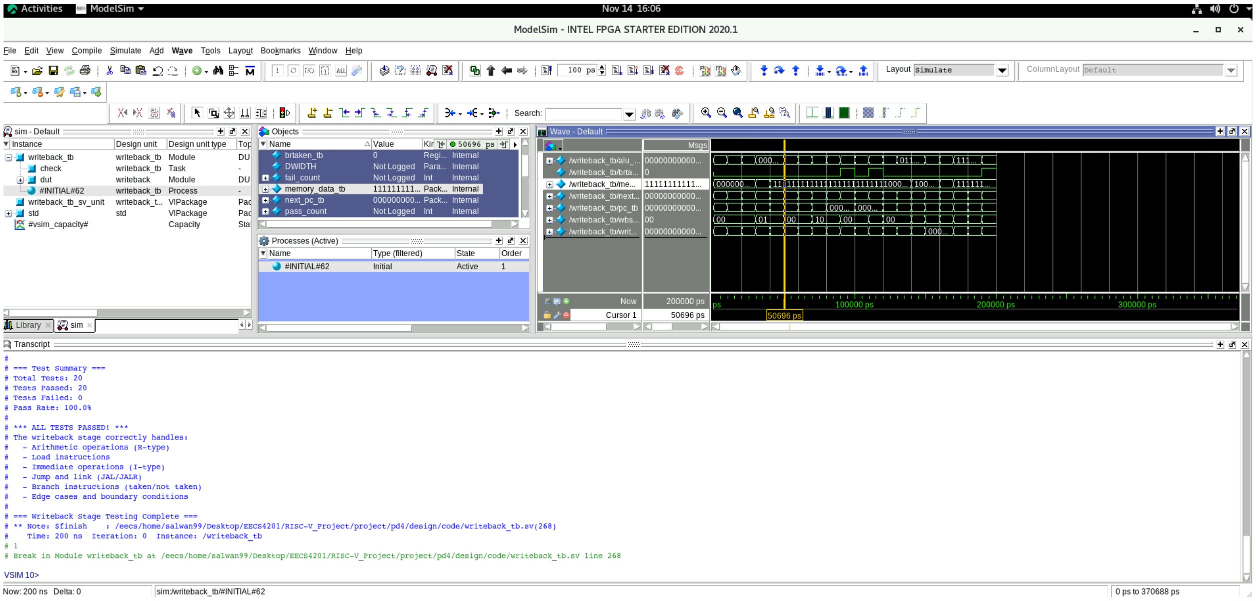
PD4 Report Document

Team Name: MuxMasters
Salwan Aldhahab (218597963)
Mavra Muzmmal (219075969)

EECS 4201 E
Prof. Anirudh M Kaushik
Lassonde School of Engineering
York University

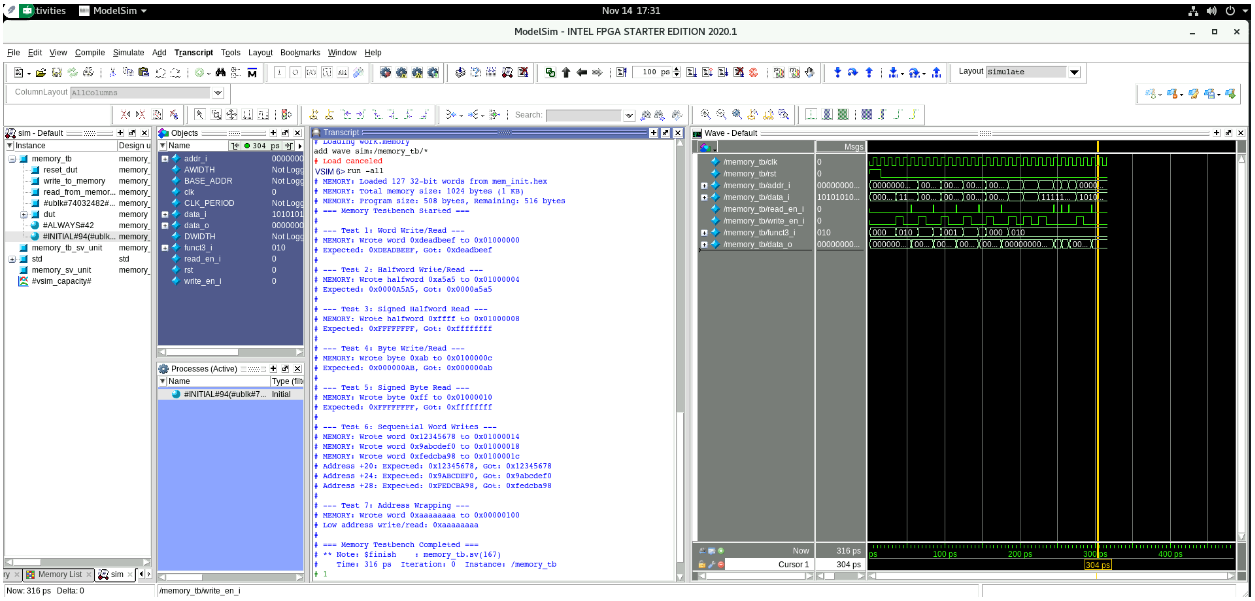
ModelSim Our Testbench Waveform Screenshots:



```
#
# === Test Summary ===
# Total Tests: 20
# Tests Passed: 20
# Tests Failed: 0
# Pass Rate: 100.0%
#
# *** ALL TESTS PASSED! ***
# The writeback stage correctly handles:
#   - Arithmetic operations (R-type)
#   - Load instructions
#   - Immediate operations (I-type)
#   - Jump and link (JAL/JALR)
#   - Branch instructions (taken/not taken)
#   - Edge cases and boundary conditions
#
# === Writeback Stage Testing Complete ===
# ** Note: $finish    : /eecs/home/salwan99/Desktop/EECS4201/RISC-V_Project/project/pd4/design/code/writeback_tb.sv(268)
#    Time: 200 ns  Iteration: 0  Instance: /writeback_tb
# 1
# Break in Module writeback_tb at /eecs/home/salwan99/Desktop/EECS4201/RISC-V_Project/project/pd4/design/code/writeback_tb.sv line 268

VSIM 10>
```

