ECE 593 Final Project:

Asynchronous FIFO Design and Verification using UVM

S1, G4: Nick Allmeyer, Alexander Maso, Ahliah Nordstrom

AGENDA

01

TEAM INTRODUCTIONS

SESSION 1: TEAM 4

NICK ALLMEYER
(nall2@pdx.edu)
-Current quarter at PSU: ***9***
- Graduation Date: ***Spring 2025***
- Current Internship / Offer: ***N/A***

ALEXANDER MASO
(amaso@pdx.edu)
-Current quarter at PSU:
- Graduation Date:
- Current Internship / Offer:

AHLIAH NORDSTROM
(ahliah@pdx.edu)
-Current quarter at PSU: ***14***
- Graduation Date: ***June 2024***
- Current Internship / Offer: ***PGE***

02

PROJECT INTRODUCTION

VERIFY AN ASYNC FIFO

C. Cummings FIFO #1

Our design is based off FIFO #1, with ***half*** full/empty signals instead of ***almost***.

CLASS BASED

Code was split into reusable, class-based chunks used to verify the FIFO design

UVM

Class based code was used as a stepping stone for UVM

N. ALLMEYER

- SPECIFIC TASKS…

A. MASO

- SPECIFIC TASKS…

A. NORDSTROM

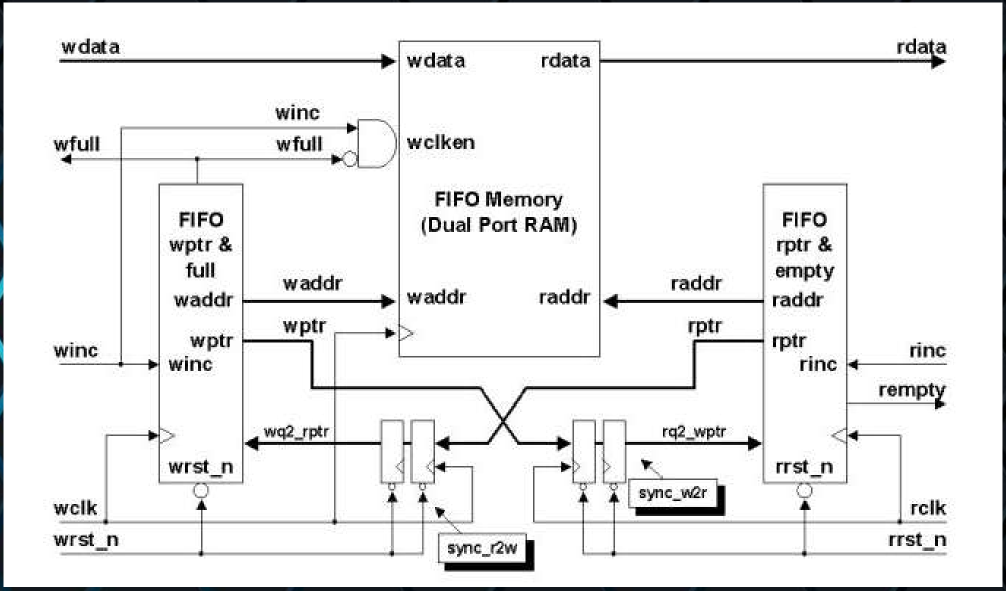
- SPECIFIC TASKS…

03 DESIGN IMPLEMENTATION

SOURCE OF ASYNC FIFO

SOURCE

Clifford E. Cummings FIFO #1 partitioning with synchronized pointer comparison



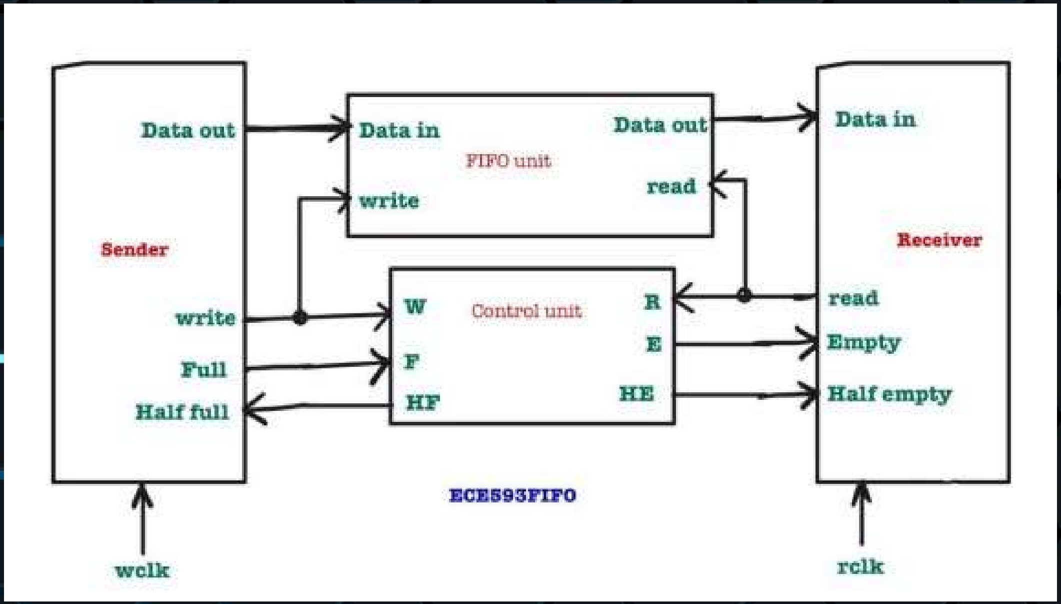
CONTRIBUTIONS

Memory Buffer:
- Optimized to support higher data throughput and reduced latency

Pointer Management:
- Improved pointer increment logic
- Improved detection for flag conditions

