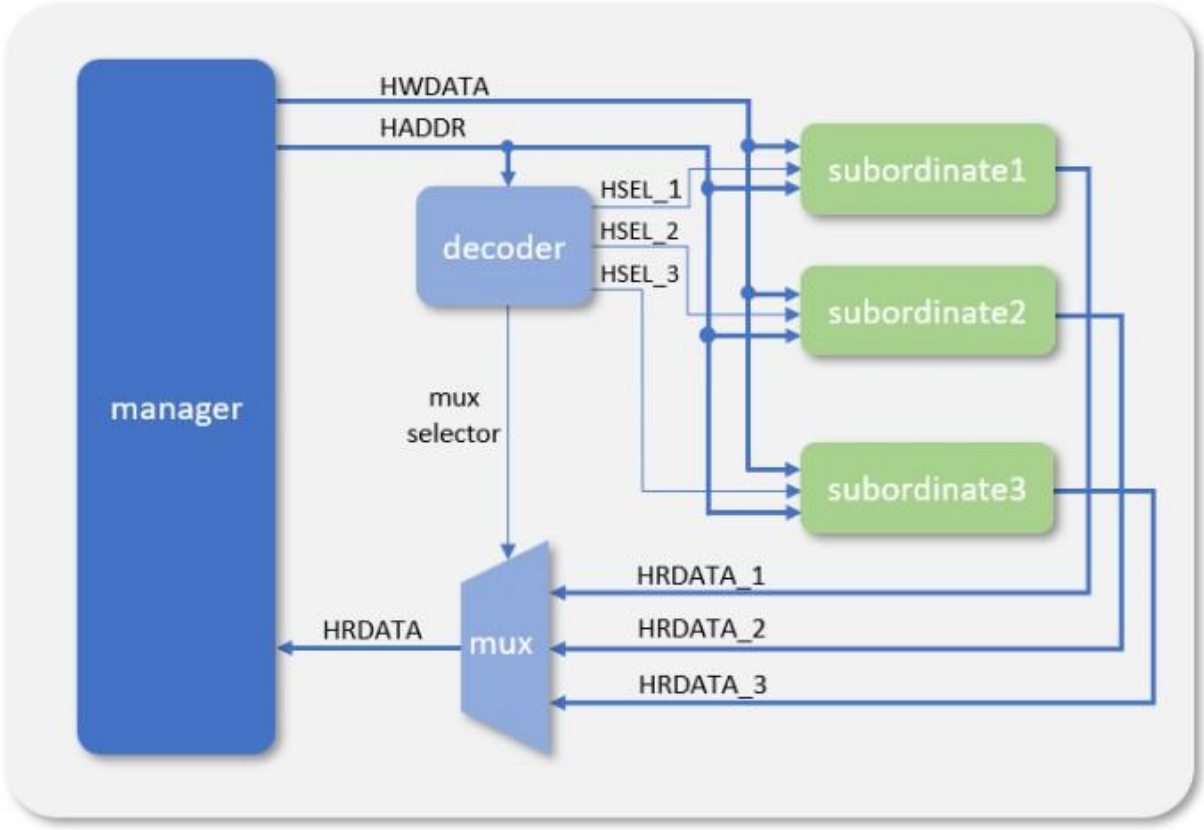
AMBA AHB Lite Protocol



Created by:

SAI PRASAD PADHI

27/1/2026

Overview of AHB-Lite

AHB-Lite is a simplified version of the **AMBA AHB (Advanced High-performance Bus)** protocol, developed by **ARM** as part of the **AMBA (Advanced Microcontroller Bus Architecture)** family. It is designed for **high-speed, high-bandwidth** communication within a system-on-chip (SoC).

Key Features:

- Single master support (unlike AHB, which allows multiple masters).
- Supports **burst transfers** (sequential data transfers).
- Supports **pipelined** operation with **separate address and data phases**.
- Simplified interface and **lower area and power** than full AHB.
- Widely used in embedded systems and microcontrollers.

Uses and Importance

Uses:

- **On-chip communication** between processor cores and memory or peripherals.
- Used in **SoC designs**, especially for **ARM Cortex-M** based microcontrollers.
- Interfaces like **AHB to APB bridges** use AHB-Lite.
- Internal interconnect of **memory-mapped peripheral devices**.

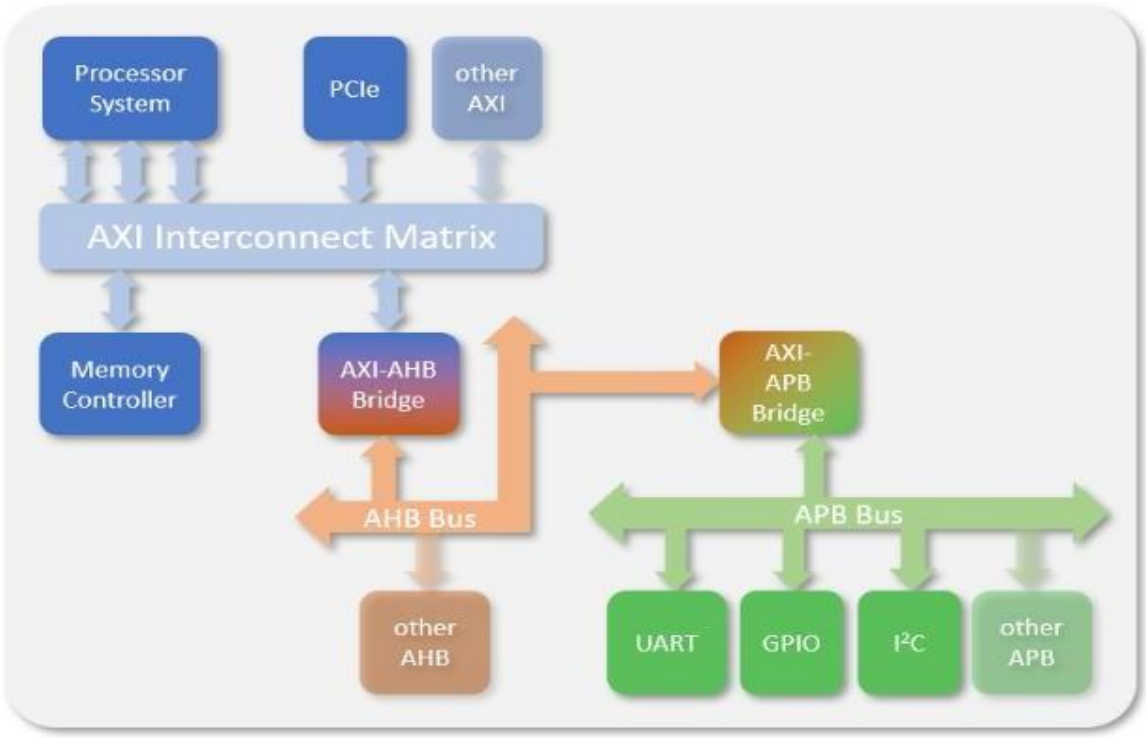
Importance:

- **Simple** yet **efficient** for single-master systems.
- Pipelining improves performance without the complexity of multi-master arbitration.
- Maintains compatibility with the **AMBA** ecosystem and IP reuse.
- Offers **predictable timing** and **deterministic** performance.

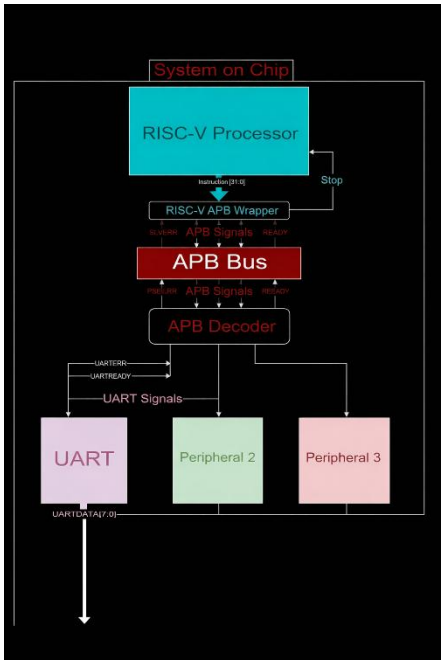
Applications of AHB-Lite

- ARM Cortex-M based microcontrollers (e.g., STM32, NXP LPC).
- Embedded SoCs with **single bus master** (CPU core).
- Internal memory/peripheral interfacing (RAM, GPIO, UART, etc.).
- Base interconnect for bridging to other buses (e.g., AHB to APB).

How do we use it?



The diagram shows the main system bus is AXI, with its interconnect fabric. A bridge connects one of the AXI ports to an AHB bus, though multiple busses might be supported. On the **AHB bus**, another bridge connects an **APB bus**, though it might have been connected directly to an AXI interconnect port. In general, though, there would be a hierarchy, with lower bandwidth and latency tolerant peripherals further away from the main system bus, and high speed, low latency peripherals nearer the main bus.



Note: if you want to check on a real life example try checking my **System on Chip Repository**, where **RISC-V processor** is connected to **multiple peripherals (UART)** using **APB bus.**

Understanding AHB-Lite Data Transfer

• Address Phase vs Data Phase Concept:

AHB-Lite operates using **pipelined bus cycles**, meaning **address and control signals for the next transfer can be issued while data is still being transferred for the current one**. This is enabled by the **address phase** and **data phase**.

Phase	Description
Address Phase	In this phase, the master places the address and control signals (`HADDR`, `HTRANS`, `HSIZE`, `HBURST`, `HWRITE`) on the bus.
Data Phase	The slave either provides `HRDATA` (read) or accepts `HWDATA` (write). This happens in the **cycle after the address phase**.

Why is this important?

Pipelining allows the next address to be issued without waiting for the current data to complete, improving throughput significantly.

• Transfer Types:

AHB-Lite supports **different types of burst transfers** to accommodate a wide range of data movement patterns. The type of transfer is determined by the **HBURST** signal.

HBURST[2:0]	Transfer Type	Description
`3'b000`	**SINGLE**	A single transfer of data.
`3'b001`	**INCR** (Incrementing)	A burst of unspecified length. Address increases after each beat.
`3'b010`	**WRAP4**	4-beat burst with wrapping. Address wraps around on boundary.
`3'b011`	**INCR4**	4-beat burst with incrementing addresses.
`3'b100`	**WRAP8**	8-beat burst with wrapping.
`3'b101`	**INCR8**	8-beat burst with incrementing addresses.
`3'b110`	**WRAP16**	16-beat burst with wrapping.
`3'b111`	**INCR16**	16-beat burst with incrementing addresses.

- **SINGLE (`HBURST = 3'b000`)**
  - Transfers **one data word** (8/16/32-bit).
  - Common in simple read/write transactions.
  - HTRANS = NONSEQ only.

- **INCR (`HBURST = 3'b001`)**
  - Used for **undefined-length bursts**.
  - **Address auto-increments** by the transfer size each cycle.
  - Requires HTRANS = NONSEQ followed by SEQ on next beats.
  - Master must signal the end using HTRANS = IDLE or HTRANS = BUSY.

- **INCR4, INCR8, INCR16 (`HBURST = 3'b011/101/111`)**
  - Fixed-length **incrementing bursts** of 4, 8, or 16 beats.
  - Address increments linearly.
  - Used when burst length is known in advance.
  - Better for optimizing bandwidth in memory systems.

- **WRAP4, WRAP8, WRAP16 (`HBURST = 3'b010/100/110`)**
  - Same as fixed-length bursts, **but with address wrapping**.
  - Useful for cache-line operations or circular buffers.
  - Address wraps back to the starting boundary after reaching the burst length.

  Example (WRAP4, 32-bit):
  Start at address `0x20`, the sequence is:
  `0x20`, `0x24`, `0x28`, `0x2C`, then wraps back to `0x20`.

