Real-Time DSP on NI PXIe-7972R (Kintex-7) + NI-5782R

A Reproducible LabVIEW FPGA Report and Handoff Guide
With Worked Examples: Look-Ahead Biquad and Phase Estimation

Arash Ganjei

October 23, 2025

Abstract

This report documents a reproducible FPGA/host workflow for two real-time DSP tasks on NI PXIe-7972R (Kintex-7) with NI-5782R I/O: (i) a k=1 look-ahead biquad that sustains a 125 MHz Single-Cycle Timed Loop (SCTL), and (ii) a per-pulse weighted phase estimator driven by precomputed coefficients. On the FPGA, all multiplies map explicitly to DSP48E1 slices and sums use a balanced adder tree with a *uniform* one-tick pipeline on every branch, yielding the expected pure z^{-1} delay. Timeseries are streamed via DMA, and all fixed-point signals are carried as Q2.13 with a single final cast before egress.

Estimator weights are generated offline in MATLAB (two vectors, N=6250 each), exported as CSV, quantized on the host to Q4.14 (18-bit word, 4-bit integer), and transferred over two Host→TargetDMA FIFOs into dual-clock BRAM on the FPGA. A fabric-clock loader performs blocking DMA reads and writes BRAM sequentially; a one-bit `Weights_Ready` gate releases the 125 MHz SCTL only after all 12,500 values are valid. The host VI binds the bitfile, performs large-chunk DMA reads, scales fixed-point to `DBL`, builds waveforms with dt=8 ns, and provides low-rate controls (e.g., stimulus settings, decimation).

Validation shows the filter matches a fixed-point MATLAB reference with relative RMS $\mathbf{8.127 \times 10^{-5}}$ (NMSE $\mathbf{-81.80}$ dB) and only the intended +1-sample delay; Welch PSDs overlap across 1–60 MHz. The phase estimator achieves a steady-window error of $\mathbf{2.729 \times 10^{-3}}$ rad while preserving index-exact parity in per-pulse summaries. We include figures of the actual VIs, a clear register/stream map, and precise data types so future users can rebuild and extend the system without guesswork. (Terminology such as "homodyne slope" and "SCTL pipelining" is defined in §1.2.)

Contents

List of Figures

Chapter 1

Introduction

1.1 Question and Success Criteria

Q1 (Filter). Can a look-ahead biquad meet 125 MHz SCTL timing on PXIe-7972R with NI-5782R while preserving dynamics (only a pure delay) and achieving fixed-point accuracy better than -80 dB NMSE?
Q2 (Phase). Can a weighted per-pulse phase estimator run in real time and match a MATLAB reference within a few 10^{-3} rad RMS?

Why these thresholds (rationale). The -80 dB NMSE target for the filter is chosen to be comfortably above the expected fixed-point noise floor for Q2.13. With $\Delta = 2^{-13}$ and quantization noise power $\Delta^2/12$, a unit-variance signal yields an ideal floor near -90 dB; allowing headroom for coefficient rounding, occasional saturation margins, and I/O scaling, a practical criterion of ≤ -80 dB ensures the hardware error is dominated by quantization rather than algorithmic defects. For the estimator, a few$\times 10^{-3}$ rad RMS matches the tolerance used in our MATLAB studies for per-pulse parity and is tight enough to expose indexing or normalization mistakes while remaining achievable with Q4.14$\times$Q2.13 products at 125 MHz.

1.2 Contributions

- A reproducible *project skeleton* (controller, FPGA target, 5782R Multi-Sample CLIP, DMA streams, host VI).
- A look-ahead biquad with uniform 1-cycle pipeline on *all* branches, multiplies on DSP48E1, balanced adder tree, single final cast to Q2.13.
- A phase-estimation pattern: host-generated weights $\rightarrow$ Host$\rightarrow$Target DMA $\rightarrow$ BRAM $\rightarrow$ SCTL MACs (legal clocking with loader handshake).
- Quantitative validation against a fixed-point reference (filter) and per-pulse metrics (phase).

Terminology used in this report

SCTL pipelining. Split long arithmetic into one-tick stages at 125 MHz; with uniform 1-cycle latency on all branches, the effect is a pure z^{-1} delay.

Homodyne slope. Small-signal $\partial V/\partial\phi$ (V/rad) near the operating point; used to normalize estimator gains.

Fixed-point notation. Signals are Q2.13; weights are typically Q4.14 (18-bit word, 4 integer bits).

Chapter 2

Platform Overview & Quickstart

This chapter gives the *big picture*: what runs where, how data moves, what artifacts exist (bitfiles, FIFOs, memories), and how you go from "fresh clone" to "plots on screen." Concrete click-by-click setup, control names, and data types live in Chapter 3; FPGA internals are in Chapters 5 and 6.

2.1 System at a glance

- **Hardware.** NI PXIe-7972R (Kintex-7) carrier with NI-5782R I/O. The 5782R exposes one 125 MHz I/O Module Clock that sample-locks FPGA math to AI/AO.
- **Host.** A single Host VI opens the FPGA reference, pushes configuration (low-rate registers, one-shot weight loads), and pulls timeseries via DMA for plotting/logging.
- **Bitfiles.** Two builds: `Filter_LA_Biquad.lvbitx` and `Phase_Estimator.lvbitx`. Each has its own top-level VI under the FPGA Target.
- **Streams.** Target→HostDMAs for timeseries; Host→TargetDMAs for one-time tables (e.g., estimator weights).

2.2 Roles and responsibilities

FPGA filter bitfile. Runs a k=1 look-ahead biquad inside a 125 MHz Single-Cycle Timed Loop (SCTL). All multiplies land on DSP48E1 slices; a balanced adder tree plus *uniform* one-tick pipeline yields a pure z^{-1} delay. Generates on-card stimuli (square/saw/sine), optionally decimates *streams-to-host*, and never blocks.

FPGA phase-estimator bitfile. Drives AO with a synthesized sine (phase subtraction on card), reads AI, and performs per-sample MACs against precomputed weights stored in BRAM. A fabric-clock *loader loop* copies weights from two Host→TargetDMAs into dual-clock memories, then raises a `Weights_Ready` flag that gates the 125 MHz SCTL.

Host VI. One VI to rule them all: opens the FPGA reference, loads estimator weights (once), configures low-rate registers (stimulus/divisors/decimation), and performs high-throughput DMA

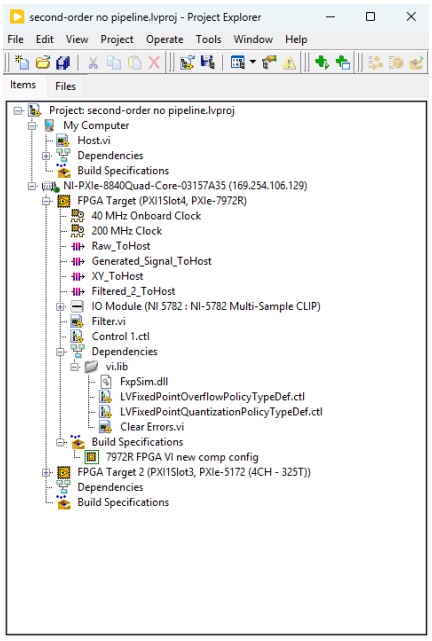

Figure 2.1: Project Explorer: Host VI under *My Computer*; FPGA Target (PXI1SlotX) with NI-5782R; DMAs and build specs.

reads in large chunks, converting fixed-point to `DBL` and building waveforms with dt=8 ns for plotting and logging.

2.3 Data planes (clean separation)

Control plane (low-rate).
Read/Write Control nodes carry scalars and flags (e.g., stimulus selects, decimation factor, run/idle). This traffic is not for continuous timeseries.

Stream plane (high-rate).
DMA FIFOs carry contiguous time data: raw/AI, filtered, and AO-prewire signals (Target→Host) as well as one-shot tables like weights (Host→Target). Chunk sizes of 65k–131k samples keep plots smooth and PCIe overhead low.

2.4 Artifacts and naming (what to look for)

- **Bitfiles:** `Filter_LA_Biquad.lvbitx`, `Phase_Estimator.lvbitx`.
- **Target→HostDMAs (examples):** `Raw_ToHost`, `AI0_ToHost`, `Filtered_ToHost`, `AO_PreWire_ToHost`.
- **Host→TargetDMAs (weights):** `Host_To_Target_Weights_Phi`, `Host_To_Target_Weights_I`.
- **BRAMs:** `Weights_Phi`, `Weights_I` (dual-port, dual-clock).
- **Gates/flags:** `Weights_Ready` (boolean), decimation factor N_{dec}.

2.5 Quickstart (what you actually do)

If you just cloned the repo and want graphs:

1. **Open the project.** Confirm the FPGA Target matches your chassis slot and the NI-5782R uses the *Multi-Sample CLIP*. The I/O Module Clock (125 MHz) must be visible.
2. **Build or bind.** Either build the FPGA VIs or bind *Open FPGA VI Reference* on the Host VI to the provided `.lvbitx`.
3. **(Estimator only) Load weights.** From MATLAB export two CSVs (6250 lines each). The Host VI reads, quantizes to Q4.14, and writes each to its DMA. The FPGA loader writes BRAM and asserts `Weights_Ready`.
4. **Hit Run.** Start the Host VI. You should see graphs update as DMA chunks arrive; low-rate controls (stimulus, decimation, mode) respond immediately.

