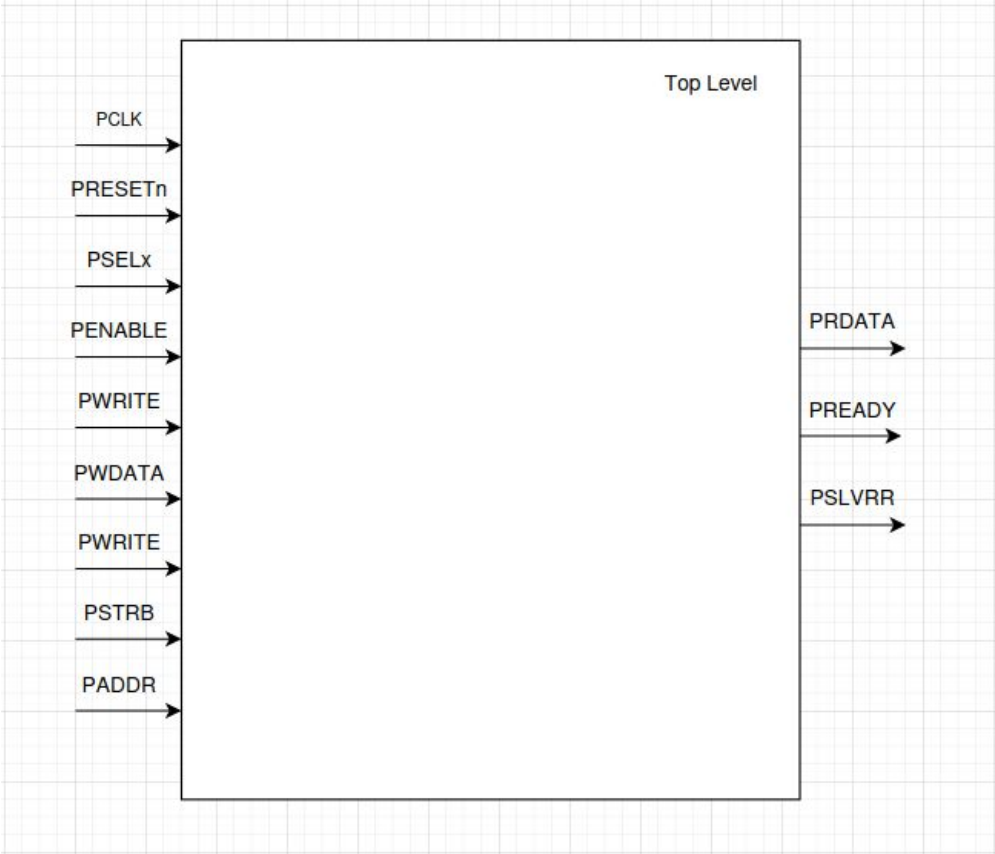
# Feature

- **Protocol:** Follows AMBA APB, that requires least two cycle to perform any transaction with IDLE, SETUP & ACCESS phases.
- **Memory Size:** 64 KB total capacity (65,536 bytes).
- **Data Bus:** 64-bit wide (`PWDATA/PRDATA`)
- **Addressing:** 32-bit byte address (`PADDR[31:0]`) with parameterizable base address.
- **Endianness & Alignment:** Little-endian; access must be 8-byte aligned

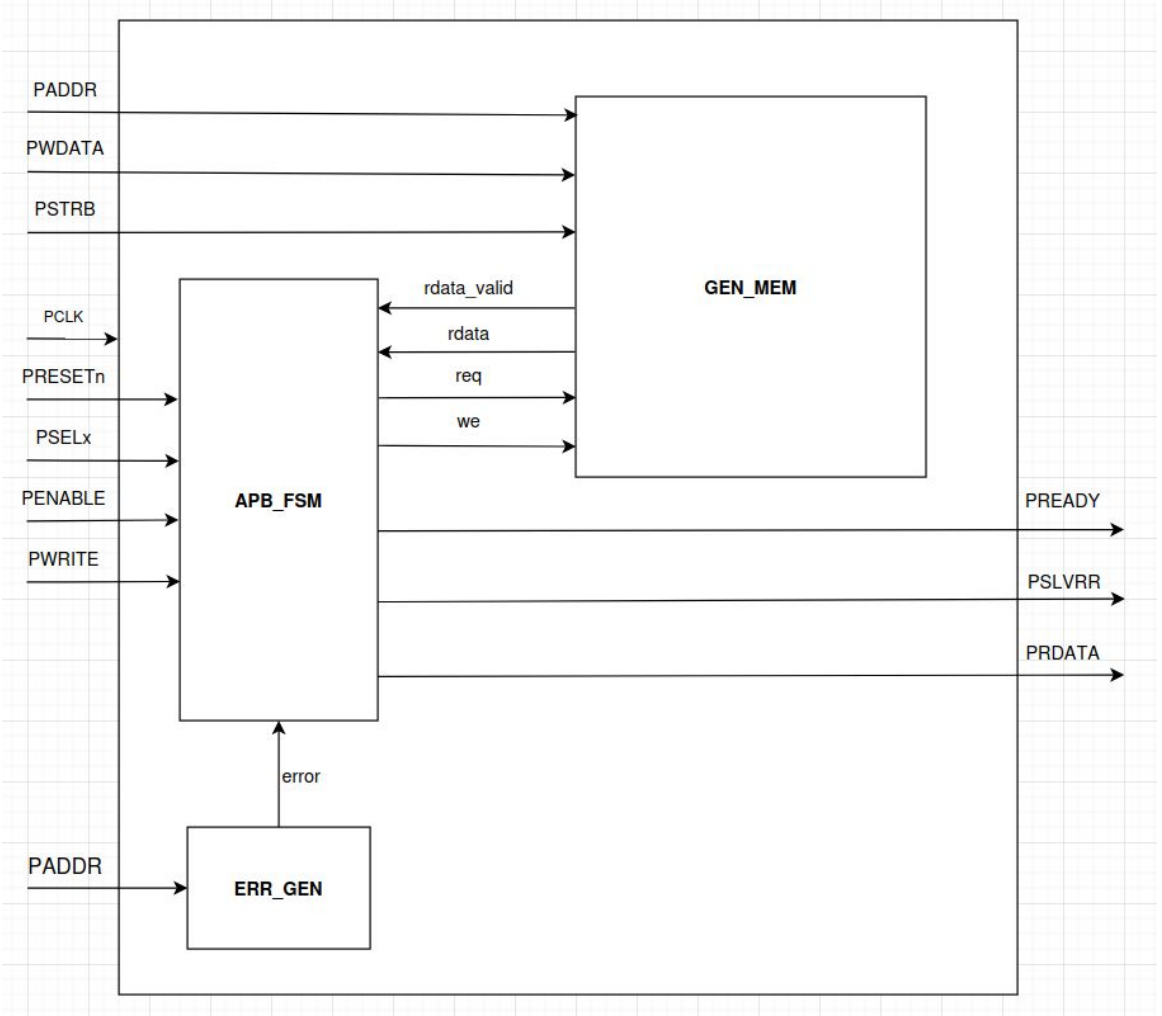
# Feature

- **Error Conditions (PSLVERR=1):**
    - Address outside 64 KB window.
    - Misaligned access.
- **Throughput:** One request per APB transfer, no outstanding transactions.
- **Reset:** Active-low asynchronous (`PRESETn`), ensures known-good idle state.
- **Strobe**: Supports byte-enable writes via `PSTRB[7:0]`

