International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2019): 7.583

New Feature Verification Simplified with Automated Verification Component Build, Centralized Test Plan and Guidelines for DUT Boundaries and Integration Strategies

Ankit Chandankhede

Abstract: ***Modern applications demand newer architectures and enhanced feature to process tasks at high speed. Such architectural changes or newer mega features necessitates modification to existing architecture or completely newer architecture and often affects different design units of the architectures. Monumental changes across the architecture introduces numerous controls, finite state machine, newer pipeline, data paths and configurations. These heavy changes are highly prone to architectural gaps and design bugs. Thus, different pre silicon verification strategies are absolute to meet essential to not only meet the high functional quality of design as well as performance. Verification of new mega features faces numerous challenges such as comprehensive testplanning, bringing up of these new features and executions of test plan. Addressing these challenges is critical to flawless execution to achieve high quality of design for A0 production and meet the timelines to market. This paper proposes an alternative approach to existing pre - silicon verification of mega new features such as multi - context, ray tracing and multiple compute pipeline in addition to existing compute and 3D shading pipelines of graphics architecture [1]. Our novel approach provides improvements in every step of the pre - silicon verification stages including test planning, feature integration, bring up, building for API, test constraint for building basic testcases and protocol related coverages based on the protocol definitions of packet information to finding bugs in early stages of design such as interoperability and exposing architectural gaps. This paper details challenges in verification of process and precise steps to overcome these challenges using automating scripts, regression management and effective debugging strategies.***

Keywords: presilicon verification, architecture changes, feature integration, design challenges, debugging strategies

1. Introduction

Different verification methodologies such as simulation, emulation, formal, gate level simulation performance and post silicon verification to ensure the quality of design in terms of functionality and performance [4] [3]. Each verification strategy can be deployed at appropriate level of DUT due to its advantages and disadvantages which are explained later in this paper, thus deploying particular pre - silicon verification strategies at particular DUT to find the bug early in design cycles allows to converge on functionality as well as performance of architecture and design. This paper focuses on improving pre - silicon verification for new mega feature verification. Approach deploys automation on creating testing constraints, base sequences based on the protocol description, automated register testing, building and reusability of APIs and verification components including BFM, coverages, scoreboards and checkers, integration of design containers, defining synchronous DUT boundaries for effective verification and centralized test planning.

2. Implementation

Centralized Test planning:
Verification of an architecture or product is stages at different levels of verification DUTs, ranging from verification at IP, sub system, super sub system and SOC [5]. Each DUT and team usually creates test plan based off of the architectural and design specifications which are tailored to a particular DUT. Unit level and subsystem test plans are often covers more extensive testing. It covers scenario pertaining to a unit level design specification but may lack testing from system level point of view as it focuses in boundary checks and internals of unit's scenarios such as finite state machines, flushing of internal cache/SRAM, clearing internal temporary registers and stressing of credits on interfaces.

Whereas subsystem and System on Chip (SOCs) DUTs focuses on system level scenario and interoperability across units or subsystems. Cross feature scenarios are also covered extensively from system level point of view. Mega Feature enabling such as multi - context or raytracing at higher subsystems across other features is critical to ensure the protocols, interoperability and architectural assumptions are functional and meets the performance.

Due to differences of scenarios at different levels of DUT, there is chance of missing scenario between system level to unit level testing as feature testing at unit as opposed to SOC may show different sequencing than real time workload and hence synchronizing system level scenarios and sequencing at SOC and Unit level is critical. This can be achieved by creating system level scenarios per feature, protocols and data paths. Further system level feature or testcases in graphics processors can be controlled by submitting system command in particular sequence which allows test sequences to be matching the real - world scenarios. Thus, system level scenarios and test sequencing can be scaled to different levels of DUT and allows reusability.

Since the system level testcases can be run at unit level, it provides opportunity to clear the functionality of feature at IP or subsystem level and hence reduces the debugging and bringing up cycle at SOC [9].