For the exact wizard steps, control names, and types, see Chapter 3. For math and timing structure, see Chapters 5 and 6.

2.6 What the figures show (and where they live)

Each chapter embeds screenshots of the *actual* VIs so a reader can map text to blocks at a glance:

- Figures 5.1 to 5.3: filter core, stimulus, and streaming.
- Figures 6.2 to 6.4: estimator weight loader, MAC SCTL, and AO path.
- Host-side patterns for *Open FPGA VI Reference*, DMA reads, and Read/Write Control appear in Figures 4.1 to 4.3.

Note for future students (sampling rate for the real experiment). The physical input in the planned experiment is centered at 84.6 MHz. In this work we used a single AI lane at 125 MS/s. For the live experiment you'll configure both AI lanes to raise the effective sampling to 250 MS/s (details and one-line rationale are mentioned at the end of Chapter 6); this was deferred here while the optical bench was not yet driving the FPGA directly.

Chapter 3

Project Setup: Targets, I/O, DMA, and Host Binding

3.1 Add controller, chassis, and the FPGA target

1. File → New Project.
2. Right-click the project root → New → Targets and Devices...
3. Select **Existing target or device** and discover your PXI system (or pick it manually).
4. In the chassis tree, add the **FPGA Target**:
   - Choose **PXIe-7972R**, and place it at the correct slot (e.g., **PXI1Slot4**).
5. *Sanity check.* The new **FPGA Target** node should appear under the chassis.

3.2 Add NI-5782R and select the correct CLIP

1. Right-click **FPGA Target** → New → I/O Module.
2. Pick **NI-5782R**.
3. Right-click the 5782R node → Properties:
   - **Socketed CLIP** → choose **NI-5782R (Multi-Sample)**.
   - Click **OK**.
4. *You should now see* AI/AO terminals and the **I/O Module Clock (125 MHz)** in your VI.

3.3 Clocks and timing choices

- **Fast compute clock:** use **I/O Module Clock (125 MHz)** for any SCTL that touches AI/AO or timing-critical math.
- **Auxiliary clock(s):** keep **40 MHz Onboard** (default) for utility loops. If desired, add a derived **200 MHz** clock for non-blocking loaders.
- **Rule:** no blocking primitives (e.g., FIFO.Read with timeout −1) inside SCTLs.

3.4 Create DMA FIFOs (names, directions, types)

We standardize DMA names and element types so host/FPGA code and this report stay in sync.

Target→Host (timeseries)

Create each under the **FPGA Target** with:

- **Name** → `Raw_ToHost` (Element type: `I16`)
- **Name** → `AI0_ToHost` (Element type: `I32` carrying Q2.13)
- **Name** → `Filtered_ToHost` (Element type: `I32` carrying Q2.13)
- **Name** → `AO_PreWire_ToHost` (Element type: `I32` carrying Q2.13)

Common settings (each FIFO's *Properties*):

- **Direction:** *Target to Host*
- **Arbitration:** *Lossless*
- **Requested depth:** `16384` elements (safe default; increase if your host is busy)

Host→Target (tables/weights)

- **Name** → `Host_To_Target_Weights_Phi` (Element type: `FXP Q4.14` or raw `I16` interpreted as Q4.14)
- **Name** → `Host_To_Target_Weights_I` (Element type: `FXP Q4.14` or raw `I16` interpreted as Q4.14)

Settings:

- **Direction:** *Host to Target*
- **Arbitration:** *Lossless*
- **Requested depth:** `8192`–`16384` elements

3.5 Block memories (BRAM) for weights

Create two memories on the FPGA:

- **Name** `Weights_Phi` — **Depth** `6250`, **Word** `18-bit` (Q4.14 or I16), **Ports** *Dual-port*.
- **Name** `Weights_I` — same configuration.

Important property: enable *separate clock domains* so **Port A** (write) can run on a fabric/aux clock while **Port B** (read) runs at 125 MHz. **Discipline:** write *only* during the loader phase; the SCTL reads *after* a ready flag is raised (no simultaneous R/W).

3.6 Top-level FPGA VI scaffolding

1. Drop a **Single-Cycle Timed Loop** and bind it to **I/O Module Clock (125 MHz)**.
2. Inside the SCTL, place:
   - AI/AO nodes as needed.
   - DSP48-backed math (filter or estimator MACs).
   - DMA *Write* nodes for the Target→Host streams, gated by a decimation counter as required.

3. Outside the SCTL, add a **while loop** on a fabric/aux clock for the *loader*:
   - **FIFO Read** from `Host_To_Target_Weights_Phi/I` with *Timeout* -1 (blocking).
   - Addressed writes into `Weights_Phi/I`.
   - When each reaches 6250 writes, assert a `Weights_Ready` boolean (single-element memory/register).
4. At the SCTL boundary, implement a simple **run gate**:

```
if (Weights_Ready==FALSE)  then  IDLE  else  RUN.
```

3.7 Host VI: binding, reads/writes, and defaults

Open/close. Place **Open FPGA VI Reference** (bind to the `.lvbitx` or to the FPGA VI in **PXI1SlotX**); thread the FPGA ref + error wire through all nodes to **Close FPGA VI Reference**.

Weight load (one-shot).

1. Read each CSV into `DBL[6250]` via **Read Delimited Spreadsheet**.
2. Convert to **FXP Q4.14** (18-bit word, 4 integer) with *saturate* on overflow.
3. **FIFO Write** the full arrays to `Host_To_Target_Weights_Phi` and `...Weights_I`. *Timeout:* `5000 ms`.

Timeseries reads (continuous). For each Target→Host FIFO, use:

- **Number of elements:** `65 536–131 072`
- **Timeout:** `5000 ms`
- Convert `I32` Q2.13 to `DBL` by multiplying by 2^{-13}; build a waveform with $dt = 8\,\text{ns}$ (times decimation, if used).

Low-rate controls. Expose knobs/flags (e.g., stimulus selects, decimator N_{dec}) via a single **Read/Write Control** node.

3.8 Checklists (compile and run)

Before compile

- 5782R CLIP is **Multi-Sample**.
- SCTL clock is **I/O Module Clock (125 MHz)**.
- All SCTL multiplies mapped to **DSP48** (use primitives/SubVIs as needed).
- BRAMs for weights are **dual-port** with **separate clock domains** enabled.

Before first run

- Host weight CSVs each have **exactly 6250** finite values.
- Host converts to **Q4.14** (saturate) and writes the full arrays (no partial enqueues).
- The loader loop's address counters reach 6250; `Weights_Ready` flips to **TRUE**.

If graphs are empty or choppy

- **Empty:** wrong bitfile/slot binding, or `Weights_Ready` still FALSE (estimator held in IDLE).
- **Timeouts/chop:** increase DMA chunk size, raise host timeout to $\sim$5 s, or reduce UI work during reads.
- **Axis off:** set waveform $dt = 8\,\text{ns} \times N_{\text{dec}}$.

3.9 Naming summary (for quick cross-reference)

Item	Name	Type/Notes
T→H FIFO	`Raw_ToHost`	`I16`
T→H FIFO	`AI0_ToHost`	`I32` carrying Q2.13
T→H FIFO	`Filtered_ToHost`	`I32` carrying Q2.13
T→H FIFO	`AO_PreWire_ToHost`	`I32` carrying Q2.13
H→T FIFO	`Host_To_Target_Weights_Phi`	`FXP Q4.14` or `I16` (Q4.14)
H→T FIFO	`Host_To_Target_Weights_I`	`FXP Q4.14` or `I16` (Q4.14)
BRAM	`Weights_Phi`	depth 6250, 18-bit, dual-clock
BRAM	`Weights_I`	depth 6250, 18-bit, dual-clock
Flag	`Weights_Ready`	boolean (set TRUE after both tables loaded)

Chapter 4

Host VI (Second-Order Filter): Reference, Plots, and Controls

This chapter documents the *Host_VI.vi* that drives the **Filter** bitfile. The Host does three things: (i) open/close the FPGA reference cleanly, (ii) stream timeseries from multiple Target→Host DMAs and plot/log them, and (iii) push slow-changing controls via a single Read/Write Control node.

4.1 Block-level overview

The Host diagram has two parallel lanes threaded by the same *FPGA ref* and *error* wires:

1. **Stream+Plot lane (producer)**: four `FIFO Read` chains (`Raw_ToHost`, `AI0_ToHost`, `Filtered_ToHost`, `AO_PreWire_ToHost`) → type handling → `Build Waveform` → graphs (and optional logger).
2. **Controls lane (slow)**: one `Read/Write Control` node exposing `Square_Divisor_N`, `Saw_AMP`, `Sine_Delta`, `Noise_Enable`, `DMA_Decimation_N`, and any run/flag bits.