- **Simple Transfer Example**
  - **Read Transfer:**

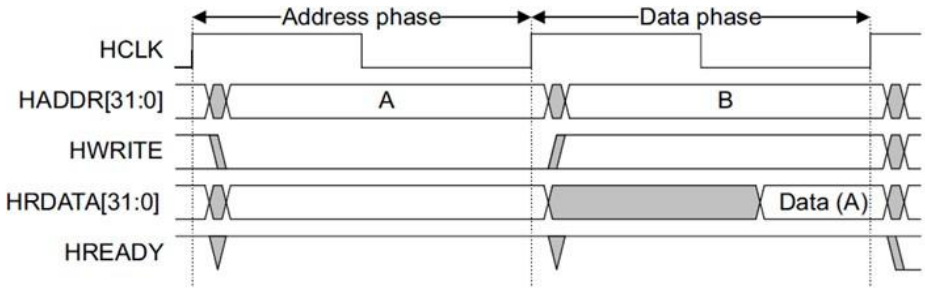


Figure 3-1 Read transfer

- **Write Transfer:**

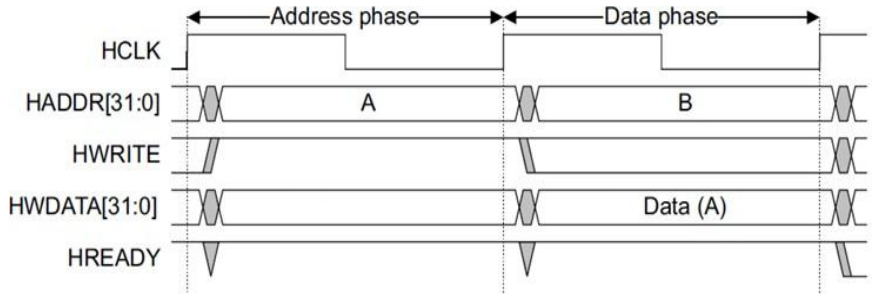


Figure 3-2 Write transfer

In a simple transfer **with no wait states**:

1. The master drives the address and control signals onto the bus after the rising edge of HCLK.

2. The slave then samples the address and control information on the **next rising edge** of HCLK.

3. After the slave has sampled the address and control it can start to drive the appropriate HREADY response.

This response is sampled by the master on **the third rising** edge of HCLK.

Our Implementation

General Structure

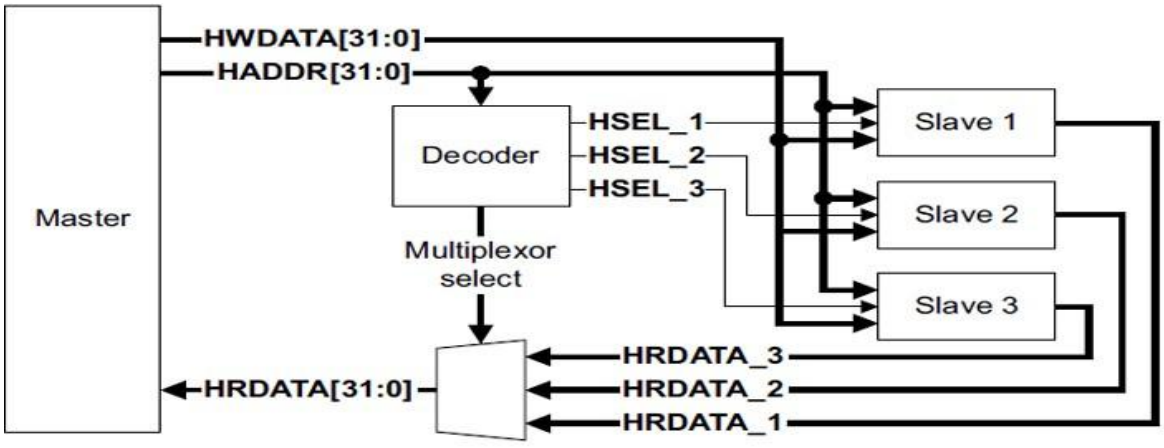
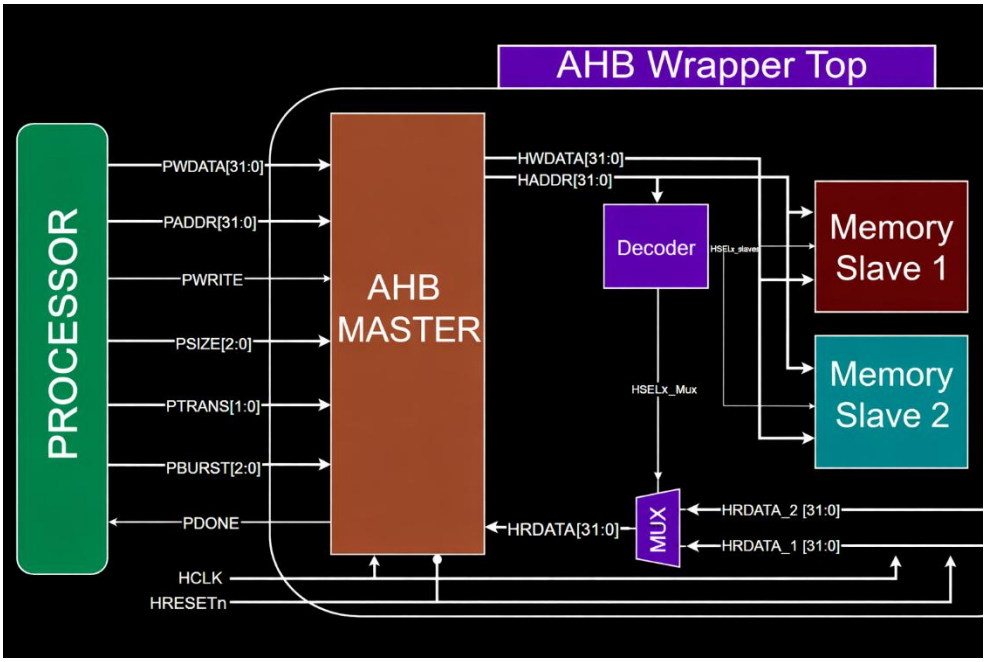


Figure 1-1 AHB-Lite block diagram

Implemented Structure



Note: In this project, we chose to implement **only the essential and most commonly used burst types**:

Supported:

- **SINGLE transfer** (`HBURST = 3'b000`)
- **INCR (incrementing burst)** (`HBURST = 3'b001`)

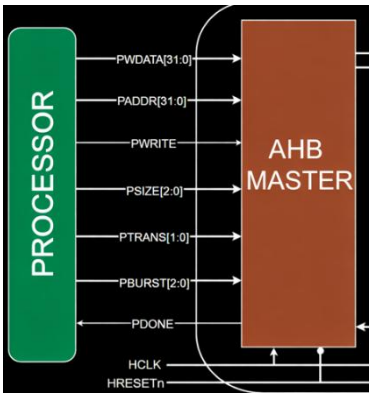
These two cover:

- Arbitrary memory access
- Multi-beat data movement
- Effective verification of address phase and data phase separation

Explanation

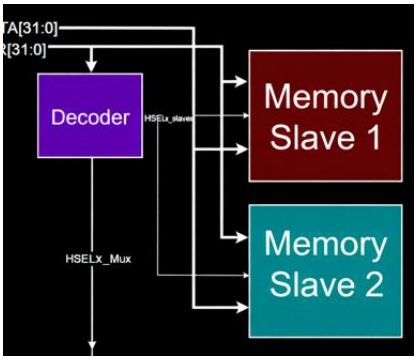
- **Processor Assumption**:
  We assume an external **processor module** that communicates with the AHB Master. This enables us to **mimic real-world scenarios** and test full interactions between:
  - Processor → AHB Master
  - AHB Master → AHB Decoder
  - AHB Decoder → AHB Slaves

- **Processor-to-Master Interface:**
  The processor issues transactions to the Master using control signals such as `PADDR`, `PWDATA`, `PWRITE`, `PTRANS`, `PBURST`, etc.
  These signals **stimulate the Master**, mimicking typical processor behavior.



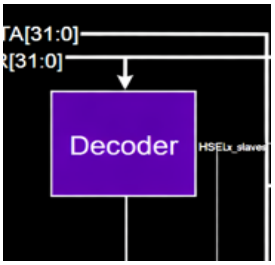
- **PDONE Signal**:
  A custom signal `PDONE` is assumed as an output from the AHB Master back to the processor.
  It acts as a **"transaction complete" flag**, going high after every successful AHB transfer, allowing the processor to:
  - Synchronize operations
  - Avoid issuing overlapping transactions

- **Dual-Slave Architecture**:
  While the original AHB documentation often shows **three or more slaves**, in our implementation we included:
  - `AHB_Slave_1`: A **regular RTL-coded slave**, acting as a larger memory for general use
  - `AHB_Slave_2`: A **smaller memory** block, specifically included to test **slave switching** and decoder functionality



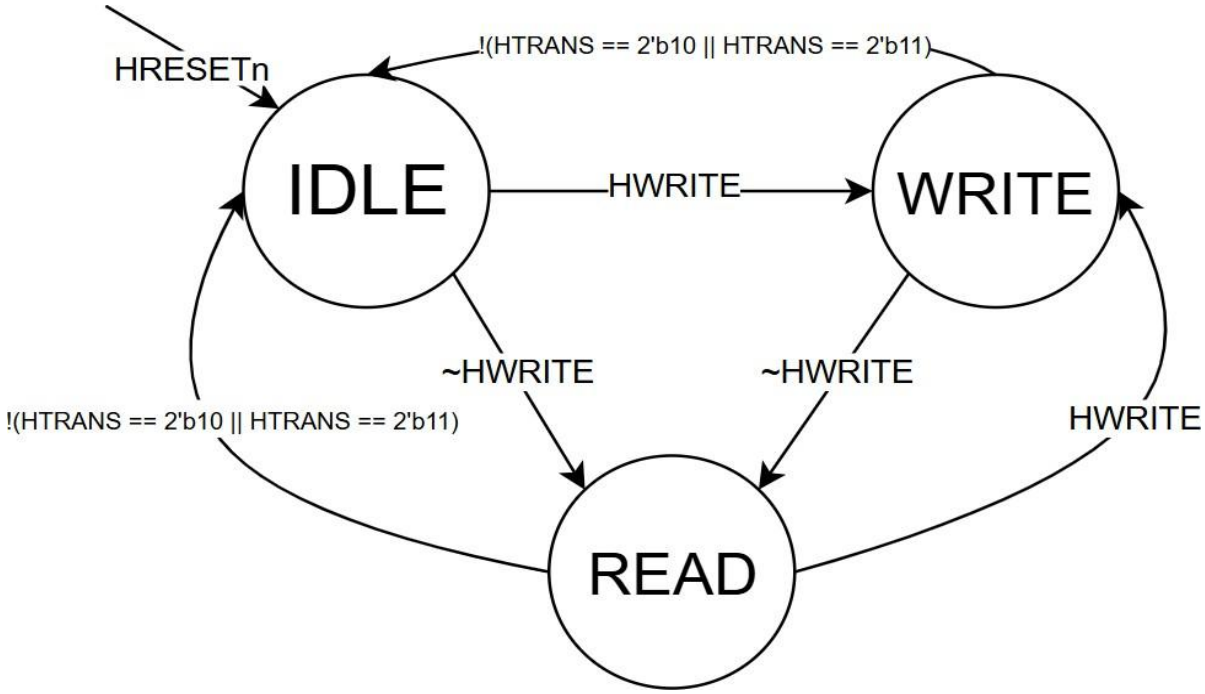
- **Slave 1 – Regular RTL Memory**:
  - Designed in a straightforward RTL style
  - Acts like a general-purpose memory
  - Receives a **full range of test scenarios** in the testbench (single and burst transfers, various sizes)

- **Decoder Logic**:
  A decoder connects the master to slaves and routes transactions using address bits:
  - `HADDR[31:28]` (or appropriate top bits) are used to generate `HSELx` signals
  - Only one slave is selected at a time, based on the address range



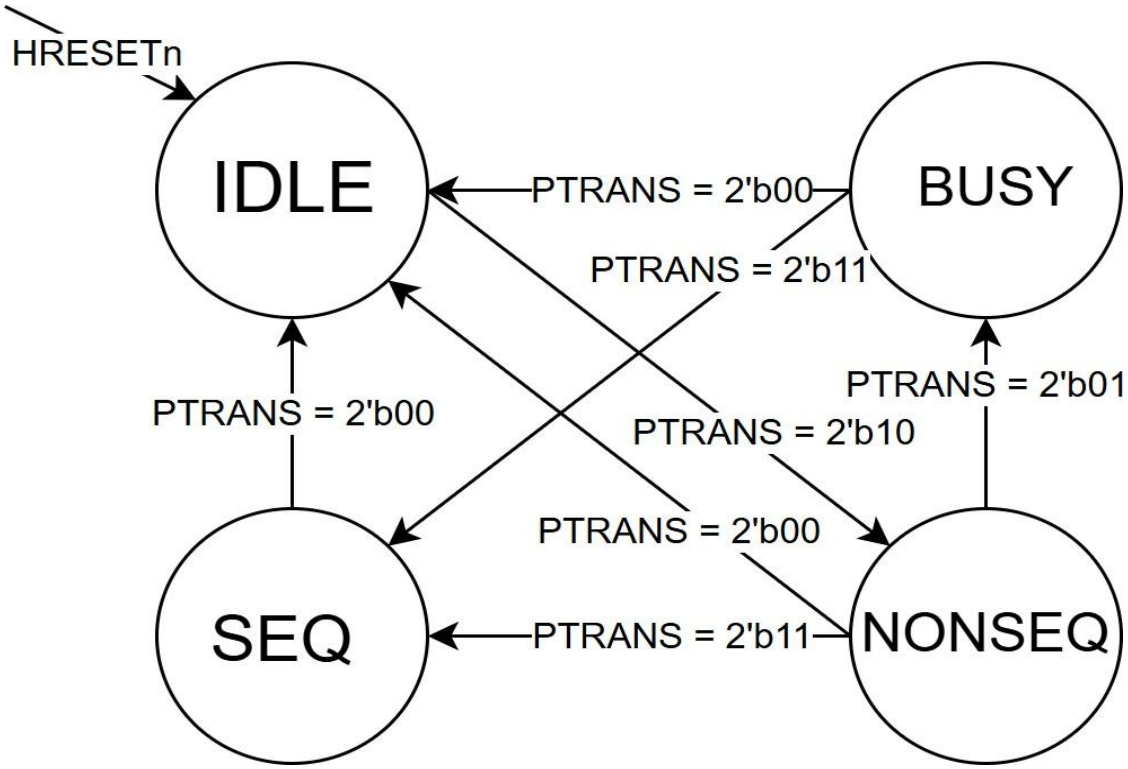
- **Slave 2 – FSM-Based Implementation:**
  - Contains a smaller memory
  - RTL implemented using a **Finite State Machine (FSM)** coding style
  - This highlights that the same AHB functionality can be realized using **alternative design approaches**
  - Receives **lighter test stimulus** in the testbench due to reduced size

  - **Slave 2 FSM**



- **Design Modularity:**
  The structure was kept modular to:
  - Simplify testing and debugging
  - Facilitate adding more slaves or peripherals in future iterations

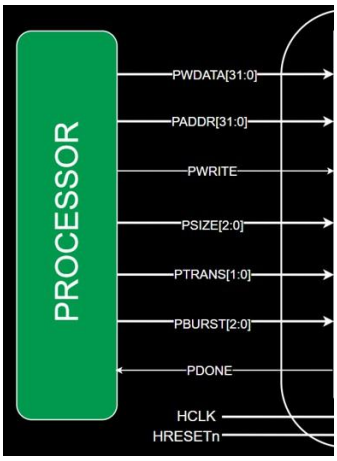
Master FSM



Verifying Functionality

Flow of AHB-Lite Testbench:

1. Mimicking the Processor



In this testbench, we simulate a **processor** that interacts with the AHB-Lite bus. Instead of hardcoding test logic inside the master, we expose control signals such as:

- PADDR: Address to access
- PWDATA: Data to write
- PWRITE: Read or write selection
- PSIZE, PTRANS, PBURST: Transaction type and size

This allows the testbench to **act like a processor**, sending address and data simultaneously, even though AHB-Lite requires a separation between address and data phases. This mimics realistic system behavior because:

- **The processor doesn't know the bus protocol**; it simply issues transactions.
- The **master handles protocol details**, breaking the transaction into address and data phases.

This setup allows you to simulate end-to-end system behavior across master, decoder, mux, and slaves.

Note: Although the testbench sends both PADDR and PWDATA in the same cycle (cycle N), the master module ensures HADDR appears at **cycle N** and HWDATA at **cycle N+1**, respecting the **address phase → data phase** separation required by the AHB-Lite protocol.

2. Waveform Comparison: Expected vs. Our Implementation

We compare **standard AMBA AHB-Lite protocol waveforms** from documentation with our simulation results. Two categories are tested:

a. Single Transfers (HBURST = 3'b000)

- Expected **Write** transfer:

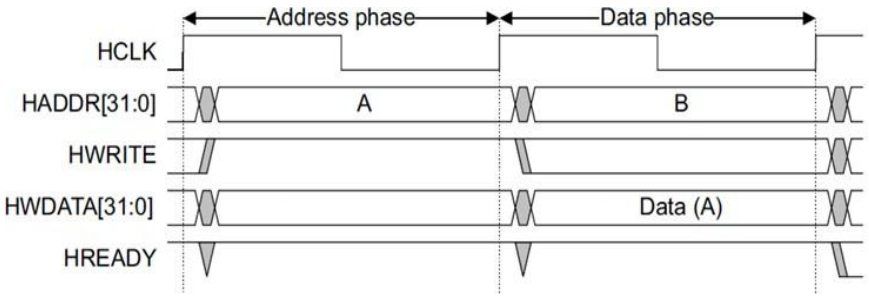
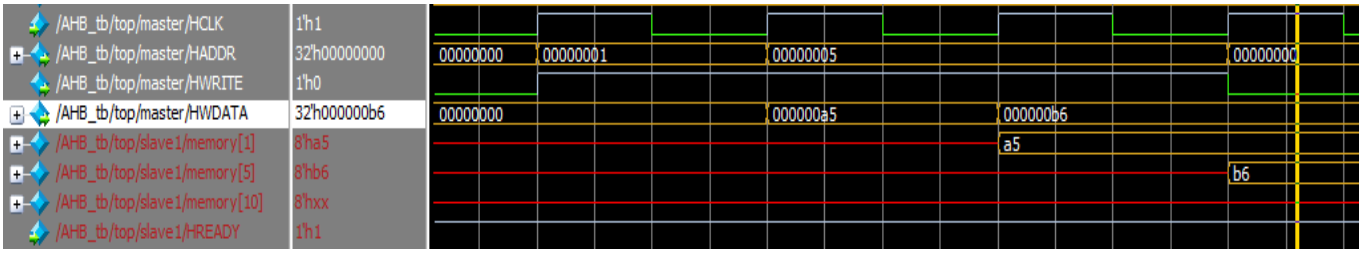


Figure 3-2 Write transfer

- Waveform Generated:



- Expected **READ** transfer:

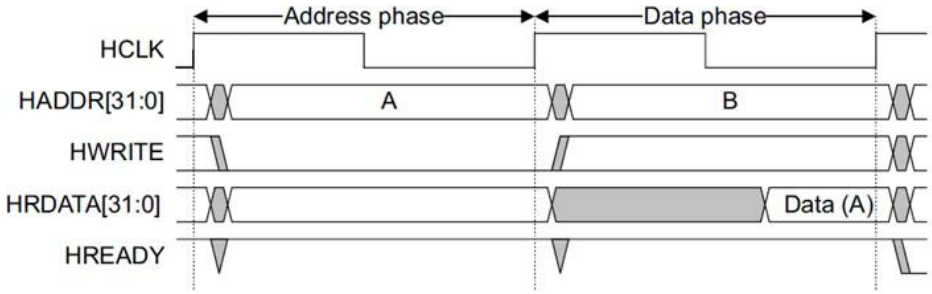
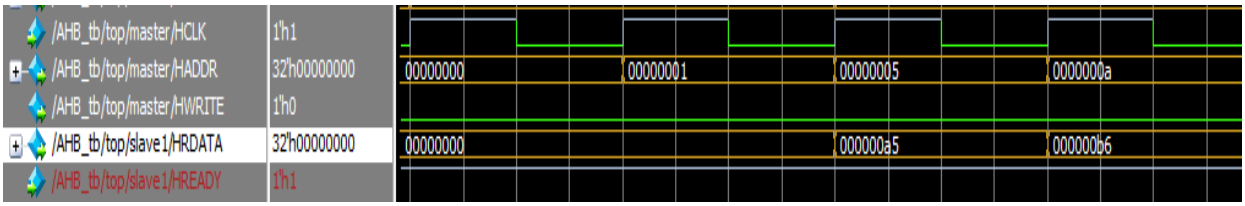


Figure 3-1 Read transfer

- Waveform Generated:



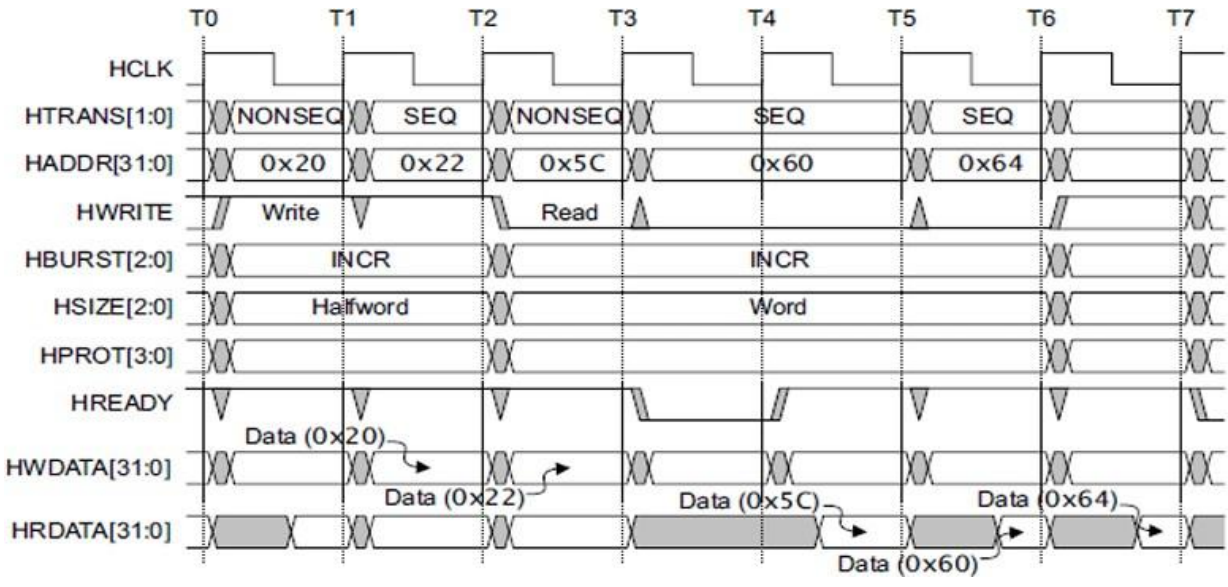
In previous two comparisons we observe:

- In Write transfer: generated waveform matches the expected waveform in the AHB documentation
- In Write Transfer: HADDR appears at **cycle N**, HWDATA at **cycle N+1** and data is accessed be the salve at **cycle N+2** exactly **like the specs stated!**
- In Read Transfer: generated waveform matches the expected waveform in the AHB documentation
- In Read Transfer: HADDR appears at **cycle N** and HRDATA at **cycle N+1** exactly **like the specs stated!**

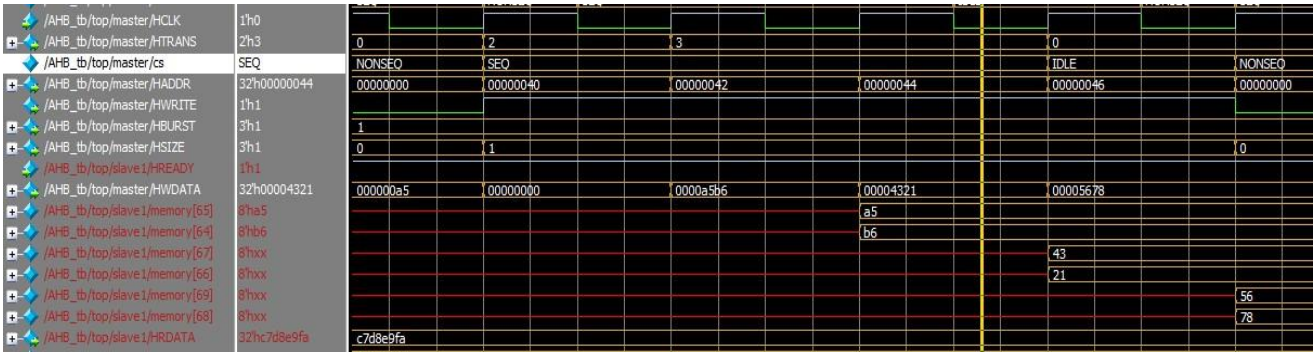
b. Incrementing Burst Transfers (HBURST = 3'b001)

- HTRANS sequence: **First NONSEQ**, then **several SEQ**
- Master **auto-increments** the address by size:
  - +1 for 8-bit
  - +2 for 16-bit
  - +4 for 32-bit

- Expected Write & Read transfers:



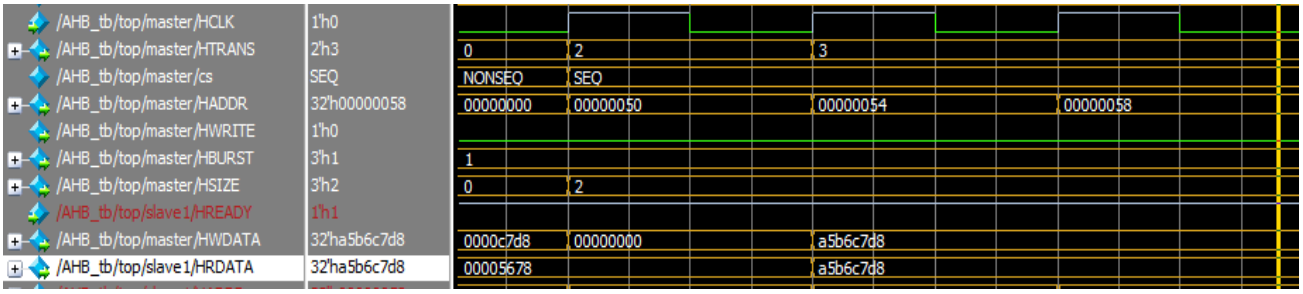
- Waveform Generated (Write):



In previous comparison we observe:

- In Write Transfer: generated waveform matches the expected waveform in the AHB documentation
- In Write Transfer: HADDR appears at **cycle N**, HWDATA at **cycle N+1** and data is accessed be the salve at **cycle N+2** exactly **like the specs stated!**
- As (HSIZE = 2'b01) this indicates that the transfer is **16 bit** transfer so we notice that the address is **incrementing by 2 every cycle!**
- In accessing memory, we see that we write 16 bits successfully in our memory in the **N+3 cycle.**

- Waveform Generated (Read):



In previous comparison we observe:

- In Read Transfer: generated waveform matches the expected waveform in the AHB documentation
- In Read Transfer: HADDR appears at **cycle N** and HRDATA at **cycle N+1** exactly **like the specs stated!**
- As (HSIZE = 2'b10) this indicates that the transfer is **32 bit** transfer so we notice that the address is **incrementing by 4 every cycle!**
- You may ask, **why the data read didn't change as we read from two different addresses**?! but the answer is that we actually wrote the **same data in these two addresses.**

3. Verification of Slave 1

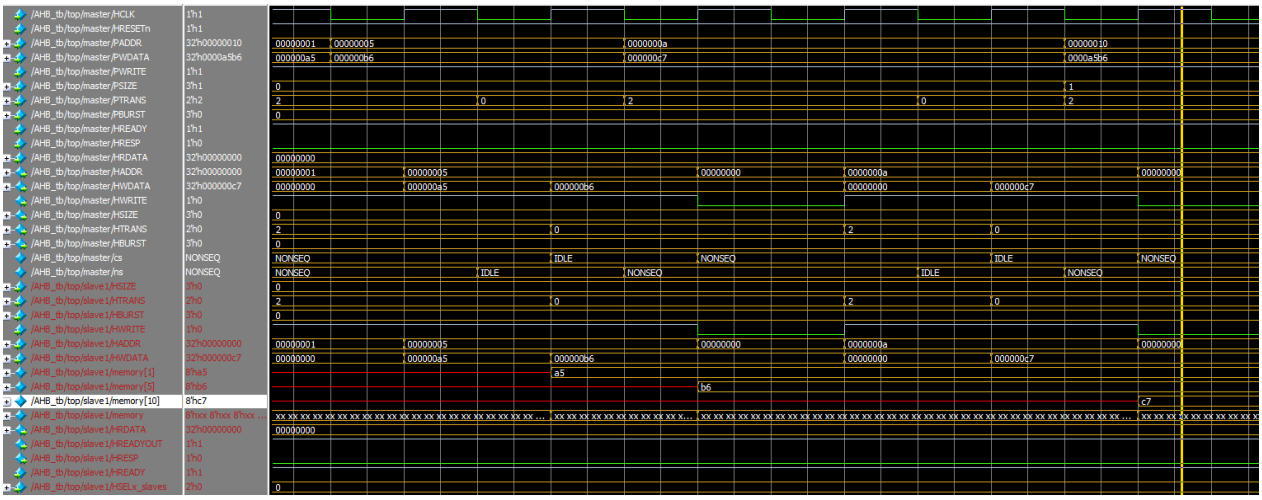
We validate all the following scenarios using **slave 1**:

- First we will write in several places using single transfer (HBURST = 0).
- Then we will read from the exact same addresses using **single transfer.**

Single Transfers:

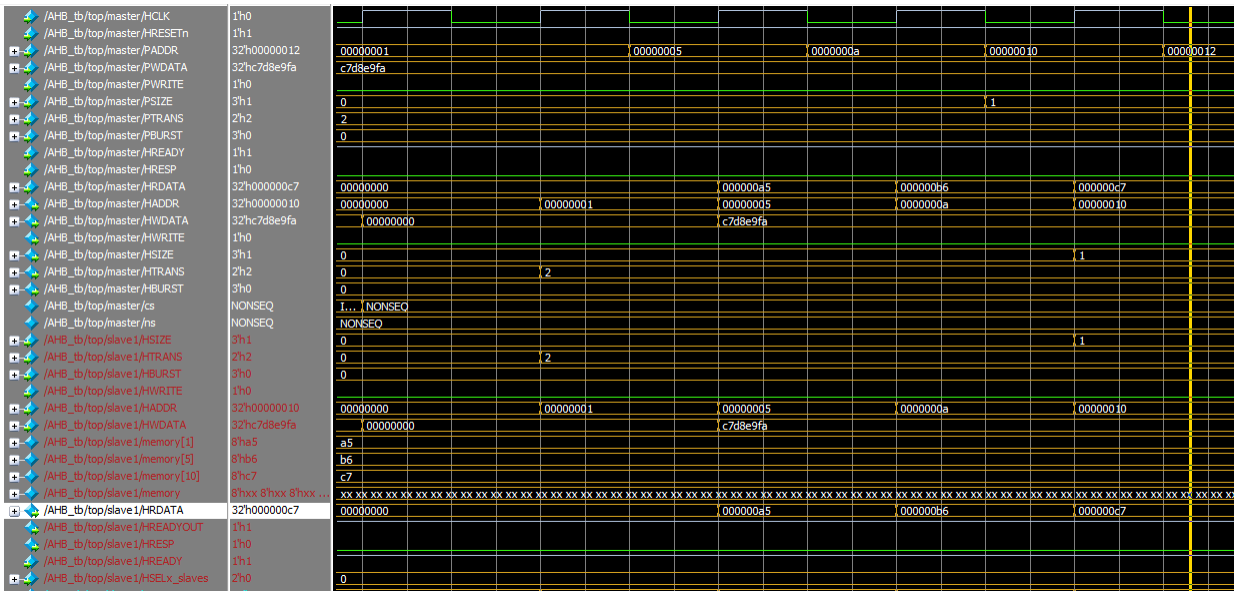
- Write & read at different addresses:
  - **8-bit:** addresses 1, 5, 10

Writing:



We notice that we wrote data in 3 different addresses, and **respecting address phase and data phase**

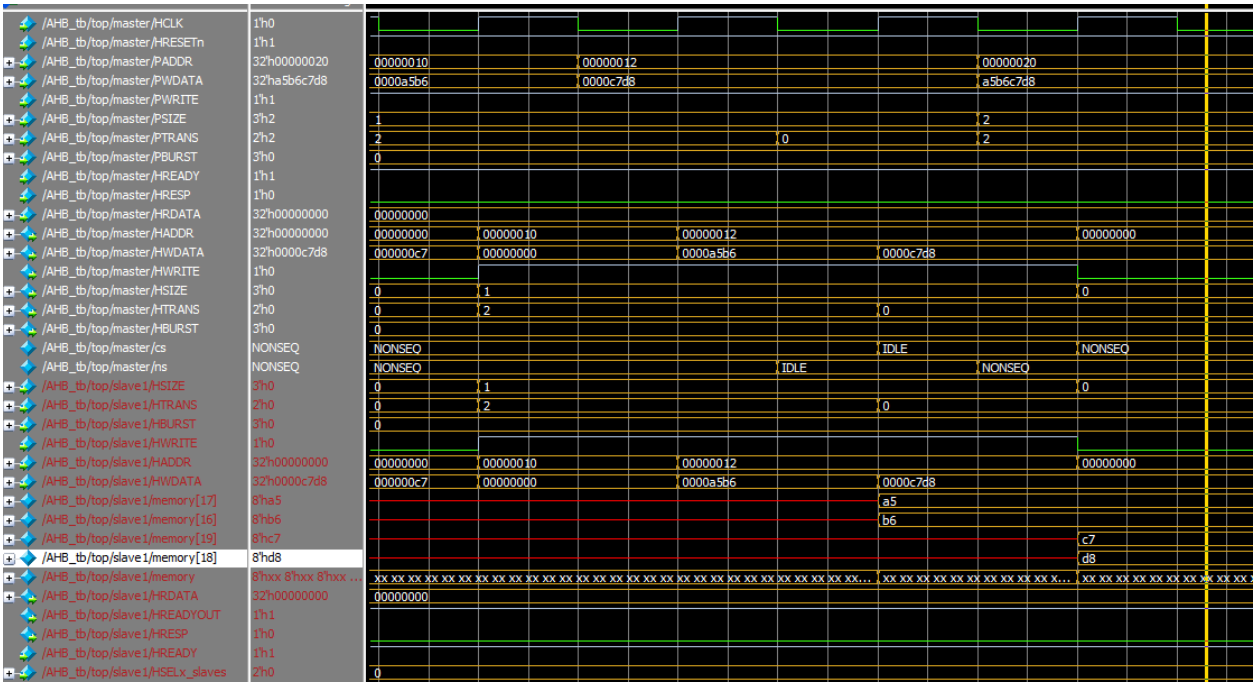
Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase (Focus on HADDR & HRDATA)**.

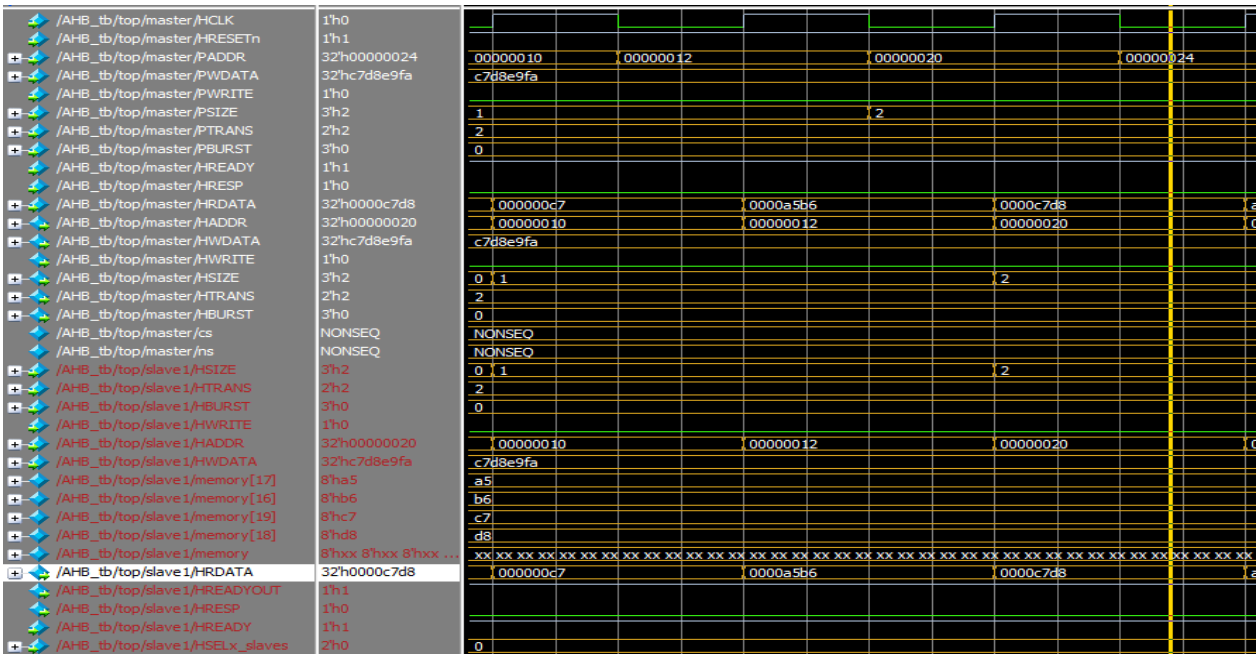
- **16-bit:** addresses 16, 18

Writing:



We notice that we wrote data in 3 different addresses, and **respecting address phase and data phase**

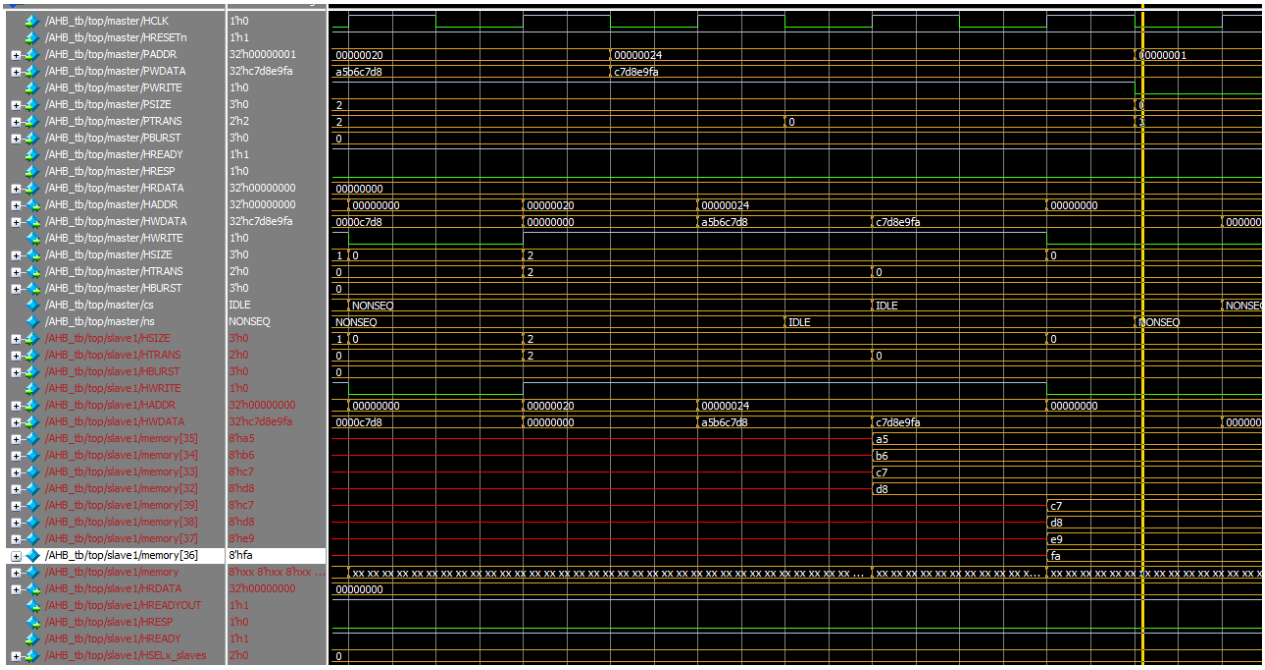
Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase (Focus on HADDR & HRDATA)**.

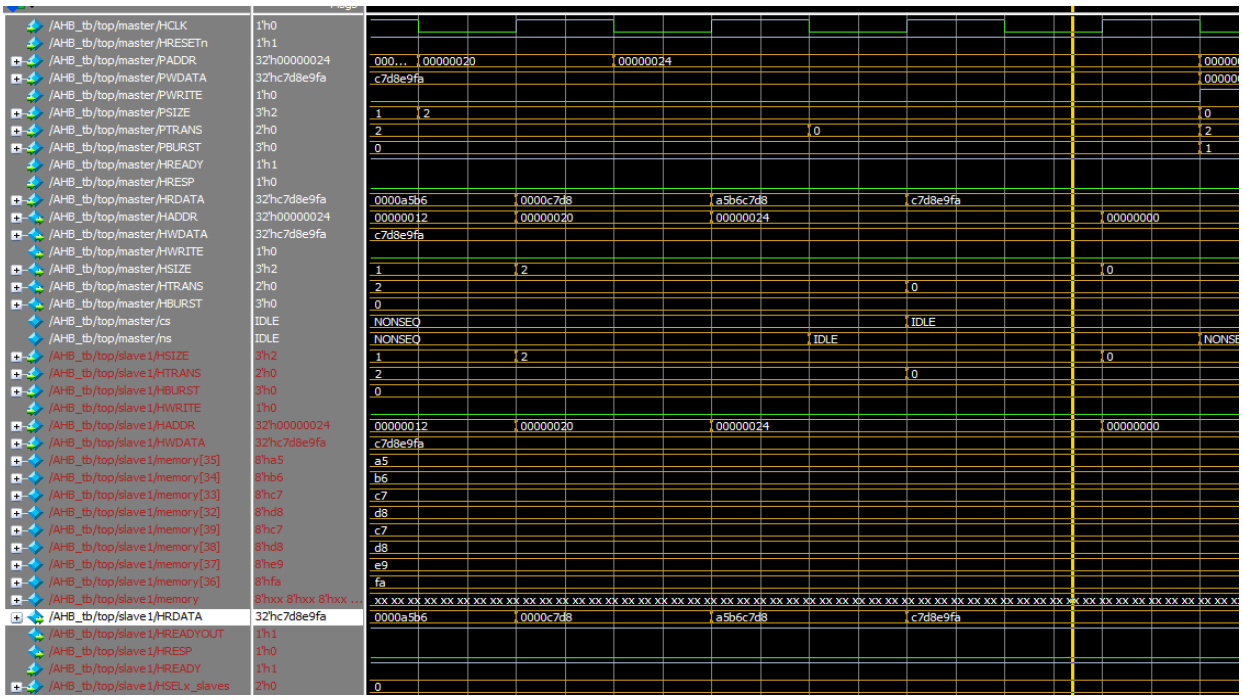
- 32-bit: addresses 32, 36

Writing:



We notice that we wrote data in 2 different addresses, and **respecting address phase and data phase**

Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase (Focus on HADDR & HRDATA)**.

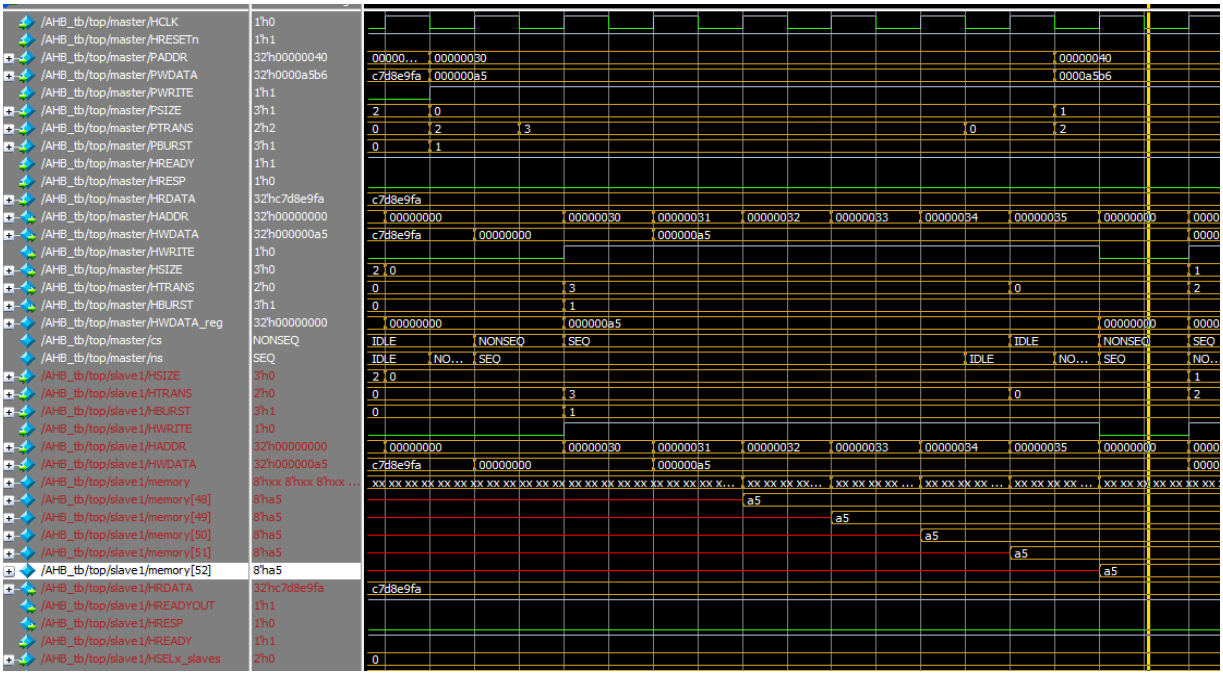
From previous waveforms we observe:

- **Data is read back and matches written data**, confirming correct storage.
- **HSELx_slaves** signal is **2'b00** all along the verification of **Slave 1**

Incrementing Burst Transfers:

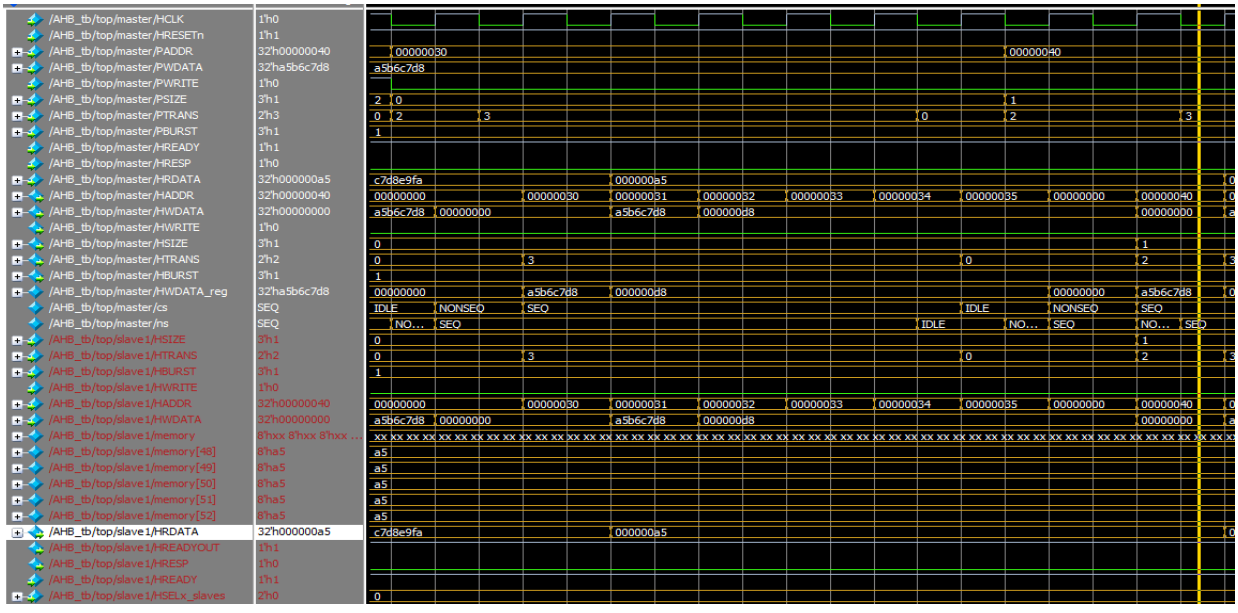
- **8-bit**: burst write/read of **5 sequential bytes** (start at address 48)

Writing:



- We notice that we wrote data in 2 different addresses, and **respecting address phase and data phase**
- Notice how the address is **being incremented by itself** as this is an **incrementing burst,** we had defined the start **address to be 0xa5** and we wrote the value 0xa5 in **five sequential addresses**

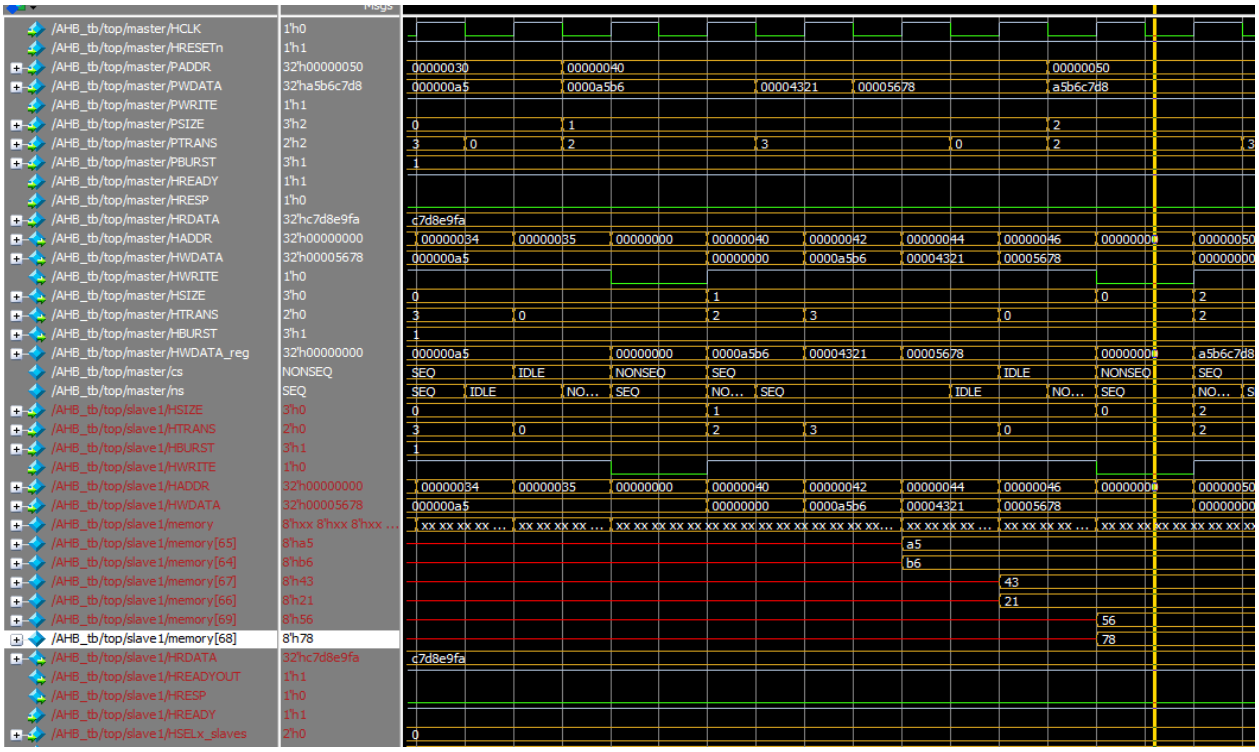
Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase (we are reading the same value (0xa5))**.

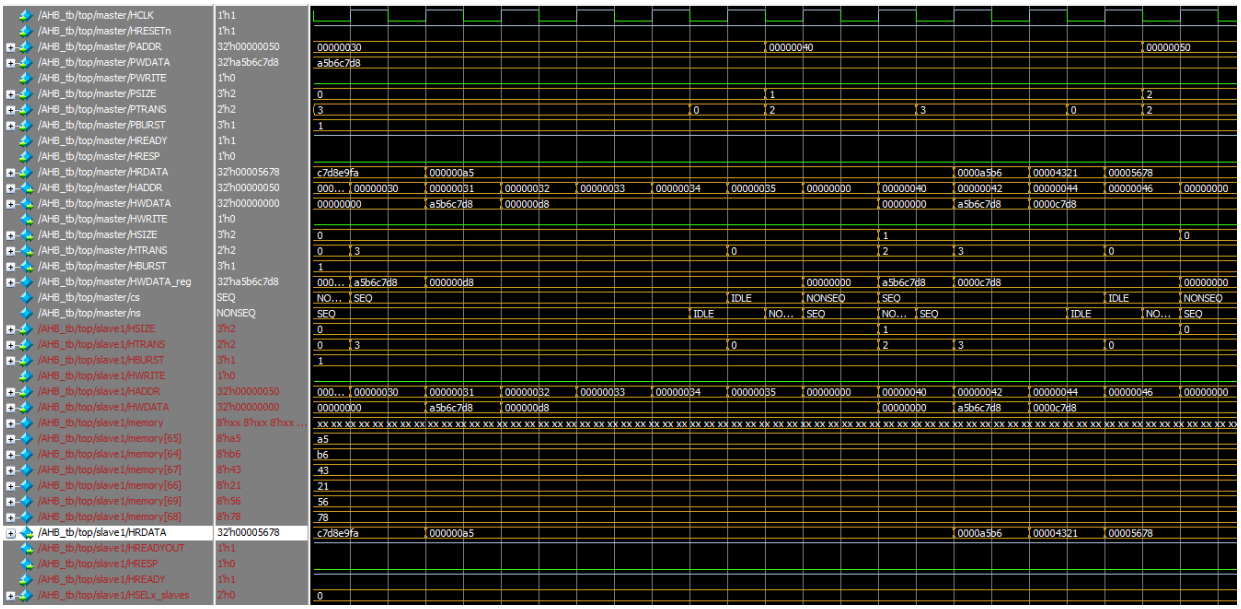
- **16-bit**: **3 sequential** halfwords (start at 64)

Writing:



- We notice that we wrote data in 3 different addresses, and **respecting address phase and data phase**
- Notice how the address is **being incremented by itself** as this is an **incrementing burst,** we had defined the start **address to be 0x40** and we wrote the different values in **three sequential addresses**

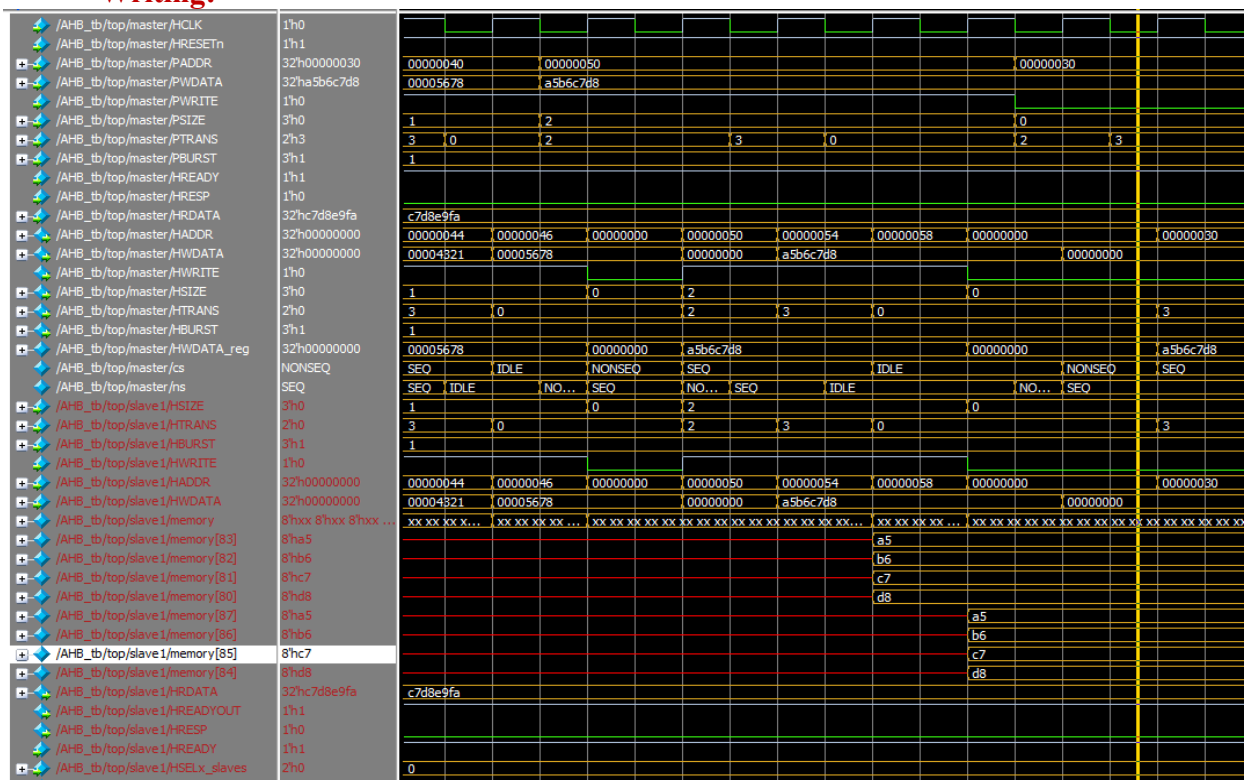
Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase (we are reading different values)**.

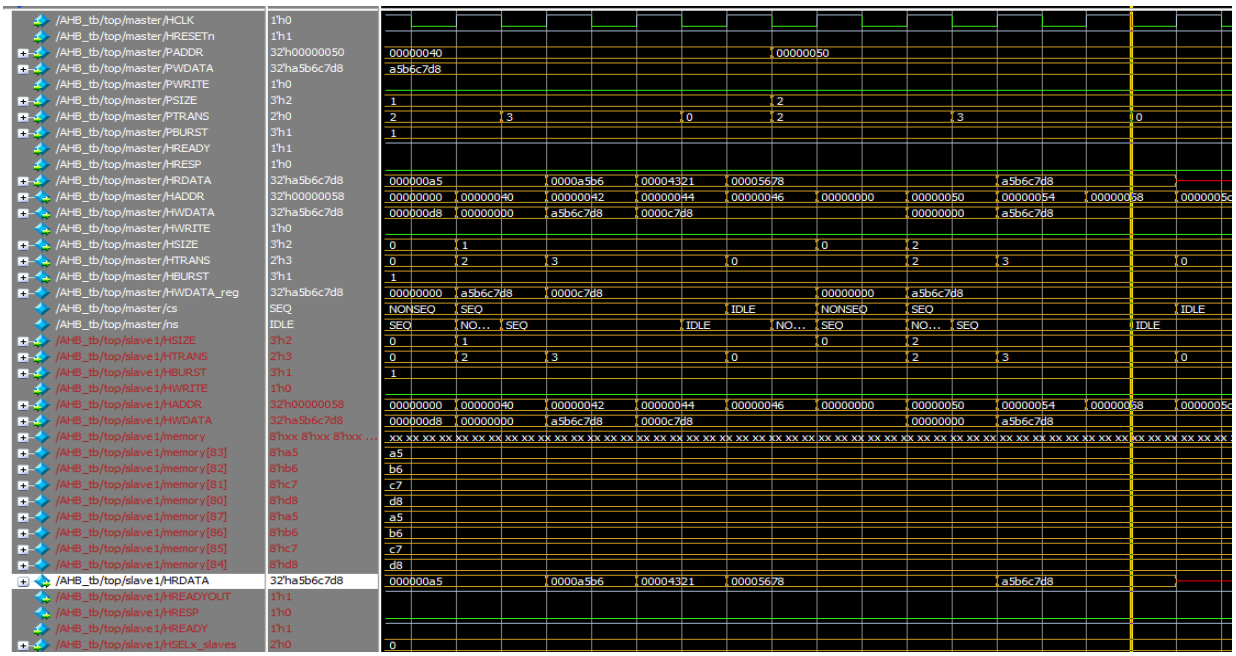
- **32-bit: 2 sequential words** (start at 80)

Writing:



- We notice that we wrote data in 2 different addresses, and **respecting address phase and data phase**
- Notice how the address is **being incremented by itself** as this is an **incrementing burst,** we had defined the start **address to be 0x50** and we wrote the value 0xa5b6c7d8 in **three sequential addresses**

Reading:



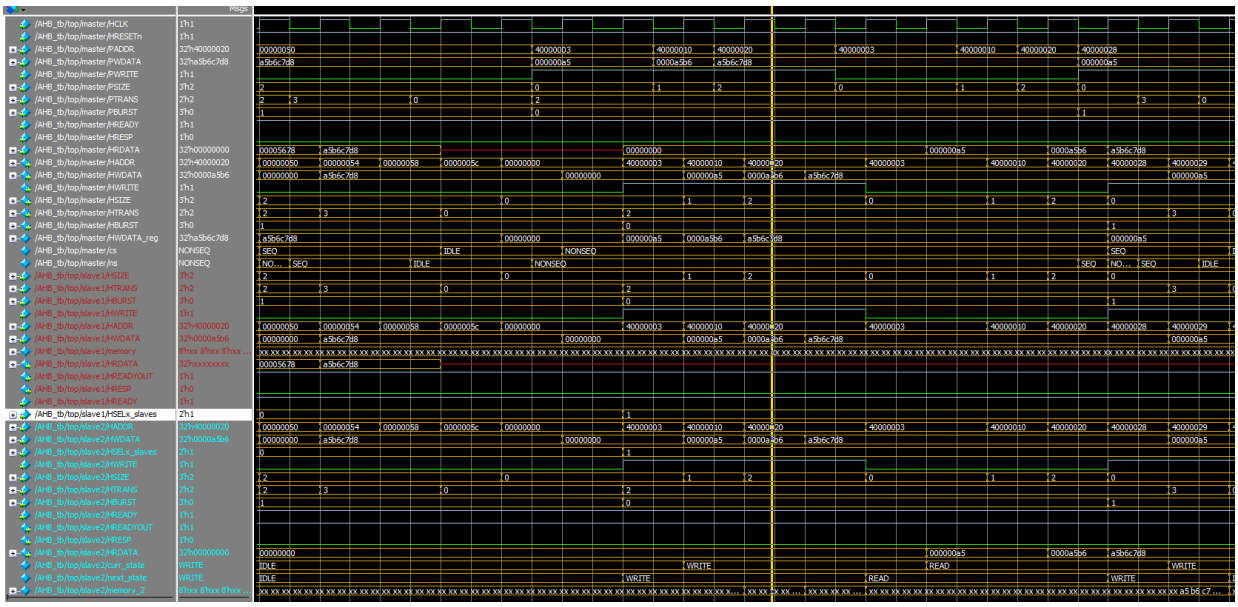
HRDATA has the same values we had written before, also we **are respecting address phase and data phase (we are reading the same value (0xa5b6c7d8))**.

From previous waveforms we observe:

- **Data is read back and matches written data**, confirming correct storage.
- **HSELx_slaves** signal is **2'b00** all along the verification of **Slave 1**
- In incrementing bursts the **address is incremented based on the** `HSIZE` **value**
- All transactions are verified via waveform observations, and `HRDATA` matches expected output.

4. Slave Switching and Decoder Verification

We change the address range to select **slave 2** (e.g., `32'h4000_0000`). The decoder logic activates the correct slave using `HSEL`.



Waveform shows `HSEL[1] = 1`, `HSEL[0] = 0` for slave 2.

From previous waveforms we observe:

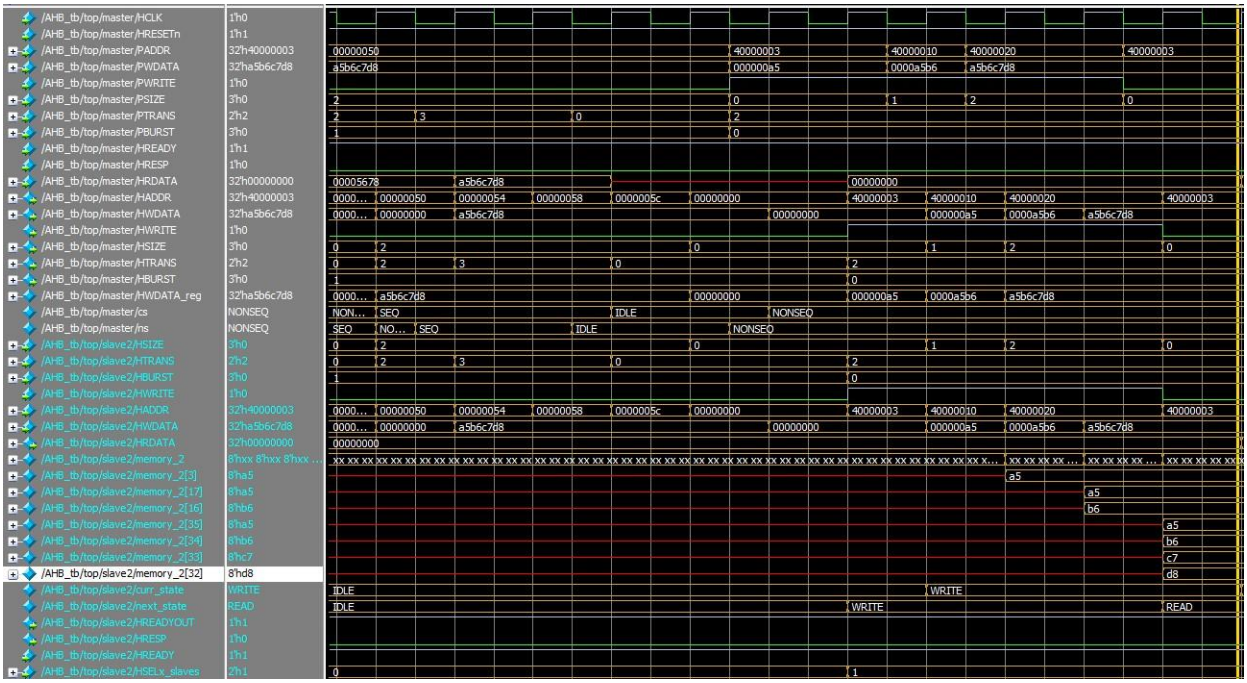
- **Smooth switching between slaves** which indicates we can insert **as many salves we want!**

5. Verifying Slave 2

Smaller memory leads to lighter testing:

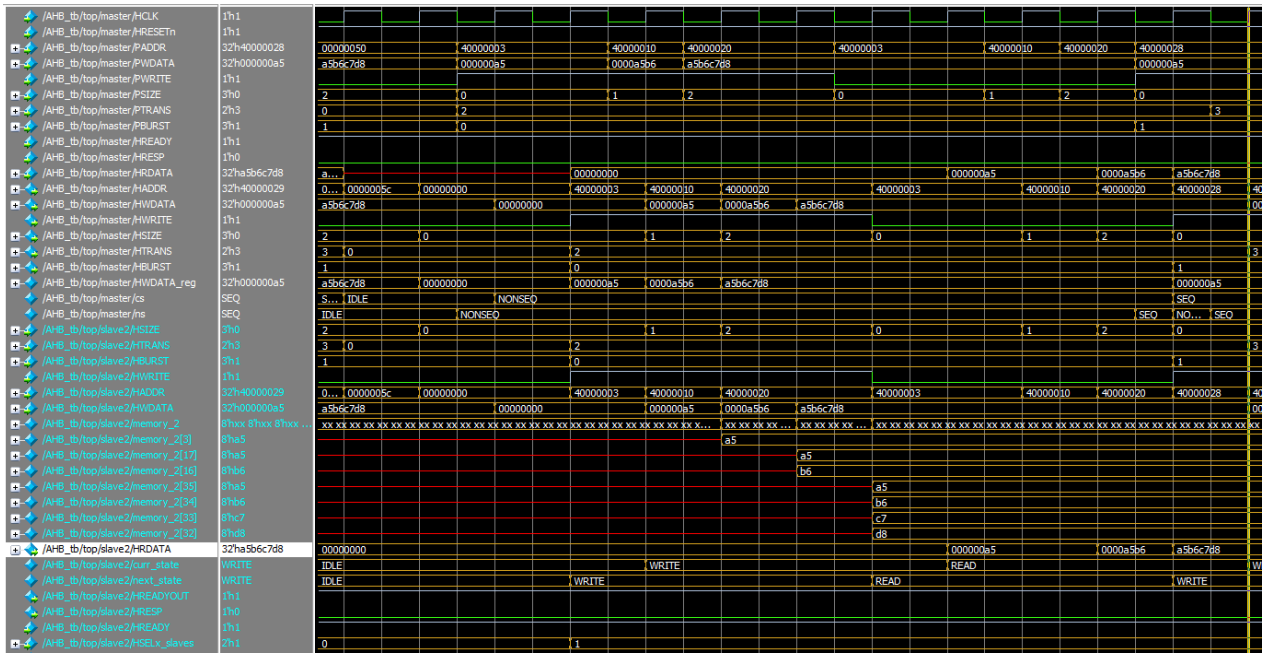
- 8-, 16-, and 32-bit **single transfer write/read**

Writing:



We notice that we wrote data in 3 different addresses, and **respecting address phase and data phase**