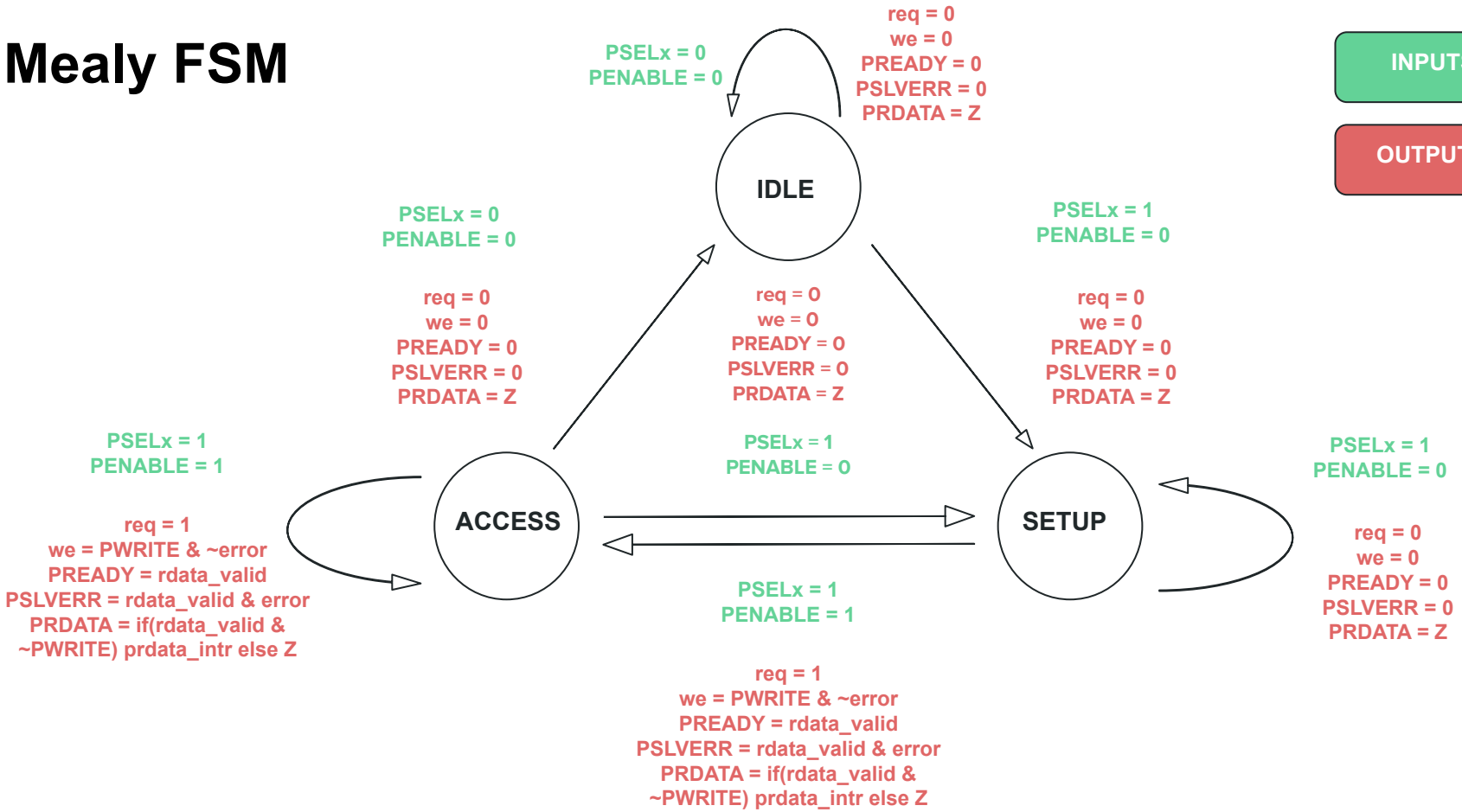
# Top Level IOs



# Microarchitecture



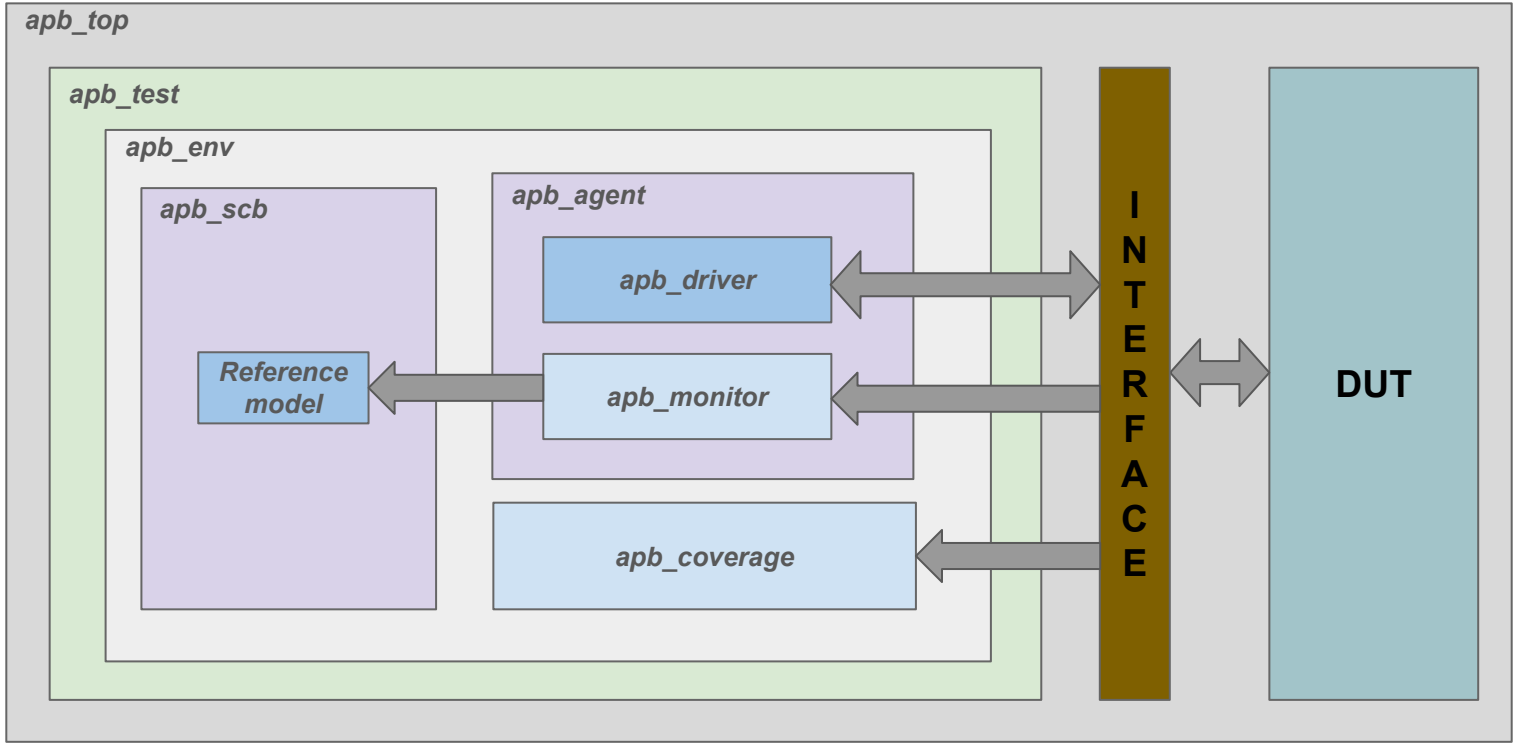
# Mealy FSM


INPUTS
OUTPUTS
IDLE
ACCESS
SETUP
PSELx = 0
PENABLE = 0
req = 0
we = 0
PREADY = 0
PSLVERR = 0
PRDATA = Z
PSELx = 0
PENABLE = 0
req = 0
we = 0
PREADY = 0
PSLVERR = 0
PRDATA = Z
PSELx = 1
PENABLE = 0
req = 0
we = 0
PREADY = 0
PSLVERR = 0
PRDATA = Z
PSELx = 1
PENABLE = 1
req = 1
we = PWRITE & ~error
PREADY = rdata_valid
PSLVERR = rdata_valid & error
PRDATA = if(rdata_valid &
~PWRITE) prdata_intr else Z
PSELx = 1
PENABLE = 0
PSELx = 1
PENABLE = 1
req = 1
we = PWRITE & ~error
PREADY = rdata_valid
PSLVERR = rdata_valid & error
PRDATA = if(rdata_valid &
~PWRITE) prdata_intr else Z
PSELx = 1
PENABLE = 0
req = 0
we = 0
PREADY = 0
PSLVERR = 0
PRDATA = Z

# Operation

- **Write Operation:**
  - SETUP: `PSEL=1, PENABLE=0, PWRITE=1`.
  - ACCESS: `PENABLE=1`; slave asserts `PREADY` once rdata_valid is asserted; write occurs with byte enables applied.
  - Complete while `PREADY=1`.
- **Read Operation:**
  - SETUP: `PSEL=1, PENABLE=0, PWRITE=0.`
  - ACCESS: `PENABLE=1`; slave asserts `PREADY` once rdata_valid is asserted.
  - Complete while `PREADY=1`

# Testbench Architecture



# Test Plan

| Section | Feature | Purpose | Test/Assertion |
|---|---|---|---|
| 2.1 | Active-LOW Reset | Ensures all peripherals and bus logic return to a known safe state during reset. | apb_reset_test<br>apb_reset_while_trans_test<br>apb_reset_assert |
| 2.1 | Write/Read | Eligible for Write and Read transaction using PWRITE signal | apb_successive_wr_test<br>apb_b2b_wr_test<br>apb_random_addr_wr_test<br>apb_random_stress_test<br>apb_valid_write_assert<br>apb_valid_read_assert |
| 2.1<br>3.2 | PSTRB | Enables sparse data writes by selecting which byte lanes of the data bus are valid during a write. | apb_strobe_test |

# Test Plan (Cont)

| Section | Feature | Purpose | Test/Assertion |
|---|---|---|---|
| 2.1 | PSLVERR | For invalid transfer, error will occur | apb_slave_error_aor_test<br>apb_addr_misalligned_test |
| 1.1 | Two-phase transfer | all transfers require at least two cycles, | apb_violation_test<br>apb_sel_enable_assert<br>apb_pslverr_assert |
