CHAPTER 1

INTRODUCTION

The Advanced High-performance Bus (AHB) protocol is a widely used on-chip bus protocol in the field of digital electronics and computer architecture. It is part of the Advanced Microcontroller Bus Architecture (AMBA) developed by ARM Holdings. The AHB protocol is designed to facilitate efficient communication between different components, such as microprocessors, memory modules, and peripheral devices, within a system-on-chip (SoC) or a microcontroller.

The AHB protocol provides a set of rules and guidelines that define how data and control signals are transferred between the master and slave devices connected to the bus. It establishes a standardized interface that allows for the interconnection and communication of various IP (intellectual property) blocks within a system, promoting compatibility and interoperability.

1.1 About the protocol

AMBA AHB is a bus interface suitable for high-performance synthesizable designs. It defines the interface between components, such as masters, interconnects, and slaves.

AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- Burst transfers.
- Single clock-edge operation.
- Non-tristate implementation.
- Wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB slaves are internal memory devices, external memory interfaces, and high-bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB slaves, for system performance reasons, they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between the higher performance AHB and APB is done using an AHB slave, known as an APB bridge.

Figure 1-1 shows a single master AHB system design with the AHB master and three AHB slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master.

AHB also supports multi-master designs by the use of an interconnect component that provides arbitration and routing signals from different masters to the appropriate slaves.

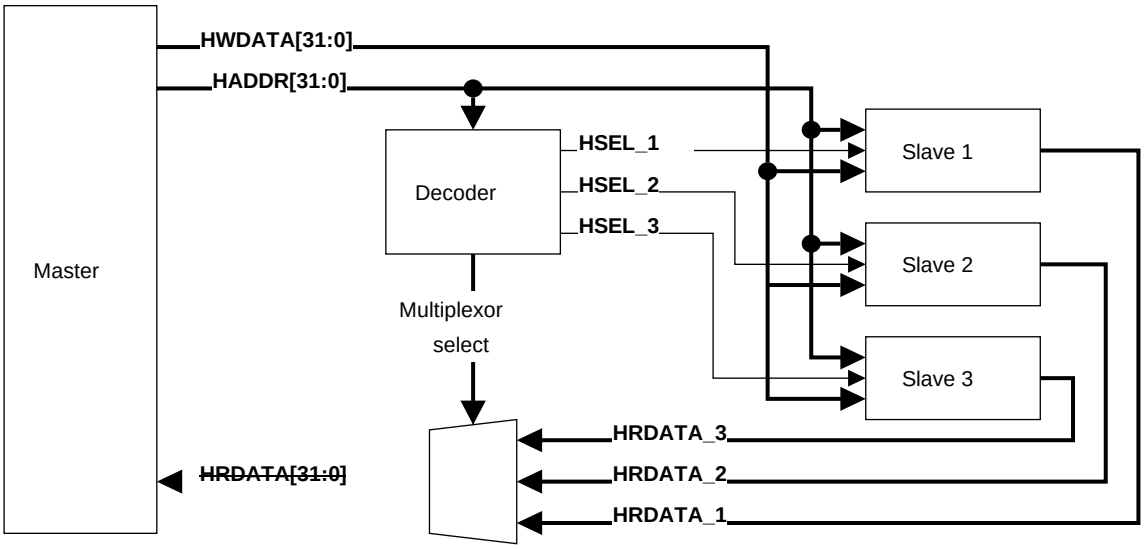

Figure 1.1 – AHB Block Diagram

Figure shows only the main address and data buses and typical data routing. Not all signals are shown.

1.1.1 Master

A master provides address and control information to initiate read and write operations. Figure shows a master interface

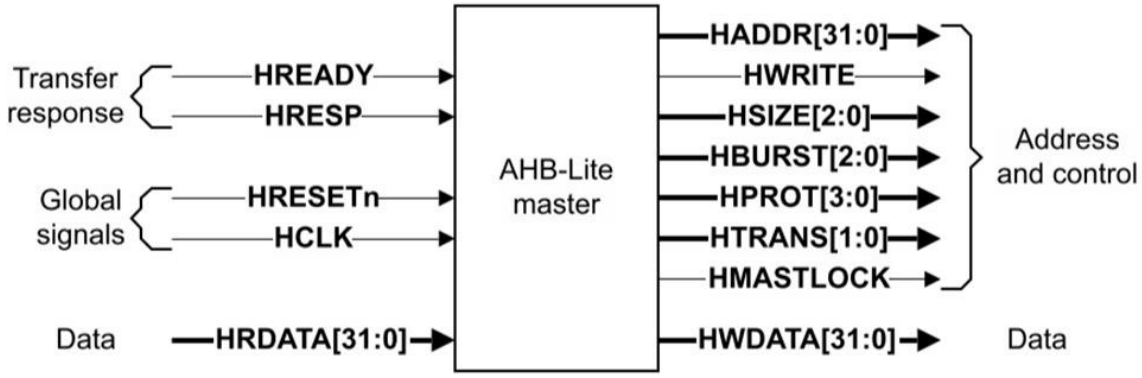

Figure 1.2 – Master Interface

1.1.2 Slave

A slave responds to transfers initiated by masters in the system. The slave uses the HSELx select signal from the decoder to control when it responds to a bus transfer. The slave signals back to the master:

- The completion or extension of the bus transfer.
- The success or failure of the bus transfer.

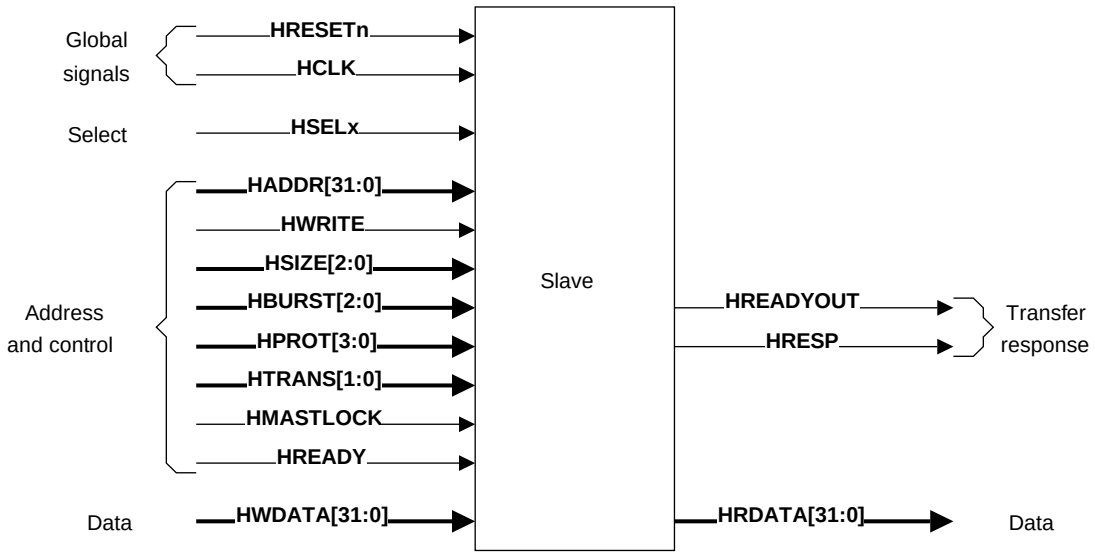

Figure 1.3 – Slave Interface

1.1.3 Interconnect

An interconnect component provides the connection between masters and slaves in a system.

A single master system only requires the use of a Decoder and Multiplexor, as described in the following sections.

A multi-master system requires the use of an interconnect that provides arbitration and the routing of signals from different masters to the appropriate slaves. This routing is required for address, control, and write data signaling. Further details of the different approaches used for multi-master systems, such as single layer or multi-layer interconnects, are not provided within this specification.

See Multi-layer AHB Technical Overview (ARM DVI 0045) for more information about implementing a multi-layer AHB-Lite interconnect.

Decoder

This component decodes the address of each transfer and provides a select signal for the slave that is involved in the transfer. It also provides a control signal to the multiplexor.

A single centralized decoder is required in all implementations that use two or more slaves.

Multiplexor

A slave-to-master multiplexor is required to multiplex the read data bus and response signals from the slaves to the master. The decoder provides control for the multiplexor.

1.2 Operation

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be:

- Single.
- Incrementing bursts that do not wrap at address boundaries.
- Wrapping bursts that wrap at particular address boundaries.

The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master.

Every transfer consists of:

Address phase One address and control cycle.

Data phase One or more cycles for the data.

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using HREADY. This signal, when LOW, causes wait states to be inserted into the transfer and enables the slave to have extra time to provide or sample data.

The slave uses HRESP to indicate the success or failure of a transfer.

.

CHAPTER 2

LITERATURE SURVEY

[1] **AMBA AHB Protocol specification – ARM.**

[2] "**Design and Verification of AMBA AHB**" by Dr. Priyanka Choudhury and Perumalla Giridhar. The first international conference on advanced technologies in intelligent control, the environment, computing, and communication is taking place in 2019. (ICATIECE).

In the group of AMBA transports, the AHB is a superior exhibition transport. It is a convention for a framework's modules to speak with each other. ARM has made AHB principles that permit the association between the off-chip outer memory interface and the on-chip memory processor. The AHB that upholds one expert and four slaves is introduced, planned, and checked in this review. The central AHB Convention building blocks — the Decoder, Slave, Expert, and Multiplexers — are created in this review. The AMBA-AHB convention can be involved by the application as long as it agrees with AHB models. Verilog is utilized to plan the decoders, expert, multiplexers, and slaves.

[3] "**Effective Design and Implementation of AMBA AHB Bus Protocol Using Verilog"**
by Deeksha L and Shivakumar B.R Conference on Intelligent Sustainable Systems in 2019 (ICISS).

Plan and execution of AMBA AHB are examined in this work. Transport connector engineering essentially affects how well the PC performs. The transmission of information and directions between fringe gadgets and memory, or between the memory and central processor, may be hampered by a framework transport that is inadequately fabricated. The on chip correspondence standard for superior execution microcontrollers are laid out under the "High level Microcontroller Transport Design (AMBA) determination." The AMBA show family incorporates AHB. The AMBA AHB is planned for framework modules with high clock rates and incredible execution. The "AHB" represents the superior exhibition spine transport of the framework. Low-power full scale cell fringe assignments can be successfully associated with central processors, with the assistance of AHB outer off-chip memory and on-chip memorie's can be connected. AHB peripherals are easy to incorporate into any plan stream since all sign advances just impact the rising clock edge.

[4] **Design and Verification of AHB Protocol using Universal Verification Methodology (UVM)**

An assortment of rules for correspondence between particular semiconductor parts is known as the on-chip show. The most famous convention being used today in microcontrollers and Soc plans is AMBA. AMBA conventions are ordered into various gatherings in light of their qualities and mechanical turns of events. The three primary transports that make up AMBA are the High level Elite Exhibition Transport, the High level Framework Transport and the High level Fringe Transport. Since AHB is better than the other two transports concerning execution, transfer speed, and similarity with framework modules with high clock frequencies, the framework planners pick AHB as their essential choice. In the VLSI field, affirmation is likewise a vital figure deciding if a methodology is powerful and meets models. The plan of the AHB show, which empowers a solitary master and a few slaves in Verilog and affirms utilizing gear affirmation dialects like Structure Verilog and standard Frameworks like Broad Really take a look at Method, is the fundamental focal point of this review (UVM). The plan is reenacted and confirmed utilizing an EDA instrument named Questasim, a Guide Designs Progressed confirmation device.

[5] **Design of I2C Master Core with AHB Protocol for High Performance Interface**

A block that speaks with any IC and the processor is depicted in this paper. By and large, this requires two unmistakable in the middle between. To try not to utilize outer peripherals, nonetheless, both are consolidated here. Consequently to join the advantages of IIC and AHB as far as speed and execution, the Between Incorporated Circuit ace module was added to the AHB convention. At the point when these components are thought of, it is found that the essential expert module (AHB) has 100 percent utilitarian inclusion. Utilizing the two-stage cycle approach, the information are gained in a contention and sans traffic way.

[6] **Verification of AHB Protocol using UVM**

This paper depicts how the UVM (Widespread Confirmation Technique) was utilized to check the AHB Convention. The High level Superior Presentation Framework Transport (AHB) is fit for supporting different bosses and slaves. Burst moves, split exchanges, single-clock edge activity, and a more extensive information transport setup are completely made conceivable by it. Check IP offers a sharp procedure to approve AHB parts including Slave, Mediator, Expert and Decoder. The best system for accomplishing CDV (Inclusion Driven Confirmation), which limits the time spent confirming a plan by incorporating UVM is utilized to affirm the AHB Convention, which naturally produces tests, self-checking test seats, and inclusion measurements. Confirmation Parts, generally alluded to as VCs, are the structure blocks of an UVM test seat. This study centers around VCs that were created to work with Verilog, Framework C, and VHDL.

CHAPTER 3

SIGNAL DESCRIPTIONS

3.1 Global signals

Name	Source	Description
HCLK	Clock source	The bus clock times all bus transfers. All signal timings are related to the rising edge of **HCLK**..
HRESETn	Reset controller	The bus reset signal is active LOW and resets the system and the bus. This is the only active LOW signal..

Table 3.1 – Global Signals

3.2 Master signals

Name	Destination	Description
HADDR[31:0]	Slave and decoder	The 32-bit system address bus.
HBURST[2:0]	Slave	The burst type indicates if the transfer is a single transfer or forms part of a burst. Fixed length bursts of 4, 8, and 16 beats are supported. The burst can be incrementing or wrapping. Incrementing bursts of undefined length are also supported.
HMASTLOCK	Slave	When HIGH, indicates that the current transfer is part of a locked sequence. It has the same timing as the address and control signals.
HPROT[3:0]	Slave	The protection control signals provide additional information about a bus access and indicate how an access should be handled within a system. The signals indicate if the transfer is an opcode fetch or data access, and if the transfer is a privileged mode access or a user mode access.

HPROT[6:4]	Slave	The 3-bit extension of the **HPROT** signal that adds extended memory types. This signal extension is supported if the AHB5 Extended_Memory_Types property is True.
HSIZE[2:0]	Slave	Indicates the size of the transfer, that is typically byte, halfword, or word. The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HNONSEC	Slave and decoder	Indicates that the current transfer is either a Non-secure transfer or a Secure transfer. This signal is supported if the AHB5 Secure_Transfers property is True.
HEXCL	Exclusive Access Monitor	Exclusive Transfer. Indicates that the transfer is part of an Exclusive access sequence. This signal is supported if the AHB5 Exclusive_Transfers property is True.
HTRANS[1:0]	Slave	Indicates the transfer type of the current transfer. This can be: • IDLE • BUSY • NONSEQUENTIAL • SEQUENTIAL.
HWDATA[31:0][a]	Slave	The write data bus transfers data from the master to the slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this can be extended to enable higher bandwidth operation.
HWRITE	Slave	Indicates the transfer direction. When HIGH this signal indicates a write transfer and when LOW a read transfer. It has the same timing as the address signals, however, it must remain constant throughout a burst transfer.

Table 3.2 – Master Signals

3.3 Slave signals

Name	Destination	Description
HRDATA[31:0][a]	Multiplexor	During read operations, the read data bus transfers data from the selected slave to the multiplexor. The multiplexor then transfers the data to the master. A minimum data bus width of 32 bits is recommended. However, this can be extended to enable higher bandwidth operation.
HREADYOUT	Multiplexor	When HIGH, the **HREADYOUT** signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer.
HRESP	Multiplexor	The transfer response, after passing through the multiplexor, provides the master with additional information on the status of a transfer. When LOW, the **HRESP** signal indicates that the transfer status is OKAY. When HIGH, the **HRESP** signal indicates that the transfer status is ERROR.
HEXOKAY	Multiplexor	Exclusive Okay. Indicates the success or failure of an Exclusive Transfer. This signal is supported if the AHB5 Exclusive_Transfers property is True.

Table 3.3 – Slave Signals

3.4 Decoder signals

Name	Destination	Description
HSEL*x*[a]	Slave	Each slave has its own slave select signal **HSELx** and this signal indicates that the current transfer is intended for the selected slave. When the slave is initially selected, it must also monitor the status of **HREADY** to ensure that the previous bus transfer has completed, before it responds to the current transfer. The **HSELx** signal is a combinatorial decode of the address bus.

Table 3.4 -Decoder Signals

3.5 Multiplexor signals

Name	Destination	Description
HRDATA[31:0]	Master	Read data bus, selected by the decoder.
HREADY	Master and slave	When HIGH, the **HREADY** signal indicates to the master and all slaves, that the previous transfer is complete.
HRESP	Master	Transfer response, selected by the decoder.
HEXOKAY	Master	Exclusive okay, selected by the decoder.

Table 3.5 – Multiplexor Signals

CHAPTER 4

TRANSFERS

4.1 Basic transfers

A transfer consists of two phases:

Address Lasts for a single HCLK cycle unless its extended by the previous bus transfer.

Data Might require several HCLK cycles. Use the HREADY signal to control the number of clock cycles required to complete the transfer.

HWRITE controls the direction of data transfer to or from the master. Therefore, when:

• HWRITE is HIGH, it indicates a write transfer and the master broadcasts data on the write data bus, HWDATA[31:0]

• HWRITE is LOW, a read transfer is performed and the slave must generate the data on the read data bus, HRDATA[31:0].

The simplest transfer is one with no wait states, so the transfer consists of one address cycle and one data cycle. First figure shows a simple read transfer and second figure shows a simple write transfer.

