2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Challenges and Solutions for Creating Virtual Platforms of FPGA and SASIC Designs

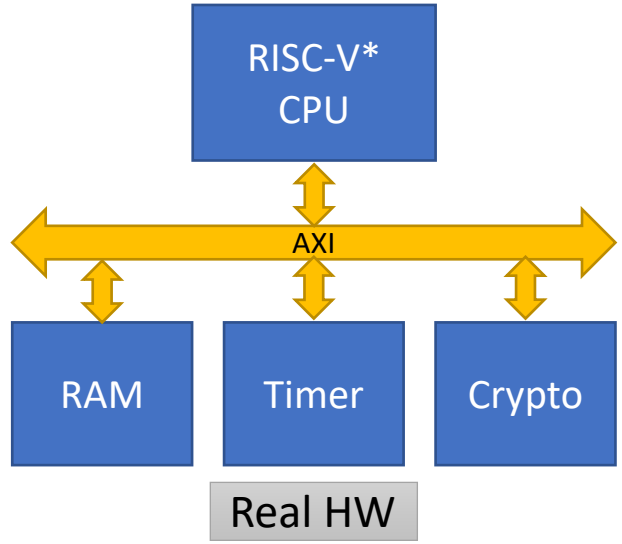
Kalen Brunham and Jakob Engblom, Intel Corp.

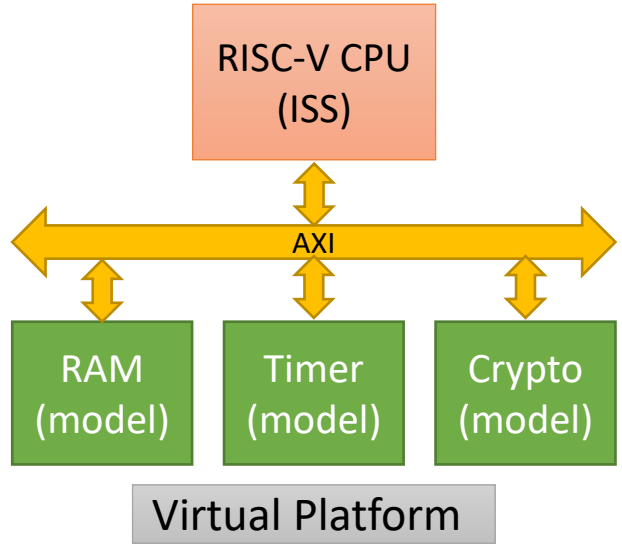
intel

accellera SYSTEMS INITIATIVE

What is a Virtual Platform?

- A model of a hardware (HW) system (board) that can run the same software (SW) binary as the actual HW
  - Simulates the processor cores and SW visible HW peripherals
  - Simulator application can run on a different host processor type than the system being modeled
  - Common VP technologies include the Simics® simulator, QEMU, etc.
- A Virtual Platform typically includes:
  - Instruction set simulator (ISS) for the processor cores
  - LT-TLM models for each peripheral in the system
  - LT-LM models for all board level components
  - Infrastructure to connect models and ISS together and facilitate debug of code running on the ISS

