Macrocad DEVELOPMENT INC.

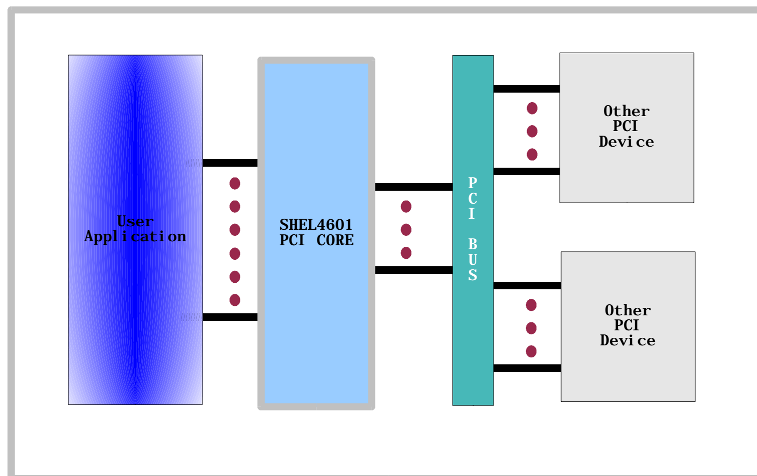
Macrocad Development Inc.
VCM 4601 PCI Synthesizable Core Model

Description:

Macrocad's VCM 4601 synthesizable core provides an easy way to implement a full featured, flexible PCI core. A four stage pipeline supports PCI bursts. Six I/O or memory aperture spaces are supported for 32 bit applications and three are supported for 64 bit applications. Synchronous design results in easy synthesis. Signaling handshakes to and from the application side are easy to interface with. A full test bench is included in the deliverables. This test bench includes automatic timing checks, and transaction logging.

Features:

- PCI 2.2 Compliant
- Synchronous
- Synthesizable
- Initiator and Target Functions
- 32 or 64 bit transfers
- Flexible configuration space registers
- 33 MHz and 66MHz operations supported
- Designed *by* hardware engineers *for* Hardware engineers
- System Backbone, Test Bench Included
- Function, timing checks are Automated
- Independent Master, Slave, Bus Log, and Support Models
- VHDL and Verilog available



PCI CORE DESCRIPTION

The VCM4601 PCI core implements a full function PCI initiator, (master) and target, (slave) interface. The PCI core also includes the configuration space as defined in the PCI spec. This core can operate as a 32 or 64 bit PCI core. This is determined at compile time. The PCI core can be configured to compile as a 64 bit core even though there is a 32 bit application interface.

The PCI core is comprised of an outer shell model and the core model. The shell includes IO pad buffers. The application interface does not contain any IO pad buffering since there are no bi-directional or tristate buffers on the application interface. The core level contains the blocks which make up the PCI / Application function. The core level contains the PCI configuration space and target selection function, address generation, PCI state tracking, target and initiator state machines, parity function, and the send and receive data paths.

In order to provide better throughput for PCI burst transfers, the PCI core provides buffering for data transactions between the PCI bus and the application bus. The PCI core will react with a stall of the appropriate bus if the buffer is empty when providing data, or full when accepting data. The data burst buffering is actually comprised of 3 separate buffers. There are two buffers operating in the "send" mode, where the PCI core provides data to the PCI bus. These are the initiator write buffer and the target read buffer. The "receive" mode buffer is used for both initiator reads and target writes.

The PCI core can be configured to operate in various ways depending on the compile time options which are established.

p_cfg_disco	disconnect with data on cfg target access, (no burst)
p_cfg_ini	initial values of configuration space registers
p_cfg_msk	write masks for configuration space registers

The bus size compile time constants can be set by setting the **PCI_64** and **APP_64** defines. If these are not set, the default is that both bus sizes are 32 bits wide.

PCI CORE SIGNAL DESCRIPTIONS

The PCI interface is comprised of signals specified by the PCI spec, (v2.2). The application interface is comprised of generic signals which can be connected to the user's PCI application logic. These signals are shown below. The signals are grouped in two main categories. These are PCI bus and Application bus signal groups.

PCI Bus Signal Group

Signal Name	*Direction*	*Size*	*Description*
ad	I/O	32	PCI address/data bus
c_ben	I/O	4	PCI command and byte enable <low>
framen	I/O	1	PCI frame start <low>
devseln	I/O	1	PCI device select <low>
irdyn	I/O	1	PCI initiator ready <low>
trdyn	I/O	1	PCI target ready <low>
stopn	I/O	1	PCI stop frame <low>
par	I/O	1	PCI parity bit
perrn	I/O	1	PCI parity error indication <low>
serrn	I/O	1	PCI address parity error indication <low>
id_sel	I	1	PCI configuration device selection bit
reqn	O	1	PCI bus request <low>
gntn	I	1	PCI bus grant <low>
int	O	4	PCI interrupt
rstn	I	1	PCI reset <low>
clk	I	1	PCI clock

The 64 bit PCI bus extensions are optional, and are determined at compile time.

PCI Bus Signal Group, 64 Bit Extensions

Signal Name	*Direction*	*Size*	*Description*
d	I/O	32	PCI address/data bus
ben	I/O	4	PCI byte enable <low>
req64n	I/O	1	PCI request 64 bit transfer <low>
ack64n	I/O	1	PCI acknowledge 64 bit transfer <low>
par64	I/O	1	PCI parity bit

The optional signals shown below are determined at compile time.

