



Macrocad Development Inc.

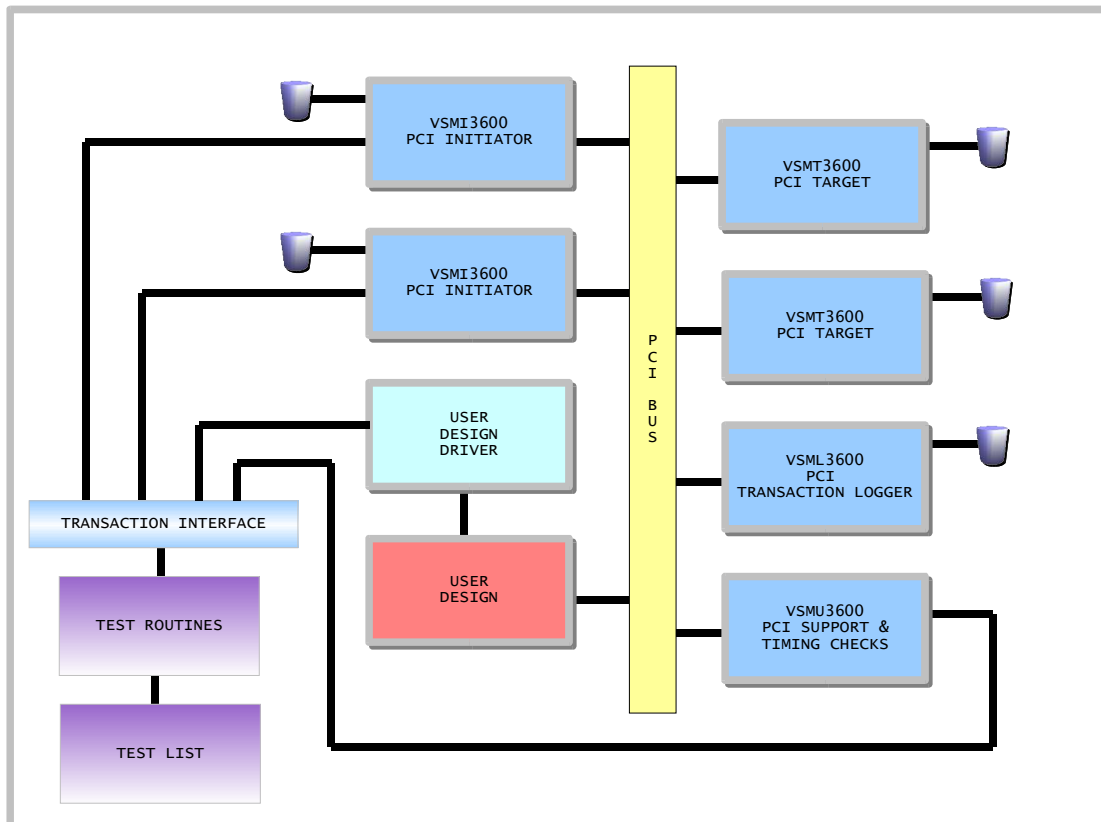
VSM 3600 PCI Bus Functional Models

Description:

Macrocad's VSM 3600 models are HDL behavior models for creating a virtual reality system simulation environment based on the PCI bus specifications. System developers will appreciate the ease of use, and features which allow the designer to create a realistic PCI system environment. Each model is an independent agent - there are no interdependencies to constrain the designer, or to make the system artificial in its operation. Operating parameters are passed to each model to characterize their behavior. Master models execute an instruction list so PCI transaction sequences are easy to control.

Features:

- ◆ Independent agents
- ◆ Realistic system simulation
- ◆ Automated function and timing checks
- ◆ PCI 2.2 Compliant
- ◆ All PCI transactions are logged
- ◆ Debug features are included
- ◆ Each PCI frame controllable
- ◆ Agents operates from an instruction list
- ◆ 64 bit & 66MHz options supported
- ◆ Designed by hardware engineers for hardware engineers
- ◆ System backbone and tests Included
- ◆ Includes master / slave / logger / support



System Models

SYS_3600 System Model

This model is the system interconnect backbone for the PCI based system simulation environment. All PCI devices, structural or behavioral "plug" in to this backbone. A variety of masters and slaves can be instantiated on this system backbone, as well as user designs. Lists of test are also included at this level. These test instructions call routines which communicate through a transaction interface bus to the PCI bus agent models, and user designs.

VSMU3600 Bus Support Model

This model provides the basic arbitration function for PCI bus devices, generates the PCI clock, checks signal timing during PCI transactions, and generates PCI reset. The user can choose from several arbitration schemes. There is an optional PCI bus activity display, (to Std Out). Control of clocks, reset, error checking, and arbitration is accomplished through the transaction interface. Except for generating the clock, reset and arbitrating between competing masters, the VSMU3600 model does not participate in any PCI bus activity.

