5-2019

Verification of SHA-256 and MD5 Hash Functions Using UVM

Dinesh Anand Bashkaran
dab8730@rit.edu

VERIFICATION OF SHA-256 AND MD5 HASH FUNCTIONS USING UVM

by
Dinesh Anand Bashkaran

GRADUATE PAPER

Submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE
in Electrical Engineering

Approved by:

____

Mr. Mark A. Indovina, Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

____

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK
MAY 2019

To my family, friends and Professor Mark A. Indovina for all of their endless love, support, and encouragement throughout my career at Rochester Institute of Technology

Abstract

Data integrity assurance and data origin authentication are important security aspects in commerce, financial transfer, banking, software, email, data storage, etc. Cryptographic hash functions specified by the National Institute of Standards and Technology (NIST) provides secure algorithms for data security. Hash functions are designed to digest the data and produce a hash message; a hash is a one-way function which is highly secured and difficult to invert. In this paper, two such hash algorithms are verified using the Universal Verification Methodology (UVM). UVM is IEEE 1800 standard developed to assist in the verification of digital designs; it reduces the hurdle in verifying complex and sophisticated designs.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This paper is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Dinesh Anand Bashkaran

May 2019

Acknowledgements

I would like to thank my advisor, professor, and mentor, Mark A. Indovina, for all of his guidance and feedback throughout the entirety of this project. He is the reason for my love of digital hardware design and drove me to pursue it as a career path. He has been a tremendous help and a true friend during my graduate career at RIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The motivation behind the development of SystemVerilog was to develop a unified language to fulfill the needs of user to design and verify. It was initially promoted by Accellera system initiative, after several revisions it became an IEEE standard of Hardware Description and Verification Language (HDVL). SystemVerilog supports object-oriented programming (OOP) concepts to provide diverse needs in verification and modeling. Despite providing enormous features, SystemVerilog failed to adopt widespread practice. Numerous cultural and practical challenges questioned the adoption of SystemVerilog, which paved way for new libraries, tool kits and methodology guidance. This led to development of the Universal Verification Methodology (UVM), a library which is comprised of SystemVerilog and OOP concepts. UVM is a class library used to create test environment and is a powerful package including support for transaction-level modeling (TLM), phasing, re-usability etc. UVM provides infrastructure which is reliable, robust and improves the quality of a testbench. In this project, such an environment is used to verify the algorithm of hash functions. Hash functions maps any arbitrary data to fixed size data, the values returned by hash functions are called hashes or digests. They are an important and ubiquitous cryptography building block. The size of hash value depends on the algorithm used to produce it. This project uses MD5 and SHA-256 algorithm to build hash functions, where MD5 produces a 128 bit hash (32 characters) and SHA-256 produces a 256 bit hash (64 characters).

1.1 Research Goals

To create an environment that can verify the hash functions MD5 and SHA-256. The following tasks are considered for success of project.

1. Understand the algorithm of hash functions MD5 and SHA-256.
2. Build a testing environment in UVM and develop a connection between the test components and RTL core of hash functions.
3. Analyze different test plans to verify the functionality of IP.
4. Analyze possible assertion driven methods.
5. Analyze the functionality and timing relation of RTL core in comparison with a software model.
6. Collect the functionality and coverage test results.
7. Perform gate level simulation to verify the correctness on technology dependent netlist.
8. Achieve enough functional coverage to eliminate corner case bugs.

Chapter 2

Bibliographical Research

In any design, majority of the time is spent in verifying the design. Verification could be a hurdle in product release and success in market but with the help of modern language and tools, the time spent can be drastically reduced. The motivation behind verification is to check the functionality of design. There are many techniques that can be used to build a testing environment. Without standardized techniques, verification could increase the cost of using intellectual property (IP), electronic design automation (EDA) tools and increases the time spent in verification. To address this issue, the verification language SystemVerilog was introduced by IEEE Standard 1800. SystemVerilog provides powerful, reusable, flexible environment for verification which counters the cost, complexity and time [3]. SystemVerilog integrates Verilog Hardware Description Language (HDL) and OOP concepts and provides features like coverage, assertions, SystemVerilog DPI (Direct Programming Interface), and constrained random stimulus.

SystemVerilog lacked certain features like macros to automate the creation of utility methods, metaprogramming, code patching which emerged the evolution of the UVM library. The myth, “UVM is making amends for inadequacies of SystemVerilog”; was proved fallacious by [4]. The paper discusses the certain features of UVM can be performed by SystemVerilog in its own right. However, SystemVerilog lacks in macros which is still unanswered. UVM outplays any other verification technique and its application programming interface (API) defines a base class library (BPI) which is used to develop scalable and modular components for functional verification. In fact, UVM has its own drawbacks which will be resolved in next revision[5]. [6] is a book used as reference throughout the project. The author describes the concepts of UVM from scratch and provides various examples where the concepts are implemented. Overall this text provides adequate information about UVM along with a few case studies which is enough to start building your own testbench.

