Automated RTL Update for Abutted Design

Wonkyung Lee[1], Ayoung Kwon[2], Soyeong Kwon[3], Youngsik Kim[4], Seonil Brian Choi[5]

Samsung Electronics Co., Ltd., Seoul, Korea
([1]wonkyung.lee@samsung.com, [2]ah0.kwon@samsung.com, [3]soyeong.kwon@samsung.com,
[4]ys31.kim@samsung.com, [5]seonilb.choi@samsung.com)

Introduction: The most important factors to be considered for SoC(System on Chip) chip design are performance, power, and area. Among them, the absolute factor directly related to cost of chip manufacture is the area. Therefore, new technologies to reduce chip area are being continuously researched. One of the various technologies to reduce chip area is to design abutted floorplan. Abutted floorplan can reduce area by eliminating top level routing area in the chip. For abutted design, RTL(Register Transfer Level) should be updated for reflecting design modification requested by physical designers, without human error, in a short time. This paper introduces the RTL update automation methodology for abutted design.

Challenges: Due to huge size of mobile AP(Application Processor) SoC design, the limitation of EDA(Electronic Design Automation) tool capacity and computing resources, full chip integration is performed by divide-and-conquer. Therefore, SoC full chip design is divided into a lot of function blocks and the integration of each function block is performed by block integration engineer. Function block designer is in charge of creation of block-level RTL, sanity check of the generated RTL like lint, low power design, performance/area estimation and so on. After doing the works, RTL of function block is released to SoC top designer to make SoC top RTL for next chip design step such as physical implementation and function verification.

For abutted design, RTL update requests coming from physical designers are to create *feed-thru* which is a connection to be passing through multiple function blocks, and to move function logics of SoC top into multiple function blocks. Hence, RTL modification regarding abutted design should be performed by many block integration engineers simultaneously because it is possible to make SoC top design after all function blocks are developed. Moreover, the RTL modification should be rapidly executed not to cause any delay of chip production because the RTL modifications according to abutted design are requested at late stage of chip design and the amount of RTL that needs modification is huge. Also, if there is mismatch between function blocks, SoC top design is difficult to complete due to the mismatch. These mean that automation of RTL update for abutted design is inevitable.

Metadata Based SoC RTL Integration Automation: In order to develop RTL design automation flow, there are several ways to modify the Verilog based RTL design. However, if the amount and complexity of design modification is huge, it is very difficult to modify Verilog RTL codes directly without converting Verilog RTL design into design metadata.

A representative metadata technology in the semiconductor industry is IP-XACT. IP-XACT is IEEE Std 1685 as IEEE standard and XML (eXtensible Markup Language) schema that defines and describes electronic components and their designs. Goals of IP-XACT are to enable exchanging of complex component libraries between EDA tools for SoC design, to ensure delivery of compatible component descriptions from multiple component vendors, to describe configurable components using metadata, and to enable the provision of EDA vendor-neutral scripts for component creation and configuration [1]. Also, it is possible to develop various automation methodologies because IP-XACT is machine readable. There are seven main contents in IP-XACT as bus definition, abstraction definition, component, abstractor, design, design configuration, and generator chain. In our design flow, bus definition, abstraction definition, component, and design has been used. Bus definition includes a name of bus such as AXI, AHB and logical port list which is composed of the bus definition is described in abstraction definition. Component includes IP specification information such as port list, bus interface, SFR (Special Function Register), file list, and so on. In design, instance and connection information are included. Fig. 1. show an example of IP-XACT component.

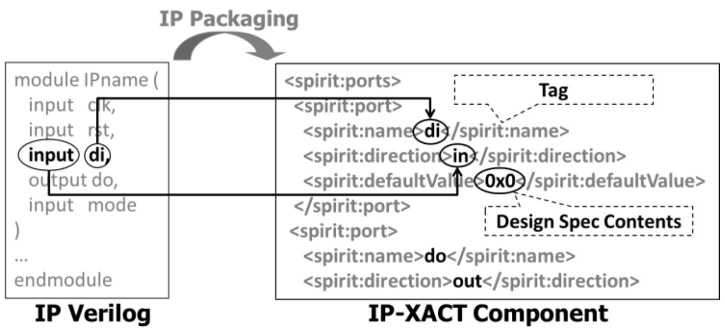


Fig. 1. IP-XACT Component Example

We are using IP-XACT in integration flow for generating RTL for function block and SoC top. Fig. 2. shows SoC RTL integration flow using IP-XACT.