VSML3600 Transaction Logger

This model logs all PCI bus transactions to a file. This includes burst, configuration, and special cycles. Bus ownership, parity, termination status, etc. are logged for each transaction. Each transaction is assigned a frame number, which is consistent for all logs files generated by this, or any master or slave model. This model does not participate in any PCI activity, but monitors all PCI activity.

VSMT3600 Target or Slave Model

This model acts as a PCI bus slave, or target. It responds to all PCI transactions if the address and selection criteria are met. PCI transactions which this model participates in, can be logged to a file. Each target model has a unique configuration space. The base address registers of the configuration space are used to determine the behavior of the target model. This includes the memory and IO aperture settings. This configuration information can be accessed via the normal configuration cycle, thus putting it under program control. Register assignments in the configuration space are based on the PCI spec v2.2.

VSMI3600 Initiator or Master Model

This model acts as a PCI bus master. It will request the bus and, when granted, execute a PCI bus transaction in response to a transaction request. PCI

transactions which this model participates in, can be logged to a file. Each initiator model can execute a variety of PCI burst and non-burst frames. The frame termination, parity and other errors, data pacing characteristics and data pattern can be modified on a frame by frame basis.

System Test Creation and Execution

In order to construct a test which involves the participation of several models, each model must be properly configured, and set up at the system level. Once the models are configured, the transactions specified must be consistent with the test objective. For example, an access to a memory location outside the devices' memory aperture will result in a master abort PCI transaction. This would not achieve the test objective if that objective was to transact data with a specific PCI slave device. It would achieve the test objective, if that objective was to test the PCI device's selection logic, (as a way of verifying the PCI slave did not respond to an address outside its specified aperture). Thus tests can be constructed to verify the ability to handle all normal data transactions to a specific PCI device. Tests can also be constructed to verify that a specific PCI device handles error conditions properly.

Test Flow

The test is determined by the list of transaction instructions which is executed by the models. Each instruction request is deposited in an instruction cue. The model interprets the request and translates the instruction into a PCI transaction. In the case of a PCI transfer request, the model manages all the PCI protocol handshaking, and timing constraints. It executes the PCI transfer, and at the completion, will respond by returning the results and status of the PCI transfer on the transaction return vector.

Model Configuration and Operating Parameters

Operational parameter information can be entered as constants at the system (SYS_3600) level. These constants include the log file names, enables, speed, data width, etc. The parameters are to be considered constants, but some are used to set variables which can be changed dynamically during simulation time. One such example is `p_debug_level`. This is used to load an initial value at the start of simulation time into the `r_debug_level` register. This `r_debug_level` register can be dynamically changed during simulation, so display options can be switched on and off based on simulation time, sequential test instructions, PCI status, error conditions, or other events. The logger, initiator, target, and support models also have a local debug switch. The `p_debug_local` parameter is bit-wise ored at the start of simulation time with the `p_debug_level` parameter before it is loaded into the `r_debug_level` register. In this way, the user can set a global debug value, and individually set values local to the model which enable added debug features.

Model	Parameter	Radix	Description
VSML3600 VSMI3600 VSMT3600 VSMU3600	VERSION	31:0	This contains the version number for the model.
VSML3600 VSMI3600 VSMT3600	p_log_file_name	string	This string indicates the name of the log file.
VSML3600 VSMI3600 VSMT3600	p_log_enable	integer	If set to 1, this will enable the generation of a PCI bus transaction log file.
VSML3600 VSMI3600 VSMT3600 VSMU3600	p_debug_level	15:0	15:8 RESERVED 7 RESERVED 6 RESERVED 5 Operation information reported 4 Model information reported 3 Timing information reported 2 RESERVED 1 Data transaction errors reported 0 Stop on error enabled
VSMI3600 VSMT3600 VSMU3600	p_clk_period	integer	This determines the PCI clock speed for the PCI bus. The models currently support 33 and 66 Mhz clock speeds.
VSMI3600 VSMT3600	p_dev	integer	This indicates the arbitration slot for the model.
VSMI3600 VSMT3600	p_pest	integer	This enables the pessimistic timing method used for generating PCI signals.
VSMI3600 VSMT3600	p_data_size	integer	This determines the address and data width for the PCI bus agent. The models currently support 32 and 64 bit PCI bus widths.
VSMT3600	p_cfg_mask_bar	31:0	This will determine the size of the apertures used in the target. See the PCI spec for the Base Address Register definition.
VSMU3600	p_pdly	integer	Propagation delay for reset and grant
VSMU3600	p_arb_type	integer	Arbitration type

Initiator Model Operational Controls

The VSM 3600 models are each independent PCI bus agents. The transaction interface used to communicate with the models consists of two 192 bit ports. The request port is an input to the model, and the response port is an output from the model. Each 192 bit vector has three 64 bit fields, data, address and control. By constructing a request vector, the tests within the test bench can make the initiator execute any defined PCI bus frame. In addition, there are requests which are sideband requests. These sideband requests will not result in a PCI frame, but are used for control, status inquiry, and otherwise communicate with the model. Below is a detailed list of the fields and a breakdown of the functions.