Verification component building and reusing strategies:
Verification testbenches at different level such as IP, subsystem and SOC has different boundaries of designs and often shared same boundaries of design and other part of designs are mimicked through behavioral functional model (BFM), UVC which includes monitors and drivers. Drivers models the protocol to drive the packet information on to interface of design whereas monitors collects the information from the interface of design which can be used to in slave BFM to respond back with appropriate response or collect coverage or feed to checkers for checking the integrity of the packet information from design. This infrastructure can be shared and reused as boundaries of DUTs are same or shared similar protocol such as AXI.

As each interface follows specific packet information and protocol and monitor and drivers needs specific class based on struct, constraints, enumeration data types and memory components. These protocols and packet information can be easily combined to automatically create enum, constraints [6] [7], packet information and thus sequence items using script which provides compile clean and early head start to build sequence items and structural code for building systemverilog or UVM based API and UVCs.

For confidentiality the concept will explained using hypothetical protocol

Hypothetical Memory packet information (Field)	Subfields	Dependencies	Valid values	Width
Cmd			READ=0, WRITE=1, ATOMIC=2	3: 0
Address		Cmd==ATOMIC	0 - 3ffe_ffff	31: 0
Address		Cmd==SYS_MEM_WRITE, Cmd==SYS_MEM_READ	fff0_ 0000 - 4000_0000	
Address		Cmd==READ	0 - 3fef_ffff	
Address		Cmd==WRITE	0 - 3fef_ffff	
SIZE		Cmd==ATOMIC	2: 5	3: 0
SIZE		Cmd==READ	0: f	
SIZE		Cmd==WRITE	0: f	
Vld			1	
Data			All	255: 0
Datavld			1	
Data_byte_enable		Cmd==ATOMIC	0 - fff0	
		Cmd==WRITE/READ	0 - ffff	
System_Command	Cmd_type		Flush=0, Compute=1, pixel_shader=2, geometry_shader=3, Vertex_shader=4	3: 0
System_Command	Cmd_vld		0: 1	1
System_Command	Cmd_header		All	255: 0

Above table 1. Shows the standard tabulating field of the packet information as follows:

- Fields are listed with its dependencies, valid values and width.
- The dependencies of fields within the packet information on specific value of field to be constraint to a specific values are mentioned in valid values column.
- If the values mentioned in valid value column are “All”, all values are valid for that particular field.
- Width of the fields
- If a field has subfield, subfileds are considered as an entry of Field and script creates struct

Enumeration type generation:
Script generates enumeration as follows:
typedef enum bit [3: 0] {READ, WRITE, ATOMIC, SYS_MEM_READ, SYS_MEM_WRITE} Cmd_e;
typedef enum bit [3: 0] {Flush, compute, pixel_shader, geometry_shader, vertex_shader} Cmd_type_e;

Struct generation:
If a field includes subfield, that field is considered for a struct and automatic structs are created as follows with specific subfields mentioned until the next entry is mentioned in field entry.
typedef struct {
Cmd_type_e cmd_type; bit Cmd_vld;
bit [255: 0] Cmd_header;
} System_command_e;

Packet /Sequence item structure with constraints:
Based on the table provided, script generates the automated packets with fields and subfields with constraint and dependencies which provides compile clean and base sequence items for sequences or API [7].
class packet;
/* **Field decleration:**
Fields are created based off the table with enum and struct created in previous steps //
*/
rand bit [31: 0] Address; rand Cmd_e Cmd;
rand bit [3: 0] SIZE; rand bit vld;
rand bit [255: 0] Data; rand bit Datavld;
rand bit [15: 0] Data_byte_enable;
rand System_command_e System_command;
/***Constraints:**
Constraints are created with dependencies and value expected for mentioned values of the fields.
Following autogenerated constraints shows address range being constrained based on the type of command as has been reflected from table 1.
*/
constraint Address_c { solve Cmd before Address;
(Cmd==ATOMIC) - > Address inside { [0: 32'hffe_ffff]};

```
(Cmd==SYS_MEM_WRITE) - > Address inside { [32'hfff0_0000: 32'h4000_0000]};
(Cmd==SYS_MEM_READ) - > Address inside { [32'hfff0_0000: 32'h4000_0000]}; (Cmd==READ) - > Address inside { [0: 32'h3ffe_ffff]};
(Cmd==WRITE) - > Address inside { [0: 32'h3ffe_ffff]};

}
constraint vld_c { vld ==1;
}
constraint SIZE_c { solve Cmd before SIZE;
(Cmd==ATOMIC) - > SIZE inside { [0: 4'hf]};
(Cmd==READ) - > SIZE inside { [0: 4'hf]};
(Cmd==WRITE) - > SIZE inside { [0: 4'hf]};
}
constraint Data_byte_enable_c { solve Cmd before Data_byte_enable;
(Cmd==ATOMIC) - > Data_byte_enable inside { [0: 16'hfff0]};
(Cmd==WRITE || Cmd==READ) - > Data_byte_enable inside { [0: 16'hffff]};
}
endclass
```

