Automation Methodology for Bus Performance Verification using IP-XACT

Taeyoung Jeon, Gunseo Koo, Youngsik Kim, Seonil Brian Choi
Samsung Electronics Co., Ltd., Hwasung-si, Korea
(ty.jeon; gunseo.koo; ys31.kim; seonilb.choi@samsung.com)

***Abstract*-Bus fabric of Modern SoC is becoming massive as the number of DMAs (Direct Memory Access) is increasing. Thus, a bus performance verification is required to measure the performance of each master which reflects the congestion of the bus fabric. However, preparation for the verification needs huge amount of time and implies the possibility of human error as the bus performance test must be processed in a full-chip level environment to simulate the bus fabric and a large number of verification targets are distributed in the design. Here, we propose an automation methodology to reduce the TAT (Turn Around Time) of the bus performance verification by using IP-XACT as a meta-data of design. With this methodology we have experimentally demonstrated that over 90% of verification targets are automatically processed and TAT is reduced to 1 minute from 2 weeks. We expect to apply this methodology to all projects which requires the bus performance verification.**

I. Introduction

Modern SoC is challenging to provide multiple features in a single system. To achieve this goal, SoC designers focus on integrating more application specific hardware in the chip. In such circumstance, the number of bus components such as MMU (Memory Management Unit), bus arbitration logic and asynchronous bridge increases as more DMA masters are implemented. In the aspect of each master, it is important to know how their performance is affected by the bus components on the data path. However estimating the impact is not straightforward.

As a solution, a bus performance verification is introduced to measure the performance of all masters while the bus components are influencing the data transaction. The bus performance verification specifies bandwidth, latency and MO (Multiple Outstanding) as a performance metrics and measures them at the pre-silicon level while all of the bus components are in operation. The bus performance verification is executed under the emulation environment, because it requires full-chip level environment where all of the bus components are fully implemented without an abstraction. In addition, the test measures over a few gigabytes per second which means that cycle-based simulation is excessively time consuming. Considering a large DUT (Design Under Test) and fast run time, emulation environment suits for the bus performance test.

Approximately over hundreds of the masters are the verification targets, so it is barely impossible to bring-up each master. In the aspect of the bus performance verification, internal algorithm of the master is not the verification target. Outgoing traffic is what the bus performance verification is focusing on. In such circumstances, the bus performance verification uses TG (Traffic Generator) to replace each master. By using TG, user can avoid the bring-up of each master and can ensure the controllability of the test environment. Fig. 1 shows the TG replacement of real masters in the SoC. Every master which is a verification target is replaced by TG for the test.

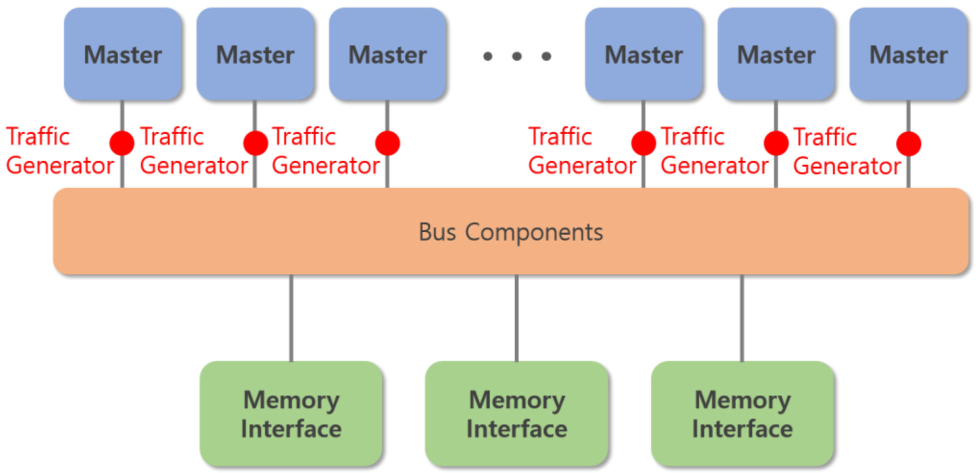


Figure 1. TG replacement of verification targets in the SoC.
Multiple layers of bus components are abstracted as a single block

