Hardware Accelerated AHRS
CSEE-4840 Spring 2026

Anubhav Vandkar (av3336) | Jaxson Natalini Robinson (jnr2154) |
Zongyang Li (zl3621) | Darshan Ramakrishnaiah (dr3412)

1. Abstract

This technical document describes a project to display a basic AHRS display. Any primary flight display mainly showcases the Roll, Pitch and the Yaw of an aircraft, generally gathered by the Accelerometer, Gyroscope and Magnetometer, ultimately helping in figuring out the flight parameters of an aircraft.

The novelty of this project comes with the hardware accelerated Kalman filter execution on the Terasic DE1-SoC FPGA. The readings for the system come from an MPU-9250 sensor, giving us 9DOF values. The interfacing of the sensor will be done by the I2C driver, on the HPS of the FPGA. The raw data is processed, and is ultimately placed on the on-chip SRAM, which the Kalman filter streams from.

A Kalman filter is an optimal estimation algorithm that predicts the state of a system by combining any noisy sensor measurements with a mathematical model of the system. It works in a continuous cycle of predicting the next state and then correcting that prediction using real-time data to minimize uncertainty. Essentially, it filters out "noise" to provide a much more accurate estimate of our sensor values of roll, pitch and yaw.

2. Block Diagrams

a. System block diagram

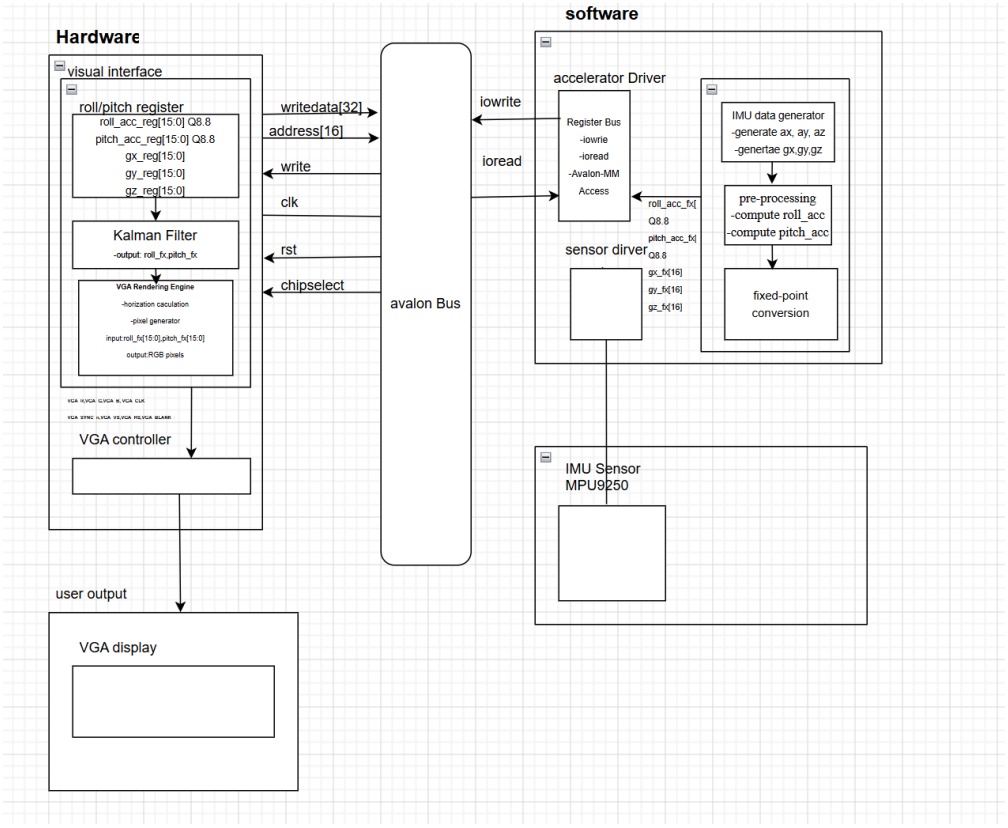

Fig1: System level block diagram

b. Sensor and I2C Driver block diagram

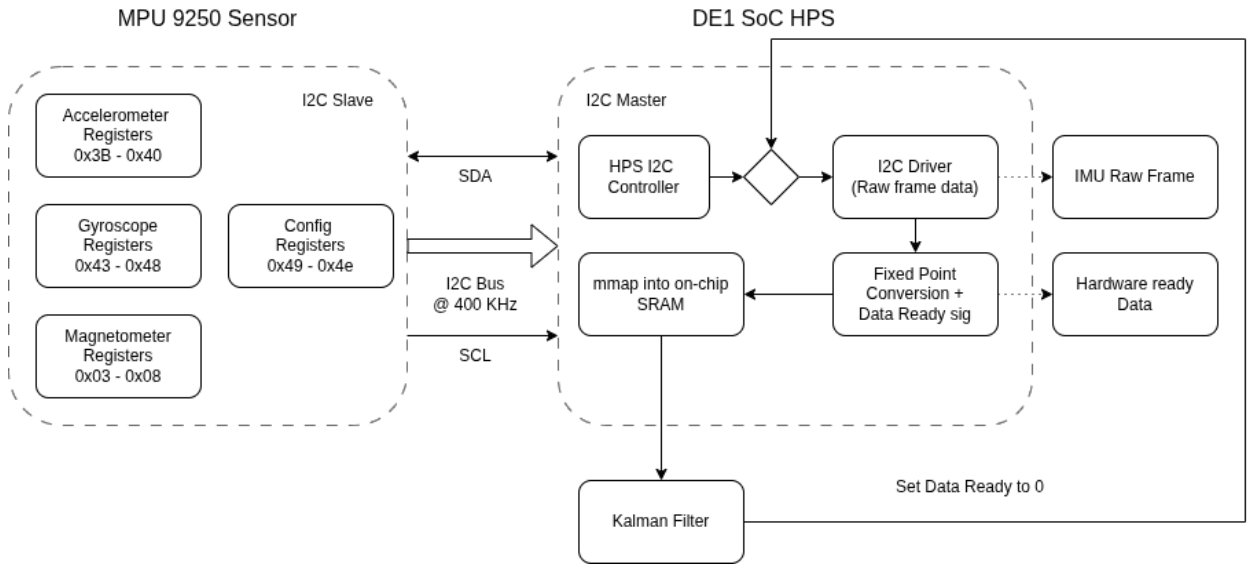

Fig2: Sensor interface

c. Kalman Filter block diagram

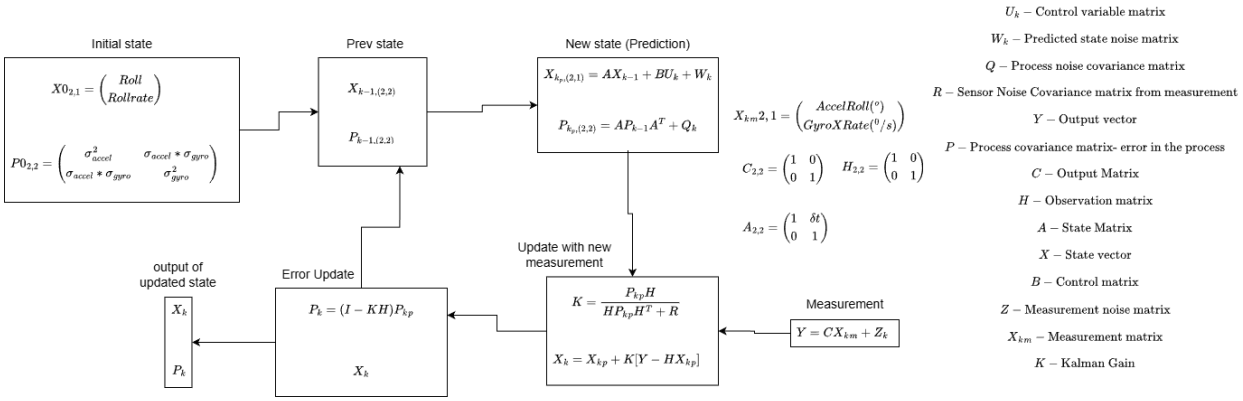

Fig3: Kalman filter estimation algorithm

The B control matrix is zeros as we are not modelling the vehicle dynamics and there is nothing to control our Roll and Roll rates. The predicted state noise matrix W is made zero for simplicity but can be added in later stages, The process covariance matrix P is made a diagonal matrix, as there is no correlation between accel and gyro values.

