MEST Module 3- Introduction to SoC Modeling with Platform Architect

Muhammed Kawser Ahmed, Abigail Butka, Christophe Bobda

March 30, 2024

1 Objective

The learning objective of this module is to gain hands-on experience in System-on-Chip modeling with Platform Architect along with peripheral IP integration, AXI and APB bus protocols, and system-level debugging. In this module you will use the Synopsys Platform Architect software to build your own SoC. Synopsys Platform Architect is a SystemC TLM standards-based graphical environment for capturing, configuring, simulating, and analyzing the system-level performance and power of multicore systems and next-generation SoC architectures. The Figure 1 shows the modeling flow of abstracting hardware-software design. The imported SystemC, IP modules, and HDL blocks can be combined to create a system.

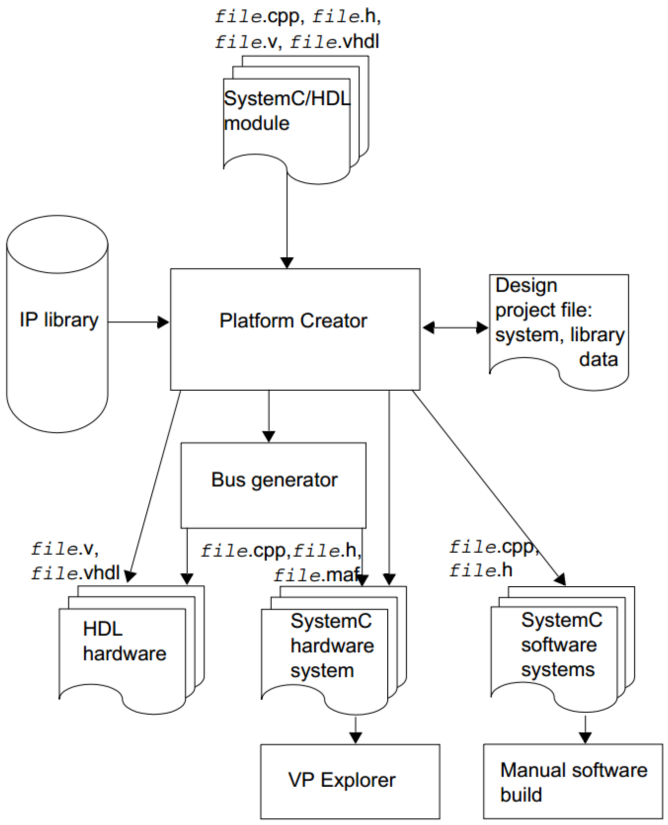


Figure 1: Flow of SoC/ASIC/MPSoC Modeling Using Platform Architect

2 Equipment and Software Needed for this Module

1. Synopsys Platform Architect S - 2021.09.
2. Synopsys VCS compiler.
3. Linux OS

3 Module Description

3.1 Configuring Platform Architect

In this task, we will configure the script file needed for launching the Platform Architect.

TCL Support Platform Architect has a support of TCL interfacing that can be evoked both from GUI or batch mode. All the commands that is performed in GUI can also be performed using TCL commands. At start-up, Platform Creator reads in the following Tcl files in the following order:

- All file.tcl files in $COWAREPCPLUGIN
- $HOME/.pcshrc
- ./.pcshrc

User commands can be defined in either of these locations. When using the GUI, all commands executed can be logged to a file. To start logging commands, type the following on the Console tab page:
start_log fileName To stop logging, execute:
end_log

GUI Execution To execute the Platform Architect graphically we can use:
pct [project file ..] [script file ...]
where, project file refers to the platform architect specific project file (file.xml) and script file refers to the user defined TCL scripts.
To obtain an overview of the help, the user can execute:
pct -h $\| --help$
To check the version
pct -v $\| --version$

3.2 Platform Architect System Blocks and Encapsulations

All the blocks in Platform Architect can be divided into four elements:

1. Blocks - Blocks are used to represent the SystemC Modules
2. Buses - Buses serve as the foundational components of bus architecture and are delineated within a specific bus library, such as the AMBA® Bus Library, which encompasses nodes like AHB, APB, input-stage, and output-stage buses. These buses are produced through a bus generator embedded within the bus library itself. Moreover, the bus library outlines the functionalities and customizable settings for each type of bus.
3. Bridges - Bridges facilitate conversions, such as protocol and addressing mode adjustments, within connections linking bus nodes and between system blocks and bus nodes. These bridges may either be predefined components or dynamically generated by the bus generator.
4. Hardware-software interfaces: A hardware-software interface entails the implementation of communication between hardware and software components. It enables either memory-mapped I/O or interrupt-driven communication mechanisms between these blocks. These interfaces are specified and cataloged within a hardware-software interface library.

