Mud Viscosity Monitoring and Control System for HP Mud Pumps in Oil Drilling Using Zynq-7000 SoC

2024HT01603 SANKETH SAHA

Table of content

All related documents, files, images and video are in the Project Link: https://github.com/sanketh0798/Viscosity-Sensor-with-Zynq7000-and-DSP-RTL.git

1. Project Overview

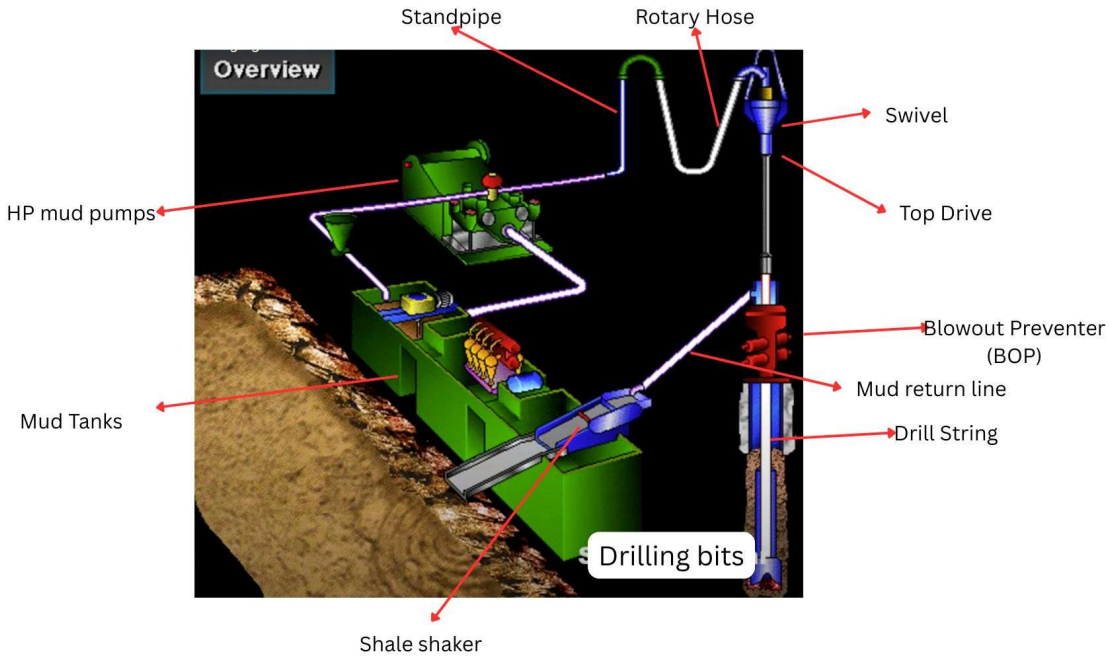
1.1 Introduction

The Mud Viscosity Monitoring and Control System is a specialized embedded system designed to accurately measure and process viscosity data from drilling mud in oil drilling operations. Utilizing Xilinx's Zynq-7000 System-on-Chip (SoC) architecture, this system combines the processing power of a dual-core ARM Cortex-A9 processor with programmable logic resources to perform real-time signal processing of viscosity sensor data.

1.2 Problem Statement

In oil drilling operations, the viscosity of drilling mud is a critical parameter that directly affects drilling efficiency and safety. Viscosity that is too high increases pumping pressure requirements and can lead to formation fracturing, while viscosity that is too low reduces the mud's ability to carry drill cuttings and maintain wellbore stability. There are two major Rig equipment involved during drilling, first the Top Drive and second the High Pressure Mud Pumps (HPMPs).

Functionality of Top Drive and HP Mud Pumps in Drilling Rigs



Component	Function	Interaction with Mud
Top Drive	Rotates drill string, allows mud passage	Mud flows through top drive into pipe
HP Mud Pump	Pressurizes and circulates mud	Moves mud from tanks to drill string

Interaction with Mud and Mud Processing

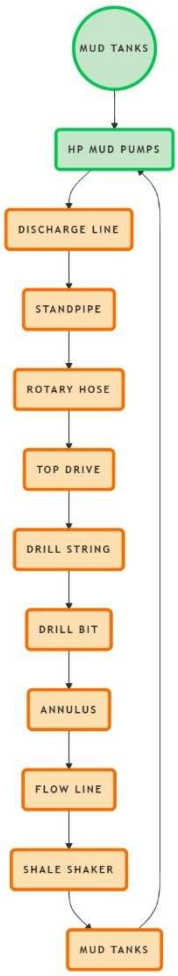
- **Mud Preparation and Storage:** Circulation starts in the mud tanks (also called pits), where crew members prepare and condition the mud for use.
- **Mud Pumps:** The heart of the system is the mud pump. Most rigs have at least two pumps-one primary and one backup. For deeper wells or higher volume needs, multiple pumps can be combined to circulate larger volumes of mud.
- **Mud Flow Path:**
    - The pump draws mud from the tanks and sends it through the discharge line, up the standpipe, and into the rotary hose.