3. Implementation

a. MPU-9250 interfacing (I2C)

The MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer (MPU-6500), and a 3-axis magnetometer (AK8963) in one package, commonly referred to

as a 9DOF system. For an Attitude and Heading Reference System (AHRS), it provides the required inertial and magnetic data needed to calculate the aircraft's real orientation.

Relevant Technical Features

The 512-byte FIFO buffer and Interrupt pin are important for the C driver to get time-consistent data samples and prevent data loss when the HPS is busy. The device supports Fast Mode I2C (@ 400 kHz), which allows reading the required bytes of accel/gyro data plus magnetometer registers fast enough for high AHRS update rates. The AK8963 magnetometer can be accessed through an internal bypass mode or as a slave on the MPU-9250's primary bus.

Data Mapping for the imu_raw_frame_t struct

To fill the data structure efficiently, the MPU-6500 (14 bytes for Accel, temp, gyro) and AK8963 (7 bytes for Mag + status) support burst reads with auto-incrementing register addresses. A single burst read should be used for the 6 accel/gyro registers to keep the timestamp_us accurate. The sensor outputs data in Big-Endian (MSB first), so the C driver must swap bytes to correctly map them to the int16_t struct members. The mag_data flag should be mapped to the Data Ready (DRDY) bit in the AK8963's ST1 (Status 1) register to indicate a new magnetic sample.

Interfacing HPS and FPGA for Kalman Filter

Interfacing the HPS-side I2C driver with the FPGA's On-Chip SRAM creates a high speed, real time data stream for the Kalman Filter. This setup lets the HPS manage complex sensor communication while the FPGA handles the matrix math and arithmetic.