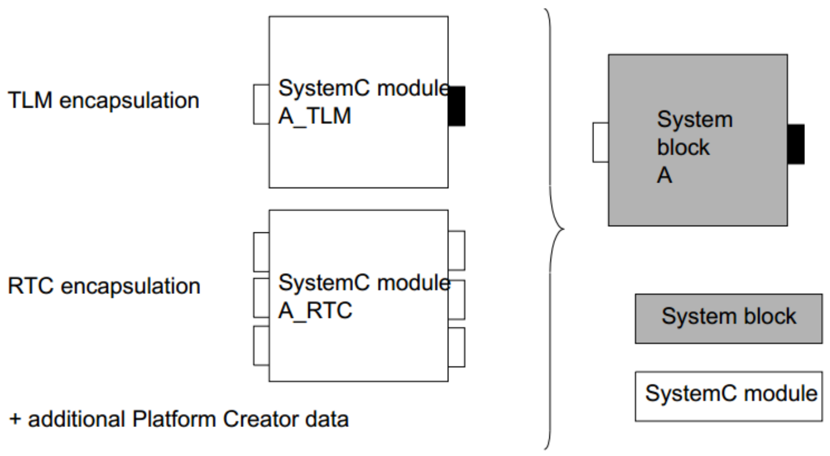


Figure 2: Platform Architect Blocks and Encapsulation

3.3 Creating a New Project and adding SoC Components

In this task, we will create a new project from GUI interface and try component for our SoC design. The SoC design will consist of:

- 1 Virtual Processing Unit (VPU)
- 1 Memory Generic
- 1 System Bus
- 1 Clock Generator
- 1 Reset Generator

1. After launching the Platform Architect, it will appear as shown below (Figure 3) if no script file or project file is selected. This window is divided into several main areas: the Menu Bar, Toolbars, Library Browser, Central Area, and Details Area. Further details about these areas can be found in the official manual.

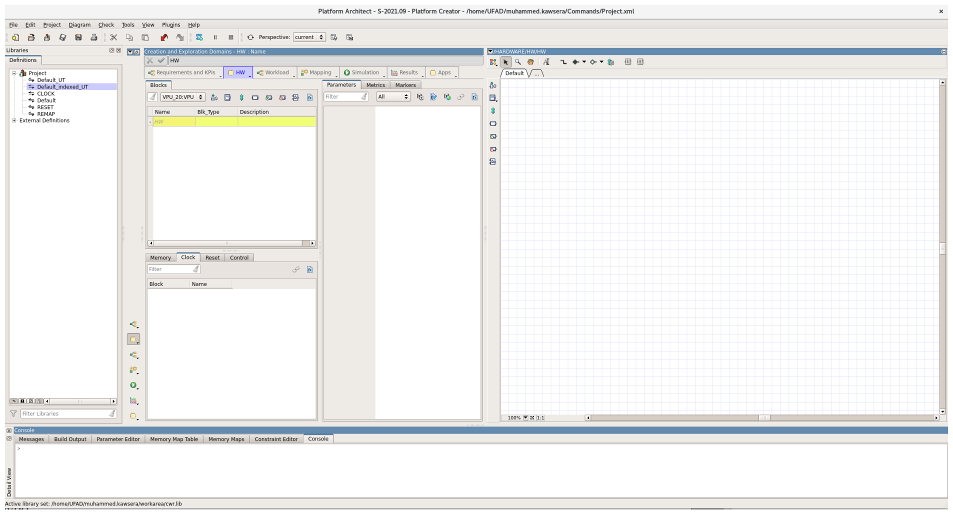
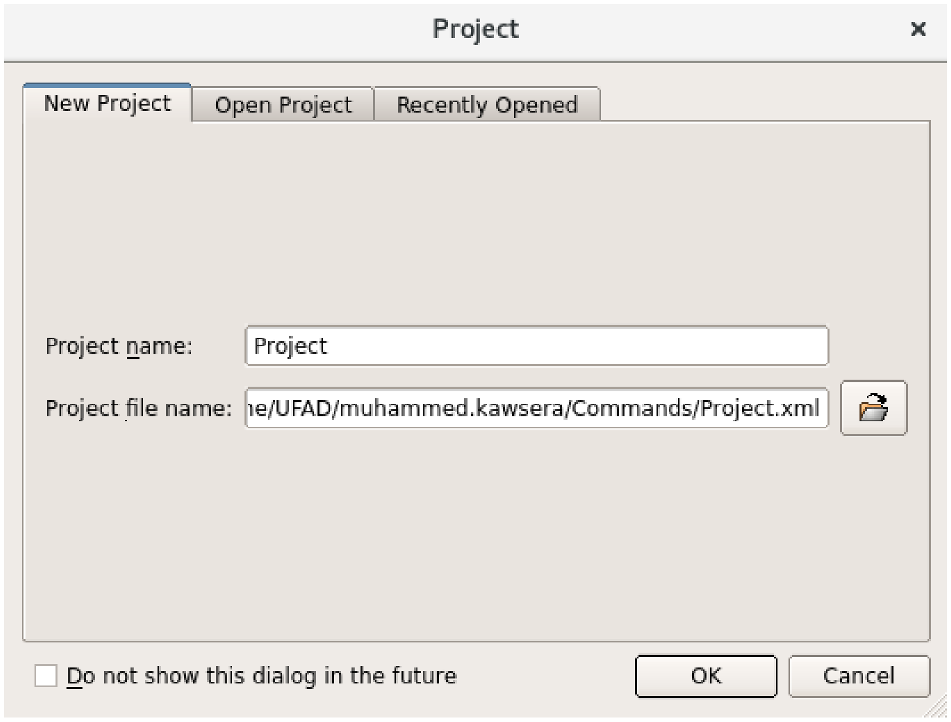