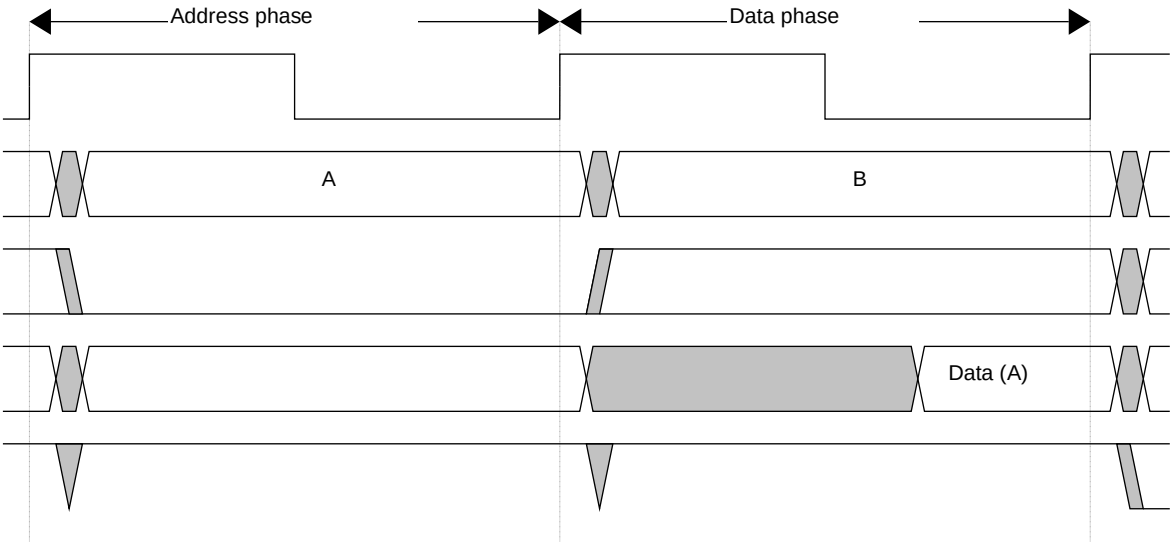

Figure 4.1 – Read Transfer

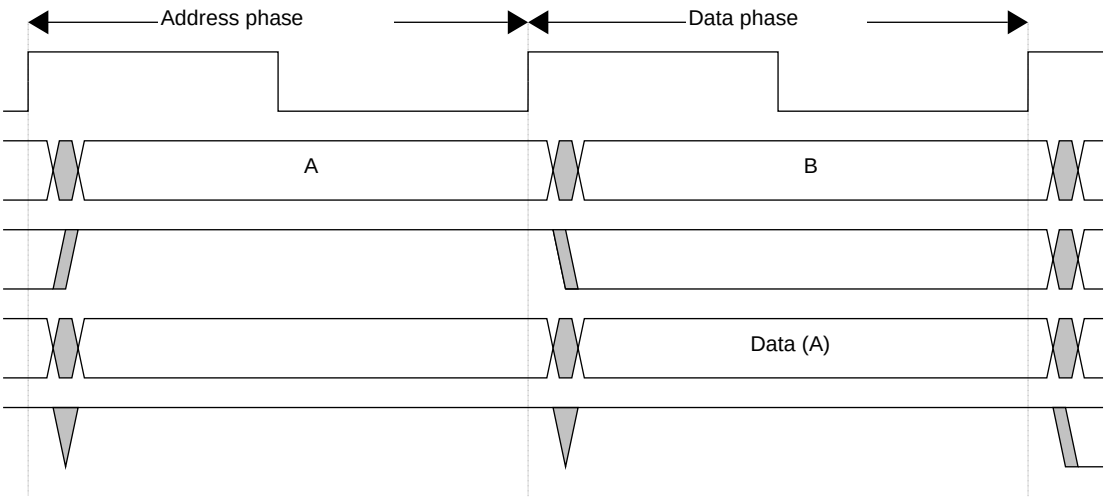

Figure 4.2 – Write Transfer

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

This simple example demonstrates how the address and data phases of the transfer occur during different clock cycles. The address phase of any transfer occurs during the data phase of the previous transfer. This overlapping of address and data is fundamental to the pipelined nature of the bus and enables high performance operation while still providing adequate time for a slave to provide the response to a transfer.

A slave can insert wait states into any transfer to enable additional time for completion. Each slave has an HREADYOUT signal that it drives during the data phase of a transfer. The interconnect is responsible for combining the HREADYOUT signals from all slaves to generate a single HREADY signal that is used to control the overall progress.

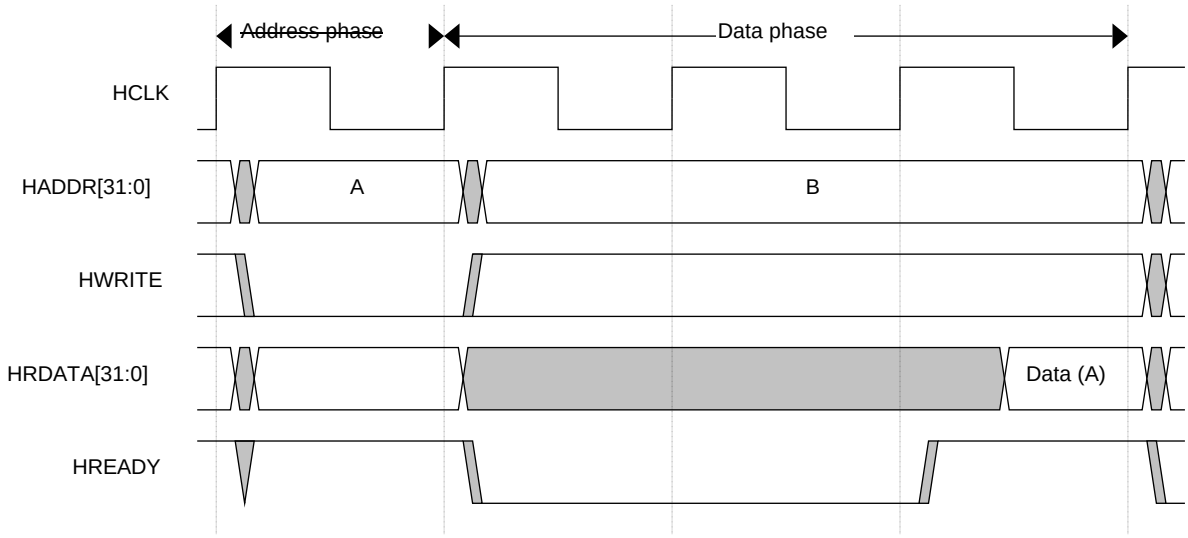

Figure 4.3 – Read Transfer with two wait states

.

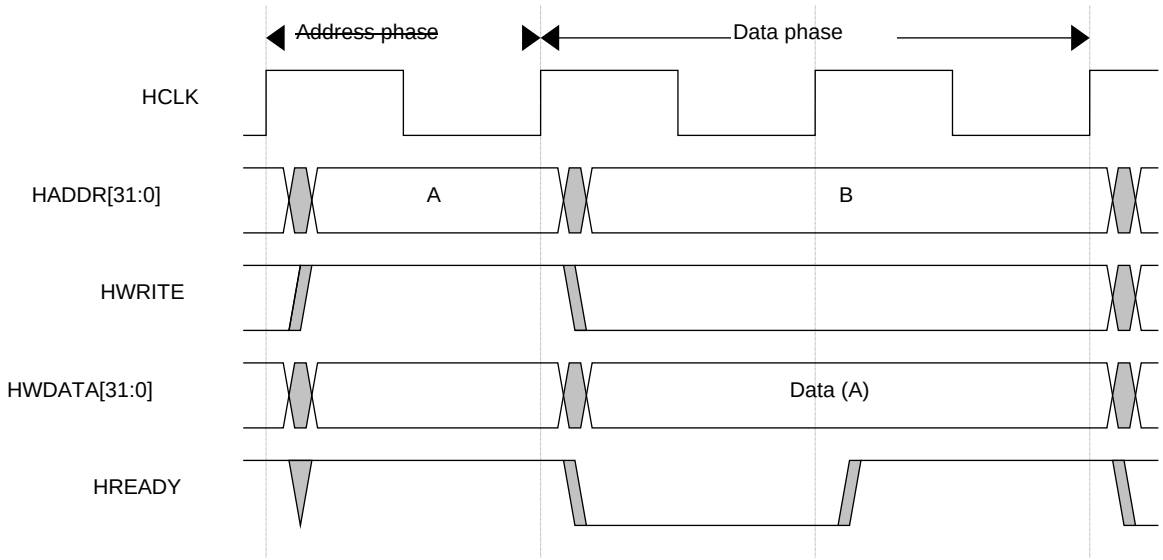

Figure 4.4 – Write state with one wait state

When a transfer is extended in this way it has the side-effect of extending the address phase of the next transfer.

Figure shows three transfers to unrelated addresses, A, B, and C with an extended address phase for address C.

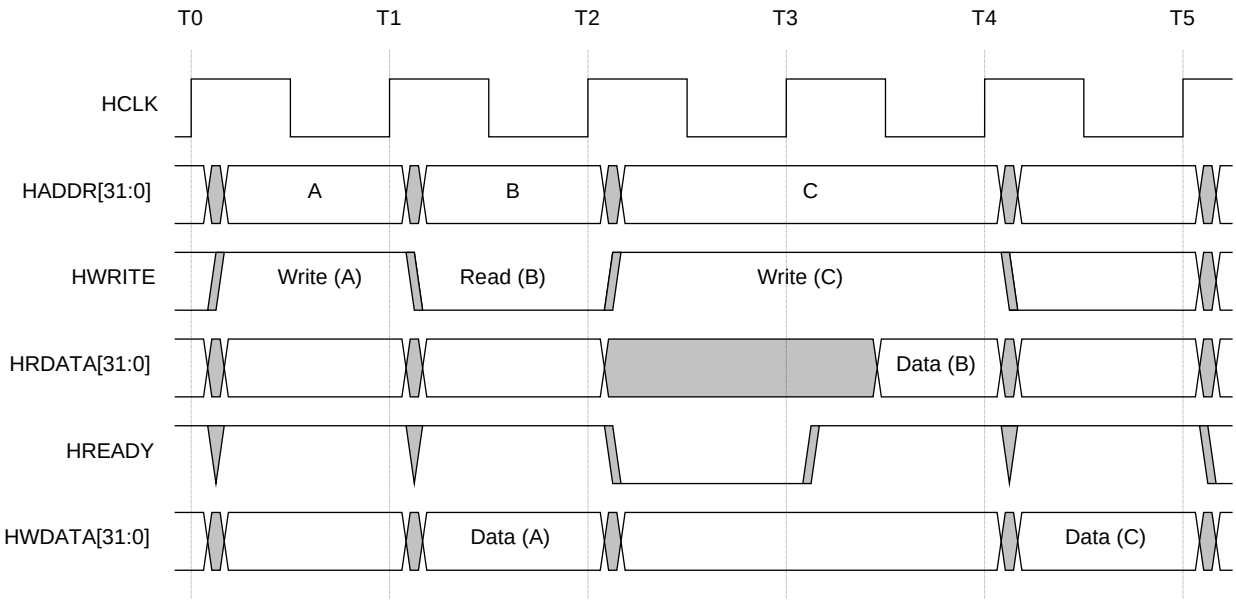

Figure 4.5 – Multiple Transfers

In Figure 4.5:

- the transfers to addresses A and C are zero wait state
- the transfer to address B is one wait state
- extending the data phase of the transfer to address B has the effect of extending the address phase of the transfer to address C.

4.2 Transfer types

HTRANS[1:0]	Type	Description
0b00	IDLE	Indicates that no data transfer is required. A master uses an IDLE transfer when it does not want to perform a data transfer. It is recommended that the master terminates a locked transfer with an IDLE transfer. Slaves must always provide a zero wait state OKAY response to IDLE transfers and the transfer must be ignored by the slave.
0b01	BUSY	The BUSY transfer type enables masters to insert idle cycles in the middle of a burst. This transfer type indicates that the master is continuing with a burst but the next transfer cannot take place immediately. When a master uses the BUSY transfer type the address and control signals must reflect the next transfer in the burst. Only undefined length bursts can have a BUSY transfer as the last cycle of a burst. Slaves must always provide a zero wait state OKAY response to BUSY transfers and the transfer must be ignored by the slave.
0b10	NONSEQ	Indicates a single transfer or the first transfer of a burst. The address and control signals are unrelated to the previous transfer. Single transfers on the bus are treated as bursts of length one and therefore the transfer type is NONSEQUENTIAL.
0b11	SEQ	The remaining transfers in a burst are SEQUENTIAL and the address is related to the previous transfer. The control information is identical to the previous transfer. The address is equal to the address of the previous transfer plus the transfer size, in bytes, with the transfer size being signaled by the **HSIZE[2:0]** signals. In the case of a wrapping burst the address of the transfer wraps at the address boundary.

Table 4.1 – Transfer Types

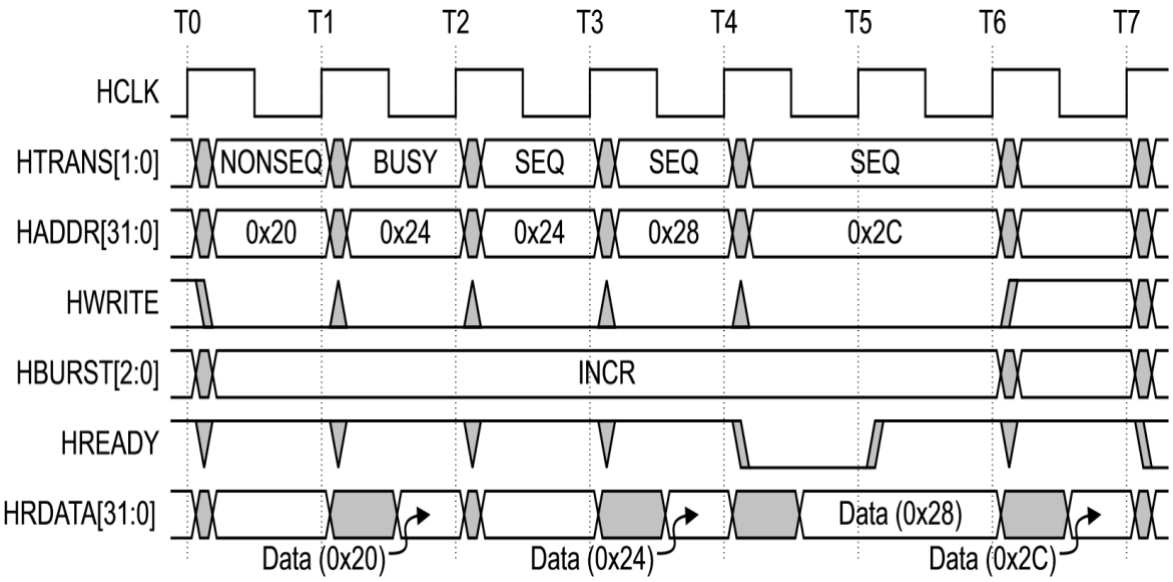

Figure 4.6 – Transfer type examples

In the above figure:

T0-T1 The 4-beat read starts with a NONSEQ transfer.

T1-T2 The master is unable to perform the second beat and inserts a BUSY transfer to delay the start of the second beat. The slave provides the read data for the first beat.

T2-T3 The master is now ready to start the second beat, so a SEQ transfer is signaled. The master ignores any data that the slave provides on the read data bus.

T3-T4 The master performs the third beat. The slave provides the read data for the second beat.

T4-T5 The master performs the last beat. The slave is unable to complete the transfer and uses HREADYOUT to insert a single wait state.

T5-T6 The slave provides the read data for the third beat.

T6-T7 The slave provides the read data for the last beat.

4.3 Locked transfers

If the master requires locked accesses, then it must also assert the HMASTLOCK signal. This signal indicates to any slave that the current transfer sequence is indivisible and must therefore be processed before any other transfers are processed.

Typically, the locked transfer is used to maintain the integrity of a semaphore, by ensuring that the slave does not perform other operations between the read and write phases of a microprocessor SWP instruction.

Figure shows the HMASTLOCK signal with a microprocessor SWP instruction.

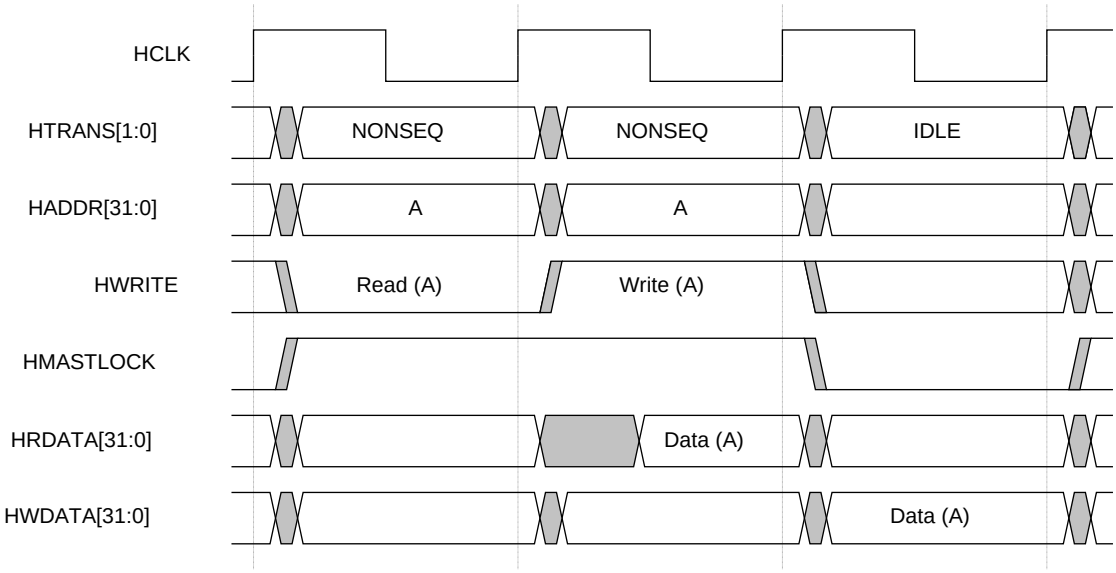

Figure 4.7 – Locked Transfers

Most slaves have no requirement to implement HMASTLOCK because they are only capable of performing transfers in the order they are received. Slaves that can be accessed by more than one master, for example, a Multi-Port Memory Controller (MPMC) must implement the HMASTLOCK signal.