Code coverage is one of the important features in a verification methodology. However, the accuracy of code coverage is often disrupted by excitation of functional blocks. A coverage method considering conditional checks and observations can significantly increase the accuracy of code coverage [7]. The paper also addresses the issue with narrowed technique which can be implemented in this project. Functional coverage is another aspect that extensively gives the percentage of logic functions used. Functional coverage plays a vital when it comes to implementation of complicated SOC with numerous IP in field-programmable gate array (FPGA). One such implementation of Functional coverage in SystemC based on aspect orientated programming is discussed in [8]. To verify the results of DUT, a reference model is required which is usually developed in SystemC. SystemC is unified platform based on C++ developed by Open SystemC Initiative. It is composed of hardware class library and simulation kernel. It can work in parallel in RTL and produce the results to compare in scoreboard section. [9] discusses the usage of SystemC using case study, in fact the implementation can be simply reproduced in this project. [10] is a good reference to understand the re-usability of testbench. The DUT is I2C and AMBA AXI lite, both IP's are verified using the same framework in SystemVerilog. This paper provides ideas to build a single flexible framework that can be used irrespective of DUT with minor changes. This approach would increase the verification process of MD5 and SHA-256.

[1] is a very good reference to understand the algorithms used in hash functions. It is one of the oldest papers published which discuss all the properties, features, benefits and applications of hash functions. This reference is extensively used to understand the hash functions. Implementation of any algorithm is not easy even though the algorithm is understood to its extreme. Paper [11] presents a good approach to implement hash functions in hardware. Stages of hash functions are discussed from which the hash function can be constructed in any language. Mathematical equations are used to describe the stages of hash functions. Overall, the functions performed are OR, AND and XOR but, how it is performed is decided by the hash function in the respective stage. [12] presents the hardware level implementation of SHA-256. The algorithm is implemented in HDL, tested and verified using FPGA. The paper also discusses the benchmark and optimization solutions for the RTL. These results can be considered as benchmark in developing hash functions at RTL level.

Hash functions are low collision resistant, if the two identical hash messages are generated from two distinct input. $H(A) = H(B)$, hash message of A and B are same but A != B, the original message A and B are distinct. Hash function is collision free only if the generated hash message is not identical for two distinct data. Reliability of hash functions against the attacks is totally dependent on collision property of hash function. Paper [13] presents a unique approach to increase the reliability of hash functions. Hash function can be made collision free using this approach which deals with altering the mathematical equation in stages of operations. The paper also provides a benchmark result, the chances of collision is $1/22$. These references stands out to be extremely helpful for designers to design hash functions in highly secured manner. Though this project focuses mainly on verifying the hash functions using UVM, research was made to understand the hash functions completely.

“How safe are the hash functions”, is unanswered so far in this context. [14] discusses the vulnerability index of hash functions and provides an approach to increase the vulnerability of hash functions against chain attacks. Any given hash function is constructed using several nodes which together forms cycles. If a hash function is truly random then the length of hash cycles is $2^n/2$. Depending on the size of n, the vulnerability of hash function increases. Generally, for the value of N greater than 128 hash functions are considered safe from chain attacks. However, there is no evident that commonly used hash function does have that property. To attack a hash function, one must find a node and repeatedly hash from it until a cycle is found. With the equation that decides the vulnerability index of Hash function, couple of methods are proposed to increase the vulnerability of hash functions. The proposed algorithm is used on MD5 and SHA hash functions, it dramatically increased the vulnerability. When hash functions are constructed using such algorithms, there are pretty much safe against chain attacks.

Hash functions can be used on any arbitrary string to produce a fixed hash message. The size of fixed message depends on the algorithm which are usually 128-512 bits. Certain application requires variable hash message which becomes uncertain with fixed algorithm and fixed hash message size. [15], provides an approach for MD5 hash function to produce variable hash message without compromising the stages of algorithm. This provides high integrity algorithm with user defined hash message. The only difference in proposed algorithm is the compression block after padding stage where the hash message is compressed to ‘n’ bits. This enables MD5 to take up to 512 bits and convert into user defined hash message making it flexible for low end application. There are several hash functions but all of them do the same job. Depending on the application and the security level choosing hash function can be arduous. [16] presents a solution to this by discussing the various hash functions and suitable application for them. Database indexing, symbol tables, network processing algorithms are few real time applications discussed with dynamic, cryptographic, robust and string hashing techniques. In certain applications where the memory allocation for hash functions can be inadequate, a chain hashing technique comes into play where the hash functions are linked using external memory.

[17] presents partial work of this project, verification of SHA-256 algorithm using UVM. Paper presents the analysis of the SHA-256 algorithm, and a UVM platform is used to validate SHA-256 as IP. In fact the same procedure can be reproduced to test the SHA-256 and MD5. This reference is used over the implementation of project. Note that although a few components of testing environment might change with respect to DUT, the framework remains the same. Saying that, SHA 256 or MD5 does not make a major difference in the UVM framework. [18] is a good reference to understand the re-usability of UVM testing framework. The paper presents the concepts of re-usability using few case studies.[17] can be reproduced to construct a UVM framework for SHA-256 followed by [18] to reuse the framework for MD5 algorithm. These two are the most relevant and significant references used over the implementation of this project.[19] presents a method to implement automated verification. UVM is a complete framework that includes coverage metrics, self-checking testbenches and automatic test generation. Even though the DUT here is not complex but rather simple IP's, complete utilization of UVM is another goal.