| 3.1.2<br>3.3.2 | Wait-state insertion | PREADY is used to extend an APB transfer | apb_pready_assert |
| 2.1 | PADDR supports 32 bit. But requirement is 64K memory, so 16 bit | If address is out of range slave error will occur | apb_slave_error_aor_test<br>apb_boundary_test |

# Test Cases

| TEST | PURPOSE | RESULT | IMPLEMENTATION |
|---|---|---|---|
| **apb_reset_test** | Verify that when reset is asserted, all APB slave signals and internal states return to default values | PASS:2<br>FAIL:0 | 1. Set PRESETn = 0<br>2. Perform write to any address<br>3. Perform read to any address<br><br>**Pass Condition:** if actual PRDATA is 0 the test will pass |
| **apb_reset_while_trans_test** | Verify during the transaction, if reset, DUT can perform write or read | PASS | 1. Perform write to any address<br>2. Set PRESETn = 0 while PENABLE is HIGH<br>3. Perform read to any address<br><br>**Pass Condition:** if actual PRDATA is 0 the test will pass |
| **apb_successive_wr_test** | Verify correct memory operation when writing and reading across the full address range | PASS:65535<br>FAIL:0 | 1. Write full memory<br>2. Read full memory<br><br>**Pass Condition:** Compare read data with expected values. |

# Test Cases (cont)

| | | | |
|---|---|---|---|
| **apb_b2b_wr_test** | Validate correct operation for back-to-back accesses to consecutive memory locations | PASS:131070<br>FAIL:0 | 1. Multiple times write in same address<br>2. Read same address<br>3. Write same address<br>4. Read same address<br><br>**Pass Condition:** Compare read data with expected values. |
| **apb_slave_error_aor_test** | Verify that the slave asserts error when an address out of range is accessed | PASS:100<br>FAIL: 0 | 1. Write at addresses below BASE_ADDR and beyond BASE_ADDR + MEM_SIZE_K*1024 - 1<br><br>**Pass Condition:** If PSLVRR then the test will pass |
| **apb_random_addr_wr_test** | Validate memory correctness under randomized accesses | PASS:65535<br>FAIL: 0 | 1. Generate random valid addresses within memory range.<br>2. Perform writes with random data.<br>3. Perform read with random data.<br><br>**Pass Condition:** If expected data and actual data is matched the test will pass |