Interfacing Workflow

First, the C program uses the I2C driver to poll the sensor (IMU) at 1KHz. The C program then converts the raw sensor bytes into the fixed point format needed by the Kalman Filter. The HPS writes the formatted imu_frame_data struct directly into the On-Chip SRAM using an mmap pointer. Finally, the HPS sets a ready flag in the SRAM to 1. The Kalman Filter on the FPGA, which monitors that memory address, detects the high signal, pulls the data for its calculations, and writes the results back to the SRAM. The FPGA then sets the ready flag back to 0, signaling the HPS to start the next 1ms cycle.

Platform Designer (Qsys) Setup

To set up this data path in Platform Designer, the HPS-to-FPGA (H2F) AXI Master bridge must be enabled. This is the physical connection for the HPS to access FPGA resources. An On-Chip Memory component needs to be instantiated and set to the desired size (up to 512 KB for the Cyclone V M10K blocks) and configured as Dual-Port. For interconnects, Port 1 (HPS Side) connects the HPS h2f_axi_master to the SRAM's s1 slave port (Platform Designer handles the AXI-to-Avalon conversion). Port 2 (FPGA Side) exports the SRAM's s2 port to the top-level Verilog, giving the Kalman Filter direct, private access. Finally, a base address (e.g., 0x0000_0000) must be assigned in the Address Map tab,

which translates to a physical CPU address (e.g., 0xC0000000) for the mmap call.

The MPU-9250 is a 9-axis MotionTracking device that combines a 3-axis gyroscope, 3-axis accelerometer (MPU-6500), and a 3-axis magnetometer (AK8963) in one package, commonly referred to as a 9DOF system. For an Attitude and Heading Reference System (AHRS), it provides the required inertial and magnetic data needed to calculate the aircraft's real orientation.

