Report on Fixed-Point Implementation and Hardware Mapping of a Scalar MMSE-Kalman Filter Progress

Brandon Crudele

November 25th, 2025

Abstract

This report details the work completed in implementing and testing a 1x1 and 2x2 complex scalar Kalman filter in Verilog. Work includes top-level integration, test bench creation, and debugging issues related to fixed-point scaling and pipelining. Although larger 4x4 and 8x8 matrices were not tested in detail, the methodology and theory discussed provide a foundation for scaling. Code snippets are included along with the progress made during the internship here at InnoMountain.

Contents

1 Introduction

The goal of this project was to implement a Kalman filter in fixed-point Verilog and verify its operation via test benches. This project in Verilog was further developed by me, extending the foundation laid by Sri Phanindra Perali. I focused on understanding MMSE and modified Kalman theory, analyzing existing Verilog modules, and creating top-level and scalar test benches to validate filter functions.

Key contributions include:

- Understanding and implementing a 1x1 and 2x2 scalar Kalman filter without explicit matrix inversion.
- Modifying top-level Verilog (`fpga_top_min.v`) to interface with the 2x2 core.
- Building test benches (`tb_fpga_top_min.v` and `tb_kf_scalar.v`) to validate filter behavior.
- Debugging fixed-point scaling, pipelining, and initialization issues.

2 Kalman Filter Theory and Implementation

2.1 1x1/2x2 Scalar Implementation

For scalar and 2x2 systems, explicit matrix inversion is not required. The gain K can be computed as:

$$K = \frac{P_{k|k-1}}{P_{k|k-1} + R_k}$$

and the state update as:

$$\hat{x}_k = \hat{x}_{k|k-1} + K(z_k - \hat{x}_{k|k-1})$$

$$P_k = (1 - K)P_{k|k-1}$$

This formulation is applied element-wise to the 2x2 complex system (4 real channels). For 1x1 and 2x2 experiments, no matrix inversion is needed because each scalar Kalman filter can operate independently.

Listing 1: Scalar Kalman gain and state update in kf_scalar.v

```
// Predict
wire signed [WX-1:0] x_pred = x_reg;
wire signed [WP-1:0] P_pred = P_reg + Q_k;

// Compute Kalman gain: K = P / (P + R)
wire signed [WP-1:0] denom = P_pred + R_k;
wire signed [WP-1:0] K_q;
div32_pipe #(.LAT(DIV_LAT), .WF_P(WF_P)) UDIV (
    .clk(clk),
    .rst_n(rst_n),
    .num(P_pred),
    .den(denom),
    .quo(K_q)
);

// Innovation
wire signed [WX-1:0] innov0 = z_k - x_pred;

// Compute update: x_hat = x_pred + K * innovation
// P = (1-K) * P_pred
wire signed [WX-1:0] dx = K_q * innov0 >> WF_P; // scale for Q format
wire signed [WP-1:0] Pn = (1 <<< WF_P - K_q) * P_pred >> WF_P;

// Commit update on pipeline finish
always @(posedge clk or negedge rst_n) begin
    if (load_init) begin
        x_reg <= x0; P_reg <= P0; x_hat <= x0;
    end else if (commit) begin
        x_reg <= x_pred + dx;
        P_reg <= Pn;
        x_hat <= x_pred + dx;
        m_valid <= 1'b1;
    end
end
```

2.2 Top-Level FPGA Module

The fpga_top_min.v module instantiates a 2x2 Kalman filter core and provides a simple FSM for control.

Listing 2: Top-level FSM snippet

```
localparam S_RESET = 0, S_LOAD = 1, S_IDLE = 2, S_MEASURE = 3;
always @(posedge clk_300) begin
    if (rst) state <= S_RESET;
    else begin
        case(state)
            S_RESET: load_init <= 1; state <= S_LOAD;
            S_LOAD: load_init <= 0; state <= S_IDLE;
            S_IDLE: en <= 1; state <= S_MEASURE;
            S_MEASURE: begin
                en <= 0;
                if (valid_all) state <= S_IDLE;
            end
        endcase
    end
end
```

2.3 2x2 Kalman Core

The `kf_mimo2x2_core.v` module instantiates 8 scalar filters for the real and imaginary parts of a 2x2 complex channel. Each filter operates independently, taking z_k measurements and producing $\hat{x}$ outputs.

Listing 3: Instantiation of scalar filters inside 2x2 core

```
kf_scalar k11r (..., .z_k(z11_re), ..., .x_hat(h11_re));
kf_scalar k11i (..., .z_k(z11_im), ..., .x_hat(h11_im));
kf_scalar k12r (..., .z_k(z12_re), ..., .x_hat(h12_re));
kf_scalar k12i (..., .z_k(z12_im), ..., .x_hat(h12_im));
...
```

For the 2x2 complex system, each of the four real channels (real and imaginary parts of the 2x2 matrix) is treated independently as a scalar Kalman filter. The innovation for each channel is computed separately:

$$\text{innovation}_{ij} = z_{ij} - \hat{x}_{ij|k-1}$$

where $i, j \in \{1, 2\}$ correspond to the row and column of the matrix. Each innovation is then scaled by its corresponding Kalman gain K_{ij} to update the channel estimate:

$$\hat{x}_{ij} = \hat{x}_{ij|k-1} + K_{ij} \cdot \text{innovation}_{ij}$$

3 Test Benches and Verification

To verify the correctness and stability of the Kalman filter implementation, I created two test benches, each targeting a different level of the design hierarchy.

