IP-XACT based SoC Interconnect Verification Automation

YoungRae Cho[1] YoungSik Kim[2] Seonil Brian Choi[3]

Samsung Electronics Co., Ltd., Seoul, Korea
([1]yr76.cho@samsung.com, [2]ys31.kim@samsung.com, [3]seonilb.choi@samsung.com).

Abstract- In complex modern Systems on Chip (SoCs), verifying its backbone interconnect with large numbers of bus masters and slaves is one of the most significant challenges. It requires verifying various interface protocols, conversions of different transaction types, and large combinational transaction paths. To cope with this problem, we used Verification Intellectual Property (VIP) for each protocol and a verification platform for interconnect. However, configuring each VIP to connect them with entire master and slave interfaces is a cumbersome manual process. It is also very challenging to update its configuration to comply with frequent specification changes that accommodate market changes. By automatically generating these configurations for the interconnect verification environment using IP-XACT, we can dramatically reduce the time and the effort required for SoC backbone interconnect verification.

Keywords - Interconnect Verification, IP-XACT, Automation

I. INTRODUCTION

To reduce the cost of silicon chips, the SoC industry is trending to merge various functionalities (such as GPS, Modem, Wi-Fi, Bluetooth) into a single chip or Systems on Chip (SoC). Its complexity, as well as its size, increases rapidly. This causes that a backplane (backbone interconnect) in the SoC includes a significantly large number of master IPs (such as CPU cores and other MCUs) and slave IPs (memories or Bluetooth). Thus, it is challenging to verify the backplane considering reachability, routing correctness, memory map, and address space, deadlock-free, performance validation (bottleneck-free), etc. The dedicated backplane verification environment is required where it consists of the backplane RTL and the verification IPs replacing real master and slave IPs. However, it is very time consuming and error-prone to create this verification environment since its test bench is used to be coded manually and every master and slave IPs are configured individually.

To overcome these challenges, the SoC industry shifted the paradigm for SoC integration from a text-oriented coding to an IP-XACT [1] (i.e., metadata) based generation. After the cluster level integration and full chip level integration are performed with the metadata, cluster and full chip RTLs are generated instead of manually coded. The IP-XACT metadata used in SoC integration originally includes only the specification information required for integration. To improve a verification productivity, the IP-XACT based flow is extended to package the specification is necessary for the backplane verification into the IP-XACT efficiently and utilize the specification information to generate a dedicated verification environment automatically. In this paper, the challenges in creating verification environment for the SoC backplane verification are discussed. The approach to resolving them and the enhancement result due to the automation are presented followed by the conclusion.

II. CHALLENGES IN SOC INTERCONNECT VERIFICATION

We have been applying the UVM-based verification environment setup to create the verification environment of SoC backplane, as shown in Figure 1. In this environment, to connect VIPs to the backplane interfaces instead of real IP, we force the IP interface to connect to the VIP. For example, we force interface of IPs such as host CPU

which connected with the slave interface of the backplane to connect to master VIP. And also force slave interface of IPs such as memory controllers or peripherals to slave VIPs.

The use of VIPs for the interfaces which have different protocols enables the generation of transactions through the interconnect which randomly covers the address space and the interconnect routes. In this environment, the protocol validation also can be checked automatically by the VIP included checkers. To verify the end to end transactions, it is needed appropriate system level scoreboard [3]. So we also install Interconnect Validator VIP[4] which is a system-level VIP that monitors transactions within a interconnect fabric and performs checking if the transactions behave correctly.