- The rotary hose, a strong and flexible component, connects to the standpipe and moves with the swivel as the drilling assembly moves.
- Mud then passes through the swivel and down the Kelly and drill string (or through the top drive on rigs equipped with one).
- The mud travels down the drill string to the bit, where it jets out of nozzles to clear cuttings away from the bit.

Mud Circulation Flow

1. **Mud Preparation:** Mud is mixed and conditioned in surface mud tanks (pits).
2. **Pumping:** HP mud pumps draw mud from the tanks and send it under high pressure through the discharge line, up the standpipe, and into the rotary hose.
3. **Through the Top Drive:** On rigs equipped with a top drive, mud flows through a passage in the top drive and enters the drill string.
4. **Down the Drill String:** Mud travels down the drill string to the drill bit at the bottom of the well.
5. **At the Bit:** The mud jets out through nozzles in the bit, cooling and lubricating it, and carrying rock cuttings away from the drilling face.
6. **Up the Annulus:** Mud, now laden with cuttings, flows up the annular space between the drill string and the wellbore to the surface.
7. **Return and Processing:** The mud returns via the flow line to the shale shaker, where cuttings are removed. The cleaned mud is then returned to the mud tanks for recirculation. The mud, now carrying rock cuttings, moves up the annulus (the space between the drill string and the wellbore). It then flows through the return line (or flow line) to the shale shaker, which removes the cuttings from the mud. The cleaned mud returns to the mud tanks, ready to be recirculated by the pump, completing the loop.

The flow is:



Mud viscosity significantly affects the rate of penetration (ROP) during drilling, primarily by influencing the mechanical interaction between the drill bit, the rock formation, and the drilling fluid.

How Mud Viscosity Affects Rate of Penetration ROP

- **Higher Viscosity Reduces ROP:** Increasing the mud's plastic viscosity generally decreases the rate of penetration. This happens because thicker (higher viscosity) mud creates more mechanical friction and resistance around the drill bit and drill string, which leads to a "hold-down" effect that slows the bit's ability to break the rock efficiently.

- **Mechanical Friction and Hydraulic Effects:** High-viscosity mud increases the resistance to flow, which means more energy is required to pump it downhole. This reduces the hydraulic efficiency at the bit nozzles, impairing the bit’s cleaning and cooling action, and thus lowering drilling speed.

- **Cuttings Transport:** While some viscosity is necessary to carry cuttings out of the wellbore, excessive viscosity can hinder the flow and increase annular pressure losses, negatively impacting penetration rate.
- **Temperature and Mud Type Influence:** Mud viscosity decreases with increasing temperature, which can slightly improve ROP in deeper, hotter formations. Also, oil-based muds tend to have better rheological properties that can support higher ROP compared to water-based muds at similar viscosities.

Current scenario: Nowadays, in regular interval sample of the mud is being tested according to which the HPMP motor VFD parameters are modified. Manual measurements of mud viscosity are typically infrequent and susceptible to human error, resulting in suboptimal drilling performance.

1.3 Solution Overview

This project implements an automated system that:

1. Continuously measures drilling mud viscosity using analog sensors
2. Processes the sensor data using dedicated Digital Signal Processing (DSP) hardware
3. Outputs control signals to high-pressure mud pumps to optimize their operation
4. Enables real-time monitoring and adjustment of mud properties

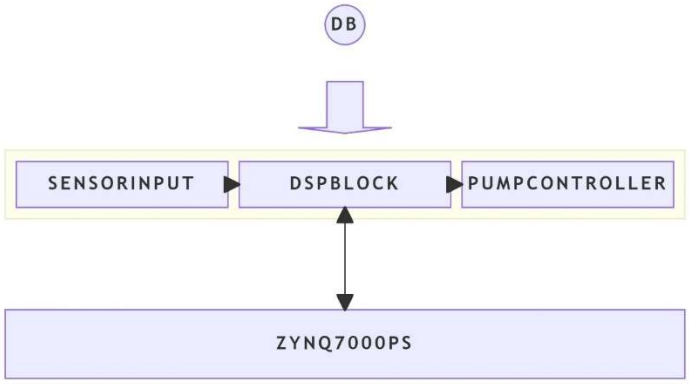
The system improves drilling performance by maintaining optimal mud viscosity, which directly increases the Rate of Penetration (ROP) while drilling, reducing overall drilling time and costs.