Transaction request

NAME	SIZE	FUNCTION	SIZE	DESCRIPTION
mac_req 191:128	64	address 63:0	64	Address for requested PCI frame
mac_req 127:64	64	data 63:0	64	Data for requested PCI frame
mac_req 63:0	64	control 63:0	64	Control fields for requested PCI frame

Request address field

NAME	SIZE	FUNCTION	DESCRIPTION
w_addr_req	64	address 63:0	Address for requested PCI frame

Request data field

NAME	SIZE	FUNCTION	DESCRIPTION
w_data_req	64	data 63:0	Data for requested PCI frame, or sideband register

Initiator Request control field

NAME	SIZE	FIELD	SIZE	DESCRIPTION
Control 63:56	8	OPT7	8	Number of bytes requested, bits 15:8
Control 55:52	4	TAGS	4	Request tag
Control 51:48	4	TAGS	4	Response tag
Control 47:40	8	BRST	8	Number of bytes requested, bits 7:0
Control 39:32	8	PATT	8	Data pattern The beat contains the data which is in the data field. The subsequent data transferred is generated on the basis of the patterns defined below. : 0x00 : data = data : 0x01 : data = 0xffffffffffffff : 0x02 : data = data : 0x03 : data = invert data : 0x04 : data = increment data : 0x05 : data = decrement data : 0x06 : data = byte increment data : 0x07 : data = byte decrement data : 0x08 : data = byte rotate left : 0x09 : data = byte rotate right : 0x0a : data = bit rotate left : 0x0b : data = bit rotate right : 0x0c : data = alternate byte invert : 0x0d : data = nibble swap : 0x0e : data = xor upper 32 with lower : 0x0f : data = mutant crc pattern : 0x12 : data = increment each byte by 0x08 : 0x13 : data = decrement each byte by 0x08 : others : data = RESERVED
Control 31:24	8	OPT3	8	Option 3 register : 31 : Post PCI transaction Do not wait for PCI transfer to complete, return transaction on next clock : 30 : Enable automatic resume after disconnects : 29:28 : RESERVED : 27 : disable status return : 26 : fast back-to-back frames enable : 25 : set lock : 24 : hold request until frame end
Control 23:16	8	OPT2	8	Option 2 register : maximum latency timeout
Control 15:8	8	OPT1	8	Option 1 register : 15 : generate 64 bit parity error : 14:12 : generate parity error on this data beat : 11 : generate serious error : 10 : generate parity error : 9 : drive lower 2 address bit request to PCI : 8 : 1 = status returned in data field 0 = data returned in data field
Control 7:4	4	CMMD	4	Sideband commands : 7:4 = 0000 : PCI instructions on 3:0, no sideband : 7:4 = 1011 : set sideband register, (write) : 7:4 = 1010 : get sideband register, (read) : others : reserved
Control 3:0	4	CMMD	4	PCI command PCI command as defined by PCI spec.

Initiator Return status field

NAME	SIZE	FIELD	SIZE	DESCRIPTION
Status 63:56	8	STS7	8	Number of bytes transacted, bits 15:8
Status 55:52	4	TAGS	4	Request tag
Status 51:48	4	TAGS	4	Response tag, mirrored from request's request tag
Status 47:40	8	BRST	8	Number of bytes transacted, bits 7:0
Status 39:32	8	RSV4	8	RESERVED
Status 31:24	8	RSV3	8	RESERVED
Status 23:16	8	STS2	8	Status 2 register : 23:18 : RESERVED : :17 : core has latency timeout : :16 : core detects interrupt
Status 15:8	8	STS1	8	Status 1 register : :15 : disconnect detected : :14 : retry detected : :13 : target abort received : :12 : master abort detected : :11 : serious error detected : :10 : parity error detected : :9 : data error detected : :8 : return status, not data
Status 7:4	4	CMMD	4	Sideband commands, mirrored from request
Status 3:0	4	CMMD	4	PCI command, received from core

The transactions can be categorized in to two general classes of transactions. The first will result in a PCI transfer(s). The second is used to control the features of the models, these are called sideband transactions. The sideband registers control these model features.