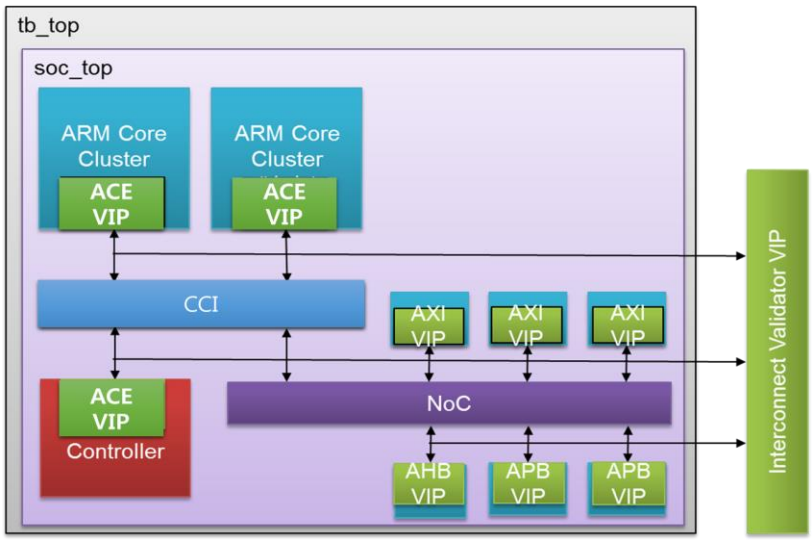

Figure 1. Interconnect Verification Environment

For VIP setup, we need to specify various configuration for each VIP such as a hierarchical location of connection points, interface type, port name and list, bit-width of each port, FIFO depth, multiple outstanding values. Also, we need to specify the spec of SoC backplane such as memory map, address space, decoding, reachability, and so on. To maintain these configurations efficiently, we used to have a spreadsheet-based tool which generates test bench. In the spreadsheet-based flow, we have to make a spreadsheet as shown in Figure 2[2]. For example, If we want to make interconnect verification environment such as Figure 1. We have to describe configurations for 2 ACE master, 3 AXI master, 1 ACE slave, 1 AHB, and 2 APB slave VIPs on spreadsheet format like Figure 2. The ‘Mater interface configurations’ of the spreadsheet should include VIP types such as ACE and AXI3, address width, data width, id width, interface name and so on which needed for VIP configuration for the five master VIPs. We also describe configurations for the four slave VIPs on ‘Slave Interface Configurations’ table on the spreadsheet. The configuration also should include VIP type, port width, name and the address range which the interface has. Moreover, then we have to describe the reachability from the Master VIPs to Slave interfaces on the ‘Master / Salve mapping table’ of the spreadsheet. If a master VIP to a slave VIP is reachable, it is ‘T’ for true or ‘F’ for false.