4.2 Open/Close FPGA VI Reference

Place **Open FPGA VI Reference** at the left of the diagram and choose either:

- *Bitfile binding:* browse to the built `.lvbitx`, or
- *VI+resource binding:* select the FPGA VI under **PXI1SlotX**.

Thread both *FPGA ref* and the *error cluster* through *every* node, then into **Close FPGA VI Reference**. This enforces a clean open–use–close lifetime.

4.3 Streaming pipeline (one pattern per FIFO)

Instantiate the same *read→scale→waveform* pattern four times—one per FIFO. Use generous chunk sizes to keep the UI smooth.

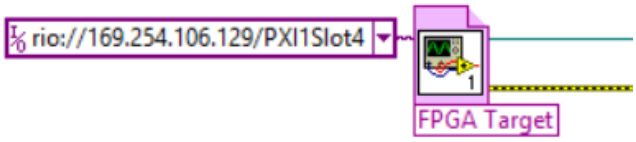

Figure 4.1: Open FPGA VI Reference feeding both lanes. **Legend:** the wire labeled *FPGA ref* is the FPGA reference (carried left→right through every node); the wire labeled *error* is the error cluster threaded through the same path to *Close FPGA VI Reference*. This identifies each line unambiguously without relying on color or pattern.

FIFO.Read parameters (good defaults)

- **Number of elements**: `65 536–131 072` (elements, not bytes).
- **Timeout (ms)**: `5000`. This lets the host block for full chunks under bursty PCIe.

Type handling and scaling

- `Raw_ToHost` is `I16` (ADC counts). Convert to `DBL` only for plots; units depend on AI range.
- `AI0_ToHost`, `Filtered_ToHost`, `AO_PreWire_ToHost` carry `I32` that represent Q2.13. Convert by multiplying by 2^{-13} into `DBL`.

Waveform construction

Build a waveform with

$$dt = \frac{1}{125\,\mathrm{MHz}} = 8 \times 10^{-9}\,\mathrm{s}.$$

If the FPGA decimator writes once every N_{dec} SCTL cycles, set $dt = 8\,\mathrm{ns} \times N_{\mathrm{dec}}$.

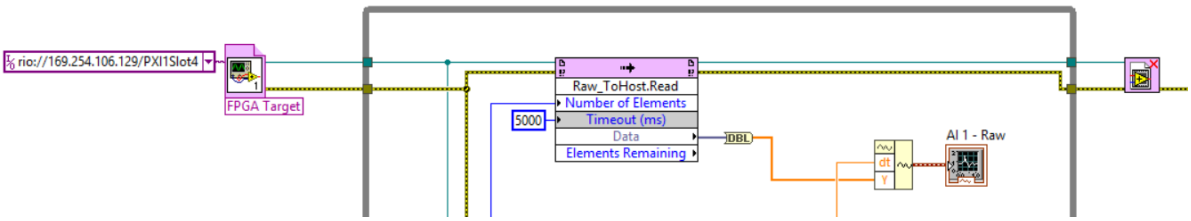

Figure 4.2: Canonical DMA read → scale Q2.13 → `Build Waveform` ($dt = 8$ ns $\times N_{\mathrm{dec}}$) → Waveform Graph. Duplicate for Raw, AI0, Filtered, AO-PreWire.

Why DMA (not Read/Write Control) for timeseries? Read/Write Control performs register-level accesses over PCIe and will not sustain multi-MS/s streams. **DMA FIFOs** buffer on-card and burst to the host without loss.

4.4 Controls via one Read/Write Control node

Expose slow knobs in a single **Read/Write Control** node (keeps the diagram tidy and avoids racey UI polling). Suggested register map:

- `Square_Divisor_N` — integer N; square toggles every N ticks, $f_{\square} = 125\,\text{MHz}/(2N)$.
- `Saw_AMP` — ramp increment (I16 LSBs).
- `Sine_Delta` — phase step (rad/tick); $f_{\sin} = \Delta/(2\pi) \cdot 125\,\text{MHz}$.
- `Noise_Enable` — boolean injector (optional in your build).
- `DMA_Decimation_N` — write every N_{dec} SCTL cycles to the DMAs; remember to scale dt on host.
- `Run/Mode/Flags` — any other mode selects.

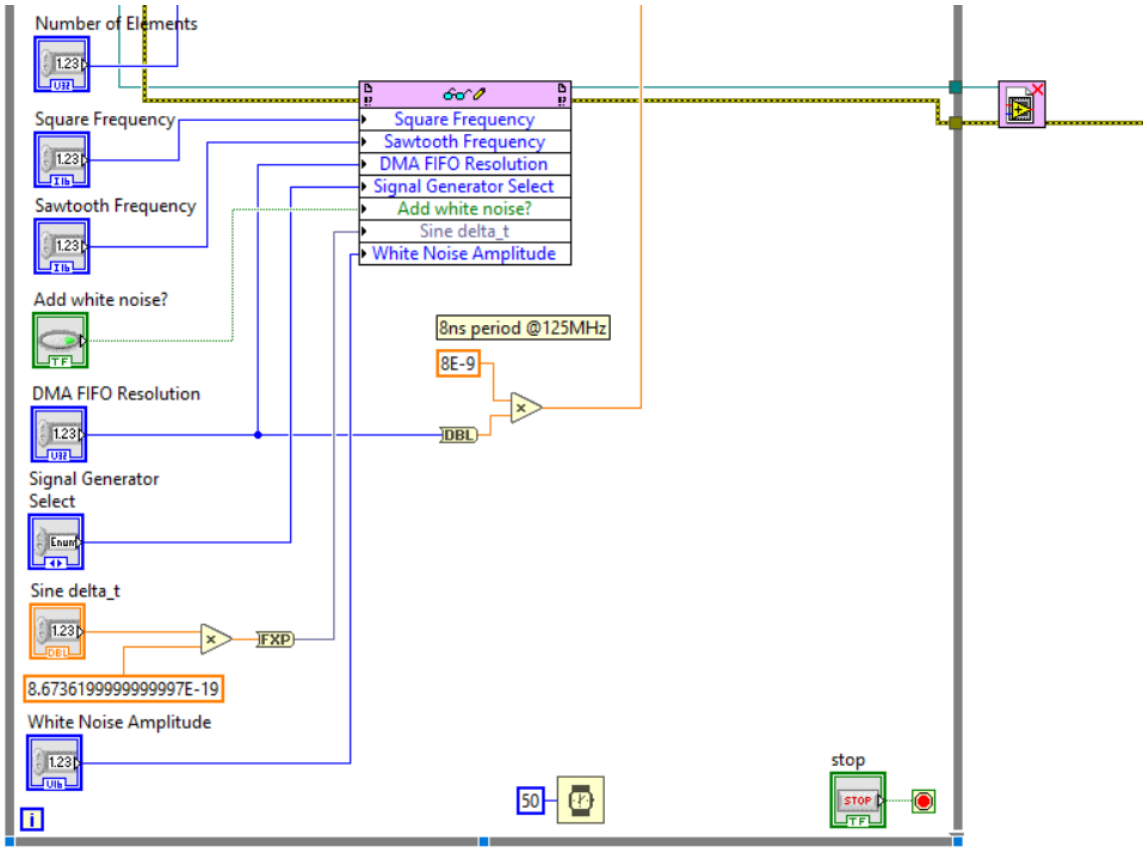

Figure 4.3: One Read/Write Control node with the register map. Keep it on the same error wire as the streaming lane.

4.5 Optional: producer/consumer logging

If you need disk logs without UI jitter:

1. The streaming lane becomes a *producer*: enqueue whole DMA chunks into a queue.
2. A separate *consumer* loop dequeues and writes TDMS/binary. This decouples PCIe bursts from disk/UI.

4.6 Putting it together (one screen)

A compact layout is: far left → **Open FPGA VI Reference**; middle → four identical DMA chains feeding graphs; right → one Read/Write Control node; bottom-right → **Close FPGA VI Reference**. Keep everything on the same error wire.

4.7 Troubleshooting (host side)

- **All zeros / no updates:** wrong bitfile/slot binding, FPGA VI not running, or the FPGA SCTL is gated (e.g., waiting on a flag in other builds).
- **Timeouts or "striped" plots:** increase *Number of elements* per read and *Timeout* to $\sim$5 s; avoid heavy UI work in the streaming loop.
- **Axis looks slow:** forgot to set $dt = 8\,\text{ns} \times N_{\text{dec}}$.
- **UI sluggish:** move logging to a consumer loop; keep the read chunk at $\geq$ 65,536.

4.8 What to verify (quick checklist)

- **Scaling:** `I32`→`DBL` multiply by 2^{-13} for Q2.13 streams.
- **Chunking:** reads are at least 65,536 elements with $\geq$ 5000 ms timeout.
- **Threading:** ref+error wires pass through *all* nodes to Close.
- **Decimation:** host dt matches FPGA decimator N_{dec}.