PCI Bus Signal Group, Optional Signals

Signal Name	*Direction*	*Size*	*Description*
clkrunn	I	1	PCI clock management
pmen	I	1	PCI power management

Application Bus Signal Group

Signal Name	Direction	Size	Description
app_i_data_snd	I	32/64	initiator cycle, data to PCI
app_i_addr_snd	I	32/64	initiator cycle, address to PCI
app_t_data_snd	I	32/64	target read cycle, data to PCI
app_i_data_rcv	O	32/64	initiator cycle, data from PCI
app_i_addr_rcv	O	32/64	initiator cycle, address from PCI
app_t_data_rcv	O	32/64	target cycle, data from PCI
app_t_addr_rcv	O	32/64	target cycle, address from PCI
app_i_bena_snd	I	4/8	initiator cycle, byte enables to PCI
app_i_bena_rcv	O	4/8	target cycle, byte enables from PCI
app_t_bena_rcv	O	4/8	target cycle, byte enables from PCI
app_i_cmmd_snd	I	4/8	initiator cycle, command to PCI
app_i_cmmd_rcv	O	4/8	target cycle, command from PCI
app_t_cmmd_rcv	O	4/8	target cycle, command from PCI
app_i_bus_req	I	1	initiator cycle, request PCI bus cycle
app_i_last_req	I	1	initiator cycle, last data in burst requested
app_i_ready	I	1	reserved
app_i_valid	O	1	initiator cycle, app data request accepted
app_i_dbeat	O	1	initiator cycle, PCI burst data valid
app_i_busy	O	1	initiator cycle, in process
app_t_hit	O	6	target cycle, target select in range
app_t_chit	O	1	target cycle, target configuration
app_t_dbeat	O	1	target cycle, target data valid
app_t_lbeat	O	1	target cycle, target last data valid
app_t_valid	O	1	target cycle, target access accepted
app_t_ready	I	1	target cycle, target is ready for data
app_i_retry_snd	I	1	initiator cycle, signal access retry
app_t_retry_snd	I	1	target cycle, signal a retry
app_t_disco_snd	I	1	Target cycle, signal a disconnect
app_t_abort_snd	I	1	target cycle, signal an abort
app_i_perr_rcv	O	1	initiator cycle, PCI parity error detect
app_i_retry_rcv	O	1	initiator cycle, PCI retry detected
app_i_disco_rcv	O	1	initiator cycle, PCI disconnect detected
app_i_lat_to_rcv	O	1	initiator cycle, PCI latency timeout detected
app_i_tabort_rcv	O	1	initiator cycle, PCI target abort detected
app_i_mabort_rcv	O	1	initiator cycle, PCI master abort detect
app_t_serr_rcv	O	1	target cycle, PCI address parity error detect
app_t_perr_rcv	O	1	target cycle, PCI parity error detect
app_int	I	1	any cycle, interrupt
app_core32	I	1	Indicates a 32 bit application
pci_buf_ena	I	1	PCI IO pad buffers enable, CardBus option
app_rst	I	1	any cycle, reset
app_clk	I	1	any cycle, clock

PCI CORE FUNCTIONS

The PCI core has the functions necessary to implement a complete PCI solution. This includes a target and initiator PCI device with configuration support.

Initiator Function

As a PCI initiator, the application asserts **app_i_bus_req** when it wants a PCI transaction to occur. If the requested transaction is the last of a burst, the application asserts the **app_i_last_data**. The application must provide valid address, byte enable, and command information on the **app_i_addr_snd**, **app_i_ben_snd**, **app_i_cmd_snd** busses. When it is writing to the PCI bus, then valid data must also appear on the **app_i_data_snd** bus. It must keep this information on these signals until the core's handshake has asserted, **app_i_valid** . This indicates to move to the next data location if a PCI burst is in progress. Assertion of the **app_i_busy** signal indicates that the device has been granted the bus, and the PCI frame has started. The **app_i_dbeat** signal indicates that the data on the **app_i_data_rcv** bus is valid and can be used as a write strobe signal for the application. If a burst is in process, address and byte enable information is advanced when the **app_i_dbeat** signal is detected.

Target Function

As a PCI target, the application interface will indicate that the address decode, PCI instruction, and configuration logic has detected a PCI access in this device's address aperture. When a normal access falls within range, a signal in the **app_t_hit** vector will become active. The handshake between the PCI core and the application will be done with 3 signals. The **app_t_ready** signal indicates that the application has the requested read data available on the **app_t_data_snd** bus. In the case of a PCI write, the **app_t_ready** signal indicates that it is ready to accept data on the **app_x_data_rcv** bus with valid address, and byte enable, and command information on the **app_t_addr_rcv**, **app_t_ben_rcv**, and **app_t_cmd_rcv** busses. The **app_t_valid** signal indicates that the PCI core has accepted the valid data on the **app_t_data_snd** bus. The **app_t_dbeat** signal indicates when the write data is valid on the **app_x_data_rcv** bus, and there is valid address, and byte enable, and command information on the **app_t_addr_rcv**, **app_t_ben_rcv**, and **app_t_cmd_rcv** busses. The **app_t_dbeat** signal can thus be used as a write strobe for the application.