Relevant Technical Features

The 512-byte FIFO buffer and Interrupt pin are important for the C driver to get time-consistent data samples and prevent data loss when the HPS is busy. The device supports Fast Mode I2C (@ 400 kHz), which allows reading the required bytes of accel/gyro data plus magnetometer registers fast enough for high AHRS update rates. The AK8963 magnetometer can be accessed through an internal bypass mode or as a slave on the MPU-9250's primary bus.

Data Mapping for the imu_raw_frame_t struct

To fill the data structure efficiently, the MPU-6500 (14 bytes for Accel, temp, gyro) and AK8963 (7 bytes for Mag + status) support burst reads with auto-incrementing register addresses. A single burst read should be used for the 6 accel/gyro registers to keep the timestamp_us accurate. The sensor outputs data in Big-Endian (MSB first), so the C driver must swap bytes to correctly map them to the int16_t struct members. The mag_fresh flag should be mapped to the Data Ready (DRDY) bit in the AK8963's ST1 (Status 1) register to indicate a new magnetic sample.

Interfacing HPS and FPGA for Kalman Filter

Interfacing the HPS-side I2C driver with the FPGA's On-Chip SRAM creates a high speed, real time data stream for the Kalman Filter. This setup lets the HPS manage complex sensor communication while the FPGA handles the matrix math and arithmetic.

Interfacing Workflow

First, the C program uses the I2C driver to poll the sensor (IMU) at 1KHz. The C program then converts the raw sensor bytes into the fixed point format needed by the Kalman Filter. The HPS writes the formatted imu_frame_data struct directly into the On-Chip SRAM using an mmap pointer. Finally, the HPS sets a ready flag in the SRAM to 1. The Kalman Filter on the FPGA, which monitors that memory address, detects the high signal, pulls the data for its calculations, and writes the results back to the SRAM. The FPGA then sets the ready flag back to 0, signaling the HPS to start the next 1ms cycle.

Platform Designer (Qsys) Setup

To set up this data path in Platform Designer, the HPS-to-FPGA (H2F) AXI Master bridge must be enabled. This is the physical connection for the HPS to access FPGA resources. An On-Chip Memory component needs to be instantiated and set to the desired size (up to 512 KB for the Cyclone V M10K

blocks) and configured as Dual-Port. For interconnects, Port 1 (HPS Side) connects the HPS h2f_axi_master to the SRAM's s1 slave port (Platform Designer handles the AXI-to-Avalon conversion). Port 2 (FPGA Side) exports the SRAM's s2 port to the top-level Verilog, giving the Kalman Filter direct, private access. Finally, a base address (e.g., 0x0000_0000) must be assigned in the Address Map tab, which translates to a physical CPU address (e.g., 0xC0000000) for the mmap call.

imu_frame_data interface (raw frame)

```
C/C++
typedef struct {
        uint32_t timestamp_us;
        uint16_t sample_count;
        int16_t ax, ay, az;
        int16_t gx, gy, gz;
        int16_t mx, my, mz;
        uint8_t mag_fresh;
} imu_raw_frame_t;
```

b. Kalman Filter

The ASM for the controller will be developed during the design phase, Roll and pitch estimation will run in parallel, each estimation will have **11 16-bit matrix multipliers** and **3 16-bit matrix adders,** each matrix multipliers will have 8 16-bit multipliers and 4 16-bit adders each matrix adder will have 4 16 bit adders, so in total **176 16-bit multipliers, 112 16-bit adders**