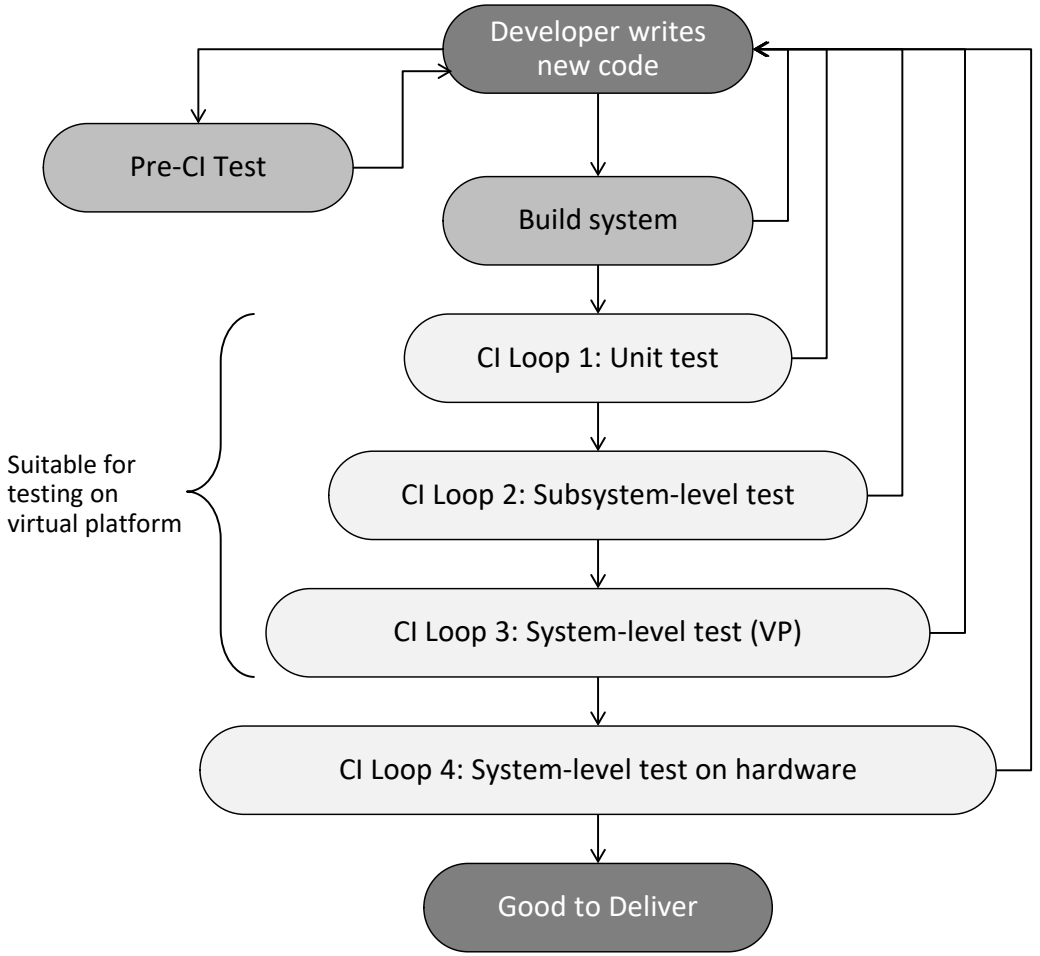




*Other names and brands may be claimed as the property of others

VP Value Proposition

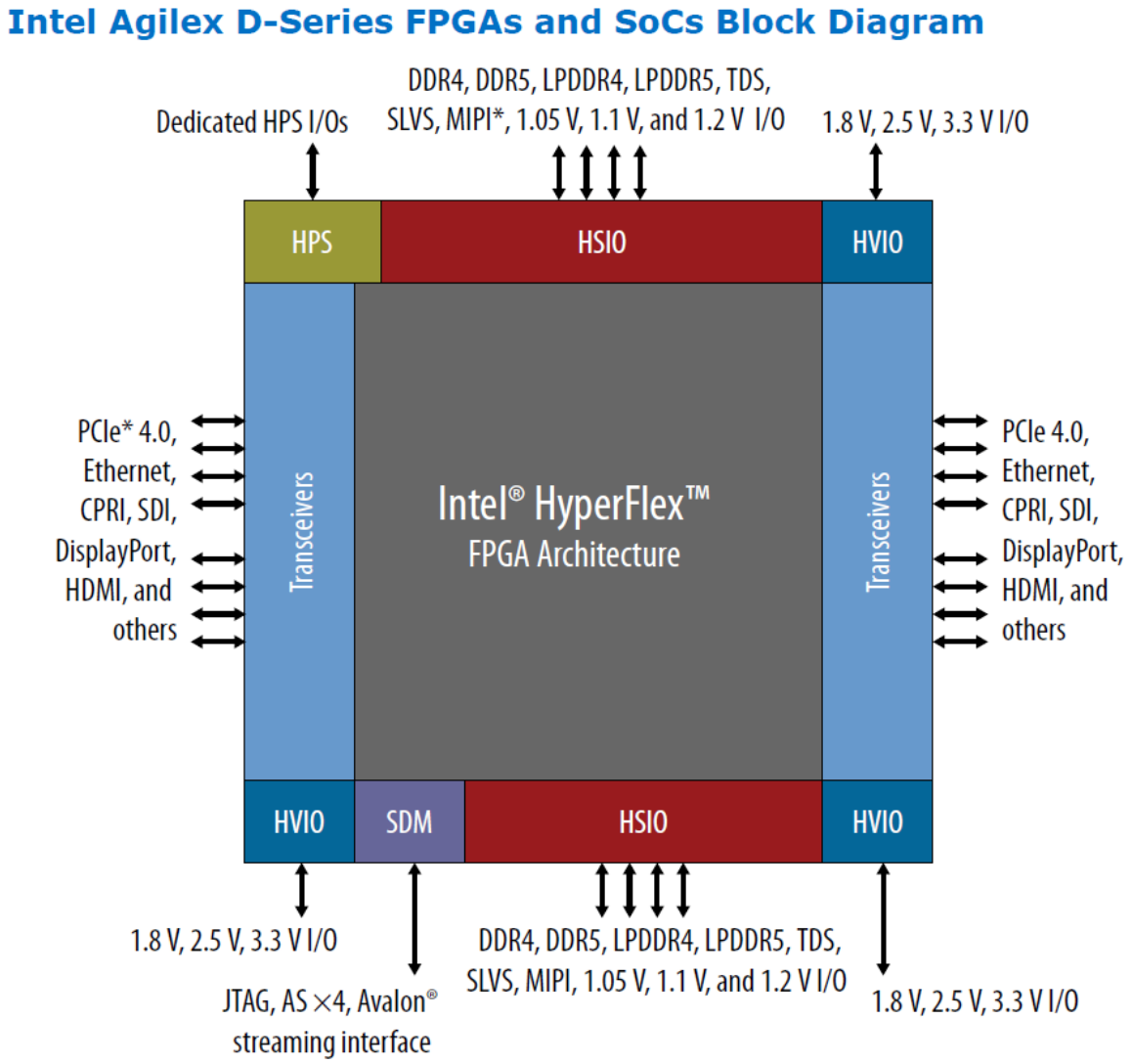
- VPs enable SW development and test before HW is available **and** before testing on HW
  - Enables SW shift-left for development and feedback
  - Removes the need for large board farms for testing and augments board farm purpose
- Initial testing can be done using a VP before running tests on HW
  - Requires sufficient detail of the VP so the same SW binary can be used on the VP and on the actual HW
- Use of VPs is standard at many large companies



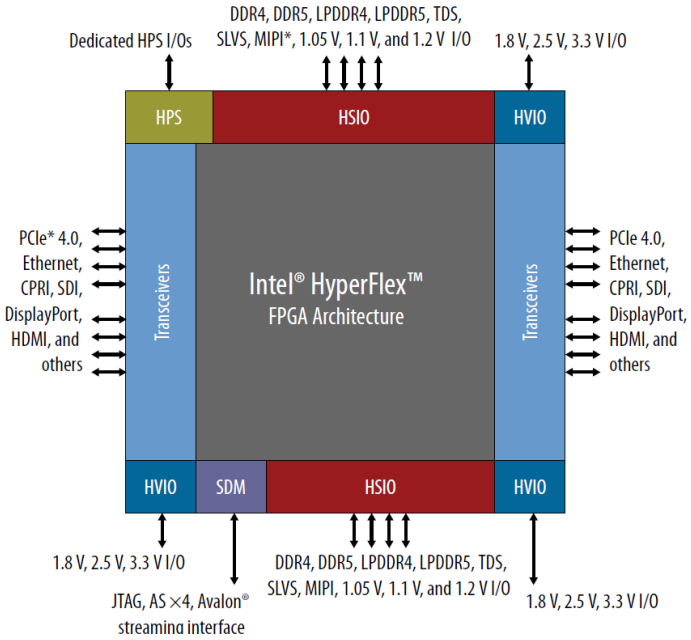
What is an FPGA?

(Field-Programmable Gate Array)

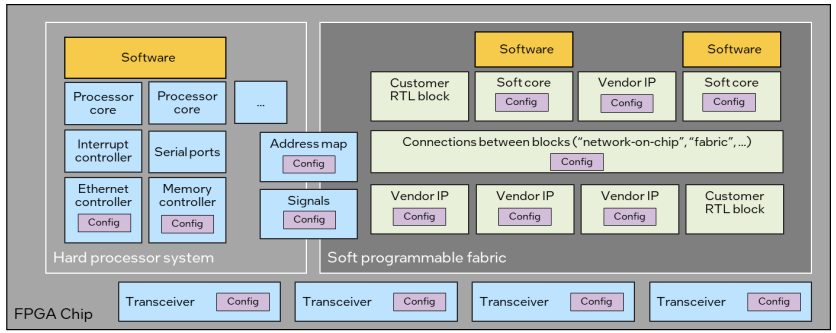
- FPGA is a semiconductor device where the end function is defined after manufacturing
  - Vendors provides “blank” devices and complete toolchains for use in targeting their devices
  - Customers create the end design
- Devices provide many features *hard* and *soft* which can be used in a customer design
  - Vendors typically provide IP for interfaces, accelerators and processors
  - Customers choose which IPs, and their exact functionality
- Structured ASICs (SASICs) share many of these same properties



FPGA Engagement Model



?