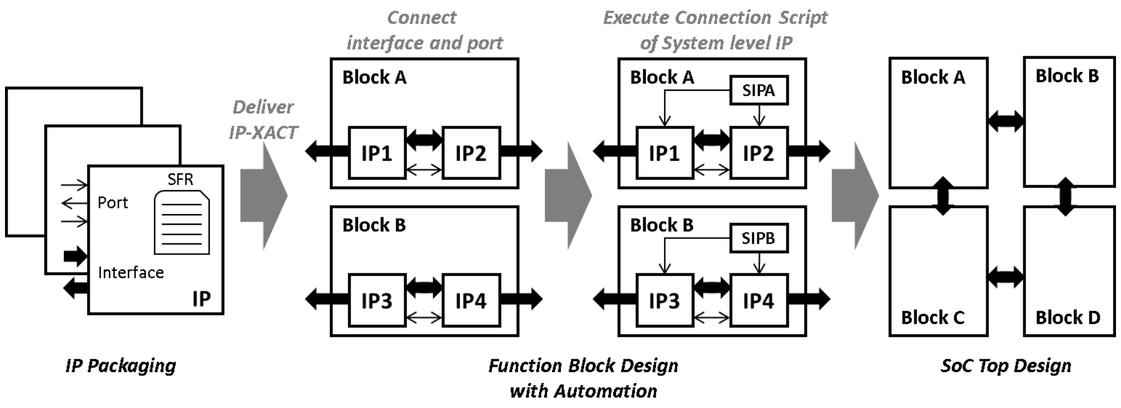

Connect interface and port
Execute Connection Script of System level IP
Deliver IP-XACT
SFR
Port
Interface
IP
Block A
IP1
IP2
Block B
IP3
IP4
SIPA
SIPB
Block C
Block D
IP Packaging
Function Block Design with Automation
SoC Top Design

Fig. 2. SoC Design Flow using IP-XACT.

The First step of RTL design flow is to make IP-XACT component of IP using IP packaging flow. During IP packaging, a description of the IP port list is performed and added into IP-XACT. Also bus interfaces are added into IP-XACT by mapping logical ports defined in abstraction definition and physical ports existed in the actual IP. These IP packaging task are performed by IP designers. IP-XACT components of all IPs which are used in a SoC project should be delivered for SoC top-level integration because IP-XACT component of IP is a mandatory deliverable for integration. IP-XACT of several IPs such as legacy or purchased IP is made by SoC integration team. After obtaining the all IP-XACT components from IP design teams, function blocks would be integrated as IP-XACT design. In IP-XACT design, there are two kinds of connection such as interface and port. Interfaces are connected through interface connection between IPs. When the interface-connection is performed, logical ports with same name as the port name defined in the abstraction definition are physically connected. Interface connection covers 50% of overall connection in terms of port in a SoC design. The remaining ports are connected through port connections called as adhoc connection. In adhoc connections, there are inter-IP connections and connections to constant values such as 0 and 1. Generally in a SoC design, there are thousands of interface connection and over 50,000 adhoc connections. Therefore, automation methodology for adhoc connection is essential to prevent human errors and to reduce TAT(Turn-Around Time).

Most of adhoc connections are related to system level IP. System level IPs control chip level operation such as clock and power management, test mode, testmux, interrupt and so on. We developed various RTL design automation methodologies based on IP-XACT metadata for the system level IP, such that a design modification, like IP instantiations and adhoc connections. They can be performed by executing the modification script on the metadata-based design platform. A specification of system level IP is described in dedicated editor and function block designers describe IP information related to the system level IP using the editor. After collecting information of all IPs, system level IP designers generate RTL of their system level IP from the specification and also they make IP connection scripts for the system level IP integration at function block. The developed RTL codes and connection scripts for system level IP are delivered to function block and SoC top designers. Then function block designers execute the connection script for integration of system level IP. In case of SoC top design, almost ports are connected by interface-connection between function blocks and other connections related with system level IPs are performed by the connection scripts. After completion of SoC top integration, RTL codes for SoC top design are delivered to physical and verification engineers. For example, at the previous design flow for testmux IP integration, function block integration engineers connected all ports manually by referring integration guide documents. As a result, a lot of time was spent for IP integrations, and for resolving a lot of design issues caused by human errors. However, in current flow, if someone describes test mode specification of SOC design, this specification is automatically converted into several scripts that can be executed for generating testmux RTL codes, instantiating testmux IP, and port connecting between testmux IP and other IPs. And then, each function block designer executes the generated scripts to integrate the testmux into their own function block. At the SoC top integration, using an updated function block, SoC top integration can be executed with the generated scripts. Once all designs are updated with execution of the scripts, metadata of SoC design would be updated and then RTL codes of SoC top and each function block would be generated from the metadata-based design platform. Due to these automation methodologies using IP-XACT as metadata, it's possible to modify RTL in a very short time without any human errors.