Chapter 5

FPGA Filter VI: Math, Generators, and Streaming

This chapter details the FPGA implementation of the k=1 look-ahead biquad, the on-card stimulus generators, and the streaming path to the host. The design goals were (i) a clean 125 MHz Single-Cycle Timed Loop (SCTL) with uniform one-tick latency on *all* branches, (ii) DSP48E1-based multiplies/MACs with a balanced adder tree, and (iii) a single final cast to Q2.13.

5.1 Top-level architecture

A single SCTL clocked from the **I/O Module Clock (125 MHz)** contains:

1. the look-ahead biquad math (parallel DSP48 multiplies → balanced adder tree → state update),
2. optional stimulus generators that can drive AO (square, saw, sine),
3. DMA writers case-gated by a decimation counter for host plotting/logging.

5.2 Signal path and numerics

Formats. The streaming signal uses Q2.13 end-to-end. Coefficients are held in narrower FXP (e.g. Q2.16) on DSP inputs; all intermediate sums remain wide and only the *final* node casts to Q2.13. If saturation is required, apply it exactly once at this final cast.

Look-ahead form (k=1). The direct-form II recursion is re-expressed to tolerate a uniform pipeline without changing the poles/zeros:

$$y[n] \;=\; \sum_{m=0}^{4} b'_m \, x[n-m] \;+\; A_2 \, y[n-2] \;+\; A_4 \, y[n-4].$$

All five products p_i are computed in parallel, registered once, and then reduced by a balanced adder tree.

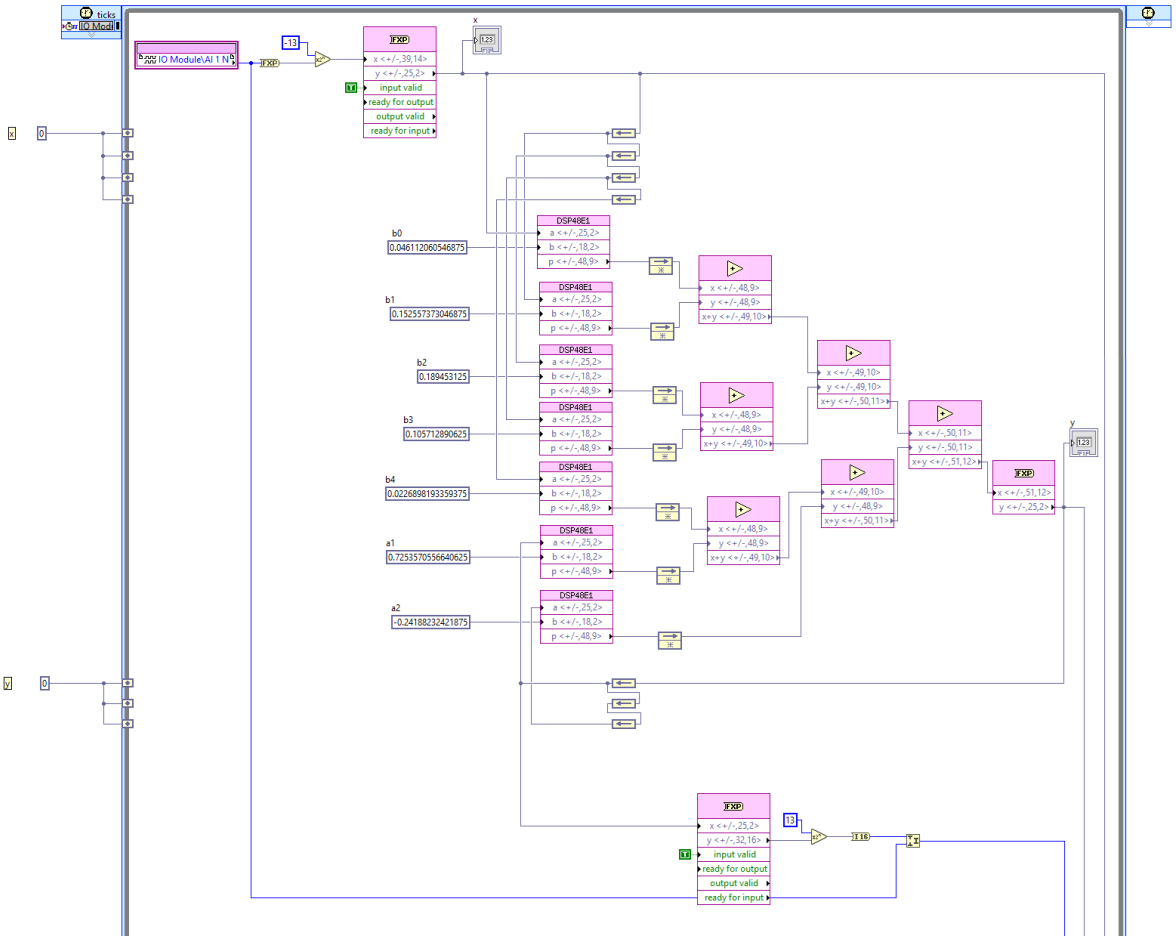

Figure 5.1: Core filter region in the SCTL: parallel DSP48E1 multiplies, uniform 1-cycle registers on *every* branch, balanced adder tree, single final cast to Q2.13.

5.3 DSP48E1 mapping (multiplies and MACs)

Each product

$$p_0 = b'_0 x[n],\ p_1 = b'_1 x[n-1],\ p_2 = b'_2 x[n-2],\ p_3 = A_2 y[n-2],\ p_4 = A_4 y[n-4]$$

is mapped to a separate DSP48E1 with **input and output registers enabled**. Keep the P outputs in 48-bit precision and feed the adder tree with these registered values. This approach:

- guarantees short, repeatable paths to meet 125 MHz,
- avoids pushing multiplies into LUT fabric,
- preserves numeric headroom until the final cast.

5.4 Balanced adder tree (avoid C→P cascades)

Summing via chained DSP C ← P creates serial dependencies that lengthen the critical path. Instead use a **balanced** 2-level tree in fabric (or HT adders):

$$s_0 = p_0 + p_1, \quad s_1 = p_2 + p_3, \quad s_2 = s_0 + s_1, \quad y_{\text{wide}} = s_2 + p_4.$$

Place *one* pipeline register on *every* branch feeding the first add (including $b'_0 x[n]$). This uniform one-tick latency ensures the only observable change is a pure z^{-1} delay.

5.5 Pipelining and state update

1. **Uniform one-tick delay.** Every input to the reduction tree is delayed by exactly one cycle. This keeps all paths time-aligned and maintains the intended transfer function.
2. **Final cast.** Convert y_{wide} to Q2.13 once, immediately before (i) writing the output state registers and (ii) streaming to DMA.
3. **Accounting for latency.** The measurable I/O effect is +1 sample; this fixed delay is handled in the host validation (§7).

5.6 Stimulus generators and AO/AI loopback

Three simple sources help bring-up and validation; they can be muxed to AO and looped back to AI1 for self-test.

Square (divider N)

Toggle a flip-flop every N ticks, $f_\square = 125\,\text{MHz}/(2N)$. Drive amplitude and duty via constants or host controls.

Sawtooth (I16 modulo)

An I16 accumulator steps by $4 \cdot \text{AMP}$ each tick; wrap at ± 32768 forms the ramp. Four AO phases can be offset by $\{0, \text{AMP}, 2\text{AMP}, 3\text{AMP}\}$.

Sine (phase accumulator)

A phase accumulator $\phi \leftarrow \phi + \Delta$ feeds an HT Sine; frequency $f_{\sin} = (\Delta/2\pi) \cdot 125\,\text{MHz}$. Scale to I16 full-scale for AO.

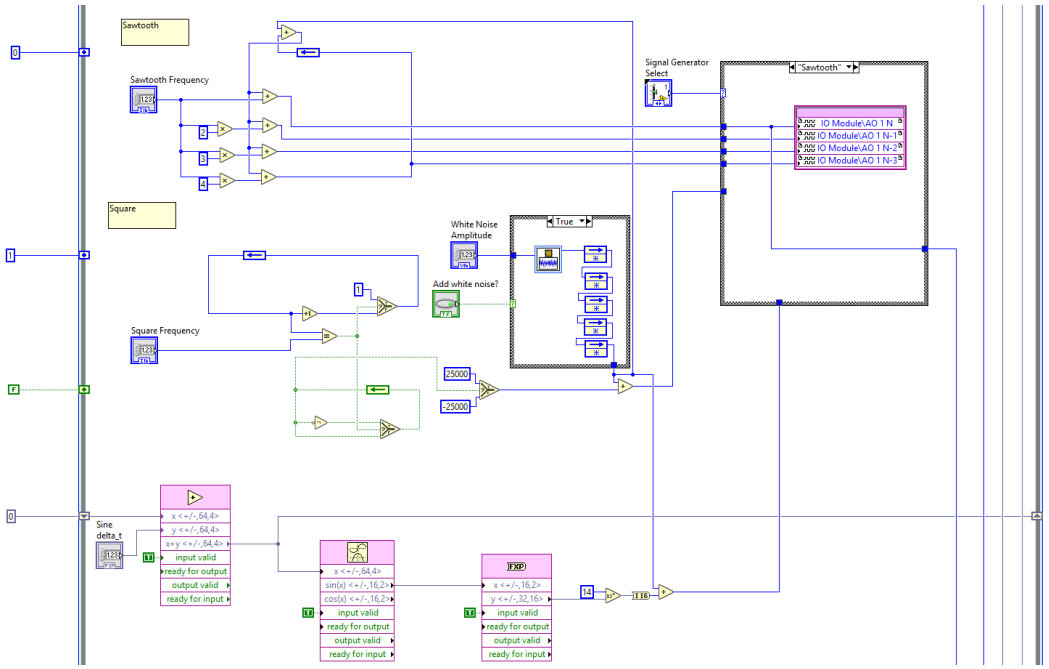

Figure 5.2: Stimulus block: square (divider N), saw (I16 modulo), sine (phase accumulator + HT Sine). AO can use four interleaved phases; AO→AI1 jumper enables self-test.