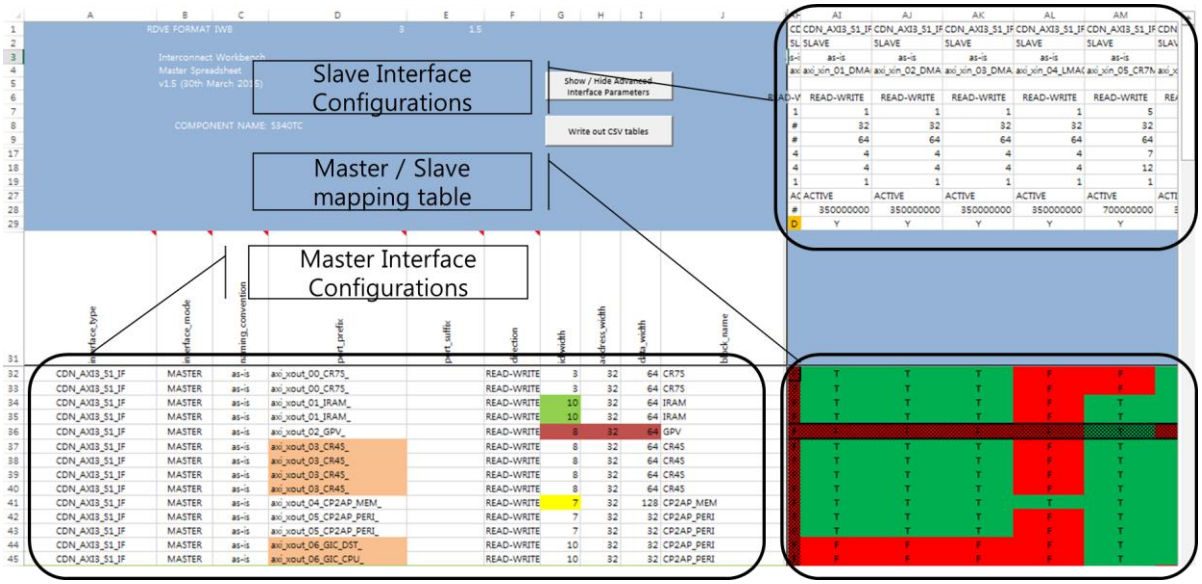

Figure 2. Spreadsheet-based VIP configuration

Once we specify all configuration of VIP and backplane on the spreadsheet manually, we generate verification environment and then we have to connect the generated test bench to the DUT. The port level connection of each interface between VIP and backplane is needed to be done also manually such that the hierarchical location of each VIP instance and signal mapping between the VIP instance and the actual DUT's signal should be described as shown in Figure 3.

```
`define XOUT_17_VPATH   tbench_top.dut.BLK_AUD.AUD_CORE.CTRL
`define XOUT_18_VPATH   tbench_top.dut.BLK_GPS.GPS_CORE.PU
`define HOUT_00_VPATH   tbench_top.dut.BLK_CAM.CAM_CORE.CU

      force axi_ica_xout_18_if0_arid            = `XOUT_18_VPATH.i_AXIS2_ARID;
      force axi_ica_xout_18_if0_arvalid         = `XOUT_18_VPATH.i_AXIS2_ARVALID;
      force `XOUT_18_VPATH.o_AXIS2_ARREADY      = axi_ica_xout_18_if0_aready;
      force axi_ica_xout_18_if0_araddr          = `XOUT_18_VPATH.i_AXIS2_ARADDR + 32'h0408_0000;
      force axi_ica_xout_18_if0_arlen           = `XOUT_18_VPATH.i_AXIS2_ARLEN;
      force axi_ica_xout_18_if0_arsize          = `XOUT_18_VPATH.i_AXIS2_ARSIZE;
      force axi_ica_xout_18_if0_arburst         = `XOUT_18_VPATH.i_AXIS2_ARBURST;
      force axi_ica_xout_18_if0_arprot          = `XOUT_18_VPATH.i_AXIS2_ARPROT;
      force axi_ica_xout_18_if0_arlock          = `XOUT_18_VPATH.i_AXIS2_ARLOCK;
      force axi_ica_xout_18_if0_arcache         = `XOUT_18_VPATH.i_AXIS2_ARCACHE;
```

Figure 4. Example code for instance hierarchy and map between the VIP instance and the actual DUT's signal

From above procedure description, we can easily understand that configuring entire VIPs and typing-in to the spreadsheet is not an easy job since our SoC consists of hundreds of master and slave interfaces should be connected with VIPs for interconnect verification. The amount of routing information we had to define is multiplied by the number of masters and slaves. The codes for the port connection between VIP ports and RTL ports we have to manually type-in is also tens of thousands of lines. Moreover, during the SoC integration progress, SoC spec is frequently changed, which causes RTL modification such as instance name, hierarchy, port name, or routing spec of each backplane component. These SoC spec changes need to be applied to the verification environment also, and this work is not an easy job to comply every RTL change.

It is a hectic, time-consuming, error-prone process, and due to this challenges, verification group faces many incorrect verification failures which also results in the massive amount of waste of engineering resource and sometimes hides real design bugs.

III. IP-XACT BASED VERIFICATION AUTOMATION

As mentioned in the introduction chapter, our SoC integration process was changed from text-based to metadata-based flow, and the metadata is IP-XACT. Originally we included design spec information in IP-XACT, for example, an IP-XACT component for each IP includes port list, bit-width, bus interfaces, memory map, register definition, and so on. In IP-XACT standard, there are set of schema for capturing spec of a bridge, i.e., routing information between its input interfaces (i.e., slave interfaces) and output interfaces (i.e., master interfaces) for interconnect IPs such as CCI and AXI2APB Bridge.

For SoC or subsystem level interconnect verification, all interconnect components should have routing information. However, in design integration flow, the routing information is not needed, and it should be additionally added for interconnect verification.

As shown in Figure 5, the routing information in IP-XACT consists of the input bus interface and the memory map, the subspace map of the memory map, the reference from the subspace map to the output interface, the base address, and address space for the output interface. This description is quite complicated to express routing information for interconnect components with many input and outputs, and it is quite a burden to IP designers.