PCI COMMAND SUPPORT

PCI Instruction	Initiator	Target	Description
0000	-	-	Interrupt Acknowledge
0001	-	-	Special Cycle
0010	X	X	IO Read
0011	X	X	IO Write
0100	-	-	RESERVED
0101	-	-	RESERVED
0110	X	X	Memory Read
0111	X	X	Memory Write
1000	-	-	RESERVED
1001	-	-	RESERVED
1010	X	X	Configuration Read
1011	X	X	Configuration Write
1100	-	X	Memory Read Multiple
1101	-	-	Dual Address Cycle
1110	-	X	Memory Read Line
1111	-	X	Memory Write Invalidate

This core is designed for adapter applications. Applications which require bridge functionality, memory subsystem support, cache support, or host processor support will require some minor modification. As a result, PCI signals which are not used in adapter applications are not included, such as lockn. The instructions which are passed from the application to the PCI when the application is an initiator are the same definition as the PCI instructions. Thus as an initiator, some of these unsupported commands can be presented at the application interface, but the result of these commands including response behaviors is outside the specification of this core.

The PCI core supports memory, i/o and configuration reads and writes. The configuration instructions preclude bursting. To the PCI core, the read multiple and read line commands are the same as the memory read command. Likewise, the memory write and invalidate is treated the same as the memory write command. There is no allowance for any memory cache support in the PCI core.

PCI FEATURE SUPPORT

The PCI core supports the following PCI device features, as controlled by the command register of the configuration space.

Configuration Register Index	*Command Register Bit*	*Support*	*Description*
04	0	X	IO Space Enable
04	1	X	Memory Space Enable
04	2	X	Initiator Enable
04	3	-	Special Cycle Response Enable
04	4	-	Memory Write Invalidate Enable
04	5	-	VGA Palette Snoop
04	6	X	Parity Error Detect Enable
04	7	-	Address Data Stepping Enable
05	8	X	Serious Error Enable
05	9	X	Fast Back to Back Enable
05	10-15	-	RESERVED

The PCI Status Register functionality is shown in the table below.

Configuration Register Index	*Status Register Bit*	*Support*	*Description*
06	0-4	-	RESERVED
06	5	X	66MHz Enable
06	6	-	UDF Enable
06	7	X	Fast Back to Back Capable
07	8	X	Master Detected Parity Error
07	9-10	X	DEVSEL Timing
07	11	X	Signaled Target Abort
07	12	X	Received Target Abort
07	13	X	Received Master Abort
07	14	X	Signaled System Error
07	15	X	Detected Parity Error

PCI CONFIGURATION SPACE

The PCI configuration space is accessed as a target. Some of the locations are dynamic storage, (these locations can be modified by PCI configuration space transactions), and some are static. The initial values are configurable by the designer at compile time.

Byte Range	Bytes	Register Name	Description
00:01	2	r_cfg_vendor_id	Vendor ID
02:03	2	r_cfg_device_id	Device ID
04:05	2	r_cfg_command	Command
06:07	2	r_cfg_status	Status
08	1	r_cfg_revision_id	Revision ID
09:0b	3	r_cfg_class_code	Class Code
0c	1	r_cfg_cache_line_size	Cache Line Size
0d	1	r_cfg_latency_timer	Latency Timer
0e	1	r_cfg_header_type	Header Type
0f	1	r_cfg_bist	BIST
10:13	4	r_cfg_bar_0	Base Address Registers
14:17	4	r_cfg_bar_1	Base Address Registers
18:1b	4	r_cfg_bar_2	Base Address Registers
1c:1f	4	r_cfg_bar_3	Base Address Registers
20:23	4	r_cfg_bar_4	Base Address Registers
24:27	4	r_cfg_bar_5	Base Address Registers
28:2b	4	r_cfg_cardbus_cis_pointer	CardBus CIS Pointer
2c:2d	2	r_cfg_subsystem_vendor_id	SubSystem Vendor ID
2e:2f	2	r_cfg_subsystem_id	SubSystem ID
30:33	4	r_cfg_expansion_rom_base_address	Expansion ROM Base Address
34:3b	8	-	RESERVED
3c	1	r_cfg_interrupt_line	Interrupt Line
3d	1	r_cfg_interrupt_pin	Interrupt Pin
3e	1	r_cfg_minimum_grant	Minimum Grant
3f	1	r_cfg_maximum_latency	Maximum Latency

The configuration bytes' write access is determined by user defined compile time parameters. The initial value for these registers is also defined by a compile time parameter. With this scheme, the user can determine which bits of which registers can be written, and what the initial value which be at reset time. This is done be setting compile time parameters, (constants). The mask compile time parameter is **p_cfg_msk** and the initial value compile time parameter is **p_cfg_ini**. These are located in the inc/usr_4601.v file, (the inc/usr_4601.vhd file for VHDL).

PCI ADDRESS AND SELECTION LOGIC

The PCI selection logic is contained in the configuration space model. This is due to the close connection with the speed sensitive address aperture select logic, and its close connection to the PCI configuration space. Both normal IO and memory access selections and configuration selection is accomplished in this functional block. The PCI core supports up to 6 base address registers as a 32 bit core, and up to 3 as a 64 bit core, with each one having either the IO or memory definitions for the bits. The 32 bit Base Address Register definitions are described below.