5.7 Streaming to host (DMA + decimator)

We expose four streams for analysis:

- `Raw_ToHost` — I16 ADC counts (pre-filter).
- `AIO_ToHost` — I32 carrying Q2.13 (post-format adapter).
- `Filtered_ToHost` — I32 Q2.13 (biquad output).
- `AO_PreWire_ToHost` — I32 Q2.13 (what is sent toward AO).

A free-running counter enables **DMA Write** every N_{dec} SCTL cycles. This keeps compute at full rate while giving the host a manageable plot bandwidth. The host sets waveform $dt = 8\,\text{ns} \times N_{\text{dec}}$ (see §4.3).

5.8 Timing-closure checklist

- Multiplies in **DSP48E1** with input/output regs *enabled*.
- **Balanced** adder tree; avoid DSP C→P cascades for reductions.
- **Uniform** single pipeline register on *every* branch into the first add.
- Keep intermediate precision wide; perform exactly *one* final cast to Q2.13.
- If still tight, move non-critical logic (e.g. debug taps) out of the SCTL.

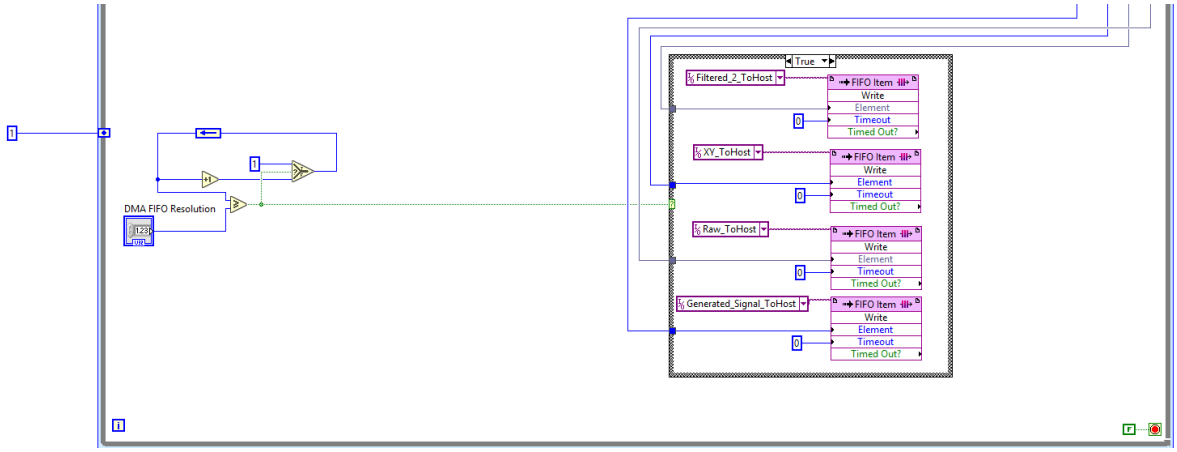

Figure 5.3: DMA writers gated by a decimation counter; compute remains full-rate, only the streams are thinned for the host.

5.9 Common pitfalls (and fixes)

- **Mismatched latencies.** If $b'_0 x[n]$ lacks the same one-tick register as the other branches, the sum is time-skewed. *Fix:* enforce the uniform register on all five paths.
- **Unintended LUT multiplies.** Generic multiply nodes can spill into fabric. *Fix:* use explicit DSP48E1/HT nodes pinned to DSP.
- **Mid-tree casts.** Truncating during the reduction raises noise/folds tones. *Fix:* cast once at the very end.
- **Host plots look "slow."** Host forgot to scale dt by N_{dec}. *Fix:* apply $dt = 8\,\text{ns} \times N_{\text{dec}}$.

Chapter 6

Phase-Estimator VI: AO Drive, MACs, Streaming, and Weight Loader

This chapter describes the four blocks inside `Phase_Estimator.vi`: (i) AO sine drive with commanded phase subtraction, (ii) the main 125 MHz SCTL that performs per-sample MACs using precomputed weights, (iii) Target→Host streaming of timeseries and per-pulse summaries, and (iv) the **weight-loading pipeline** from MATLAB→CSV→Host→FPGA BRAM with a race-safe start gate. The weight path is specified down to data types, timeouts, address counters, and the cross-clock memory setting required by LabVIEW FPGA.

6.1 Why weights are precomputed (avoid ÷ and $\sqrt{\cdot}$ on FPGA)

The per-sample weights would require division and square-root if computed on card. These operations either (i) need deep pipelines that complicate an SCTL at 125 MHz, or (ii) run too slowly to be practical. We therefore *precompute* all weights offline and treat the FPGA as a high-rate MAC engine with fixed coefficients.

6.2 Offline generation in MATLAB and file format

Two weight vectors are used per pulse:

$$\{g_k^{(\phi)}\}_{k=0}^{N-1}, \qquad \{g_k^{(i)}\}_{k=0}^{N-1}, \qquad N = 6250.$$

We generate them in MATLAB and export as two CSV files, one value per line, no headers:

- `weights_phi_N6250.csv` (6250 lines)
- `weights_i_N6250.csv` (6250 lines)

Total payload: $2N = 12{,}500$ numbers. Sanity: finite values, magnitudes within the fixed-point range used below.

6.3 Host VI: read CSVs, quantize to Q4.14, send over two DMAs

The Host loads the two CSVs, converts each `DBL[6250]` array to fixed-point, then enqueues them onto *two* Host→Target DMA FIFOs.

6.3.1 Parse & quantize

Read with **Read Delimited Spreadsheet** (no transpose). Convert to FXP with:

$$\text{word length} = 18, \quad \text{integer bits} = 4 \;\Rightarrow\; \textbf{Q4.14}.$$

Q4.14 covers $[-8,\, 8-2^{-14}]$ with $\approx 6.1\times10^{-5}$ LSB. Use *Saturate* on overflow. This matches the SCTL MAC numerics (weights Q4.14 × signals Q2.13).

6.3.2 Ship on two DMAs

Create:

- `Host_To_Target_Weights_Phi` (element: FXP Q4.14 or I16 that encodes Q4.14),
- `Host_To_Target_Weights_I` (same).

Host **FIFO Write**: enqueue each 6250-element array in a single call; set *Timeout* ≈ 5000 ms. Keeping the boundary in FXP (or I16 bit-pattern) avoids run-time scaling mistakes.

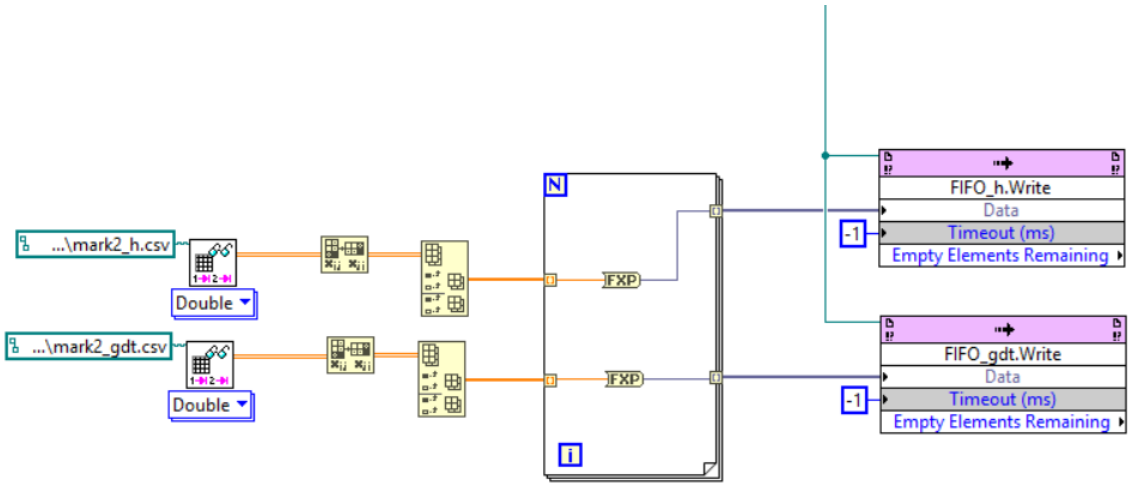

Figure 6.1: Host CSV ingest and quantization: two `DBL[6250]` arrays → Q4.14 → two Host→Target DMA writes.