Abutted Design: Fig. 3 shows overall design flow for abutted design.

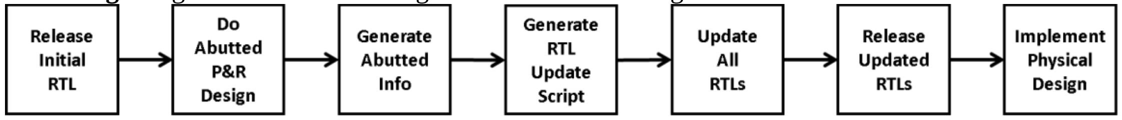


Fig. 3. Design Flow for Abutted Design.

First deliverable for abutted design is initial RTL codes of function blocks and SoC top with top level function logic. Physical implementation engineers synthesize the delivered RTL codes and perform P&R (placement & route) using the synthesized netlists. In this step, top-level channel routes are existed for routing of top level function logic such as testmode control IP, data communication modules between function blocks, and so on. After fixing locations of the function blocks, the top level function logics are moved into adjacent function blocks considering physical timing. If all top level function logics are relocated into multiple function blocks, the relocated information related with abutted design is delivered to RTL designer for updating RTL codes of all function blocks and SoC top. To modify RTL codes of function blocks and SoC top, top-level logic designers generate connection script to reflect the abutted information to RTL codes. The generated connection scripts are automatically executed at function blocks and SOC top design to modify the RTL codes. The updated RTL codes are delivered to physical implementation engineers. To make successful implementation of abutted design in time, automation flow for RTL updating without any error in a short time is essential because this process must be repeated whenever P&R is changed. Furthermore, P&R can be changed frequently in SoC project.

RTL Update Item: Fig. 4 shows various RTL design modifications for abutted design.

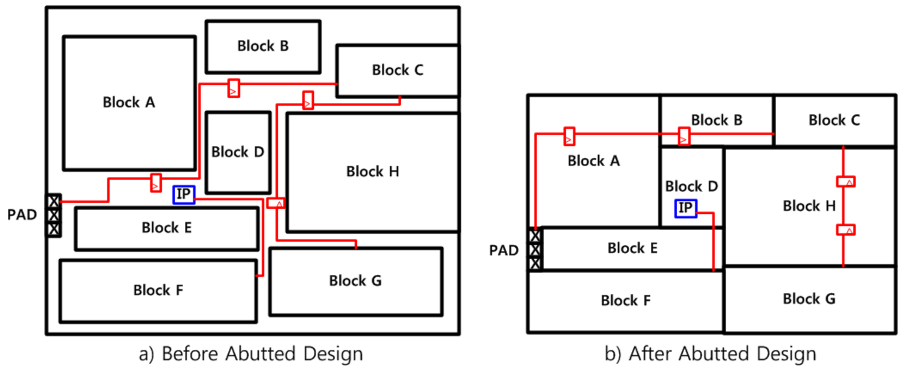


Fig. 4. Design Modification for Abutted Design.

RTL design modifications due to abutted design are as follows.

Test mode control IP movement: In our SoC design, test mode control IP which generates test mode selection signal is located at top level like IP with blue line in Fig. 4. Typically, number of test mode in SoC design is around

50 and number of function block is around 30. Therefore, there are many connections between the test mode control IP and function block. For abutted design, the control IPs should be moved into a function blocks located in the center of the chip. And this movements lead RTL updates of SoC top and function block design.

Data communication between blocks: There is data communication between function blocks. To exchange data between function blocks, the data could be communicated through top-level channels. When use top-level routing channel, to satisfy physical timing conditions, additional logic such as register slices could be added to SoC top design. In order to manage register slices at the port group level, we developed register slice IP and it is designed to be configurable for setting various timing conditions without design changes.

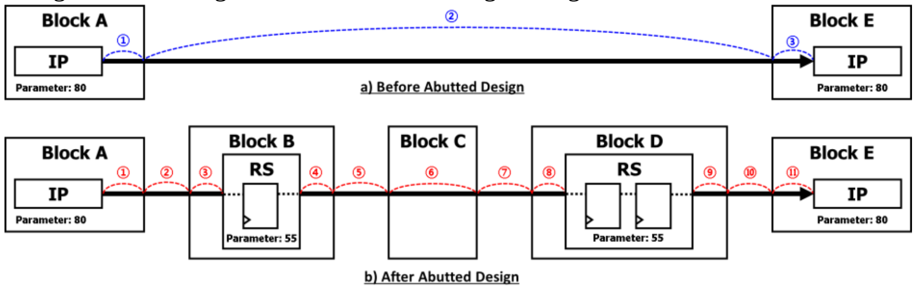


Fig 5. Before/After abutted design connection related to data communication.