Base Address Register	*Index*	*Bits 31:4*	*Bit3*	*Bit 2*	*Bit 1*	*Bit 0*
r_cfg_bar_0	10:13	BAR0 31:4	BAR0 3	BAR0 2	RSVD	IF = 1
		BAR0 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_1	14:17	BAR1 31:4	BAR1 3	BAR1 2	RSVD	IF = 1
		BAR1 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_2	18:1b	BAR2 31:4	BAR2 3	BAR2 2	RSVD	IF = 1
		BAR2 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_3	1c:1f	BAR3 31:4	BAR3 3	BAR3 2	RSVD	IF = 1
		BAR3 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_4	20:23	BAR4 31:4	BAR4 3	BAR4 2	RSVD	IF = 1
		BAR4 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_5	24:27	BAR5 31:4	BAR5 3	BAR5 2	RSVD	IF = 1
		BAR5 31:4	prefetchable	type	type	IF = 0

The 64 bit Base Address Register definitions are shown below.

Base Address Register	*Index*	*Bits 31:4*	*Bit3*	*Bit 2*	*Bit 1*	*Bit 0*
r_cfg_bar_0	10:13	BAR0 31:4	Addr BAR0 3	Addr BAR0 2	RSVD	IF = 1
		BAR0 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_1	14:17	BAR0 63:36	BAR0 35	BAR0 34	BAR0 33	BAR0 32
r_cfg_bar_2	18:1b	BAR1 31:4	BAR1 3	BAR1 2	RSVD	IF = 1
		BAR1 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_3	1c:1f	BAR1 63:36	BAR1 35	BAR1 34	BAR1 33	BAR1 32
r_cfg_bar_4	20:23	BAR2 31:4	BAR2 3	BAR2 2	RSVD	IF = 1
		BAR2 31:4	prefetchable	type	type	IF = 0
r_cfg_bar_5	24:27	BAR2 63:36	BAR2 35	BAR2 34	BAR2 33	BAR2 32

MODEL FUNCTION DESCRIPTIONS

The core4601 model has only unidirectional signals. The bidirectional signal functions for the PCI bus are contained in the shell level, shel4601. The structure of these models are shown below.

```
.--------------------------------
|- shel4601 : top
`--------------------------------
1 |- pci_pad_01 : p403
1 |- pci_pad_01 : p408
1 |- pci_pad_01 : p464
1 |- pci_pad_01 : p411
1 |- pci_pad_01 : p402
1 |- pci_pad_01 : p414
1 |- pci_pad_01 : p407
1 |- pci_pad_32 : p164
1 |- pci_pad_01 : p409
1 |- pci_pad_01 : p413
1 |- pci_pad_01 : p405
1 |- pci_pad_01 : p400
1 |- pci_pad_01 : p412
1 |- pci_pad_01 : p406
1 |- pci_pad_01 : p404
1 |- pci_pad_04 : p232
1 |- pci_pad_01 : p401
1 |- pci_pad_32 : p132
1 |- pci_pad_01 : p564
1 |- pci_pad_01 : p364
1 |- pci_pad_04 : p264
1 |- core4601   : c100

.--------------------------------
|- core4601 : c100
`--------------------------------
1 |- statem_pci   : b200
1 |- trackm_pci   : b300
1 |- trackm_pci   : b400
1 |- trackm_pci   : b500
1 |- statem_ini   : i100
1 |- statem_tgt   : t100
1 |- addr_gen     : a700
1 |- addr_gen     : a900
1 |- statem_par   : p100
1 |- statem_par   : p200
1 |- stage_tgt    : p800
1 |- stage_ini    : p900
1 |- confg_space  : c100
1 |- dp_a_rcv     : d700
1 |- dp_i_snd     : d700
1 |- dp_t_snd     : d700
```

The functional blocks in the core4601 model are comprised of control blocks, data path blocks and the configuration space block. The data path blocks are `dp_t_snd`, `dp_i_snd`, `dp_a_rcv`, `stage_ini` and `stage_tgt`. The flow control blocks are `statem_in`, `statem_tgt`, `statem_par` and `trackm_pci`. The address and instruction block is the `addr_gen` model. The configuration space and target selection block is the `confg_space` model. The `statem_pci` model supports the data, address, parity, configuration and control functional blocks.

Control Block Descriptions

The control blocks handle the interface handshaking between the application interface, the data path blocks and the PCI bus.

`statem_pci`

This model tracks the PCI address, data and control signals. These signals are provided to the rest of the PCI core models. There are 2 register ranks for the PCI data and byte enable signals, and 3 register ranks for the PCI control signals. Rank 0 is the signals as they appear on the PCI bus. Because of the limited setup before the clock for these signals, most of the functions will use the second rank of these signals, (Rank 1) in order

to avoid timing difficulties during synthesis. The signals on Rank 2 are clocked versions of Rank 1 signals.

In addition to these 3 ranks of PCI signals received from the PCI bus, the PCI address and command are extracted from the PCI transfers and are stored for use by other PCI functional block models. The address and command are stored until the next PCI frame is started and a new address and command is detected.

trackm_pci

This model tracks the PCI bus states as PCI transfers are in progress. This information is used to help the PCI initiator and target state machines drive the signals and control the flow of data between the PCI bus and the Application interface.

statem_ini

This model contains the initiator state machine. It generates the PCI signals, and signal enables when the core acts as a PCI initiator.

The initiator state machine will respond to transfer requests from the application bus by requesting the PCI bus, (using the PCI req / gnt handshake), and when the PCI bus is granted, it will drive the PCI signals necessary to accomplish the requested PCI transfer. It will return data and status of this transfer and indicate completion.