Initiator Sideband Registers

NAME	ADDRESS PNEUMONIC	SIZE	FUNCTION	DESCRIPTION
r_mask	p_data_mask	64	R/w	Data compare mask register : 63:0 : data mask register used to mask PCI data mis-compares
r_pace	p_data_pace	64	R/w	Data pacing PCI data pacing register with data Selects data pacing for each data beat : 63:60 : delay 16 th data beat of burst : 59:56 : delay 15 th data beat of burst : 55:52 : delay 14 th data beat of burst : 51:48 : delay 13 th data beat of burst : 47:44 : delay 12 th data beat of burst : 43:40 : delay 11 th data beat of burst : 39:36 : delay 10 th data beat of burst : 35:32 : delay 9 th data beat of burst : 31:28 : delay 8 th data beat of burst : 27:24 : delay 7 th data beat of burst : 23:20 : delay 6 th data beat of burst : 19:16 : delay 5 th data beat of burst : 15:12 : delay 4 th data beat of burst : 11:8 : delay 3 rd data beat of burst : 7:4 : delay 2 nd data beat of burst : 3:0 : delay 1 st data beat of burst
r_optn	p_optn	64	R/w	PCI frame options : 63:8 : RESERVED : 7:0 : load byte enable mask with data
r_debug_level	p_debug	64	R/w	Debug switches : 63:16 : RESERVED : 15:0 : debug switch register see debug switch definitions
r_frame_cnt	p_frame_cnt	64	R/w	PCI frame counter : 63:32 : RESERVED : 31:0 : PCI frame count
r_xactn_cnt	p_xactn_cnt	64	R/w	Application transaction counter : 63:32 : RESERVED : 31:0 : Application transaction count
r_retry_int	p_retry_int	64	R/w	Initiator interval between retries : 63:0 : The number of clocks between initiator retries to a target device if the device signals a target retry.
r_retry_lmt	p_retry_lmt	64	R/w	Initiator retry limit : 63:0 : Number of retries before posting an error indicating that the target device did not respond.
r_blk_cmmd	p_blk_cmmd	64	R/w	Block move command : 63:1 : RESERVED : 0 : Deposit file data into buffer
r_blk_stat	p_blk_stat	64	R/w	Block move status : 63:3 : RESERVED : 2 : Initiator busy : 1 : Block load in progress : 0 : RESERVED
r_buf_name	p_buf_name	64	R/w	Block buffer file name : 63:0 : 8 character file name
r_buf_locat	p_buf_locat	64	R/w	Block buffer location : 63:0 : 8 character file location info

The data mask register will apply a mask to the data checking circuit. On any normal PCI frame, this mask will cause this checking circuit to ignore bits where

the mask bit is set to 1. This can be useful for looping or polling for a bit to be set indicating completion of a task.

The data pacing register will delay data availability to the PCI bus for each cycle of the subsequent PCI frames based on the contents of this register.

The debug register is comprised of switches which control the debug and error reporting mechanisms.

The option register controls the byte enable mask. The byte enable mask is used to create non-contiguous PCI data transfers. These are not usually useful, but they are allowed by the PCI spec, and thus should be checked.

The retry interval register establishes the time in PCI clocks between retrying a PCI transfer to a PCI target device.

The retry limit register establishes the maximum number of retries allowed before posing an error.

The retry registers enable the initiator model to execute automatic retries to a PCI device with the following constraints. If the limit register is set to 0 and if a retry is received from a PCI target device, it will report this back as part of the status returned on the transaction. This will be the end of the current transaction, and the initiator activity. However, if the retry limit register is greater than 0, then if a retry is received from a PCI target device it will retry the access to the target device after the number of PCI clocks determined by the interval register. It will continue to retry the PCI transfer the number of times indicated in the limit register. Thus until the target device responds with data, an abnormal PCI frame termination, or the retry limit is reached, the initiator device is busy and unavailable for transactions, since a transaction is still active. The log files will of course detail all the PCI activity, regardless of transaction activity.

Initiator Response data field

NAME	SIZE	FUNCTION	DESCRIPTION
r1_dat_rcv	64	PCI data	Normal data response from PCI frame
r_sbr_datr	64	Sideband data	Sideband register data
r_stat	64	status	Initiator status returned on error : 63:12 : RESERVED : 11 : serious error detected : 10 : parity error detected : 9 : target abort detected : 8 : master abort detected : 7 : PCI transfer data error detected : 6 : PCI serious error reported : 5 : PCI parity error reported : 4 : latency timeout detected : 3 : lock cycle detected : 2 : interrupt detected : 1 : retry detected : 0 : disconnect detected

The initiator and PCI frame status can be returned instead of data. This can happen because of specifically selecting this through the opt1 field in the request control field, (control bit 8). This can also happen if an error is detected by the initiator in the PCI frame. The transaction control return field will indicate if status is being returned rather than data by having control field bit 8 set.

Test Call Definition

Manipulating each bit of each transaction could be tedious. Thus there are test routines which will automatically take care of some housekeeping issues. Each bus agent which can be controlled through a transaction interface will have a routine which will automatically generate the request tag, and also check if the return is status or data. The test routine is called with one line such as what is shown below. This call will result in the initiator requesting and gaining access to the PCI bus, and executing the requested PCI frame. This example is a test routine call for the device in slot 4, (routine dev_4). The user simply provides the address, data, type of PCI frame requested, and also the byte count and data pattern.

```
dev_4(64'h000001034,64'h00000000,v_datr,8'h04,8'h04,p_wr_cfg);
```

----- instruction type
----- data pattern
----- burst byte length
----- returned PCI data / status
----- expected PCI data
----- PCI address
----- device slot routine name

Typical Test Sequence

In this example, the master device in slot 4 writes to the configuration space for device 2. Device 2 is a target device connected in slot 2, and uses address bit 12 to enable configuration selection.

```
dev_4(64'h0000000000001004,64'h0000000000000003,v_datr,8'h04,8'h04,p_wr_cfg);
dev_4(64'h0000000000200403,64'h0000000044000000,v_datr,8'h01,8'h00,p_wr_mem);
dev_4(64'h0000000000200403,64'h0000000044000000,v_datr,8'h01,8'h00,p_rd_mem);
```

Device 4 then writes one byte to a location within device 2's address aperture. It then reads back this data. All the data checks and timing checks are automated.

```
v_slv2_ep0 = 64'h0000000000200000;
v_datw    = 64'hf92e1d47a038b712;
v_offset  = 8'h00;
for (v_burst=8'h01; v_burst<=8'h13; v_burst=v_burst+8'h01)
begin
  for (v_offset=8'h00; v_offset<8'h0a; v_offset=v_offset+8'h01)
  begin
    v_addr = v_slv2_ep0 + {56'h0,v_offset};
    dev_1(v_addr,v_datw,v_datr,v_burst,8'h0a,p_wr_mem);
    dev_1(v_addr,v_datw,v_datr,v_burst,8'h0a,p_rd_mem);
  end
end
```

In this example, device 1 will write, then read device 2 in a loop which will skew address and burst values across boundaries to check corner conditions. Variables are used to simplify this process.

Retry Operations

The retry registers enable the initiator model to execute automatic retries to a PCI device with the following constraints. If the limit register is set to 0 and if a retry is received from a PCI target device, it will report this back as part of the status returned on the transaction. This will be the end of the current transaction, and the initiator activity. However, if the retry limit register is greater than 0, then if a retry is received from a PCI target device it will retry the access to the target device after the number of PCI clocks determined by the interval register. It will continue to retry the PCI transfer the number of times indicated in the limit register. Thus until the target device responds with data, an abnormal PCI frame termination, or the retry limit is reached, the initiator device is busy and unavailable for transactions, since a transaction is still active. The log files will of course detail all the PCI activity, regardless of transaction activity.

Block Move Operations

The initiator can move large blocks of data from a file to the PCI bus.

Block move size limit

The block move is limited by the block size limit, 32K bytes.

File specification

The file name is constrained to the 8.3 limit with a 3 character path name. This means that the file name must be 8 characters and the file name extension must be 3 characters separated by a "."). The directory or path name is 3 characters with a path delimiter. The file name and path specification is determined by 2 64 bit sideband registers, `r_buf_locat` and `r_buf_name`. Each of these registers is capable of storing 8 characters.

For example, below are examples for the register values and the resulting file and path.

```
r_buf_name : file0001
r_buf_locat : tst/.dat
resulting file name : tst/file0001.dat
```

```
r_buf_name : patternd
```

r_buf_locat : dat\mem
resulting file name : dat\pattern.d.mem

The register data is written to and stored as ASCII data, so conversions are helpful. There is a simple conversion program for VHDL `f_ascii2hex` which converts ASCII data to hex. The verilog language has builtin conversion features.

The characters are limited to a-f, A-F, 0-9, ., \, and /.

File data format

The file format should be 64 bits of data on each line. The register used to start the file read is the `r_blk_cmmd` register. This is also a sideband register. When this registers bit 0 is set to 1, the specified file will be read and each line will be deposited into the initiator's `r_pat_mem` memory. The initiator can then move this data to the PCI bus, (a PCI write burst operation), by writing another sideband register, `r_blk_cmmd`.

Selecting the buffer memory

The pattern field of the transaction control request will determine if the PCI write data comes from the pattern generator of the buffer memory. If the pattern field is set to x"FE", then the buffer memory will be selected.

Resume

The block moves can be long, so the initiator might not move all the data before being interrupted, either by a latency timeout, or a target disconnect. If the data is not completely written, the initiator will request the bus, and resume the operation as soon as it has been granted the bus. It will continue to do this until the block move is complete. This is controlled by the transaction control bit 30. If this bit is set, then automatic resumes will take place. If the bit is not set, then the transaction will complete if the initiator block move is interrupted for any reason, or when it completes the block move.

Block move execution

Block moves are executed just like any other PCI bursted write. The byte length is specified in the normal way, and the other control operations should work normally. In this way it is just like any other PCI write, but it benefits from the added features like resume and posted operations. Block operations are currently limited to starting on a 64 bit boundary, (8 byte boundary).

Posted operations

The block move can be either posted, or atomic. If it is atomic, the transaction interface will not respond until the entire block is transferred. This is controlled by transaction control bit 31. If this bit is set, then the block move is considered posted, and the transaction will be complete as soon as the transaction request has been issued. This means that the test flow will continue, but the initiator is not available for operations, because it is busy with the block move. In order to know when the block move is complete, a sideband register `r_blk_stat` is provided. This sideband register indicates the busy status. Sideband register accesses are still allowed during these block moves.

The resume and atomic features associated with block moves, are not limited to block moves. These can also work with pattern generated PCI operations as well.

Target Model Operational Controls

The VSMT3600 PCI target model is a passive target device, but can be controlled through the configuration space registers. The definition of these configuration space registers is defined in the PCI specification, 2.2. This spec defines two 32 bit registers which are reserved at location 0x34 and 0x38. In the case of the VSMT3600 target model, these are defined as follows:

NAME	ADDRESS	SIZE	DESCRIPTION
r_cfg_pac1	0x34	32	Target data pacing register : 31:28 : delay data beat 8 : 27:24 : delay data beat 7 : 23:20 : delay data beat 6 : 19:16 : delay data beat 5 : 15:12 : delay data beat 4 : 11:8 : delay data beat 3 : 7:4 : delay data beat 2 : 3:0 : delay data beat 1
r_cfg_optn	0x38	32	Target frame option register : 31:24 : retry for this many access attempts : 23:20 : RESERVED : 19:12 : generate disconnect or error on this frame beat : 11 : RESERVED : 10 : RESERVED : 9 : 1 = generate disconnect without data : 8 : 0 = generate disconnect with data : 8 : generate disconnect : 7 : enable burst for config cycles : 6 : generate 64 bit parity error : 5 : generate parity error : 4 : generate target abort : 3:2 : RESERVED : 1:0 : delay device select

The behavior of the target can be controlled by accessing these configuration space register. The subsequent accesses will respond according to the values contained in these two registers. When the target is configured to generate a retry, it will retry for the number of access attempts in r_cfg_optn bits 31:24. Subsequent access attempts will result in a normal response. If another retry is desired, this register must be reloaded.

In addition, there are sideband registers which can control aspects of the target model's behavior.

Target Sideband Registers

VSM3600 DATA SHEET

NAME	ADDRESS PNEUMONIC	SIZE	FUNCTION	DESCRIPTION
r_optn	p_optn	64	R/w	PCI frame options : 63:0 : RESERVED
r_debug_level	p_debug	64	R/w	Debug switches : 63:16 : RESERVED : 15:0 : debug switch register see debug switch definitions
r_frame_cnt	p_frame_cnt	64	R/w	PCI frame counter : 63:32 : RESERVED : 31:0 : PCI frame count
r_xactn_cnt	p_xactn_cnt	64	R/w	Application transaction counter : 63:32 : RESERVED : 31:0 : Application transaction count

Logger Model Operational Controls

The VSML3600 PCI bus transaction logger tracks the data transactions within PCI frames. It can detect and report on errors which happen on the PCI bus. In cases where errors are purposefully generated in order to check the proper response of a design, it may be useful to turn off error reporting and stop on error features which are on by default. The switches for these error detection features are in the debug register, `r_debug_level`. In order to access the PCI bus logger's debug register, a sideband transaction is used. In this way, the user can turn the PCI bus logger's stop on error and error reporting features off temporarily for specific tests. This can all be accomplished through the same test bench mechanism. The `r_frame_cnt` register is also available through a sideband transaction.