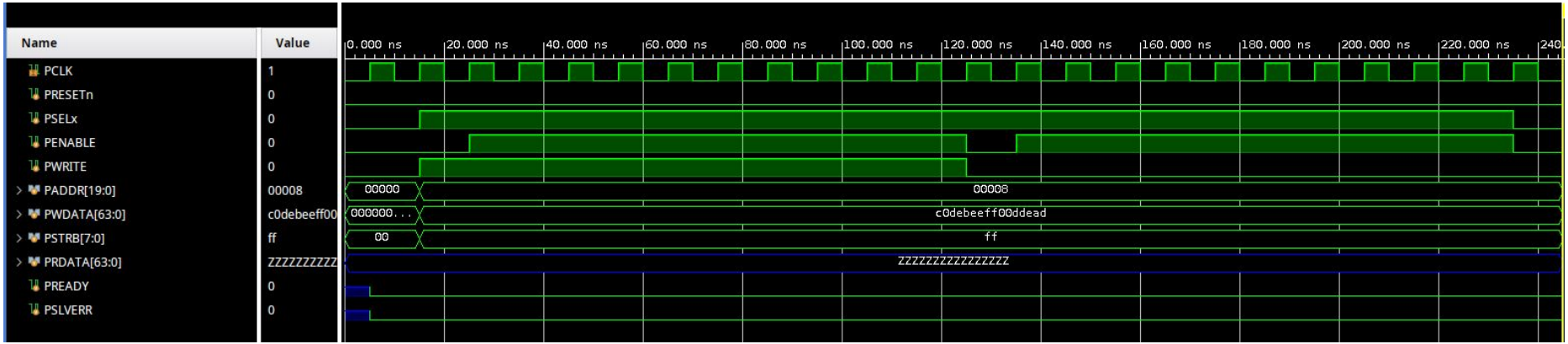
# Test Cases (cont)

| | | | |
|---|---|---|---|
| **apb_strobe_test** | Checking write can perform based on strobe or not | PASS:65535<br>FAIL: 0 | 1. Generate random strobe<br>2. Perform writes with random data.<br>3. Perform read with random data.<br><br>**Pass Condition**: If expected data and actual data is matched the test will pass |
| **apb_addr_misalligned_test** | To check APB can write to misaligned address or not | PASS:57343<br>FAIL: 0 | 1. Perform Write & Read to all misaligned address<br><br>**Pass Condition:** If write not happen, the test will pass |

# Test Cases (cont)

| | | | |
|---|---|---|---|
| **apb_boundary_test** | To check upper and lower boundary are capable to perform write and read | PASS: 2<br>FAIL: 0 | 1. Write hx0 and hxFFF8<br>2. Read both address<br><br>**Pass Condition**: If expected data and actual data is matched the test will pass |
| **apb_random_stress_test** | To check the stress handling of dut | PASS: 193205<br>FAIL: 0 | 1. Run all the tests randomly multiple times<br><br>**Pass Condition**: If expected data and actual data is matched the test will pass |
| **apb_violation_test** | To check while violation in protocol, slave can write and read or not | PASS: 1<br>FAIL: 0 | 1. Keep enable off while write/read data.<br><br>**Pass Condition**: If slave can write or read, the test will fail |

# Reset Test

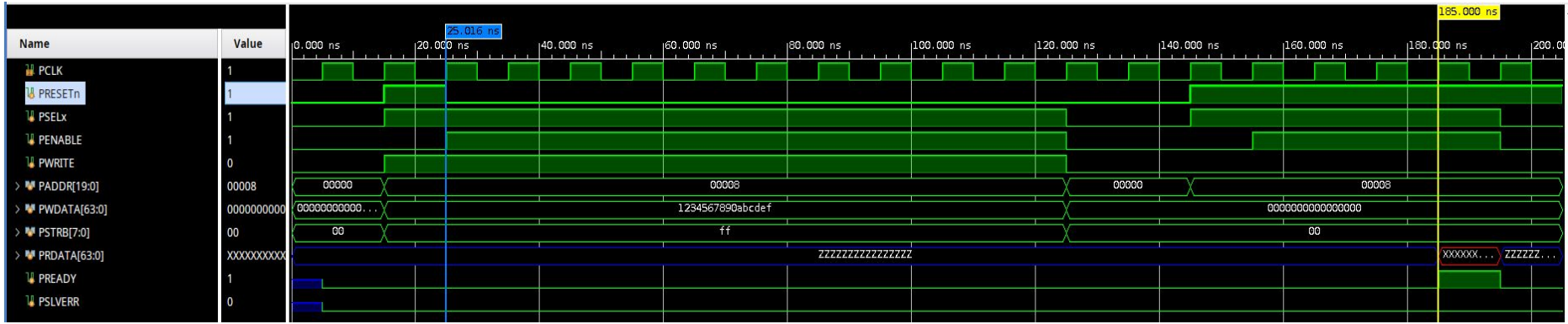



## Assertion Failed

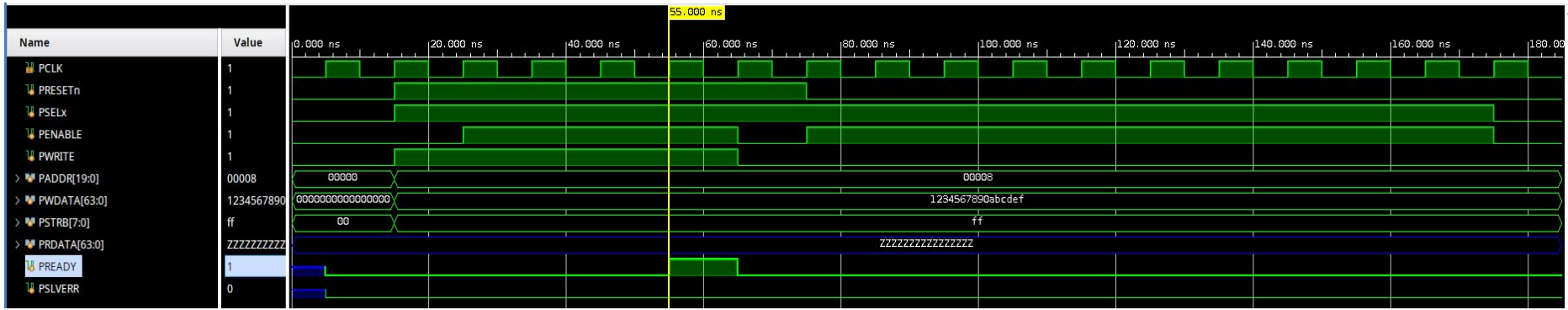
```
Time: 225 ns Started: 225 ns Scope: /apb_tb/inf File: /home/fazlul-karim/Desktop/apb_mem/verification/apb_interface.sv Line:86
Error: Signals not reset correctly during PRESETn LOW
Time: 225 ns  Iteration: 0  Process: /apb_tb/inf//apb_tb/inf  Scope: apb_tb.inf  File: /home/fazlul-karim/Desktop/apb_mem/verification/apb_interface.sv Line: 87
```

