400G IPU Case Study: Demonstrating Shift-left of Soft Logic RTL Development on Real World Design using Intel Agilex FPGA SystemC Model

Pravat K Nayak, Intel Technology India Pvt Ltd, Bangalore, India (*pravat.kishor.nayak@intel.com*)

Vikrant Kapila, Intel Technology Asia Pte Ltd, Singapore, Singapore (*vikrant.kapila@intel.com*)

Pushpa Naik, Intel Technology India Pvt Ltd, Bangalore, India (*pushpa.naik@intel.com*)

Niketkumar Sharma, Intel Technology India Pvt Ltd, Bangalore, India (*niketkum@sc.intel.com*)

***Abstract*— A new methodology and modeling framework is presented in this work, based on SystemC, which enables a significant shift-left of soft-logic development of Intel Agilex FPGA devices in the context of data center solutions. This approach represents a departure from existing ESL design and verification methodologies that have not been utilized for this purpose previously. We have deployed this methodology in a real world 400G SmartNIC solution implementation. Our methodology helped to shift-left customer soft logic RTL development by 6+ months and uncovered major functional issues to instill customer confidence ahead of FPGA availability.**

Keywords—Cloud Service Provider, Infrastructure Processing Unit, Data Path Accelerator, FPGA soft logic RTL development, SystemC, system-level design and simulation

I. RELATED WORK

The industry and academia offer several C/C++ and loosely timed TLM CPU core models that provide high simulation speed and limited timing accuracy. For example, ARM fast models are instruction accurate ISSs with loosely timed TLM interfaces, which target fast execution of software binaries [1]. These models are instruction and bit accurate, but no details of the micro-architecture are modeled. They are best targeted for early software development, debug, and testing. Industry commercial tools, like e.g., Virtualizer enable companies to accelerate and “shift-left” their software development [2] [3]. QEMU Hybrid Simulations QEMU [4] is an open-source dynamic binary translation emulator supporting many CPU models. The industry typically deploys commercial instruction accurate models while the most popular open-source simulator in academia is gem5. IP modelled in SystemC with TLM base protocol interfaces can be connected to any gem5 IP [5]. All these flows and methodologies were aimed to enable shift-left for SW development. Similarly, there are numerous industrial and academia tools and technologies which have enabled shift-left for definition and optimization of System Architecture for Power & Performance [6]. All above discussed technologies have failed to address shift-left of FPGA soft-logic development for FPGA based data center solution

The public cloud is the backend behind a massive and exponential growing percentage of online software services [7] [8] [9]. These services consume exabytes of storage, petabytes of network bandwidth and millions of processor cores. Software Defined Network (SDN) policies of public cloud change rapidly (weeks to months) so they require a solution that could provide software like programmability while providing hardware like performance. Hence FPGA based Infrastructure Processing Unit (IPU) is the best-suited solution for this requirement. FPGA based IPU enables cloud service providers to constantly evolve, improve and differentiate their data center infrastructure maximizing performance at optimized cost [10].

The FPGA-based IPU has a heterogeneous architecture that includes various hardened blocks such as embedded SRAMs, transceivers, accelerators, and I/O protocol blocks. These blocks are similar to those found in custom ASICs, and therefore a platform is required that allows for parallel development and verification of both the hardened blocks, such as accelerators, within the FPGA-based IPU, as well as the user-side programmable soft logic developed by cloud service providers. However, designing such a platform is a significant challenge as Intel has limited insights into the different data flows that customers may map into the FPGA, while customers find it challenging to develop an accurate soft logic without a model of the configurable data path accelerators present within FPGA IPUs. As a result, cloud service providers have requested a pre-RTL functional model framework that would enable them to develop an accurate soft logic six or more months before RTL availability.

II. APPLICATION

The methodology presented in this paper addresses requirements to shift-left of FPGA soft-logic development for Intel® FPGA® customers. We have demonstrated applicability of methodology and modeling framework using a case study featuring design of next generation SmartNIC device. Our proposed methodology is generic and should be applicable across FPGA industry.

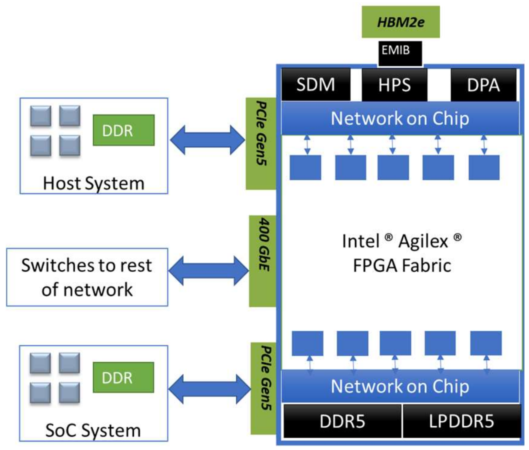

Figure 1 IPU Block Level View

Our team created functional model of the Intel Agilex® FPGA, depicted in Figure 1, using SystemC. The model includes hardened subsystems such as the configurable Data Path Accelerator (DPA). Customer maps custom-soft logic on Intel Agilex® FPGA fabric to form Infrastructure Processing Unit (IPU). As shown in Figure 1, it is connected to Host, SoC and Network as SmartNIC serves networking and storage flows in data center. The model was made available to Intel customers more than 9 months prior to the availability of any device in Intel Quartus® for customer design. We also developed cycle accurate SystemC-RTL transactors to enable RTL co-simulation. Modeling framework allowed our customers' RTL team to work on their programmable soft logic RTL simultaneously with the actual RTL of the accelerator, which is already hardened within the FPGA. In addition to this, our internal pre-silicon verification team utilized the model as a golden reference to reduce functional verification effort and achieve high accuracy in verifying the RTL of FPGA device subsystems e.g DPA.

III. DPA MODEL DEVELOPMENT

DPA model development was aimed to help shift-left the soft logic development before the actual hardened subsystem is available and that is depicted in Figure 2. Apart from minor differences all the three instances of DPA have got similar functionalities in them. The SoC and Host DPA can send or receive data from PCIe Host while the NW DPA sends and receives from network interface.

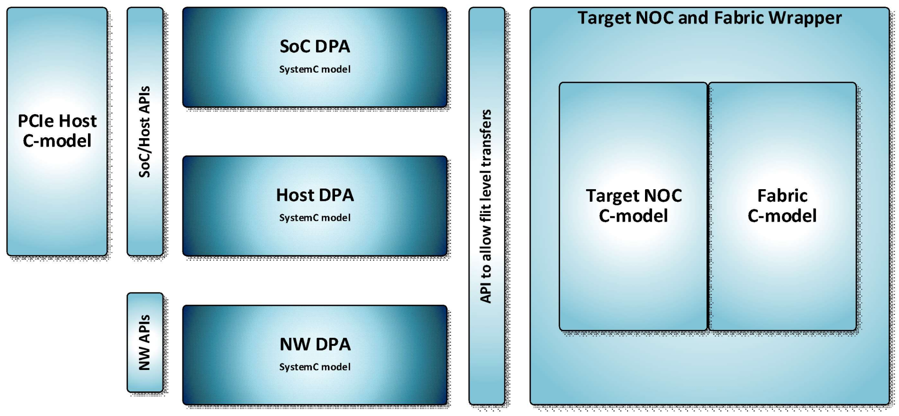

Figure 2 DPA Model TB using APIs

Fabric C-model is where the customer soft logic resides and data communication to individual DPAs are done through the Target NOC model. APIs also enable communication among the DPAs to efficiently transfer data to required DPA without giving additional load to the interface between DPAs and Fabric wrapper. The DPA models along with its initiator and target APIs are developed and validated much before the RTL development started allowing customer to make the development of soft logic functionally accurate and optimized to the hardened subsystem.

During the later part of the project when the customer is ready with their RTL code for soft logic, they would like to validate it without depending on the availability of pre-verified RTL of hardened subsystem. Thus, the second requirement is to enable the DPA model to work in a hybrid testbench environment. As part of this framework, we have developed the required DPIs that enable conversion from System Verilog side (shown in green) to SystemC side (shown in blue) in Figure 3.

Since the DPIs enabled communication from System Verilog side to SystemC model. Thus, the same model was used as golden reference model to verify RTL of hardened subsystem developed within Intel. So the same model helped Intel to validate RTL code of their hardened subsystem while helping the customer to shift-left their RTL development of soft logic.

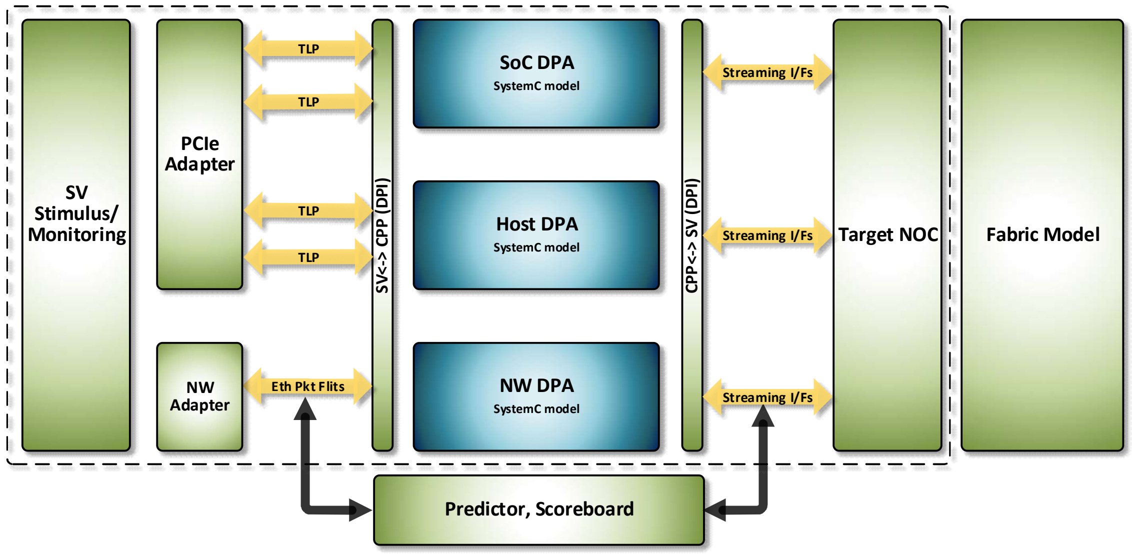

Figure 3 DPA Model in Hybrid Testbench

IV. DPA API DEVELOPMENT

DPA Model is developed using IEEE SystemC/TLM standards, model interface is consistent with SystemC Interfaces i.e. input/output ports and TLM sockets. In order to enable use of this model without SystemC interfaces and know-how of SystemC requirements, C++ APIs support has been provided on top of these interfaces as shown in Figure 2.

There two sets of APIs named as Master APIs and Slave APIs. Initiator/Target terminology is in accordance with DPA Model's use. DPA model will implement target APIs to receive transaction and will invoke Initiator APIs to initiate the transaction.

Note: DPA Model have three configurations i.e. Network, Host and Soc. APIs are provided keeping this configuration incorporated.

There are many APIs available to help configure the DPA and then enable communication in and out of DPAs. The below lists few of the Initiator and target APIs along with its usage.

```
void api_ftnoc_to_dpa_req(dpa_name dpaName, uint32_t physicalInterfaceNo,
ftnocInterface &ftnocDPAReq)
```

Description:

API to send transaction from TileNoc/Fabric interface to DPA Model. As this interface is present for all three DPAs NETWORK/HOST/SOC and there are six (0-5) interface instances within given DPA, User should provide the required input arguments:

1. dpaName -> NETWORK/ HOST/SOC
2. physicalInterfaceNo → 0/1/2/3/4/5
3. ftnocDPAReq ->

Fields	Size	Description
uint32_t tuser_sop:1	1 Bit	Start of packet indicator
uint32_t tuser_valid_bytes:5	5 Bits	# of valid bytes; 0 = 32B
uint32_t tuser_error:1	1 Bit	stream xfer error
uint32_t tuser_metadata:6	6 Bits	Internal metadata description
bool tlast	True/False	End of Packet
unsigned char* data	256 Bits/ 32 Bytes	*Variable size of 32B is mandatory

```
void api_rtnoc_to_dpa_req(dpa_name dpaName, rtnocInterface &rtnocDPAReq)
```

Description:

API to send transaction from TileNoc/Remote DPA interface to DPA Model. As this interface is present for all three DPAs NETWORK/HOST/SOC, User should provide the required input arguments:

1. dpaName NETWORK/ HOST/ SOC
2. rtnocDPAReq ->

Fields	Size	Description
uint64_t tuser_sop:1	1 Bit	Start of packet indicator
uint64_t tuser_valid_bytes:6	6 Bits	# of valid bytes; 0 = 64B
uint64_t tuser_error:1	1 Bit	stream xfer error
uint64_t tuser_metadata:55	55 Bits	Internal metadata description
bool tlast	True/False	End of Packet
unsigned char* data	512 Bits/ 64 Bytes	Payload data *Variable size of 64B is mandatory

```
void api_mac_to_dpa_tx(uint32_t physicalInterfaceNo, macInterface
&macDPATx)
```

Description:

API to send transaction from NIF's MAC interface to DPA Model. This interface is present only for NETWORK DPA but there are two (0-1) such interface instances, User should provide the required input arguments:

1. physicalInterfaceNo → 0/1
2. macDPATx ->

Fields	Size	Description
uint32_t tid	12 Bits	command ID update [11:10] Port [9:0] Transaction ID
bool tlast	True/False	End of Packet
unsigned char* tuser	80 Bits/ 10 Bytes	Internal TUSER description

void api_icq_to_dpa_rx(uint32_t physicalInterfaceNo, icqecqInterface &icqRx)

Description:

API to send transaction from NIF's ICQ interface to DPA Model. This API is applicable only for NETWORK DPA but there are two (0-1) such interface instances, User should provide the required input arguments:

1. physicalInterfaceNo → 0/1
2. icqRx ->

Fields	**Size**	**Description**
uint32_t tid	4 Bits	Transaction index.
uint32_t tdest	3 Bits	Indicates the Logical Port ID associated with the transaction
uint32_t tuser	19 Bits	Internal TUSER description
bool tlast	True/False	End of Packet Indicator
unsigned char* data	512 Bits/ 64 Bytes	Packet Data *Variable size of 64B is mandatory
unsigned char* metadata	256 Bits/ 32 Bytes	Packet Metadata *Variable size of 32B is mandatory

void api_dpa_to_ftnoc_fab_req(dpa_name dpaName, uint32_t physicalInterfaceNo, ftnocInterface &dpaFtnocReq)

Description:

API to send transaction from DPA Model to TileNoc/Fabric interface. As this interface is present for all three DPAs NETWORK/HOST/SOC and there are six (0-5) interface instances within given DPA, User should provide the required input arguments:

1. dpaName -> NETWORK/ HOST/ SOC
2. physicalInterfaceNo → 0/1/2/3/4/5
3. dpaFtnocReq ->

Fields	Size	Description
uint32_t tuser_sop:1	1 Bit	Start of packet indicator
uint32_t tuser_valid_bytes:5	5 Bits	# of valid bytes; 0 = 32B
uint32_t tuser_error:1	1 Bit	stream xfer error

1) api_dpa_to_rtnoc_fab_req

void api_dpa_to_rtnoc_fab_req(dpa_name dpaName, rtnocInterface &dpaRtnocReq)

Description:

API to send transaction from DPA Model to TileNoc/Remote DPA interface. As this interface is present for all three DPAs NETWORK/HOST/SOC, User should provide the required input arguments:

1. dpaName -> NETWORK/ HOST/ SOC
2. dpaRtnocReq ->

Fields	Size	Description
uint64_t tuser_sop:1	1 Bit	Start of packet indicator
uint64_t tuser_valid_bytes:6	6 Bits	# of valid bytes; 0 = 64B
uint64_t tuser_error:1	1 Bit	stream xfer error
uint64_t tuser_metadata:55	55 Bits	Internal metadata description
bool tlast	True/False	End of Packet
unsigned char* data	512 Bits/ 64 Bytes	Payload data *Variable size of 64B is mandatory

void api_dpa_to_ecq_tx(uint32_t physicalInterfaceNo, icqecqInterface &ecqTx)

Description:

API to send transaction from Network DPA to NIF's ECQ interface. This API is applicable only for NETWORK DPA but there are two (0-1) such interface instances, User should provide the required input arguments:

1. physicalInterfaceNo → 0/1
2. ecqTx ->

Fields	Size	Description
uint32_t tid	4 Bits	Transaction index *common for data and metadata interface
uint32_t tuser	1 Bit	Internal TUSER description