Synchronization Logic:
- Developed unified sync model

ASYNC FIFO DESIGN



SPECIFICATIONS

- 80Mhz sender clk freq
- 0 WRITE idle cycles
- 0 READ idle cycles
- 50Mhz receiver clk freq
- 120 WRITE burst size

Modules

- Memory
- Read Pointer
- Sync
- Write Pointer
- Top

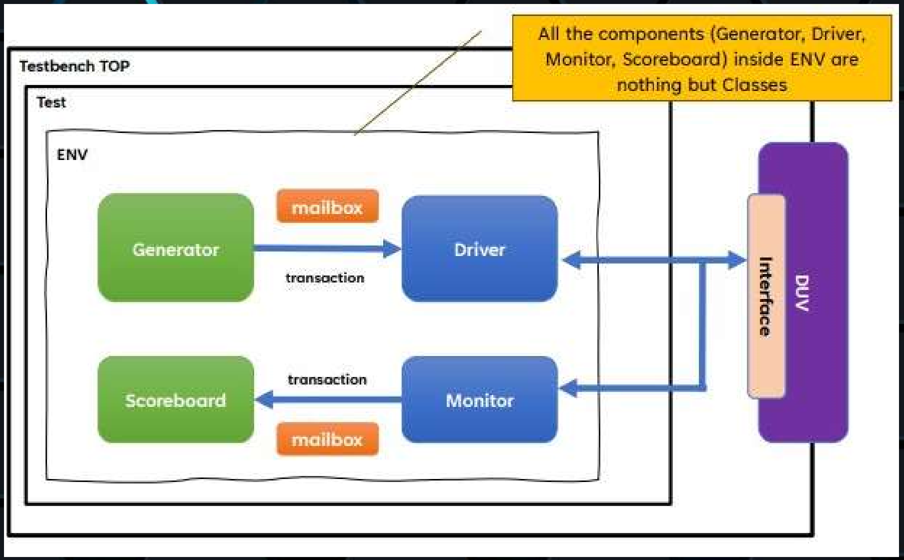
04

CLASS BASED VERIFICATION

HOW WE VERIFIED DUT

Contains external signals for FIFO and internal signals for the BFM. Also contains a reset_fifo task to reset the FIFO and generate both the write and read locks for our two domains

INTERFACE



ENVIRONMENT

Serves as the central structure that contains the major testbench components. Each component is implemented as a class, interacting and synchronizing through transactions and mailboxes

HOW WE VERIFIED DUT: CLASSES

DRIVER

- Drives transactions to FIFO and updates flags
- Uses mailbox to receive from generator and send to scoreboard

GENERATOR

- Generates transactions for W&R operations then sends them to driver and monitor
- Uses mailbox to communicate with driver and monitor

MONITOR

- Monitors read transactions from he FIFO and updates transaction with the current status of FIFO
- Uses mailbox to receive transactions from generator and send to scoreboard

SCOREBOARD

- Checks correctness of data read from FIFO
- Uses mailbox to receive transactions from the driver and monitor.