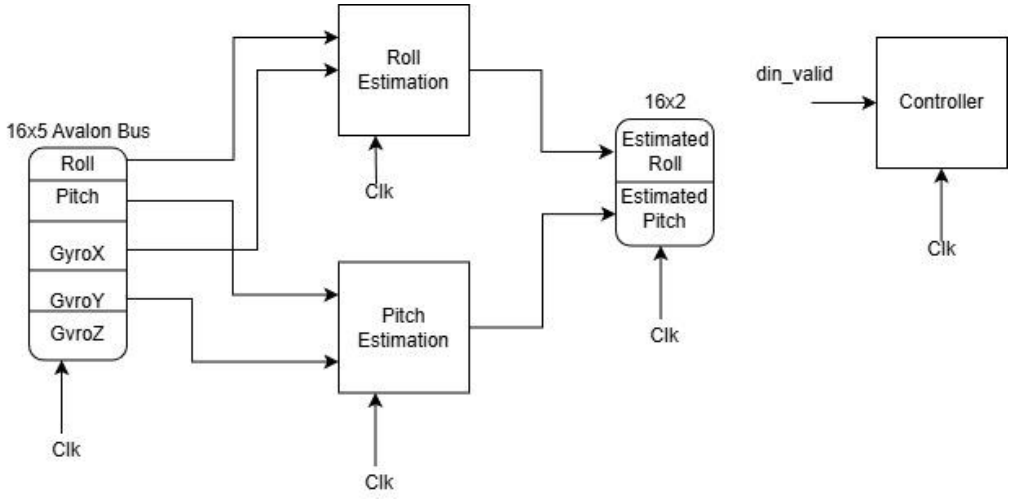

Fig4: Kalman filter hardware top level block diagram

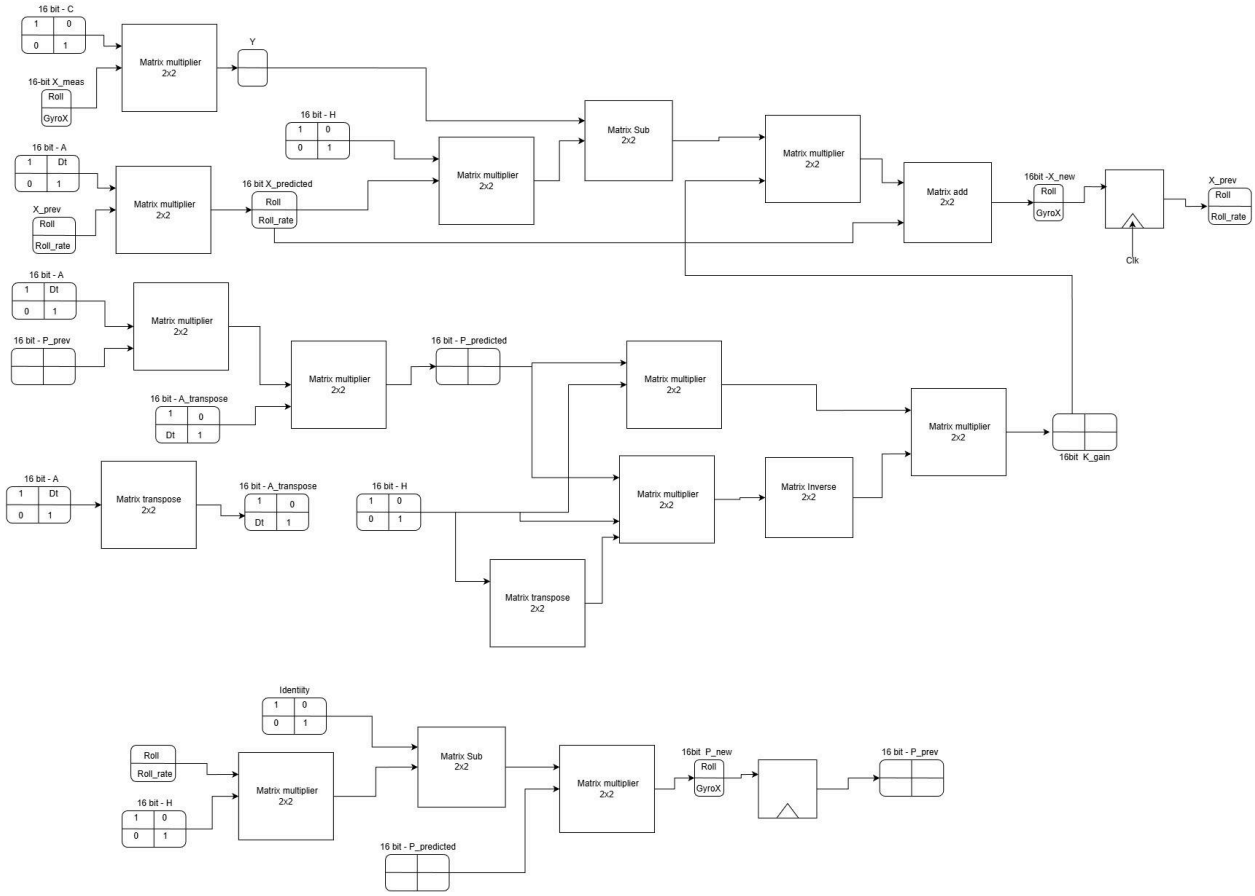

Fig5: Roll estimation Datapath

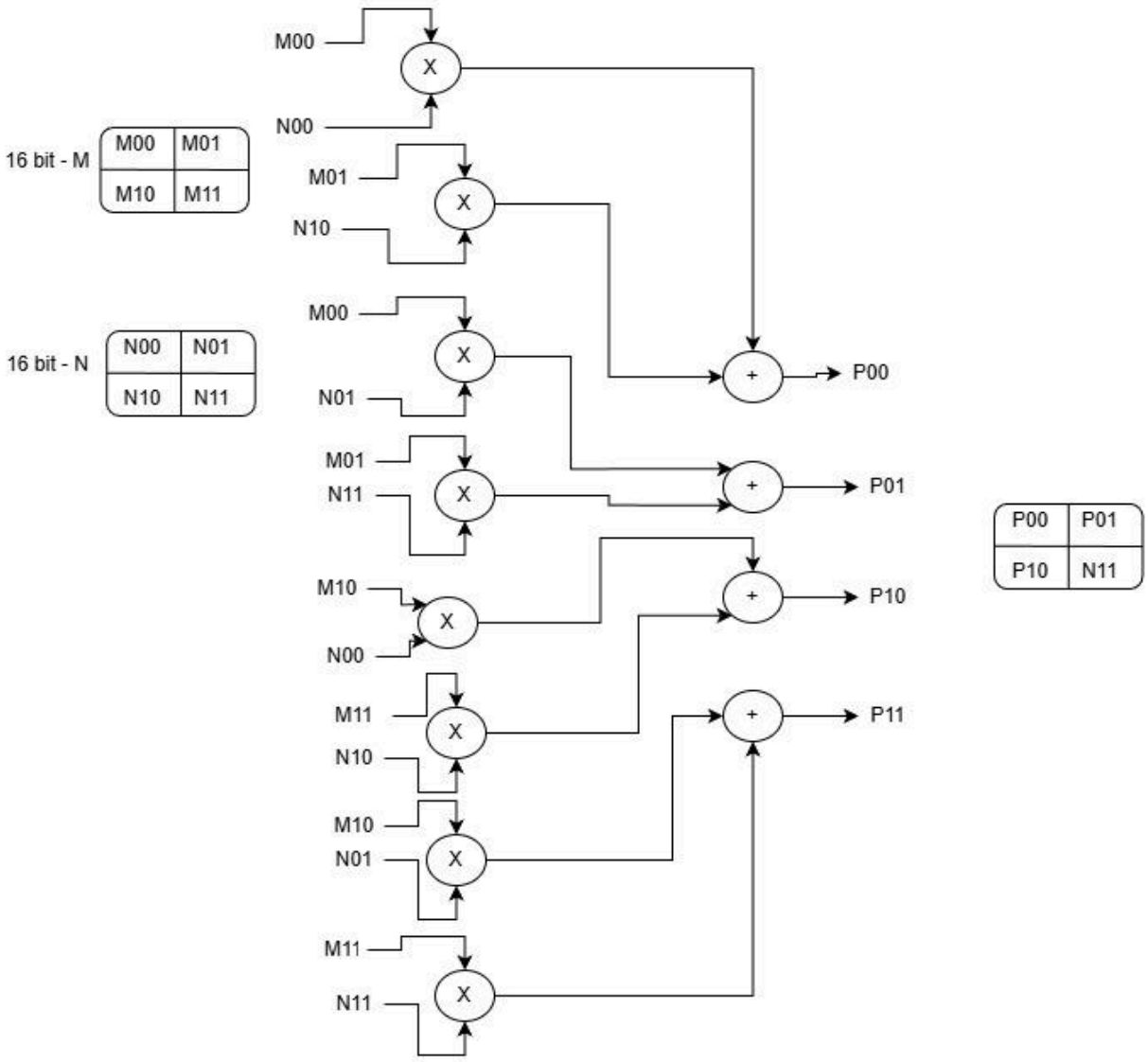

Fig6: Matrix multiplication datapath