This model also controls the behavior of the PCI bus signals used by the initiator during PCI transfers. This includes the normal and abnormal PCI transfer terminations. It also controls the interaction and data flow between the PCI core and the application interface.

statem_tgt

The target state machine controls the target read and write PCI transfers and also the configuration transfers. It responds to PCI frames and determines if the PCI frame is requesting access which is within any of the valid PCI apertures indicated in the configuration register space.

statem_par

The parity state machine determines if parity was correct on the PCI transfer is which data and parity are received from the PCI bus, and generates parity when the VCM4601 PCI core is sending data to the PCI bus. This includes target reads, initiator writes, and the address phase of the PCI frame, (initiator only).

Configuration Space Model

confg_space

The configuration space is a set of 14 32 bit registers which control the behavior of the PCI core. They also indicate the status of the PCI core. These registers are defined in the PCI specification and in the Macrocad PCI core data sheet. This module also contains the selection mechanism for PCI access to the core when it is acting as a target. This includes both the normal 6 target memory and IO apertures for 32 bit applications, and 3 apertures for 64 bit applications.

Data Path Models

The remaining models in the core4601 comprise the data paths. The data path for the initiator and target, are divided into the send and receive data paths. The receive data path is used for initiator reads and target writes. The send data path is used for initiator writes and target reads.

Send Data Path

stage_ini

This model is a 1 rank register with a multiplexor on the input. This allows the initiator address, configuration data, and initiator 64 bit data or 32 bit data to be driven onto the PCI bus. It shortens the path required for timing sensitive signals such as trdy for initiator operations.

stage_tgt

This model is a 1 rank register with a multiplexor on the input. This allows configuration data, and target 64 bit data or 32 bit data to be driven onto the PCI bus. It shortens the path required for timing sensitive signals such as irdy for target operations.

dp_i_snd

This model provides the buffering needed for 66MHz operations. It results in fewer PCI bus stalls due to the buffer depth. The data from this buffer goes to the **stage_ini** model. The application interface can fill the buffer prior to acquiring the PCI bus for posted writes.

dp_t_snd

This model provides the buffering needed for 66MHz target operations. It results in fewer PCI bus stalls due to the buffer depth. The data from this buffer goes to the **stage_tgt** model. The application interface can fill the buffer prior to acquiring the PCI bus for deferred reads, (retries).

Receive Data Path

dp_a_rcv

This model buffers the data from the PCI to the application interface. There is no difference between the initiator read and target write behavior. The data is pipelined and is made available, only stalling the PCI bus when the application interface cannot accept data, and this buffer is full.

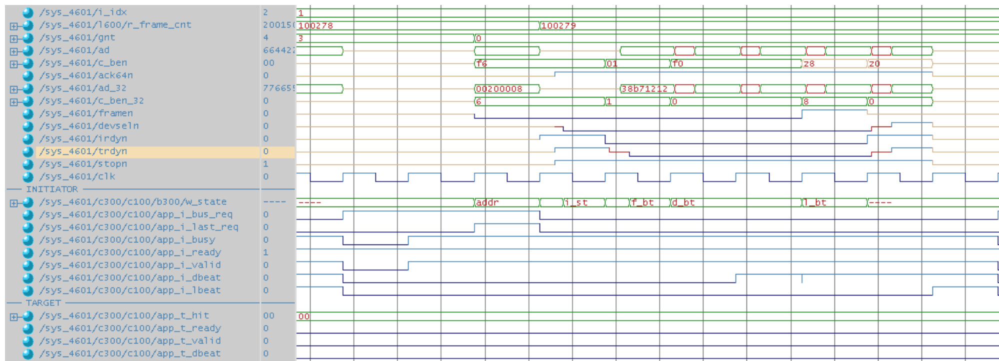
Address Models

addr_gen

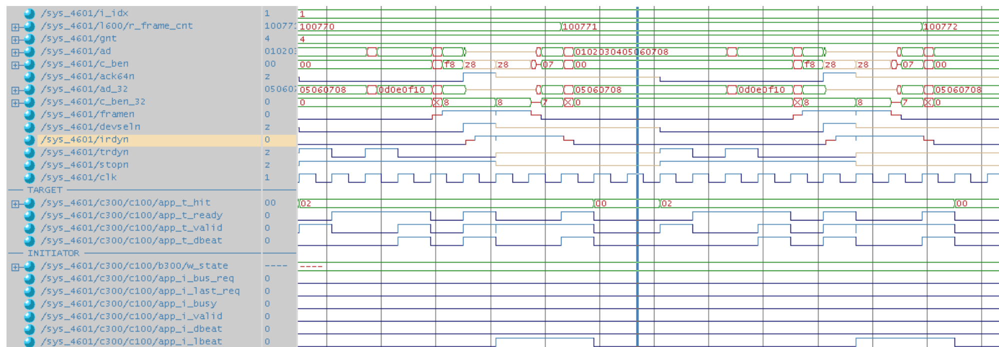
The address generator models will store the address and PCI command information for the current frame. This model is used for PCI initiator transfers and a separate instantiation is used for target operations. This is because the application interface and PCI frame “busy” states overlap, but do not start and end on the same clock.

APPLICATION INTERFACE

The PCI and applications operations will not be starting or ending on the same clock, but the skew may differ depending on the PCI response. In the case of an initiator operation, the application interface will be “busy before the PCI interface, where as in the case of the target transfers, the opposite is true. The application interface will be busy until the last data is transferred. The figure below show how a PCI read has the application interface “busy” before and after the PCI bus “busy”.



The figure blow shows the interface signals for a target operation.



The application interface is a handshake interface which has uni-directional signals for communication.

Initiator Interface

To transfer data as a PCI initiator, the application simply asserts the address, command byte enables and data values on the application bus, then raises the **app_bus_req** signal. The core will respond to the **app_bus_req** by asserting the **app_i_busy** signal when it has acquired the PCI bus. The application detects that the PCI core has stored this request when it detects the **app_i_valid** signal. The PCI core needs to know which application request is the last in a burst cycle, so the application must assert the **app_i_last_req** signal for the last burst request.

The PCI core will also indicate the PCI ``data beat'' activity on the PCI bus. The data byte enables and address will be presented to the application interface, and will be validated

by the signal app_i_dbeat (last data transfer in a PCI burst). The PCI core will indicate that it can not accept data by not responding with the app_i_valid signal.

Target Interface

The target handshake starts with the PCI core detecting a PCI frame where the PCI address falls within one of 6 memory or I/O apertures defined in the configuration space model, (3 apertures for 64 bit configured PCI cores). The app_t_hit signals indicate that there is a PCI frame which requires participation by the application, either by accepting PCI write data, or providing PCI read data.

The application responds to the app_t_hit signals by asserting app_t_valid when it can accept PCI write data, or when it has asserted PCI read data on the application interface.

PCI TIMING CONSIDERATIONS

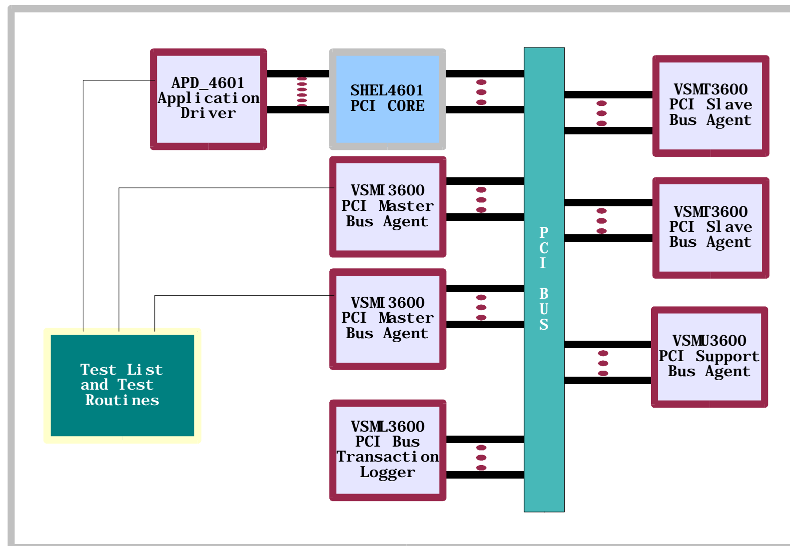
The PCI data bus does not get driven directly by a registered output. This is due to the timing constraints a 66MHz for a multiplexed bus like PCI. The PCI core has to switch between the address, initiator write data, target read data, and configuration read data. To accomplish this within the timing budget for PCI, the staging registers for the target and initiator data paths are multiplexed before being driven onto the PCI bus. This one level of gating should not cause timing constraint problems during synthesis. The staging register is loaded with the information to be driven on the PCI bus for the next clock cycle. This increases the latency (as indicated by **irdyn**) by one clock for initiator writes. The advantage is that the PCI core can shorten the path from **trdyn** to providing the next data to be driven on the PCI bus.

Check the timing constraint document, (critical_path.pdf), for more details.

TEST BENCH DESCRIPTION

SYS_4601 System Model

The VCM4601 PCI core comes complete with a comprehensive test bench. This includes the virtual PCI initiators, targets, PCI transfer logger, PCI support, (PCI clock and reset generation, timing checks, and arbitration), and tests needed for complete design verification. The SYS_4601 model is the system backbone for PCI based system simulation environments. All PCI devices, structural or behavioral “plug” in to this backbone. A variety of initiators and targets can be assembled on this system backbone, as well as user designs. All of these PCI design verification models support both 32 and 64 bit PCI cycles at either 33MHz or 66MHz.



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 chose from several arbitration schemes. There is an optional PCI bus activity display, (to Std Out).

Compile time constants:

```
p_pdly          // arbitrary propagation delay
p_seed_limit    // upper limit of seed for arb delay
p_timewidth     // width of time field in monitor statements
p_pagecount     // lines per page in monitor statements
p_dsp           // enables display task
p_arb_type      // 3 enables round robin arbitration
                // 2 enables highest bit position arbitration
                // 1 enables simple quasi random arbitration
                // 0 disables arbitration
```

VSML3600 Transaction Logger

This model logs all PCI bus transactions to a file. This includes burst, configuration, and special cycles. Bus ownership, parity, etc. are all 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 transaction, but monitors and logs all PCI activity.

Compile time constants:

```
p_log_file_name // log file name
p_log_enable    // log file enable switch
```

VSMT3600 Slave Model

This model acts as a PCI bus slave or target. It responds to all PCI defined transactions which fall within it memory, IO or configurations apertures. The response characteristics of this target device can be controlled on a frame by frame basis. Operational parameter information can be entered as constants at the system (SYS_4601) level. PCI transactions which this model participates in, can be logged to a file. Configuration information is based on the PCI spec v2.2.

Compile time constants:

```
p_dev            // device slot number
p_data_size      // data width 32, 64
p_log_enable     // slave transaction log ena
p_io_dword_range // select delay if byte range
p_cfg_file_name  // config initialization file
p_log_file_name  // transaction log file
p_data_type      // 1= 1M register mem array
                 // 2= internal dynamically allocated memory
                 // 3= external dynamically allocated memory
```

VSMI3600 Master Model

This model acts as a PCI bus master, or initiator. It will request the bus and, when granted, execute a PCI bus transaction in response to a transaction request which has been deposited in the instruction request cue. This model will execute bursts, and these can be modified to create stall, disconnect, abort, retry, bus error, and other defined conditions. These can be controlled on a frame by frame basis from the test routines. As with the slave, operational parameter information can be entered as constants at the system (SYS_4601) level. PCI transactions which this model participates in, can be logged to a file. Configuration information is based on the PCI spec v2.2.

Compile time constants:

```
p_debug_local   // local debug switches
p_log_file_name // log file name
p_cfg_file_name // config init file name
p_dev           // device slot number
p_log_enable    // log file enable switch
p_data_size     // data width, 32 or 64
```

APD_4601 Driver Model

This model is a driver for the SHEL4601 PCI core. It provides an interface between the application interface signals of the SHEL4601 PCI core, and the test routines which comprise the test bench instructions. It uses the same transaction interface that is used for the other design verification models.

Compile time constants:

```
p_dev           // NOT USED, but required
p_debug_local   // local debug switches
p_log_enable    // NOT USED, but required
p_log_file_name // log file name
p_data_size     // data width, 32 or 64
```

TEST INTERFACE

A transaction interface is used to facilitate communications between the test instruction list and the design verification models. The control and return fields are described below:

Request control field

NAME	***SIZE***	***FIELD***	***SIZE***	***DESCRIPTION***
Control 63:56	8	RSV7	8	RESERVED
Control 55:52	4	TAGS	4	Request tag
Control 51:48	4	TAGS	4	Response tag
Control 47:40	8	BRST	8	Burst length requested
Control 39:32	8	PATT	8	Data pattern In all cases, the first 64 bits are the data which is in the data field. The subsequent bits of data transferred are generated on the basis of the patterns defined below. : 0x00 : data = data : 0x01 : data = 0xffffffffffffffff : 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 : 31:28 : 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 : 15:9 : RESERVED : :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.

Return status field

NAME	SIZE	FIELD	SIZE	DESCRIPTION
Status 63:56	8	RSV7	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	BRST	8	Number of bytes transacted
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 : retry detected : :14 : disconnect 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

Control of specific behaviors for the design verification models is handled by using transactions to “sideband”registers. Transactions to these sideband registers do not result in any PCI activity. In most cases, these transactions are used to set up specific error or performance characteristics in the design verification models. These characteristics can include clock rates, parity error generation, PCI bus stalls, retry responses, etc. A list of sideband registers is shown below.

Sideband registers

NAME	ADDRESS PNEUMONIC	SIZE	FUNCTION	DESCRIPTION
r_optn	p_optn	64	R/w	Option register : 63:32 : RESERVED : 31:24 : RESERVED : 23:10 : RESERVED : 19:16 : Interrupt, send to PCI : :15 : RESERVED : 14:11 : data beat to disconnect : :10 : send target abort : :9 : RESERVED : :8 : send disconnect : 7:6 : RESERVED : 5:0 : Application aperture enable
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 16th data beat of burst : 59:56 : delay 15th data beat of burst : 55:52 : delay 14th data beat of burst : 51:48 : delay 13th data beat of burst : 47:44 : delay 12th data beat of burst : 43:40 : delay 11th data beat of burst : 39:36 : delay 10th data beat of burst : 35:32 : delay 9th data beat of burst : 31:28 : delay 8th data beat of burst : 27:24 : delay 7th data beat of burst : 23:20 : delay 6th data beat of burst : 19:16 : delay 5th data beat of burst : 15:12 : delay 4th data beat of burst : 11:8 : delay 3rd data beat of burst : 7:4 : delay 2nd data beat of burst : 3:0 : delay 1st data beat of burst
r_retry_int	p_retry_int	64	R/W	Retry interval : 63:32 : RESERVED : 31:0 : Interval between retries
r_retry_lmt	p_retry_lmt	64	R/W	Retry limit : 63:32 : RESERVED : 31:0 : Limit for retries
r_xact_req	p_frame_cnt	64	R/W	Transaction count : 63:32 : RESERVED : 31:0 : Transaction counter
r_debug_level	p_debug	64	R/W	Debug switches : 63:16 : RESERVED : 15:0 : Interval between retries

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 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.

Model Configuration and Operating Parameters

Operational parameter information can be entered as constants at the system (SYS_4601) level. These constants include the file names and paths for log files, etc. Each instantiated model which can participate in a PCI transaction, (VSMI3600 and VSMT3600), will have configuration information loaded from a file at the start of simulation time. This configuration information can also be loaded if specifically called as part of a test routine, or the registers 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.

Model parameters can be set to unique values for each instantiated model. These 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 only used to load an initial value, (at the start of simulation time), into the r_debug_level register. This 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.

