Versal ACAP CIPS Verification IP

DS996 (v1.0) November 16, 2022 **Product Specification**

Summary

The Versal® adaptive compute acceleration platforms (ACAP) Control, Interfaces, and Processing System (CIPS) Verification Intellectual Property (VIP) supports the functional simulation of Versal ACAP applications. It is targeted to enable the functional verification of programmable logic (PL) by mimicking the processor system (PS)-PL interfaces and OCM memories of PS logic. This VIP is delivered as a package of System Verilog modules. VIP operation is controlled by using a sequence of System Verilog tasks.

Features

- Narrow transfers are supported for 32/64-bit transfers.
- 8/16/32-bit width registers are supported.
- ACE interface supports only AXI4 traffic, all sideband signals are ignored.
- ACP interface supports limited AXI4 features.
  - For ACP, AxSIZE is tied to 4 (128-bit).
  - AxBURST = 1 (only INCR is supported).
- PL clock/PL reset/PS-PL/PL-PS interrupts supported in API.
- On CCI ports,
  - No coherency supported.
  - Support for striping across the four CCI interfaces on 1 KB, 2 KB, and 4 KB boundaries.
  - Only one CCI port can be accessed at a time.
  - Bus split is not supported.

  For example, CCI boundary configured for 1K,

  AWADDR: 3FC, AWLEN:1 (2 beats), AWSIZE: 2 (32-bit), AWBURST: INCR.

  This transaction accesses addresses from 3FC to 3FC + (2 × 4) that is, 3FC to 404. The above transaction crosses the 1K boundary and it cannot be used by the CCI decoding logic/DeMUX. The current Versal ACAP CIPS VIP can take care of the initiated transactions which are not crossing CCI boundaries.

  This means:

  - Transaction-1: AWADDR: 3FC, AWLEN:0 (1 beat), AWSIZE: 2 (32-bit), AWBURST: INCR

- Transaction-2: AWADDR: 400, AWLEN:0 (1 beat), AWSIZE: 2 (32-bit), AWBURST: INCR

TIP: *If there is a HANG in simulation,*

- Check if routing configuration is provided before the start of traffic.
- When a different master tries to access the same slave, ensure to disable the earlier routing configuration and enable the new routing configuration.
- Check VIP and AXI interfaces are connected to the single clock source.

IP Facts

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal ACAP
Supported User Interfaces	AXI4
Resources	Not Provided
Provided with Core	
Design Files	Not Provided
Example Design	System Verilog
Test Bench	N/A
Constraints File	N/A
Simulation Model	System Verilog
Supported S/W Driver[2]	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 75889
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.
3. To take advantage of all the features of this IP, you need simulators that support advanced simulation capabilities.
4. To use the virtual part of the AXI Verification IP, the IP must be in a Verilog hierarchy.

Limitations

Address decoding is not modeled inside the Versal ACAP CIPS VIP. Therefore, routing configuration needs to be provided through API before initiating the traffic.

The entire Versal ACAP CIPS VIP works in a single clock domain (that is, all the slave and masters of Versal ACAP CIPS VIP work on the same frequency).

All of the Versal ACAP CIPS VIP AXI4 interface widths are static (that is, Address and Data widths cannot be changed through the CIPS GUI). If you attempt to change the widths, the behavior is undefined.

The following features are not supported:

- Versal ACAP CIPS VIP does not handle any configuration information passed through the GUI, except enable and disable of PL AXI and NoC interfaces.
- Versal ACAP CIPS VIP does not model any AXI4 path delays.
- No AXI4 ID support.
- CPM5 is not supported.
- QoS (AxQOS) is not supported.
- Cache coherency is not supported.
- Exclusive Access (AxLOCK) is not supported.
- Bufferability (AxCACHE) is not supported.
- AxREGION is not supported.
- Latency Modeling for OCM/REG is not supported.
- ACELITE/NPI are not modeled.
- Interconnect arbitration is not modeled as per the RTL.
- Outstanding transactions are not supported for AXI4 interfaces.
- Low power modeling is not supported.
- AXI4 user signals cannot be exercised. If exercised, the behavior is undefined.

Functional Description

The Versal ACAP CIPS VIP is used to check the Versal ACAP CIPS-based BD design integrity to initiate AXI traffic on NoC and PL AXI ports.

