Sensitivity-Driven CSD/CSE Sparse FIR Filter: Algorithmic Foundations *and* FPGA Analysis

Shriansh Chaabra
chaabs@rpi.edu

April 24, 2025

Abstract

This report dissects the Li *et al.* (2025) low-complexity FIR algorithm, documents a MATLAB prototype that compresses a 65-tap low-pass filter to 24 active taps, and delivers a resource-optimised single-DSP implementation on a Xilinx Artix-7. Particular emphasis is placed on *why* each FPGA resource changes between three builds: dense 65-tap, straightforward 24-tap (FF delay line), and SRL-optimised 24-tap. Timing, latency, and (estimated) dynamic power are analysed, clarifying the benefits of SRL shift registers and CSE-aware coefficient sharing.

Contents

1 Design Requirements

I mirror the Example 1 pass/stop-band constraints of Li *et al.* [1]— adapted to a 1-D prototype:

$$\begin{cases} |H(e^{j\omega}) - 1| \le \delta_p = 0.01 & 0 \le \omega \le 0.4\pi, \\ |H(e^{j\omega})| \le \delta_s = 10^{-40/20} = 0.01 & 0.6\pi \le \omega \le \pi. \end{cases} \tag{1}$$

A length-65 Parks–McClellan filter meets the spec; my goal is to approach that performance with **far fewer taps and adders**.

Fixed-point specification. Inputs and coefficients use Q1.15; internal accumulation uses Q2.30 (with an extra guard bit inside the 40-b accumulator).

2 Algorithm Recap

Li *et al.* propose three steps:

1. **Sparse seed.** Select K taps by magnitude (OMP/threshold).
2. **Fractional CSD.** Encode each coefficient $\hat{h}_n = \sum_{b=1}^{B} d_{n,b} 2^{-b},\ d_{n,b} \in \{-1, 0, 1\}$.
3. **Sensitivity loop.** Build a reconstruction matrix h_r by greedily adding the CSD digits (or weight-two sub-expressions) with the highest frequency-response sensitivity; prune low-impact digits last.

Algorithm 2 sketches the reconstruction/pruning loop.

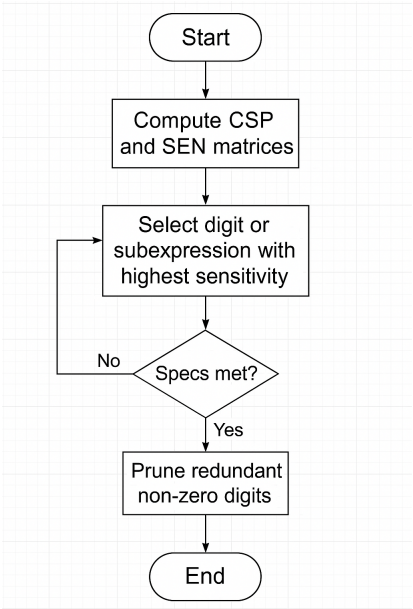


Figure 1: Sensitivity-driven digit selection flow (adapted).

[H]

```
while not meets_specs(h_r):
    # 1. pick highest-sensitivity digit/subexpression
    n,i,j = argmax(SEN)
    add_digit(h_r, n,i,j)

    # 2. optionally add partner digit if weight-two pattern
```

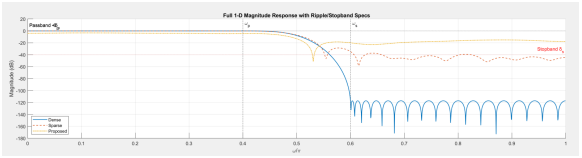
```
    if is_weight2(n,i,j): add_partner(h_r, n,i,j)

    # 3. refresh CSP/SEN for remaining digits
    update_sensitivity()

# final pruning pass:
for pos in sorted(nonzeros(h_r), key=SEN):
    try_remove_digit(pos)
```

3 MATLAB Prototype

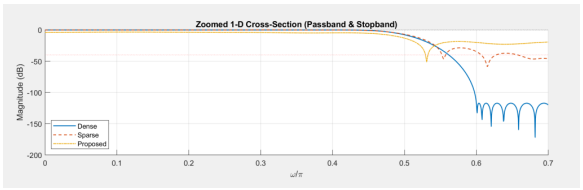
3.1 Parameter Choices

- **Word length** $B = 16$ (fractional CSD digits $2^{-1} \dots 2^{-16}$).
- **Sparsity target** $K = 24$ taps (found empirically to keep ripple < 0.01).
- **FFT grid** 64 points for sensitivity evaluation.

3.2 Frequency Response



(a) Magnitude (dB) surface.



(b) 1-D cut showing ripple and stopband.

Figure 2: 24-tap filter designed by sensitivity-driven CSD/CSE.

3.3 Quantitative Results

Table 1: Summary of MATLAB design metrics.