Test Flow

The test is determined by the list of transaction instructions which is executed by the APD_4601 PCI core driver model, or the PCI bus master model, VSMI3600. Each instruction is deposited in the requested instruction cue in the appropriate model. The model interprets the request (which has a generic format), and translates the instruction into a PCI transaction, or a core interface transaction. The PCI core or master model manages all the PCI protocol handshaking, and timing constraints. They execute the PCI transfer, and at the completion, will respond by returning the results and status of the PCI transfer to the test routines. A typical instruction list in the test routines are shown below:

```
v_addr = 64'h0000000000204003;
v_darw = 64'h0011223344556677;
dev_4(v_addr, v_datw, v_datr, 8'h09, 8'h04, p_wr_mem);
```

In this case, the selected PCI device, (slot number 4) will execute a 9 byte memory write burst, with a data pattern, (select pattern number 4), at address 0x0000000000204003 with starting data 0x0011223344556677. The completion of the PCI burst will mean the last data written will be returned. Also note that the data is justified to the 0, or right most byte, (PCI is defined as little endian). The actual PCI burst will start at data byte lane 3, due to the starting address. So for the first burst cycle, that data in byte lane 3 will be 0x77. This memory write burst could be followed by a memory read burst which will check for the selected number of bytes at the selected address with the selected pattern. Data error checking for PCI reads can be controlled, turned on, off or stop on error can also be invoked. In this way it is easy to set up loops for changing address, burst size, and other parameters to check for corner conditions.

PCI Transfer Logging

PCI transfers are logged by each device, and the PCI bus logger model. Below a series of PCI transfers is shown initiated by device 4, and targeting devices 5 and 2.

Below the bus log for device 4 is shown.

```
 200388       4      1 Mem Write......  0000000000500008 554433221100----    W
 200388       4      2 Mem Write......  0000000000500010 aa88664422007766    W
 200388       4      3 Mem Write......  0000000000500018 ------884401eecc    W
 200389       4      1 Mem Read.......  0000000000500008 554433221100----    W
 200389       4      2 Mem Read.......  0000000000500010 aa88664422007766    W
 200389       4      3 Mem Read.......  0000000000500018 ------884401eecc    W
 200390       4      1 Mem Write......  0000000000200000 ------------10--
 200391       4      1 Mem Read.......  0000000000200000 ------------10--
 200392       4      1 Mem Write......  0000000000200000 ----------2010--
 200393       4      1 Mem Read.......  0000000000200000 ----------2010--
 200394       4      1 Mem Write......  0000000000200000 --------302010--
 200395       4      1 Mem Read.......  0000000000200000 --------302010--
 200396       4      1 Mem Write......  0000000000200000 --------302010--
 200396       4      2 Mem Write......  0000000000200000 ------40--------
 200397       4      1 Mem Read.......  0000000000200000 --------302010--
 200397       4      2 Mem Read.......  0000000000200000 ------40--------
```

Below the bus log for device 5 is shown.

```
 200388       4      1 Mem Write......  0000000000500008 554433221100----    W
 200388       4      2 Mem Write......  0000000000500010 aa88664422007766    W
 200388       4      3 Mem Write......  0000000000500018 ------884401eecc    W
 200389       4      1 Mem Read.......  0000000000500008 554433221100----    W
 200389       4      2 Mem Read.......  0000000000500010 aa88664422007766    W
 200389       4      3 Mem Read.......  0000000000500018 ------884401eecc    W
      0       5      1 Write SBR......  0000000000000010 00000000000311bd
      0       5      1 Write SBR......  0000000000000010 0000000000049741
      0       5      1 Write SBR......  0000000000000010 0000000000061a89
      0       5      1 Write SBR......  0000000000000010 0000000000064195
FRAME   MASTER BURST COMMAND            ADDRESS          DATA                FLAGS
------- ------ ----- --------------     ---------------- ----------------   ---------
 420001       4      1 Config Read....  0000000000020018 --------00550001     D
 420002       4      1 Config Write...  0000000000020018 --------ffffffff     D
 420003       4      1 Config Read....  0000000000020018 --------ffff0003     D
 420004       4      1 Config Write...  0000000000020018 --------00550001     D
 420005       4      1 Config Write...  0000000000020030 00000000--------     D
```

Below the bus log for device 2 is shown.

```
      0       2      1 Write SBR......  0000000000000010 00000000000189a5
 200390       4      1 Mem Write......  0000000000200000 ------------10--
 200391       4      1 Mem Read.......  0000000000200000 ------------10--
 200392       4      1 Mem Write......  0000000000200000 ----------2010--
 200393       4      1 Mem Read.......  0000000000200000 ----------2010--
 200394       4      1 Mem Write......  0000000000200000 --------302010--
 200395       4      1 Mem Read.......  0000000000200000 --------302010--
 200396       4      1 Mem Write......  0000000000200000 --------302010--
 200396       4      2 Mem Write......  0000000000200000 ------40--------
 200397       4      1 Mem Read.......  0000000000200000 --------302010--
 200397       4      2 Mem Read.......  0000000000200000 ------40--------
```

Application Transaction Logging

In the example below, the PCI core is accessed as a target. There is no frame count, since the PCI code's driver does not have visibility on the PCI bus.

Below the transaction log for device 3 is shown:

```
TGT-        0       0       1 Mem Write......  0000000000300008 605040302010----    W
TGT-        0       0       1 Mem Write......  0000000000300010 c0a0806040218070    W
TGT-        0       0       1 Mem Write......  0000000000300018 ----------4200e0    W
TGT-        0       0       1 Mem Read.......  0000000000300008 605040302010----    W
TGT-        0       0       1 Mem Read.......  0000000000300010 c0a0806040218070    W
TGT-        0       0       1 Mem Read.......  0000000000300018 ----------4200e0    W
```