The clock and reset needs to be provided to the Versal ACAP CIPS VIP. The `versal_cips_ps_vip_clk` is a VIP clock and it needs to be driven by a valid clock source (the same clock that is driven to the AXI4 interfaces). Reset to the Versal ACAP CIPS VIP is active-Low.

The following shows the example code for driving clock and reset:

```
force tb_top.DUT.design_1_i.versal_cips_0.inst.PS9_VIP_inst.inst.versal_cips_ps_vip_clk
= clock_source;
tb_top.DUT.design_1_i.versal_cips_0.inst.PS9_VIP_inst.inst.por_reset(0);
repeat(20)@(posedge clock_source);
tb_top.DUT.design_1_i.versal_cips_0.inst.PS9_VIP_inst.inst.por_reset(1);
```

The Versal ACAP CIPS VIP has two internal AXI masters named R5_API and A72_API. These AXI4 masters are exercised through the APIs in the Application Programming Interfaces.

The Versal ACAP CIPS VIP models an OCM memory with supporting address range (`0xFFFC_0000-0xFFFF_FFFF`).

The Versal ACAP CIPS VIP implements the register space.

- Functional aspects of these registers are not modeled.
- Default values of the registers can be read.
- Error response is not modeled for register space.

If AXI SLVERR response for a particular PS register address range is expected, Versal ACAP CIPS VIP returns the SLVERR response.

Note: By default, Versal ACAP CIPS VIP is always enabled. To disable, set `cips.enablePSVIPsimulation` parameter to 0 using `set_param` command.

```
set_param cips.enablePSVIPsimulation 0
```

For more information on using `set_param`, refer *Vivado Design Suite Tcl Command Reference Guide* (UG835).

TIP: *Versal ACAP CIPS VIP does not support post-synthesis simulation.*

The following table shows the available ports to debug in the Versal ACAP CIPS VIP.

Table 1: **Debug Port Names**

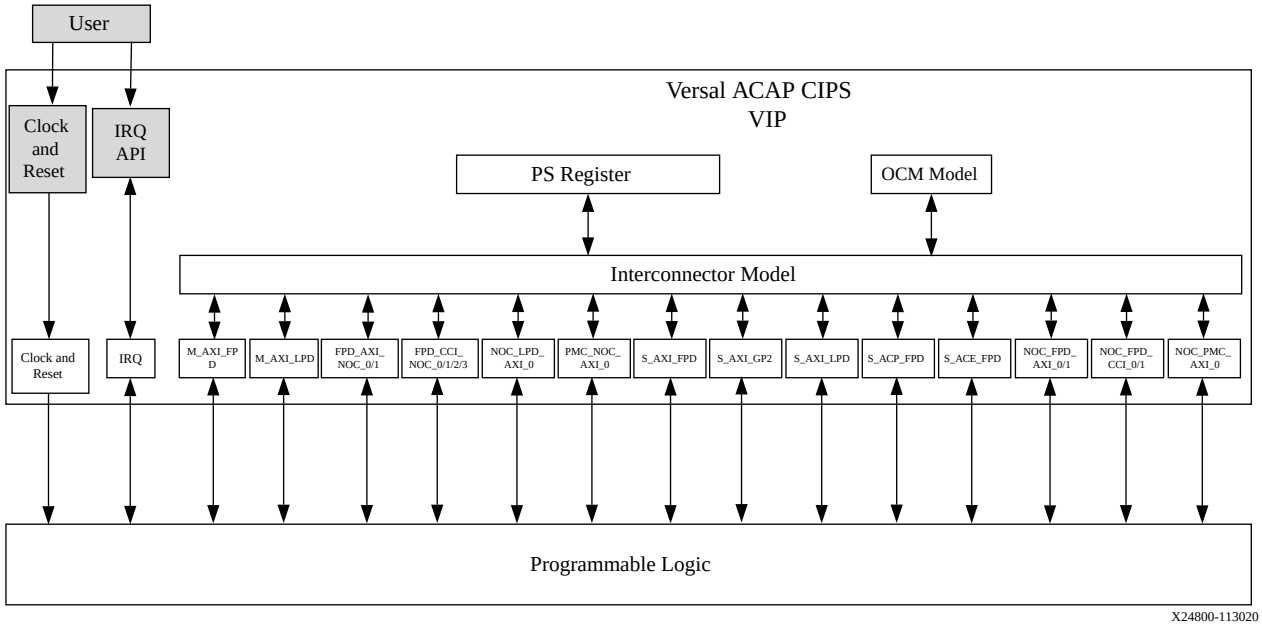
GUI Name	Actual VIP Port Name
M_AXI_FPD	MAXIGP0
M_AXI_LPD	MAXIGP2
N/A	OCM
N/A	IFPSCPMCFG
FPD_AXI_NOC_0	IFPSNOCNCIAXI0
FPD_AXI_NOC_1	IFPSNOCNCIAXI1
N/A	IFPSNOCPCIAXI0
N/A	IFPSNOCPCIAXI1
FPD_CCI_NOC_0	IFPSNOCCCIAXI0
FPD_CCI_NOC_1	IFPSNOCCCIAXI1
FPD_CCI_NOC_2	IFPSNOCCCIAXI2
FPD_CCI_NOC_3	IFPSNOCCCIAXI3
NOC_LPD_AXI_0	IFPSNOCRPUAXI0
PMC_NOC_AXI_0	IFPMCNOCAXI0
N/A	IFPSCPMPCIE
S_AXI_FPD	SAXIGP0
S_AXI_GP2	SAXIGP2
S_AXI_LPD	SAXIGP4
S_ACP_FPD	SAXIACP
S_ACE_FPD	SACEFPD
NOC_FPD_AXI_0	IFNOCPSNCIAXI0
NOC_FPD_AXI_1	IFNOCPSNCIAXI1
NOC_FPD_CCI_0	IFNOCPSCCIAXI0
NOC_FPD_CCI_1	IFNOCPSCCIAXI1