Fields	Size	Description
bool tlast	True/False	End of Packet Indicator *Only for data, metadata's tlast is set if it is being sent
unsigned char* data	512 Bits/ 64 Bytes	Packet Data *Variable size of 64B is mandatory
unsigned char* metadata	256 Bits/ 32 Bytes	Packet Metadata *Variable size of 32B is mandatory

V. RESULTS

Our team implemented a methodology and modeling framework for the development of a SmartNIC for data centers. This approach allowed Intel customers to create their own soft logic and perform system-level testing involving interactions between FPGA soft logic and the numerous hardened accelerators present in the Intel Agilex® FPGA. To demonstrate the effectiveness of this approach, we conducted an in-depth study of CSP's data center networking use cases for their 400G IPU solution. Our solution enabled a significant advancement in the customer's RTL development timeline, accelerating it by 6 months or more. Through this process, we were able to uncover over ***10+ major functional*** issues with the command structures and interfaces used for these interactions 6+ months ahead of RTL availability of FPGA device. Moreover, by running over 30 different networking and storage flows with varying traffic characteristics such as packet size mix, traffic burst, traffic class, etc., we were able to instill confidence in the proposed hardened accelerator present within the FPGA.

We plan to deploy the above solution for broad customers as early engagement to optimize customer RTL development on their use-cases. One of the key challenges faced while modeling the hardened ASIC of the heterogenous FPGA architecture was the unavailability of the complete and accurate IPXACT file that lists all the control and status registers until the RTL micro architecture document is available. Since the modeling activities starts much before RTL level micro architecture begins thus decoupling IPXACT from RTL level micro architecture into system level architecture would help further accelerate the overall development cycle.

VI. CONCLUSION

The paper presented a methodology and modeling framework that enables shift-left for soft-logic RTL development for Intel Agilex® FPGA devices. The effectiveness of this approach is demonstrated by a SmartNIC case study to accelerate the soft logic RTL development by 6 months while the RTL development of FPGA device RTL was still under progress. The framework while enabling accelerated time to market for customer, enabled a close collaboration between Intel and customer to refine and optimize interaction between FPGA and customer soft logic. It is important to highlight that 10+ major issues which we discovered early in design cycle were found very late in design cycle of similar previous generation devices leading to either delay in product or failure to realize certain functionality due to functional incorrectness. Future work will focus on making use of the same model within design software like Quartus Prime to speed up functional simulation.

VII. REFERENCES

[1] ARM, "Fast Model," 2018. [Online]. Available: https://developer.arm .com/tools-and-software/simulation-models/fast-models. [Accessed April 2023].

[2] Synopsys, "Silicon Design & Verification - Virtualizer," 2017. [Online]. Available: https://www.synopsys.com/verification/virtual-prototyping/virtualizer.html. [Accessed April 2023].

[3] A. P. a. J. C. T. Holmes, "SoC Development and Prototype with VDK," in 16th International Workshop on Microprocessor and SOC Test and Verification (MTV), Austin, Texas, 2015.

[4] F. Bellard., "QEMU, a fast and portable dynamic translator," in In USENIX Annual Technical Conference (ATEC), Anaheim, CA, 2005.

[5] J. C. M. J. a. N. W. Christian Menard, "System simulation with gem5 and SystemC: The keystone for full interoperability," in International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Pythagoreio, 2017.

[6] D. T. K. a. D. H. K. Vikrant Kapila, "Case-Study: Generating a Workload Model of the Chrome Browser from Android Execution Traces for Early Analysis of Power and Performance Trade-Offs," in *Intel DTTC*, 2017.

[7] Synopsys, "Virtual Prototyping Book," 2019. [Online]. Available: Available: https://www.synopsys.com/verification/virtual-prototyping/vp-book.html. [Accessed April 2023].

[8] Amazon, "Amazon Web Services," 2022. [Online]. Available: https://aws.amazon.com. [Accessed April 2023].

[9] Google, "Google Cloud Platform," 2022. [Online]. Available: https://cloud.google.com. [Accessed April 2023].

[10] A. P. S. M. D. C. a. A. D. Daniel Firestone, "Azure Accelerate Networking: SmartNICs in in Microsoft Research," Microsoft, USA, 2018.

[11] Microsoft, "Microsoft Azure," 2022. [Online]. Available: https://azure.microsoft.com. [Accessed April 2023].