TESTBENCH

- Orchestrates execution of generator, driver, monitor, & scoreboard

TRANSACTION

- Defines data structure for a single transaction in the FIFO environment
- I/O signals and constraints for randomization

CHALLENGES AND LEARNINGS

TRIPS
Venus is the second planet from the Sun

COMPANIES
Mercury is the closest planet to the Sun

RECOGNITION
Jupiter is the biggest planet of them all

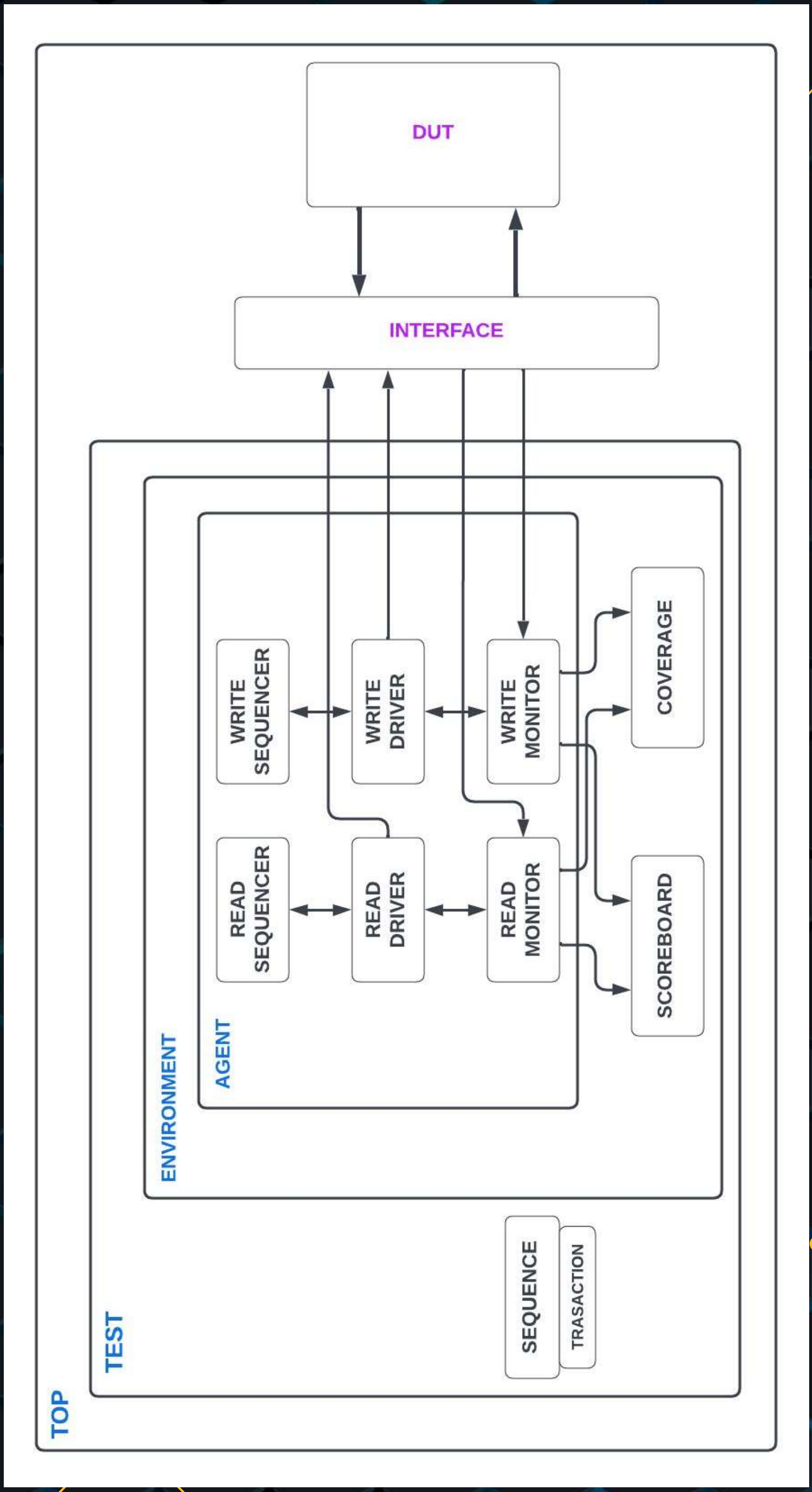
BIG SALARY
Saturn is composed of hydrogen and helium