Platform Architect - S-2021.09 - Platform Creator - /home/UFAD/muhammed.kawsera/Commands/Project.xml
File Edit Project Diagram Check Tools View Plugins Help
Libraries
Creation and Exploration Domains - HW - Name
HARDWARE/HW/HW
Console
Active library set: /home/UFAD/muhammed.kawsera/workarea/cwr.lib

Figure 3: Initial Platform Architect Window

2. Create a new Platform Architect Project. In the GUI, this can be performed from the File Menu. Click the New Project button, which will prompt the new project window. Provide the new project name and location, and click OK after finishing.



Project name: Project

Project file name: ıe/UFAD/muhammed.kawsera/Commands/Project.xml

☐ Do not show this dialog in the future

Figure 4: New Project Window

3. Add each item of the Virtual Processing Unit (VPU), Memory Generic, System Bus, Clock Generator, and Reset Generator to the hardware design. These items can be added from the toolbar (Figure 5 on the left side of the Hardware Window Area.

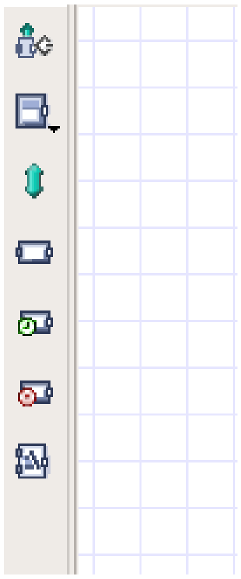


Figure 5: Hardware Toolbar Window

4. Start connecting the wires for the components of the SoC. The block design would be something like this (Figure 6):

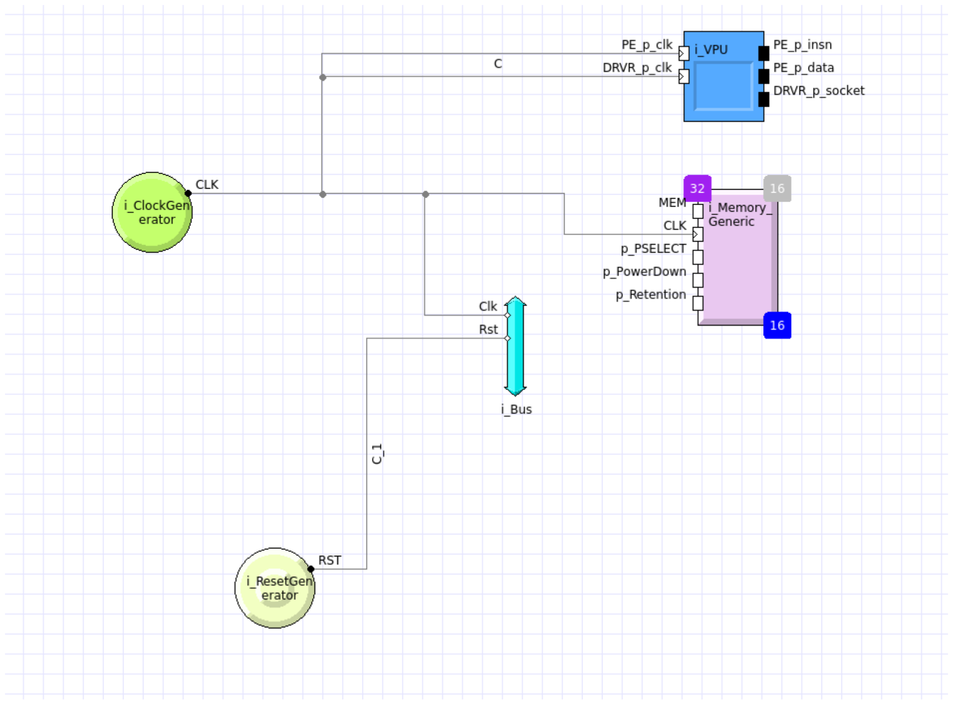


Figure 6: Block Design of the SoC Design

5. In Platform Architect, buses are configurable components, and the number of ports and types can be specified during their addition to the design. Additionally, a bus may have multiple master or slave ports. For simplicity, we haven't defined any new properties. Furthermore, for verification purposes, it is possible to define assertions for a bus.

6. **VPU**

   The Virtual Processing Unit (VPU) is a generic block equipped with a task manager and a scheduler, responsible for controlling the set of tasks mapped to it. This block operates as a standard hierarchical block within Platform Architect, allowing access to its internal structure via double-clicking. Inside the VPU block, the instantiation of processing models, memory drivers, and communication helpers is recommended. By default, the VPU includes a memory driver and a processing element.