The problem of the bus performance verification is that the preparation of TG replacement takes a lot of time. TG generates data transactions based on protocols like AXI or ACE, so all of the signals consisting the protocol must be known to the TG. A TG database file is used to describe design information such as signal hierarchy and bit-width of signals. The TG database file is delivered to the emulation image builder which replaces the masters to the TG based on the TG database file. When we need multiple TGs in a single emulation image, we must write down all signal information of every TG to database file. For example, when a single TG uses 50ea signals and we try to use 100ea of TGs then database file should have 5000ea signal information. If we manually process this job, it would be tedious and time consuming. In addition, manual work might cause any human errors. Errors in the TG database file are not detected at the emulation image build step, they are observed by inaccurate result while the test is performing.

This study suggests automation methodology for TG replacement to reduce verification TAT and human errors. This methodology applies IP-XACT to obtain the signal information. IP-XACT provides meta-data of the design. Therefore various kind of verification methodologies utilize IP-XACT for automation [1]. In this methodology IP-XACT plays significant role because the DUT (Design Under Test) is large as a full-chip sized SoC which means that parsing meta-data requires less effort while reading the whole RTL (Register Transfer Level) code is much harder.

II. Background

A. *Bus Performance Verification*

Recent SoCs are adding new features into their system to meet the increased demand of the market. Typically, SoC is designed with a cascade of bus components, each of which has a complex algorithm and configuration. When integrating bus components into the SoC, it is essential to ensure that the masters connected to the bus components are served with their targeted minimum latency and bandwidth.

This kind of verification is very challenging in some points. First, to measure the performance of the masters, a huge amount of data transactions is tested which is above a few gigabytes per second. Second, the verification must be done under full-chip level environment that all of the bus components are fully included in the test-bench which affects the test run time. Third, the test must be done for all of the masters in the SoC. Usually there are over a hundred of masters in the SoC, which makes the test-bench complicated.

In such situation, the bus performance verification is established to measure the performance of each master. It uses emulation environment and synthetic traffic patterns. Emulation environment is applied to obtain the fast runtime compared to the simulation environment [2]. Emulation takes advantages in running large DUT and in measuring heavy traffics in a timely manner. The synthetic traffic patterns are used for convenient control of traffic patterns. TG is used to generate the synthetic traffic patterns. By applying TGs to all verification target, the verification TAT can be reduced significantly. Figure. 2 shows the concept of the bus performance verification. The verification measures the performance metrics of the data path from the master to memory interface.