LEADERSHIP
Neptune is the farthest planet from the Sun

DIVERSITY
Despite being red, Mars is a cold place

05

UVM BASED VERIFICATION


DUT
INTERFACE
TOP
TEST
ENVIRONMENT
AGENT
WRITE SEQUENCER
WRITE DRIVER
WRITE MONITOR
READ SEQUENCER
READ DRIVER
READ MONITOR
COVERAGE
SCOREBOARD
SEQUENCE
TRASACTION

HOW WE VERIFIED DUT: ARCHITECTURE

01

TOP

Instantiates the verification environment

02

TEST

Instantiates the environment and controls the overall verification process by configuring and managing the execution of different test scenarios.

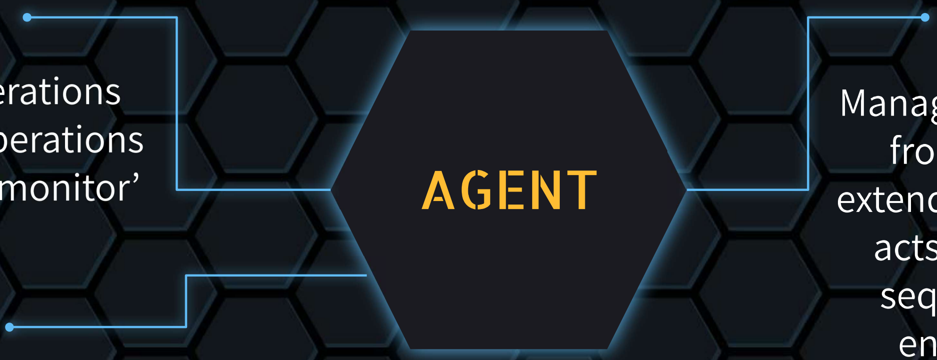
03

ENVIRONMENT

Encapsulates the set-up of the agent and thus the sequencer, monitor, and driver, as well as the scoreboard and coverage.

HOW WE VERIFIED DUT: ARCHITECTURE

An active agent containing sequencers, drivers and monitors for both read and write operations. It's used to hold and instantiate these classes and we only have one interface and thus, only one agent.



MONITOR

Monitoring of read operations separately from write operations by extending the ‘uvm_monitor’ and vise versa

DRIVER

Monitoring of read operations separately from write operations by extending the ‘uvm_monitor’ and vise versa

SEQUENCER

Manages the flow of transactions from sequence to driver by extending the ‘uvm_sequencer’. It acts as the intermediary that sequences transactions and ensures they are correctly managed and forwarded to the driver.

HOW WE VERIFIED DUT: ARCHITECTURE

SCOREBOARD

Compares expected results with actual results of all operations. It verifies that data read from the FIFO matches data written to it.

SEQUENCE

Generates sequences of read transactions separately from write transactions by extending the ‘uvm_sequence’ and vice versa.

TRANSACTION

Extends ‘uvm_sequence_item’ and represents basic unit of data transfer in UVM environment. All necessary signals and data for a single transaction are encapsulated here.

TEST SCENARIOS AND COVERAGE

T#01	T#02	T#03	T#04	T#05	T#06	T#07
	🟦					
					🟨	

🟦 FAILED

PASSED 🟨

45% CODE COV

...

FUNCTIONAL COV **30%**

...

06

OVERALL CHALLENGES AND LEARNING

07

DEMO AND Q&A