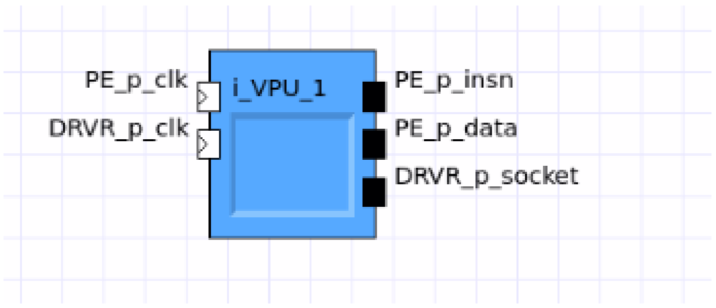
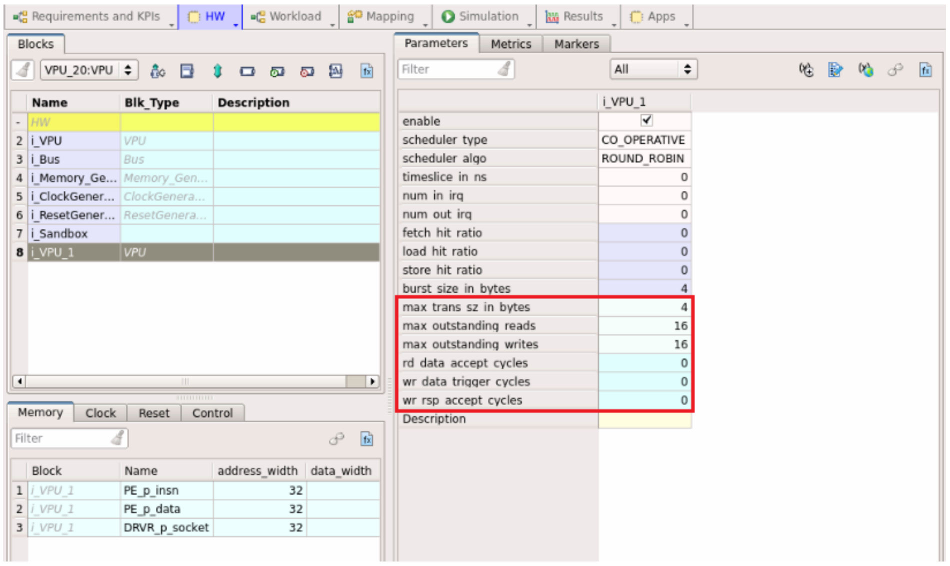


Figure 7: Virtual Processing Unit (VPU)

The memory driver (DRVR) is a generic block tasked with generating memory transactions, with read and write transactions dispatched via the initiator socket. Within the VPU Block, integration of the memory driver occurs, utilizing ports such as **DRVR_p_socket** and **DRVR_p_clk**. The former designates the initiator socket required for memory transactions, typically set to AXI, while the latter indicates the input clock, usually linked to the main source clock.



	Name	Blk_Type	Description
-	HW		
2	i_VPU	VPU	
3	i_Bus	Bus	
4	i_Memory_Ge...	Memory_Gen...	
5	i_ClockGener...	ClockGenera...	
6	i_ResetGener...	ResetGenera...	
7	i_Sandbox		
8	i_VPU_1	VPU	

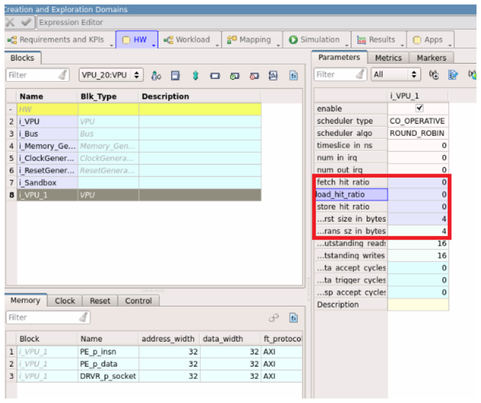
	i_VPU_1
enable	✓
scheduler type	CO_OPERATIVE
scheduler algo	ROUND_ROBIN
timeslice in ns	0
num in irq	0
num out irq	0
fetch hit ratio	0
load hit ratio	0
store hit ratio	0
burst size in bytes	4
max trans sz in bytes	4
max outstanding reads	16
max outstanding writes	16
rd data accept cycles	0
wr data trigger cycles	0
wr rsp accept cycles	0
Description	

	Block	Name	address_width	data_width
1	i_VPU_1	PE_p_insn	32	
2	i_VPU_1	PE_p_data	32	
3	i_VPU_1	DRVR_p_socket	32	

Figure 8: VPU Memory Driver Paramters

Upon selecting this block in the design, its parameters are displayed in the Parameters section of the Platform Creator user interface.

The Processing Element (PE) denotes a generic processing resource within a VPU, consuming the configured processing cycles on a task. Its signals are referenced as **PE_p_insn** and **PE_p_data**. Please refer to the table illustrated in the following figure for the Processing Element parameters.