API building:

System commands are used in compute architectures by Firmware in order to submit a particular workload. Firmware uses different software languages such as C++, C sharp, java scripts and hence integrating different frameworks for interactive firmware with presilicon testbenches such as simulation and emulation raises complexity of testbench environment and demands expertise in different languages. In order to overcome this challenge, system level firmware commands should be modelled in either SystemVerilog or UVM as these test frameworks are used in most of the testbenches due to its universal acceptance. Automated frameworks propose in this paper provides base for creating systemverilog based function task to create system commands and write into system memory which is also modelled in systemVerilog.

As shown in earlier examples, automated framework provides enumerated and struct data types concentrated on type of system command mentioned in architectural or firmware definitions. These commands are put into format of table 1 and script thus is able to be created struct and enumerated types. These struct and enum data type are integrated in individual classes of each type of commands to randomize different fields of the commands based on the restrictions put in table 1. Randomized commands are submitted into the system memory from which the design fetches when the system addresses are provided to design through different register programming or explicit write sent to MMIO space depending on the design specifications.

Since system level APIs can be used at different levels of DUTs, test created based on system level API can be scaled and reused at different DUTs which refences into system memory for commands. Furthermore, these system level tests can be scaled to stress the pipelines. For example, Compute and 3D pipeline uses similar system commands and are independent pipelines in graphics architecture however shares same execution pipelines which includes arbiters, execution units, instruction cache and system cache, thus submitting independent system level commands at same time for compute and 3D engines provides an opportunity to stress the shared resource by graphics 3d shaders with compute shaders [2].

Integration:

In order to create efficient verification environments, each DUT must be able to run test and debug faster, following aspects should be taken into account.

a) Boundary of the DUT
b) Size of the design under testing environment
c) Number of BFMs required
d) Scalability of testcases across DUTs
e) Checker and scoreboard scalability across DUTs

Boundaries of the DUTs should be decided based on the factors as mentioned above. Particularly graphics architecture includes multiple pipelines, shared resources across pipelines and protocols. In order to keep nimble DUT, design under testing environment should be configurable and can be achieved by chopping the architecture into multiple clusters. Clusters should be configurable as well and can be managed by creating the containers of designs ad pipelines which can be instantiated multiple times using configurable options and thus DUT can be scaled from single instances of design or pipelines to multiple instances. Configurability of DUT results in quicker compilation of DUT and quicker testing time. Further the boundary of the DUT should be considered from design verification component perspective such that verification component can be shared across DUTs and thus reducing the development cycle of component significantly. These components can be reusable as it follows the standard protocols across boundaries. Further the boundaries of the DUTs can be efficiently managed by keeping the boundaries at input or output of arbiters as it not only needs less number of BFMs but also follows same protocols. These boundary guidelines also provides opportunity to effectively manage number of BFMs required at simulation and thus can also help emulation to reduce number of board required for the emulation environment.

Since BFM or verification components used across DUTs are same, follows the system command and memory component, test can be not only synchronized with real time firmware but also scaled across the DUTs and further resulting into quicker bringing up of testcases for specific features at concise DUTs.

As the protocols and boundaries of DUTs are similar, creating scoreboard and reusability increases which not only helps in integration but also maintaining the scoreboard checks.