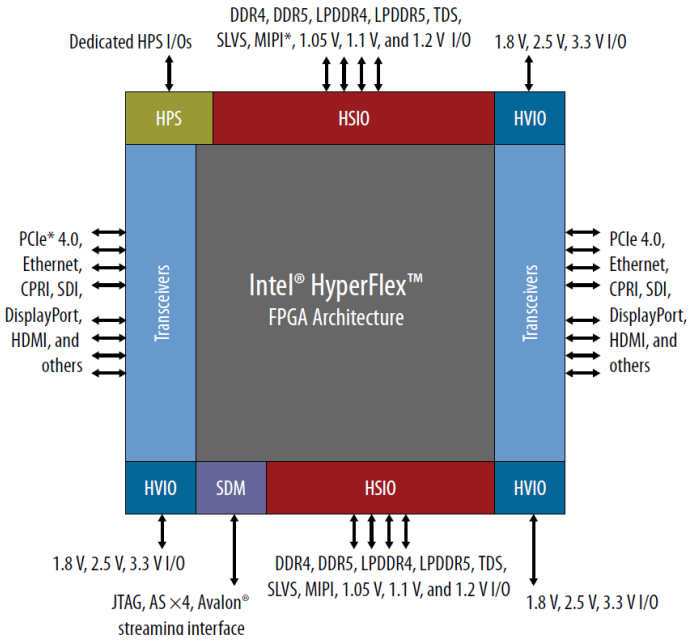


What vendors provide:
Blank unprogrammed FPGA

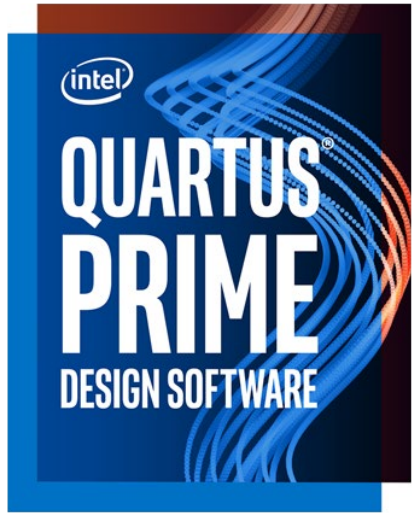
What customer want:
Digital system implemented in an FPGA

FPGA Engagement Model

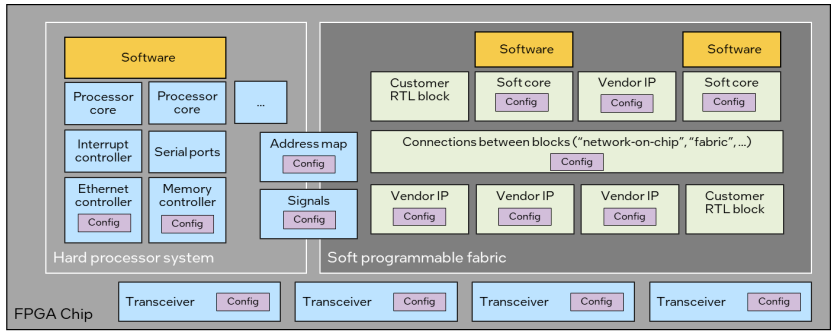
Design files and constraints (Customer)



Chip definition



Bitstream



What vendors provide:
Blank unprogrammed FPGA

FPGA Design Tools (Quartus):
Creates a legal mapping of the input design on the target device to legally configure all blocks in the FPGA

What customer want:
Digital system implemented in an FPGA

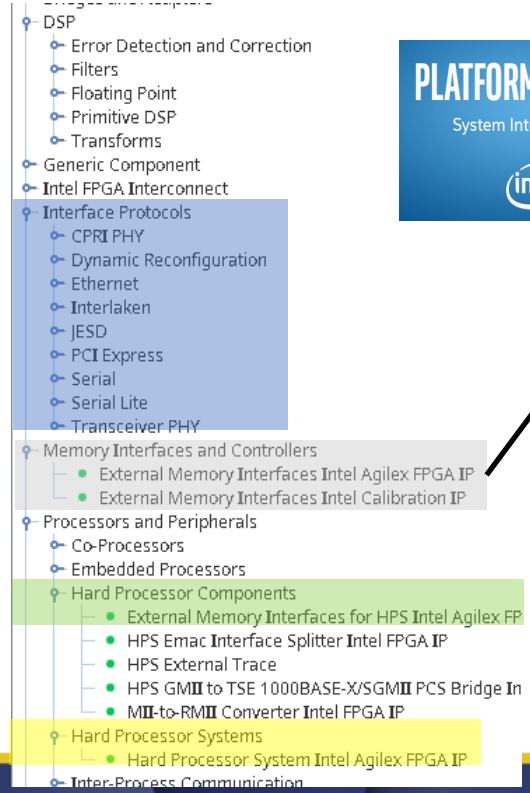
FPGA Intellectual Property Block (IP) Parameterization