Logger Sideband Registers

NAME	ADDRESS PNEUMONIC	SIZE	FUNCTION	DESCRIPTION
<code>r_optn</code>	<code>p_optn</code>	64	R/w	PCI frame options : 63:0 : RESERVED
<code>r_debug_level</code>	<code>p_debug</code>	64	R/w	Debug switches : 63:16 : RESERVED : 15:0 : debug switch register see debug switch definitions
<code>r_frame_cnt</code>	<code>p_frame_cnt</code>	64	R/w	PCI frame counter : 63:32 : RESERVED : 31:0 : PCI frame count
<code>r_xactn_cnt</code>	<code>p_xactn_cnt</code>	64	R/w	Application transaction counter : 63:32 : RESERVED : 31:0 : Application transaction count

Support Model Operational Controls

The support model generates reset, the PCI clock, arbitration services, and checks PCI signal timing. Sideband registers provide a means of controlling the clock period, reset generation and arbitration functions.

Support Sideband Registers

NAME	ADDRESS PNEUMONIC	SIZE	FUNCTION	DESCRIPTION
r_optn	p_optn	64	R/w	PCI frame options : 63:21 : RESERVED : 20 : Remove grant after frame starts : 19:16 : delay in arbitration response : 15:8 : PCI bus parking slot select : 7:0 : RESERVED
r_debug_level	p_debug	64	R/w	Debug switches : 63:16 : RESERVED : 15:0 : debug switch register see debug switch definitions
r_frame_cnt	p_frame_cnt	64	R/w	PCI frame counter : 63:32 : RESERVED : 31:0 : PCI frame count
r_xactn_cnt	p_xactn_cnt	64	R/w	Application transaction counter : 63:32 : RESERVED : 31:0 : Application transaction count
r_rst_cnt	p_reset	64	w	PCI reset : 63:8 : RESERVED : 7:0 : reset duration counter initiates PCI reset when loaded
r_cprd	p_cprd	64	R/w	PCI clock period : 63:0 : PCI clock period
r_arb	p_arb	64	R/w	PCI arbitration type : 63:8 : RESERVED : 7:0 : PCI arbitration type

The reset count duration register determines the number of clocks for the PCI reset, and when the register is written with a non-zero value, this will initiate a PCI reset.

The clock period register determines the PCI clock period in nanoseconds.

Transaction Definitions for Logger, Support, and Target models

The logger, support, and target models have a transaction interface which is used to access sideband registers.

Transaction request

NAME	SIZE	FUNCTION	SIZE	DESCRIPTION
mac_req 191:128	64	address 63:0	64	Address for requested sideband register
mac_req 127:64	64	data 63:0	64	Data for requested sideband register
mac_req 63:0	64	control 63:0	64	Control field for sideband register

Request address field

NAME	SIZE	FUNCTION	DESCRIPTION
w_addr_req	64	address 63:0	Address for requested sideband register

Return address field

NAME	SIZE	FUNCTION	DESCRIPTION
w_addr_req	64	address 63:0	RESERVED

Request data field

NAME	SIZE	FUNCTION	DESCRIPTION
w_data_req	64	data 63:0	Data for requested sideband register

Return data field

NAME	SIZE	FUNCTION	DESCRIPTION
w_data_req	64	data 63:0	Data from requested sideband register

Model Request control field

NAME	SIZE	FIELD	SIZE	DESCRIPTION
Control 63:56	8	OPT7	8	RESERVED
Control 55:52	4	TAGS	4	Request tag
Control 51:48	4	TAGS	4	Response tag
Control 47:40	8	OPT5	8	RESERVED
Control 39:32	8	OPT4	8	RESERVED
Control 31:24	8	OPT3	8	RESERVED
Control 23:16	8	OPT2	8	RESERVED
Control 15:8	8	OPT1	8	RESERVED
Control 7:4	4	CMMD	4	Sideband commands : 7:4 = 1011 : set sideband register, (write) : 7:4 = 1010 : get sideband register, (read) : others : RESERVED
Control 3:0	4	CMMD	4	RESERVED

Model Return status field

NAME	SIZE	FIELD	SIZE	DESCRIPTION
Status 63:56	8	STS7	8	RESERVED
Status 55:52	4	TAGS	4	Request tag
Status 51:48	4	TAGS	4	Response tag, mirrored from request's request tag
Status 47:40	8	STS5	8	RESERVED
Status 39:32	8	STS4	8	RESERVED
Status 31:24	8	STS3	8	RESERVED
Status 23:16	8	STS2	8	RESERVED
Status 15:8	8	STS1	8	RESERVED
Status 7:4	4	CMMD	4	Sideband commands, mirrored from request
Status 3:0	4	CMMD	4	RESERVED

Timing checks

PCI signal timing, as described in the PCI specification, is checked by the VSMU3600 support model. This model will check for signal timing compliance at either 33MHz, or 66MHz.