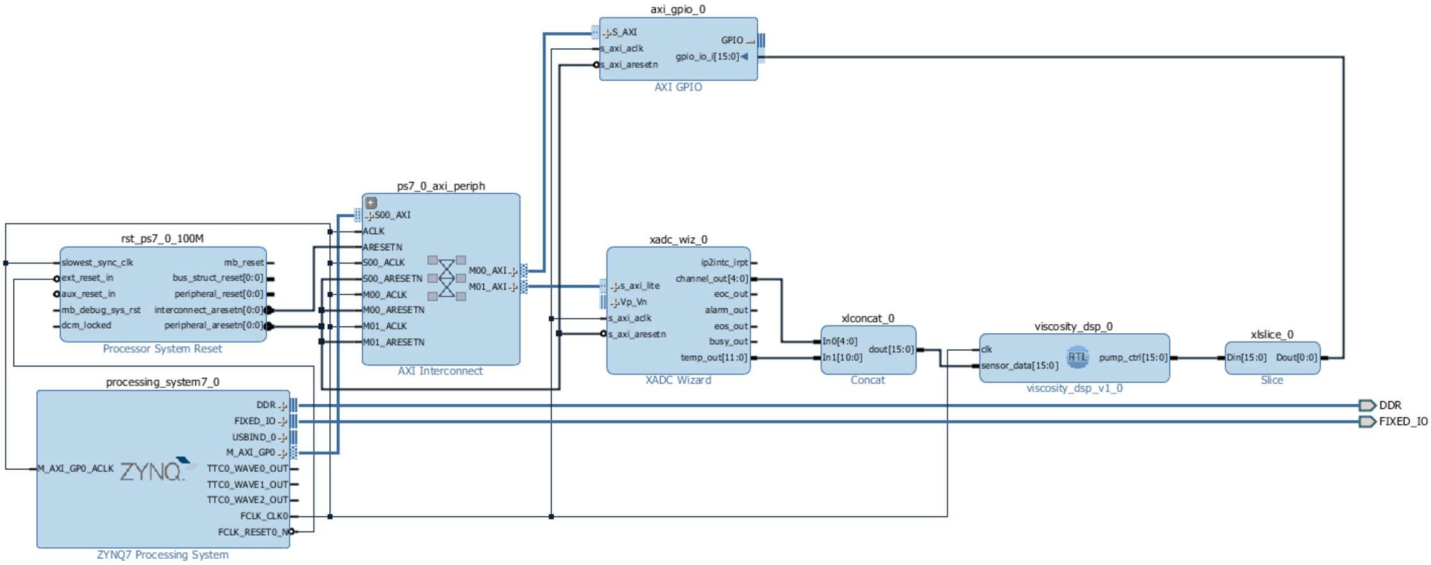
2. Detailed Implementation

2.1 System Architecture

The system is implemented using a Zynq-7000 SoC, which integrates a Processing System (PS) based on ARM Cortex-A9 with Programmable Logic (PL) based on 7-series FPGA fabric. This architecture enables a hybrid software/hardware approach where computationally intensive DSP operations are accelerated in hardware while system control is managed by software.

The block diagram shows the complete system implementation with key components:





1. **Zynq-7000 Processing System (processing_system7_0)** - Central control unit
2. **Reset Controller (rst_ps7_0_100M)** - Manages system reset signals
3. **AXI Interconnect (ps7_0_axi_periph)** - Communication infrastructure
4. **XADC Wizard (xadc_wiz_0)** - Analog-to-digital conversion for sensor inputs
5. **Custom DSP Module (viscosity_dsp_0)** - Signal processing for viscosity data
6. **AXI GPIO (axi_gpio_0)** - Digital output interface for pump control
7. **Slice Module (xslice_0)** - Bit extraction for control signal

2.2 Component Details

2.2.1 Zynq-7000 Processing System

The Zynq-7000 PS serves as the central control unit for the entire system. It provides:

- **Dual-core ARM Cortex-A9 processor** running at up to 866 MHz
- **Memory interfaces** for DDR3 and on-chip memory
- **Peripheral interfaces** including USB, Ethernet, and UART
- **AXI interfaces** for communication with the PL
- **Clock generation** for PL components (FCLK_CLK0 at 100 MHz)

The PS is configured with standard DDR and fixed I/O connections, and has M_AXI_GP0 enabled for communication with AXI peripherals in the PL[3]. The PS handles system initialization, configuration of the XADC and other peripherals, and provides a platform for potential user interface or data logging applications.

2.2.2 Reset Controller (rst_ps7_0_100M)

The reset controller synchronizes reset signals throughout the design. It:

- Receives the processor system reset signal
- Generates synchronized resets for peripherals
- Ensures proper power-up sequencing
- Provides bus_struct_reset, peripheral_reset, and other reset signals

This component is critical for reliable operation, ensuring all components start in a known state and preventing race conditions during power-up or system resets.

2.2.3 AXI Interconnect (ps7_0_axi_periph)

The AXI Interconnect implements the Advanced eXtensible Interface (AXI) protocol, providing a standardized communication framework between the PS and peripherals in the PL. This component:

- Routes transactions between master (PS) and slaves (XADC, GPIO)
- Handles address decoding and transaction control
- Supports multiple master and slave interfaces

- Manages clock domain crossing between PS and PL

The interconnect is configured with one master port (connected to M_AXI_GP0 of the PS) and two slave ports (M00_AXI and M01_AXI) connected to the XADC Wizard and AXI GPIO respectively.

2.2.4 XADC Wizard (xadc_wiz_0)

The XADC Wizard encapsulates the functionality of the Zynq-7000's integrated Analog-to-Digital Converter. This component:

- Provides 12-bit resolution ADC functionality
- Supports up to 17 external analog inputs
- Offers built-in temperature and voltage monitoring
- Delivers converted data via dedicated channel_out[4:0] ports
- Connects to the AXI bus for configuration and data access

The XADC is configured to sample the viscosity sensor data, which is likely connected to one of the auxiliary analog inputs (Vp_Vn). The converted data is output on the channel_out port, which is connected to the viscosity_dsp_0 module.

2.2.5 Custom DSP Module (viscosity_dsp_0)

The viscosity_dsp_0 module implements the digital signal processing algorithms for viscosity data analysis. This custom module:

- Receives 16-bit sensor data from the XADC
- Applies signal processing techniques to filter and analyze the data
- Outputs a 16-bit control value (pump_ctrl) for the mud pumps

The implementation leverages the DSP48E1 slices available in the Zynq-7000 PL. These are dedicated hardware blocks optimized for digital signal processing that include:

- 25×18 two's complement multiplier/accumulator
- 48-bit signal processor capabilities
- Power-saving pre-adder for filter applications
- Optional pipelining and ALU functionality

The RTL code for the viscosity_dsp module includes:

```
module viscosity_dsp(
    input wire clk,
    input wire [15:0] sensor_data,  // 16-bit ADC input
    output reg [15:0] pump_ctrl     // Processed output
);
    // DSP48E1 Configuration: P = A*B + C
    wire [29:0] dsp_out;

    DSP48E1 #(
        .USE_MULT("MULTIPLY"),
        .USE_SIMD("ONE48")
    ) dsp_inst (
        .CLK(clk),
        .A({14'b0, sensor_data}),  // 30-bit input (zero-padded)
        .B(30'h0000_2000),         // Fixed coefficient (adjust for
scaling)
        .C(30'd0),
        .P(dsp_out)
    );

    // Post-processing with pipeline registers
    always @(posedge clk) begin
        pump_ctrl <= dsp_out[25:10];  // Scale to 16 bits
    end
endmodule
```

The actual implementation would likely include more sophisticated processing, potentially using moving averages, Kalman filtering, or frequency domain analysis to accurately determine viscosity trends while rejecting noise.

2.2.6 AXI GPIO (axi_gpio_0)

The AXI GPIO provides general-purpose input/output functionality with an AXI bus interface. In this design, it:

- Connects to the AXI Interconnect as a slave device
- Provides a 16-bit GPIO interface
- Receives processed control data for the mud pumps
- Can be configured and controlled by software running on the PS

The GPIO is configured as an output that transmits the pump control signals from the PL to external pump control hardware.

2.2.7 Slice Module (xslice_0)

The Slice module performs bit extraction from wider signals. In this design, it:

- Takes the 16-bit pump_ctrl output from the DSP module
- Extracts a specific bit or range of bits for simplified control
- Outputs a reduced-width signal suitable for the pump controller

This component allows the system to convert the multi-bit DSP output into a simpler control signal that may be required by the mud pump controller hardware.

2.3 Signal Flow and Connections

The signal flow through the system follows this path:

1. **Analog viscosity sensor → XADC Wizard**
    - The analog viscosity sensor connects to the Vp_Vn differential inputs of the XADC
    - The XADC converts the analog signal to a 12-bit digital value
    - This digital value is output on the channel_out[4:0] port
2. **XADC Wizard → viscosity_dsp_0**
    - The channel_out signal from the XADC connects to the sensor_data[15:0] input of the viscosity_dsp_0 module
    - The width difference is handled through appropriate sign extension or zero-padding
3. **viscosity_dsp_0 → xslice_0**
    - The pump_ctrl[15:0] output from the viscosity_dsp_0 connects to the Din[15:0] input of the xslice_0
    - The xslice_0 extracts a subset of the bits (in this case, just bit 0)
4. **xslice_0 → AXI GPIO**
    - The Dout output from xslice_0 connects to the gpio_io_i[15:0] input of the AXI GPIO
    - The GPIO configuration determines how this signal is processed and output
5. **AXI GPIO → External Pump Control Hardware**
    - The GPIO output connects to external mud pump control circuitry
    - This could be a direct digital control or an input to additional control hardware

2.4 Software Implementation

The software running on the Zynq PS complements the hardware implementation by providing configuration, monitoring, and control functionality. SDK/Vitis code should Read the processed output from the DSP (via GPIO, or memory-mapped register if exposed). Use AXI GPIO to output the control signal to the pump (likely just a single bit if using xSlice).

```
#include "xparameters.h"
#include "xgpio.h"
#include "sleep.h"

// Replace with the actual device ID or use LookupConfig if
required by your Vitis version
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define GPIO_CHANNEL    1  // Channel 1 is usually output

int main() {
    int status;
    XGpio Gpio;

    // Initialize the GPIO driver
    status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
    if (status != XST_SUCCESS) {
        xil_printf("GPIO Initialization Failed\r\n");
        return XST_FAILURE;
    }

    // Set the direction for all signals to outputs
    XGpio_SetDataDirection(&Gpio, GPIO_CHANNEL, 0x0);

    // Main control loop
    while (1) {
        // Rread the processed value from DSP.
        // Since the DSP output is directly wired to the GPIO, just
toggle or set the GPIO pin.
        // For demonstration, we'll toggle the output every second.

        // Example: Set pump ON (assuming active-high logic)
        XGpio_DiscreteWrite(&Gpio, GPIO_CHANNEL, 1);
        sleep(1); // 1 second ON

        // Example: Set pump OFF
        XGpio_DiscreteWrite(&Gpio, GPIO_CHANNEL, 0);
        sleep(1); // 1 second OFF

        // In real application, replace the above with logic to set
the GPIO
        // based on the processed viscosity value, e.g.:
        // uint32_t pump_ctrl_value = ... // get from DSP or
memory-mapped register
        // XGpio_DiscreteWrite(&Gpio, GPIO_CHANNEL, pump_ctrl_value
& 0x1);
    }

    return 0;
}
```

2.5 Clocking and Timing Considerations

The system is clocked by the FCLK_CLK0 output from the PS, operating at 100 MHz (10ns period). This clock drives the AXI interconnect, XADC Wizard, custom DSP module, and other PL components. The timing constraints must ensure that all paths meet timing requirements at this clock frequency.

2.6 Implementation Rationale

The choice of components in this design is driven by several key considerations:

1. **Zynq-7000 SoC**: Selected for its integration of powerful ARM processing with FPGA fabric, allowing real-time processing and flexibility in implementation[2].
2. **XADC Wizard**: Chosen over external ADCs because it provides integrated, high-resolution analog acquisition capabilities without requiring additional components[4].
3. **Custom DSP Module**: Implemented in the PL to leverage DSP48E1 slices for high-performance signal processing with minimal latency[5].
4. **Slice and GPIO**: Provides flexible interfacing options for controlling external pump hardware with different control signal requirements.
5. **AXI Interconnect**: Facilitates standardized communication between components, simplifying system integration and enabling software control.