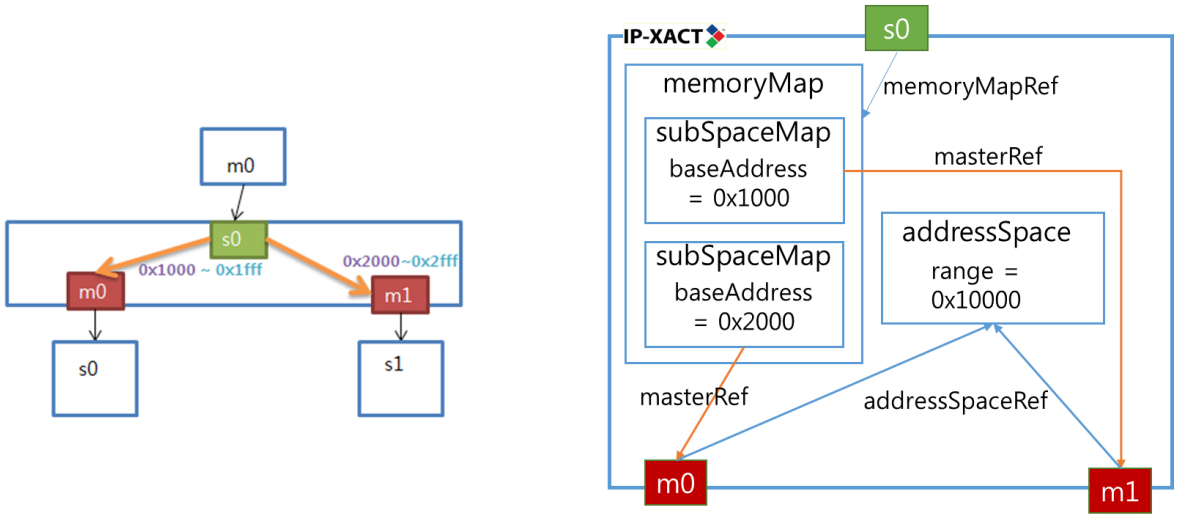

Figure 5. Routing information for Interconnect component.

To resolve this complicated issue, we developed automation script to package routing information into IP-XACT easily for the interconnect IPs.

Most of our system-level interconnects which are connecting sub-system to sub-system or other system-level interconnect to sub-system are automatically generated by a tool. In case of AMBA interconnects like NIC and CCI, they are generated by AMBA generators such as AMBA Designer [5] and the generators also generate IP-XACT which already includes routing information. Our in-house bus generators also generate there routing information as text format. So our script converts the text-based routing information into IP-XACT standard format and integrates into the IP-XACT file for the component as Figure 6.

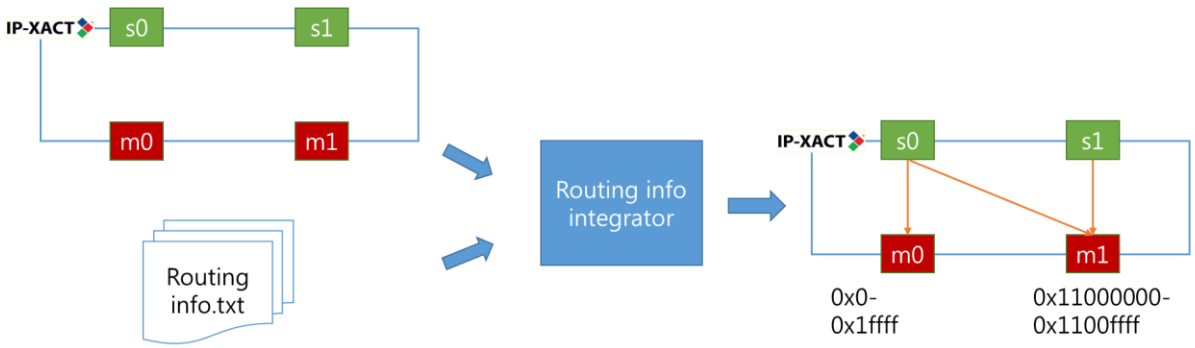

Figure 5. Routing information integration flow with routing information text.