	Name	Blk_Type	Description
-	HW		
2	i_VPU	VPU	
3	i_Bus	Bus	
4	i_Memory_Ge...	Memory_Gen...	
5	i_ClockGener...	ClockGenera...	
6	i_ResetGener...	ResetGenera...	
7	i_Sandbox		
8	i_VPU_1	VPU	

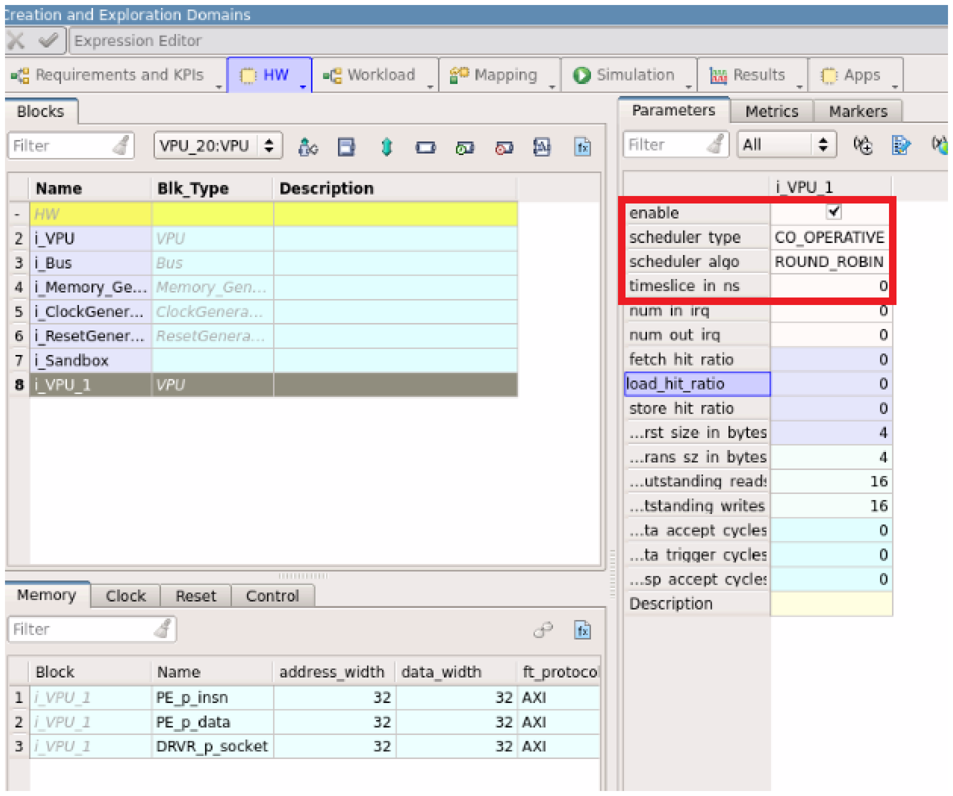
	i_VPU_1
enable	✓
scheduler type	CO_OPERATIVE
scheduler algo	ROUND_ROBIN
timeslice in ns	0
num in irq	0
num out irq	0
fetch hit ratio	0
load_hit_ratio	0
store hit ratio	0
...rst size in bytes	4
...rans sz in bytes	4
...utstanding reads	16
...tstanding writes	16
...ta accept cycles	0
...ta trigger cycles	0
...sp accept cycles	0
Description	

	Block	Name	address_width	data_width	ft_protocol
1	i_VPU_1	PE_p_insn	32	32	AXI
2	i_VPU_1	PE_p_data	32	32	AXI
3	i_VPU_1	DRVR_p_socket	32	32	AXI

Figure 9: VPU Parameters Processing

Scheduling Properties for a VPU Block There are some configuration options for the VPU Block Scheduling (Figure). These are shown below:

- enable
- scheduler_type
- scheduler_algo
- timeslice_in_ns



	Name	Blk_Type	Description
-	HW		
2	i_VPU	VPU	
3	i_Bus	Bus	
4	i_Memory_Ge...	Memory_Gen...	
5	i_ClockGener...	ClockGenera...	
6	i_ResetGener...	ResetGenera...	
7	i_Sandbox		
8	i_VPU_1	VPU	

	i_VPU_1
enable	✓
scheduler type	CO_OPERATIVE
scheduler algo	ROUND_ROBIN
timeslice in ns	0
num in irq	0
num out irq	0
fetch hit ratio	0
load_hit_ratio	0
store hit ratio	0
...rst size in bytes	4
...rans sz in bytes	4
...utstanding reads	16
...tstanding writes	16
...ta accept cycles	0
...ta trigger cycles	0
...sp accept cycles	0
Description	

