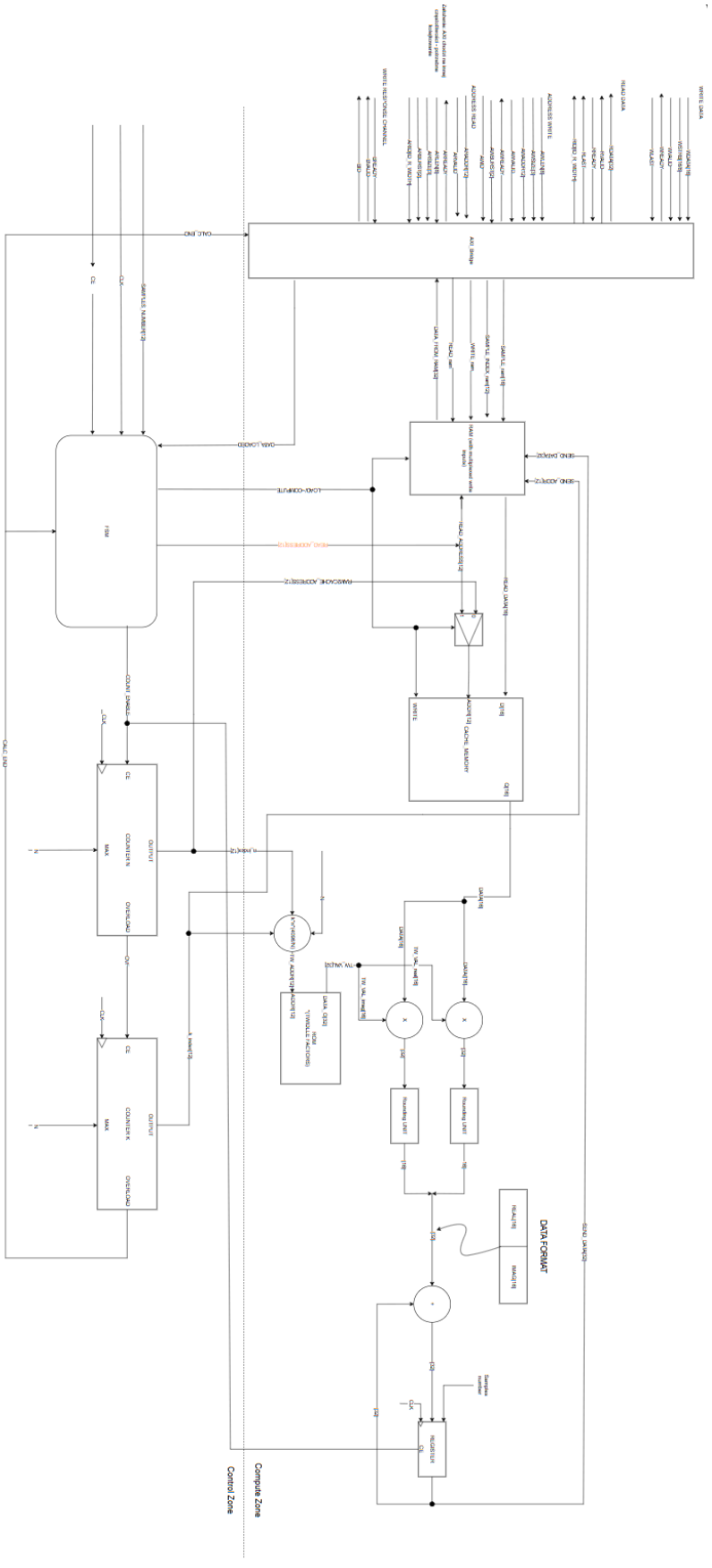
1. Diagram of MAC unit



*In case of need in other file, MAC UNIT.drawio.svg there are same diagram in better quality and possibility to easy zoom and maneuver in picture

2. Signal tables for every module.

1.1. AXI_Bidge

Signals	Width	Direction		Description	Clock domain
AWADDR	12	IN		Indicates the first memory cell in burst data write.	
AWVALID	1	IN		When set to '1', master has valid address to write.	
AWREADY	1	OUT		Set to '1' when address may be written to slave.	
AWBURST	2	IN		This signal indicates burst type. We use INCR type, it means signal ARBUST set to 2'b01.	
AWLEN	8	IN		Specifes number of tranfers in current transaction.	
AWSIZE	3	IN		This signal codes number of bytes of every transfer.	
ARID	ID_W_WIDTH	IN		This signal gives an unique ID for the transaction.	
ARADDR	12	IN		Indicates the first memory cell in burst data read.	
ARVALID	1	IN		When set to '1', master has valid address to write.	
ARREADY	1	OUT		Set to '1' when address may be written to slave.	
ARBURST	2	IN		This signal indicates burst type. We use INCR type, it means signal ARBUST set to 2'b01.	
ARLEN	8	IN		Specifes number of tranfers in current transaction.	
ARSIZE	3	IN		This signal codes number of bytes of every transfer.	
ARID	ID_R_WIDTH	IN		This signal indicates ID of transaction to be read.	
WDATA	16	IN		Data to be written do RAM.	
WSTRB	2	IN		Indicates valid bytes in transaction.	
WVALID	1	IN		Set to '1' when master has valid data to send.	

WREADY	1	OUT		Set to '1' when slave could recieve data.	
WLAST	1	IN		When set to '1' last word is being sent.	
RDATA	32	OUT		Data to be read from RAM. (it's 32 bit sample in frequency domain)	
RVALID	1	OUT		Set to '1' when RAM has valid data.	
RREADY	1	IN		Set to '1' when master could reaad	
RLAST	1	OUT		Set to '1' when last word is being sent.	
RID	ID_R_WIDTH	OUT		ID of current transaction being read from RAM.	
BREADY	1	IN		Set to '1' when master could read ID of sent transaction	
BVALID	1	OUT		Set to '1' when slave could send ID of recieved transaction.	
BID	ID_W_WIDTH	OUT		ID of already recieved transaction, will be send back to master	
SAMPLE_ram	16	OUT		Current sample to be write to RAM.	
SAMPLE_INDEX_ram	12	OUT		Address of current sample to be read to RAM.	
WRITE_ram	1	OUT		When set to '1', means that sample is being write to memory cell addressed by SAMPLE_INDEX_ram.	
READ_ram	1	OUT		When set to '1' means that data is being read from RAM	
DATA_FROM_RAM	32	IN		Sample from RAM.	
DATA_LOADED	1	OUT		Set to '1' when last sample is being loaded.	
CALC_END	1	IN		Set to '1' when whole fft has been written to RAM.	

1.2. RAM

Signals	Width	Direction		Description
SAMPLE_ram	16	IN		Current sample to be read to RAM.
SAMPLE_INDEX_ram	12	IN		Address of current sample to be read to RAM.
WRITE_ram	1	IN		When set to '1', means that RAM writes SAMPLE to memory cell addressed by SAMPLE_INDEX.
READ_ram	1	IN		When set to '1' means that data is being read from RAM.
SEND_DATA	32	IN		Copmuted data is to be stored in RAM.
SEND_ADDRESS	12	IN		Address of memory cell which it will be written SEND_DATA
LOAD/~COMPUTE	1	IN		If '1' data is being loaded from RAM to CACHE. When '0', signalizes that SEND_DATA is being written to RAM
READ_DATA	16	OUT		Data to be stored in CACHE.
READ_ADDRESS	12	OUT		Address of READ_DATA in CACHE.

1.3. FSM (finite state machine).

Signals	Width	Direction		Description	Clock domain
CE	1	IN		Enables clock.	
SAMPLES_NUMBER	12	IN		Specifes number of input samples.	
END_SEND	1	IN		Seto to '1' when data is send to cache memory.	
START_SEND	1	OUT		Set to '1' when data is going to be sending to RAM	
DATA_NUMBER	12	OUT		Number of input samples	
START_READ	1	OUT		Set to '1' when ARDATA is to be read	
LOAD/~COMPUTE	1	OUT		This signal addresses MUX, when set to '1' there is load data to cache memory. When '0' data from cache is being read.	
CALC_END	1	IN		When set to '1' means that calculation of fft has been ended.	
DATA_LOADED	1	IN		When set to '1' means that data has been loaded to RAM.	

1.4. Cache memory.

Signals	Width	Direction	Description	Clock domain
READ_DATA	32	IN	Actually taken sample from RAM given to the output.	
READ_ADDRESS	12	IN	Address of actually read sample.	
WRITE_ADDRESS	12	IN	Address of actually written data.	
DATA	32	OUT	Data stored to cache memory.	

1.5. ROM

Signals	Width	Direction	Description	Clock domain
TW_ADDR	12	IN	Address of Twiddle Factor.	
TW_VAL	32	OUT	Value of Twiddle Factor.	

1.6. Rounding unit.

Signals	Width	Direction	Description	Clock domain
UNROUNDED_SAMPLE	52	IN	Sample multiplied by Twiddle Factor, frequency domain	
ROUNDED_SAMPLE	32	OUT	Rounded sample in frequency domain	

1.7. Accumulation_unit

Signals	Width	Direction	Description	Clock domain
ROUNDED_SAMPLE	32	IN	Rounded sample in frequency domain	
SEND_DATA	32	OUT	Just buffered ROUNDED_SAMPLE.	
samp_number	12	IN	Data needed to samples scaling	

1.8. Counter_n.

Signals	Width	Direction	Description	Clock domain
N	12	IN	Number of samples, specifes max. value of counter	
COUNT_ENABLE	1	IN	If '1', counter counts.	
n_index	12	OUT	Current state of counter used as sample index.	
OVF	1	OUT	When counter reaches max. value, OVF is set to'1' for one CLK edge.	

1.9. Counter_k.

Signals	Width	Direction	Description	Clock domain
N	12	IN	Number of samples, specifes max. value of counter	
OVF	1	IN	If '1', counter counts.	
k_index	12	OUT	Current state of counter used as frequency index.	
CALC_END	1	OUT	When counter reaches max. value, CALC_END is set to '1', it means that last sample is computed	

3. Short description of each module.

3.1. AXI_Bridge

This module allows communication between RAM in MAC and external peripherals using AXI magistral. Includes all necessary canals.

3.2. RAM

Random access memory that stores samples of signal. It may be input samples in time domain or ready samples in frequency domain.

3.3. FSM (finite state machine)

This module controls behaviour of whole MAC circuit. Controls individual modules depends of current state of MAC.

3.4. Cache memory

This memory gets input samples and replaces them with new samples in frequency domain.

3.5. ROM

This memory keeps ready twiddle factors to calculate fft, only to read.

3.6. Rounding unit

Rounds a sample after multiplication.

3.7. Accumulation_unit

Keeps accumulated sample and scaling them to keep acceptable precision.

3.8. Counter_n

Counts number of accumulation in each sample.

3.9. Counter_k

Counts number of samples.

Test Plan

1. Introduction

1.1. Purpose

This document outlines the functional verification strategy for the AXI Lite interface and MAC (Multiply-Accumulate) unit. The goal is to validate:

- Correct data transactions via AXI Lite.
- Correct computational results in the MAC unit, using an FFT reference model.

1.2. Scope

The test plan focuses on:

- Unit-level verification of AXI Lite transactions and MAC computations.
- Comparing DUT results with expected values using a reference model.
- Functional correctness without considering timing constraints.

2. Test Environment

2.1. Verification Components

Component	Function
AXI Lite Driver (AxiLiteDriver)	Writes and reads data via AXI Lite.
AXI Lite Monitor (AxiLiteMonitor)	Observes and logs AXI transactions.
Data Generator (Generator)	Creates test input values (fixed-point numbers).
Scoreboard (Scoreboard and FftRadix4Scoreboard)	Compares DUT output with reference values.

2.2. Testbench Structure

```
verif/
├── driver.py # Handles AXI transactions
├── generator.py # Generates test data
├── monitor.py # Monitors AXI bus activity
├── scoreboard.py # Compares DUT output with expected results
├── Makefile #File contains parameters to execute cocotb tests
tests/
├── test_axi_lite.py # Tests AXI Lite read/write transactions
├── test_mac.py # Tests MAC functionality against an FFT model
```

2.3. Simulation Setup

- Testbench is implemented in Python (cocotb).
- DUT (Design Under Test) is written in Verilog/SystemVerilog.
- Simulation tool: Icarus Verilog.

3. Test Cases

3.1. AXI Functional Test

3.1.1. Objective

Verify that AXI Lite transactions (write and read) work correctly.

Test Step	Action
1	Generate random test data (Generator.generate()).
2	Write data to DUT memory using AxiLiteDriver.write().
3	Read back data from DUT using AxiLiteDriver.read().
4	Compare read data with written data (assert data1 == data_to_write).

3.1.2. Pass Condition

If all read data matches written data.

3.1.3. Fail Condition

If read data differs from written data.