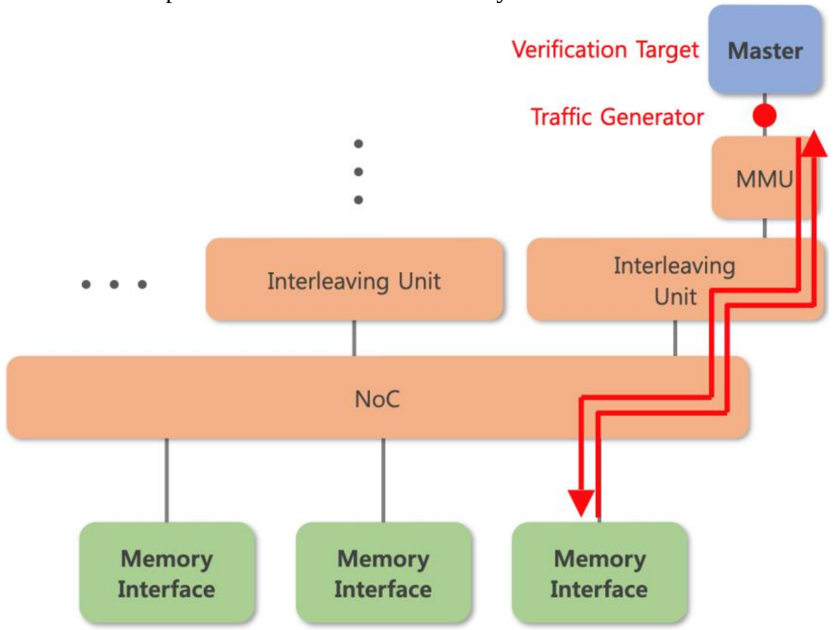


Figure 2. The concept of the bus performance verification
The verification measures performance metrics of the data path

Information about the verification target is delivered through the bus specification. The bus specification contains information of each verification target and its property like clock frequency, minimum latency, bandwidth and port hierarchy.

B. Traffic Generator

Traffic generator is a component that can send synthetic traffic instead of masters for the purpose of emulation-based verification. Figure. 3 shows the diagram of TG that replaces the master. When master is replaced by TG, real master does not have to send its traffic and TG imitates the data traffic of the real master by driving the signals of the protocol. TG makes the verification much easier because the initiating master does not have to operate. Unlike the real masters, TG does not require any vectors for the operation and can provide convenient generation of the data transactions. This characteristic is much helpful when multiple TGs are applied to the DUT, because the user can save the bring-up time for each master.

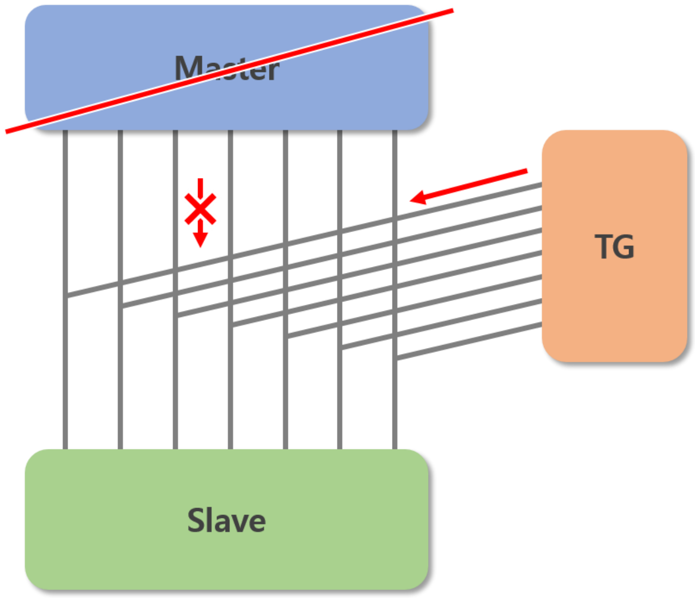
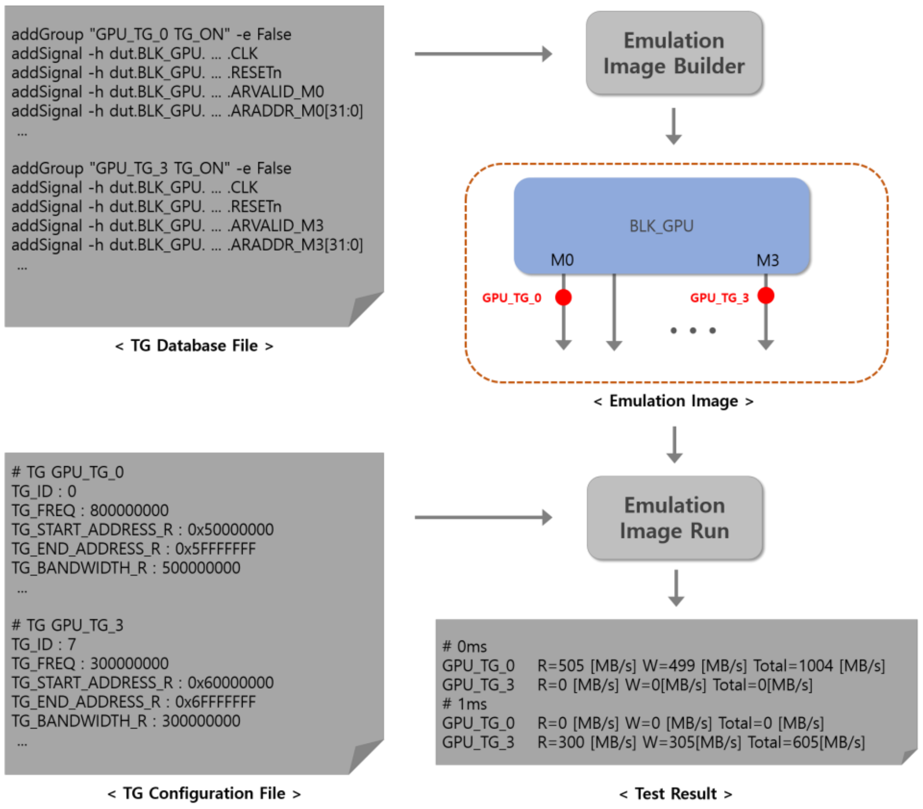