It is permitted for a master to assert HMASTLOCK for IDLE transfers at the beginning, in the middle, or at the end of a sequence of locked transfers. Using locked IDLE transfers at the start or end of a locked transfer sequence is permitted, but not recommended, as this behavior can adversely affect the arbitration of the system.

It is also permitted, but not recommended, for a master to assert HMASTLOCK for a number of IDLE transfers and then deserted HMASTLOCK without performing a non-IDLE transfer. This behavior can adversely affect the arbitration of the system.

It is required that all transfers in a locked sequence are to the same slave address region.

4.4 Transfer size

Table lists the possible transfer sizes

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size (bits)	Description
0	0	0	8	Byte
0	0	1	16	Halfword
0	1	0	32	Word
0	1	1	64	Doubleword
1	0	0	128	4-word line
1	0	1	256	8-word line
1	1	0	512	-
1	1	1	1024	-

Table 4.2 – Transfer size encoding

Use HSIZE in conjunction with HBURST, to determine the address boundary for wrapping bursts.

The HSIZE signals have the same timing as the address bus. However, they must remain constant throughout a burst transfer.

4.5 Burst operation

Bursts of 4, 8, and 16-beats, undefined length bursts, and single transfers are defined in this protocol. It supports incrementing and wrapping bursts:

- Incrementing bursts access sequential locations and the address of each transfer in the burst is an increment of the previous address.

- Wrapping bursts wrap when they cross an address boundary. The address boundary is calculated as the product of the number of beats in a burst and the size of the transfer. The number of beats is controlled by HBURST and the transfer size is controlled by HSIZE.

For example, a four-beat wrapping burst of word (4-byte) accesses wraps at 16-byte boundaries. Therefore, if the start address of the burst is 0x34, then it consists of four transfers to addresses 0x34, 0x38, 0x3C, and 0x30. HBURST[2:0] controls the burst type.

Table lists the possible burst types.

HBURST[2:0]	Type	Description
0b000	SINGLE	Single transfer burst
0b001	INCR	Incrementing burst of undefined length
0b010	WRAP4	4-beat wrapping burst
0b011	INCR4	4-beat incrementing burst
0b100	WRAP8	8-beat wrapping burst
0b101	INCR8	8-beat incrementing burst
0b110	WRAP16	16-beat wrapping burst
0b111	INCR16	16-beat incrementing burst

Table 4.3 Burst signal encoding

Masters must not attempt to start an incrementing burst that crosses a 1KB address boundary.

Masters can perform single transfers using either:

- SINGLE transfer burst.
- Undefined length burst that has a burst of length one.

All transfers in a burst must be aligned to the address boundary equal to the size of the transfer. For example, word transfers must align to word address boundaries (HADDR[1:0] = 0b00), and halfword transfers to halfword address boundaries (HADDR[0] = 0). The address for IDLE transfers must also be aligned, otherwise during simulation it is likely that bus monitors could report spurious warnings.

4.5.1 Burst termination after a BUSY transfer

After a burst has started, the master uses BUSY transfers if it requires more time before continuing with the next transfer in the burst.

During an undefined length burst, INCR, the master might insert BUSY transfers and then decide that no more data transfers are required. Under these circumstances, it is

acceptable for the master to then perform a NONSEQ or IDLE transfer that then effectively terminates the undefined length burst.

The protocol does not permit a master to end a burst with a BUSY transfer for fixed length bursts of type:

- Incrementing INCR4, INCR8, and INCR16.
- Wrapping WRAP4, WRAP8, and WRAP16.

These fixed length burst types must terminate with a SEQ transfer.

The master is not permitted to perform a BUSY transfer immediately after a SINGLE burst. SINGLE bursts must be followed by an IDLE transfer or a NONSEQ transfer.

4.5.2 Early burst termination

Bursts can be terminated by either:

• Slave error response.

• Multi-layer interconnect termination.

Slave error response

If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is also acceptable for the master to continue the remaining transfers in the burst.

If the master cancels the remaining transfers in the burst then it must change HTRANS to indicate IDLE during the two-cycle Error response.

If the master does not complete that burst then there is no requirement for it to rebuild the burst when it next accesses that slave. For example, if a master only completes three beats of an eight-beat burst then it does not have to complete the remaining five transfers when it next accesses that slave.

Multi-layer interconnect termination

Although masters are not permitted to terminate a burst request early, slaves must be designed to work correctly if the burst is not completed.

When a multi-layer interconnect component is used in a multi-master system then it can terminate a burst so that another master can gain access to the slave. The slave must

terminate the burst from the original master and then respond appropriately to the new master if this occurs.

4.5.3 Burst examples

Examples of various bursts are shown in the following sections:

• Four-beat wrapping burst, WRAP4.

• Four-beat incrementing burst, INCR4.

• Eight-beat wrapping burst, WRAP8.

• Eight-beat incrementing burst, INCR8.

• Undefined length bursts, INCR.

Four-beat wrapping burst, WRAP4

Figure shows a write transfer using a four-beat wrapping burst, with a wait state added for the first transfer.

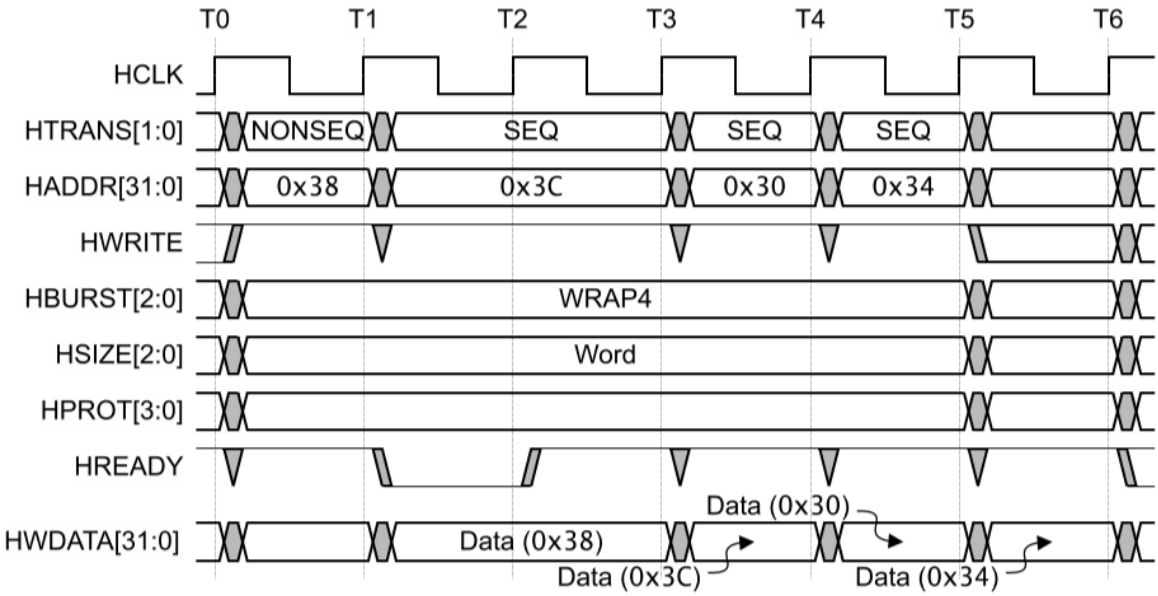

Figure 4.8 – Four beat wrapping burst

Because the burst is a four-beat burst of word transfers, the address wraps at 16-byte boundaries, and the transfer to address 0x3C is followed by a transfer to address 0x30.

Four-beat incrementing burst, INCR4

Figure 3-9 shows a read transfer using a four-beat incrementing burst, with a wait state added for the first transfer. In this case, the address does not wrap at a 16-byte boundary and the address 0x3C is followed by a transfer to address 0x40.

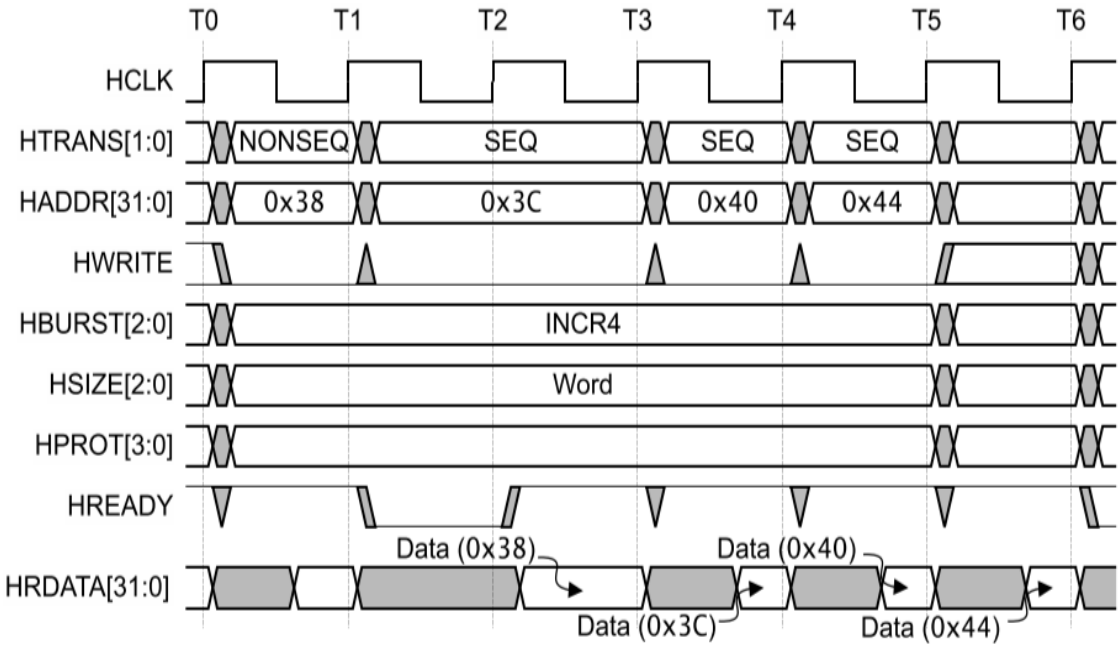

Figure 4.9 – four beat incrementing burst

Eight-beat wrapping burst, WRAP8

Figure shows a read transfer using an eight-beat wrapping burst.

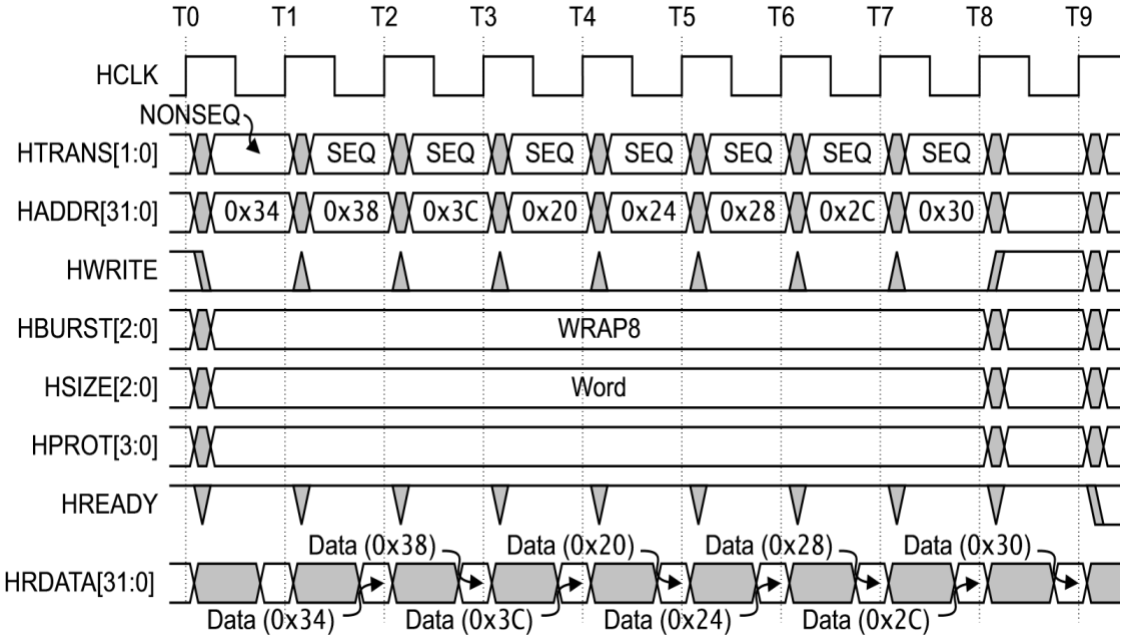

Figure 4.10 – Eight beat wrapping burst

Because the burst is an eight-beat burst of word transfers, the address wraps at 32-byte boundaries, and the transfer to address 0x3C is followed by a transfer to address 0x20.

Eight-beat incrementing burst, INCR8

Figure shows a write transfer using an eight-beat incrementing burst.

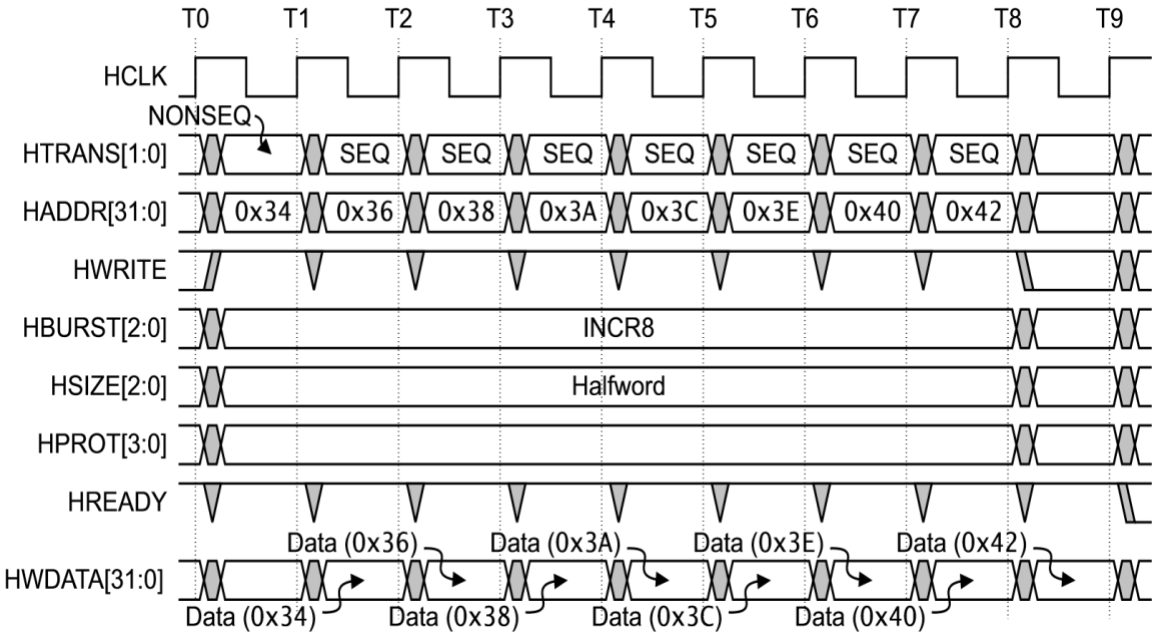

Figure 4.11 – Eight beat incrementing burst

This burst uses halfword transfers, therefore the addresses increase by two. Because the burst is incrementing, the addresses continue to increment beyond the 16-byte address boundary.

Undefined length bursts, INCR

Figure shows incrementing bursts of undefined length.

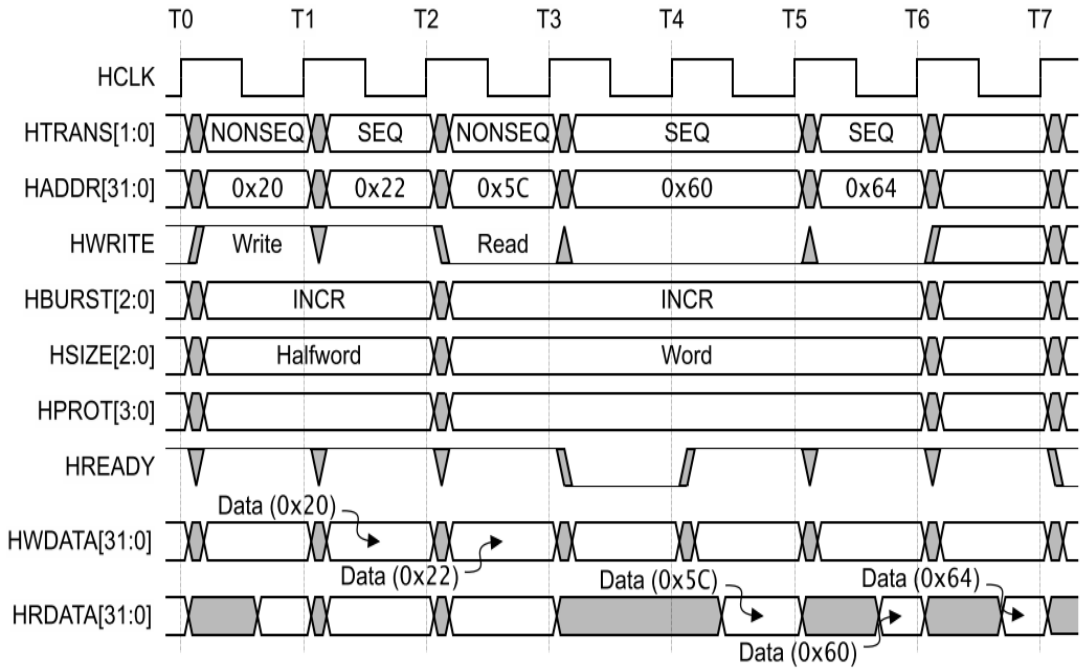

Figure 4.12 – Undefined length bursts

Figure 4-12 shows two bursts:

- The first burst is a write consisting of two halfword transfers starting at address 0x20. These transfer addresses increment by two.

- The second burst is a read consisting of three-word transfers starting at address 0x5C. These transfer addresses increment by four.

4.6 Waited transfers

Slaves use HREADYOUT to insert wait states if they require more time to provide or sample the data. During a waited transfer, the master is restricted to what changes it can make to the transfer type and address. These restrictions are described in the following sections:

• Transfer type changes during wait states.