There is an IP Catalog of available IPs

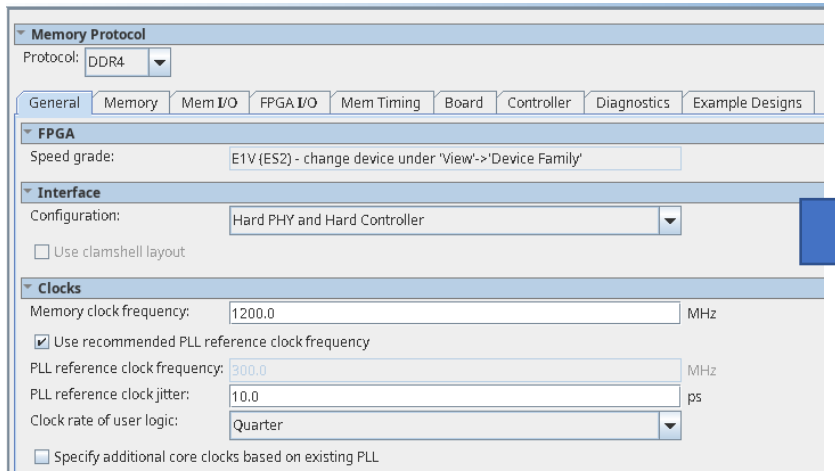
XCVR

EMIF

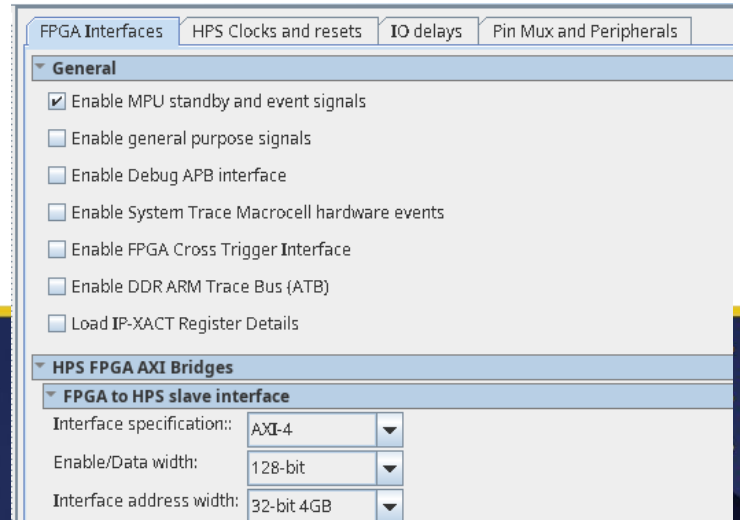
HPS



PLATFORM DESIGNER
System Integration Tool
intel



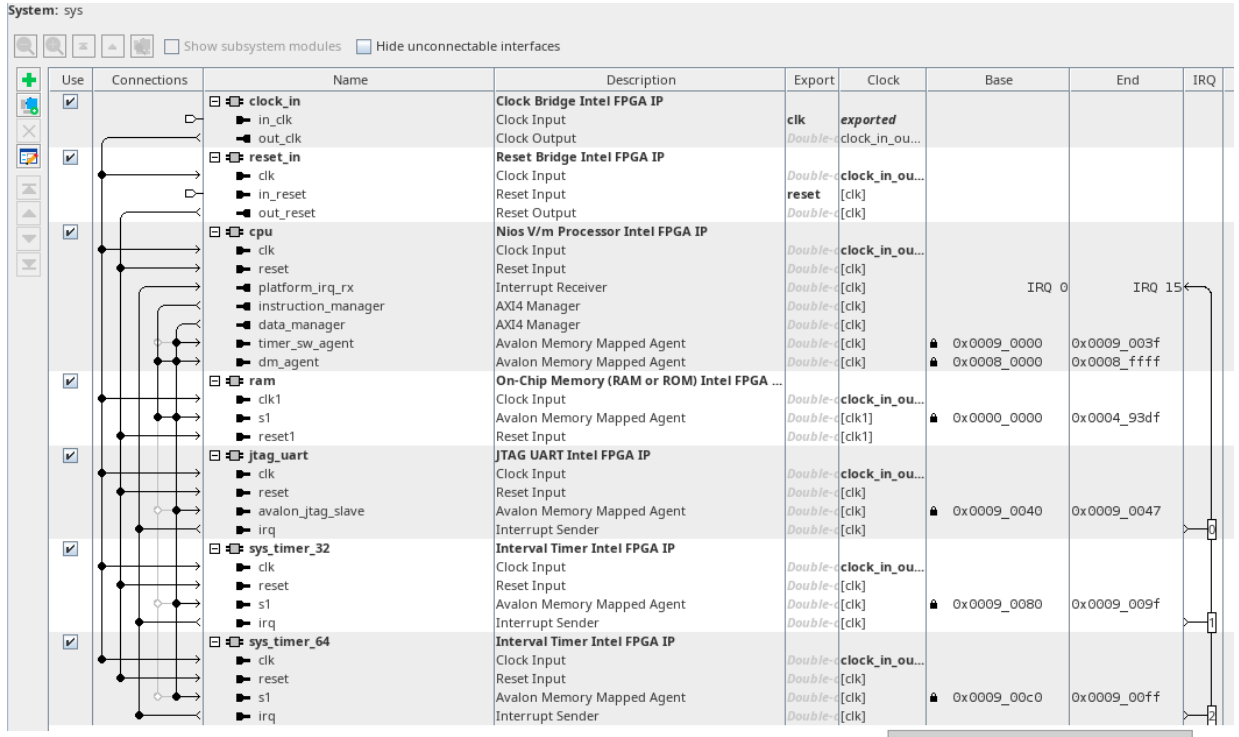
Customers pick parameters



Generates RTL that can go through FPGA toolchain or RTL simulation

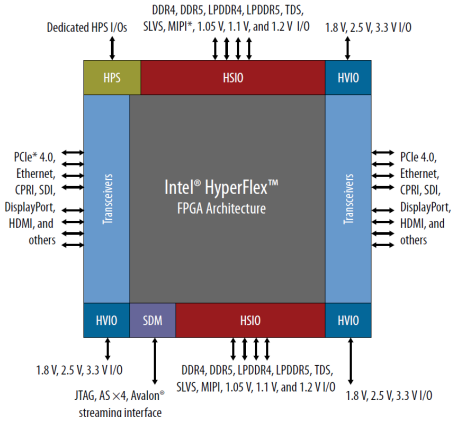
FPGA System Assembly Tools

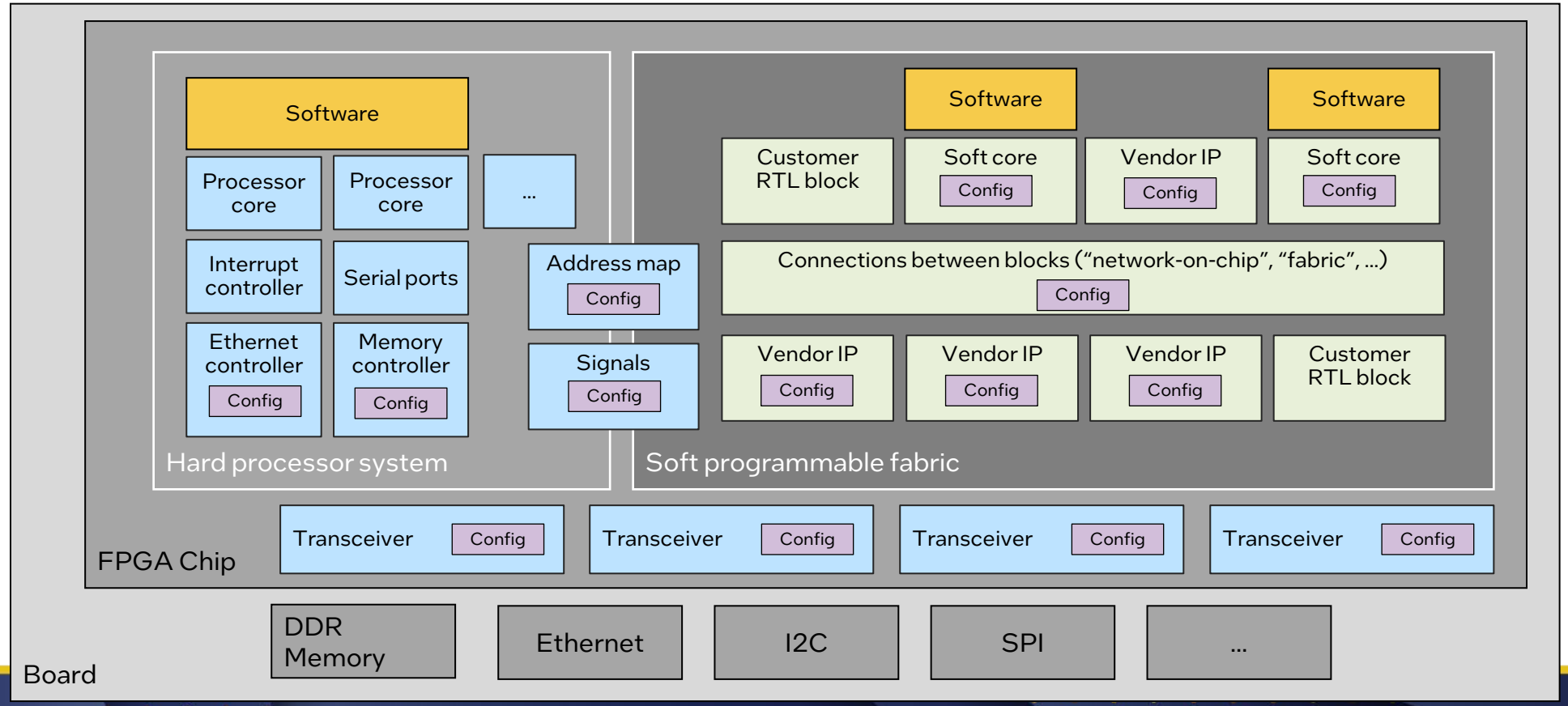
- Customers can instantiate multiple different IPs and then connect them together in their overall system
  - Assembly tool used to capture the connectivity and memory map
  - The system level, and each IP instance, generates RTL for consumption by downstream tools
- Customers come back to the system assembly tool if system properties or IP parameters need to be changed
  - Ie. legality, timing, requirements change



FPGA/SASIC design

- Modern designs are increasingly complex and commonly include a SW component





Software

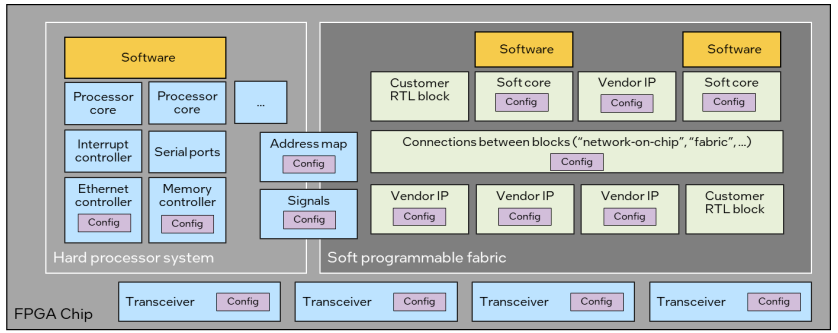
Hard IP shipping as a fixed part of the FPGA chip

“Soft” IP instantiated in the programmable fabric

User-provided configuration

Challenges for VP with FPGA/SASIC designs

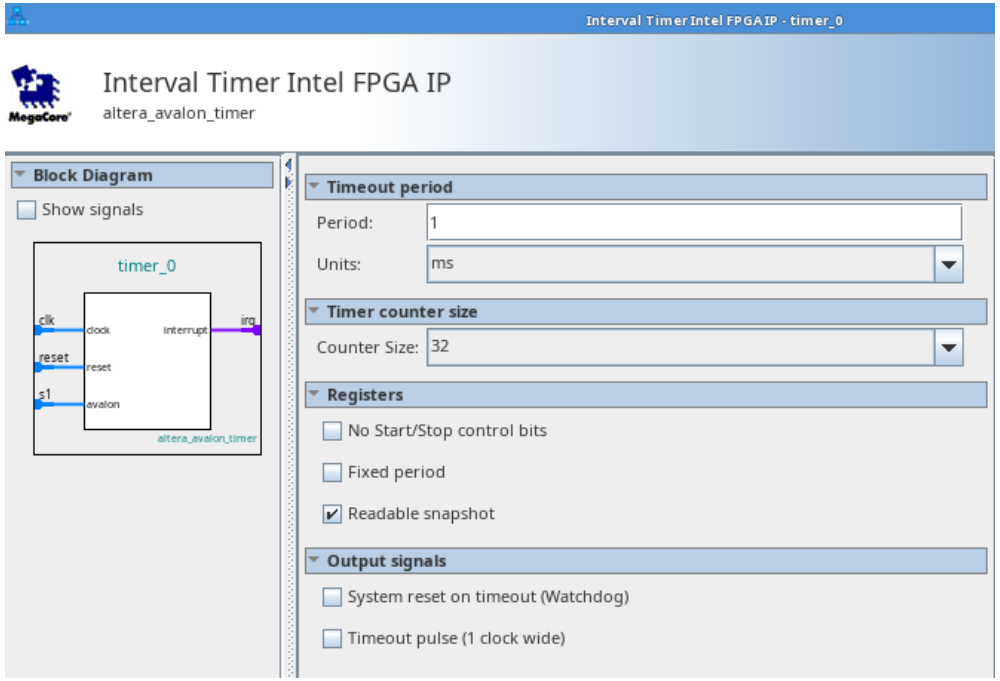
- Unlike for “fixed” ASIC, FPGA vendors cannot create a ready-made VP since customer creates end design
  - Customers must create based on their specific design
- Vendor IP is highly parameterizable, parameters can affect SW interface
  - Parameters of the IP need to be easily changeable and in-sync with the HW design
- Customers add their own IP and want it in the VP
- FPGA teams are smaller than ASIC teams
  - Relative modeling to design size means smaller number of modelers but similar complexity



What customer want:
Virtual platform of *their* digital system

SW Interface Impact of IP Parameterization

- Simple Timer from Quartus
  - Small number of IP parameters
- Large potential impact to SW interface
  - Impact of parameterization to more complex IPs even larger
- Expectation for change during design



Counter Size == 32

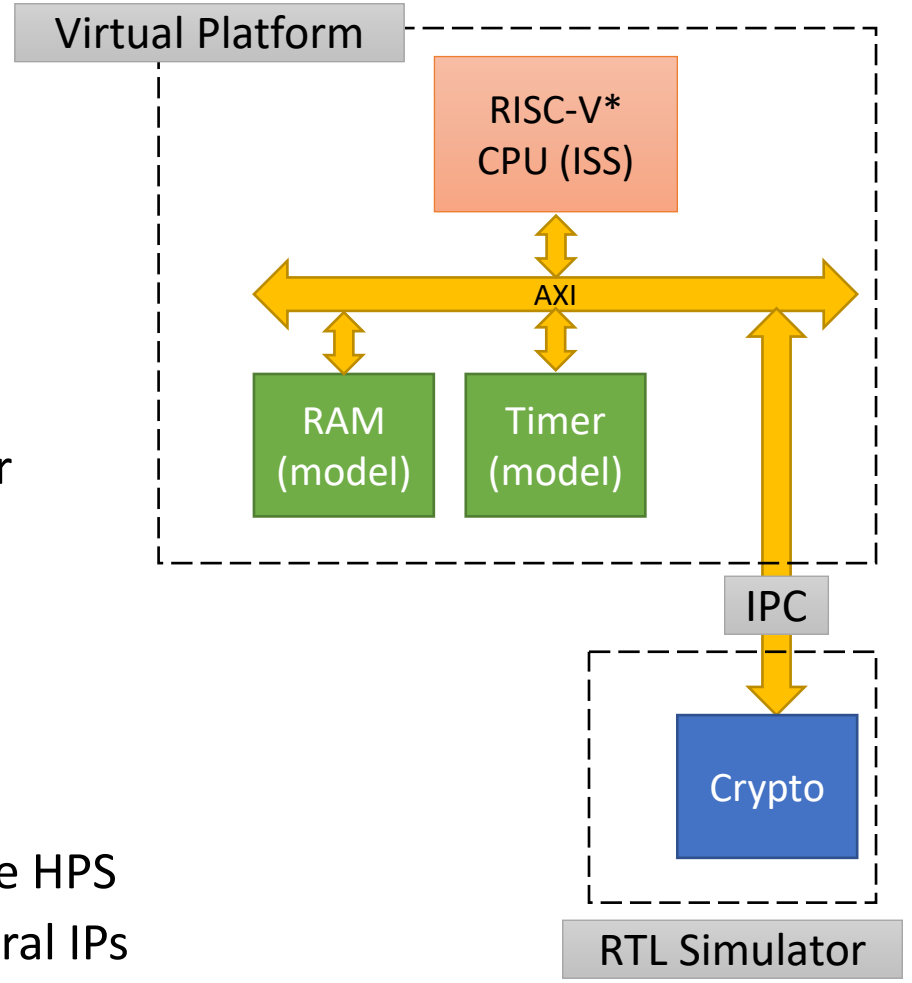
Offset	Reg Name	Description
0	status	Run[1]|TO[0]
1	control	Stop[3]|Start[2]|Cont[1]|ITO[0]
2	periodl	Timeout Period [15:0]
3	periodh	Timeout Period [31:16]
4	snapl	Counter Snapshot [15:0]
5	snaph	Counter Snapshot [31:16]

Counter Size == 64

Reg Name	Description
status	Run[1]|TO[0]
control	Stop[3]|Start[2]|Cont[1]|ITO[0]
period_0	Timeout Period [15:0]
period_1	Timeout Period [31:16]
period_2	Timeout Period [47:32]
period_3	Timeout Period [64:48]
snap_0	Counter Snapshot [15:0]
snap_1	Counter Snapshot [31:16]
snap_2	Counter Snapshot [47:32]
snap_3	Counter Snapshot [64:48]

Alternatives to VP

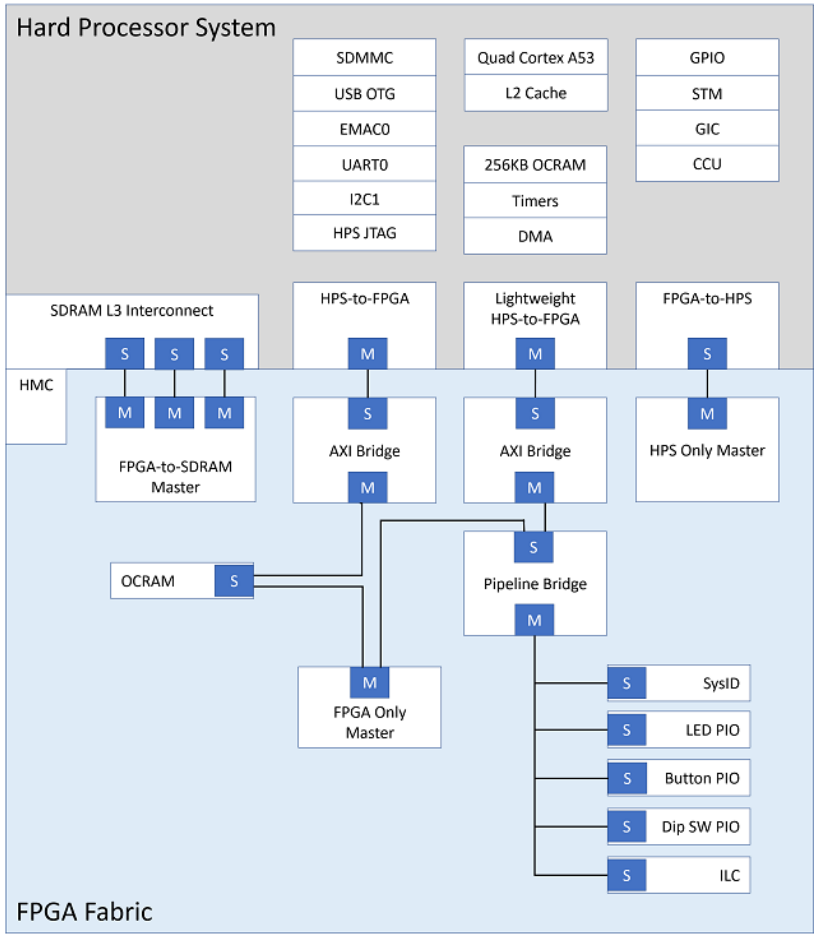
- Host compiled simulation
  - Make C/C++ models for HW
  - Requires separate build and hardware abstraction layer
- VP-Register Transfer Level (RTL) co-simulation
  - Enables use of RTL as opposed to creating a model
  - Significantly slower than a complete TLM VP
- Superset model for hard processor only
  - Remove some parameterization from the models of the HPS
  - Helpful but limiting for fabric design and other peripheral IPs
- No alternative matches the LT-TLM Virtual Platform



*Other names and brands may be claimed as the property of others

Superset Model for FPGA-SOC

- Most common VP solution for HPS of FPGAs today
- Vendor creates VP for generic HPS parameterization
  - Ignores certain legality or connectivity not actually possible
  - Create generic memory model for fabric connectivity or example design
- Superset Hard Processor System (HPS) model prevents use of negative testing
  - Peripherals not connected may still function in VP
- Not practical for soft processors
  - Processor, peripherals, and connectivity parameterized
- Does not extend to parameterizable interface IP
  - No superset model equivalent

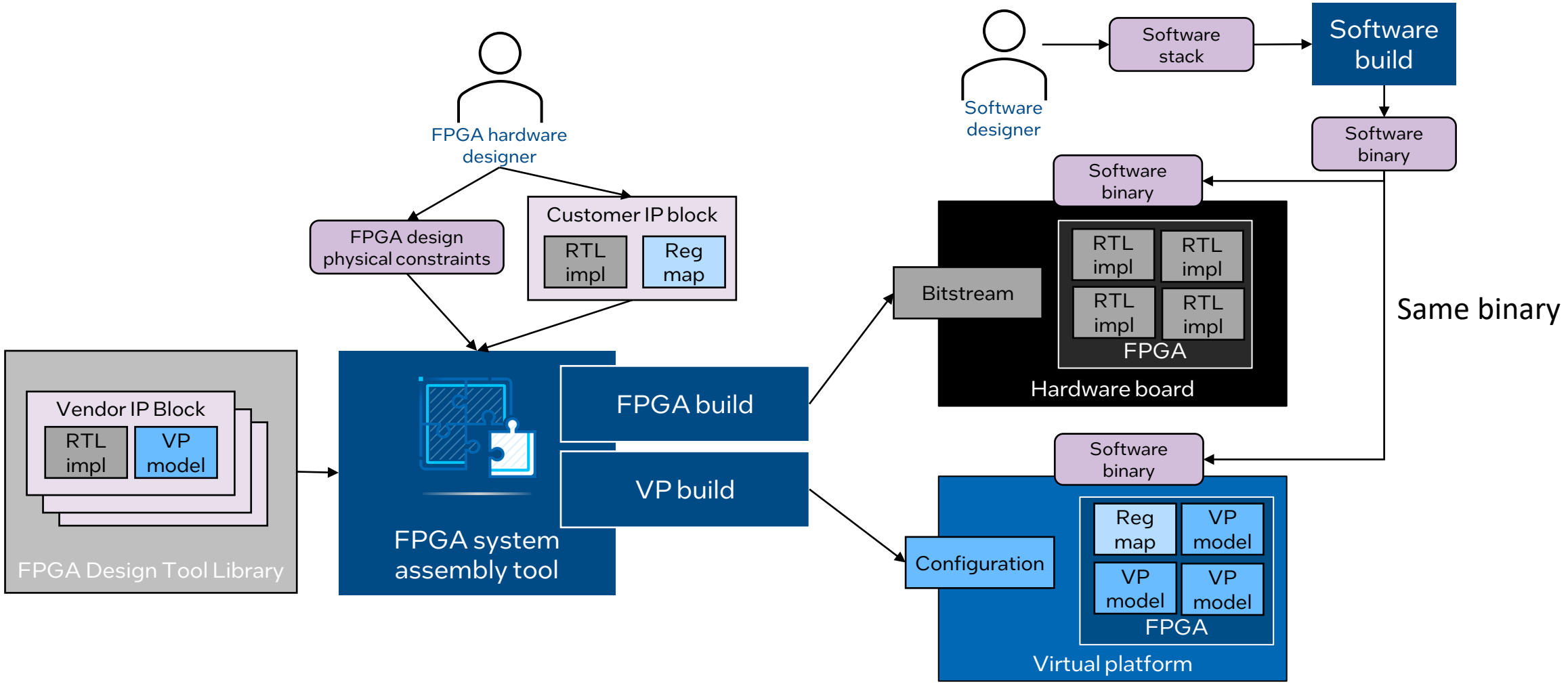


Agilex Golden Hardware Reference Design (GHRD)

Leverage FPGA Assembly Flow for VP

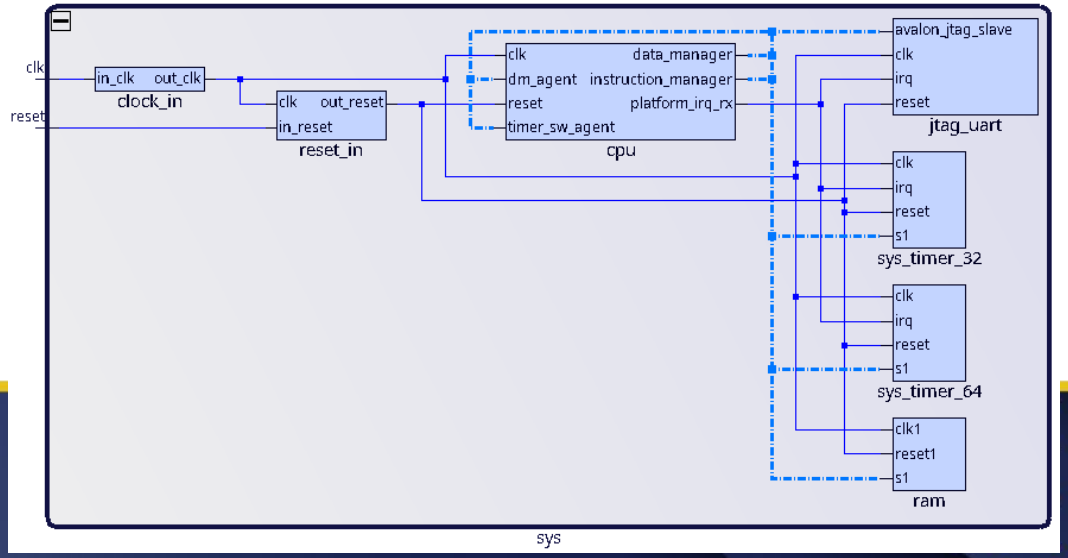
- We believe answer to creating VPs for FPGA/SASIC designs is to generate the VP from the existing FPGA/SASIC workflow
  - Enhance existing tools to generate parameterized VP models in addition to RTL
- Automatic VP generation is practical due to:
  1. Most designs rely on vendor IP for the shell of their design
  2. Designers use the vendor assembly tools to build portions of their designs, especially the processor-based systems
  3. Assembly tools offer a library of IPs
- Customer RTL can be initially represented as stub models
  - Customer RTL can be left as black box models or augmented to have side affects

Proposed VP Generation Flow



Prototype: Automatic Creation of a Simics VP

- Example design connecting a Nios V/m CPU, RAM, JTAG UART, and 2 timers
  - Design captured using Quartus Platform Designer
  - RTL generation produces RTL for each IP in system and the system itself
  - Simics generation produces DML and Python for each IP and the system
- Simics VP generation from existing design capture tool and workflow
  - Does not require use of any new FPGA design tool



Generated component

- Components used for system and subsystems
- Generated component
  - Instantiates each model and system level parameters
  - Instantiates memory map for the system and connectivity between models
- Generated code may add additional hierarchy nodes vs naïve manual VP
  - Benefit is match to the RTL and comes with minimal runtime cost

```
class fpgadesign_cpu_comp(StandardConnectorComponent):
  _class_desc = "Nios Vm CPU from cpu.ip (intel_niosv_m)"
  # Constants from ‘cpu.ip’
  TIMER_SW_AGENT_BASE_ADDRESS = 0x00090000
  TIMER_SW_AGENT_ADDRESS_SPAN = 0x00000040
  CPU_RESET_VECTOR = 0x00000000
  CPU_PC = 0x00000000
  # [...]
  def add_objects(self):
    timer = self.add_pre_obj(
       "timer_module", "nios_v_timer",
       freq_mhz=self.timebase_freq_mhz.val
    )
    phys_mem = self.add_pre_obj("phys_mem", "memory-space")
    cpu_core = self.add_pre_obj("hart","riscv-nios-v-m",
    # [...]

    # CPU core parameters fixed by the IP component parameters
    cpu_core.mhartid = 0
    cpu_core.reset_vector = self.CPU_RESET_VECTOR
    cpu_core.pc = self.CPU_RESET_VECTOR

    # Initial memory map contents
    phys_mem.map = [
       [
       self.TIMER_SW_AGENT_BASE_ADDRESS,[timer, "regs"],
       0, 0, self.TIMER_SW_AGENT_ADDRESS_SPAN,
       ]
    ]
    # [...]
```

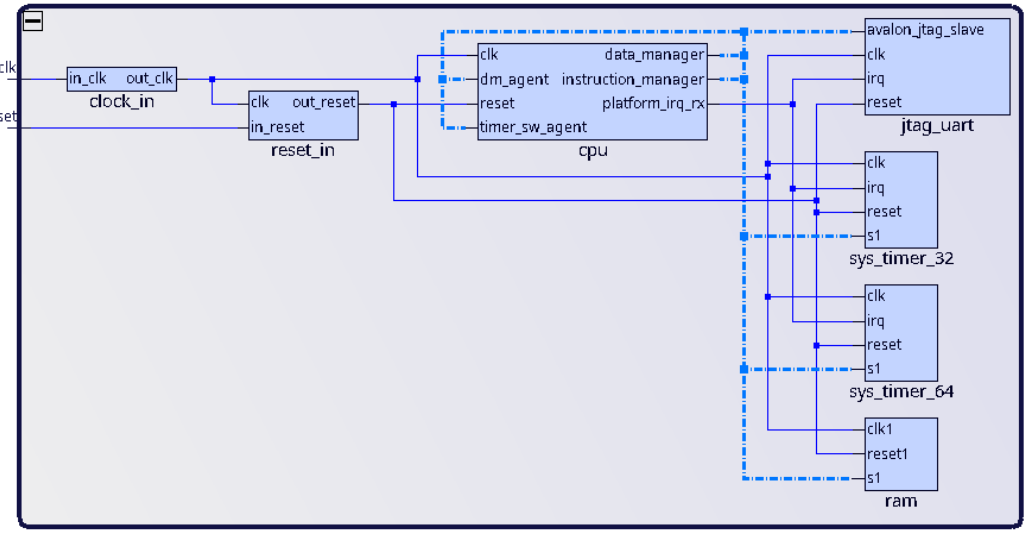
Generated Models

- In general, each IP in the system is implemented as a model in the VP
- Model created using compile time configuration based on IP parameters
  - Like RTL where parameters can have a large impact on the functionality
- Generation of IP creates wrapper for implementation code that defines parameters

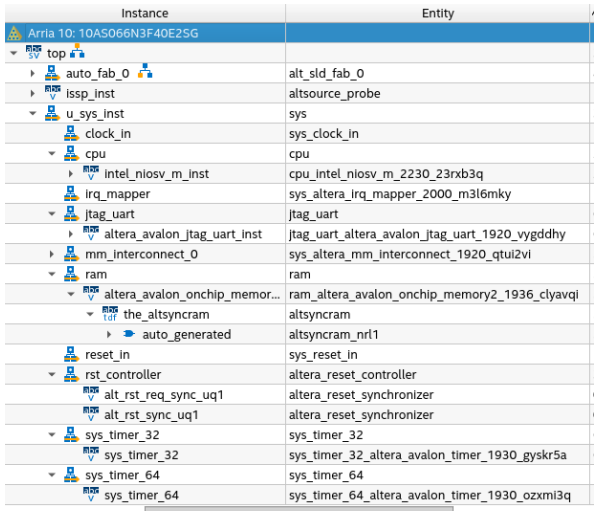
```
bank regs {
   param register_size = 2;

   register control @ 0x02 "Control register" {
      field reseved @ [15:4] is (unimpl);
      #if (no_start_stop_bits) {
         field stop @ [3] is (unimpl);
         #if (system_reset_on_timeout) {
            field start @ [2] is (read, write);
         } #else {
            field start @ [2] is (unimpl);
         }
      } #else {
         field stop @ [3] is (read, write);
         field start @ [2] is (read, write);
      }
      field cont @ [1] is (read, write);
      field ito @ [0] is (read, write);
   }
#if (counter_size == 32) {
   register period[i < counter_size/8] @ ((0x04 << 2) + i*2) is (read, write);
   #if (readable_snapshot) {
      register snap[i < counter_size/8] @ ((0x08 << 2) + i*2) is (read, write);
   } #else {
      register snap[i < counter_size/8] @ ((0x08 << 2) + i*2) is (unimpl);
   }

…
```

Prototype: Result

- Design captured in the FPGA assembly tool produces RTL and matching VP
  - Nested hierarchy that matches the HW design
  - Same SW binary used in HW can be used on generated VP



Vendor Assembly Tool



RTL Hierarchy

```
simics> list-objects -tree -hide-port-objects
└ vp0
    ├ cell
    │   └ ps
    ├ cell_context
    ├ cell_rec0
    └ u_sys_inst
        ├ cpu
        │   ├ hart
        │   │   └ vtime
        │   │       ├ cycles
        │   │       └ ps
        │   ├ phys_mem
        │   └ timer_module
        ├ jtag_uart
        │   ├ jtag_uart
        │   ├ stty
        │   └ stty_console
        │       ├ con
        │       │   ├ frontend
        │       │   ├ tcp
        │       │   └ unix_socket
        │       └ serial
        ├ ram
        │   └ ram
        │       └ image
        ├ sys_timer_32
        │   └ sys_timer_32
        └ sys_timer_64
```

Simics Hierarchy

Summary

- Key challenges in creating VPs for FPGA/SASIC designs are:
  - The end design is created by the customer
  - Design includes a large amount of vendor-provided configurable IP
  - IP parameterization can have large impact to SW visible interfaces
- VP alternatives such as RTL-VP co-simulation and superset models exist, but they do not deliver on the promise of a fast TLM-LT VP
- We have extended an existing vendor tool chain to automatically generate VP configurations that match the captured hardware design
- This solution enables FPGA/SASIC projects to create VPs that match their HW design using existing FPGA design tool flows, without manual VP coding

2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Questions?

accellera SYSTEMS INITIATIVE

2022
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

End

accellera
SYSTEMS INITIATIVE