Figure 3. TG replaces the real master and starts the traffic instead of the master

To use the TG, two files are required. The first is a TG database file which describes all the signals required by the TG. Though the TG sends the traffics based on the protocol such as AMBA AXI and ACE, the user must prepare all of the signals related to the protocol. The TG database file is delivered to the emulation image builder for the further steps. The second is a TG configuration file, which determines the transaction property. The configuration can contain target bandwidth, destination address range, QOS value and cache hint. The test-bench reads the TG configuration file while the test is running. Fig. 4 shows the entire flow of TG included emulation verification. TG database file is used for emulation image build and the TG configuration file is used for test run.



```
addGroup "GPU_TG_0 TG_ON" -e False
addSignal -h dut.BLK_GPU. … .CLK
addSignal -h dut.BLK_GPU. … .RESETn
addSignal -h dut.BLK_GPU. … .ARVALID_M0
addSignal -h dut.BLK_GPU. … .ARADDR_M0[31:0]
 …

addGroup "GPU_TG_3 TG_ON" -e False
addSignal -h dut.BLK_GPU. … .CLK
addSignal -h dut.BLK_GPU. … .RESETn
addSignal -h dut.BLK_GPU. … .ARVALID_M3
addSignal -h dut.BLK_GPU. … .ARADDR_M3[31:0]
 …
```

```
# TG GPU_TG_0
TG_ID : 0
TG_FREQ : 800000000
TG_START_ADDRESS_R : 0x50000000
TG_END_ADDRESS_R : 0x5FFFFFFF
TG_BANDWIDTH_R : 500000000
 …

# TG GPU_TG_3
TG_ID : 7
TG_FREQ : 300000000
TG_START_ADDRESS_R : 0x60000000
TG_END_ADDRESS_R : 0x6FFFFFFF
TG_BANDWIDTH_R : 300000000
 …
```

```
# 0ms
GPU_TG_0    R=505 [MB/s] W=499 [MB/s] Total=1004 [MB/s]
GPU_TG_3    R=0 [MB/s] W=0[MB/s] Total=0[MB/s]
# 1ms
GPU_TG_0    R=0 [MB/s] W=0 [MB/s] Total=0 [MB/s]
GPU_TG_3    R=300 [MB/s] W=305[MB/s] Total=605[MB/s]
```

Figure 4. Emulation based verification using TG.