Table 1: **Debug Port Names** *(cont'd)*

GUI Name	Actual VIP Port Name
NOC_PMC_AXI_0	IFNOCPMCAXI0
N/A	IFNOCPSPCIAXI0
N/A	IFCPMPSAXI0
N/A	IFCPMPSAXI1

The following figure shows the detailed architecture for the VIP logic.

Figure 1: **Versal ACAP CIPS VIP Architecture**



Application Programming Interfaces

The following tables show the APIs supported by the Versal ACAP CIPS VIP.

Data

APIs	Inputs	Outputs

`write_data`
Initiate a WRITE transaction (transfer size ≤ 16 bytes) on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns write response when the write transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_data("R5_API",44'hFFFC_0000, 4, data,response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Write address
	[4:0] wr_size: Number of bytes to be written.
	Valid Write Sizes are:
	• 0x4: 4 bytes
	• 0x8: 8 bytes
	• 0xC: 12 bytes
	• 0x10: 16 bytes
	Note: Transfer should not cross the 16-byte boundary.
	Note: start_addr should be 32-bit aligned.
	If start_addr[1:0]
	= 4'h0 -> wr_size can be 5'h4/5'h8/5'hC/5'h10
	= 4'h4 -> wr_size can be 5'h4/5'h8/5'hC
	= 4'h8 -> wr_size can be 5'h4/5'h8
	= 4'hC -> wr_size can be 5'h4
	[127:0] w_data: Data to be written
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY,
		SLVERR, DECERR]

`read_data`
Initiate a READ transaction (transfer size ≤ 16 bytes) on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns data and response when the read transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_data("R5_API",44'hFFFC_0000, 4, data, response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Read address
	[4:0] rd_size: Number of bytes to be read
	Valid Read Sizes are
	• 0x4: 4 bytes
	• 0x8: 8 bytes
	• 0xC: 12 bytes
	• 0x10: 16 bytes
	Note: Transfer should not cross the 16-byte boundary.
	Note: start_addr should be 32-bit aligned.
	If start_addr[1:0]
	= 4'h0 -> rd_size can be 5'h4/5'h8/5'hC/5'h10
	= 4'h4 -> rd_size can be 5'h4/5'h8/5'hC
	= 4'h8 -> rd_size can be 5'h4/5'h8
	= 4'hC -> rd_size can be 5'h4
		[127:0] rd_data: Valid Data from the slave
		[1:0] response: The slave read response from the following:
		[OKAY, EXOKAY,
		SLVERR, DECERR]

`write_data_32`
Initiate a WRITE transaction of 4 bytes on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns write response when the write transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_data_32("R5_API",44'hFFFC_0000, data,response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: write address
	[31:0] w_data: Data to be written
	Note: start_addr should be 32-bit aligned.
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY,
		SLVERR, DECERR]