In case of AMBA bridges like 1 by 16 AXI2APB bridge, usually they are used to connect system level interconnect with leaf IPs, they have a full matrix form input bus interfaces to output bus interfaces. The address ranged from the output interfaces are with evenly divided. For example, the 1 by 4 AXI2APB bridge, which input address range is from 0x0 to 0xffff, the address space range for each output interface is 0x4000. For this kind of bridges, the routing information can be added to input address range like Figure 6.

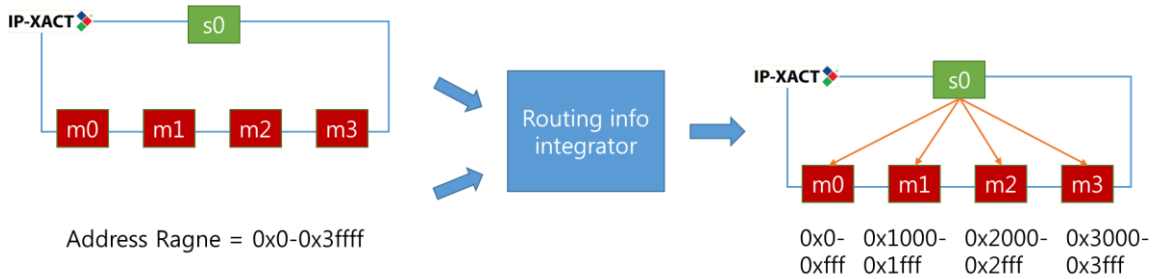

Figure 6. Routing information integration flow for AMBA bridges.

In case of one to one bridges such as protocol changer like AXI2APB, the full range of address for input interface is directly mapped to output interfaces. In this case, the routing information can be easily added by the script like Figure 7

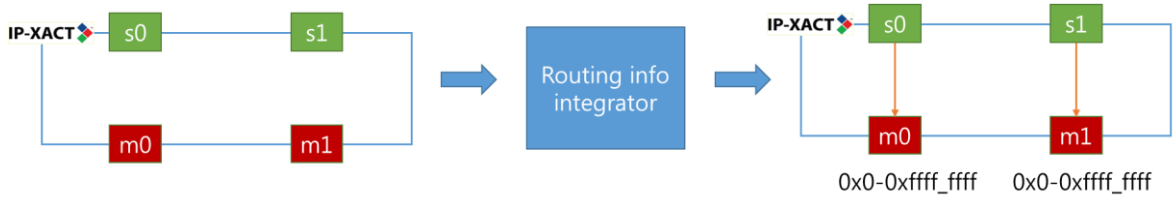

Figure 7. Routing information integration flow for one to one bridges.

In our last SoC project, routing information for 120 of 130 interconnect components are added by our automated script.

For the rest of interconnect we also used GUI based routing information integration tool [6] for designers who are not familiar with IP-XACT as Figure 8. To make routing information, using the GUI editor, designers just click input slave interface to make memory map. And click output master interface to make address space with range like 0x10000. And then click and drag the address space to the memory map with base address.

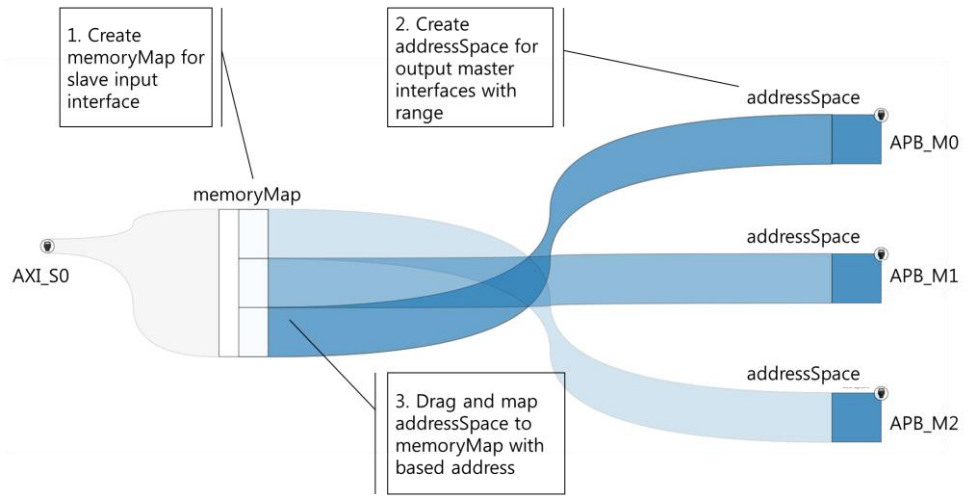

Figure 8. GUI editor for IP-XACT routing information

During the SoC integration process, each IP is instantiated in subsystems, and each subsystem is also integrated into top-level SoC. After the integration, the IP-XACT data for each subsystem and top-level SoC already contain connection specification such as port mapping and bit-with, hierarchical architecture of SoC and configurations. Using this SoC IP-XACT data, reachability(routing) and memory map from any master interface such as CPU to any slave interface such as DRAM or IP register path can be calculated and interconnect verification environment can be generated automatically by using this information.

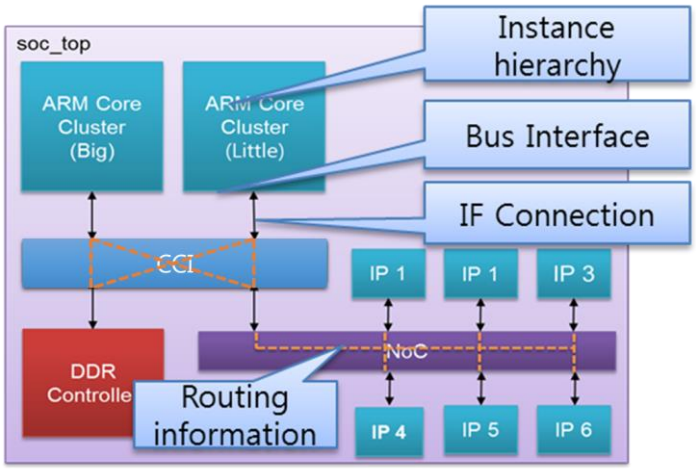

Figure 9. SoC level IP-XACT information for interconnect verification.

We collaborated with EDA and built this automated flow [5] to generate interconnect verification environment without any human intervention. Using this flow, we just load IP-XACT for the integrated SoC design on the tool and then select master and slave interfaces which should be connected to verification IP on GUI as Figure 10. Then all manual work of configuration for every VIP and SoC interconnect specification does not necessary. The IP-XACT database includes each interface information such as hierarchy, protocol type, master/slave mode, port mapping, port width and routing information from any master to any slave. So it is not needed to check the design specifications to update verification environment and configure verification environment and port mapping manually. So, whenever SoC RTL is updated during the integration process, the SoC IP-XACT data is also updated and we can quickly re-generate interconnect verification environment based on the IP-XACT.