III. Proposed Approach

This methodology suggests the automatic generation of TG database file and TG configuration file for the bus performance verification. Before this methodology was introduced, bus specification, bus architecture diagram, and RTL code were used to generate the database and configuration files. The user picks the verification target on the bus specification file, then manually searches the approximate location of the master port on the bus architecture diagram. Finally, with the location found in the bus architecture diagram user must search precise hierarchy of verification target at the RTL code. These steps were time consuming and inadequate for defining huge amount of TG information which can easily cause human errors. The biggest problem of the manual work was ambiguous expression of verification target on the bus specification and the bus architecture diagram. Bus specification did not define the full hierarchy of the verification target, they only mention the IP (Intellectual Property) name which the verification target is implemented. Also, the bus architecture diagram is a Visio file (.vsd file) which shows simplified interconnecting relationship of IP inside the SoC. Moreover, the bus architecture diagram was drawn by hand which does not guarantee the latest status of the RTL code. So, the user must read and analyze the RTL code to get the exact hierarchy of the verification target. To solve this time consuming and unreliable workflow, proposing methodology was invented. In the previous methodology, the bus specification only provides the IP name that the verification target is included, by now it is changed to provide the full hierarchy of port. This little change was very helpful to increase the accuracy of automation results.

To run the methodology, two input files are required. The first file is the bus specification file which is in excel format. The methodology reads the excel file without any additional processing. This feature is helpful when errors are found in the file because user can easily re-run the methodology file by simply editing the specification. The second input file is the IP-XACT.

As a result, of this methodology, the TG database file and TG configuration file is generated. TG database file requires the signal lists from IP-XACT and port hierarchy from the bus specification file. TG configuration file is generated with traffic configuration requirement for example bandwidth, burst size and length from the bus specification. Fig. 5 shows a simplified diagram which explains data input and output of our proposed methodology.

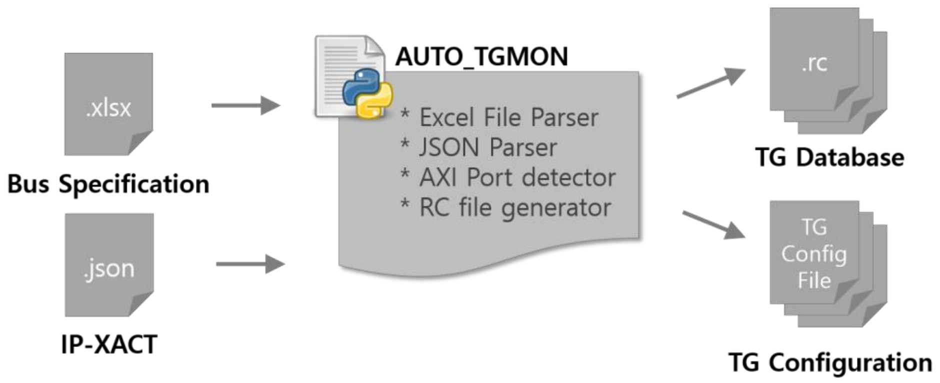


Figure 5. Input and output diagram of the methodology
Name of the methodology is "AUTO_TGMON"

The methodology is composed of three steps: extraction, find intersection and output generation. Fig. 6 shows more detailed diagram of the methodology. The extraction stage reads the input files, find intersection stage, compares the input data from both the input files. With the result from the find intersection stage, the output generation stage creates the output files.