APIs	Inputs	Outputs

`write_data_64`
Initiate a WRITE transaction of 8 bytes on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns write response when the write transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_data_64("R5_API",44'hFFFC_0000, data,response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: write address
	[63:0] w_data: Data to be written
	***Note*:** start_addr should be 64-bit aligned.
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY, SLVERR, DECERR]

`read_data_32`
Initiate a READ transaction of 4 bytes on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns data and response when the read transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_data_32("R5_API",44'hFFFC_0000, data, response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Read address
	***Note*:** start_addr should be 32-bit aligned.
		[31:0] rd_data: Valid Data from the slave[1:0] response: The slave read response from the following:
		[OKAY, EXOKAY, SLVERR, DECERR]

`read_data_64`
Initiate a READ transaction of 8 bytes on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns data and response when the read transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_data_64("R5_API",44'hFFFC_0000, data, response);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Read address
	***Note*:** start_addr should be 64-bit aligned.
		[63:0] rd_data: Valid Data from the slave[1:0] response: The slave read response from the following:
		[OKAY, EXOKAY, SLVERR, DECERR]

Burst

APIs	Inputs	Outputs

`write_burst`
Initiate a single AXI write burst transaction (transfer size ≤ 256 bytes) on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns write response when the write transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_burst("AFI_API",44'hA000_0000,0, 2, 1, 0, 0, 0, wr_m_data,4 , wrs);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Write address
	len: Burst Length
	siz: Burst Size
	burst: Burst Type
	lck: Lock Type
	0x0: No lock support
	cache: Cache Type
	0x0: No cache support
	prot: Protection Type
	0x0: No protection support
	[32767:0] data: Data to be sent
	datasize: Number of bytes (max 256 bytes)
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY, SLVERR, DECERR]

`read_burst`
Initiate a single AXI read burst transaction (transfer size ≤ 256 bytes) on one of the AXI-Master ports.
This calls the AXI VIP API. This task returns data and response when the read transaction is completed.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_burst("AFI_API",44'hA000_0000,0,2,1,0,0,0,rd_m_data,rresp);`
	[1023:0] master_name: R5_API or A72_API.
	[43:0] start_addr: Read address
	len: Burst Length
	siz: Burst Size
	burst: Burst Type
	lck: Lock Type
	0x0: No lock support
	cache: Cache Type
	0x0: No cache support
	prot: Protection Type
	0x0: No protection support
		[32767:0] rd_data: Valid Data from the slave[1:0] response: The slave Read response from the following:
		[OKAY, EXOKAY, SLVERR, DECERR]

File

APIs	Inputs	Outputs

`write_from_file`
Initiate an AXI write transaction on one of the AXI-Master ports. The write data is used from the file. Burst type used is INCR. This is a blocking task and returns only after the completion of AXI WRITE transaction. Need to pass FILE_DATA_WIDTH_128 for accessing files with 128-bit data contents.
If FILE_DATA_WIDTH_128 is passed, then the address must be 128-bit aligned.
If FILE_DATA_WIDTH_128 is not passed, then the address must be 32-bit aligned.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_from_file("R5_API","write_mgp.txt",40'h8000_0000,4,wrs)`
	[1023:0] master_name: R5_API or A72_API.
	[1023:0] File_name: File name that stores the write data.
	[43:0] start_addr: write address
	[31:0] wr_size: number of bytes to be written
	For 32-bit file contents, wr_size should be in multiples of 4.
	For 128-bit file contents, wr_size should be in multiples of 16.
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY,
		SLVERR, DECERR]

`read_to_file`
Initiate an AXI read transaction on one of the AXI-Master ports. The read data is written to a file. Burst type used is INCR. This is a blocking task and returns only after the completion of AXI READ transaction. Need to pass FILE_DATA_WIDTH_128 for accessing files with 128-bit data contents.
If FILE_DATA_WIDTH_128 is passed, then the address must be 128-bit aligned.
If FILE_DATA_WIDTH_128 is not passed, then the address must be 32-bit aligned.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_to_file("R5_API","read_mgp0.txt",40'h8000_0000,4,rresp)`
	[1023:0] master_name: R5_API or A72_API.
	[1023:0] file_name: File name that stores the write data.
	[43:0] start_addr: write address
	[31:0] rd_size: number of bytes to be read.
	For 32-bit file contents, rd_size should be in multiples of 4.
	For 128-bit file contents, rd_size should be in multiples of 16.
		[1:0] response: The slave write response from the following:
		[OKAY, EXOKAY,
		SLVERR, DECERR]