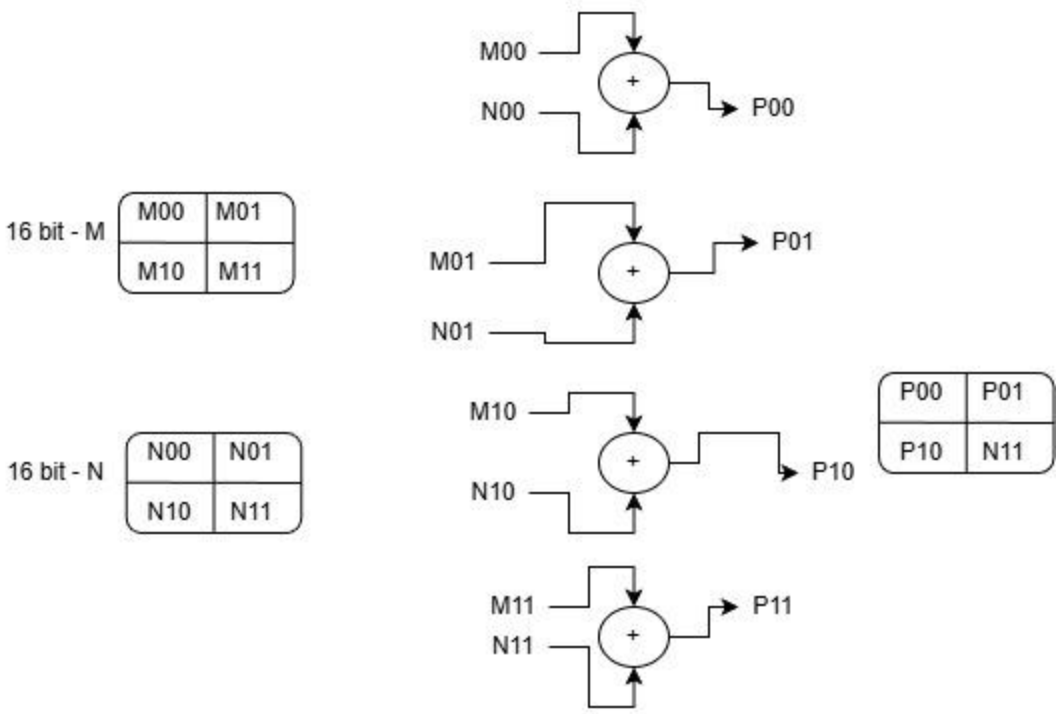

Fig7: Matrix multiplication datapath

c. VGA Display

The VGA output is managed by a standard SystemVerilog VGA controller to generate the relevant timing signals and raster scan for a 640x480 display. Instead of relying on a framebuffer to draw each frame, the display logic determines how to draw each segment of the central line by referring to a 1.2KB (480 * 10b * 2 / 8000) table stored in SRAM. This keeps the hardware / memory load minimal while maintaining a high refresh rate.