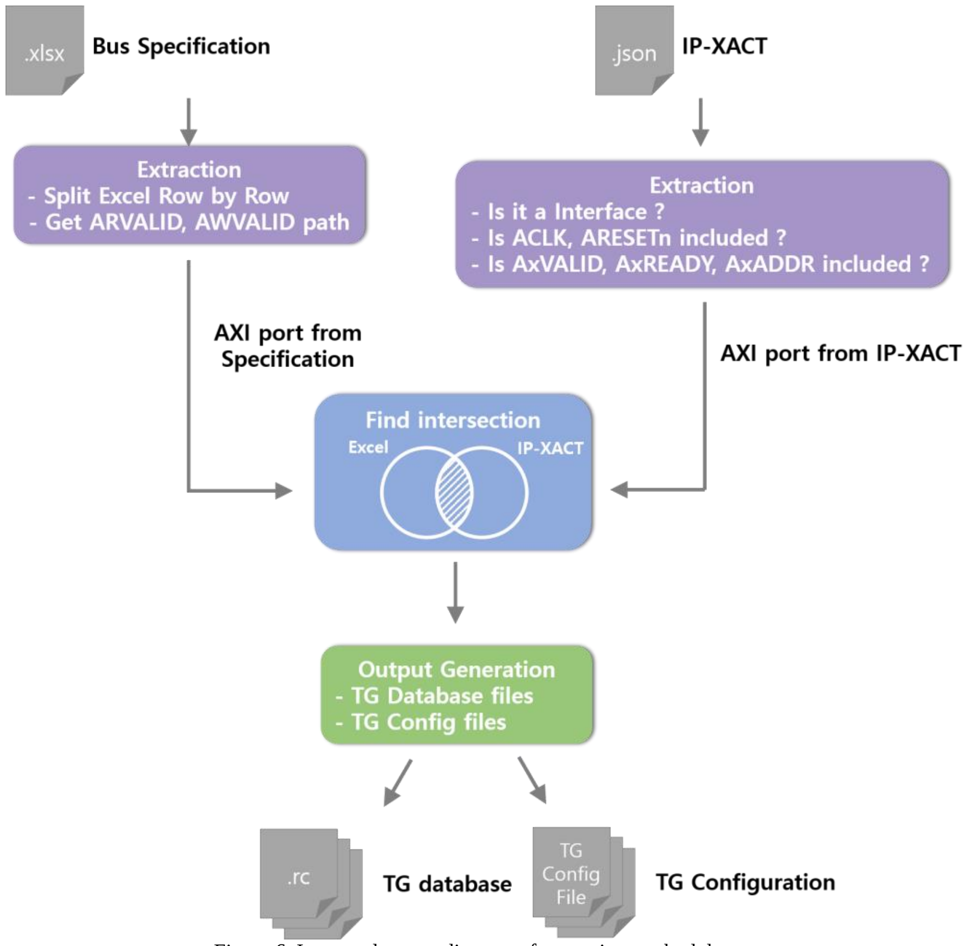


Figure 6. Input and output diagram of proposing methodology

At the extraction stage, the bus specification and the IP-XACT are parsed into a class instance of object-oriented programming language which makes the methodology to be minimally modified even if the format of the input is dramatically changed. The bus specification is delivered in excel format. Each row contains the information about the verification target which is the IP name, required bandwidth and clock frequency. In this step minor errors can be reported to the user. For example, empty or not a number in the excel file. Moreover, depending on the property, exceptionally out of range values can be reported as an error. Table. I shows an example of the bus specification file. Table. I is a simplified example and many other properties are contained in the actual table. ARVALID, AWVALID signal path is used as a representative for the port hierarchy. Cells marked in red will be reported as an error. Max freq. of MODEM DMA0 is reported as an error because 10 MHz for the clock frequency is exceptionally a small value. The other error cases are empty cells and not a number. Errors in the extraction stage can be ignored by the user depending on the property of the verification target. For example, some master port can only provide read feature, in this case write signal hierarchy has nothing to be written.

TABLE I
AN EXAMPLE OF THE BUS SPECIFICATION

Block	IP name	Max freq. (MHz)	I/F type	Minimum Required Bandwidth (GB/s)		Signal Hierarchy	
				Read	Write	Read	Write
GPU	GPU_D0	1200	ACE	10	5	dut.BLK_GPU. ... .ARVALID_M0	dut.BLK_GPU. ... .AWVALID_M0
	GPU_D1	800	ACE	8	8	dut.BLK_GPU. ... .ARVALID_M1	dut.BLK_GPU. ... .AWVALID_M1
	GPU_D2	800	ACE	0	10	dut.BLK_GPU. ... .ARVALID_M2	dut.BLK_GPU. ... .AWVALID_M2
MODEM	CPU	1200	AXI4	3	3	dut.BLK_MODEM. ... .ARVALID_CPU	dut.BLK_MODEM. ... .AWVALID_CPU
	DMA0	10	AXI3	1	N/A	dut.BLK_MODEM. ... .ARVALID_M0	
	DMA1	800	AXI3	1	1		dut.BLK_MODEM. ... .AWVALID_M1
	MOVER0	800	AXI3	1	1		dut.BLK_MODEM. ... .AWVALID_MOV

