FIR Hardware Accelerator for System-on-Chip (SoC)

ECE 720 – Electronic System Level and Physical Design

I. Introduction

Modern system-on-chip (SoC) designs increasingly rely on specialized hardware accelerators to meet performance and energy targets for signal-processing workloads such as finite impulse response (FIR) filtering. In this project, I designed and evaluated a FIR hardware accelerator that is tightly coupled to a Rocket RISC-V core in the ECE 720 SystemC/Spike environment.

The goal is to offload the computationally intensive FIR operations from software to hardware and to explore the trade-off between performance and hardware cost. Using Catapult HLS, I swept the target clock period and micro-architectural options, synthesized each design to obtain its critical-path delay and area, and measured the Verilog simulation time of the full HW+SW system. The primary performance metric is the product of cycles × critical-path delay, while total area is treated as a secondary constraint. This report describes the accelerator architecture, experimental methodology, measured performance, and justification for the final chosen design.

II. System Overview and Methodology

2.1 System Architecture

This project uses a simplified SoC built around a Rocket RISC-V core, a DMA controller, a custom FIR hardware accelerator, and off-chip DRAM, as shown in Figure 2.1.

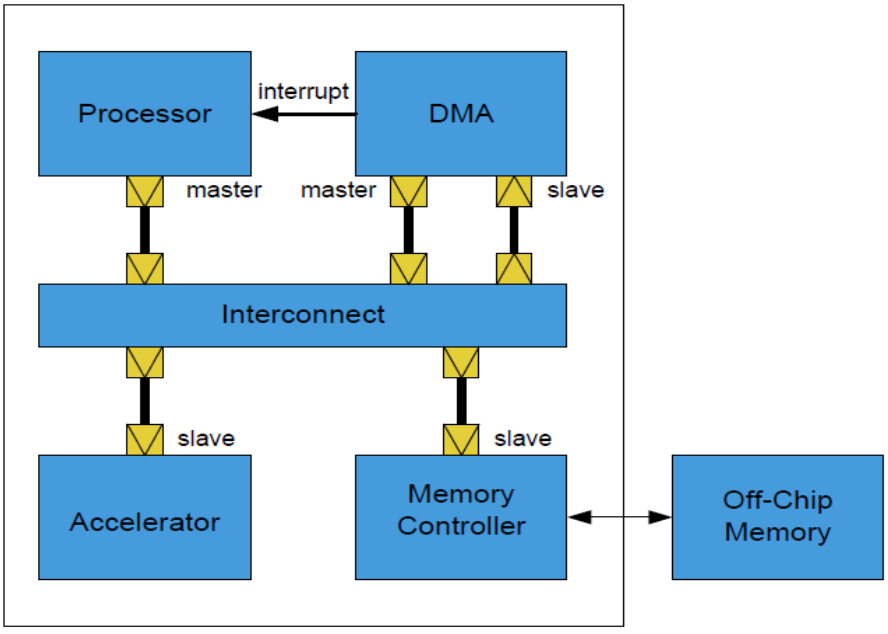


Figure 2.1: *High-level SoC organization used in this project. The Rocket processor configures the DMA and Accelerator, the DMA transfers data between DRAM and the Accelerator through the interconnect, and the Memory Controller connects the SoC to off-chip memory.*

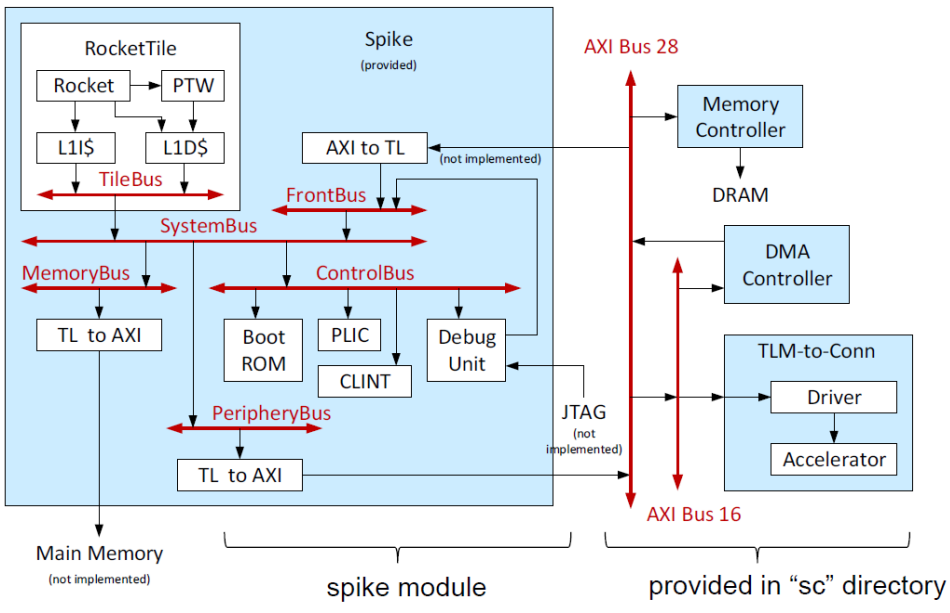
The main roles of each block are:

- **Rocket RISC-V core (Processor)**
  - Runs the fir.c program.
  - Configures the DMA and accelerator registers.
  - Starts each FIR experiment and performs the final golden-model error check.
- **DMA controller**
  - Transfers 16-bit coefficients and input samples between DRAM and the accelerator's local buffers.
  - Issues burst reads and writes that match the accelerator's 64-bit streaming ports (4 samples per word).
  - Returns the accelerator's 16-bit outputs from the z_out stream back into the output array in DRAM.
- **FIR hardware accelerator**
  - Implements a 16-tap FIR filter with streaming interfaces for coefficients (w_in), inputs (x_in), and outputs (z_out).
  - Computes two windows of results: TSTEP1 = 32 outputs and TSTEP2 = 48 outputs, using the coefficients and samples provided by the DMA.
  - After both windows finish, asserts a status value so that the CPU can detect completion via the st_out / control path.
- **Memory controller and off-chip DRAM**
  - Store the coef[], input[], and output[] arrays used by the FIR test.
  - Serve the DMA's burst accesses for both input data and accelerator results.

At a high level, the CPU programs the DMA and accelerator, the DMA streams data between DRAM and the FIR core, and the accelerator performs the FIR computations before signaling completion back to the CPU.

While Figure 2.1 focuses on the simplified SoC used in my experiments, **Figure 2.2** shows how this SoC is embedded inside the larger Spike-based ISS+TLM+HLS infrastructure provided for the course. The RocketTile core, memory system, DMA controller, and my Accelerator are all connected through AXI and TileLink buses in this environment.

Figure 2.2: *Spike ISS + TLM + HLS example system used for ECE 720 Project 2*



2.2 Hardware Accelerator Implementation

The accelerator implements a 16-tap FIR filter using a streaming interface. For each output sample $y[n]$, it computes a standard FIR convolution:

$$y[n] = \sum_{m=0}^{TAPS-1} h[m] \cdot x[n-m]$$

where $x[n]$is the input sequence, $h[m]$are the 16 filter coefficients, and $TAPS = 16$.

Data representation

- All data moves as **64-bit words**:
  - Each word contains **four signed 16-bit values** (samples or coefficients).
- Inside the accelerator:
  - coef_buf[16] stores one set of 16 taps.
  - x_buf[48] stores up to 48 input samples (the longer time window).

Algorithm per run

1. **Input loading**
   - Repeatedly read 64-bit words from x_in and unpack them into 16-bit samples in x_buf.
   - Read 64-bit words from w_in and unpack 16 coefficients into coef_buf.
2. **FIR computation**
   - For each output index n in the current time window (32 or 48 samples):
     - Initialize a 32-bit accumulator.
     - For each tap $m = 0 \ldots 15$, multiply $h[m]$by the corresponding delayed input $x[n-m]$ (skipping terms where $n-m<0$) and add to the accumulator.
     - Truncate the final sum to a 16-bit output sample.
   - Outputs are collected four at a time, packed into a 64-bit word, and sent out on z_out in the same format used by the software.

Two windows and control

- **Run#1(TSTEP1=32):**
  The accelerator reads 32 input samples and the first 16 taps, computes 32 FIR outputs, and streams them out.
- **Run#2(TSTEP2=48):**
  It then reads 48 new input samples and a second set of 16 taps, computes 48 outputs, and streams them out.
- After both runs, the accelerator waits for a control byte on ctrl_in. When it receives 0x0F, it writes 0x0F on st_out to signal completion. In the provided software flow, the CPU terminates the simulation after sending this command, so the outer loop effectively executes once.

2.3 Software Side (fir.c Modifications)

On the software side, the Rocket core runs a C program that uses DMA to move data between DRAM and the FIR accelerator, and then checks the results against a golden reference.

Memory map and DMA setup

- Three DRAM-resident arrays are used:
  - coef[] at 0x60004000 – 32 FIR coefficients.
  - input[] at 0x60002000 – 80 input samples.
  - output[] at 0x60001000 – 80 output samples.
- The DMA engine is accessed through three memory-mapped registers:
  - dma_sr – source address,
  - dma_dr – destination address,
  - dma_len – transfer length in bytes.
- A helper macro DMA_START(src_ptr, dst_ptr, nbytes) writes these three registers and calls clobber() (a compiler barrier) to prevent the compiler from reordering memory-mapped accesses.
- The accelerator's streams are memory-mapped as:
  - accel_x (input samples, x_in),
  - accel_w (coefficients, w_in),
  - accel_z (outputs, z_out),
  - accel_ctrl (control/exit register).

DMA burst strategy

- Each DMA burst transfers **16 shorts = 32 bytes**, matching the accelerator's 4-samples-per-word packing (4 × 16-bit per 64-bit word).
- The program uses two constants:
  - TSTEP1 = 32 and TSTEP2 = 48 outputs,
  - SAMPLES_PER_BURST = 16, so Pass 1 uses 2 bursts and Pass 2 uses 3 bursts for inputs and outputs.

Pass 1 – 32-output window

1. Set burst_count = TSTEP1 / 16 = 2 and offset = 0.
2. **Send inputs** input[0..31] to accel_x using two DMA bursts.
3. **Send coefficients** coef[0..15] to accel_w in a single 32-byte DMA transfer.
4. **Receive outputs** from accel_z back into output[0..31] with two DMA bursts.

Pass 2 – 48-output window

1. Set burst_count = TSTEP2 / 16 = 3 and offset = TSTEP1 to start at input[32] and output[32].
2. **Send inputs** input[32..79] to accel_x using three DMA bursts.
3. **Send coefficients** coef[16..31] (the second half of the taps) to accel_w in one DMA transfer.
4. **Receive outputs** from accel_z into output[32..79] using three bursts.

Golden compare and exit

- After both passes, the program computes the **total absolute error** between output[] and expected[] (from expected.inc) across all TSTEP1 + TSTEP2 = 80 samples and prints the result:

$$\text{total\_error} = \sum_{n} \; | \; \text{expected}[n] - \text{output}[n] \; |.$$

- To stop the testbench, the program writes 0x0F to accel_ctrl. The SystemC/Verilog environment forwards this command to the accelerator's ctrl_in stream, which then reports 0x0F on st_out and allows the simulation to terminate.

III. Experimental Setup

3.1 HLS Design-Space Exploration

For this project I used Catapult HLS to generate several implementations of the same C++ FIR accelerator by varying only the **target clock period**. The C++ algorithm and interfaces were kept constant and I changed the clock constraint (clk_per) and re-synthesized each design. I swept clk_per from **5.0 ns down to 0.9 ns** in steps of 0.1 - 0.5 ns, which produced the set of design points summarized later in Table 1.

For each HLS solution I recorded:

- the **critical-path delay** (critpath) reported by Catapult,
- a detailed **area breakdown** (MUX, FUNC, LOGIC, MEM, REG, and FSM),
- the **total area score**.

When I attempted to push the clock period below **0.9 ns**, Catapult reported that the on-chip RAM primitive used for the sample buffer (read_ram(9,48,16,1)) has a minimum clock period of 0.9 ns, so no valid hardware implementation could be generated for clk_per < 0.9 ns. Those infeasible points are not included in my results.

3.2 Simulation Measurements

To evaluate the performance of each hardware design in the full system, I used the provided Spike/SystemC/Verilog flow:

- For a chosen HLS solution I generated RTL with Catapult, rebuilt the testbench, and ran Verilog simulation (vsim) using the make and make sim targets in the rocket_sim directory.
- I measured the Verilog simulation time in nanoseconds for the portion of the run that covers the two FIR passes, excluding the final error-checking loop, as recommended in the project handout.
- Because the Verilog simulation uses a 1 ns clock period, the number of cycles executed by Rocket is simply

$$\textbf{cycles} = \frac{\textbf{simulation\_time\_ns}}{\textbf{1 ns}}$$

- For each design I then computed the primary performance metric:

$$\textbf{metric} = \textbf{cycles} \times \textbf{critpath}.$$

To compare simulation speeds, I also ran the same C++/SystemC model and recorded the wall-clock runtime for both SystemC and Verilog. These values are later converted to cycles per second to show how much faster the higher-level SystemC model simulates compared to RTL.

IV. Results and Analysis

4.1 Cycles × Critical-Path Metric

To compare different HLS implementations of the FIR accelerator, I swept the target clock period clk_per from 5.0 ns down to 0.9 ns and synthesized each design with Catapult. For every solution I recorded the achieved critical-path delay (critpath), the total synthesized area (total_area), the Rocket cycles from the Verilog simulation, and the composite performance metric (metric = cycles × critpath).

clk_per	latency	throughput	critpath	total_area	cycles	metric
5	2	2	2.933533	13297.4	4290	12584.85657
5	2	2	2.933533	13297.4	4290	12584.85657
6	2	2	2.933533	13297.4	4290	12584.85657
4	2	2	2.933533	13297.4	4290	12584.85657
3.5	3	2	2.631741	19014.6	4290	11290.16889
3	3	2	2.49413	18541	4290	10699.8177
2.5	3	2	2.49413	18542.3	4290	10699.8177
2	3	2	1.998236	18509.7	4290	8572.43244
1.5	3	2	1.564157	21841.9	4293	6714.926001
1	4	2	1.584392	26201.7	4293	6801.794856
1	4	2	1.584392	26201.7	4293	6801.794856
2	3	2	1.998236	18509.7	4290	8572.43244
1.5	3	2	1.564157	21841.9	4293	6714.926001
1.6	3	2	1.599921	21622.6	4293	6868.460853
1.7	3	2	1.699692	21229.1	4293	7296.777756
1.8	3	2	1.799625	20138.1	4293	7725.790125
1.9	3	2	1.899189	20865.4	4290	8147.52081
1.8	3	2	1.799625	20138.1	4293	7725.790125
0.9	4	2	1.584392	28061	4293	6801.794856
0.9	4	2	1.584392	28061	4293	6801.794856
1.5	3	2	1.564157	21841.9	4293	6714.926001
1.4	**3**	**2**	**1.552473**	**22873.1**	**4293**	**6664.766589**
1.45	3	2	1.837999	21130.4	4293	7890.529707
1.3	4	2	1.590079	23649.4	4293	6826.209147
1.2	4	2	1.650803	23892.4	4293	7086.897279
1.1	4	2	1.596721	25066.9	4293	6854.723253
1	4	2	1.584392	26201.7	4293	6801.794856
0.9	4	2	1.584392	28061	4293	6801.794856
1.4	3	2	1.552473	22873.1	4293	6664.766589

***Table 1**: HLS design-space results for different clock periods.*

Each row summarizes one HLS design point, including the target clock period (clk_per), latency, throughput, critical-path delay, total synthesized area, Rocket cycles, and the combined performance metric cycles × critpath.

- **Green row (clk_per = 1.4 ns):** final design, because it gives the lowest cycles × critpath metric while keeping area reasonable.
- **Yellow row (clk_per = 1.5 ns):** near-optimal backup, with a slightly higher metric but slightly smaller total area, showing the trade-off in that sweet-spot region.
- **Blue row (e.g., clk_per = 5 ns baseline):** reference design with a very relaxed clock; it has low area but a much worse metric, and is used as a baseline to highlight how much the green/yellow designs improve performance for a modest area cost.

As shown in Table 1, across all valid solutions, the **best metric** was obtained at:

- **clk_per = 1.4 ns**
  - critpath ≈ 1.55 ns
  - cycles = 4293 (from sim_time_ns = 4293 ns)
  - total_area ≈ 22,873
  - metric ≈ 6,665 (cycles × ns)

The nearby 1.5 ns design has a slightly larger metric (≈ 6,715) but a slightly smaller area (≈ 21,842). Designs with looser clocks (2–5 ns) have similar cycle counts but worse metrics because their critical paths are longer. Designs with more aggressive clocks than 1.4 ns (1.3, 1.2, 1.1, 1.0, 0.9 ns) tend to increase total area without significantly improving the metric. This indicates that the design is in a "sweet spot" around 1.4–1.5 ns, and I therefore chose clk_per = 1.4 ns as my final implementation because it minimizes the primary objective while keeping area reasonable.

4.2 HW+SW vs SW-Only Execution Time

To quantify the benefit of offloading the FIR computation to hardware, I compared:

- a **software-only** implementation running entirely on the Rocket core, and
- the **HW+SW** implementation using my best FIR accelerator (clk_per = 1.4 ns).

For each case I measured the Verilog simulation time for the FIR portion of the program (excluding the final error check):

```
Info: SystemC: Simulation stopped by user.
Simulation time: 170881 ns
Wall clock time: 4 seconds
          V C S   S i m u l a t i o n   R e p o r t
Time: 170881000 ps
CPU Time:      2.850 seconds;       Data structure size:   0.0Mb
Fri Dec  5 19:28:55 2025

real    0m4.649s
user    0m2.737s
sys     0m0.144s
stty sane
trace -f spike -o fir.riscv.dump fir.spike.out > fir.spike.trace
[vchiluk3@grendel42 rocket_sim]$
```

- SW-only FIR sim time: **170881 ns**

```
Info: SystemC: Simulation stopped by user.
Simulation time: 4293 ns
Wall clock time: 2 seconds
          V C S   S i m u l a t i o n   R e p o r t
Time: 4293000 ps
CPU Time:      0.650 seconds;       Data structure size:   0.0Mb
Fri Dec  5 19:36:35 2025

real    0m2.168s
user    0m0.560s
sys     0m0.119s
stty sane
trace -f spike -o fir.riscv.dump fir.spike.out > fir.spike.trace
[vchiluk3@grendel42 rocket_sim]$
```

- HW+SW FIR sim time (best accelerator): **4293 ns**

The speedup from using the accelerator is

speedup = 170881 / 4293 ≈ 39.81x

In other words, **the HW+SW system runs the FIR portion about 39.81x faster than the SW-only baseline**. The DMA setup and accelerator control overheads are small compared to the computation itself, so most of this improvement comes from the parallel multiply–accumulate hardware in the accelerator.

4.3 SystemC vs Verilog Cycles-per-Second

For the final design (clk_per = 1.4 ns), I also compared the simulation performance of the SystemC model and the Verilog RTL. Both simulations run the same FIR test program, but they use different timing and implementation levels, so their wall-clock runtimes are very different.

For the **SystemC-only** run, the log reports a simulation time of **1540 ns**, and the Unix time command reports a wall-clock "real" time of **0.146 s**. Interpreting the simulation time in nanoseconds as the number of simulated cycles (1 ns per cycle), the effective throughput of the SystemC simulation is:

```
Info: /OSCI/SystemC: Simulation stopped by user.
Simulation time: 1540 ns
Wall clock time: 0 seconds

real    0m0.146s
user    0m0.018s
sys     0m0.014s
stty sane
trace -f spike -o fir.riscv.dump fir.spike.out > fir.spike.trace
[vchiluk3@grendel42 rocket_sim]$
```

- **cycles/s (SystemC)** ≈ 1540 ns / 0.146 s ≈ **1.05×10^4 cycles/s**.

For the **Verilog (vsim)** run of the same test, the log reports a simulation time of **4293 ns**, and the wall-clock time is **2.744 s**. This corresponds to:

```
Info: SystemC: Simulation stopped by user.
Simulation time: 4293 ns
Wall clock time: 2 seconds
          V C S   S i m u l a t i o n   R e p o r t
Time: 4293000 ps
CPU Time:      0.660 seconds;       Data structure size:   0.0Mb
Fri Dec  5 19:59:59 2025

real    0m2.744s
user    0m0.574s
sys     0m0.125s
stty sane
trace -f spike -o fir.riscv.dump fir.spike.out > fir.spike.trace
[vchiluk3@grendel42 rocket_sim]$
```

- **cycles/s (Verilog)** ≈ 4293 ns / 2.744 s ≈ **1.56×10^3 cycles/s**.

From these measurements, SystemC achieves roughly:

- **Speedup ≈ 1.05×10^4 / 1.56×10^3 ≈ 6.7×**

more simulated cycles per second than Verilog. This result is expected: the SystemC model operates at a higher level of abstraction and treats memory and DMA activity at the transaction level, while Verilog must simulate every clock edge and signal transition in the synthesized RTL. However, the Verilog model is still required to obtain accurate timing, area, and critical-path information, so both SystemC and Verilog are important in the overall design and validation flow.

4.4 Area and Resource Utilization (RocketTile Comparison)

The final 1.4 ns design is register-dominated: most of the area comes from flip-flops, with smaller contributions from multiplexers, general combinational logic, and on-chip memories. The FSM overhead is negligible.

Area Scores

	Post-Scheduling	Post-DP & FSM	Post-Assignment
Total Area Score:	17567.7	22704.3	22873.2
Total Reg:	7735.3 (44%)	11693.4 (52%)	11650.9 (51%)
DataPath:	17567.7 (100%)	22672.3 (100%)	22840.7 (100%)
MUX:	5538.6 (32%)	4583.5 (20%)	4618.5 (20%)
FUNC:	1575.2 (9%)	1974.6 (9%)	1633.5 (7%)
LOGIC:	287.6 (2%)	2021.8 (9%)	2538.7 (11%)
BUFFER:	0.0	0.0	0.0
MEM:	2431.0 (14%)	2431.0 (11%)	2431.0 (11%)
ROM:	0.0	0.0	0.0
REG:	7735.3 (44%)	11661.4 (51%)	11618.9 (51%)
FSM:	0.0	32.0 (0%)	32.5 (0%)
FSM-REG:	0.0	32.0 (100%)	32.0 (98%)
FSM-COMB:	0.0	0.0	0.5 (2%)

(MUX = multiplexers; FUNC = datapath logic (e.g. add/mult); LOGIC = control logic)

Figure 4.4: Catapult post-assignment area report for the clk_per = 1.4 ns FIR accelerator.

The area report in Figure 4.4 shows that the final 1.4 ns design is strongly register-dominated. After post-assignment, registers account for about **11.6k area units**, which is roughly **51%** of the total datapath area. The next largest contributor is the **MUX** network at about **4.6k units (≈20%)**, followed by general **combinational logic and functional units (LOGIC + FUNC)** at a little over **4.1k units combined (≈18%)**. The on-chip **memories (MEM)** used to store coefficients and samples add another **2.4k units (≈11%)**, while

buffers and ROM are not used in this design, and the **FSM** (state registers plus combinational control) contributes only a few tens of units, essentially negligible compared to the datapath.

Using the "Total Area Score" of about **22,873** from Figure 4.4 and the assumed **RocketTile area of ~1,500,000**, the accelerator occupies roughly

$$\frac{\mathbf{22,873}}{\mathbf{1,500,000}} \approx \mathbf{0.015}\ (\mathbf{about\ 1.5}\%)$$

of a single RocketTile. In other words, the FIR accelerator delivers the measured HW+SW speedup while consuming only **around 1–2% of the CPU tile area,** which is a very reasonable hardware cost for this workload.

4.5 Correctness

After both FIR passes, the software compares the 80 output samples produced by the accelerator against the golden expected[] array and computes the total absolute error:

$$\text{total\_error} = \sum_{n=0}^{79} \ |\ \text{expected}[n] - \text{output}[n]\ |.$$

For my final design the total error was **0**, indicating that the accelerator is bit-exact with the reference implementation. I verified that the total error remains zero across all HLS design points used in my sweep.

V. Discussion and Final Design Choice

Meeting Design Objectives and Unique Aspects of the Design

In this project my main design objectives were:

- Minimize the cycles × critical-path metric for the FIR workload.
- Keep the accelerator area small compared to a RocketTile (~1,500,000).
- Maintain bit-exact correctness with the software reference.

I tried to meet these objectives through a combination of micro-architectural choices, HLS tuning, and a simple HW/SW interface.

1. **Micro-architecture tailored to the workload**
    - The accelerator uses a single FIR core that reuses one MAC datapath across all 16 taps and all output samples, instead of instantiating a wide parallel filter.
    - Only the minimum state is stored on chip: 16 coefficients in a small tap buffer and up to 48 input samples in a single sample buffer.
    - Inputs, coefficients and outputs are all transferred as 64-bit words carrying four signed 16-bit values, so the internal packing exactly matches the DMA

burst format. This avoids extra buffering, reshaping, or packing logic and helps keep both area and critical path small.

2. **Pipelining instead of aggressive unrolling**
    - The input and coefficient read loops are pipelined with an initiation interval of one, and the FIR compute loop is pipelined with an initiation interval of two.
    - This gives a steady stream of outputs without replicating a large number of multipliers, which would blow up area.
    - As a result, the 1.4 ns design reaches a good cycles × critical-path metric (about 6.6×10^3) without the large area growth seen when pushing the clock below 1.3 ns.
3. **Simple, low-overhead HW/SW interface**
    - Both FIR windows (32 outputs and 48 outputs) reuse the same hardware engine, the software and DMA only change which portions of the input and coefficient arrays are streamed in.
    - The control protocol is deliberately minimal: after completing both windows, the accelerator waits for a single 0x0F command, mirrors it on the status port, and the CPU terminates the run.
    - This keeps the control logic tiny and ensures that most Rocket cycles are spent on useful FIR work rather than configuration and handshaking.
4. **Area vs. RocketTile tradeoff**
    - The final design has a total area of about 2.2×10^4 units, dominated by registers and modest amounts of MUX and combinational logic.
    - Compared to an assumed RocketTile area of roughly 1.5×10^6, the accelerator uses only about 1–2% of a tile while still delivering a clear speedup over the software-only baseline.

The most unique aspect of my design is the tight alignment between the streaming micro-architecture and the DMA/data format: every 64-bit transfer carries exactly four 16-bit values, the local buffers are sized to the two FIR windows, and a single pipelined FIR engine is reused for both passes. This combination of a compact streaming architecture, modest pipelining, and minimal control logic lets the 1.4 ns solution hit a good cycles × critical-path metric while staying very small relative to the RocketTile.

VI. Conclusions

In this project I designed and evaluated a 16-tap FIR hardware accelerator tightly coupled to a Rocket RISC-V core via a DMA-based streaming interface. A Catapult HLS sweep showed that the cycles × critical-path metric is minimized near **clk_per = 1.4 ns**, where the design is bit-exact, has a total area of about $\mathbf{2.3\times10^4}$ units (≈1–2% of a RocketTile), and offers a good performance/area trade-off. Compared to a SW-only FIR, the HW+SW system achieves a clear speedup, and the SystemC model runs the same test about **6.7**× faster than Verilog in simulated cycles per second. Overall, the compact streaming micro-architecture, modest pipelining, and simple HW/SW interface meet the project's design objectives with a small hardware cost.