Display Generation

The VGA controller continuously produces the horizontal and vertical synchronization signals, along with the current pixel coordinates **(x,y)** for the active video region. During each frame, the FPGA steps through all 480 visible scanlines. For each scanline **y**, the controller consults a table containing two values: the left boundary $\mathbf{x_{start}[y]}$ and the right boundary $\mathbf{x_{stop}[y]}$. If the current pixel position satisfies $\mathbf{x_{start}[y] \leq x \leq x_{stop}[y]}$, the pixel is drawn in the foreground color (white); otherwise, it is drawn in the background color. Essentially, the displayed image is formed by a set of horizontal line segments, one for each visible row.

Motivation Behind Table-Based Line Rendering

This approach is well suited to our application because the shape being displayed is essentially a 2D line art profile that changes over time. Instead of storing and scanning a full pixel map, the FPGA only needs a small table of per-row segment boundaries. The line drawing algorithm in software computes these

boundaries ahead of time and writes them into memory, while the VGA logic simply reads them during display time. This significantly reduces memory usage and keeps the hardware implementation efficient.

Software and Hardware Interaction

After each frame, the software updates the table with the newest segment positions derived from the AHRS output. The new line positions are written to shared memory, and the FPGA uses them during the next display cycle. This creates a clean division of labor: the software handles the higher-level calculation of the line geometry, while the FPGA handles real-time pixel generation and VGA timing. To avoid visible tearing or partially updated frames, the table is updated once per frame rather than continuously during scanout. This ensures that each frame is drawn using a consistent set of line positions. The result is a stable, real-time visualization that reflects the latest orientation estimate produced by the AHRS pipeline.

VGA display interface

```
Python
module vga_display (
// Input Registers
input logic clk,
input logic reset,
input logic [15:0] writedata,
input logic [15:0] address,
input logic write,
input logic chipselect,

// Output Registers
output logic [7:0] VGA_R,
output logic [7:0] VGA_G,
output logic [7:0] VGA_B,
output logic VGA_HS,
output logic VGA_VS,
output logic VGA_CLK,
output logic VGA_BLANK_n,
output logic VGA_SYNC_n
);
```

4. Resource Budgets

Kalman filter

The main hardware constraints of this design are the arithmetic resources required by the Kalman filter core and the memory required for VGA rendering support structures. Since the HPS performs sensor interfacing and pre-processing, the FPGA only needs to support fixed-point Kalman estimation and real-time VGA generation.

Roll and pitch estimation are implemented in parallel. Each estimation block contains:

- 11 16-bit matrix multipliers
- 3 16-bit matrix adders

Each 16-bit matrix multiplier contains:

- 8 16-bit multipliers
- 4 16-bit adders

Each 16-bit matrix adder contains:

- 4 16-bit adders

Therefore, for one estimation path:

- Multipliers :88
- Adders :56

Kalman filter runs in parallel so it has- $2\times88=176$ 16-bit multipliers, $2\times56=112$ 16-bit adders, 14 16x4 bit registers, 5 16x2 bit registers

Memory Resource

The VGA does not use a framebuffer. Instead, it used a table-based line rendering approach. For each of the 480 visible scanlines, two 10-bit values are stored:

- xstart[y]
- xstop[y]

Thus, the required storage:

$480\times10\times2=9600$ bits

Which is: 9600/8 = 1200 bytes

In addition, roll_fx, pitch_fx, gx_fx,gy_fx,gz_fx are each 16 bits. The register storage overhead is minimal.

Timing Budget

The VGA output operates at 640×480 resolution at 60 Hz, which requires a frame period of approximately:$1/60\approx16.67$ ms

The HPS must update the input measurement data at a rate sufficient to support the specified refresh rate. Given that each update requires writing only four 16-bit values to the FPGA, the bandwidth requirements for the Avalon bus are very low.

On the FPGA side, Kalman filter estimation and VGA rendering are implemented in hardware; this enables real-time operation while maintaining stable display timing.

Overall Budget Summary

- 176 16-bit multipliers
- 112 16-bit adders
- 1200 byte SRAM/table memory for line rendering
- Minimal register storage
- 60 Hz real-time VGA timing requirement