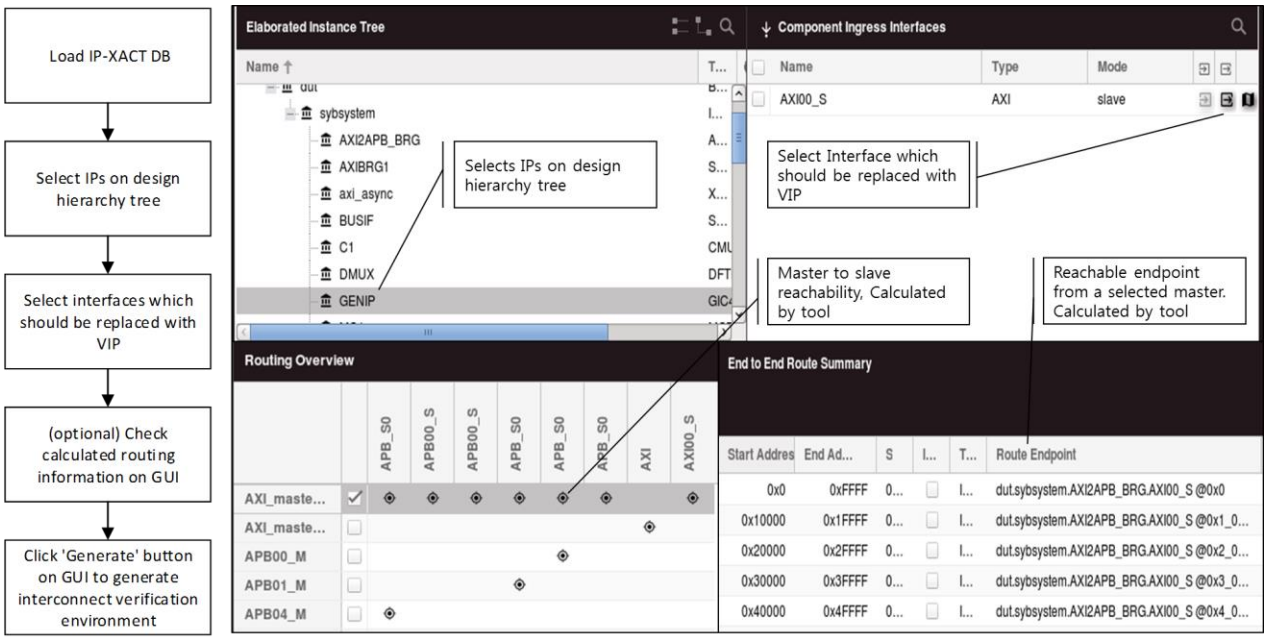

Figure 10. Working flow and GUI of IP-XACT based interconnect verification environment generation

IV. RESULT

Our recent SoC interconnect verification environment includes more than 50 master and 400 slave interfaces which connected with VIPs. It took over two weeks to create an initial version of interconnect validation environment by the previous manual process. However, IP-XACT-based automation shortens this work into only one day. In manual processes, just end-to-end interconnect instances (i.e., leaf IP's interface) was supposed to be connected with verification IPs. In IP-XACT-based automation, however, any interfaces inside any hierarchical-level of SoC can be selected as verification point to be connected with Verification IP. IP-XACT based verification environment automation also enables to generate multiple test bench for subsystem or SoC and different test bench behavior. In our project, we can generate more than 5 different TB for SoC and each sub-system.

These automation reduced time to test bench modification to comply architectural or design dramatically. So we also can generate new test bench for every RTL update at an early stage of the project, which happens one or two times for every week. As a result, we obtain less number of false-negative report, time to run more tests, finding more interconnect bugs in the early stage of the project.

V. CONCLUSION

In this paper, we introduced our experience for verifying complicated SoC level interconnect verification. We already developed interconnect verification environment using interface VIP and interconnect validator VIP [4] and excel based automated flow. However, the configuration of every VIP to connect them into DUT's entire master/slave interface is an entirely manual process. Moreover, it is difficult to update the configuration to comply DUT speciation change because the number of masters and slaves are enormous and yet the DUT speciation is frequently changed during the SoC integration process. Thanks to IP-XACT based design integration flow, we can get much information from this metadata for interconnect verification environment. However, routing information for all interconnect component which is needed for interconnecting verification should be added in the IP-XACT, and it is a little burden for design integration flow. So we developed a method to add this information for each

interconnect components. We can automatically generate interconnect verification environment based on the IP-XACT, and it dramatically reduced our turnaround time to setup interconnect verification environment. As a result, we obtain less number of false-negative report, time to run more tests, finding more interconnect bugs in the early stage of the project.

REFERENCES

[1] IEEE. *"1685-2009 IEEE standard for IP-XACT, Standard Structure, Packaging, Integrating, and Reusing IP within Tool Flows."*.

[2] Cadence Interconnect Workbench. [Online] *https://ip.cadence.com/ipportfolio/verification-ip/interconnect-solution/interconnect-workbench*

[3] François Cerisier and Mike Bartley, "SoC Interconnect Verification Challenge," Design & Reuse, 2014

[4] Cadence Interconnect Validator. [Online] *https://ip.cadence.com/ipportfolio/verification-ip/interconnect-solution/interconnect-validator-coherent*

[5] AMBA Designer user guide [Online] https://developer.arm.com/products/system-ip/corelink-interconnect/corelink-network-interconnect-family/docs/100459/latest/system-ip-tooling/amba-designer

[6] Cadence Verification Workbench User Guide