3.2. MAC Functional Test

3.2.1. Objective

Validate MAC computation using an FFT reference model.

Test Step	Action
1	Generate test input data (Generator.generate()).
2	Write test data to DUT (AxiLiteDriver.write()).
3	Start MAC computation (MAC_nRADIX signal is set).
4	Read DUT output using AxiLiteDriver.read().
5	Compare DUT output with the FFT reference model (FftRadix4Scoreboard.compare_to_dut_output()).
6	Compute error metrics (MSE, MAE, max error).

3.2.2. Pass Condition

If MSE (Mean Squared Error) < 0.002 (by default).

3.2.3. Fail Condition

If MSE ≥ 0.002 (by default).

4. Test Execution

4.1. Running Tests

4.1.1. Change Directory

```
cd ./FFT/verif
```

4.1.2. Creating Python Virtual Environment

```
python3 -m venv myenv
source myenv/bin/activate
```

4.1.3. Running Test

```
make
```

4.1.4. Changing Test

To run the selected test, uncomment the code snippet in Makefile and comment out the code snippet that runs the other test.

4.1.4.1. test_mac.py

```
TOPLEVEL = top_fft_tb
MODULE = test_mac
```

4.1.4.2. test_axi_lite.py

```
TOPLEVEL = top_axi_lite_tb
MODULE = test_axi_lite
```

5. Reporting and Analysis

5.1. Log Collection

- AXI transactions are logged by AxiLiteMonitor.
- DUT output vs reference model errors are logged by FftRadix4Scoreboard.

5.2. Visualization

A plot (fft_comparison.png) is generated for MAC error analysis:

- DUT vs Reference FFT Magnitude
- Error Magnitude Plot

Testbench

1. AxiLiteDriver (verif/driver.py)

1.1. Brief Description

The Driver (AxiLiteDriver) is responsible for directly interacting with the AXI bus, writing and reading data.

1.2. Initialization

1.2.1. Stores a reference to the DUT and the clock.

1.2.2. Prepares the driver for AXI transactions.

1.3. Write Operation

1.3.1. Writes a burst of data to a specified AXI address.

1.3.2. Controls the AWVALID, AWREADY, WDATA, WSTRB, and WLAST signals.

1.3.3. Waits for AWREADY before sending data.

1.3.4. Sends data in bursts, waiting for WREADY.

1.3.5. Handles BVALID signaling to complete the write transaction.

1.4. Read Operation

1.4.1. Reads a burst of data from a given AXI address.

1.4.2. Controls the ARVALID, ARREADY, and RDATA signals.

1.4.3. Waits for ARREADY before receiving data.

1.4.4. Collects received data while RVALID is asserted.

2. Generator (verif/generator.py)

2.1. Brief Description

The Generator creates fixed-point test values, simulating real-world data input.

2.2. Initialization

2.2.1. Optionally seeds the random number generator for reproducibility.

2.3. Data Generation

2.3.1. Generates a list of random floating-point values in the range [-1, 1].

2.3.2. Converts the floating-point values into signed 16-bit fixed-point numbers.

2.3.3. Handles negative values correctly (ensures compatibility with signed binary representation).

2.3.4. Returns the generated fixed-point values.

3. AxiLiteMonitor (verif/monitor.py)

3.1. Brief Description

The Monitor (AxiLiteMonitor) passively listens to transactions, logging any AXI activity.

3.2. Initialization

3.2.1. Stores a reference to dut and clk.

3.2.2. Initializes an empty list transactions to store observed read/write operations.

3.3. Monitoring Loop

3.3.1. Continuously checks the AXI bus for transactions.

3.3.2. Detects and logs read transactions when RVALID and RREADY are asserted.

3.3.3. Appends read data to the transactions list.

3.3.4. Outputs debug information about observed transactions.

4. Scoreboard (verif/scoreboard.py)

4.1. Brief Description

The Scoreboard checks whether the outputs match expected values, acting as the primary validation tool.

4.2. Initialization

4.2.1. Stores an empty list expected for expected results.

4.2.2. Initializes an errors counter.

4.3. Adding Expected Data

4.3.1. Stores expected reference values.

4.4. Checking DUT Output

4.4.1. Compares DUT output against expected values.

4.4.2. Logs any mismatches and counts errors.

4.4.3. Reports whether all outputs are correct.

5. FftRadix4Scoreboard (verif/scoreboard.py)

5.1. Brief Description

The FFT Scoreboard performs more advanced validation, comparing FFT computations to a reference implementation.