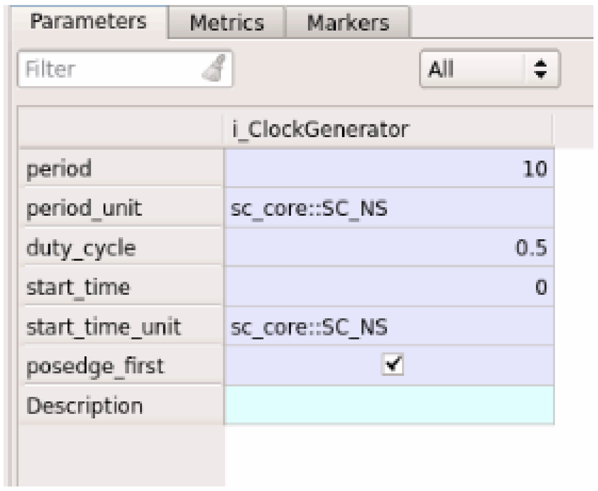
	Block	Name	address_width	data_width	ft_protocol
1	i_VPU_1	PE_p_insn	32	32	AXI
2	i_VPU_1	PE_p_data	32	32	AXI
3	i_VPU_1	DRVR_p_socket	32	32	AXI

Figure 10: VPU paramters Scheduling

Interrupt Properties of a VPU VPU supports two types of interrupt interfaces:

- Input interface: Receives interrupt signal from an external peripheral.
- Output interface: Generates interrupt signal to an external peripheral.

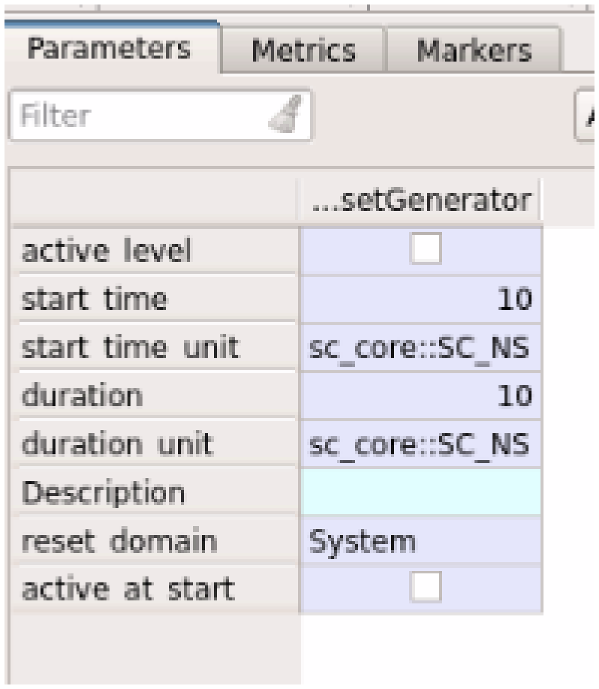
7. **Clock and Reset** The ClockGenerator module generates a configurable clock signal. Figure 11 shows the properties of generated clock.



	i_ClockGenerator
period	10
period_unit	sc_core::SC_NS
duty_cycle	0.5
start_time	0
start_time_unit	sc_core::SC_NS
posedge_first	✓
Description	

Figure 11: Clock Generator Properties

Figure 12 shows the basic properties of the reset block generator.



	...setGenerator
active level	☐
start time	10
start time unit	sc_core::SC_NS
duration	10
duration unit	sc_core::SC_NS
Description	
reset domain	System
active at start	☐

Figure 12: Reset Generator Properties

3.4 Adding Custom IP and HDL Code

Platform Architect can add custom IPs, HDL codes or even SystemC components to the hardware design. For HDL code and IPs we need to define ports and protocols. But SystemC modules will be automically wrapped when they are imported in system blocks. These system blocks will not have extra port or block parameter sets and need to manually add those.

1. **Protocols**

   Each protocol possesses the following attributes: a) A name, b) Available versions (for various abstraction levels and directions), c) Parameters, d) Address width, and e) Data width. These properties may be general or specific to the abstraction level. Both data width and address width are essential properties required by the hardware architecture creation tool.

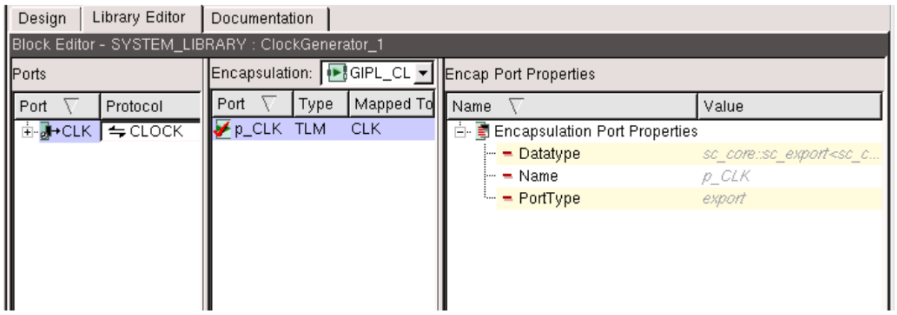
2. **Ports** At the TLM abstraction level, SystemC describes port types which call (sc_port) and implement (sc_export) an interface. A system block port has the following properties:

   - A name
   - A direction (in, out, inout)
   - An initiator/target flag
   - A protocol
   - Protocol parameters

   Only the values for the general parameters of a protocol for a port need to be present. For abstraction-level specific parameters, these values do not necessarily need to be present. When these abstraction-level specific parameter values are omitted, the port cannot be exported at that abstraction level.