2.7 Testbench simulation code

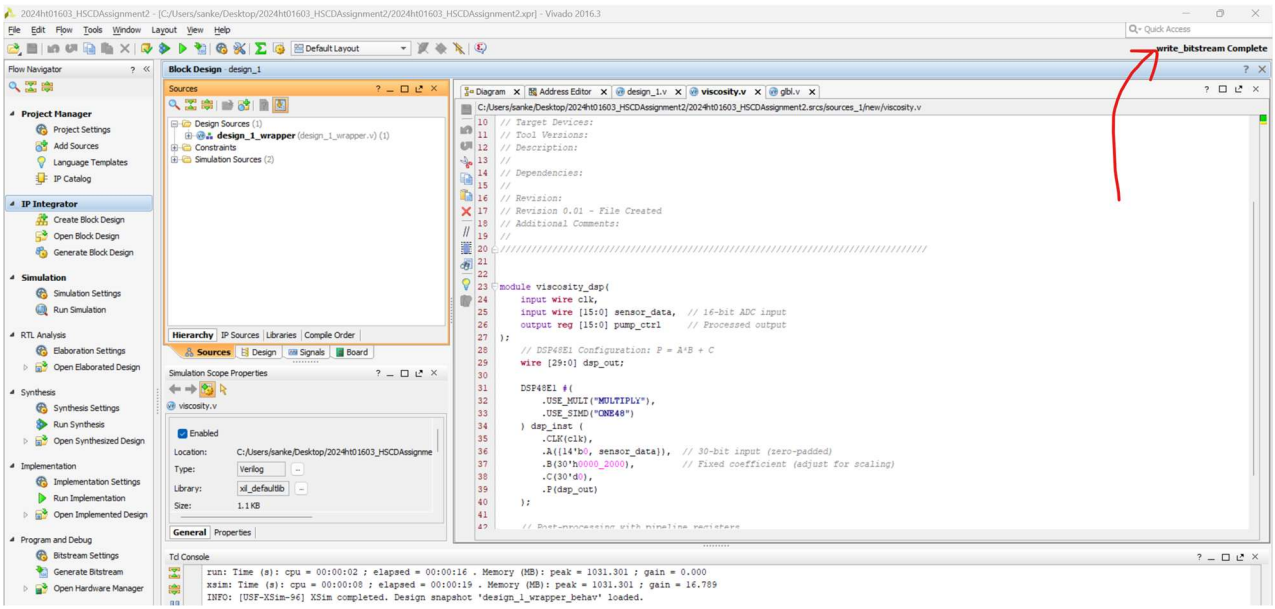
Below is the comprehensive testbench for Zynq-based viscosity processing system, focusing on the viscosity_dsp module and its integration with the sensor input and pump control logic.

```
`timescale 1ns / 1ps

module tb_viscosity_system();

// Parameters
parameter CLK_PERIOD = 10;  // 100 MHz clock (10 ns period)

// Signals
reg clk;
reg resetn;
reg [15:0] sensor_data;
wire [15:0] pump_ctrl;
```

```
// Instantiate DUT (viscosity_dsp module)
viscosity_dsp dut (
    .clk(clk),
    .sensor_data(sensor_data),
    .pump_ctrl(pump_ctrl)
);

// Clock generation
initial begin
    clk = 0;
    forever #(CLK_PERIOD/2) clk = ~clk;
end

// Reset and stimulus
initial begin
    // Initialize
    resetn = 0;
    sensor_data = 16'h0000;

    // Release reset
    #100 resetn = 1;

    // Test Case 1: Normal viscosity (mid-range)
    sensor_data = 16'h4000;
    #200;

    // Test Case 2: High viscosity
    sensor_data = 16'hC000;
    #200;

    // Test Case 3: Low viscosity
    sensor_data = 16'h1000;
    #200;

    // Test Case 4: Step response
    for (integer i=0; i<16; i=i+1) begin
        sensor_data = i << 12;
        #50;
    end

    // End simulation
    #100;
    $finish;
end

// Monitor results
initial begin
    $timeformat(-9, 2, " ns", 10);
    $monitor("At time %t: Sensor=0x%h Pump_Ctrl=0x%h",
             $time, sensor_data, pump_ctrl);
end

endmodule
```

2.8 Output snippets

Bitstream generated



3. Future Upgrade Options

3.1 Using customIP and BRAM, DMA etc

NOTE: I had started with a plan to implement this project with customIP for processing, BRAM, FIFO core, high speed AXI stream but it became way too complicated for me to complete, thus shifted to simpler version of this to just demonstrate the PS-DSP transfer which is implemented above.

