VeriFlow Technologies India (P) Ltd

AHB Master VIP

Version 0.4, September 26, 2008

Ambujavalli

VeriFlow Technologies India (P) Ltd

Rev. #	Designer	Description	Date Released
0.1	Ambujavalli	Initial Draft	September 8, 2008
0.2	Ambujavalli	Added bus level tasks	September 15,2008
0.3	Prabuddha	Review and update	September 18, 2008
0.4	Ambujavalli	Added Sample log messages	September 26, 2008

VeriFlow Technologies India (P) Ltd

Table of Contents

VeriFlow Technologies India (P) Ltd

1 Introduction

This is the user document for the AHB master VIP. The model is AMBA version 2.0-compliant and supports most of the functionality of the standard.

1.1 Intended Audience

The AHB master VIP is targeted for use by

- developers of AHB system
- developers of AHB master and slave

The model can be used as a golden reference against the AHB master being developed or can be used to drive transactions to an AHB slave being developed.

1.2 Intended Usage

The master module should be instantiated and connected on the test-bench. In most SOC configurations, the port connections of the master will have to be hierarchically connected to the AHB bus, it being internal to the SOC.
In a system level application, the model can be instantiated multiple times and hooked up into a AHB bus controller (arbiter/decoder/mux) to setup a complex system level test environment.

1.3 Delivery Overview

The VIP is delivery consists of the following two files. The first is the main module definition and the second is an include file containing constant definitions.

AHBMaster.sv
AHBDefines.svh

2 AHB Master and the AHB system-bus

The AHB master is a verification IP coded as a Verilog behavioral model. It has a set of Verilog tasks that allow user-tests to initiate any AHB bus transactions on AMBA AHB BUS and to configure the master VIP. The diagram below shows the placement of the master with respect to the AHB bus.

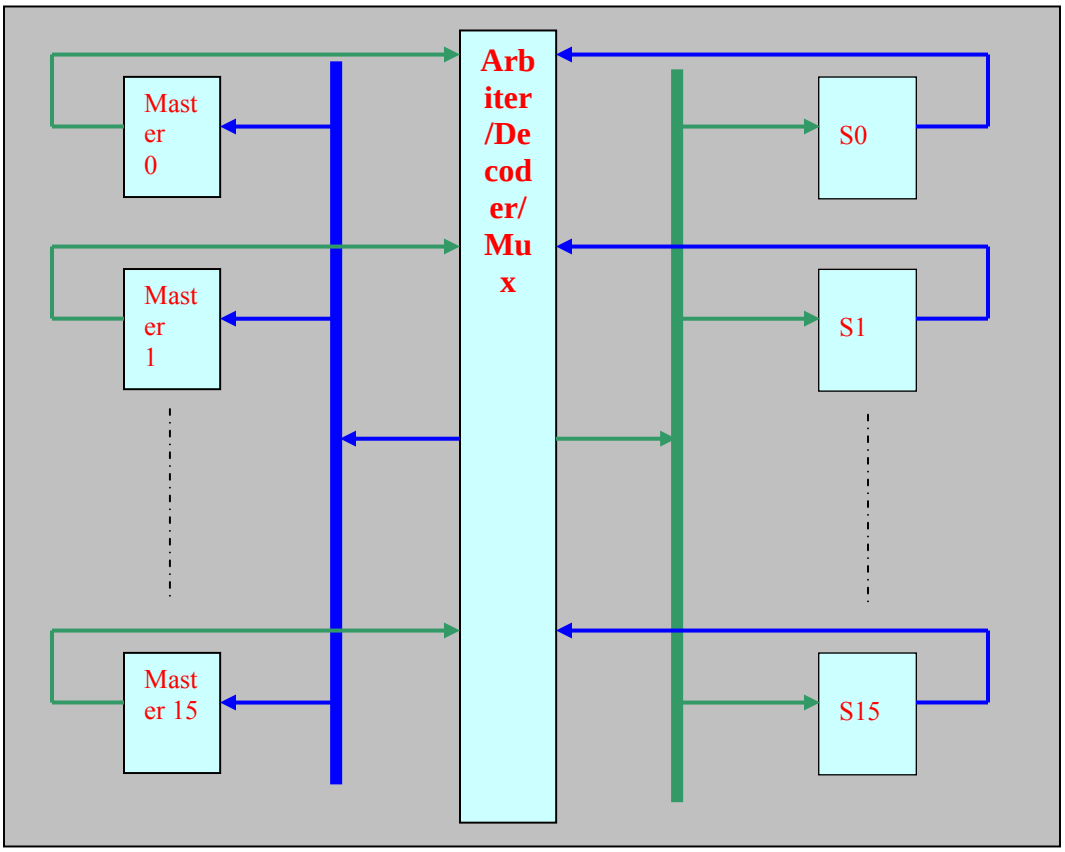

Figure-1 – Placement of AHB Master in the AHB sub-system

.

3 Features supported by the Master

- AMBA version 2.0-compliant Advanced High-performance Bus (AHB) Master.
- Supports 32 bit wide address and data bus
- Provides transaction level interface to drive SINGLE, BURST and WRAP transactions
- Supported all the three data-widths of the 32-bit wide AHB data bus: BYTE, HALFWORD and WORD.
- Supports Locked transfers
- Supports Endianness via configuration task
- 8 deep command FIFO enables back to back transactions
- Supports burst length of 1024 beats (4KBytes maximum). The master automatically breaks the transaction at one Kilo Byte boundaries.
- Supports BUSY insertion for single and burst transfers
- All four types of slave responses (OK, ERROR, RETRY, SPLIT) and slave wait states are handled by the AMBA AHB bus master.
- In the event of RETRY and SPLIT, the bus master automatically re-issues the transaction until all data is transferred.
- In the event of ERROR response, if the master has been configured to continue on error response, bus master completes the remaining beats of the transaction. When unconfigured, the transaction is aborted at the point of error.

4 User- test Interface

4.1 Transaction level command task interface

The VIP provides 8-deep command FIFO to submit commands from the user-test. There is also a seperate 8-deep response FIFO to store the results of the command executed. The VIP executes the commands in the FIFO and the response FIFO is filled with status and data (for read command only) at the end of the transaction.

The VIP accepts only valid commands and issues tag_id[7:0] for each accepted command. User-test should use this tag_id to get the response (see: **get_status**()) or data (see **get_data**()) for that command. Commands may be rejected if the arguments are invalid in which case error code is returned in the tag_id[6:0] with bit-7 set.

After issuing a command the user-test should check for valid tag_id by checking the bit tag_id[7] for zero. If tag_id[7] is set then the command was rejected by the AHB master. The tag_id[6:0] can be decoded to get the type of error. (See Note: **Tag_id**)

When the user-test issues a valid command, the command is pushed into FIFO and the task returns immediately back to the user-test. If command FIFO is full the Master VIP blocks till a transaction is completed and removed from the command FIFO. The response and data for a command executed successfully by the master are available for user access until it needs to be overwritten by the subsequent 8th command. When this happens and the command FIFO has to be overwitten, then the corresponding response FIFO is cleared.

4.2 Bus-level task interface

All bus level tasks except **wait_for_grant** () are nonblocking in nature. In bus level task calls, the master bus signals are driven to the set values immediately and the task returns control to the user-test. Bus signals hold the same value till it is overwritten by other bus-level tasks or transaction level commands. It is the responsibility of the user to introduce delays as neccessary and drive the AHB signals using the bus-level tasks to emulate a complete AHB transaction.

4.3 Configuration task interface

The configuration tasks configures master and the task reurns the control immediately to the user-test. The configured parameters hold the set value till the master is again configured or reset.

5 Design Implementation Details

The AHB master model starts execution of a command from the command-FIFO by first issuing a bus request. When the bus is granted to it by the arbiter, it drives the AHB bus signals according to the command type and the configuration selected.

In case of early burst termination, the AHB master automatically requests for the bus and completes the transaction when the bus is granted subsequently. The next command will be executed only after completion of this command.

5.1 Error handling

The transaction gets terminated only if Master receives an Error response from slave and the Master was not configured to continue on Error response. If it was configured to continue on error, then the AHB master model completes the rest by re-arbitrating for the bus and start with HTRANS_NONSEQ and recomputed HBURST. The error count is incremented for each error encountered during the transaction. The user can retrieve this error count using the response task **get_status** ().

5.2 1-KB address cross over transaction handling

The minimum address space that can be allocated to a single slave is 1-kB. All bus masters are designed such that they will not perform incrementing transfers over a 1kB boundary, thus ensuring that a burst never crosses an address decode boundary. The master splits the transaction when there is a one Kilo Byte boundary cross over and an IDLE is inserted at the boundary. This is neccessary to ensure data phase for the transfer

at the boundary completes without retry or split. Only after this, a new transaction will be issued at the start of the next boundary for the remaining beats.

5.3 Wrap address handling

For WRAP transactions, if an early burst termination happens then depending on the current transfer position within the boundary, the transaction may be split into two transactions.

- one transaction up to the end of boundary and
- another from start of boundary .

5.4 Retry or Split response handling

By the time master receives SPLIT or RETRY response the address for the following transfer would have already been broadcast onto the bus. The bus master issues an IDLE and rebuild the transaction for the address for which RETRY or SPLIT was recieved. This may be done several times until all data are transferred. The master treats both SPLIT and RETRY responses in the same manner. The retry/split count is incremented for each retry/split reponse encountered during the transaction. The user can retrieve these retry count and split count using the response task **get_status** ().

5.5 HBUSREQ and HLOCK generation

By default, for fixed length bursts the master deasserts HBUSREQ upon receiving HGRANT. By enabling the task **config_extended_request** (), request generation can be extended upto the last address phase of the transaction.

Whenever the lock bit in the command is set, the HLOCK is held asserted till the end of the last address phase of the transaction as no other master should be granted the bus until this signal is LOW.

As per AHB protocol recommendation, the user can insert in IDLE after any locked transaction by enabling the task **config_lockend_idle** (). If this task is not enabled, a master issuing continuous locked transactions will hold the bus, till all locked transactions gets executed.

5.6 HBURST generation

Depending on the xfr_count and wrap bit of a transaction level command the HBURST is generated. see **Note: HBURST**

6 List of AHB Master VIP Tasks

This section describes the tasks that the user-test should use to initiate a transaction, to get the status or data for a transaction and to configure the AHB Master.

6.1 Transaction level command tasks

6.1.1 Command to initiate read transaction on AHB bus

```
task read (
input   reg [31:0] address,
input   reg [2:0]   xfr_size,
input   reg [10:0] xfr_count,
input   reg          wrap,
input   reg          lock,
output  reg [7:0]  tag_id);
```

Description: For a valid read command, the parameters of the command are pushed into the command FIFO. When the AHB master is ready to execute this command, it issues a read transaction on the AHB BUS with the given parameters and the read data is stored in the response FIFO. Once the transaction is completed, the user-test can retrieve this data using the response task **get_data** () and the status of the transaction using the response task **get_status** ().

address	Specifies the starting address for the read transaction.
xfr_size	Specifies the data transfer size (**Note: Hsize**)
xfr_count	Specifies the number of transfers for this transaction Range - minimum 1; maximum 1024
wrap	1 - address will wrap around the boundary. 0 - Incrementing address (**Note: Wrap Boundary**)
lock	0 - perform unlocked transaction 1 - Perform locked transaction (**Note: Lock**)
tag_id	Value returned by the master to be used to get the status and get the data for this command. (**Note: Tag_id**)

```
Example : read (32'h1111_2220, `HSIZE_WORD, 4, 0, 0,tag2);
```

32'h1111_2220	the starting address for the read transaction
`HSIZE_WORD	data size is 32 - bit
4	Transfer count is 4.
0	Not a wrap transfer. The master will issue INCR4 burst..
0	Unlocked transaction
tag2	Variable name to store tag-id

6.1.2 Command to initiate write transaction on AHB bus

```
task write (
input   reg [31:0] address,
input   reg [2:0]   xfr_size,
input   reg [10:0] xfr_count,
input   reg          wrap,
input   reg          lock,
input   reg [31:0] array_write_data [`BUF_SIZE-1:0],
output  reg [7:0]  tag_id);
```

Description: For a valid write command, the parameters of the command are pushed into the command FIFO. When the AHB master is ready to execute this command, it issues a write transaction on the AHB BUS with the given parameters. The user-test can get the status of the transaction using the response task **get_status** () to find out when the transaction completes.

Address	Specifies the starting address for the write transaction.
xfr_size	Specifies the data transfer size (**Note: Hsize**)
xfr_count	Specifies the number of transfers for this transaction Range - minimum 1; maximum 1024
Wrap	1 - address will wrap around the boundary. 0 - Incrementing address (**Note: Wrap Boundary**)

Lock	0 - perform unlocked transaction 1 - Perform locked transaction (**Note: Lock**)
tag_id	Value returned by the master to be used to get the status for this command. (**Note: Tag_id**)
array_write_data	System Verilog array pointer. Array is 32 bit wide and a maximum depth of 1K. . To be declared in the TB as: **reg [31:0] array_write_data [1023:0]**;

Example : write(32'h1111_2222, `HSIZE_HALFWORD, 4, 1, 1, array_write_data, tag1);

32'h1111_2222	the starting address for the write transaction.
`HSIZE_HALFWORD	data size is 16-bit
4	Transfer count is 4. The master will issue WRAP4 command
1	address will wrap around the boundary (32'h1111_2222, 32'h1111_2224, 32'h1111_2226, 32'h1111_2220)
1	locked transaction
array_write_data	System Verilog array pointer. Each entry contains half-word data right alignedSee Array initialization below
tag1	Variable name to store tag-id

Array initialization: example
array_write_data[0] = 32'h 0000_abcd;
array_write_data[1] = 32'h 0000_1235;
array_write_data[2] = 32'h 0000_7533;
array_write_data[3] = 32'h 0000_ab87;

Note: For BYTE transfers also the data should be right aligned

6.1.3 NOTES for Command Task

6.1.3.1 Note: Wrap Boundary

Wrap address boundary equal to the data transfer size (in bytes) multiplied by the number of beats in the transfer (4, 8 or 16). For example, a four-beat wrapping burst of word (4-byte) accesses will wrap at 16-byte boundaries. Therefore, if the start address of the

transfer is 0x34, then it consists of four transfers to addresses 0x34, 0x38, 0x3C and 0x30.

6.1.3.2 Note: Lock

This indicates to the arbiter that the master is performing a number of indivisible transfers and the arbiter must not grant any other bus master access to the bus once the first transfer of the locked transfers has commenced.

6.1.3.3 Note: Hsize

2’b10	`HSIZE_WORD	data size is 32-bit
2’b01	`HSIZE_HALFWORD	data size is 16-bit
2’b00	`HSIZE_BYTE	data size is 8-bit

6.1.3.4 Note: HBURST

xfr_count	Wrap	Generated **HBURST**	Type Description
1	Don't care	SINGLE	Single transfer
Other than 0,1,4,8,16 And <1025	Don't care	INCR	Incrementing burst of unspecified length
4	1	WRAP4	4-beat wrapping burst
4	0	INCR4	4-beat incrementing burst
8	1	WRAP8	8-beat wrapping burst
8	0	INCR8	8-beat incrementing burst
16	1	WRAP16	16-beat wrapping burst
16	0	INCR16	16-beat incrementing burst
0 or greater than1024	Don't care		**COMMAND REJECTED** by the master

6.1.3.5 Note: Tag_id

The user-test should check this for valid id or error id. If tag_id is an error id (bit-7 is set), the command was not accepted by the master. If valid id, this should be used to get the status of the transaction and to get the read data from AHB master.

Invalid tag_id is generated for the following conditions by setting the corresponding bit in the tag_id.

- tag_id[0] - ADDRESS ERROR: Misaligned starting address for the given hsize and the master is not configured to allow misaligned addressing.
- tag_id[1] - SIZE ERROR: Size argument is other than BYTE, HALFWORD or WORD
- tag_id[2] - TRANSFER COUNT ERROR: Transfer length is zero or greater than 1024.
- tag_id[3] - WRAP ERROR: Wrap bit set and beat count other than 4, 8 or 16
- tag_id[6:4] – Reserved. Always 0.

6.2 Transaction level response tasks

6.2.1 Response Task to get data from AHB master

task **get_data** (input reg [7:0] tag_id, output reg read_done, output reg [31:0] array_read_data[`BUF_SIZE-1:0], output reg [10:0] read_data_count);
Description: Gets the read data for the read command in response FIFO of the AHB master into the buffer array_read_data and the count of valid data in the array_read_data into the parameter read_data_count .

tag_id	equal to master returned tag_id for the read command
read_done	0 - NOT_DONE -indicates the transaction is not yet complete. 1 - DONE - indicates the transaction has completed and read_data_count , array_read_data are valid. Note: Once the read_done shows DONE, it will continue to show DONE until the corresponding command FIFO is overwritten. Once a new command is overwritten, the response FIFO is cleared and the read_done is changed to NOT_DONE.
array_read_data	System Verilog array pointer. Array is 32 bit wide and a maximum depth of 1K .Valid read data available from index 0 to read_data_count -1. To be declared in the TB as: **reg [31:0] array_read_data [1023:0]**;
read_data_count	Number of actual data entries stored in the array_read_data. This value should normally match the xfr_count used for the read() command. However, in case of ERROR response when transaction is aborted, the actual data count may be less.

Example : **get_data (tag2, read_done, array_read_data, read_data_count)**; when used with command : read(32'h1111_2222, `HSIZE_HALFWORD, 4, 1, 1, tag2);

tag2	Equal to tag_id returned by the master for the read command
read_done	1
array_read_data	System Verilog array pointer. See Note: Received Array Data
read_data_count	4

Note: Received Array Data
array_read_data[0] has data for address 32'h1111_2222
array_read_data[1] has data for address 32'h1111_2224
array_read_data[2] has data for address 32'h1111_2226
array_read_data[3] has data for address 32'h1111_2220

6.2.2 Response Task to get status from AHB master

task **get_status** (

input reg [7:0] tag_id,

output reg status_done,

output reg [15:0] error_count,

output reg [15:0] retry_count,

output reg [15:0] split_count);

Description: Gets the status of a command which has been pushed into the command FIFO. This task returns immediately wihtout advancing time.

tag_id	equal to master returned tag_id for the command for which the status is to be known.
status_done	0 - NOT_DONE - indicates the transaction is not yet complete. 1 - DONE - indicates the transaction has completed and and overall_hresp, error_count, retry_count, split_count are valid Note: Once the status shows DONE, it will continue to show DONE until the corresponding command FIFO is overwritten. Once a new command is overwritten, the response FIFO is cleared and the status is changed to NOT_DONE.

error_count	Count of error responses encountered for this command.
retry_count	Count of retry responses encountered for this command.
split_count	Count of split responses encountered for this command.

Example : **get_status(tag3, status_done, error_count, retry_count, split_count)**
when used with command : read(32'h1111_2222, `HSIZE_HALFWORD, 4, 1, 1, tag3);

tag2	Equal to tag_id returned by the master for the read command
status_done	1
error_count	0
retry_count	3
split_count	1

6.3 Bus level tasks

6.3.1 Task to drive hbusreq and hlock

task **drive_req_lock**(
input reg hbusreq,
input reg hlock);

Description: Drives hbusreq and hlock with the given value and returns immediately.

6.3.2 Task to wait for grant

task **wait_for_grant**( );

Description: Task blocks till hgrant is asserted.

6.3.3 Task to drive htrans idle

task **drive_idle** ( );

Description: Drives HTRANS_IDLE on HTRANS and returns immediately.

6.3.4 Task to drive htrans busy

task **drive_busy** (
input reg [31:0] haddr,
input reg hwrite,
input reg [2:0] hsize,
input reg [2:0] hburst);

Description: Drives HTRANS_BUSY on HTRANS and haddr, hwrite, hsize, hburst with the given values and returns immediately.

6.3.5 Task to drive htrans nonseq

task **drive_nonseq** (
input reg [31:0] haddr,
input reg hwrite,
input reg [2:0] hsize,
input reg [2:0] hburst);

Description: Drives HTRANS_NONSEQ and haddr, hwrite, hsize, hburst with the given values and returns immediately.

6.3.6 Task to drive htrans seq

task **drive_seq** (input reg [31:0] haddr, input reg hwrite, input reg [2:0] hsize, input reg [2:0] hburst);
Description: Drives HTRANS_SEQ and haddr, hwrite, hsize, hburst with the given values and returns immediately.

6.3.7 Task to drive hwdata

task **drive_hwdata** (input reg [31:0] hwdata);
Description: Drives hwdata with the given value and returns immediately .

6.3.8 Task to get hrdata

task **get_hrdata** (output reg [31:0] hrdata);
Description: Returns hrdata from the bus.

6.4 Configuration tasks

6.4.1 Task to configure busy cycle for subsequent transactions

task **config_busy_gen** (input reg [9:0] busy_start_cnt, input reg [3:0] busy_len, input reg b_random);
Description: Configure or unconfigure the master to insert busy cycles for the subsequent transactions.

busy_start_cnt	Specifies the starting point where BUSY transfer type is inserted in a burst. Busy can not be inserted as a penultimate transfer in a fixed length burst 0 – busy inserted before NON_SEQ
busy_len	specifies number of clocks the busy state is held continuously. **0 – stops busy generation**
b_random	0 – Use busy_start_cnt and busy_len as specified 1 – AHB master selects random values for busy start and busy length for each transaction. • range for start is 0 to busy_start_count • range for length is 1 to busy_len

Example : config_busy_gen (3, 8, 0);

3	after the 3rd transfer BUSY transfer type is inserted.
8	specifies BUSY transfer type is held for 8 clocks
0	random mode is not activated.

6.4.2 Task to configure the master to continue on error response

task **config_continue_on_error**(input reg flag);
Description: Configure or unconfigure the master to continue the current transactions in the case of error response received from AHB slave.

Flag Default - 0	1 – If an error response for an address occurs, the transaction continues with next address. 0 – The transaction is aborted for an error response.

6.4.3 Task to configure misaligned_addressing_mode

task **misaligned_addressing_mode** (input reg flag);	
Description: Configure or unconfigure the master to allow misaligned addressing mode.	
flag Default - 0	1 – Master accepts commands even with misaligned address. The HADDR is driven with the given value as is. This way, misaligned addressing can be forced on the AHB bus to emulate an error condition. 0 – Master rejects commands with misaligned adress. Word transfers must aligned to word address boundaries (that is A [1:0] = 00), half word transfers must be aligned to half word address boundaries (that is A[0] = 0)

6.4.4 Task to configure Endianness

task **config_endian** (input reg flag);	
Description: Configure the master for either big endian or little endian	
Flag Default value: 0	1 – Big Endian. 0 – Little_Endian

6.4.5 Task to configure Lock_end_idle

task **config_lockend_idle** (input reg flag);	
Description: Configure or unconfigure the master to insert HTRANS_IDLE at the end of locked transactions.	
flag Default value: 0	1 – Insert two IDLE's at the end of locked transaction 0 – Continue with next transaction (if any pending commands in the FIFO)

6.4.6 Task to configure Extended request

task **config_extended_request** (input reg flag);	
Description: Configure or unconfigure the master to generate extended request.	
flag Default value: 0	1 – for fixed length bursts generate request upto the last address phase 0 – for fixed length bursts request is deasserted upon receiving the grant

6.4.7 Task to configure transaction message logging

task **config_msg_logging** (input integer level);	
Description: Configure the master for required message logging level.	
level Default value: 0	0 – no logging 1 – log only the reponses of the transaction 2 – log the reponses and data of the transaction

7 Using the Master

This section gives details about the Master entity with port connection details, instantiation and task usage examples. Sample log messages are also shown.

7.1 Master port connections

The following figure shows the port connections and module definition for the behavioral model of the Master VIP.

```
module AHBMaster (
        hclk,
        hresetn,

        // Input From AHB
        hready,
        hgrant,
```

```
        hresp,
        hrdata,

        // Output To AHB
        htrans,
        hwrite,
        hbusreq,
        hsize,
        hburst,
        hprot,
        hlock,
        haddr,
        hwdata
);   // END module AHBMaster

input           hclk;
input           hresetn;

input           hready;
input           hgrant;
input  [1:0]    hresp;
input  [31:0]   hrdata;

output [1:0]    htrans;
output          hwrite;
output          hbusreq;
output [2:0]    hsize;
output [2:0]    hburst;
output [3:0]    hprot;
output          hlock;
output [31:0]   haddr;
output [31:0]   hwdata;
```

7.2 Instantiation

The Master is instantiated in the test-bench as below:

```
AHBMaster  master0 (
        .hclk           (hclk),
        .hresetn        (hresetn),
        .hready         (hready),
        .hgrant         (hgrant),
        .hresp          (hresp),
        .hrdata         (hrdata),
        .htrans         (htrans),
```

```
.hwrite      (hwrite),
.hbusreq     (hbusreq),
.hsize       (hsize),
.hburst      (hburst),
.hprot       ( ),
.hlock       (hlock),
.haddr       (haddr),
.hwdata      (hwdata)
);
```

7.3 Task usage flow

The master model should be configured after the reset has been issued. The reset will bring the master to default configuration.

It may be noted that configuring the master affects all subsequent transactions and any non-executed transactions in the command FIFO. In order to make sure that a new configuration affects only the subsequent transactions, the status of the previous commands should be checked for status-done.

Sequence for initiating a transaction on the AMBA AHB BUS:

- Call a command task (**read** or **write**) with appropriate arguments.
- Check the tag_id returned by the Master to know whether the command has been accepted or rejected by the Master.
- If valid tag_id the command has been accepted by the master. If invalid tag_id decode the tag_id to get the error type and correct the command and re_issue.
- Use the tag_id to check the status of the transaction by calling the **get_status** task. Check the status_done returned by the Master to know whether the transaction has been completed or not.
- For read transactions, use the tag_id to get the data by calling the **get_data** task.

7.4 Task usage example

```
initial begin
    // Wait for reset to complete
    ….
    // Configure the master and other components of the system
    master0.config_busy_gen(1,2,1,1);
    master0.config_Continue_on_Error(1);
    master0.misaligned_addressing_mode (1);
    master0.config_endian (1);
    master0.config_extended_request (1);
    …..
    .....
```

```
// Setup and start transaction
$display ("Initialize write data array");
arr_write_data0[0] = 32'h aaaa_bbbb;

$display ("1. Master0 Performing single  write  ");
master0.write(32'h1111_2220,`HSIZE_WORD, 1, 0, 0, arr_write_data0,tag1);

//get the write status

$display ("2. Get the status of the transaction from master0 ");

status_done = 0;
while (!status_done) begin
  master0.get_status(tag1, status_done, error_count, retry_count, split_count);
 @(posedge hclk);                    // NOTE: Need to advance time within loop

end //while

$display ("2. Master0 Performing single read ");
master0.read (32'h1111_2220,`HSIZE_WORD, 1, 0, 0,tag2);


$display ("3. Reading data from master0 read buffer");
read_done = 0;
while (!read_done) begin
  master0.get_data(tag2, read_done, arr_read_data0,read_data_count);
 @(posedge hclk);                    // NOTE: Need to advance time within loop

end //while

end
```

7.5 Bus-level task usage flow

Scenario: Single write followed by single read

1) Issue REQ
2) Wait for GRANT
3) Drive NS write
4) Drive HWDATA
5) Delay one clock
6) Drive NS read
7) Delay one clock
8) Drive IDLE
9) Release REQ
10) Delay one clock
11) Get HRDATA

7.6 Bus-level task usage example

```
initial begin
// Wait for reset to complete
….
        drive_req_lock( 1,0);
        wait_for_grant( );
        // Issue single word-write
        drive_nonseq (32'h1111_2220, 1, `HSIZE_WORD, `HBURST_SINGLE);
        drive_hwdata (32'h1234_5678);
        @ (posedge hclk);
        // Issue single word-read
        drive_nonseq (32'h1111_2220, 0, `HSIZE_WORD, `HBURST_SINGLE);
        @ (posedge hclk);
        drive_idle ( );
        drive_req_lock (0, 0);
        @ (posedge hclk);
        get_hrdata ( hrdata);
        @ (posedge hclk);
end
```

7.7 Sample log messages

Following are the displayed sample log messages for response log and data log .Note

config_msg_logging ().

7.7.1 Response log

425000ps Master0 Split response for addr 00001879

778000ps Master1 Retry response for addr 00003546

325000ps Master2 Error response for addr 00006543 abort the transaction

476000ps Master3 Error response for addr 00001466, continue with next addr 00001467

7.7.2 Data log

425000ps Master0 Read_data = 2f563216 Addr = 00001460

676000ps Master1 Write_data = 2f563216 Addr = 00006574

VeriFlow Technologies India (P) Ltd