Preload

APIs	Inputs	Outputs

`pre_load_mem`
Preload OCM with random_data/all_zeros/all_ones. Based on the address and data type specified, the data is loaded in OCM. Address must be 32-bit aligned.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.pre_load_mem(2'b10,44'hFFFF_0000,4);`
	[1:0] data_type: Random, zeros or ones.
	• 00/11 – Random
	• 01 – Zeros
	• 10 – Ones
	[43:0] start_addr: Start Address from where OCM should be initialized with the data
	[31:0] no_of_bytes: Number of data bytes to be loaded
		None

`pre_load_mem_from_file`
Preload OCM with data from a file. Based on the address specified, the data is loaded in OCM.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.pre_load_mem_from_file("preload_ocm.txt",addr1,no_bytes_transfer_ocm);`
	[1023:0] file_name: File name (max. 256 characters)
	[43:0] start_addr: Start Address from where OCM should be initialized with data from the file.
	[31:0] Num lines: Number of lines (each row should match 32 bits)
	Note: start_addr should be 32-bit aligned.
		None

Memory

APIs	Inputs	Outputs

`peek_mem_to_file`
This reads from the OCM and prints to a file.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.peek_mem_to_file("read_ocm_test_sanity.txt"},addr1,no_of_bytes);`
	[1023:0] file_name: File name
	(max. 128 characters; Read data is written to this file).
	[43:0] start_addr: Start Address to read the data from.
	[31:0] no_of_bytes: Number of data bytes to be read.
		None

`write_mem`
Back door write to the OCM memory. Based on the address specified, the data is written to OCM. Address must be 32-bit aligned.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_mem(data,40'hFFFF_2000, no_of_bytes);`
	[8191:0] data: Write data to be written to memory.
	[43:0] start_addr: Start Address to write the data from.
	[15:0] no_of_bytes: Number of data bytes to be write (max. 128 bytes).
		None

`read_mem`
Back door read from the OCM memory. Based on the address specified, the data is read from OCM.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_mem(40'hFFFF_4000, 128, rd_m_data);`
	[43:0] start_addr: Start Address to read the data from.
	[15:0] no_of_bytes: Number of data bytes to be read.
		[8191:0] data: read data from memory.

Clock

APIs	Inputs	Outputs
`pl_gen_clock` This API generates the PS to PL clocks with the frequency in MHz as the input along with the clock port number. For example: `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.pl_gen_clock(0,100)` → Generates PMCRCLKCLK[0] with 100 MHz frequency	[1:0] clk_num: • 2'b00 → PMCRCLKCLK[0] • 2'b01 → PMCRCLKCLK[1] • 2'b10 → PMCRCLKCLK[2] • 2'b11 → PMCRCLKCLK[3] [31:0] freq_in_mhz: This can be any integer value	None
`cpm_osc_clk_div2_gen_clock` This API generates clock on CPMOSCCLKDIV2 port of CPM, and the frequency is in MHz which is passed as input . For example: `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.cpm_osc_clk_div2_gen_clock(100)` → Generates 100 MHz frequency on CPMOSCCLKDIV2 port	[31:0] freq_in_mhz: This can be any integer value	None
`cpm_gen_clock` This API generates clock on LPDCPMINREFCLK port of CPM and the frequency is in MHz which is passed as input . For example: `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.cpm_gen_clock(100)` → Generates 100 MHz frequency on LPDCPMINREFCLK port	[31:0] freq_in_mhz: This can be any integer value	None

Reset

APIs	Inputs	Outputs
`por_reset` Soft reset for VIP. It is a mandatory reset. It is Active-Low reset. For example: `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.por_reset(1)` → Deasserts the BFM's reset `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.por_reset(0)` → Asserts the BFM's reset	por_reset_ctrl: 1-bit input reset to VIP. por_reset_ctrl is Active-Low input.	None
`pl_gen_reset` This API will assert the PS to PL resets. For example: `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.pl_gen_reset(4'hE)` → Deasserts the PLRST1N, PLRST2N, and PLRST3N. → Asserts the PLRST0N	[3:0] reset_ctrl Depending on the reset_ctrl value, one of the reset is asserted. reset_ctrl is Active-Low input. For example: • reset_ctrl: 4'hE, PLRST0N is asserted • reset_ctrl: 4'hD, PLRST1N is asserted • reset_ctrl: 4'hB, PLRST2N is asserted • reset_ctrl: 4'h7, PLRST3N is asserted • reset_ctrl: 4'h3, PLRST2N, PLRST3N are asserted. Any combination of above is also possible.	None