• Address changes during wait states.

4.6.1 Transfer type changes during wait states

When the slave is requesting wait states, the master must not change the transfer type, except as described in:

• IDLE transfer.

• BUSY transfer, fixed length burst.

• BUSY transfer, undefined length burst.

IDLE transfer

During a waited transfer, the master is permitted to change the transfer type from IDLE to NONSEQ. When the

HTRANS transfer type changes to NONSEQ the master must keep HTRANS constant, until HREADY is HIGH. Figure 3-13 shows a waited transfer for a SINGLE burst, with a transfer type change from IDLE to NONSEQ.

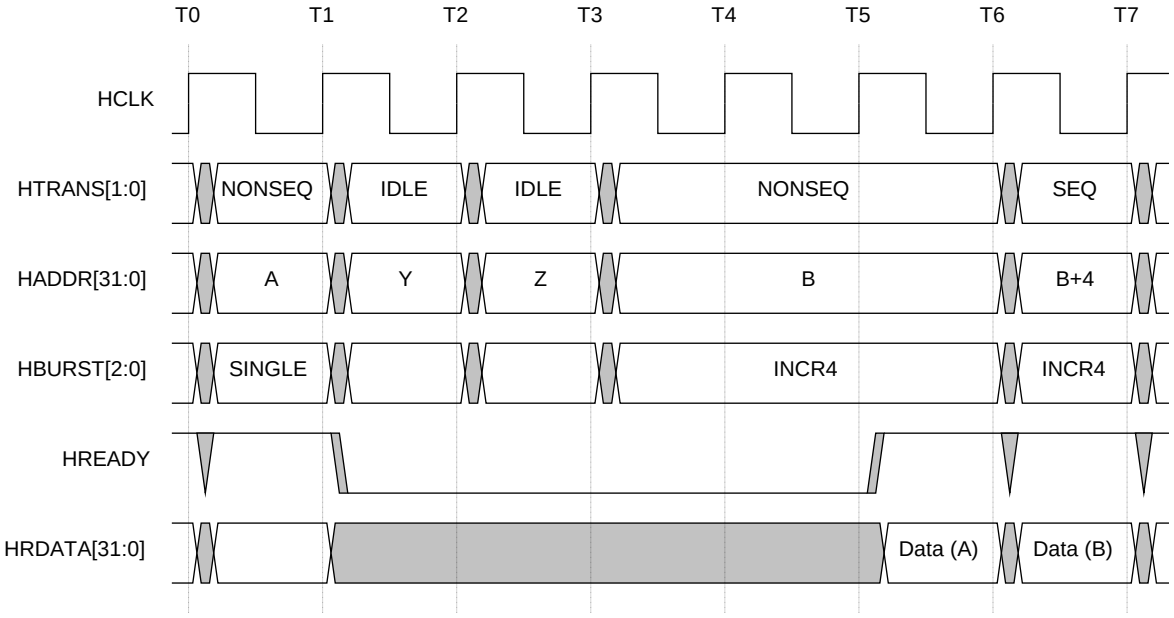

Figure 4.13 – Waited transfer, IDLE to NONSEQ

In Above Figure:

T0-T1 The master initiates a SINGLE burst to address A.

T1-T2 The master inserts one IDLE transfer to address Y. The slave inserts a wait state with HREADYOUT = LOW.

T2-T3 The master inserts one IDLE transfer to address Z.

T3-T4 The master changes the transfer type to NONSEQ and initiates an INCR4 transfer to address B.

T4-T6 With HREADY LOW, the master must keep HTRANS constant.

T5-T6 SINGLE burst to address A completes with HREADY HIGH and the master starts the first beat to address B.

T6-T7 First beat of the INCR4 transfers to address B completes and the master starts the next beat to address B+4.

BUSY transfer, fixed length burst

During a waited transfer for a fixed length burst, the master is permitted to change the transfer type from BUSY to SEQ. When the HTRANS transfer type changes to SEQ the master must keep HTRANS constant, until HREADY is HIGH.

Figure shows a waited transfer in a fixed length burst, with a transfer type change from BUSY to SEQ.

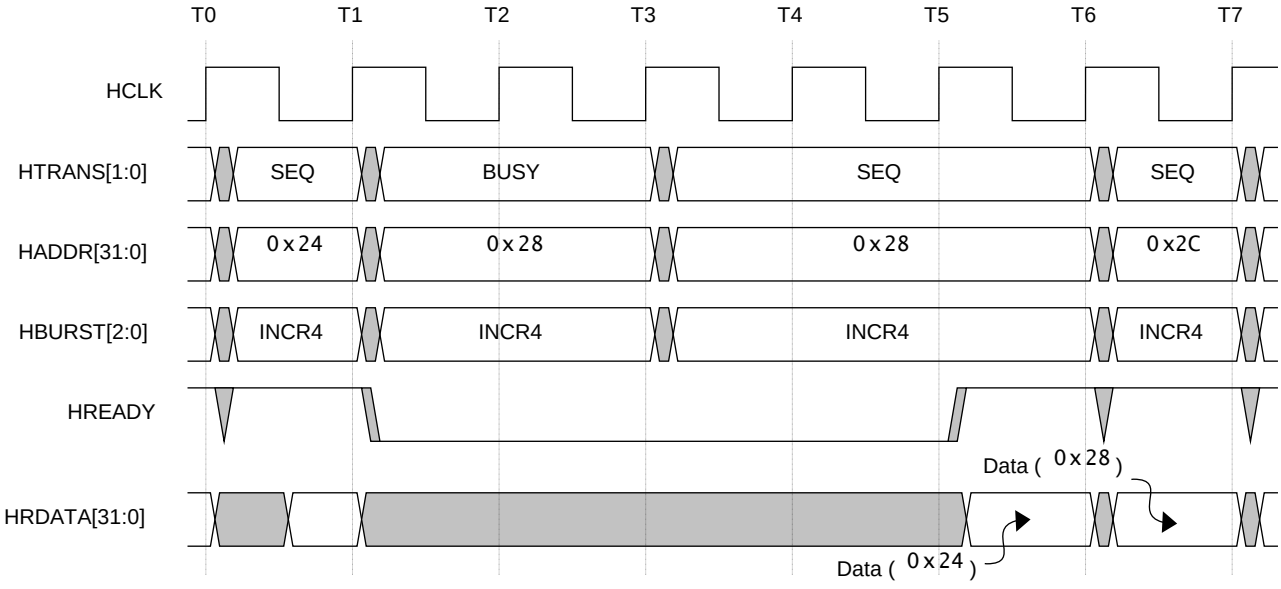

Figure 4.14 – Waited transfer, BUSY to SEQ for a fixed length burst

In Above Figure:

T0-T1 The master initiates the next beat of the INCR4 burst to address 0x24.

T1-T3 The master inserts a BUSY transfer to address 0x28. The slave inserts wait states with HREADYOUT = LOW.

T3-T4 The master changes the transfer type to SEQ and initiates the next beat of the burst to address 0x28.

T4-T6 With HREADY LOW, the master must keep HTRANS constant.

T5-T6 Beat to address 0x24 completes with HREADY HIGH.

T6-T7 Third beat of the INCR4 transfer to address 0x28 completes and the master starts the final beat to address 0x2C.

BUSY transfer, undefined length burst

During a waited transfer for an undefined length burst, INCR, the master is permitted to change from BUSY to any other transfer type, when HREADY is LOW. The burst continues if a SEQ transfer is performed but terminates if an IDLE or NONSEQ transfer is performed.

Figure below shows a waited transfer during an undefined length burst, with a transfer type change from BUSY to NONSEQ.

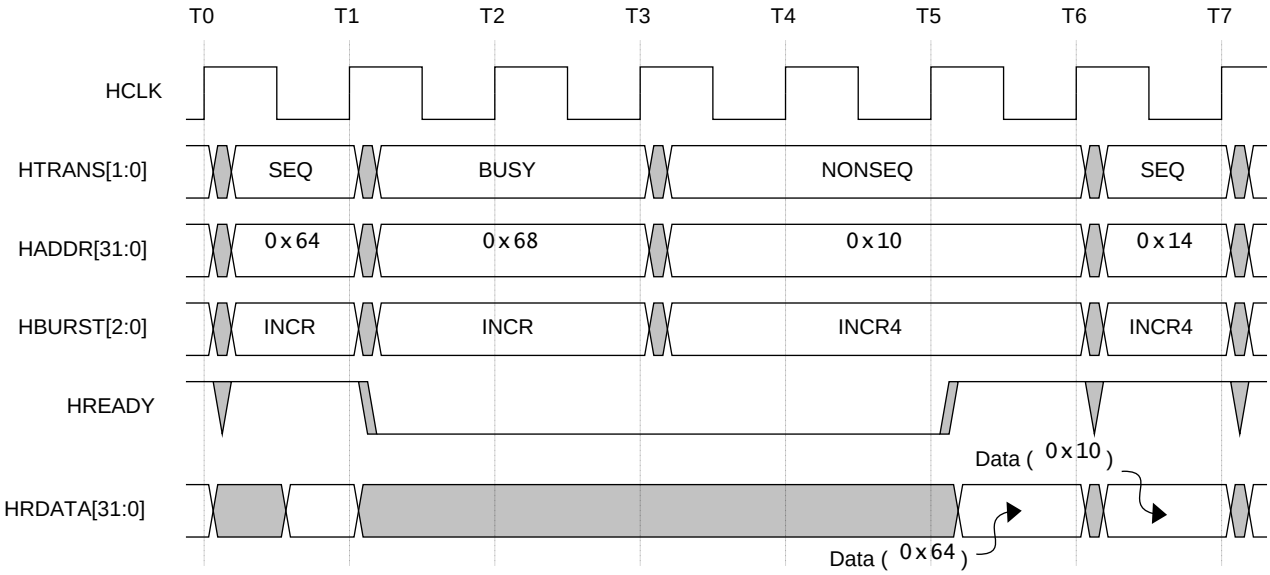

Figure 4.15 – Waited transfer, BUSY to NONSEQ for an undefined length burst

In Figure:

T0-T1 The master initiates the next beat of the INCR burst to address 0x64.

T1-T3 The master inserts a BUSY transfer to address 0x68. The slave inserts wait states with HREADYOUT = LOW.

T3-T4 The master changes the transfer type to NONSEQ and initiates a new burst to address 0x10.

T4-T6 With HREADY LOW, the master must keep HTRANS constant.

T5-T6 Undefined length burst completes with HREADY HIGH and the master starts the first beat to address 0x10.

T6-T7 First beat of the INCR4 transfer to address 0x10 completes and the master starts the next beat to address 0x14.

4.6.2 Address changes during wait states

When the slave is requesting wait states, the master can only change the address once, except as described in:

• During an IDLE transfer.

• After an ERROR response.

During an IDLE transfer

During a waited transfer, the master is permitted to change the address for IDLE transfers. When the HTRANS transfer type changes to NONSEQ the master must keep the address constant, until HREADY is HIGH.

Figure shows a waited transfer for a SINGLE burst, with the address changing during the IDLE transfers.

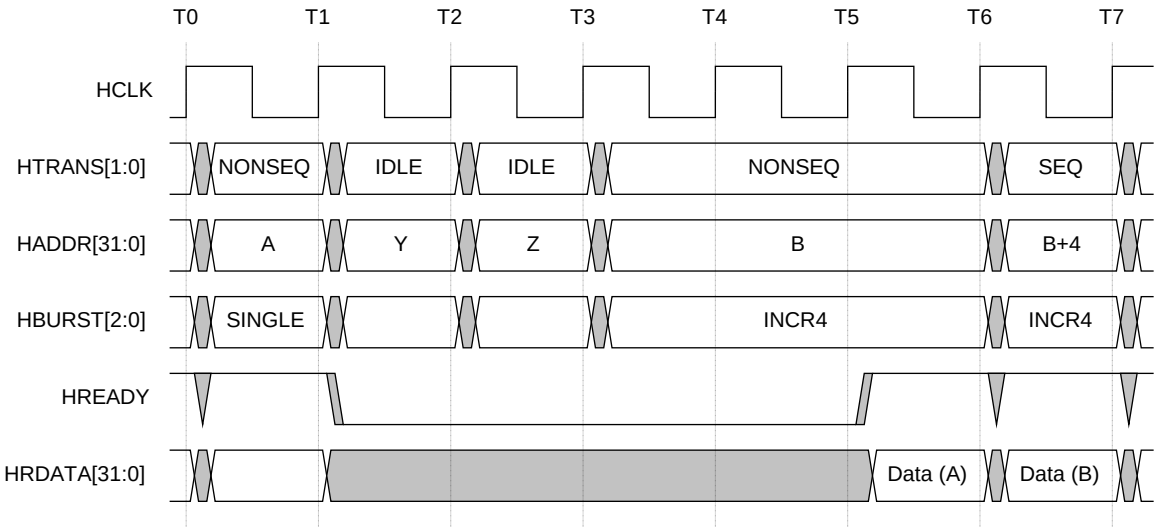

Figure 4.16 – Address changes during a waited transfer, with an IDLE transfer

In Figure 4.16:

T0-T1 The master initiates a SINGLE burst to address A.

T1-T2 The master inserts one IDLE transfer to address Y.
The slave inserts a wait state with **HREADYOUT** = LOW.

T2-T3 The master inserts one IDLE transfer to address Z.

T3-T4 The master changes the transfer type to NONSEQ and initiates an INCR4 transfer to address B. Until **HREADY** goes HIGH, no more address changes are permitted.

T5-T6 SINGLE burst to address A completes with **HREADY** HIGH and the master starts the first beat to address B.

T6-T7 First beat of the INCR4 transfer to address B completes and the master starts the next beat to address B+4.

After an ERROR response

During a waited transfer, if the slave responds with an ERROR response then the master is permitted to change the address when **HREADY** is LOW.

Figure shows a waited transfer, with the address changing following an ERROR response from the slave.

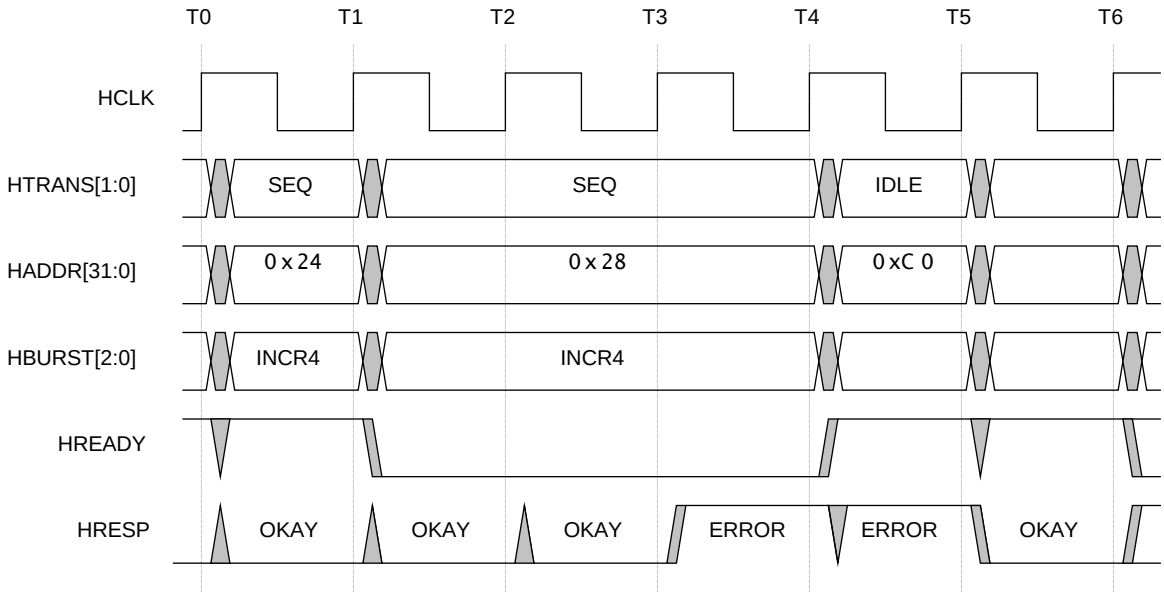

Figure 4.17 – Address changes during a waited transfer, after an ERROR

In Figure 4.17:

T0-T1 The master initiates the next beat of the burst to address 0x24.

T1-T3 The master initiates the next beat of the burst to address 0x28. The slave responds with OKAY.

T3-T4 The slave responds with ERROR.

T4-T5 The master changes the transfer type to IDLE and is permitted to change the address while HREADY is LOW.The slave completes the ERROR response.

T5-T6 The slave at address 0xC0 responds with OKAY.

4.7 Protection control

Issue A of this specification defined a 4-bit HPROT signal, which is described in this section.

The protection control signals, HPROT[3:0], provide additional information about a bus access and are primarily intended for use by any module that implements some level of protection.

The signals indicate if the transfer is:

• An opcode fetch or data access.

• A privileged mode access or user mode access.

For masters with a memory management unit these signals also indicate if the current access is Cacheable or Bufferable. Table below lists the HPROT signal encoding.

HPROT[3] Modifiable	HPROT[2] Bufferable	HPROT[1] Privileged	HPROT[0] Data/Opcode	Description
-	-	-	0	Opcode fetch
-	-	-	1	Data access
-	-	0	-	User access
-	-	1	-	Privileged access
-	0	-	-	Non-bufferable
-	1	-	-	Bufferable
0	-	-	-	Non-cacheable
1	-	-	-	Cacheable

Table 4.4 – Protection signal encoding

4.8 Memory types

AHB5 defines the Extended_Memory_Types property. This property defines whether an interface supports the extended memory types described in this section. If this property is not defined, then the interface does not support the extended memory types.

This issue of the specification adds additional **HPROT** signaling and provides a more detailed list of requirements for each of the memory types.

Table 3-5 shows the meaning of each **HPROT** bit and Table 3-6 on page 3-46 provides a mapping between **HPROT[6:2]** and the memory type.

Bit	Name	Description
HPROT[0]	Data/Inst	When asserted, this bit indicates the transfer is a data access. When deasserted this bit indicates the transfer is an instruction fetch.
HPROT[1]	Privileged	When asserted, this bit indicates the transfer is a privileged access. When deasserted this bit indicates the transfer is an unprivileged access.
HPROT[2]	Bufferable	If both of **HPROT[4:3]** are deasserted then, when this bit is: • Deasserted, the write response must be given from the final destination. • Asserted, the write response can be given from an intermediate point, but the write transfer is required to be made visible at the final destination in a timely manner.
HPROT[3]	Modifiable	When asserted, the characteristics of the transfer can be modified. When deasserted the characteristics of the transfer must not be modified.
HPROT[4]	Lookup	When asserted, the transfer must be looked up in a cache. When deasserted, the transfer does not need to be looked up in a cache and the transfer must propagate to the final destination.
HPROT[5]	Allocate	When asserted, for performance reasons, this specification recommends that this transfer is allocated in the cache. When deasserted, for performance reasons, this specification recommends that this transfer is not allocated in the cache.
HPROT[6]	Shareable	When asserted, indicates that this transfer is to a region of memory that is shared with other masters in the system. A response for the transfer must not be provided until the transfer is visible to other masters. When deasserted, indicates that this transfer is Non-shareable and the region of memory is not shared with other masters in the system. A response for the transfer does not guarantee the transfer is visible to other masters.

Table 4.5 – Meaning of HPORT bits

4.8.1 Data or Instruction

All transfers include the Data or Instruction protection bit **HPROT[0]**:

- When asserted, this bit indicates the transfer is a data access.
- When deasserted this bit indicates the transfer is an instruction fetch.

The protocol defines this indication as a hint. It is not accurate in all cases, for example, where a transaction contains a mix of instruction and data items.

This specification recommends that a master sets **HPROT[0]** HIGH, to indicate a data access unless the access is specifically known to be an instruction access.

4.8.2 Unprivileged or Privileged

All transfers include the Privileged or Unprivileged protection bit, **HPROT[1]**:

- When asserted, this bit indicates the transfer is a Privileged access.
- When deasserted this bit indicates the transfer is an Unprivileged access.

4.8.3 Memory type

This section provides additional information on the **HPROT** Protection Control signals and how these signals relate to different memory types. Table 3-6 shows the mapping between **HPROT[6:2]** signaling and the memory type. Bit combinations that Table 3-6 does not show are not permitted. The Device memory type E suffix indicates that an early write response is permitted. The Device memory type nE suffix indicates that an early write response is not permitted and that the write response must come from the final destination.

HPROT[6] Shareable	HPROT[5] Allocate	HPROT[4] Lookup	HPROT[3] Modifiable	HPROT[2] Bufferable	Memory Type
0	0	0	0	0	Device-nE
0	0	0	0	1	Device-E
0	0	0	1	0	Normal Non-cacheable, Non-shareable
0	0 or 1	1	1	0	Write-through, Non-shareable
0	0 or 1	1	1	1	Write-back, Non-shareable
1	0	0	1	0	Normal Non-cacheable, Shareable
1	0 or 1	1	1	0	Write-through, Shareable
1	0 or 1	1	1	1	Write-back, Shareable

Table 4.6 HPORT[6:2] mapping to memory type

4.8.4 Device memory requirements

For all Device memory, that is Device-nE and Device-E, the required behavior is:

- Read data must be obtained from the final destination.
- Transfers must not be split into multiple transfers or merged with other transfers.

- Reads must not be prefetched or performed speculatively.
- Writes must not be merged.
- All read and write transfers from the same master to the same slave must remain ordered.
- The size of the transfer, as indicated by **HSIZE**, must not be changed.
- A burst of transfers is permitted to be broken into a number of smaller bursts. However, the total number of NONSEQ and SEQ transfers in the original burst must be the same as the total number of NONSEQ and SEQ transfers in the resultant smaller bursts.
- The only change permitted to **HPROT** is to convert a transfer from Bufferabe to Non-bufferable.

Additionally, for Device-nE:

- Write response must be obtained from the final destination.

Additionally, for Device-E:

- The write response can be obtained from an intermediate point.
- Write transfers must be observable to all other masters at the point that a write response is given.
- Write transfers must arrive at the final destination in a timely manner.

4.8.5 Normal memory requirements

For all Normal memory, that is, Normal Non-cacheable memory, Write-through, and Write-back, the required behavior is:

- Reads can be speculative.
- Reads can fetch more data than required.
- Writes can be merged.
- The characteristics of the transfer, as indicated by **HBURST** and **HSIZE** can be changed.
- Read and write transfers from the same master to addresses that overlap must remain ordered.
- For Shareable transactions the response must only be given when the transfer is visible to all other masters.

Additionally, for Normal Non-cacheable memory:

- Write transfers must be made visible at the final destination in a timely manner.

- Read data must be obtained either from:
  —The final destination.
  —A write transfer that is progressing to its final destination.
- If read data is obtained from a write transfer:
  —It must be obtained from the most recent version of the write.
  —The data must not be cached to service a later read.
- Reads must not cache the data obtained for later use.

Additionally, for Write-through:

- The write response can be obtained from an intermediate cache or buffer.
- Read data can be cached in an intermediate cache or buffer.
- A cache lookup is required for read and write transfers.
- Write transactions must be made visible at the final destination in a timely manner.

Additionally, for Write-back:

- The write response can be obtained from an intermediate cache or buffer.
- Read data can be cached in an intermediate cache or buffer.
- A cache lookup is required for read and write transfers.
- Write transactions are not required to be made visible at the destination.

CHAPTER 5

BUS INTERCONNECTION

5.1 Interconnect

An interconnect component provides the connection between masters and slaves in a system.

A single master system only requires the use of a Decoder and Multiplexor, as described in the following sections.

A multi-master system requires the use of an interconnect that provides arbitration and the routing of signals from different masters to the appropriate slaves. This routing is required for address, control, and write data signaling. Further details of the different approaches used for multi-master systems, such as single layer or multi-layer interconnects, are not provided within this specification.

See Multi-layer AHB Technical Overview (ARM DVI 0045) for more information about implementing a multilayer AHB-Lite interconnect.

5.2 Address decoding

An address decoder provides a select signal, HSELx, for each slave on the bus. The select signal is a combinatorial decode of the high-order address signals. Simple address decoding schemes are encouraged to avoid complex decode logic and to ensure high-speed operation.

A slave must only sample the HSELx, address, and control signals when HREADY is HIGH, indicating that the current transfer is completing. Under certain circumstances it is possible that HSELx is asserted when HREADY is LOW, but the selected slave has changed by the time the current transfer completes.

The minimum address space that can be allocated to a single slave is 1KB, and the start and the end of the address region must exist on a 1KB boundary. All masters are designed so that they do not perform incrementing transfers over a 1KB address boundary. This ensures that a burst never crosses an address decode boundary.

Figure shows the HSELx slave select signals generated by the decoder.

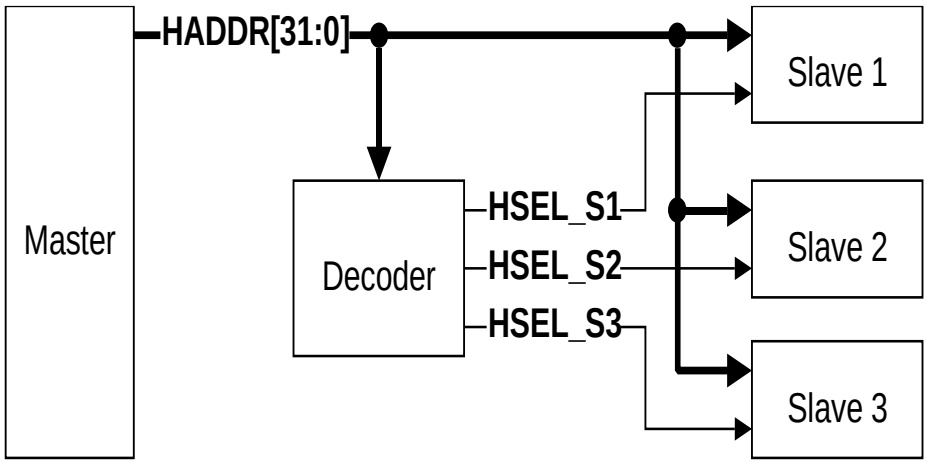

Figure 5.1 – Slave select

5.2.1 Default slave

If a system design does not contain a completely filled memory map then an additional default slave must be implemented to provide a response when any of the nonexistent address locations are accessed.

If a NONSEQUENTIAL or SEQUENTIAL transfer is attempted to a nonexistent address location then the default slave provides an ERROR response.

IDLE or BUSY transfers to nonexistent locations result in a zero wait state OKAY response.

5.2.2 Multiple slave select

A single slave interface is permitted to support multiple slave select, HSELx, signals. Each HSELx signal corresponds to a different decode of the higher order address bits.

This permits a single slave interface to provide multiple logical interfaces, each with a different location in the system address map. The minimum address space that can be allocated to a logical interface is 1KB. This approach removes the need for a slave to support the address decode to differentiate between the logical interfaces.

A typical use case for multiple HSELx signals is a peripheral that has its main data path and control registers at different locations in the address map. Both locations can be accessed through a single interface without the need for the slave to perform an address decode.

5.3 Read data and response multiplexor

The AHB protocol is used with a read data multiplexor interconnection scheme. The master drives out the address and control signals to all the slaves, with the decoder selecting the appropriate slave. Any response data from the selected slave, passes through the read data multiplexor to the master.

Figure shows the multiplexor interconnection structure required to implement a design with three slaves.

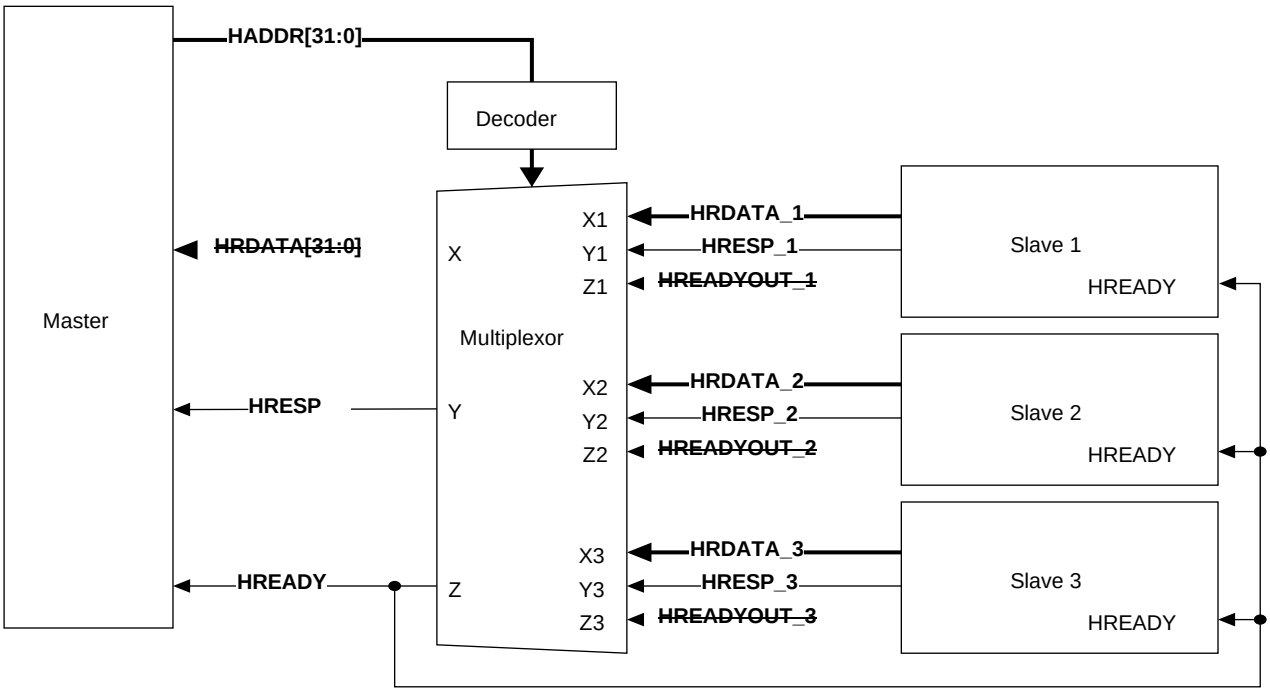

Figure 5.2 – Multiplexor Interconnection

CHAPTER 6

SLAVE RESPONSE SIGNALLING

6.1 Slave transfer responses

After a master has started a transfer, the slave controls how the transfer progresses. A master cannot cancel a transfer after it has commenced.

For components that support the AHB5 Exclusive_Transfers property, see Exclusive access signaling on page 8-72 for details of the additional HEXOKAY transfer response signal.

A slave must provide a response that indicates the status of the transfer when it is accessed. The transfer status is provided by the HRESP signal. Table 5-1 lists the HRESP states.

Table shows that the complete transfer response is a combination of the HRESP and HREADYOUT signals.

HRESP	Response	Description
0	OKAY	The transfer has either completed successfully or additional cycles are required for the slave to complete the request. The **HREADYOUT** signal indicates whether the transfer is pending or complete.
1	ERROR	An error has occurred during the transfer. The error condition must be signaled to the master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition with **HREADYOUT** being asserted in the second cycle.

Table 6.1 – HRESP signal response

The below table lists the complete transfer response based on the status of the HRESP and HREADYOUT signals.

	HREADYOUT	
HRESP	**0**	**1**
0	Transfer pending	Successful transfer completed
1	ERROR response, first cycle	ERROR response, second cycle

Table 6.2 – Combined HRESP and HREADYOUT signal response

This means the slave can complete the transfer in the following three ways:

• Immediately complete the transfer.

• Signal an error to indicate that the transfer has failed.

• Insert one or more wait states to enable time to complete the transfer.

These three slave transfer responses are described in:

• Transfer done.

• Transfer pending.

• ERROR response.

6.1.1 Transfer done

A successful completed transfer is signaled when HREADY is HIGH and HRESP is OKAY.

6.1.2 Transfer pending

A typical slave uses HREADYOUT to insert the appropriate number of wait states into the data phase of the transfer. The transfer then completes with HREADYOUT HIGH and an OKAY response to indicate the successful completion of the transfer.

When a slave inserts a number of wait states prior to completing the response, it must drive HRESP to OKAY.

6.1.3 ERROR response

A slave uses the ERROR response to indicate some form of error condition with the associated transfer. Usually this denotes a protection error such as an attempt to write to a read-only memory location.

Although an OKAY response can be given in a single cycle, the ERROR response requires two cycles. To start the ERROR response, the slave drives HRESP HIGH to indicate ERROR while driving HREADYOUT LOW to extend the transfer for one extra cycle. In the next cycle HREADYOUT is driven HIGH to end the transfer and HRESP remains driven HIGH to indicate ERROR.

The two-cycle response is required because of the pipelined nature of the bus. By the time a slave starts to issue an ERROR response then the address for the following transfer has already been broadcast onto the bus. The two-cycle response provides sufficient time for the master to cancel this next access and drive HTRANS[1:0] to IDLE before the start of the next transfer.

If the slave requires more than two cycles to provide the ERROR response then additional wait states can be inserted at the start of the transfer. During this time HREADY is LOW and the response must be set to OKAY.

Figure shows a transfer with an ERROR response.

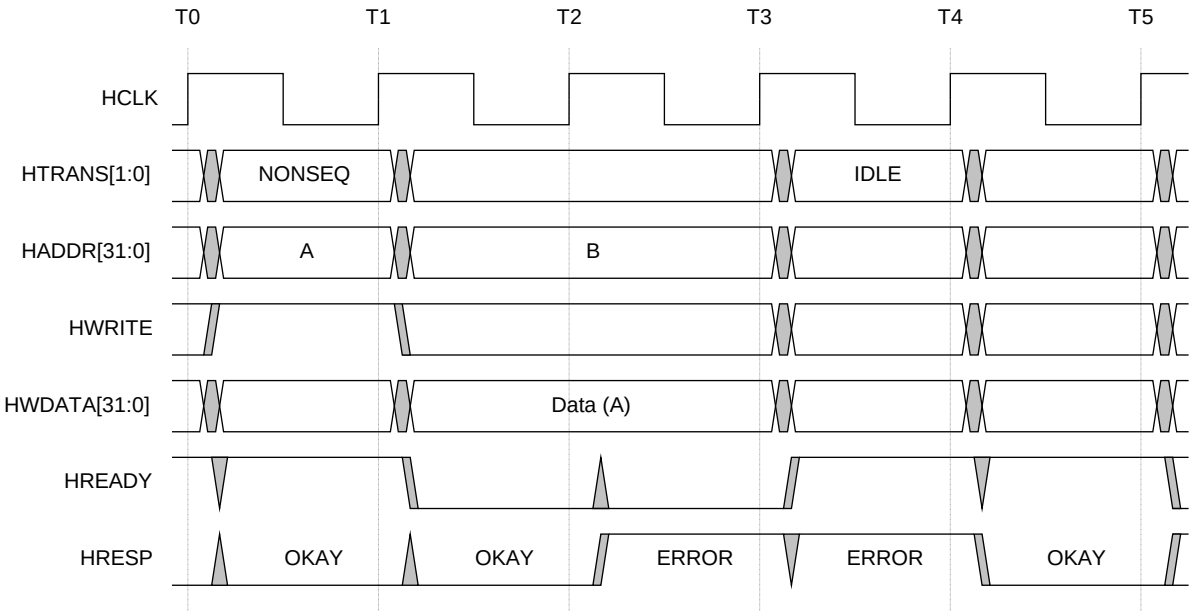

Figure 6.1 – ERROR response

In Figure:

T1-T2 The slave inserts a wait state and provides an OKAY response.

T2-T3 The slave issues an ERROR response. This is the first cycle of the ERROR response because HREADY is LOW.

T3-T4 The slave issues an ERROR response. This is the last cycle of the ERROR response because HREADY is now HIGH.The master changes the transfer type to IDLE. This cancels the intended transaction to address B, that was registered by a slave at time T2.

T4-T5 Slave responds with an OKAY response.

If a slave provides an ERROR response then the master can cancel the remaining transfers in the burst. However, this is not a strict requirement and it is acceptable for the master to continue the remaining transfers in the burst.

CHAPTER 7

DATA BUSES

7.1 Data buses

Separate read and write data buses are required to implement an AHB system. Although the recommended minimum data bus width is specified as 32-bits, this can be changed as described in Data bus width on page 6-65.

The data buses are described in:

• HWDATA.

• HRDATA.

• Endianness.

7.1.1 HWDATA

The master drives the write data bus during write transfers. If the transfer is extended then the master must hold the data valid until the transfer completes, as indicated by HREADY HIGH. See Clock on page 7-68 for details on holding signals valid across multiple cycles.

For transfers that are narrower than the width of the bus, for example a 16-bit transfer on a 32-bit bus, the master only has to drive the appropriate byte lanes. The slave selects the write data from the correct byte lanes. See Endianness on page 6-61 for details of the byte lanes that are active for a little-endian and big-endian system.

7.1.2 HRDATA

The appropriate slave drives the read data bus during read transfers. If the slave extends the read transfer by holding HREADY LOW, then the slave only has to provide valid data in the final cycle of the transfer, as indicated by HREADY HIGH.

For transfers that are narrower than the width of the bus, the slave is only required to provide valid data on the active byte lanes. The master selects the data from the correct byte lanes.

A slave only has to provide valid data when a transfer completes with an OKAY response. ERROR responses do not require valid read data.

7.2 Endianness

AHB supports both big-endian and little-endian systems. Two approaches to big-endian data storage are supported. AHB5 introduces the Endian property to define which form of big-endian data access is supported.

BE8 Byte-invariant big-endian. The term, byte-invariant big-endian, is derived from the fact that a byte access (8-bit) uses the same data bus bits as a little-endian access to the same address.

BE32 Word-invariant big-endian. The term, word-invariant big-endian, is derived from the fact that a word access (32-bit) uses the same data bus bits for the Most Significant (MS) and the Least Significant (LS) bytes as a little-endian access to the same address.

Additional information on byte-invariant big-endian can be found in the section Byte invariance on page 6-63.

The following set of equations defines which data bits are used for little-endian, byte-invariant big-endian, and word- invariant big-endian accesses.

The equations use the following variables: address The address of the transfer.

Data_Bus_Bytes The number of 8-bit data bus byte lanes.

INT(x) Rounded down integer value of x.

7.2.1 Little endian

When a little-endian component accesses a byte, the following equation shows which data bus bits are used:

Byte_Lane = Address –(INT(Address / Data_bus_Bytes)) × Data_Bus_Bytes

Data is transferred on DATA[(8 × Byte_Lane) + 7 : (8 × Byte_Lane)]

When larger little-endian transfers occur, data is transferred such that:

• The LS byte is transferred to the transfer address.

• Increasingly significant bytes are transferred to sequentially incrementing addresses.

7.2.2 Byte-invariant big-endian

When a byte-invariant big-endian component accesses a byte, the following equation shows which data bus bits are used:

Byte_Lane = Address –(INT(Address / Data_bus_Bytes)) × Data_Bus_Bytes

Data is transferred on DATA[(8 × Byte_lane) + 7 : (8 × Byte_lane)]

When larger byte-invariant big-endian transfers occur, data is transferred such that:

• The MS byte is transferred to the transfer address.

• Decreasingly significant bytes are transferred to sequentially incrementing addresses.

7.2.3 Word-invariant big-endian

When a word-invariant big-endian component accesses a byte, the following equation shows which data bus bits are used:

Address_Offset = Address –(INT(Address / Data_Bus_Bytes)) × Data_bus_Bytes

Word_Offset = (INT(Address_Offset / 4)) × 4

Byte_Offset = Address_Offset – Word_Offset

Data is transferred on DATA[(8 × (Word_Offset + 3 – Byte_Offset)) + 7 : 8 × (Word_Offset + 3 – Byte_Offset)]

For a 32-bit bus, the Word_Offset will always be zero and therefore the equation simplifies to: DATA[(8 × (3 – Byte_Offset)) + 7 : 8 × (3 – Byte_Offset)]

For halfword and word transfers using word-invariant big-endian, data is transferred such that:

• The most significant byte is transferred to the transfer address.

• Decreasingly significant bytes are transferred to sequentially incrementing addresses.

For transfers larger than a word using word-invariant big-endian, data is split into word size blocks:

• The least significant word is transferred to the transfer address.

• Increasingly significant words are transferred to incrementing addresses.

The 32-bit data bus in Table 6.1, Table 6.2 and Table 6-3 can be extended for wider data bus implementations.

Burst transfers that have a transfer size less than the width of the data bus have different active byte lanes for each beat of the burst.

The below tables show the byte lanes on a 32-bit bus that are active in a little-endian or byte- invariant big-endian system. The active byte lanes are identical in both cases, but the locations of the most significant and least significant bytes differ.

Transfer size	**Address offset**	**DATA[31:24]**	**DATA[23:16]**	**DATA[15:8]**	**DATA[7:0]**
Word	0	Active[MS]	Active	Active	Active[LS]
Halfword	0	-	-	Active[MS]	Active[LS]
Halfword	2	Active[MS]	Active[LS]	-	-
Byte	0	-	-	-	Active
Byte	1	-	-	Active	-
Byte	2	-	Active	-	-
Byte	3	Active	-	-	-

Table 7.1 - Active byte lanes for a 32-bit little-endian data bus

Transfer size	**Address offset**	**DATA[31:24]**	**DATA[23:16]**	**DATA[15:8]**	**DATA[7:0]**
Word	0	Active[MS]	Active	Active	Active[LS]
Halfword	0	Active[MS]	Active[LS]	-	-
Halfword	2	-	-	Active[MS]	Active[LS]
Byte	0	Active	-	-	-
Byte	1	-	Active	-	-
Byte	2	-	-	Active	-
Byte	3	-	-	-	Active

Table 7.2 – Active byte lanes for a 32-bit byre-invariant big-endian data bus

7.2.4 Byte invariance

The use of byte-invariant big-endian data structures simplifies accessing a mixed-endian data structure in a single memory space.

Using byte-invariant big-endian and little-endian means that, for any multi-byte element in a data structure:

• The element uses the same continuous bytes of memory, regardless of the endianness of the data.

• The endianness determines the order of those bytes in memory, meaning it determines whether the first byte in memory is the MS byte or the LS byte of the element.

• Any byte transfer to a given address passes the eight bits of data on the same data bus wires to the same address location, regardless of the endianness of any data element of which the byte is a part.

7.3 Data bus width

One method to improve bus bandwidth without increasing the frequency of operation is to make the data path of the on-chip bus wider. The increased layers of metal and the use of large on-chip memory blocks such as embedded DRAM are driving factors that encourage the use of wider on-chip buses.

Specifying a fixed width of bus means that, in many cases, the width of the bus is not optimal for the application. Therefore an approach has been adopted that enables flexibility of the width of bus but still ensures that modules are highly portable between designs.

The protocol allows the data bus to be 8, 16, 32, 64, 128, 256, 512, or 1024-bits wide. However, it is recommended that a minimum bus width of 32 bits is used. A maximum bus width of 256 bits is adequate for almost all applications.

For read and write transfers, the receiving module must select the data from the correct byte lane on the bus. Replication of data across all byte lanes is not required.

The following sections describe:

• Implementing a narrow slave on a wide bus.

• Implementing a wide slave on a narrow bus.

• Implementing a master on a wide bus.

7.3.1 Implementing a narrow slave on a wide bus

Figure shows how a slave module that has been originally designed to operate with a 32-bit data bus can be converted to operate on a 64-bit bus. This only requires the addition of external logic, rather than any internal design changes, the technique is therefore applicable to hard macrocells.

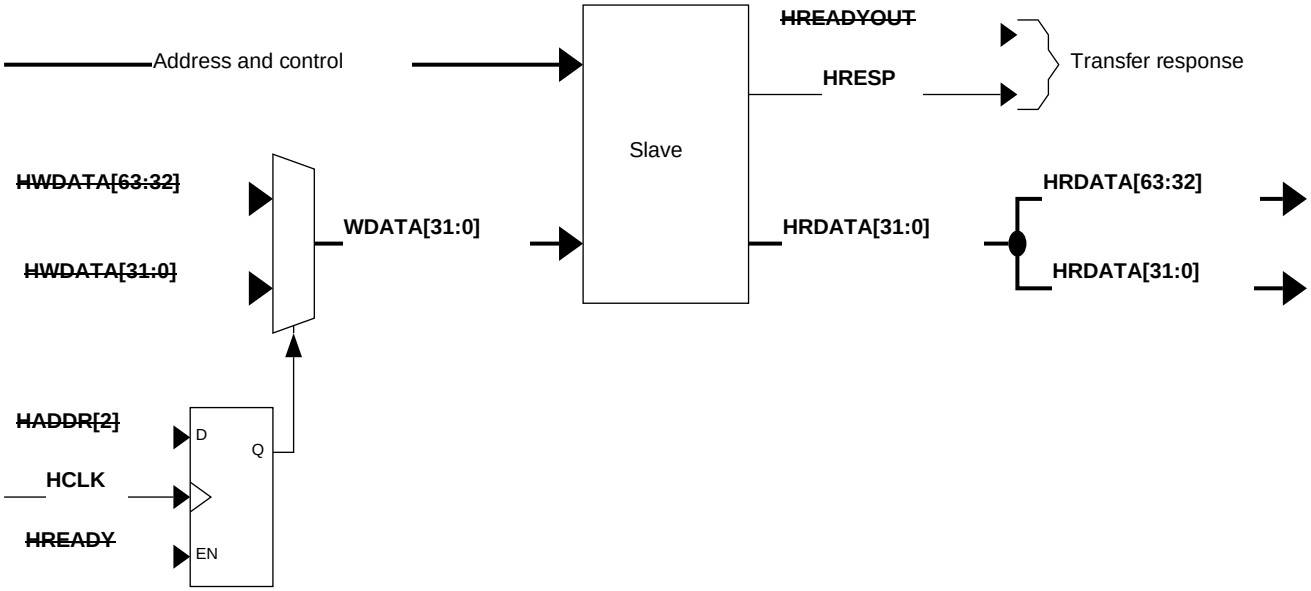

Figure 7.1 – Narrow slave on a wide bus

For the output, when converting a narrow bus to a wider bus, do one of the following:

• Replicate the data on both halves of the wide bus as Figure shows.

• Use additional logic to ensure that only the appropriate half of the bus is changed. This results in a reduction of power consumption.

A slave can only accept transfers that are as wide as its natural interface. If a master attempts a transfer that is wider than the slave can support then the slave can use the ERROR transfer response.

7.3.2 Implementing a wide slave on a narrow bus

Predesigned or imported slaves can be adapted to work with a narrower data bus by using external logic. Figure 7.2 shows a wide slave being implemented on a narrow bus.

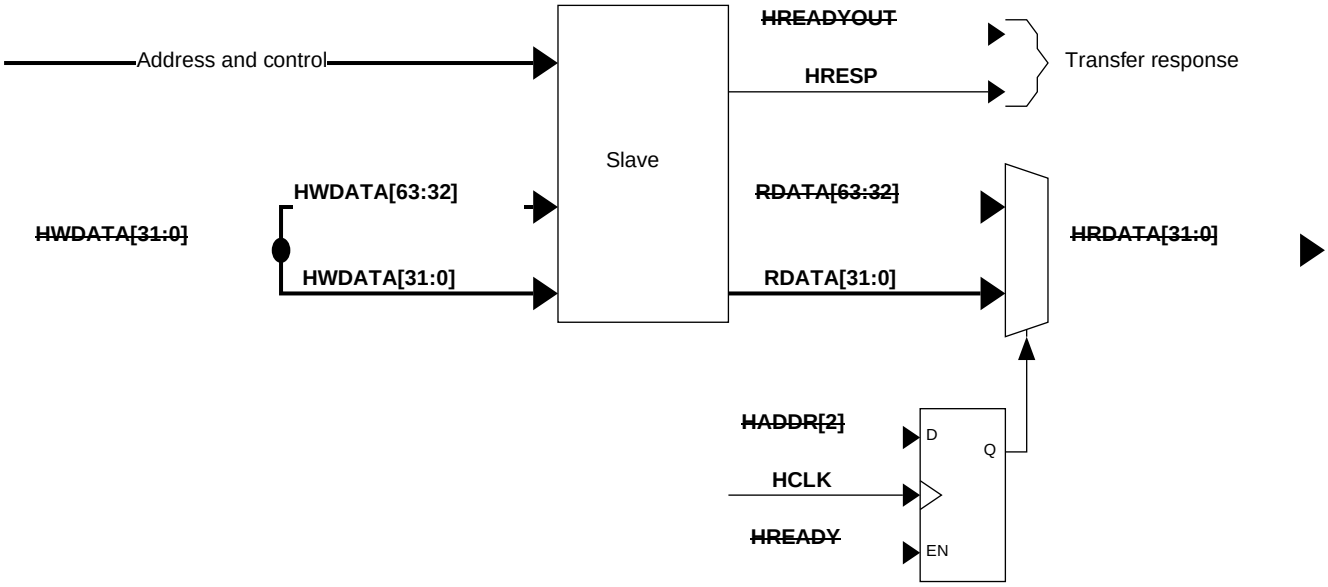

Figure 7.2 – Wide slave on a narrow bus

7.3.3 Implementing a master on a wide bus

Masters can be modified to work on a wider bus than originally intended in the same way that the slave is modified to work on a wider bus. Do this by:

• Multiplexing the input bus.

• Replicating the output bus.

CHAPTER 8

CLOCK AND RESET

8.1 Clock and reset requirements

This section describes the requirements for implementing the HCLK and HRESETn signals.

8.1.1 Clock

Each component uses a single clock signal, HCLK. All input signals are sampled on the rising edge of HCLK. All output signal changes must occur after the rising edge of HCLK.

Signals that are described as being stable are required to remain at the same value when sampled at different rising clock edges in an extended transfer. However, it is possible that these signals can glitch after clock edges, returning to the same value as previously driven.

It is IMPLEMENTATION DEFINED whether an interface is glitch free between rising clock edges.

AHB5 defines the Stable_Between_Clock property. This property is defined to determine if an interface guarantees that signals that are required to be stable remain stable between rising clock edges.

If this property is True, it is guaranteed that signals that are required to be stable remain stable and glitch free between rising clock edges.

If this property is False, or is not defined, signals can glitch between rising clock edges.

8.1.2 Reset

The reset signal, HRESETn, is the only active LOW signal in the protocol and is the primary reset for all bus elements. The reset can be asserted asynchronously, but is deasserted synchronously after the rising edge of HCLK.

A component must define a minimum number of cycles for which the reset signal must be asserted to ensure that the component is fully reset and the outputs are at the required reset values.

During reset all masters must ensure the address and control signals are at valid levels and that HTRANS[1:0] indicates IDLE.

During reset all slaves must ensure that HREADYOUT is HIGH.

CHAPTER 9

SOFTWARE REQUIREMENTS

9.1 About Software

Software: Xilinx Vivado

Version: Vivado 2018.2

Xilinx Vivado is a sophisticated design software suite developed by Xilinx for designing and implementing digital circuits and systems. It provides a comprehensive set of tools and features that enable engineers to create complex designs using programmable logic devices like FPGAs.

Vivado offers a user-friendly graphical interface and supports various design entry methods such as high-level synthesis, schematic capture, and hardware description languages. It includes advanced synthesis and optimization capabilities, automatically transforming high-level behavioral descriptions into optimized gate-level representations.

The suite provides powerful FPGA implementation tools, including physical synthesis, placement, and routing algorithms. These tools optimize the design for specific FPGA devices, maximizing performance and resource utilization. Vivado also supports partial reconfiguration, allowing dynamic updates to specific portions of an FPGA design.

Vivado incorporates extensive debugging and verification features, including simulation and real-time signal analysis. Designers can verify their designs through functional and

timing simulations, ensuring correctness before implementation. The software suite also includes tools like the Vivado Integrated Logic Analyzer and Vivado Serial I/O Analyzer for advanced debugging.

Vivado offers an IP ecosystem with a wide range of pre-designed functional blocks that can be integrated into designs, saving time and effort. It supports collaboration and version control, allowing concurrent work and easy integration with third-party tools.

Xilinx Vivado is a powerful and comprehensive design software suite that streamlines the FPGA design process, accelerating time-to-market and enabling engineers to create innovative digital designs for various applications.

CHAPTER 10

IMPLEMENTATION and RESULTS

10.1 MASTER RTL SCHEMATIC

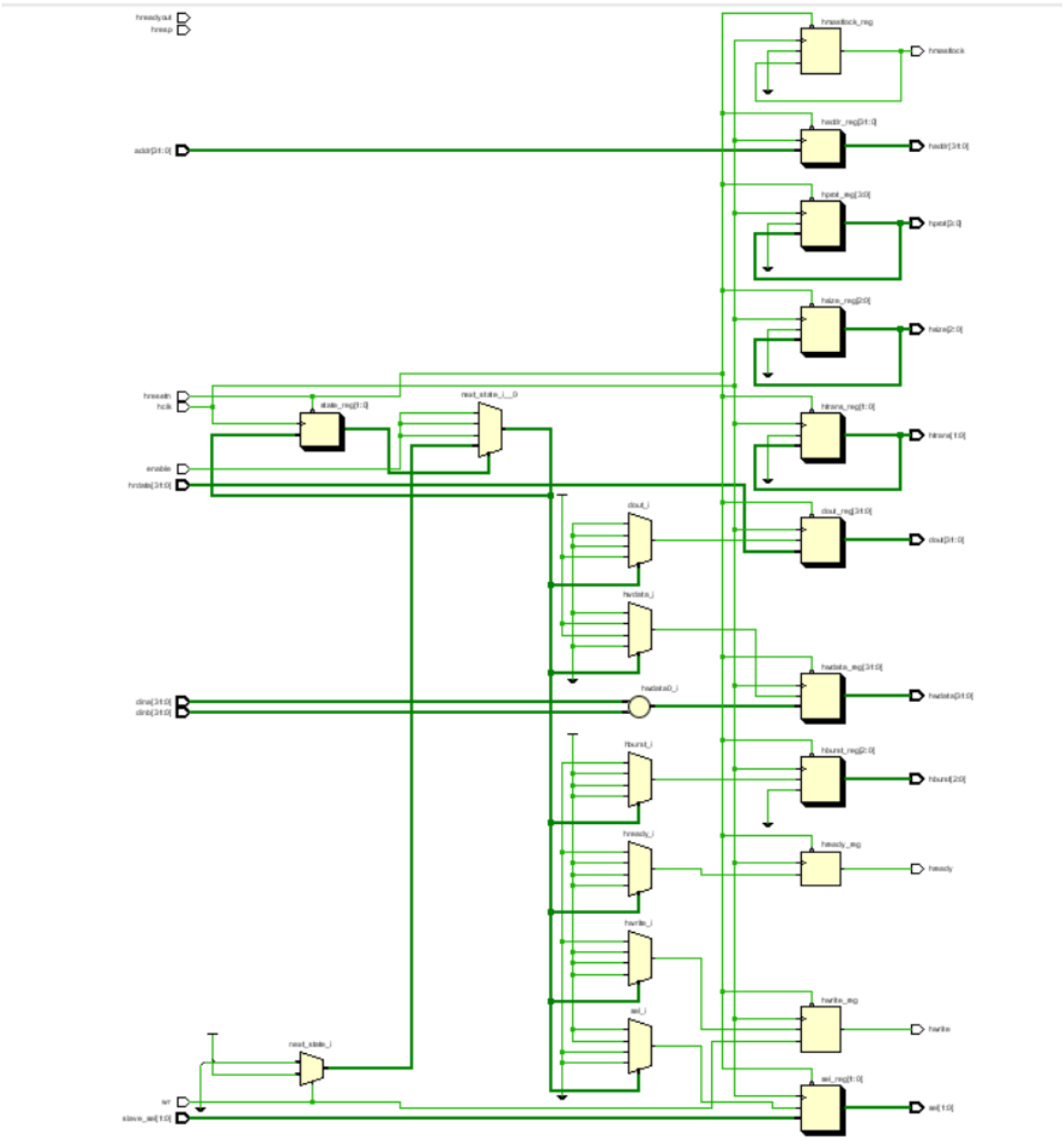

Figure 10.1 – Master RTL schematic

10.2 MASTER SIMULATION

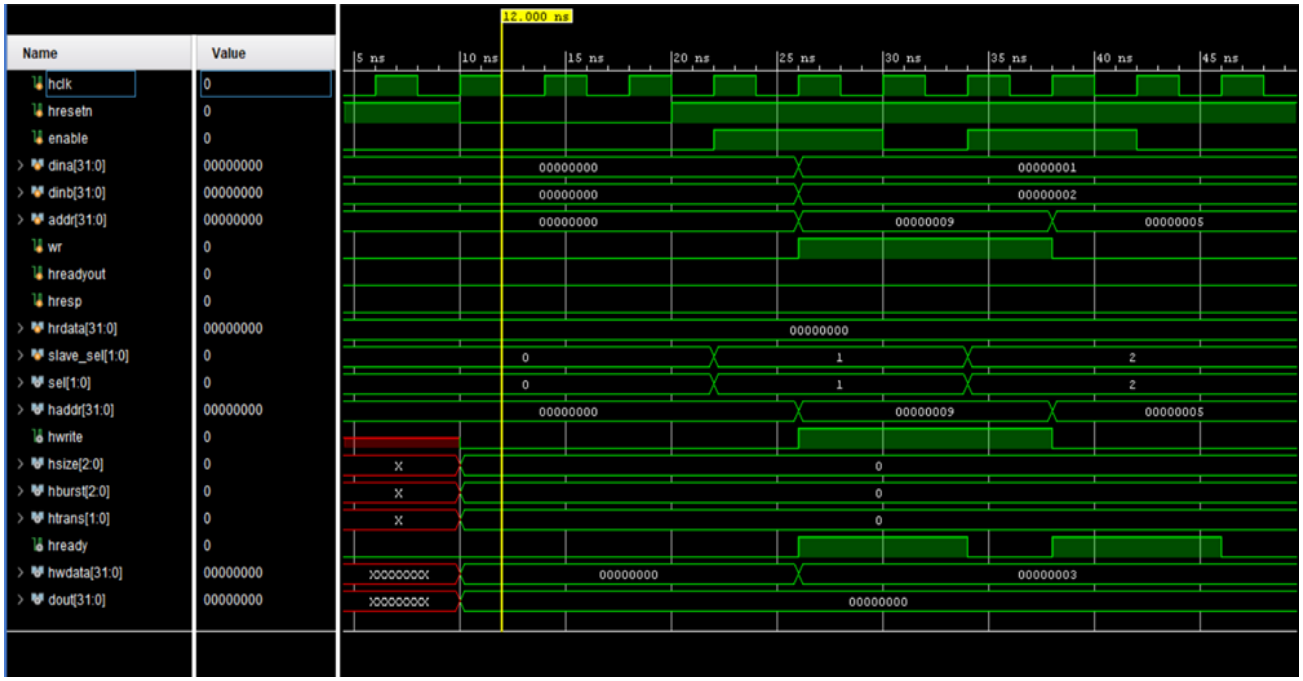

Figure 10.2 Master Simulation

10.3 SLAVE RTL SCHEMATIC

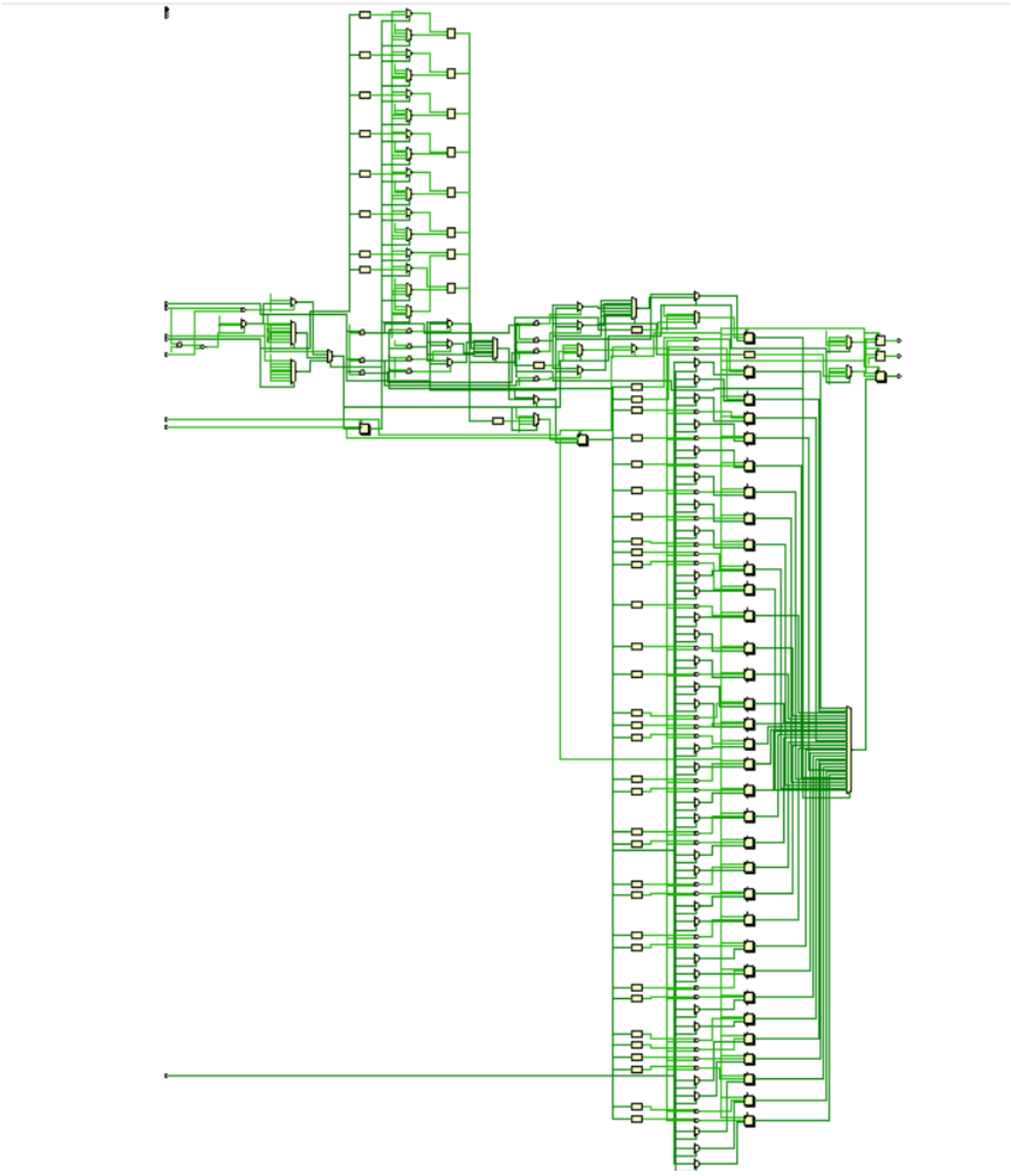

Figure 10.3 – Slave RTL Schematic

10.4 SLAVE SIMULATION

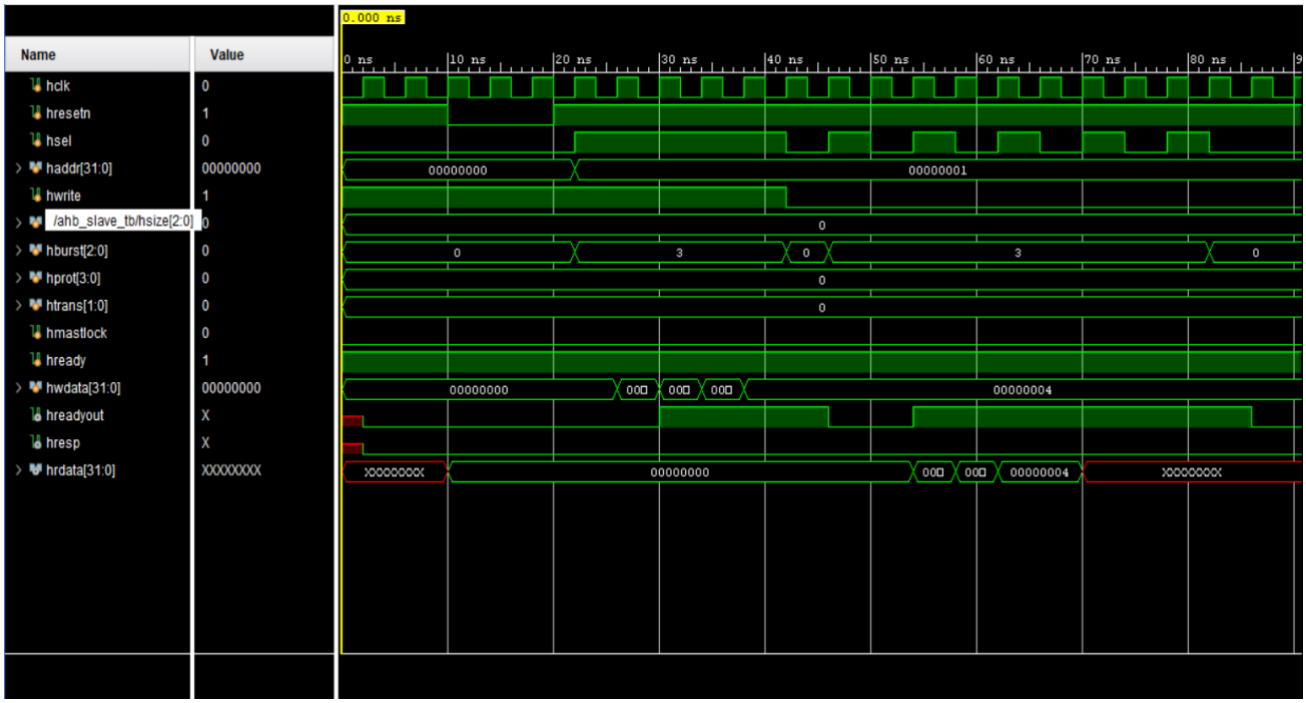

Figure 10.4 – Slave simulation

10.5 DECODER RTL SCHEMATIC

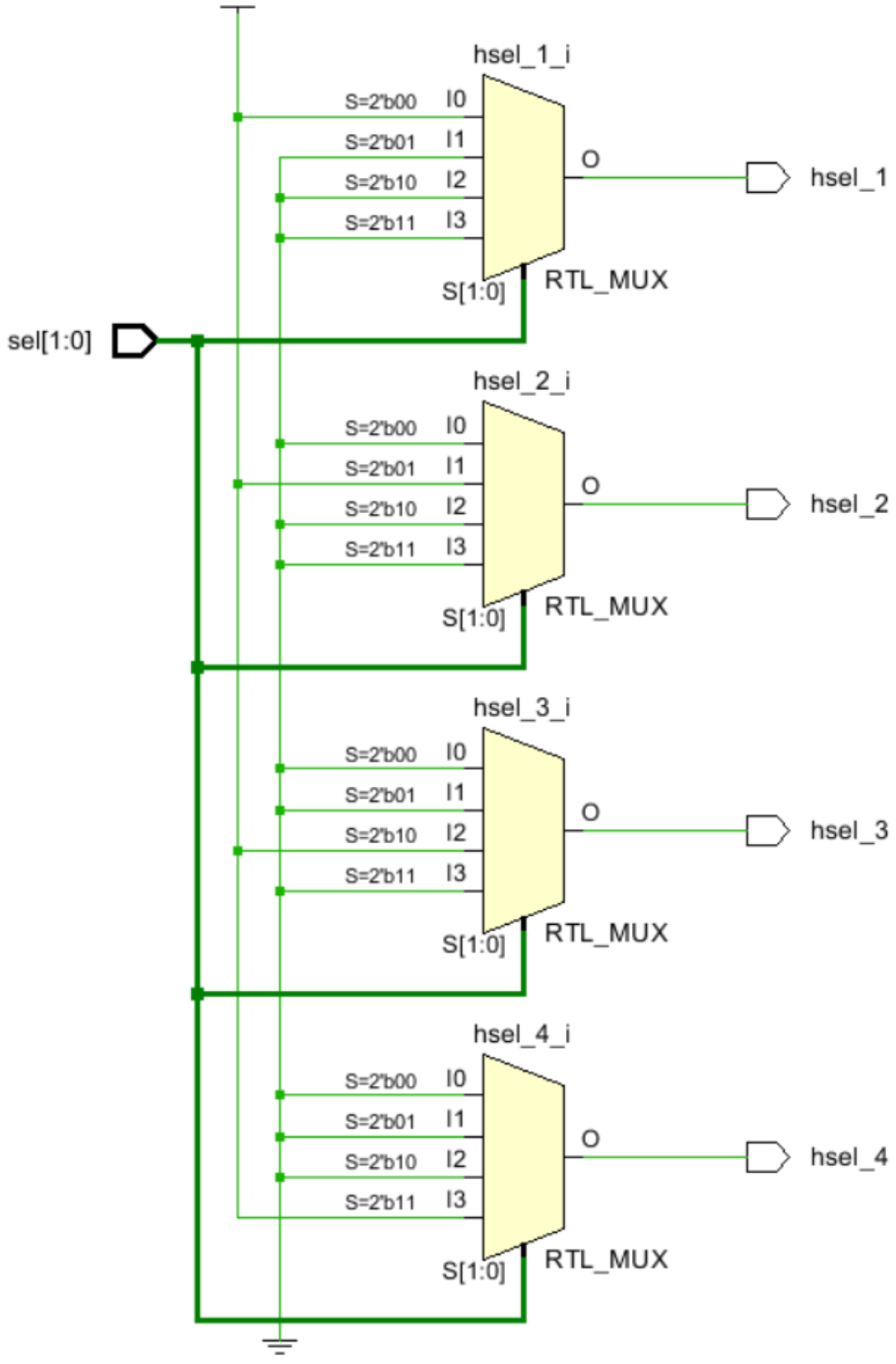

Figure 10.5 Decoder RTL schematic

10.6 DECODER SIMULATION

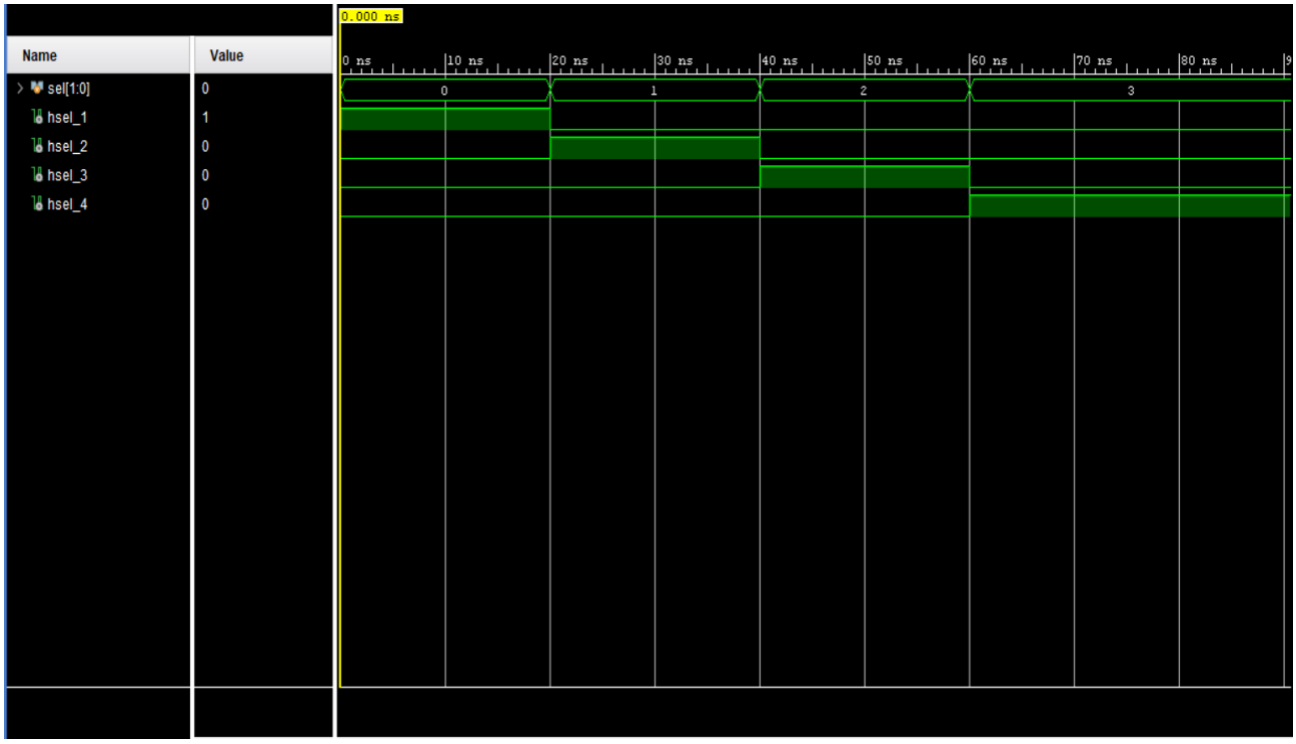

Figure 10.6 Decoder simulation

10.7 MULTIPLEXER RTL SCHEMATIC

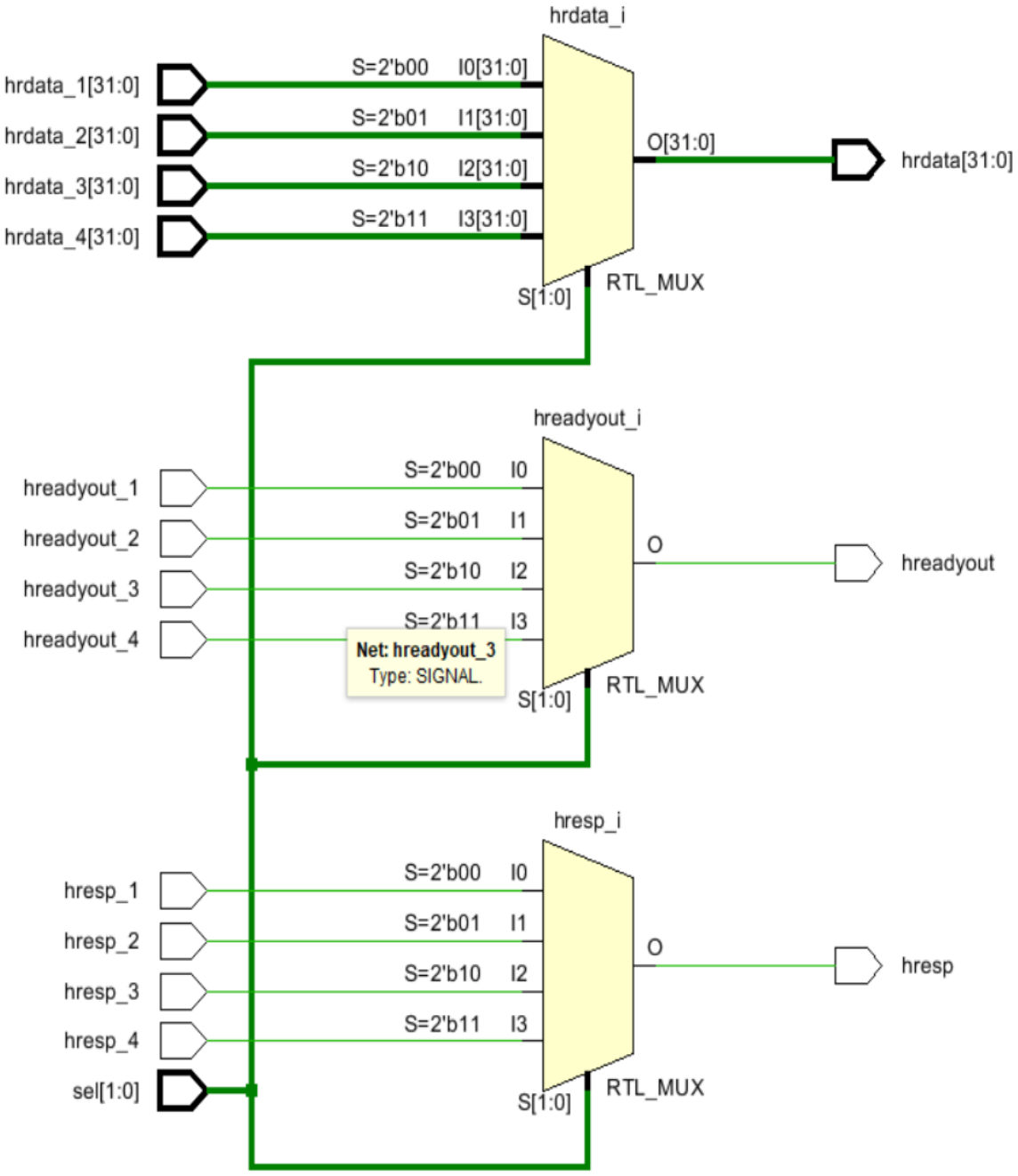

Figure 10.7 - Multiplexor RTL schematic

10.8 MULTIPLEXER SIMULATION

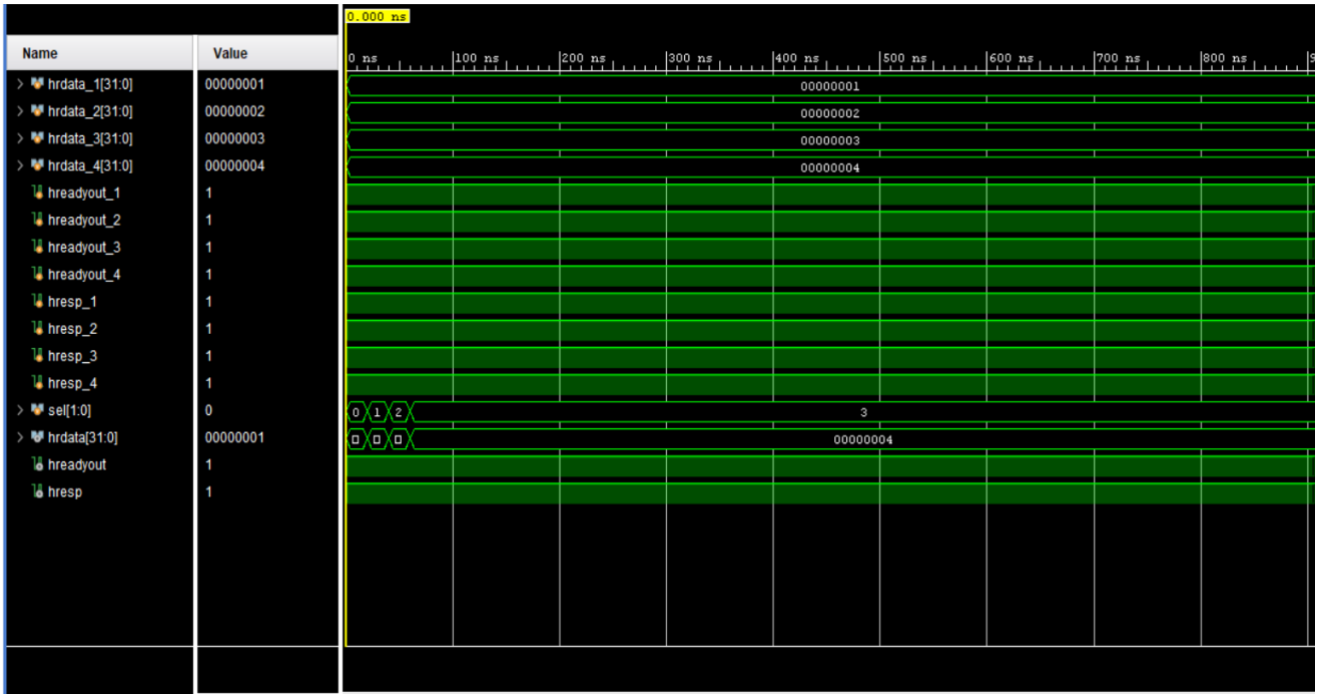

Figure 10.8 Multiplexor simulation

10.9 TOP MODULE RTL SCHEMATIC

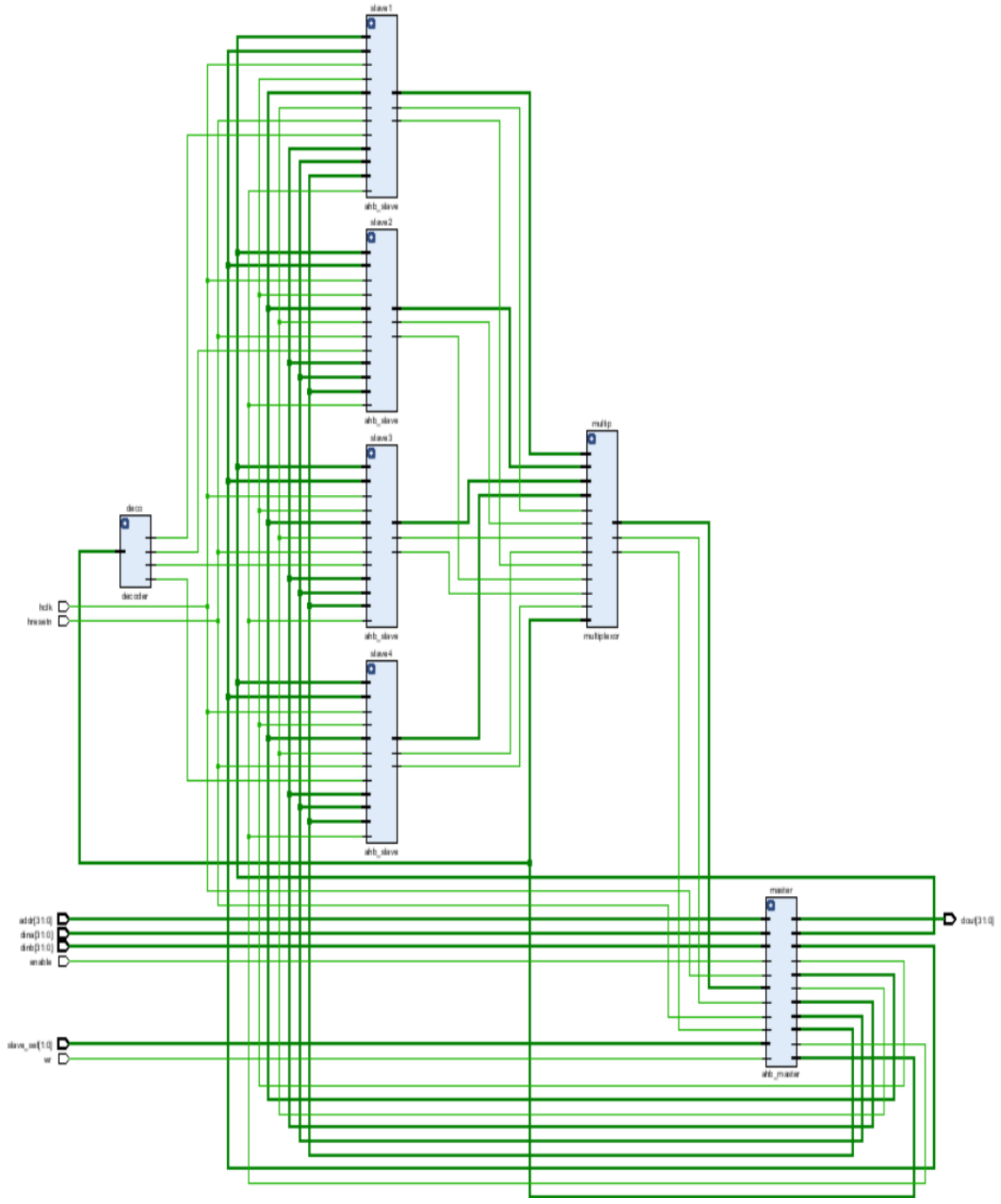

Figure 10.9 – Top module RTL schematic

10.10 TOP MODULE SIMULATION

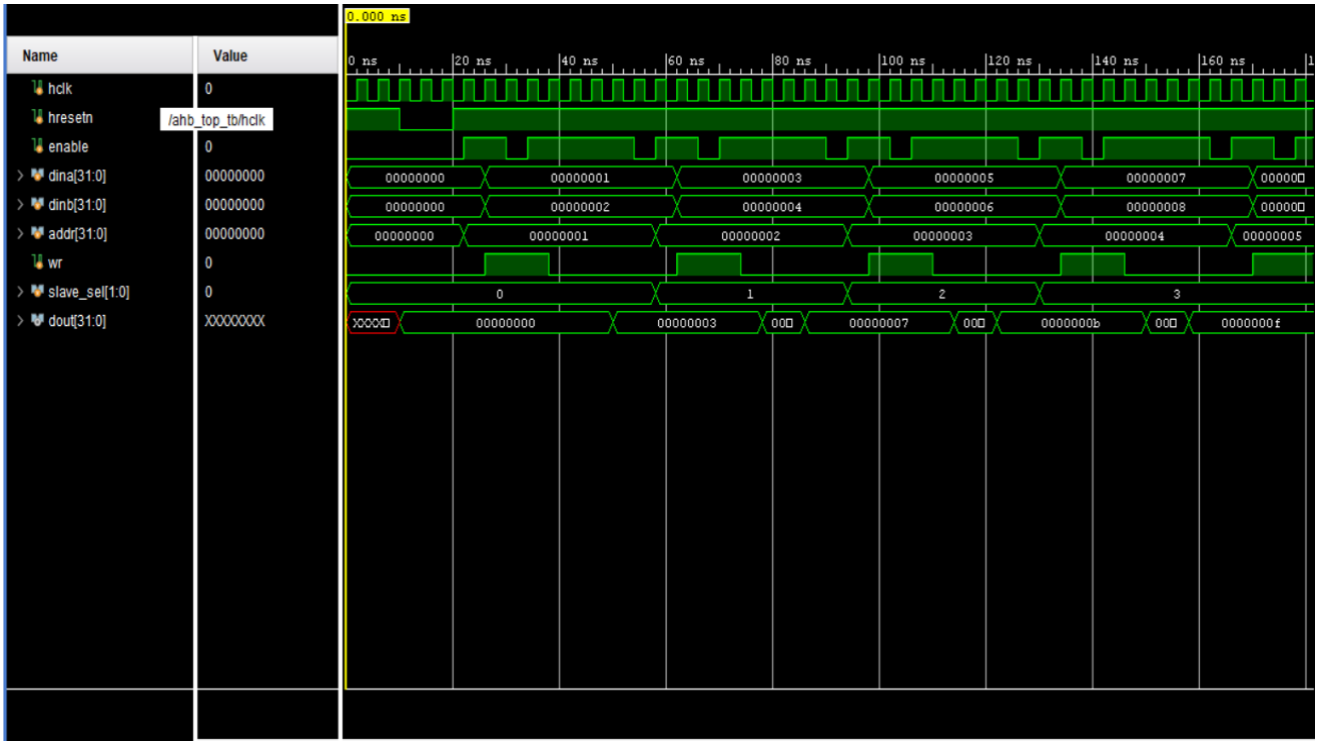

Figure 10.10 – Top module simulation

CHAPTER 11

CONCLUSION and FUTURE SCOPE

AMBA AHB give superior execution, high clock recurrence and equal correspondence with multi-ace transport the executives for information move tasks from the memory connected with the expert or slave fringe gadgets. While making elite execution implanted microcontrollers, AMBA AHB can uphold on-chip interchanges guidelines. AHB can utilize a multi-facet, slave-side refereed, elite execution transport network. The solicitation and award signals in slave-side discretion are not the same as those in ace side mediation since the expert in the previous simply starts a burst exchange and hangs tight for the slave reply prior to happening to the following exchange. Thusly, in the primary case, an exchange or an exchange could be the arbitral unit. In any case, the transport network of ARM just gives cooperative assertion and move based fixed-need mediation plans. SV Statements can be utilized to further develop both the inclusion information that the checkers give and the perceivability inside attestation circuits. The front-end applications can likewise get extra kinds of investigate information, similar to flag conditions.

REFERENCES

1. AMBA -AHB Protocol specification – ARM.
2. Perumalla Giridhar, Dr Priyanka Choudhury “Design and Verification of AMBA AHB”. 2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing and Communication Engineering.
3. Deeksha L, Shivakumar B.R “Effective Design and Implementation of AMBA AHB Bus Protocol using Verilog”.2019 International Conference on Intelligent Sustainable Systems.
4. D. Jayapraveen and T. G. Priya, "Design of memory controller based on AMBA AHB protocol", Elixir Comp. Sci. Engg. 51A, vol. 2, pp. 11115-11119, 2018.
5. Soo-Yun Hwang and Kyoung-Sun Jhang, "An Improved Implementation Method Of AHB Bus Matrix", SOC Conference 2017 Proceedings IEEE International, pp. 211-214.
6. P.Harishankar, Mr.Ajay Sharma , “ Design and Synthesis of Efficient FSM for Master and Slave Interface in AMBA AHB, International journal of Engineering Development and Reasearch ”
7. Shraddha divakar, Archana Tewari “Multichannel AMBA AHB with Multiple Conference on Signal Processing and Communication”
8. Dilip K , Vijaya Prakash A M “Design and Verification of AHB Protocol Using System Verilog and Universal Verification Methodology(UVM).
9. Chrisspear ,System Verilog for Verification New York: Springer , Rath .A.W .Esen .V and Ecker.W, A transaction Oriented UVM-Based Library for verification of analog behaviour .
10. K.Lakshmi Sirisha , Sir C.R .Reddy “Design and Implementation of AMBA AXI to AHB Bridge”