3. **Editing a Port or Protocols**

   Platform Architect allows editing, adding, removing or renaming a port/protocols. To perform these operations, in the Library Browser, double-click on the specific block. It will pop up the Block Editor panel on Library Editor tab page similar like below: At the left side, all the defined



Port	Protocol
CLK	CLOCK

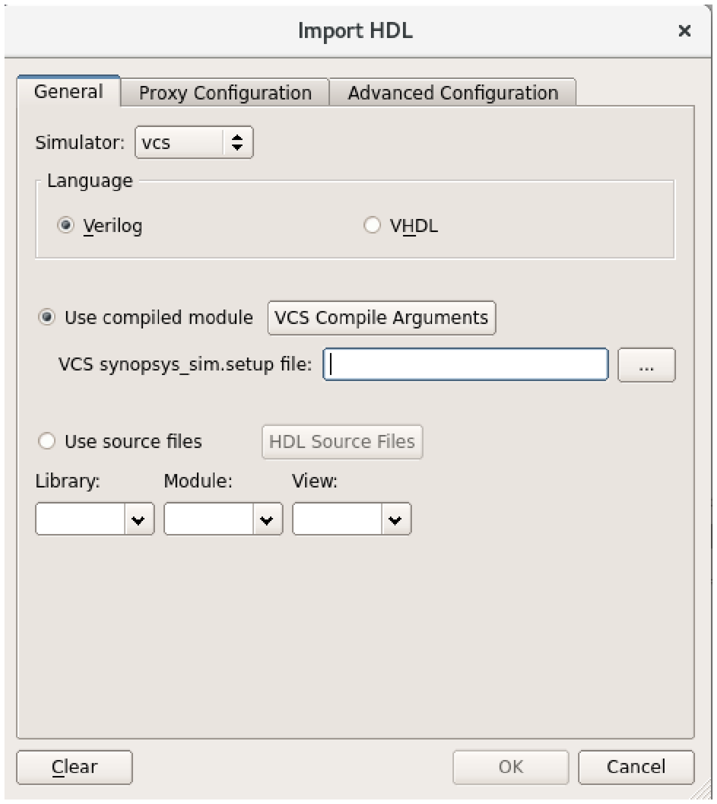
Port	Type	Mapped To
p_CLK	TLM	CLK

Name	Value
Encapsulation Port Properties	
Datatype	sc_core::sc_export<sc_c...
Name	p_CLK
PortType	export

Figure 13: Editing a Port or Protocol

   ports will be shown. In encapsulation column, the port mapping will be shown.

4. **Adding Custom HDL Code:** Before importing an HDL module, we need to set up the HDL Simulator environment. The Import HDL wizard can be found in the menu bar by selecting Project- Import HDL. This window allows us to import HDL from either Verilog or VHDL. Additionally, we can specify the compiled module instead of the source HDL files. The simulator options are vcs, mti or ncsim. Clicking HDL source files button will open the Compile HDL source files window where the user can specify the paths of the source files including compile arguments.



Simulator: vcs

Language: [✓ Verilog] VHDL

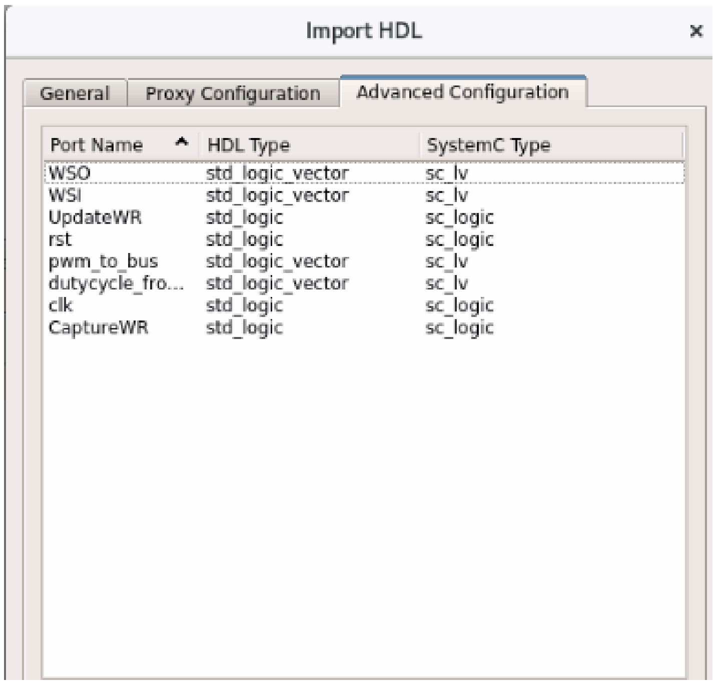
[✓ Use compiled module]

VCS synopsys_sim.setup file:

Use source files

Library: Module: View:

Figure 14: Adding Custom HDL Code



Port Name	HDL Type	SystemC Type
WSO	std_logic_vector	sc_lv
WSI	std_logic_vector	sc_lv
UpdateWR	std_logic	sc_logic
rst	std_logic	sc_logic
pwm_to_bus	std_logic_vector	sc_lv
dutycycle_fro...	std_logic_vector	sc_lv
clk	std_logic	sc_logic
CaptureWR	std_logic	sc_logic

Figure 15: Adding Custom HDL Code

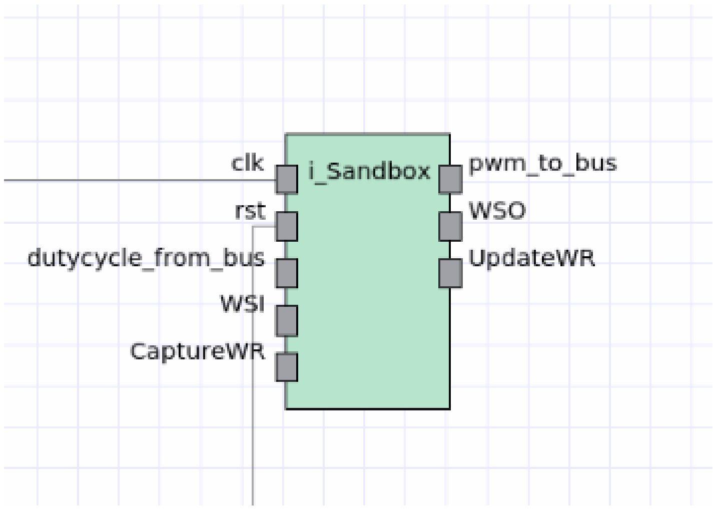


Figure 16: Adding Custom HDL Code

5. **Adding SystemC Module** Upon importing SystemC modules into Platform Creator, they undergo wrapping into system blocks. These blocks are initially equipped with default mappings linking the SystemC ports to the system block ports. Upon importing a system block, several manual steps must be executed:

   - Consolidate various encapsulations into a single block: Each SystemC module is initially wrapped as an individual system block. However, multiple modules may represent distinct encapsulations of the same system block. These encapsulations can be consolidated into a unified system block.
   - Modify the properties of the block as follows: name, ports, and their respective properties.

   SystemC source files including process involves of providing three information: a) The include path, b) The preprocessor defines and c) The source files.

At the level of TLM abstraction, SystemC defines various port types such as those for calling (sc_port) and implementing (sc_export) interfaces, along with channels that implement interfaces. Ports that call an interface must always be linked to a port implementing the same interface or to a channel. This characteristic is maintained in the specific properties of TLM ports, with the Port type parameter storing this information. Its value could be one of four options: port, export, interface, or unknown. The first three correspond to SystemC types sc_port, sc_export, and sc_interface respectively, while "unknown" is used by Platform Architect when the port type cannot be determined and must be set manually.

For imported blocks, this property is established during import based on the type of SystemC ports. The compatibility of a bus library with sc_ports, sc_exports, or sc_interfaces depends on the chosen TLM modeling style. When establishing a port on a bus by connecting it to a port, typically the inverse of the block port is chosen. However, for channel-based communication like OCP, both the bus and peripheral ports must be of type "port." When creating an initiator or target port within a hierarchy and connecting it to a bus while the bus library supports different port types, the type for the hierarchical port must be manually selected.

6. **Initiator Porst and Target Ports** An initiator port serves as an outlet through which transactions are dispatched onto a bus, whereas a target port acts as a destination for transactions received on a bus. Usually, an initiator port is associated with a bus master, while a target port is associated with a bus slave. Initiator and target ports are available for below ports :

   - Memory
   - Clock
   - Reset
   - Remap
   - Control

3.5 Generate HDL Netlist of the SoC Design

Platform Architect can generate an HDL netlist that describes your design, which can include both HDL and SystemC blocks and buses. If your system comprises solely HDL blocks, the exported design is prepared for compilation and execution by your HDL simulator. However, if your system includes some SystemC blocks, you will still receive an HDL netlist, and Platform Architect will automatically generate HDL wrapper blocks around your SystemC blocks. There is a warning that BCA port are not supported for netlist creation.

The export command extract the simulation netlist with the settings from the current project. The Command for this: **export_simulation**. We can create a simulation build configuration for the current project using **create_simulation_build_config** command.

1. **Running Simulation Externally:** If your system is entirely composed of HDL blocks, you have the option to conduct the simulation within your chosen Verilog/VHDL environment. The systemfilelist.f file comprises a comprehensive list of all HDL source files in a bottom-up manner. This list serves as an initial reference for constructing your simulation using the HDL simulator of your preference.

2. **Running inside the Platform Architect**

   System that has been exported as HDL designs can be run in Platform Architect along with SystemC. The nature of this simulation will HW/SW cosim. The platform will create a SystemC Wrapper around every top level HDL block which will instantiate the HDL block with proper ports and interfaces. To run the simulation, one should click the play button.