Interrupt

APIs	Inputs	Outputs

`read_interrupt`
Use this API to poll the interrupt status at any given time. This is a non-blocking task to poll the interrupt status and returns immediately with current state on interrupt lines.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_interrupt()`
It returns the interrupt status (16-bit) for {PLPSIRQ1,PLPSIRQ0}.
	None	[15:0] irq_status:
		Interrupts generated by PL.

`wait_interrupt`
Use this API to wait for any of the interrupt to occur. This is a blocking task to wait for an interrupt to occur and returns immediately with any of the four interrupt lines asserts.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.wait_interrupt(4)`
• Above call waits for an interrupt on PLPSIRQ0[2]
• It also returns the interrupt status (16-bit) for {PLPSIRQ1,PLPSIRQ0}.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.wait_interrupt(14)`
• Above call waits for an interrupt on PLPSIRQ1[6]
• It also returns the interrupt status (16-bit) for {PLPSIRQ1,PLPSIRQ0}.
	[3:0] irq: Interrupt line number
	irq: 0 → PLPSIRQ0[0],
	1 → PLPSIRQ0[1],
	2 → PLPSIRQ0[2],
	3 → PLPSIRQ0[3],
	4 → PLPSIRQ0[4],
	5 → PLPSIRQ0[5],
	6 → PLPSIRQ0[6],
	7 → PLPSIRQ0[7],
	8 → PLPSIRQ1[0],
	9 → PLPSIRQ1[1],
	10 → PLPSIRQ1[2],
	11 → PLPSIRQ1[3],
	12 → PLPSIRQ0[4],
	13 → PLPSIRQ0[5],
	14 → PLPSIRQ1[6],
	15 → PLPSIRQ1[7].
		[15:0] irq_status:
		Interrupts generated by PL.
		irq_status = {PLPSIRQ1[7:0] ,PLPSIRQ0[7:0]}.

APIs	Inputs	Outputs

`trigger_interrupt`
Use this API to trigger an interrupt pin (IRQFPD/IRQLDP/IRQPMC) of the PL.
This is a non-blocking task for triggering the interrupts.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.trigger_interrupt(1,10)`
	[1:0] intr_type: To specify either IRQFPD/TRQLPD/IRQPMC
	• 01: IRQFPD
	• 10: IRQLPD
	• 11: IRQPMC
	[7:0] interrupt_no: Interrupt pin number/index of the particular index type.
	For example: If, intr_type: 10, interrupt_no: 32.
	Then, IRQLPD[32] will be triggered.
		None

`clear_interrupt`
Use this API to clear an interrupt pin (IRQFPD/IRQLDP/IRQPMC) of the PL.
This is a non-blocking task for clearing the interrupts.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.clear_interrupt(1,10)`
	[1:0] intr_type: To specify either IRQFPD/TRQLPD/IRQPMC
	• 01: IRQFPD
	• 10: IRQLPD
	• 11: IRQPMC
	[7:0] interrupt_no: Interrupt pin number/index of the particular index type.
	For example: If, intr_type: 10, interrupt_no: 32.
	Then, IRQLPD[32] will be cleared.
		None