Writeback_tb.sv



```
# Loading work.memory
add wave sim:/memory_tb/*
# Load canceled
VSIM 6> run -all
# MEMORY: Loaded 127 32-bit words from mem_init.hex
# MEMORY: Total memory size: 1024 bytes (1 KB)
# MEMORY: Program size: 508 bytes, Remaining: 516 bytes
# === Memory Testbench Started ===
#
# --- Test 1: Word Write/Read ---
# MEMORY: Wrote word 0xdeadbeef to 0x01000000
# Expected: 0xDEADBEEF, Got: 0xdeadbeef
#
# --- Test 2: Halfword Write/Read ---
# MEMORY: Wrote halfword 0xa5a5 to 0x01000004
# Expected: 0x0000A5A5, Got: 0x0000a5a5
#
# --- Test 3: Signed Halfword Read ---
# MEMORY: Wrote halfword 0xffff to 0x01000008
# Expected: 0xFFFFFFFF, Got: 0xffffffff
#
# --- Test 4: Byte Write/Read ---
# MEMORY: Wrote byte 0xab to 0x0100000c
# Expected: 0x000000AB, Got: 0x000000ab
#
# --- Test 5: Signed Byte Read ---
# MEMORY: Wrote byte 0xff to 0x01000010
# Expected: 0xFFFFFFFF, Got: 0xffffffff
#
# --- Test 6: Sequential Word Writes ---
# MEMORY: Wrote word 0x12345678 to 0x01000014
# MEMORY: Wrote word 0x9abcdef0 to 0x01000018
# MEMORY: Wrote word 0xfedcba98 to 0x0100001c
# Address +20: Expected: 0x12345678, Got: 0x12345678
# Address +24: Expected: 0x9ABCDEF0, Got: 0x9abcdef0
# Address +28: Expected: 0xFEDCBA98, Got: 0xfedcba98
#
# --- Test 7: Address Wrapping ---
# MEMORY: Wrote word 0xaaaaaaaa to 0x00000100
# Low address write/read: 0xaaaaaaaa
#
# === Memory Testbench Completed ===
# ** Note: $finish    : memory_tb.sv(167)
#    Time: 316 ps  Iteration: 0  Instance: /memory_tb
# 1
```

memory_tb.sv

Challenges Faced During PD4 Deliverables:

One of the biggest challenges we faced while doing the pd4 was we misunderstood the pipeline requirements. At first I thought we needed to fully pipeline the write back and memory stages, so I kept adding extra registers and forwarding paths which were not required and it caused persistent errors and unnecessary complexity. At the same time we were facing a mismatch issue in the instruction memory that made the simulator flag important warnings - specificall, the funct3_i port was accidentally treated as 1 bit instead of 3 bits. We corrected this with the proper assignment of 3'b010 which helped to stabilize the memory interface and cleared the compilation issues.

After fixing the signalling I encountered another issue which was the program counter (PC) which refused to move from the base address. Debugging showed that the Control Unit had been unintentionally skipped in the top-level wiring. Important control signals, such as branch and jump, therefore defaulted to zero, totally stopping the process. This error was comparable to my previous misunderstanding of the pipeline steps, when my presumptions resulted in erroneous or insufficient connections. The PC impasse was resolved and appropriate control flow was restored by reintroducing the Control Unit outputs.

We also struggled with the memory behavior, especially the wraparound logic. The memory must loop back to the first address instead of increasing memory size, which i didn't know that memory was reaching the last address. In the process of resolving this, I discovered a structural problem with the initialization loop: the $display message was positioned within the for-loop, resulting in repeated printing. The module operated as anticipated and the simulation output became clear and accurate when the proper wraparound was implemented, the loop structure was cleaned up, and the memory addressing was adjusted.