6.4 FPGA loader loop, dual-clock BRAM, and the ready handshake

A *fabric-clock while loop* performs blocking DMA reads and writes a dual-port BRAM. The main SCTL (125 MHz) remains IDLE until all weights are loaded.

6.4.1 Block memory configuration

Instantiate two memories:

- `Weights_Phi`: depth = 6250, data = Q4.14 (or I16), **dual-port**, write on *fabric/aux* clock, read on *125 MHz I/O Module Clock*.
- `Weights_I`: same as above.

Enable the option that **permits different clock domains**. LabVIEW warns about concurrent read/write; we avoid it by holding the SCTL until a ready flag is TRUE.

6.4.2 Loader logic

L1. Reset: `addrPhi=0`, `addrI=0`; set `Weights_Ready`=FALSE (1-element boolean memory/register).

L2. **FIFO Read** on each Host→Target DMA with *Timeout* = −1 (blocking). Wire *Timed out?* into a case so BRAM writes occur only when data arrived.

L3. For each value: write `BRAM[addr]` ← value; increment address; stop at 6250.

L4. When *both* memories reach 6250, set `Weights_Ready` ← `TRUE` and quiesce (or block for a future reload).

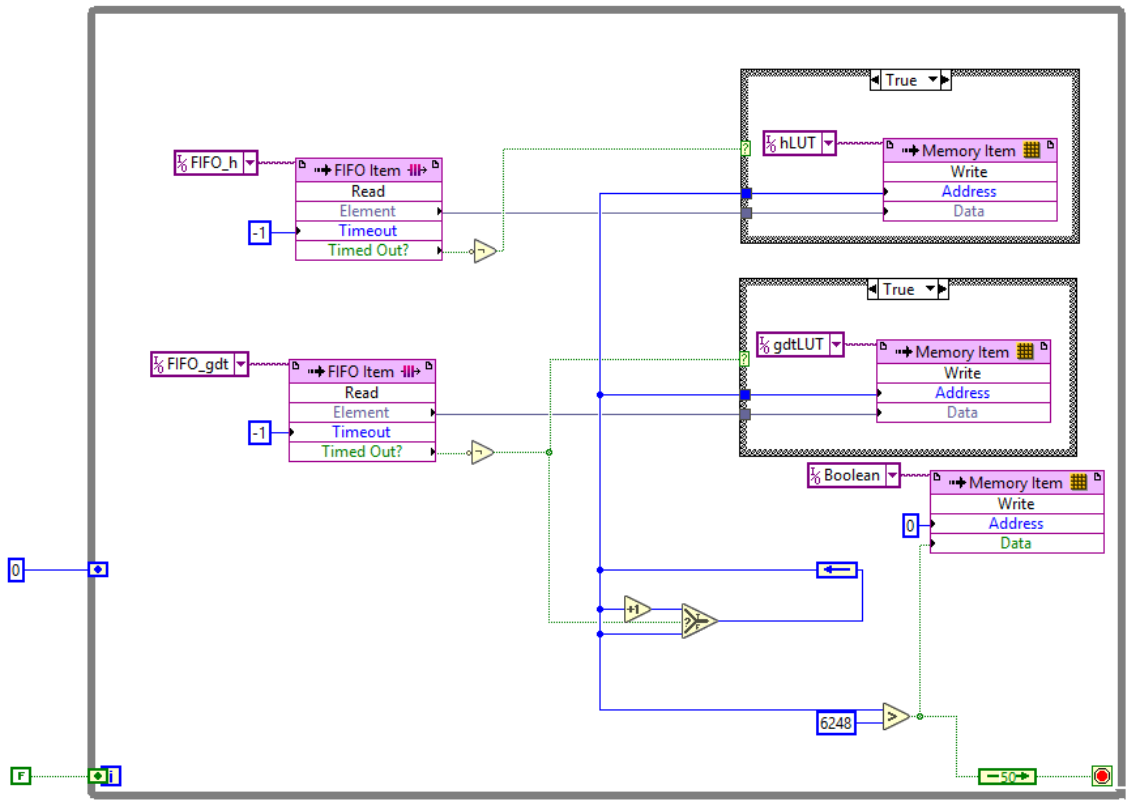

Figure 6.2: Loader while loop (fabric clock): blocking DMA reads (Timeout = −1) → addressed writes into dual-clock BRAM; assert `Weights_Ready` when both reach 6250.

Why not inside the SCTL? SCTLs cannot block; a FIFO Read with -1 timeout is illegal there. Splitting writer (fabric loop) and reader (SCTL) resolves this and keeps the 125 MHz loop deterministic.

6.5 Main SCTL: start gate, per-sample BRAM reads, and MACs

The SCTL (125 MHz I/O Module Clock) begins with a **run gate**:

if `Weights_Ready=FALSE: IDLE`; else: run.

Each tick:

S1. Read the current AI sample(s) and fetch the weights: $g^{(\phi)} \leftarrow$ `Weights_Phi[k]`, $g^{(i)} \leftarrow$ `Weights_I[k]`.

S2. Perform Q4.14×Q2.13 products in DSP48E1 and accumulate: $S_g\phi \mathrel{+}= g^{(\phi)}\phi_k$, $S_g i \mathrel{+}= g^{(i)} i_k$.

S3. On $k = N - 1$: compute end-of-pulse metrics ($\psi_{\text{end}}, \psi_{\text{int}}$, residual), wrap indices, clear accumulators.

Numerics: signals Q2.13; weights Q4.14; products in widened precision on the DSP P path; a single cast near outputs.

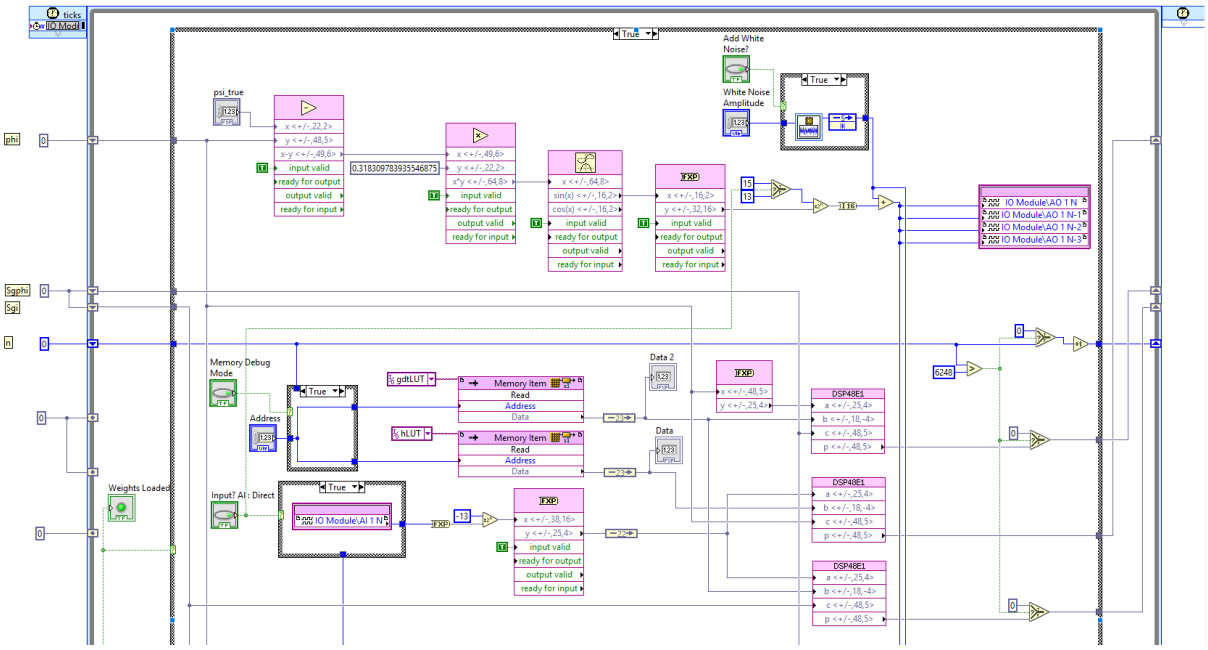

Figure 6.3: Estimator SCTL: gated start on `Weights_Ready`, per-sample BRAM reads, DSP48 MACs for $S_g\phi$ and $S_g i$, end-of-pulse reductions.

6.6 AO drive and streaming

The AO path synthesizes the experiment-frequency sine and applies commanded phase subtraction; four-phase interleave yields an effective 500 MS/s. Streaming uses Target→Host DMAs for timeseries and per-pulse summaries; optional decimation reduces host bandwidth.

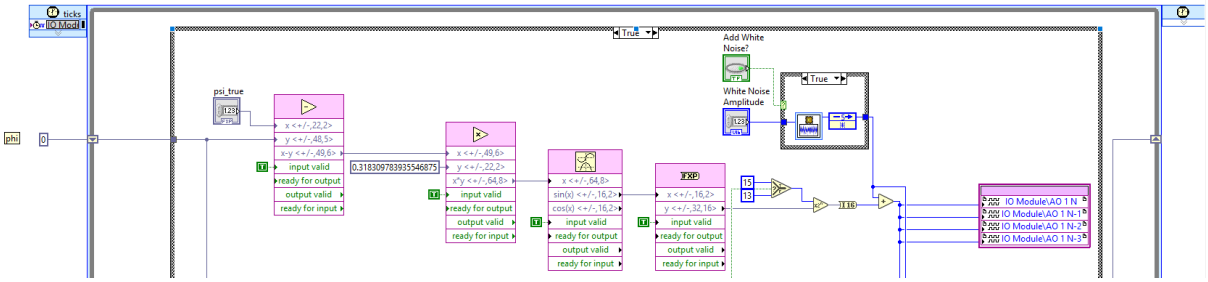

Figure 6.4: AO drive path: sine synthesis, phase subtraction, AO write (interleaved).

6.7 Pitfalls we hit (and fixes)

- **"SCTL cannot block."** Attempting FIFO Read with -1 timeout inside SCTL fails compilation. *Fix:* move reads into a fabric while loop.
- **Cross-clock BRAM warning.** Dual-clock access must be enabled. *Fix:* turn on "different clock domains," and gate the SCTL with `Weights_Ready` to avoid concurrent access.
- **Short loads.** If the host provides ¡6250 elements, the ready flag never asserts and the SCTL stays IDLE. *Fix:* host-side length checks and a "received count" indicator.
- **Type/range mismatches.** Out-of-range CSVs saturate at Q4.14 limits; document pre-scaling if needed. If using I16 DMAs, ensure a consistent Q4.14 interpretation on FPGA.

Note for future students (one-time mention of TDM×2 AI)

In these tests we used a single AI lane at 125 MS/s. For the 84.6 MHz input in the *actual* experiment, configure both NI-5782R AI lanes (TDM×2) to achieve an effective 250 MS/s and satisfy Nyquist. This requires minor changes to capture two samples per 125 MHz tick; it was deferred here because the live 84.6 MHz setup was not yet available.

Chapter 7

Results

7.1 Filter: Accuracy vs Reference

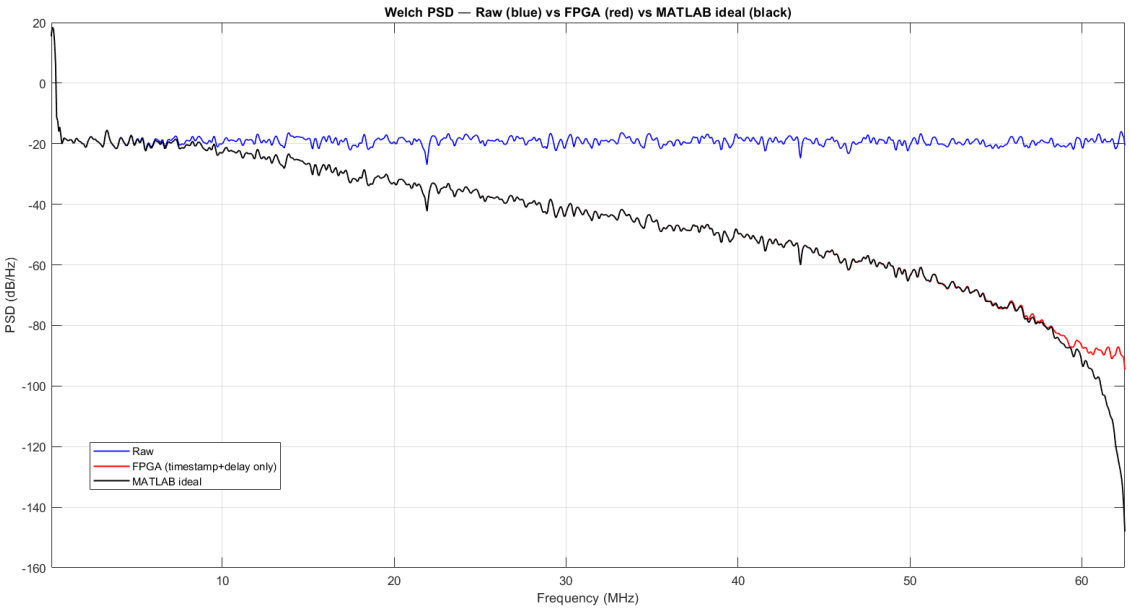

Figure 7.1: Welch PSD (one-sided) in dB/Hz: FPGA (red) overlaps MATLAB fixed-point (black) over 1 MHz–60 MHz; raw input sits above both as expected.

Key metrics (no gain/offset fit unless stated):

- Relative RMS error: $\mathbf{8.127 \times 10^{-5}}$
- NMSE: $\mathbf{-81.80}$ dB
- Added dynamics: $+\mathbf{1}$ sample pure delay (uniform pipeline)

7.2 Phase Estimator: Per-Pulse Parity

Representative numbers:

- ψ_{end} parity within ≈ 0.0001 rad at the marked pulse ($\sim 42.796\,\mu$s).
- RMS$\{\phi_{\text{HW}} - \phi_{\text{SIM}}\}$ on steady window: $\mathbf{2.729 \times 10^{-3}}$ rad ($\approx 2.71\%$ relative).
- ψ_{int} aligns after normalizing A and invS_g definitions across HW/SIM.

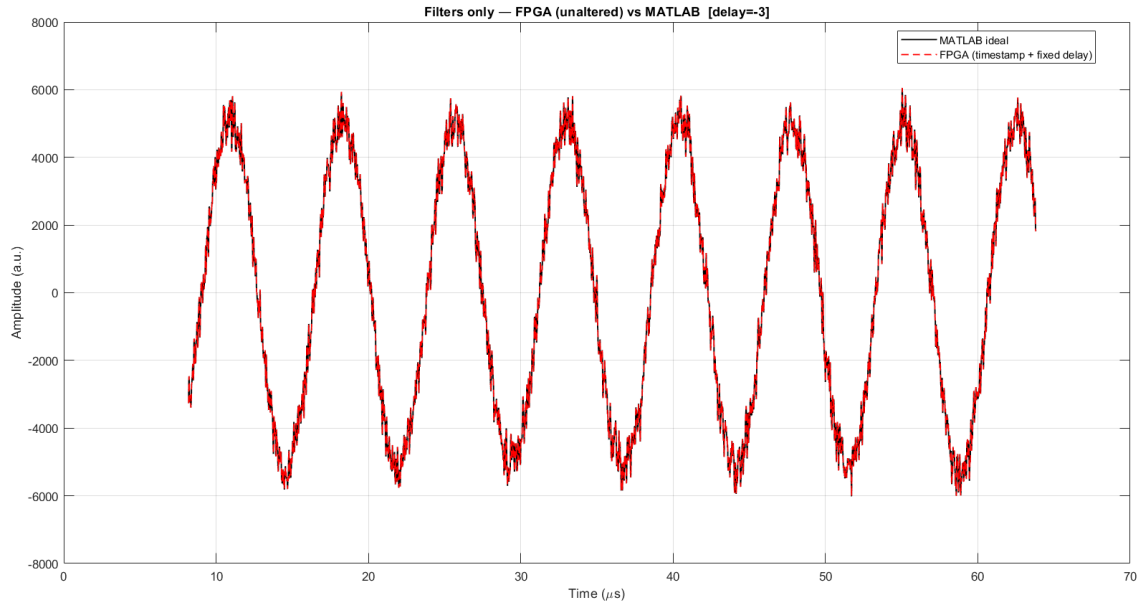

Figure 7.2: Time overlay after compensating only the known pipeline delay (-3 samples overall alignment in the capture): visual parity across cycles.

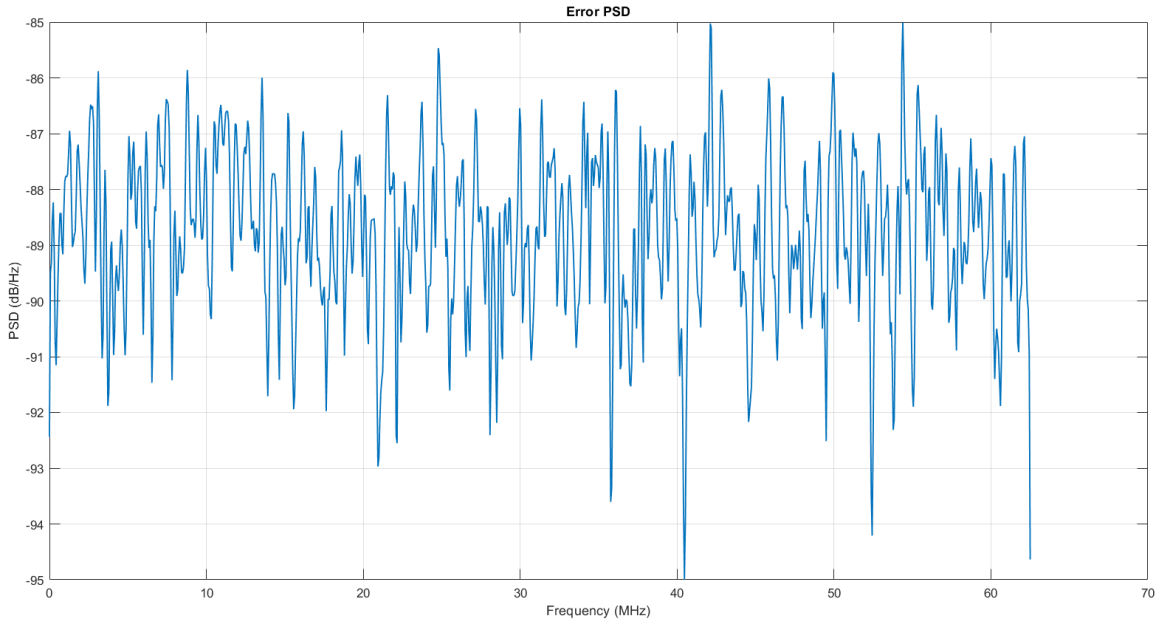

Figure 7.3: Error PSD between FPGA and MATLAB outputs. With correct full-scale normalization this is consistent with the Q2.13 quantization floor.

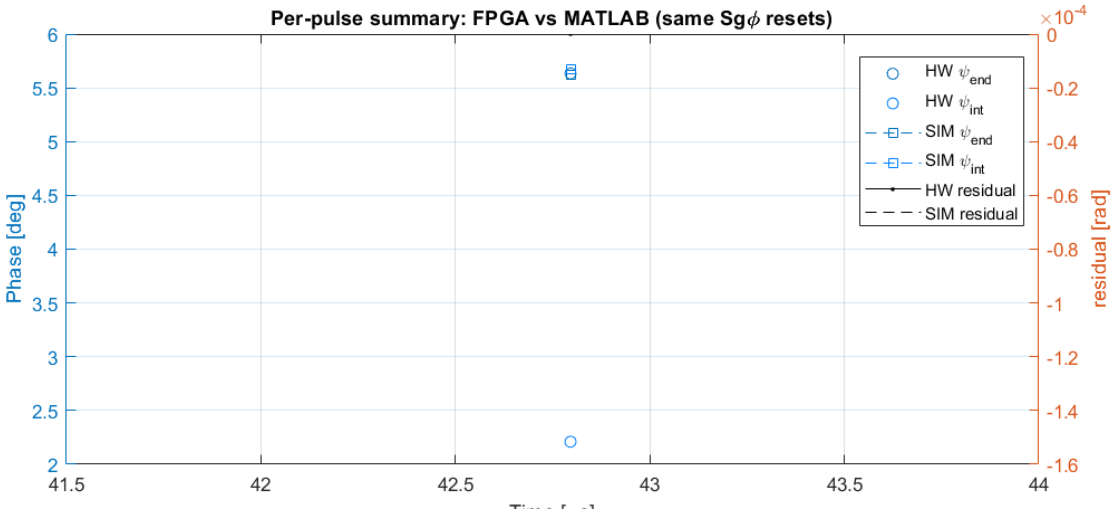

Figure 7.4: Per-pulse summary (Mark II): ψ_{end}, ψ_{int}, residual; HW (open circles) vs MATLAB (squares).

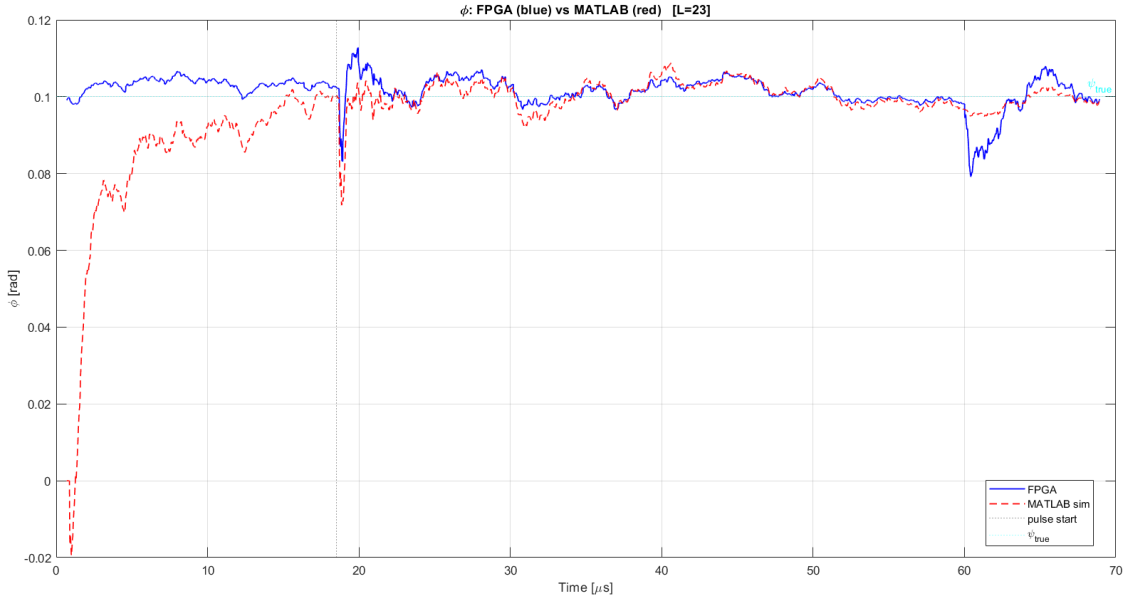

Figure 7.5: Time history of ϕ: hardware (solid) and MATLAB (dashed).

Chapter 8

Discussion

8.1 What the Numbers Mean

The biquad's NMSE -81.8 dB with only a 1-sample delay indicates correct look-ahead pipelining and error dominated by Q2.13 quantization. PSD overlap confirms that there are no hidden midtree casts or adder-latency mismatches. For the estimator, the steady-state RMS 2.7×10^{-3} rad is consistent with fixed-point noise and pulse framing; the ψ_{int} sensitivity underscores that A, S_g, and dt must be identical in HW and analysis.

8.2 Threats to Validity and Limits

- **I/O limits:** AI $\leq 250\,\text{MSa}\,\text{s}^{-1}$; AO interleave can introduce image content if misinterpreted.
- **Quantization:** Q2.13 step 2^{-13} yields a white floor $\approx \Delta/\sqrt{12}$. Saturation margins must be respected.
- **Streaming:** PCIe stalls produce bursty host reads; use deeper FIFOs and larger chunk sizes or decimation.
- **Clock domains:** Weight loading uses dual-clock BRAM with load-then-read-only discipline to avoid races.

Chapter 9

Conclusion

Q1: Yes. The look-ahead biquad meets real-time constraints at 125 MHz, preserves dynamics (only +1 sample delay), and achieves −81.8 dB NMSE. **Q2:** Yes (within scope). The weighted estimator tracks MATLAB to 2.73×10^{-3} rad RMS on a representative pulse; remaining discrepancies were normalization issues. **Next:** Try Q2.24 to push the noise floor, test dual-AI capture for higher carriers, and package DSP48 blocks as reusable SubVIs.

Bibliography

[1] National Instruments, *NI FlexRIO Documentation: PXIe-7972R and NI-5782R User Guides.*

[2] National Instruments, *LabVIEW FPGA Module Help & High-Throughput Math Nodes.*

[3] Xilinx, *7 Series DSP48E1 Slice User Guide (UG479).*

[4] A. V. Oppenheim, R. W. Schafer, *Discrete-Time Signal Processing.*

[5] P. D. Welch, "The Use of FFT for the Estimation of Power Spectra," *IEEE Trans. Audio and Electroacoustics*, 1967.

Appendix A

Validation Scripts (MATLAB)

A.1 Phase Estimator: `final_v5.m`

- **Inputs:** `phi.txt`, `sgphi.txt`, `sgi.txt`, `noise.txt` (two-column: time, value).
- **Assumptions:** N=6250, Fs=125 MHz; pulse boundaries inferred from streamed $S_g\phi$.
- **What it does:** aligns streams by timestamp, optionally grid-searches integer delay L, fits noise scale α on a steady window, computes per-pulse ψ_{end}, ψ_{int}, and residual using the *same* indices as the FPGA.
- **Outputs:** overlays (FPGA vs. sim), per-pulse plots, and steady-window RMS parity.

A.2 Filter Parity (timestamp + fixed delay)

- **Inputs:** `raw2.txt` (pre-filter), `fi2.txt` (FPGA output), both as (time,value).
- **Assumptions:** known pipeline delay of -3 samples; no gain/offset fitting or xcorr.
- **What it does:** stitches timestamps, applies only the fixed delay, runs MATLAB reference IIR with your b', A_2, A_4, and compares time/PSD/error PSD.

Appendix B

Abbreviations

Term	Meaning
FPGA	Field-Programmable Gate Array
SCTL	Single-Cycle Timed Loop
DMA FIFO	Direct Memory Access Queue
CLIP	Component-Level IP
DSP48E1	Xilinx MAC slice (A×B+C)
PSD	Power Spectral Density
Q2.13	Fixed-point with 2 integer + 13 fractional bits