# Reset While Transaction Test

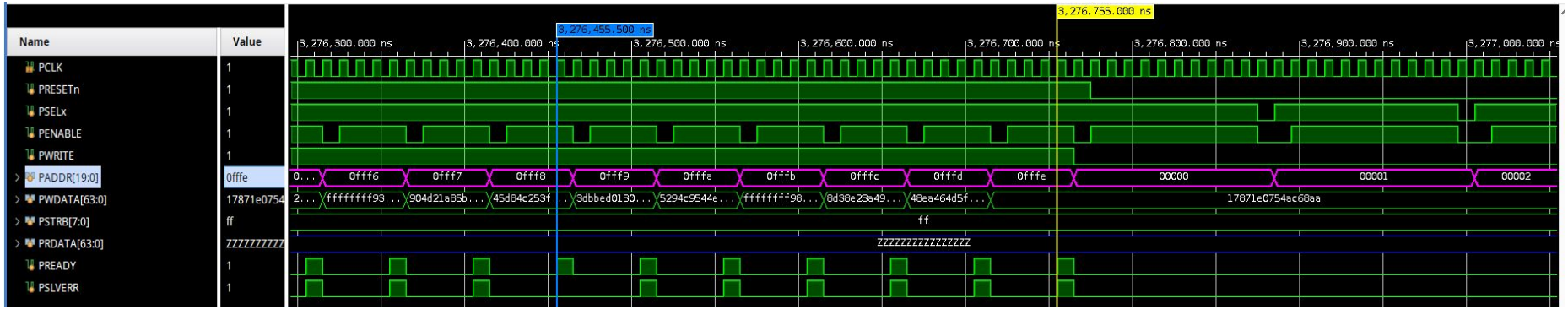
Reset while WRITE



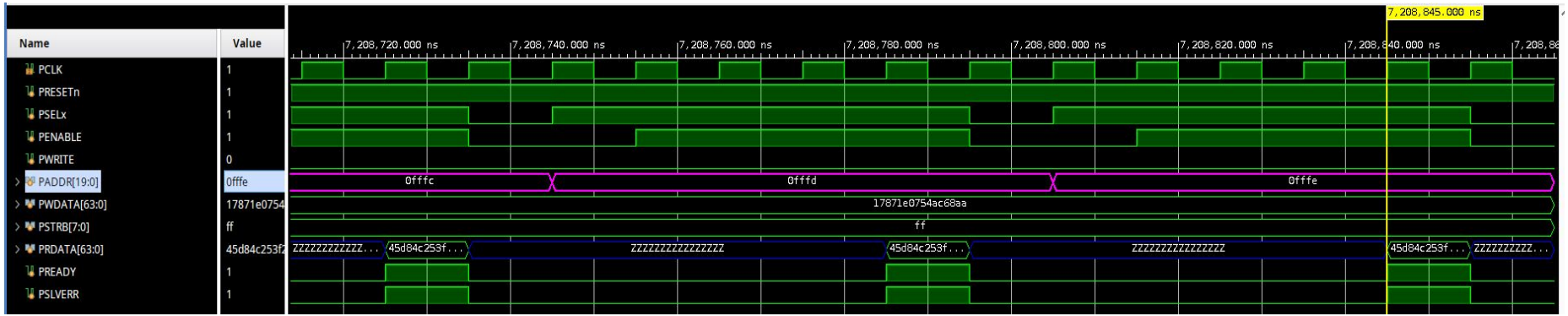

Reset while READ:



# Sequential Read Write Test



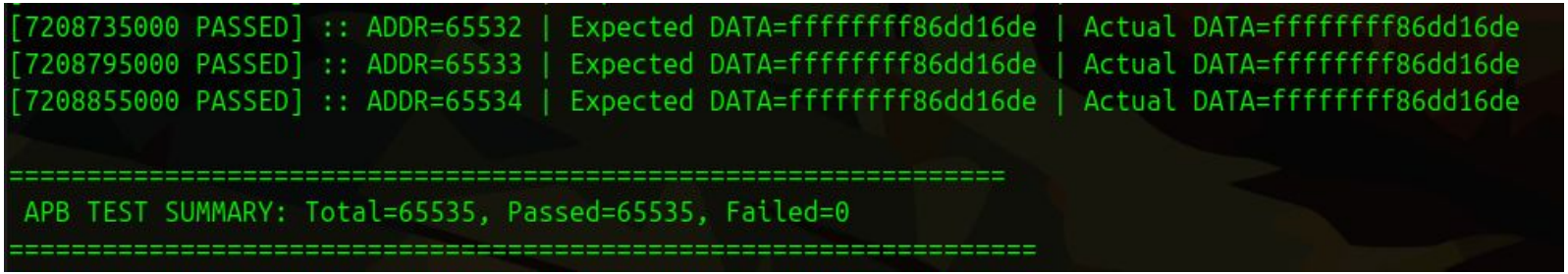

Writing full 64KB address. But write happen only first 64 address



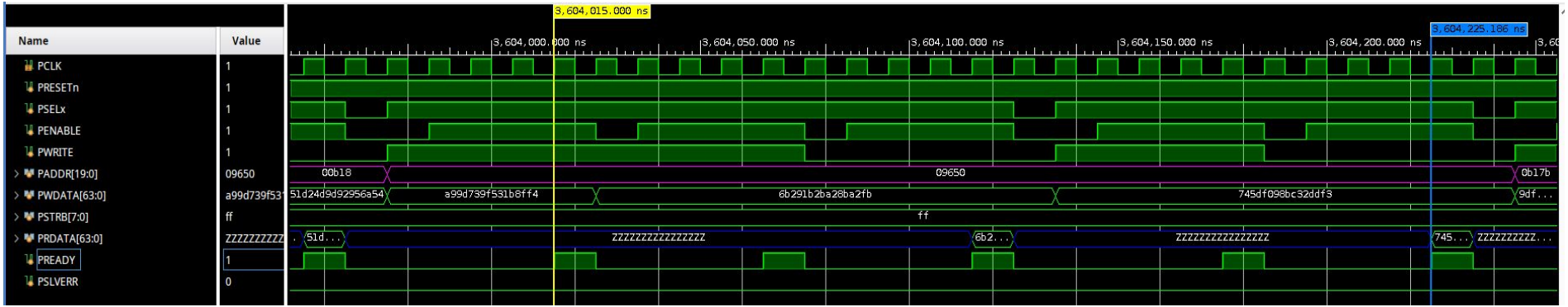

Reading full 64KB address. But write happen only first 64 address

# Sequential Read Write Test (Cont)

**Output**:



# Back to back Write Read Test

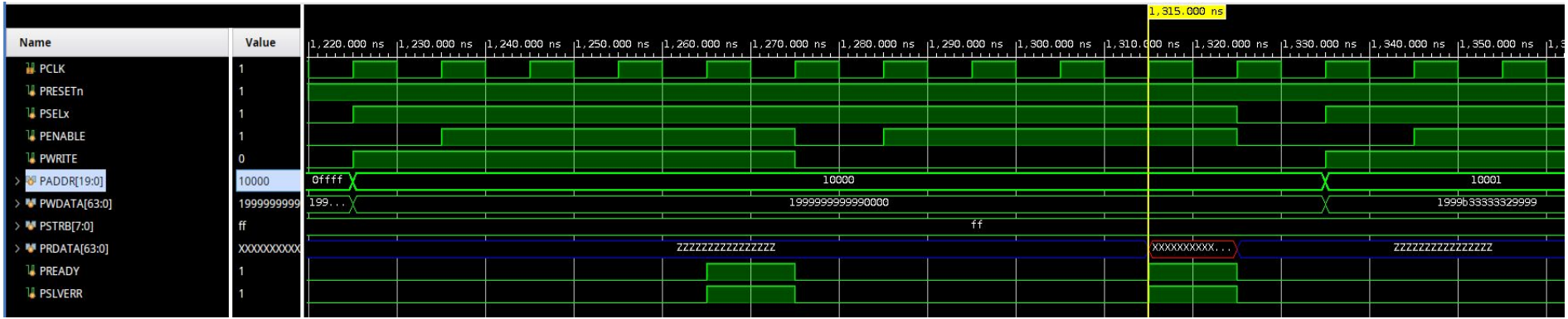



**Output**:

```
[17694075000 PASSED] :: ADDR=48523 | Expected DATA=f66cb5341f63d2f | Actual DATA=f66cb5341f63d2f
[17694185000 PASSED] :: ADDR=48523 | Expected DATA=f66cb5341f63d2f | Actual DATA=f66cb5341f63d2f
[17694345000 PASSED] :: ADDR=3063 | Expected DATA=795aff9e7603d625 | Actual DATA=795aff9e7603d625
[17694455000 PASSED] :: ADDR=3063 | Expected DATA=795aff9e7603d625 | Actual DATA=795aff9e7603d625

====================================================================
 APB TEST SUMMARY: Total=131070, Passed=131070, Failed=0
====================================================================
```

# Slave Error AOR Test

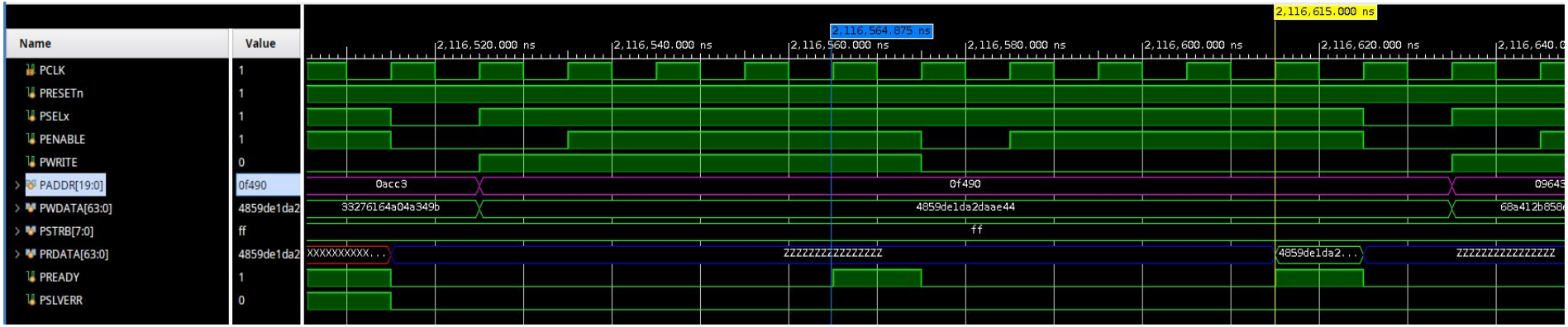



**Output**:

```
[10345000 PASSED] :: ADDR=65628 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[10455000 PASSED] :: ADDR=65629 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[10565000 PASSED] :: ADDR=65630 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[10675000 PASSED] :: ADDR=65631 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[10785000 PASSED] :: ADDR=65632 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[10895000 PASSED] :: ADDR=65633 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[11005000 PASSED] :: ADDR=65634 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx

============================================================
 APB TEST SUMMARY: Total=100, Passed=100, Failed=0
============================================================
```

# Random Write Read Test

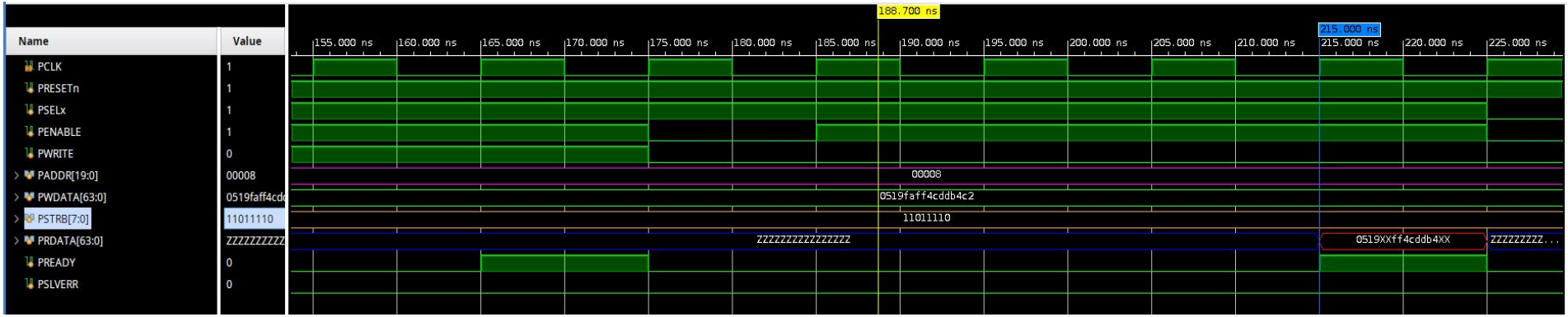



**Output**:

```
[7208305000 PASSED] :: ADDR=49032 | Expected DATA=754a5c7f5a93c4c9 | Actual DATA=754a5c7f5a93c4c9
[7208415000 PASSED] :: ADDR=44987 | Expected DATA=334db32f1616904d | Actual DATA=334db32f1616904d
[7208525000 PASSED] :: ADDR=9490 | Expected DATA=4a82219d5b35d9ff | Actual DATA=4a82219d5b35d9ff
[7208635000 PASSED] :: ADDR=33170 | Expected DATA=8191db3c0a75eb99 | Actual DATA=8191db3c0a75eb99
[7208745000 PASSED] :: ADDR=29807 | Expected DATA=4e63e0721243f801 | Actual DATA=4e63e0721243f801
[7208855000 PASSED] :: ADDR=11285 | Expected DATA=487c070188179740 | Actual DATA=487c070188179740

================================================================
 APB TEST SUMMARY: Total=65535, Passed=65535, Failed=0
================================================================
```

# Strobe Test

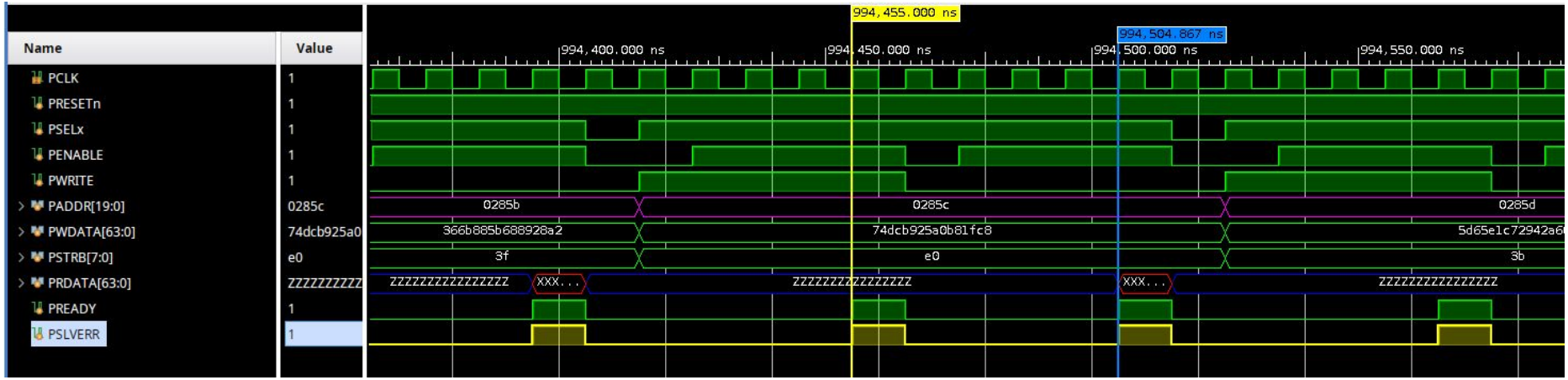



**Output**:

```
[7208305000 PASSED] :: ADDR=65529 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff
[7208415000 PASSED] :: ADDR=65530 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff
[7208525000 PASSED] :: ADDR=65531 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff
[7208635000 PASSED] :: ADDR=65532 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff
[7208745000 PASSED] :: ADDR=65533 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff
[7208855000 PASSED] :: ADDR=65534 | Expected DATA=xxxxxxxxxxxxxxff | Actual DATA=xxxxxxxxxxxxxxff

================================================================
 APB TEST SUMMARY: Total=65535, Passed=65535, Failed=0
================================================================
```

# Alignment Test

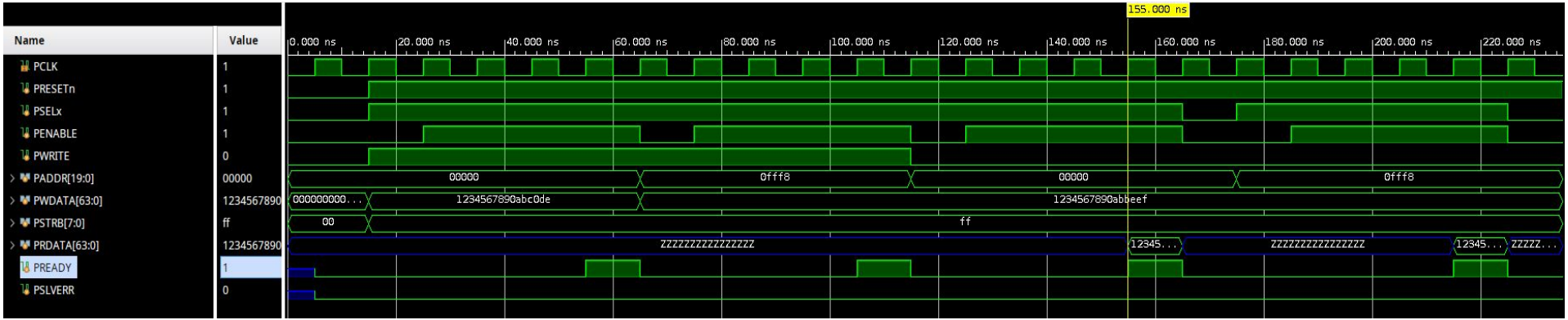



**Output**:

```
[6307405000 PASSED] :: ADDR=65531 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[6307515000 PASSED] :: ADDR=65532 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[6307625000 PASSED] :: ADDR=65533 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx
[6307735000 PASSED] :: ADDR=65534 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx

============================================================
 APB TEST SUMMARY: Total=57343, Passed=57343, Failed=0
============================================================
```

# Boundary Test

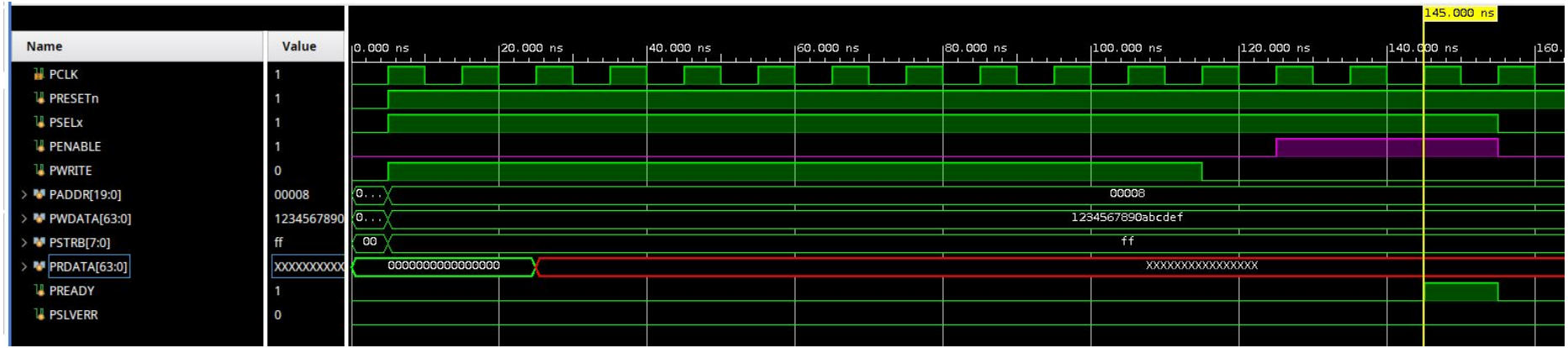



**Output**:

```
[0] :: apb_scoreboard :: Scoreboard Constructed
[0] :: apb_boundary_test :: Single Write Read Test Constructed
[165000 PASSED] :: ADDR=0 | Expected DATA=1234567890abc0de | Actual DATA=1234567890abc0de
[225000 PASSED] :: ADDR=65528 | Expected DATA=1234567890abbeef | Actual DATA=1234567890abbeef

================================================================
 APB TEST SUMMARY: Total=2, Passed=2, Failed=0
================================================================
```

# Violation Test




## Output:

```
Time: 135 ns Started: 125 ns Scope: /apb_tb/inf File: /home/fazlul-karim/Desktop/apb_mem/verification/apb_interface.sv Line:65
Error: PENABLE not asserted in cycle after PSELx HIGH
Time: 135 ns  Iteration: 0  Process: /apb_tb/inf//apb_tb/inf  Scope: apb_tb.inf  File: /home/fazlul-karim/Desktop/apb_mem/verification/apb_interface.sv Line: 66
[175000 PASSED] :: ADDR=8 | Expected DATA=xxxxxxxxxxxxxxxx | Actual DATA=xxxxxxxxxxxxxxxx

================================================================
 APB TEST SUMMARY: Total=1, Passed=1, Failed=0
================================================================
```

# Assertion Plan

| Sl No. | Test Name | Purpose |
|---|---|---|
| | | |
| 1 | apb_sel_enable_assert | Verify that when PSELx is HIGH, the next cycle has PENABLE HIGH |
| 2 | apb_pready_assert | To ensure that when PENABLE and PSELx are both HIGH until PREADY goes HIGH |
| 3 | apb_pslverr_assert | PSLVERR must assert when PSEL, PENABLE, and PREADY are HIGH |
| 4 | apb_reset_assert | To ensure apb bus goes default value |
| 5 | apb_valid_write_assert | when PREADY and PWRITE is HIGH, PWDATA must be stable |
| 6 | apb_valid_read_assert | when PREADY is HIGH and PWRITE is LOW, PRDATA must be stable |

# Assertion Plan

```
23
24      // 1. PSELx must be followed by PENABLE
25      property apb_sel_enable_assert;
26          @(posedge PCLK) disable iff(!PRESETn)
27          (PSELx && !PENABLE) |=> (PSELx && PENABLE);
28      endproperty
29
30      // 2. Must wait until PREADY when PSELx & PENABLE HIGH
31      property apb_pready_assert;
32          @(posedge PCLK) disable iff(!PRESETn)
33          (PSELx && PENABLE) |-> (PSELx && PENABLE) until_with PREADY;
34      endproperty
35
36      // 3. PSLVERR only valid when transfer is active
37      property apb_pslverr_assert;
38          @(posedge PCLK) disable iff(!PRESETn)
39          PSLVERR |-> ##0 (PSELx && PENABLE && PREADY);
40      endproperty
```

# Assertion Implementation (Cont)

```
41
42      // 4. Reset must drive signals low
43      property apb_reset_assert;
44          @(posedge PCLK) disable iff(PRESETn)
45          (!PRESETn) |-> (PADDR == 0 && PWDATA == 0 && PRDATA == 0 && PENABLE == 0
46          && PSELx == 0 && PWRITE == 0 && PREADY == 0 && PSLVERR == 0);
47      endproperty
48
49      // 5. Write data must remain stable during transfer
50      property apb_valid_write_assert;
51          @(posedge PCLK) disable iff(!PRESETn)
52          (PSELx && PENABLE && PWRITE && !PSLVERR) |-> ($stable(PWDATA) until_with PREADY);
53      endproperty
54
55      // 6. Read data must remain stable during transfer
56      property apb_valid_read_assert;
57          @(posedge PCLK) disable iff(!PRESETn)
58          (PSELx && PENABLE && !PWRITE && PREADY && !PSLVERR) |-> ($stable(PRDATA) until_with PREADY);
59      endproperty
60
```

# Assertion Implementation (Cont)

```
62
63      // Assertions
64
65      assert property (apb_sel_enable_assert)
66          else $error("PENABLE not asserted in cycle after PSELx HIGH");
67
68      assert property (apb_pready_assert)
69          else $error("PREADY not asserted while PSELx & PENABLE HIGH");
70
71      assert property (apb_pslverr_assert)
72          else $error("PSLVERR asserted without valid transfer");
73
74      assert property (apb_reset_assert)
75          else $error("Signals not reset correctly during PRESETn LOW");
76
77      assert property (apb_valid_write_assert)
78          else $error("PWDATA not stable during write transfer");
79
80      assert property (apb_valid_read_assert)
81          else $error("PRDATA not stable during read transfer");
82
```

# Coverage Plan

| Sl No. | Coverpoint | Purpose | Implementation |
|---|---|---|---|
| 1 | PSELx_cp | Ensure slave select signal toggles correctly. | Check bins sel_active and sel_inactive during various transactions. |
| 2 | PENABLE_cp | Ensure PENABLE signal toggles properly during transactions. | Sample enable active and inactive states during WRITE and READ cycles. |
| 3 | PWRITE_cp | Confirm both READ and WRITE operations are exercised. | check bins for both PWRITE=0 (read) and PWRITE=1 (write). |
| 4 | PREADY_cp | Ensure PREADY signal is properly sampled for wait states. | Include both ready and not_ready bins. |
| 5 | PADDR_cp | Verify that addresses cover the full memory range. | Check valid & invalid address bins. |
| 6 | PWDATA_cp | Ensure write data exercises different value ranges. | Drive low, mid, high data values. |
| 7 | PRDATA_cp | Ensure read data exercises different value ranges from DUT. | Sample PRDATA during read transactions with low, mid, high address bins. |
| 8 | PSLVERR_cp | Check APB slave error appear | set bins for error (1) and normal (0) responses. |
| 9 | cross_addr_data_pwrite | Ensure data written to addresses exercises all combinations. | Checking all range of address get write & read with all range of data |