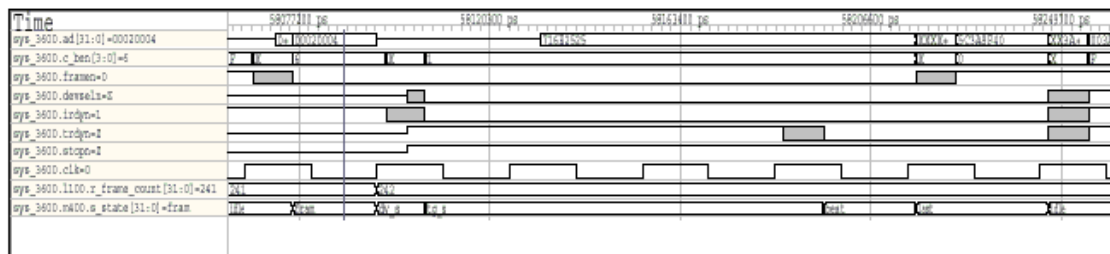
Timing generation

Each PCI model which participates on the PCI bus can generate worst case timing for design verification. The models will either generate the slowest PCI timing, ($p_pest = 0$) or both best and worst case timing, ($p_pest = 1$). These constants are set at compile time. In the case of pessimistic timing, the value of a PCI signal will be held until the fastest timing delay after the rising edge of the PCI clock. It will assume its new value at the slowest specified time after the rising edge of the PCI clock. Between the fastest and slowest times, the signal is driven to an unknown or “x” value. In this way both fast and slow timing cases can be checked while running the simulations.

Additional Features

PCI state trackers

The PCI models can display much information regarding the PCI frame in progress. There are features for displaying PCI frame tracking on waveform viewers. In this way, designers can see the particular state of the PCI bus at any time. Each model has a frame counter for correlation to the log file.



PCI transaction logging

Below are 2 excerpts from a PCI log generated by the PCI bus logger model. Both the initiator and the slave can also generate log files, but only if they participate in the current PCI frame.

```

-----
VSM3600 - VSM 3600 - PCI Bus Logger Device
Reference 'PCI Local Bus Specification Rev. 2.2'
Version - 3.40
Instantiation - sys_3600.1100
Macrocad Development INC. (C) 1997 - 2002
Macrocad Development Inc. - ALL RIGHTS RESERVED
-- CONFIDENTIAL AND PROPRIETARY --
-- USE SUBJECT TO MACROCAD LICENSE AGREEMENT --
-----

```

FRAME	MASTER	BURST	PCI CMD	ADDRESS	DATA	STATUS
241	1	1	Mem Write....	0000000000020000	716e25-----	
241	1	2	Mem Write....	0000000000020008	-----5c3a8f40	
242	1	1	Mem Read....	0000000000020000	716e25-----	
242	1	2	Mem Read....	0000000000020008	-----5c3a8f40	
243	1	1	Mem Write....	0000000000020000	6e25-----	
243	1	2	Mem Write....	0000000000020008	-----3a8f4071	
243	1	3	Mem Write....	0000000000020008	-----5c-----	
.
.
.
.
1689	4	1	Config Read..	000000000008000	86a00006-----	
1690	4	1	Config Write.	000000000008000	01400006-----	
1691	4	1	Config Read..	000000000008000	86a00006-----	
1692	4	1	Config Write.	000000000001030	00000001-----	
1693	4	1	Config Write.	000000000001038	-----00000021	D
1694	4	1	Mem Write....	0000000000020000	0000000000000000	D
1694	4	2	Mem Write....	0000000000020008	0000000000000000	
1694	4	3	Mem Write....	0000000000020010	0000000000000000	
1694	4	4	Mem Write....	0000000000020018	0000000000000000	
1694	4	5	Mem Write....	0000000000020020	-----00000000	
1695	4	1	Mem Read....	0000000000020000	0000000000000000	
1695	4	2	Mem Read....	0000000000020008	0000000000000000	P
1695	4	3	Mem Read....	0000000000020010	0000000000000000	P
1695	4	4	Mem Read....	0000000000020018	0000000000000000	P
1695	4	5	Mem Read....	0000000000020020	-----00000000	P
1696	4	1	Config write.	000000000008038	-----00000041	

The log excerpt shows the frame count, index, PCI command type, PCI address, PCI data, (64 bit justified, byte location indicated), status, byte count in the frame, and master slot. The status can indicate the PCI termination status.

KEY	CONDITION
M	Master abort
W	Wide access, (64 bit access)
T	Target abort
D	Disconnect with data
d	Disconnect without data
S	Serious error detected
P	32 bit parity error detected
p	64 bit parity error detected
R	Retry
L	Locked cycle

Contact Macrocad for more details.

www.macrocad.com