5.2. Initialization

5.2.1. Defines num_samples (default: 4096).

5.2.2. Initializes lists for input samples and expected FFT outputs.

5.2.3. Sets up counters for bit error rate (BER) analysis.

5.3. Adding Input Data

5.3.1. Stores the input samples for FFT computation.

5.3.2. Issues a warning if the number of samples does not match the expected size.

5.4. Running the Reference Model

5.4.1. Converts signed 16-bit fixed-point numbers into floating-point values.

5.4.2. Computes the FFT using NumPy (np.fft.fft).

5.4.3. Stores the expected FFT output.

5.5. Comparing DUT Output

5.5.1. Extracts real and imaginary parts from DUT's output.

5.5.2. Converts them back to floating-point representation.

5.5.3. Computes error metrics:

- Mean Squared Error (MSE).
- Mean Absolute Error (MAE).
- Maximum Error.

5.5.4. Plots DUT vs Reference comparison:

- Magnitude comparison.
- Error magnitude.

Tests

1. test_axi_lity.py

1.1. Brief Description

1.1.1. The test_axi_lite verifies that written data in AXI is correctly read back. If any discrepancy exists between write and read data, the test fails. It does not use a reference model - it simply compares input and output.

1.2. Parameters:

1.2.1. `__input_samp_num__` - Number of samples that will be written and saved.

1.2.2. `__input_burst_num__` - Number of samples in one burst, that is the amount of written data in one Axi packet.

1.3. Classes

1.3.1. `AxiLiteDriver` (from `verif.driver`) - Axi interface driver, are used to write/read operations.

1.3.2. `AxiLiteMonitor` (from `verif.monitor`) - Axi monitor driver, is monitoring Axi transmissions and log data flow.

1.3.3. `Generator` (from `verif.generator`) - Generator of random test data.

1.4. Test Description

1.4.1. Test initialization

- Clock setup
- Creating driver and monitor objects
- Generating test data

1.4.2. Writing Data via Axi

- Data in split into packets
- Iteratively sent via the driver

1.4.3. Reading Data via Axi

- Reads data form the same addresses used for writing
- Stores retrieved data in data1

1.4.4. Comparing Results

- If read data differs from written data, the test fails

1.4.5. End of the test

1.5. Pass/Fail conditions

1.5.1. Pass

All read data matches exactly the written data

1.5.2. Fail

Mismatch between read and written data

2. test_mac.py

2.1. Brief Description

2.1.1. The test_mac verifies MAC computation accuracy by comparing DUT results to a reference model.
It uses `FftRadix4Scoreboard`, which runs a reference model and checks numerical errors.
Result of the test is based on the mean squared error.

2.2. Parameters

2.2.1. `__input_samp_num__` - Number of input samples.

2.2.2. `__input_burst_num__` - Number of samples in single burst.

2.3. Classes

2.3.1. `AxiLiteDriver` (from `verif.driver`) - Axi interface driver, are used to write/read operations.

2.3.2. `AxiLiteMonitor` (from `verif.monitor`) - Axi monitor driver, is monitoring Axi transmissions and log data flow.

2.3.3. `Generator` (from `verif.generator`) - Generator of random test data.

2.3.4. `FftRadix4Scoreboard` (from `verif.scoreboard`) - Scoreboard that compares MAC results with a reference model

2.4. Test Description

2.4.1. Test initialization

- Clock setup
- Creating driver and monitor objects
- Creating data generator
- Creating scoreboard
- Starting the monitor

2.4.2. Writing Data to MAC

- Generating test data
- Splitting into bursts
- Iteratively sending data to MAC

2.4.3. Reading Results from MAC

- Reads data from the same addresses used for writing
- Stores results in data1

2.4.4. Comparing Results with the Reference Model

- Initialize scoreboard with sample count
- Pass input samples to the reference model
- Ryn the reference model
- Comprare DUT output with the reference model

2.4.5. End of the test

2.5. Pass/Fail conditions

2.5.1. Pass

The Mean Squared Error (MSE) between the DUT output and the reference model output is below a defined threshold (by default < 0.002).

2.5.2. Fail

The MSE is greater than or equal to 0.002 (MSE ≥ 0.002), indicating that MAC computation deviated significantly from the expected results. If the comparison between DUT output and reference model output fails, meaning that the numerical error is beyond the acceptable range.