Fig. 5 shows RTL codes update example regarding data communication due to abutted design. Prior to abutted design, data between Block A and Block E could be directly connected using three interface connections. However, after applying abutted design, eleven interface connections and two instances of register slice module in Block B and D are needed for the data communication without timing violation. Also, many parameters should be set in register slice IPs to avoid timing violation.

Test signal from pad: For chip test, some IO pads are directly connected to function blocks. To meet physical timing for test, a module including flip flops is added on path between pad and function block. For abutted design, the flip flops should be moved into function blocks. Unlike the register slice modules for data communication between function blocks, the flip flops should be respectively separated due to different glue logic related to each flip flop. Due to the separation of the flip flops, function blocks and SoC top design should be updated as Fig 6. Before applying abutted design, all flip flops for timing satisfaction are existed in a module. After applying abutted design, all flip flops are relocated into function blocks and many connections should be created at function blocks and SoC top.

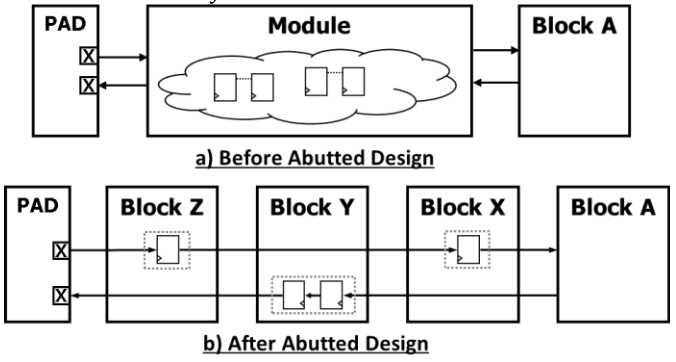


Fig 6. Before/After abutted design connection related to test signal.

RTL Update Automation Flow: Fig. 7 shows RTL update automation flow.

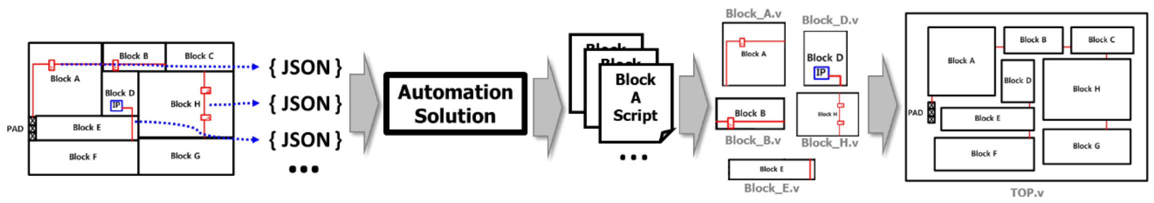


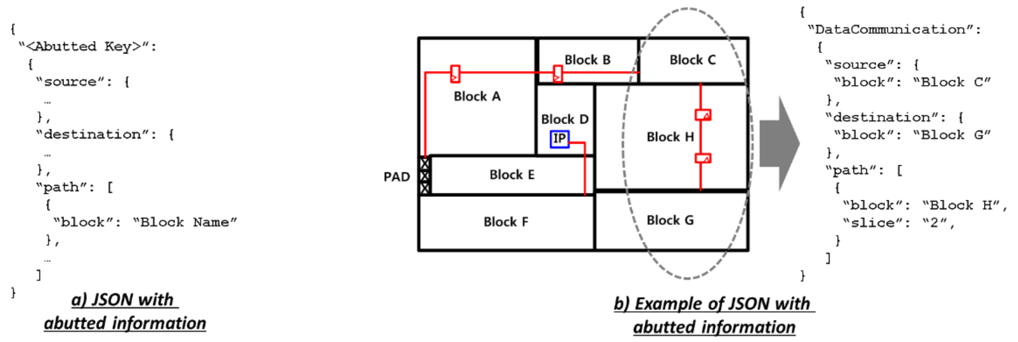
Fig. 7. RTL Update Automation Flow

To maximize the efficiency of automations for abutted design, we defined the metadata format for RTL modification requests coming from physical designers, using JSON. JSON(JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. [2]

After implementing abutted design, JSON file with abutted information should be delivered for RTL update by physical implementation engineer. In the JSON file, there should be the following items as Fig. 8 (a):