Configuration

APIs	Inputs	Outputs

`set_routing_config`
This API is used to provide the routing information inside the Versal ACAP CIPS VIP. It also prints the entire routing information configured.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.set_routing_config("R5_API","OCM",1)`
• Above API call enables the routing from R5_API to OCM
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.set_routing_config("R5_API","OCM",0)`
• Above API call disables the routing from R5_API to OCM
If two masters are trying to access the same slave, then traffic should be handled sequentially.
For example:
`R5_API and A72_API are trying to access OCM.`
`// Enable routing from R5_API to OCM`
`-`
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.set_routing_config("R5_API","OCM",1)`
`// initiate the traffic from R5_API`
`…….`
`// Disable the routing from R5_API to OCM`
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.set_routing_config("R5_API","OCM",0)`
`// Enable routing from A72_API to OCM`
`-`
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.set_routing_config("A72_API","OCM",1)`
`// initiate the traffic from A72_API`
`……`
	[8191:0] source_port_name:
	Incoming ports of Versal ACAP CIPS VIP
	Supported Source Port Names:
	• S_AXI_FPD
	• S_AXI_GP2
	• S_AXI_LPD
	• NOC_FPD_CCI_0
	• NOC_FPD_CCI_1
	• NOC_FPD_AXI_0
	• NOC_FPD_AXI_1
	• S_ACP_FPD
	• S_ACE_FPD
	• NOC_PMC_AXI_0
	• NOCPSPCIAXI0
	• A72_API
	• NOC_API
	• R5_API
	• CPMPSAXI0
	• CPMPSAXI1
	[8191:0] destination_port_name:
	Outgoing ports from the PS CIPS VIP
	Supported Destination Port Names:
	• FPD_CCI_NOC
	• M_AXI_FPD
	• M_AXI_LPD
	• FPD_AXI_NOC_0
	• FPD_AXI_NOC_1
	• PSNOCPCIAXI0
	• PSNOCPCIAXI1
	• PMC_NOC_AXI_0
	• NOC_LPD_AXI_0
	• PSCPMPCIEAXI
	• OCM
	• REG
	• PSCPMCFG
	routing_en:
	• 1: Enables routing for the specified path
	• 0: Disables routing for the specified path
		None

`get_routing_config`
This API call is used to print the configured routing information.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.get_routing_config();`
	None	None

APIs	Inputs	Outputs

`cfg_slverr_address_range`
This API call sets an address range for which SLVERR response is expected.
This API can be called multiple times for multiple such address ranges. It also prints all the configured SLVERR address ranges.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.cfg_slverr_address_range('hFFFC_0204,'hFFFC_0204,1);`
	[63:0] slverr_start_address
	[63:0] slverr_end_address
	set_slv_error
	• 0: SLVERR for the programmed range is disabled
	• 1: SLVERR for the programmed range is enabled
		None

Register

APIs	Inputs	Outputs

`write_register`
This API is for performing backdoor register write.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.write_register(32'gFFA80000,32'hAAAAAAAA,4);`
	[31:0] addr: Register address
	[31:0] data: Data to be written to the register
	int size: Number of bytes to be written. Possible values are 1, 2, 3, and 4 only.
		None

`read_register`
This API call is for reading a 32-bit register value.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.read_register(32'hFFA80000)`
	[31:0] addr: Register address	[31:0] DATA: Valid register read data

CCI Boundary

APIs	Inputs	Outputs

`select_cci_boundary`
This API can be used to select the CCI address boundary to be either 4K/2K/1K.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.select_cci_boundary(0)`
	[1:0] cci_boundary
	• 2'b00: 4K
	• 2'b01: 2K
	• 2'b10: 1K
	• 2'b11: Undefined
	Default is 4K.
		None

Warning

APIs	Inputs	Outputs

`fatal_to_warning`
This API is used to convert the FATAL messages reported for the components A72_API, R5_API, OCM, and REG space.
For example:
`versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst.fatal_to_warning("R5_API")`
	[1023:0] component_name
	Permitted Component Names are:
	• A72_API
	• R5_API
	• OCM
	• REG
		None

Example Design

An example design demonstrating the usage of the CIPS VIP is available on the Xilinx CEDStore. It can be accessed either from the Vivado**Start** Menu under **Example Designs** or downloaded from Github.

References

These documents provide supplemental material useful with this guide:

1. *Versal ACAP Register Reference* (AM012)
2. *AXI Verification IP LogiCORE IP Product Guide* (PG267)
3. *AXI Interconnect LogiCORE IP Product Guide* (PG059)
4. *Control, Interface and Processing System LogiCORE IP Product Guide* (PG352)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *Vivado Design Suite User Guide: Designing with IP* (UG896)
7. *Vivado Design Suite User Guide: Getting Started* (UG910)
8. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide: Implementation* (UG904)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
11/16/2022 Version 1.0	
Functional Description	Added details to disable Versal ACAP CIPS VIP.
Data	• Updated hierarchy in `read_data`. • Updated `start_addr` configuration for `write_data_64` and `read_data_64`.
Application Programming Interfaces	Updated hierarchy from `versal_cips_0.inst.PS9_VIP_inst.inst` to `versal_cips_0.inst.pspmc_0.inst.PS9_VIP_inst.inst`
07/14/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright