VeriFlow Technologies India (P) Ltd

AHB Slave

Version 1.0, Oct 21, 2008

Renju sebastian

Rev. #	Designer	Description	Date Released
0.1	Renju sebastian	Initial Draft	July 17, 2008
0.2	Prabuddha Khare	Add more narratives	Sep 09, 2008
0.3	Renju sebastian	Add message section	Sep 10, 2008
0.4	Renju sebastian	Add compile section	Oct 01, 2008

Table of Contents

1. Introduction

This is a user-cum-specification document for the AHB slave VIP. The document captures the usage-intent of the module and provides details of all the supported features. There is a section on usage where particulars about instantiation and interconnections are given.

1.1 Intended Audience

The AHB slave VIP is targeted for use by ASIC verification teams working on projects that use the AHB bus as one of the main communication trunk line. Usually this involves several masters and slaves with one or more CPUs to form a system-on-chip or SOC. During development and debugging, the AHB slave is needed to verify master's functionality.

1.2 Intended Usage

The AHB Slave VIP helps in the development of AHBSlave IP and AHB Master IP. It is also used to Verify VIP Models of AHBMaster. The slave module should be instantiated and connected on the test-bench. It can be also used for spurious traffic in system level verification of an SOC.

1.3 Delivery Overview

The VIP delivery consists of the following two files. The first is the main functional module and the second is an include file containing constant definitions to configure the various VIP within a AHB system.

AHBSlave.sv
AHBDefines.svh

3.Overview

The slave model is a behavioral model coded in Verilog. This is implemented as a memory model of size 64k with address-wrap on the page. It has a task interface that the test writer may use to configure the slave to generate various responses and thus create any possible AHB transaction scenarios. It supports locked transfers and early-burst termination.

4.Features Supported by Slave AHB

- Supports configurable Split generation
- Supports configurable Retry generation
- Supports configurable Wait generation
- Supports configurable Error generation
- Supports Early burst termination
- Supports Idle and Busy command
- Supports all possible AHB Burst transactions
- Supports Byte, Halfword and Fullword size transactions
- Supports Endianess

5.Early Burst termination

There are certain circumstances when a burst will not be allowed to complete and therefore it is important that any slave design which makes use of the burst information can take the correct course of action if the burst is terminated early. The slave can determine when a burst has terminated early by monitoring the HTRANS signals and ensuring that after the start of the burst every transfer is labelled as SEQUENTIAL or BUSY. If a NONSEQUENTIAL or IDLE transfer occurs then this indicates that a new burst has started and therefore the previous one must have been terminated.
If a bus master cannot complete a burst because it loses ownership of the bus then it must rebuild the burst appropriately when it next gains access to the bus. For example, if a master has only completed one beat of a four-beat burst then it must use an undefined-length burst to perform the remaining three transfers.

6.Split Transfer

SPLIT transfers improve the overall utilization of the bus by separating (or splitting) the operation of the master providing the address to a slave from the operation of the slave responding with the appropriate data.

When a transfer occurs the slave can decide to issue a SPLIT response if it believes the transfer will take a large number of cycles to perform. This signals to the arbiter that the master which is attempting the transfer should not be granted access to the bus until the slave indicates it is ready to complete the transfer. Therefore the arbiter is responsible for observing the response signals and internally masking any requests from masters which have been SPLIT.

7. Granular control via user configuration tasks from user tests.

The slave model provides user configurations via task interface. The test writer can call these tasks with appropriate parameters to enable or disable various features supported by the model.

All of the four response related tasks take a parameter that specifies the number of subsequent transactions for which the corresponding response is to be generated. The response generation automatically gets disabled after the specified number of transactions and the slave returns to the default mode in which it generates OK response without any wait-state insertions.

The starting beat on which the response is to be generated and the number of clocks for which it must be held can be specified in the parameters. The slave model supports a random-mode in which, when enabled for a specific response by passing a flag to the corresponding task, it will cause the slave to use a randomly generated value between the given limits.

Over and above the per response random mode, the model also supports a general random mode where any response is randomly generated. This must be configured using a seperate task which overrides all individually set response modes.

The following is a detailed list of tasks that are supported by AHB-Slave:

1) Configure **wait state generation** for subsequent read/write transactions.
2) Configure **retry response generation** for subsequent read/write transactions.
3) Configure **split response generation** for subsequent read/write transactions.
4) Configure **error response generation** for subsequent read/write transactions.
5) Configure slave to generate **random response** for all subsequent transactions.
6) Perform **back door read** from slave memory
7) Perfom **back door write** to slave memory
8) Get status of **hmastlock**
9) Configure **endianess**
10) Disable **all response generation** at once

7.1) Configure wait state generation for subsequent read/write transactions.

```
task config_wait_states (input reg [3:0] hrdy_start_cnt,  input reg [3:0] wait_len,
                         input reg [3:0]wait_trans_cnt , w_random,
                         input reg [3:0]wait_hmaster);
```

hrdy_start_cnt	Specifies the starting point where wait response is generated in a burst. For single transfers this must be zero otherwise wait condition will not be indicated.
wait_len	Specifies number of clocks the wait state is generated. Max up to 16 wait states supported
wait_trans_cnt	Specifies number of transactions wait state is generated. Max up to 16 wait transaction supported
w_random	If this bit is set it randomly select **hrdy_start_cnt** and **wait_len**
wait_hmaster	Specifies the master id for which the slave should apply wait states. Use 4'bxxxx to match any and all masters.

Example : config_wait_states (3, 8, 2, 0, 12);

3	Insert wait-state after 3rd beat. <NS><SEQ><SEQ><WAIT-8><SEQ>
8	Hold the wait-states for 8 clocks.
2	Wait state insertion will be done for 2 consecutive transactions
0	No random mode activated.
12	Wait state inserted for transactions issued by master 12.

7.2) Configure retry response generation for subsequent read/write transactions.

task **config_retry_gen** (input reg[3:0] **retry_start_cnt**, input reg[3:0] **retry_cnt**, input reg[3:0] **retry_trans_cnt**, **r_random**, input reg[3:0] **retry_hmaster**);

retry_start_cnt	Specifies the starting point where retry response is generated in a burst. For single transfers this must be zero otherwise wait

	condition will not be indicated.
retry_cnt	Specifies the number of retries to generate, up to 1 to 16 retries supported.
retry_trans_cnt	Specifies the number of transactions retries to generate, up to 16 retry transaction supported.
r_random	If this bit is set it randomly select **retry_start_cnt** and **retry_len**
retry_hmaster	Specifies the master id for which the slave should retry the transaction .Use 4'bxxxx to match any and all masters.

Example : config_retry_gen (3, 8, 2, 0, 12);

3	Insert retry-state after 3rd beat. <NS><SEQ><SEQ><RETRY-8><SEQ>
8	Hold the retry-states for 8 clocks.
2	Retry state insertion will be done for 2 consecutive transactions
0	No random mode activated.
12	Retry state inserted for transactions issued by master 12.

7.3) Configure split response generation for subsequent read/write transactions.

task **config_split_gen** (input reg [3:0] **split_start_cnt**, input reg [3:0] **unsplit_delay**, input reg [3:0] **split_trans_cnt**, **s_random**, input reg [3:0] **split_hmaster**);

split_start_cnt	Specifies the starting point where split response is generated in a burst. For single transfers this must be zero otherwise wait condition will not be indicated
unsplit_delay	Specifies the number of split to generate.

split_trans_cnt	Specifies the number of transaction split to generate
s_random	If this bit is set it randomly select **split_start_cnt** and **split_len**
split_hmaster	Specifies the master id for which the slave should split the transaction.Use 4'bxxxx to match any and all masters.

Example : config_split_gen (3, 8, 2, 0, 12);

3	Insert split-state after 3rd beat. <NS><SEQ><SEQ><SPLIT-8><SEQ>
8	Hold the split-states for 8 clocks.
2	Split state insertion will be done for 2 consecutive transactions
0	No random mode activated.
12	Split state inserted for transactions issued by master 12.

7.4) Configure error response generation for subsequent read/write transactions.

task **config_error_gen** (input reg[15:0] **error_start_cnt**, input reg[15:0] **err_trans_cnt**, **error_hmaster**);	
error_start_cnt	Specifies the starting point where error response is generated in a burst. For single transfers this must be zero.
err_trans_cnt	Specifies the number of transaction error to generate.
error_hmaster	Specifies the master id for which the slave should produce error transaction.Use 4'bxxxx to match any and all masters.

Example : config_error_gen (3, 1, 12);

3	Insert error-state after 3rd beat. <NS><SEQ><SEQ><ERROR-1>
1	Hold the error-states for 1 clocks.
12	Error state inserted for transactions issued by master 12.

7.5) Configure slave to generate random response for all transactions.

task **config_random_response(input reg [3:0] ul_start, input reg [3:0] ul_cnt_dly, input reg [3:0] trns_cnt);**	
u1_start	Specifies the starting point where response is generated in a burst.
u1_cnt_dly	Specifies the number of response delay
u1_cnt_dly	Specifies the number of transaction

Example : config_random_response(7, 8, 10);

7	**Starting point where response will start**
8	**Number of response delay**
10	**Number of transaction**

7.6) Perform back door read from slave memory

task **bd_read** (input reg [31:0] **r_address**, input reg [2:0] **r_size**, output reg [31:0] **r_data**);	
r_address	Specifies the address from which slave can read data.
r_size	Specifies the size that should read.
r_data	Output data

Example : bd_read (32'h1234_5678, `HSIZE_BYTE, read_data);

32'h1234_5678	Specifies the address for reading data.

`HSIZE_BYTE	Specifies byte read
read_data	Output data reg

7.7) Perform back door write to slave memory

task **bd_write** (input reg [31:0] **w_address**, input reg [2:0] **w_hsiz**e, input reg [31:0] **w_data**);	
w_address	Specifies the address to write the data
w_hsize	Specifies the size that should write the data
w_data	Specifies the data to be written in the address.

Example : bd_write (32'h1234_5678, `HSIZE_BYTE, write_data);

32'h1234_5678	Specifies the address for writing data.
`HSIZE_BYTE	Specifies byte write
write_data	Written data reg

7.8) Get status of hmastlock

task **get_lkd_status**(**output reg lk_status**);	
lkd_status	Gives the status of transaction locked or unlocked

Example : get_lkd_status (status_reg);

status_reg	if its 1 ,then its locked or unlocked if 0.

7.9) Task to configure endianess

task **set_endianess** (input reg set_flag);
set_flag **:** 0 --> little endian. 1 --> big endian

Example : set_endianess (1/0);

7.10) Task to disable all response generation

task **config_ok_gen** ();
if this task is evoked then wait, split, retry, error tasks will be disabled.

Example : config_ok_gen ();

8.Using the Slave

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

8.1 Slave port connections

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

```
module AHBSlaveBeta (hready,
                     hresp,
                     hrdata,
                     hsplit,
                     hreadyi,
                     hsel,
                     haddr,
                     hwrite,
                     hsize,
                     htrans,
                     hburst,
                     hwdata,
                     hresetn,
                     hclk,
                     hmaster,
                     hmastlock);


output          hready;
output [1:0]    hresp;
output [31:0]   hrdata;
output [15:0]   hsplit;
input           hreadyi;
input           hsel;
input           hclk;
input           hresetn;
input           hmastlock;
input           hwrite;
input [31:0]    haddr;
input [2:0]     hsize;
input [1:0]     htrans;
input [2:0]     hburst;
input [31:0]    hwdata;
input [3:0]     hmaster;
```

8.2 Instantiation and task usage

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

```
  AHBSlaveBeta slave0 (
    .hready                 ( hready    ),
    .hresp                  ( hresp     ),
    .hrdata                 ( hrdata    ),
    .hsplit                 ( hsplit    ),
    .hreadyi                ( hready    ),
    .hsel                   ( hsel      ),
    .haddr                  ( haddr     ),
    .hwrite                 ( hwrite    ),
    .hsize                  ( hsize     ),
    .htrans                 ( htrans    ),
    .hburst                 ( hburst    ),
    .hwdata                 ( hwdata    ),
    .hresetn                ( hresetn   ),
    .hclk                   ( hclk      ),
    .hmaster                ( hmaster   ),
    .hmastlock              ( hmastlock )
  );
```

The following is a typical usage of the slave from a user-test:

```
initial begin
  // Wait for reset to complete
    …
  // Configure slave for wait-generation
       slave0.config_wait_states(0, 5, 1, 0, 12);

  // Perform the test sequence
  …
  …

  $finish();

end  // End initial
```

8.3 Compiling the Slave

As slave model uses Systemverilog constructs so we need to evoke sv switch for appropriate tools

For example :

Axiom :hdloffice +sv +define+$1 +$2 +incdir+<file directory> -f ../Sv.f -o $1_svsim

VCS :vcsi -PP -R -V -notice -sverilog -ntb_opts dtm +incdir+<file directory> +systemverilogext+.sv -f trial.f |tee run.log