At the find intersection stage, two data groups from the different inputs are compared with each other. The point of the comparison is the signal hierarchy information. Since the bus specification only holds the ARVALID and AWVALID hierarchy, only VALID signals are compared between the bus specification and the IP-XACT. Find intersection stage is required to find the verification targets which is available for the automation. In later stage of the methodology ARVALID and AWVALID signal hierarchy from the bus specification is restored to full signal list containing ARADDR, ARREADY, and many other signals in the protocol by using meta-data extracted from the IP-XACT.

Comparison results are arranged into three different cases. 1) The verification target from the bus specification is found at the IP-XACT. This case means that the verification target is available for the automation. 2) The verification target from the bus specification is not found in the IP-XACT. This case is an error with some possibilities. The first possibility is that the IP designer does not use the IP-XACT. Unfortunately, there is nothing to do with this verification target. The other possibility is an error of the bus specification. By the fact that bus specification is updated by hand, errors can happen. Whenever there are major updates in the RTL code which changes the signal hierarchy, the bus specification is not guaranteed to reflect the RTL changes, whereas the IP-XACT are updated automatically. These errors in the bus specification are reported to the user. 3) The last case is that the master port is found in the IP-XACT, but not found in the bus specification file. The methodology does not care about this case because these ports are not the verification target. Master ports of the bus component correspond to this case, for example, MMU, bus arbitration logic and CDC (Clock Domain Crossing) logic.

At the output generation stage, two output files are created. Signal list for the TG database file is obtained from the IP-XACT, traffic settings for the TG configuration file are parsed from the bus specification.

IV. EXPERIMENTAL RESULTS

We applied this methodology to the test environment for development. Test was done for the 190 of masters in the design, the methodology successfully generated the output files within one minute. Also, the methodology got 22 errors, eight were from the extraction stage which is detected by reading the bus specification file. These errors on the bus specification can be fixed with little effort. The other 14 errors occurred from the find intersection stage. These errors occur when the verification target does not use the IP-XACT or the bus specification does not reflect the updated RTL. This requires the modification of the bus specification, after the deliberation with design team the verification target can be changed according to the RTL update.

After the test we finally applied this methodology to the latest design of mobile device. 155ea of the verification target were chosen for the bus performance verification. By applying this methodology, the TG database file and TG configuration file were generated within 1 minute, which was 2 weeks of work in the previous project. As a result, the bus performance verification was able to obtain the test result in an early stage of verification. Also, the bus performance test can find 17 bugs. As a workaround the bus specification changed target performance metrics or the burst length and clock frequency is adjusted to meet the performance requirement.

V. Conclusion And Future Works

We have successfully developed the methodology for the automatic generation of TG database file and TG configuration file for the bus performance verification. Also, the methodology was applied to the latest project which reduces the TAT from 2 weeks to less than 1 minute. This result shows that the methodology successfully achieved the goal.

Moreover, since the TG database file and the TG configuration file has no dependency with the bus performance verification, this methodology can be applied to all verification items which uses the TG.

References

[1] Kim, Taejin, et al. "Developing a Portable Block Testbench and Reusable SOC Verification Scenarios Using IP-XACT Based Automation." DVCon, 2020

[2] Sawant, Sanjay, and Paul Giordano. "RTL emulation: the next leap in system verification." Proceedings of the 33rd annual Design Automation Conference. 1996.