- Abutted Key: Abutted key of the JSON file is one of RTL update items for abutted design. The RTL update items are test control IP name, data communication IP name for register slice, and IP name related to test signal connection from pad.
- Source: In this section, source means a start point of a path.
- Destination: Destination means end point of the path.
- Path: Path is a block list to pass from source to destination.


```
{
 "<Abutted Key>":
  {
   "source": {
    …
   },
   "destination": {
    …
   },
   "path": [
    {
     "block": "Block Name"
    },
    …
   ]
}
```

```
{
 "DataCommunication":
  {
   "source": {
    "block": "Block C"
   },
   "destination": {
    "block": "Block G"
   },
   "path": [
    {
     "block": "Block H",
     "slice": "2",
    }
   ]
}
```
a) JSON with abutted information
PAD
Block A
Block B
Block C
Block D
IP
Block H
Block E
Block F
Block G
b) Example of JSON with abutted information

Fig. 8. JSON template and example with abutted information

As Fig. 8 (b) is shown, direct connection between block C and E in Fig. 4 (a) is changed to pass connection through block H due to abutted design. The change should be described in JSON file. Then, these metadata of requests are processed by our automation solution, resulting in scripts for modification of both function blocks and SoC top. All automation solutions were implemented with Python language. Python libraries for extracting abutted information from the JSON file and generating integration scripts with template engine were used in the automation solutions. The generated scripts for all function blocks are delivered to function block designers, the scripts are automatically executed at final step of function block integration. After completion of function block integration, function block RTL codes which are updated with abutted information are released to SoC top designers. RTL codes of SoC top design is created by executing connection script at top level and delivered to physical and verification engineers for chip implementation and verification. Also, to apply all update automation solutions in multiple function blocks at the same time, version of all function blocks should be matched. For version matching, all connections for abutted design are tagged with unique version when connection scripts are executed. Version mismatch could be prevented by checking the version whenever RTL codes of function block are delivered for SoC top integration.

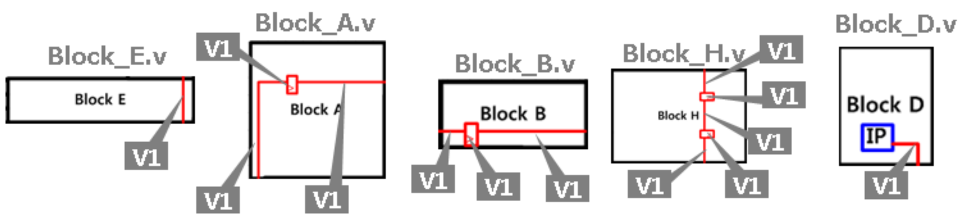


Fig. 9. Version Tag on Connection

Result: As the proposed automation solution was applied, Verilog RTL codes were modified for abutted design. Table 1 shows the comparison result of the number of port connections before / after the application of the automation solution.

	# Port Connection		Change Rate	Update Time
	Before	After	(Times)	(Second)
Test mode control IP	2157	4314	▲2.0x	1112s
Data communication	10240 (128 IF)	75520 (944 IF)	▲7.3x	162s[1)]
Test signal from pad	181	1248	▲6.9x	371s

1) Data channel is connected by bus interface(IF)

Table 1. Number of Connections between before and after abutted design

The number of port connection related to test mode control IP became double. Port connections for data communication between blocks and port connections to transmit test signal from pad are increased by 7.3 times and 6.9 times, respectively. For data communication connection, bus interfaces are used and a bus interface connection covers 80 port connections. Even though the number of port connections is increased by at least two to seven times, the all new port connections are automatically created for abutted design within thousands of seconds. Finally, RTL codes for function blocks and SoC top could be created without error.

In terms of verification, function verification of SoC design is performed at RTL level without abutted information because there is no functional change due to abutted design and abutted information is delivered after completion of physical implementation. After verifying RTL codes without abutted information, functional consistency between original RTL codes and abutted design RTL codes in terms of connection is checked by equivalence check technology.

Conclusion: In this paper, RTL automatic updating solution is proposed. To reduce the area of SoC, an abutted design is required, but a large amount of RTL code modification is required. Moreover, since it is necessary to apply the abutted design very frequently in the late design stage, it is difficult to apply the abutted design as a manual method. In order to prevent human error and to reduce development time, this solution is proposed and developed, and the effect is verified by applying to the actual mobile AP SoC project. As shown above, the number of port connections increased significantly, so RTL code changes in function blocks and SoC top should be made significantly, but RTL changes could be made without any human error due to the proposed solution. Therefore, RTL can be modified without increasing TAT for abutted design.

Reference:

[1]: IP-XACT, https://en.wikipedia.org/wiki/IP-XACT
[2]: Introducing JSON, https://www.json.org