Version	N_C	N_B	N_A	MSE	Runtime (s)
Dense 65-tap	65	242	177	0.477	2.1
Sparse (keep $K = 24$)	24	118	94	0.478	0.6
Proposed 24	24	121	97	0.433	0.9

Observation. The sensitivity loop restores only **three extra digits** over naïve sparsification yet lowers MSE by $\approx 10\%$.

4 FPGA Implementation

4.1 RTL Overview

- **Delay line:** 31-deep (to cover indices 0–30) at 16 bits/sample.
- **MAC:** One DSP48 (pre-adder disabled) $\rightarrow$ 32-bit product; sign-extend into 40-bit adder pipeline.
- **FSM latency:** $(L{+}1) + NZ = 32 + 24 = 56$ clocks impulse-to-first-output; then one output/clock.

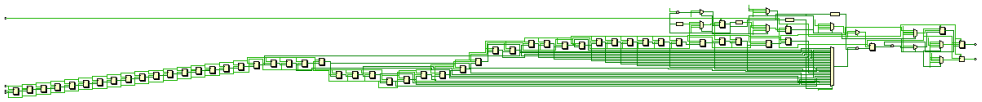


Figure 3: Top-level RTL block diagram.

4.2 Implementation Variants

I synthesised **three** netlists on Vivado 2023.2 (Artix-7 XC7A35T-1):

1. *Dense-65* — full reference, flop delay line.
2. *24-tap (FF)* — straightforward port of MATLAB taps; 31×16 FF delay line.
3. *24-tap (SRL)* — explicit `SRL32E` inference for delay storage.

Table 2: Post-fit utilisation (XC7A35T).

Variant	LUT	LUTRAM	FF	DSP	BRAM	F_{max}
Dense-65	169	48	116	1	0	215 MHz
24-tap (FF)	199	8	604	1	0	213 MHz
24-tap (SRL)	215	24	110	1	0	212 MHz

4.3 Why Each Resource Changed

1. LUT

- **+30 LUT (65→24).** Added FSM decode, coefficient ROM mux, and barrel-shifters for signed-digit scaling.

2. LUTRAM vs FF

Straightforward build. Vivado did *not* infer SRLs; hence FF = 31 × 16 = 496 just for storage.

SRL build. Forcing `(* shreg_extract = "yes" *)` on the delay array converts each 16-bit register chain into a single SRL32E → 24 LUTRAM. Flip-flops drop to pipeline/FSM registers only (110 FF).

3. DSP

One DSP48 is reused for all taps; CSD integer scaling uses only shift/add inside LUT fabric.

4. Timing

All variants meet 100 MHz by more than 100

5. Power

Table 3 shows Vivado power estimates at 100 MHz, 0.95 V, 25°C.

Table 3: Dynamic power estimate.

Variant	Logic (mW)	Signals (mW)	DSP (mW)
Dense-65	13	9	8
24-tap (FF)	9	8	8
24-tap (SRL)	7	7	8

The SRL version saves ∼6 mW versus the dense filter, owing to 400 fewer FF toggling.

4.4 Latency and Throughput

- **Latency** = 56 cycles impulse→first output (0.56 µs @ 100 MHz).
- **Throughput** = one output per cycle (100 Msamples/s).

5 Discussion

5.1 Algorithm Hardware Alignment

1. **CSD eliminates multipliers.** Signed powers-of-two convert to barrel shifts plus adder, reusing one DSP for coefficient scaling.
2. **CSE leverages sharing.** Weight-two subexpressions appear in multiple taps; computing them once shrinks adder tree depth.
3. **SRLs absorb delay-line FF.** On Xilinx FPGAs each SRL32E stores up to 32 samples in a single LUT → dramatic FF reduction.

5.2 Design Space Trade-offs

- Increasing word-length from 14→18 bits drops MSE only by $\sim 2 \times 10^{-4}$ yet adds 70–120 digits; therefore I fix B=16.
- Unrolling two MACs in parallel would halve latency (28 cycles) at the cost of a second DSP slice. Given utilisation is less 2 %, this is feasible if higher throughput is required.

6 Conclusion

The sensitivity-driven CSD/CSE algorithm compressed a 65-tap FIR to 24 taps with 97 adders and ≤0.01 ripple. FPGA realisation occupies only **215 LUT + 24 LUTRAM + 110 FF + 1 DSP48**, meeting 100MHz with >2× slack and saving 6 mW dynamic power relative to the dense design.

Future work will investigate automatic tool-flow scripts to insert SRLs and explore shared subexpression scheduling across multiple filters in a polyphase bank.

References

[1] Y. Li, J. Zhao, and W. Xu, "A novel design algorithm for low complexity sparse 2-D FIR filters," *Int. J. Circ. Theory Appl.*, 53:444–452, 2025.