Integration of such shared components and design can configurable with above suggested guidelines and simplified efficient approach.

3. Benefits and Future Directions

Suggested implementations offers efficient way of building

API, testbench components, test development and scoreboard.

Efficiency:

API, sequence items and data structure are automatically created through script based on the table 1 mentioned above. This provides a head start in developing verification components and fewer engineering efforts. As boundaries of the DUTs are decided based on the guidelines provides the testbench can be configurable, provides flexibility, easy to integrate design and verification components and debug. Additionally, configurability and reusability of test sequences based off of system level APIs provides scalability and efficiently manage the engineering effort and shorten the verification cycle.

Accuracy:

Centralized testplan provides unique opportunity to bridge gap in testplan between SOC, subsystem and IP. Due to automated approach in creating test constraints and API structures, code will compile successfully in first attempt and thus implementing the recompiling cycle due to human errors and increases accuracy of the code. Due to reusability of the verification components, system level testing can match the firmware real time scenarios and hence pre silicon verification can match the real time scenarios in the product and find early bugs.

Scalability:

Test bench components, system level APIs and test bench configurability provide opportunity to scale the size of the DUT and test sequence to stress overall architecture. As system level APIs can be reused across the DUTs and also scalable across different projects [8].

Future Enhancements:

As complexity of the architecture increases, there is always chance of improving the DUT which can be further improved using automation to create coverage and converge on the functional and code coverage. Further the existing test sequences can be fed back with coverage hit in order to self - create sequences using AI and machine learning. Debugging failures in complex architecture remains the critical aspect of verification execution cycle, simplified approach and developing tools to either automate or aid the debugging process will allow debug quicker and reduce the verification cycle.

4. Conclusion

Complexity of architecture of graphics, CPU and AI accelerators are exponentially increasing and verification of cycle is also proportionally expanding which will delay the timelines for product to be launched in market. It is paramount important to reduce the verification cycles and engineering heads required for each project and can be achieved through proposed frameworks. Automated framework proposed in this paper reduces effort for developing API, test constraint development and provides guidelines on deciding the boundaries of the DUT. System level testing framework and verification component reusability adds to the efficiency and accuracy of testplan executions.

References

[1] Youngsok Kim; Jae - Eon Jo; Hanhwi Jang; Minsoo Rhu; Hanjun Kim; Jangwoo Kim “GPUpd: A Fast and Scalable Multi - GPU Architecture Using Cooperative Projection and Distribution”

[2] Intel corporation “Introduction to the Xe - HPG Architecture White Paper”

[3] Yingpan Wu; Lixin Yu; Wei Zhuang; Jianyong Wang “A Coverage - Driven Constraint Random - Based Functional Verification Method of Pipeline Unit”

[4] Amr Hany; Ahmed Ismail; Ahmed Kamal; Mohamed Badran “Approach for a unified functional verification flow”

[5] Segev, E.; Goldshlager, S.; Miller, H.; Shua, O.; Sher, O.; Greenberg, S.; , "Evaluating and comparing simulation verification vs. formal verification approach on block level design, " Electronics, Circuits and Systems, 2004. ICECS 2004. Proceedings of the 2004 11th IEEE International Conference on, 2004

[6] Anantharaj Thalaimalai Vanaraj; Marshal Raj; Lakshminarayanan Gopalakrishnan

[7] “Functional Verification closure using Optimal Test scenarios for Digital designs”, 2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)

[8] S. Yang, R. Wille, D. Groβe and R. Drechsler, "Minimal Stimuli Generation in Simulation - Based Verification", *2013 Euromicro Conference on Digital System Design*, pp.439 - 444, 2013.

[9] Wei Ni; Jichun Zhang “Research of reusability based on UVM verification”, 2015 IEEE 11th International Conference on ASIC (ASICON)

[10] He Zhang; Chunyu Wu; Wenjing Zhang; Jiwei Wang “Design on SOC module - level functional verification platform”, 2011 Second International Conference on Mechanic Automation and Control Engineering