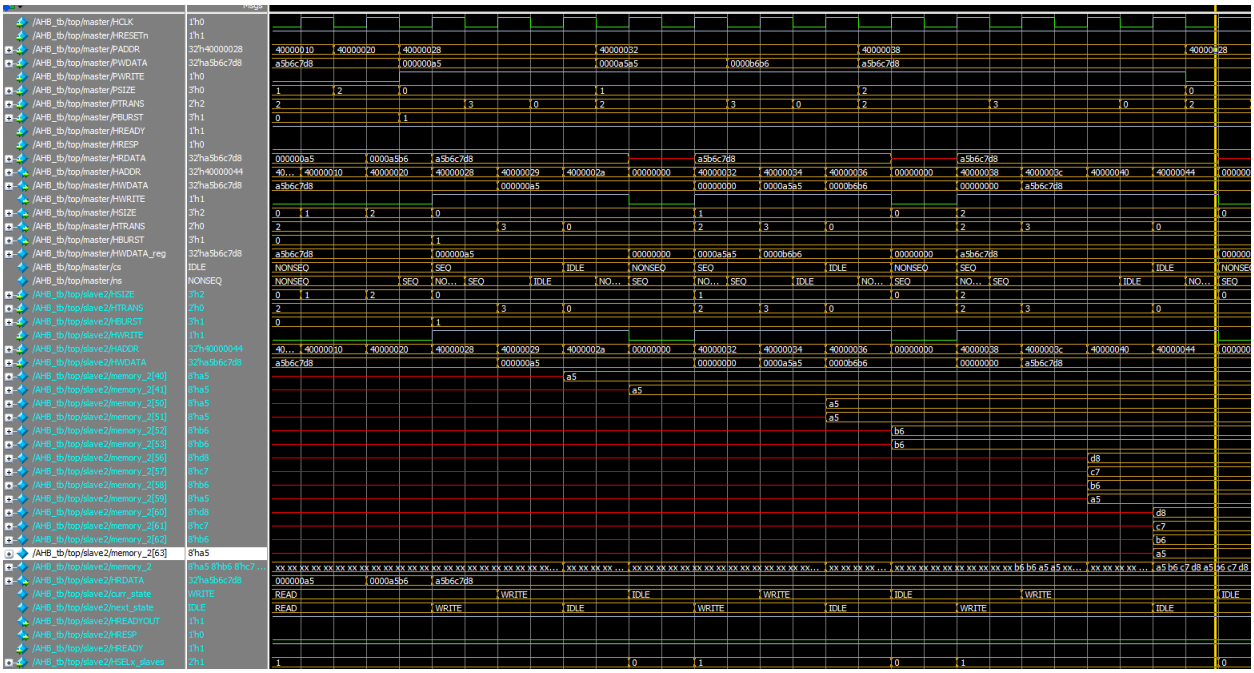
Reading:



HRDATA has the same values we had written before, also we **are respecting address phase and data phase** **(we are reading the different values)**.

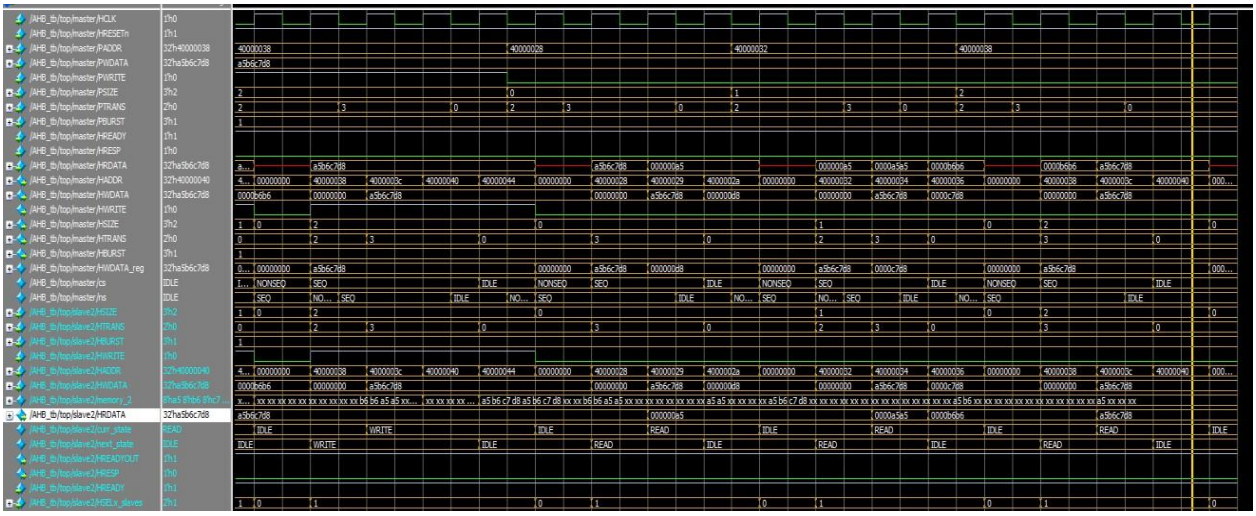
- 8-, 16-, and 32-bit **incrementing burst write/read**

Writing:



- We notice that we wrote data in 3 different addresses, and **respecting address phase and data phase**
- Notice how the address is **being incremented by itself** based on HSIZE value**,** if it's 0 **then address increments by 1**, if it's 1 **then address increments by 2**.

Reading:

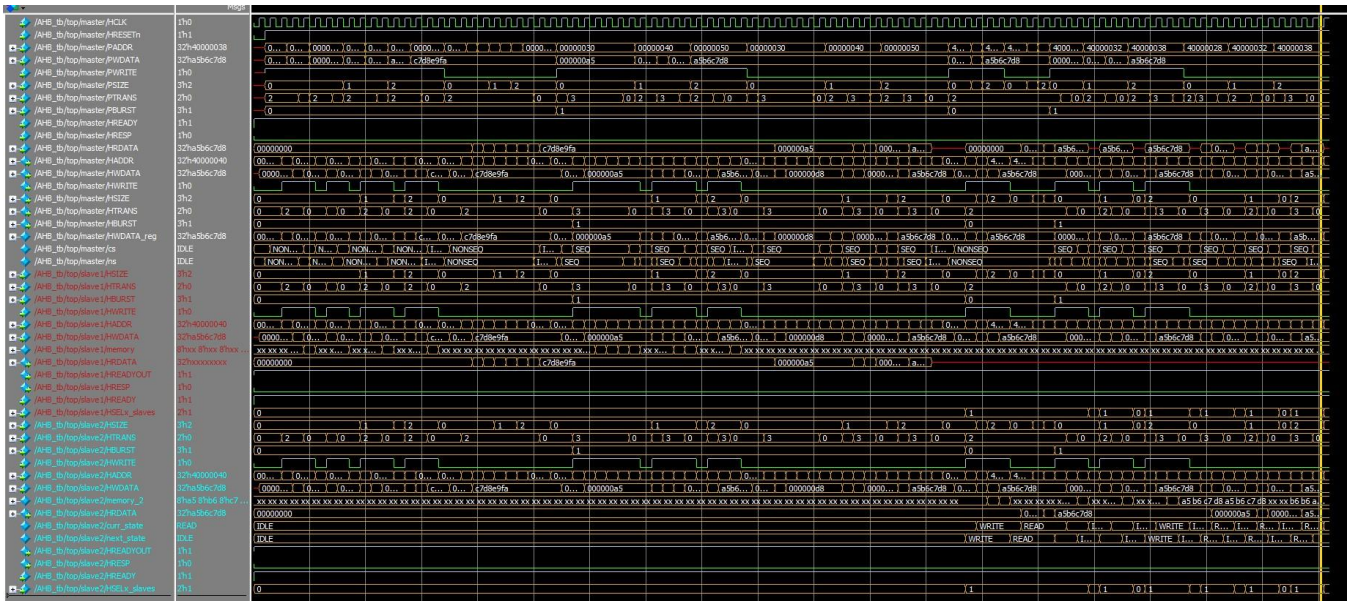


HRDATA has the same values we had written before, also we **are respecting address phase and data phase (we are reading the different values)**.

From previous waveforms we observe:

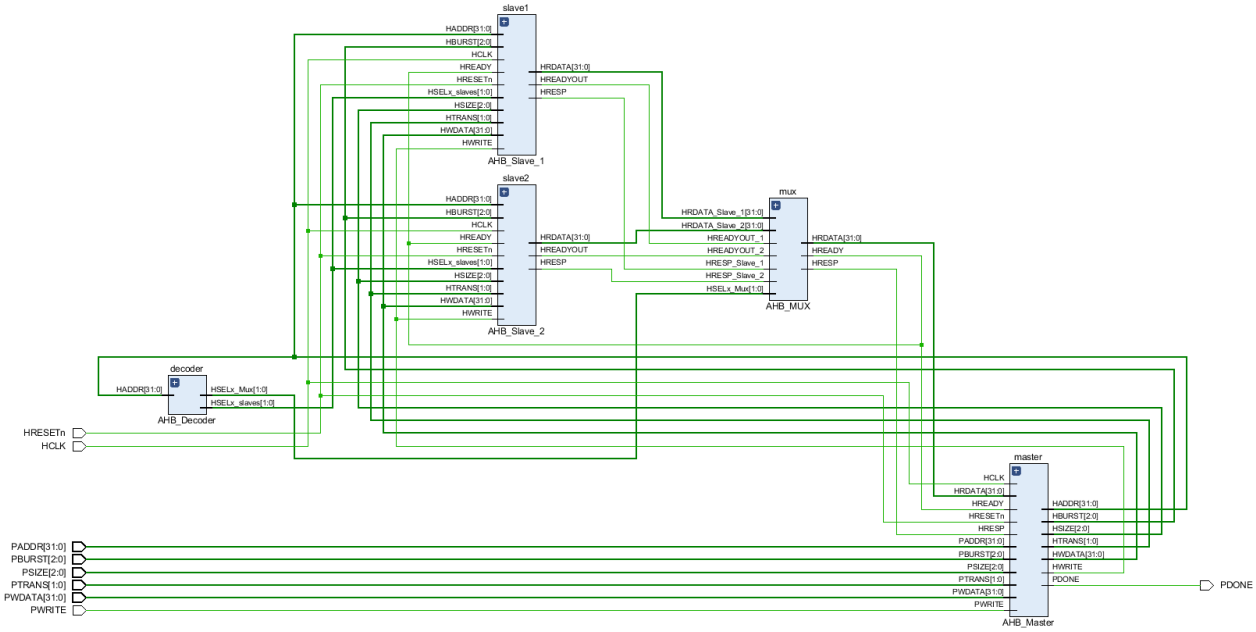
- **Data is read back and matches written data**, confirming correct storage.
- **HSELx_slaves** signal is **2'b01** all along the verification of **Slave 1** (except when IDLE state).
- In incrementing bursts the **address is incremented based on the** `HSIZE` **value**

6. Full Waveform

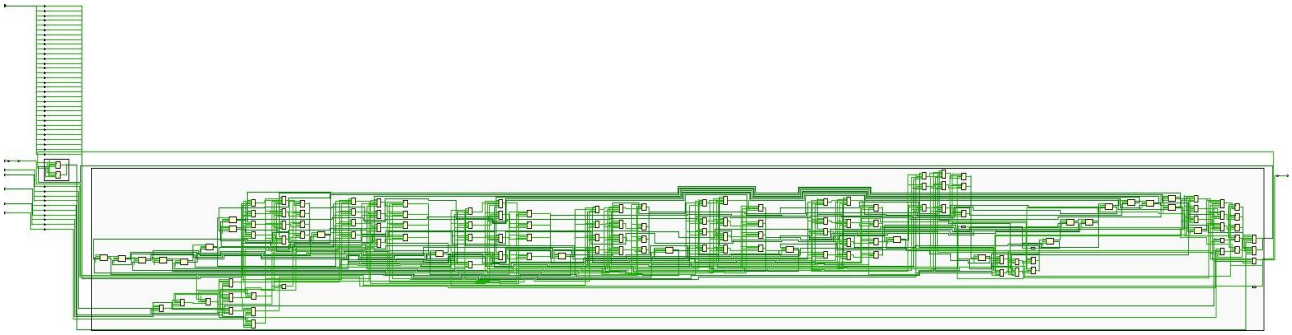


FPGA Flow using Vivado

- **Elaboration**



- **Synthesis**



References

- AMBA 3 AHB-Lite Protocol Specification from University of Michigan
- Simon Southwell's LinkedIn Post "SoC Bus and Interconnect Protocols #1: Busses (APB and AHB)"