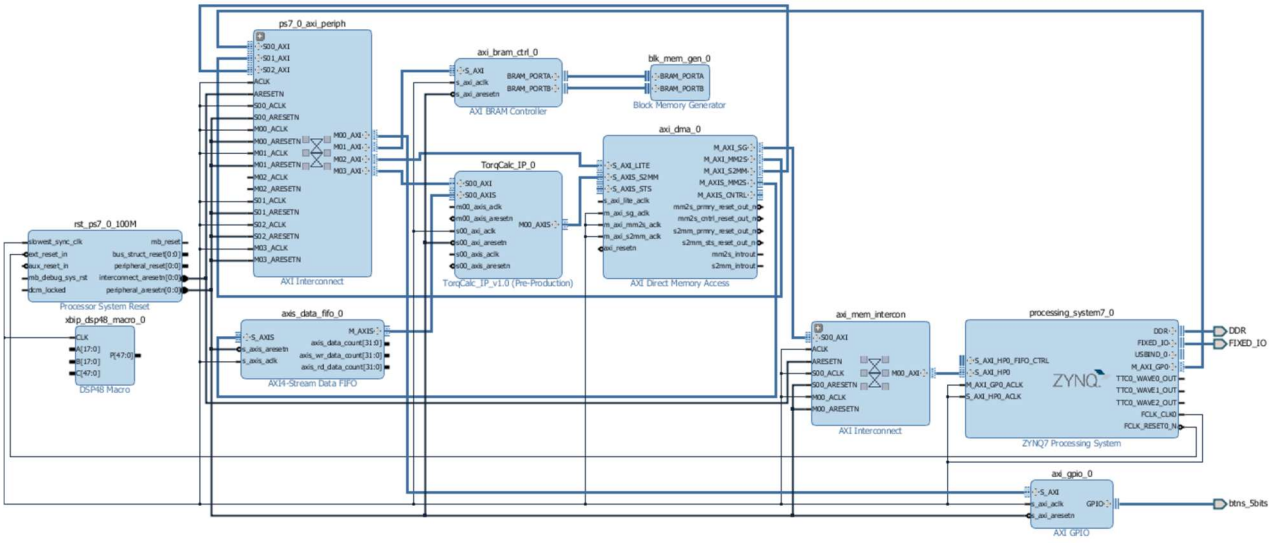
Add Custom Peripherals in PL:

- Add AXI GPIO for external sensors/actuators.
- Add BRAM Controller and BRAM for fast-access data storage (e.g., signal buffers).
- Add DSP48 slices for real-time mathematical operations (e.g., filtering, FFT).

- Optionally, add AXI Stream interfaces for high-speed data transfer between PS and PL

Functionality	**IP Name in Vivado IP Catalog**	**Category / Location**
Fast-access BRAM	Block Memory Generator	Memory & Storage Elements
AXI Interface for BRAM	AXI BRAM Controller	Processor System / Interconnect
AXI Stream FIFO	AXI4-Stream FIFO	Communication and Networking
AXI Stream Interface	Custom IP or AXI Interconnect	Search by "AXI Stream"

Block diagram for the main system:



TorqCalc_IP is the customIP to process the signal and find the required torque. The codes of this IP which includes the following.

File	Purpose
TorqCalc_IP_v1_0.v	Top-level wrapper with interface declarations
TorqCalc_IP_v1_0_S00_AXI.v	AXI4-Lite control interface
TorqCalc_IP_v1_0_S00_AXIS.v	AXI4-Stream Slave (input data processing). Calculates torque and dsp instantiation
TorqCalc_IP_v1_0_M00_AXIS.v	AXI4-Stream Master (output data streaming). Holds the logic to send processed data downstream.

The codes are uploaded in the github link, please refer https://github.com/sanketh0798/FPGA-Based-Real-Time-Torque-Calculation-Accelerator.git.

In future, outside the scope of this assignment, I would like to proceed with this project to completion, please help me with this. My BITS ID – 2024HT01603.