3.1 `tb_kf_scalar.v`

The first test bench, `tb_kf_scalar.v`, focuses on a single scalar Kalman filter. This test bench allows detailed inspection of the internal filter behavior. It includes several types of tests:

Reset Test: Verifies that upon asserting `load_init`, the filter properly initializes the state `x_reg` and covariance `P_reg` to the specified `x0` and `P0` values.

Deterministic Test: Repeatedly applies a constant measurement, for example:

```
send_sample(16'sd800);
```

to ensure the filter converges to the correct state and remains stable across multiple cycles.

Inversion-like Test: Applies a series of measurements that test expected Kalman gain behavior:

```
send_sample(16'sd1000);
send_sample(16'sd1200);
send_sample(16'sd1300);
```

With `P=1` and `R=1`, this test confirms that the gain $K \approx 0.5$, and the output `x_hat` matches the theoretical expectation. The use of pipelined innovation and multiplication stages is verified.

3.2 `tb_fpga_top_min.v`

The second test bench, `tb_fpga_top_min.v`, targets the top-level module containing a 2x2 complex Kalman filter (equivalent to 4 independent scalar filters). Key steps inside:

Input Stimulation: Each complex channel is driven with test values, for instance:

```
z11r = 16'sd26214;
z11i = -16'sd3277;
```

Completion Monitoring: The `valid_all` signal, wired to the first LED, indicates when all four scalar filters have completed an update cycle:

```
wait(leds[0] == 1'b1);
```

This allows synchronization with the pipeline before the result is checked.

Observation: Through these two test benches, simple tests were executed to confirm that the initial code provided served as a solid template for Kalman filtering in Verilog. To further validate the implementation, I compared the Verilog outputs against a MATLAB simulation of a scalar Kalman filter. Using the deterministic parameters defined in the test benches, ie. `P0 = 1.0`, `Qk = 0`, and `Rk = 1.0`, the results matched closely, demonstrating correct convergence behavior. These deterministic inputs were chosen specifically for testing, allowing predictable verification of the filter's behavior. Whoever is assigned this project next may want to further verify edge cases. I would recommend tinkering with the parameters in the test bench for further verification.

3.3 Kalman Filter Parameters

Here is a quick overview of the parameters critical to the Kalman filter implemented:

- x_0 **(Initial State)**: The initial estimate of the state.
  - If x_0 is far from the true value, the filter will converge more slowly depending on P_0 and R_k.
  - If x_0 is close to the true value, fewer updates are required for convergence.
- P_0 **(Initial Covariance)**: Represents the uncertainty in the initial state estimate x_0.
  - *High* $P_0 \rightarrow$ filter is very uncertain about x_0, so it quickly adapts to measurements.

    - *Low* $P_0 \rightarrow$ filter trusts x_0 more and updates more slowly.

- Q_k **(Process Noise Covariance)**: Represents the expected variability in the system model (how much the state might change between steps).
    - *High* $Q_k \rightarrow$ system is highly dynamic; filter will react faster to changes in z_k.
    - *Low* $Q_k \rightarrow$ system is assumed nearly constant; filter relies more on prediction than measurement.

- R_k **(Measurement Noise Covariance)**: Represents the expected noise in the measurements.
    - *High* $R_k \rightarrow$ measurements are noisy; filter updates slowly and relies more on prediction.
    - *Low* $R_k \rightarrow$ measurements are accurate; filter trusts the measurements more and updates quickly.

4 Progress and Observations

During this project, I successfully implemented and verified functional 1x1 and 2x2 Kalman filters. The work included handling fixed-point scaling, pipelining, and extensive testing using test benches. Throughout the process, I debugged several issues that were causing synthesis failures and incorrect Kalman gains. Adjustments to quantization through bit-shifting in Verilog were necessary to achieve synthesis and testing.

5 Future Work for 4x4 and 8x8 Scaling

Scaling the Kalman filter to 4x4 and 8x8 matrices presents more challenges. Given that I was new to this technology and building upon another engineer's work, this was a significant task to tackle alone. My goal is to provide guidance for future interns, so they have a clear path forward, and this report can serve as a reference to transfer knowledge.

For larger matrices, the independent scalar updates used in 1x1 and 2x2 implementations are insufficient, and multi-dimensional Kalman updates, including matrix inversion, will be required. Dr. Xueyuan Zhao recommends implementing a suitable matrix inversion technique in Verilog for the 4x4 and 8x8 projects. I suggest starting with 4x4, as the complexity of 8x8 may be overwhelming initially. One potential approach is to first re-implement the 2x2 Kalman filter using matrix inversion to establish a methodology before scaling to 8x8. The groundwork has been lightly tested, and the foundation is now in place to convert the methodology for larger matrices.

Once a matrix inversion technique is implemented, my test bench, `tb_fpga_top_min.v`, can be adapted to verify correct operation. Techniques such as Cholesky or QR decomposition could be explored for the inversion. Additionally, careful attention to pipelining and FPGA resource usage will be essential to maintain throughput, which is a key consideration for this project.

6 Conclusion

Although I had no prior background in telecommunications, my internship at InnoMountain has given me a deep appreciation for the field. Over the past months, I have learned foundational theory in telecommunications, explored IEEE and 3GPP standards, and contributed to the MMSE team. My assignment was to build upon and verify the existing Kalman filter project, and while the work is not fully complete, I have documented and tested Sri's work sufficiently to hand it off to the next engineer. This experience has been incredibly valuable, and I am grateful for the opportunity.