31	30	6.923	**4.660**	0	0	4,392	6,288
none	–	not specify	127	126	6.923	**1.137**	2	0	6,823	6,348

Best performance: among the tested points, **cyclic** partitioning paired with **higher unroll** (e.g., F=64 with unroll=64) gives the highest throughput (≈**6.878 MSPS**) but with steep area growth (DSP/LUT).

(b) Disable one knob (effect)

CASE	PARTITION	PART. FACTOR	UNROLL	II (CYCLES)	LATENCY (CYCLES)	EST CLK (NS)	THROUGHPUT (MSPS)	BRAM_18K	DSP	LUT	FF
No unrolling, partition ON	cyclic	32	1	269	268	6.912	**0.538**	0	2	2,291	3,615
Unrolling ON, no partition	none	–	not specify	127	126	6.923	**1.137**	2	0	6,823	6,348

Turning **off unrolling** (keep partitioning) limits parallel reads → **much lower throughput** (0.538 MSPS).

Q7 — Best Design (FIR128)

(a) Best-throughput architecture (what I used)

- Bitwidths from Q2; condition hoisting from Q4; loop fission (TDL + MAC) from Q5
- **Both loops unrolled by 64**, skip_exit_check, **`#pragma HLS pipeline II=1`**, and **cyclic partitioning with factor 64** on `shift_reg[]` and `c[]`

METRIC	VALUE
Estimated clock	6.916 ns
Achieved II	**21 cycles**
Latency	**20 cycles**
Throughput	**6.885 MHz**

(b) Resources (best-throughput design) & why higher than Q2 baseline

DESIGN	BRAM_18K	DSP	LUT	FF
Best (unroll=64, cyclic=64)	**0**	**54**	**5839**	**4149**
Baseline (Q2-style, no unroll/part.)	3	2	305	610

Why higher than Q2 baseline?

- **Massive parallelism from unrolling (×64):** The MAC loop is replicated into ~64 lanes to feed the pipeline (target `II=1`), so multipliers/adders and their control all multiply in count. That's why **DSP jumps 2 → 54 (~27×)** and **LUT/FF rise 305 → 5,839 (~19×) / 610 → 4,149 (~6.8×).** (Not all 64 multiplies mapped to DSPs; some were absorbed into LUTs by the tool/bitwidths.)
- **Memory banking replaces BRAM with logic:** With **cyclic factor = 64** on 128 taps, each bank is only depth-2. HLS implements these tiny banks as **registers/LUTRAM**, so **BRAM drops 3 → 0** while **LUT/FF increase** to provide the ports needed for parallel reads.
- **Extra pipeline/control registers and interconnect:** Deeper/parallel pipelines

add staging registers and wider data paths/fan-out, further inflating **FF/LUT** compared to the serialized baseline.