While reliability and data integrity is one key feature of hash function, the performance is yet unanswered. Especially when used in high end application where low latency is expected, the algorithm is expected to process rapidly. In[20, 21] both references aim at same goal using different approaches. The goal is to optimize the design by adjusting the longest delay path in netlist to provide high throughput and low latency. To overcome this hurdle, unrolling is a technique which focuses on multiple rounds of the core compression function in combinational logic, which helps in reducing the number of clock cycles required to compute the hash. The paper also discusses about pipelining the architecture for better throughput and low latency.

[22] provides a unique approach to optimize MD5 using data dependency, data forwarding and pipelining. The pipeline is 4 stages and the architecture of MD5 is altered to minimize the effect of data dependency. The architecture is tested on a Xilinx Virtex FPGA, benchmark results 1040 M bps as throughput. This architecture seems to be the best design up to date.

[23] talks about the advantage of implementing MD5 on hardware rather than software. Hardware implementations can be more immune to attacks and cannot be altered like software. A Xilinx Virtex 5 FPGA is used to implement the MD5 hardware. The implementation was performed on two ways, iterative looping and full loop unrolling. The overall throughput is 165 M bps. One such common way to increase the efficiency is to pipeline the stages of operation [2]. The paper discusses the pipelined architecture of MD5 through different stages. The pipelined architecture is implemented in FPGA, with a 64 stage pipelined MD5 architecture the throughput is 725 M bps, which is 6 times more than non-pipelined version. [24–26] are online reference for UVM which is extensively used to understand and implement Test Bench.[27, 28] provides the source code for SHA-256 in Verilog and C. [29, 30] provides the source code for MD5 in Verilog and C.

Chapter 3

Overview of Hash Functions

This chapter discusses the basics concepts, construction and applications of hash functions.

3.1 Hash Functions

A hash function takes arbitrary data to produce fixed size hash message, the size of hash message depends on hash algorithm. $h = H(M)$, M is input, and h is hash message generated by H algorithm. The reliability of the hash function depends on the algorithm and length of the message generated. Merkle-Damgard model is commonly used hash algorithm in hash functions. In general hash functions should have following properties,

3.1.1 Preimage resistance

If hash message h is given, it should be highly impossible to find message M such that $h = H(M)$. Hash functions are one-way property, hash h can be computed from message M but not other way around.

3.1.2 Second preimage resistance

Given message M1, it is highly impossible to find another message M2 such that M1 and M2 generate same hash h. 3.1

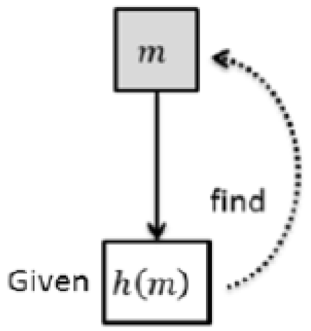

Figure 2 Preimage resistance

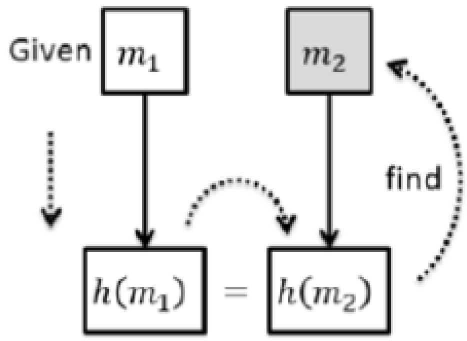

Figure 3 Second-preimage resistance

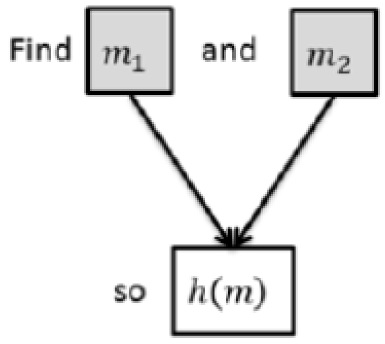

Figure 3.1: Properties of Hash Function[1]

3.1.3 Collision resistance

It should be highly impossible to find two messages S1 and S2 such that $S1 =! S2$, but $h(S1) = h(S2)$.

3.2 Construction of Hash Functions

Hash functions can be structured in several ways but the Merkle-Damgard model stands out to be popular and reliable. It has been practiced successfully in hash functions like MD5 and SHA-2. In this model, the message is padded and divided into uniform length of blocks. The compression function F processes the blocks in sequential order which generates intermediate hash. The final compression function outputs the hash message. The size of the hash message depends on the user implementation. If the compression function F is collision resistance, so is the generated hash message. SHA-2 and MD5 follow Merkle-Damgard model. They use operations like OR, XOR and AND for the compression functions. Collision-pairs was founded that makes these algorithms vulnerable to collision attacks. Although there was no evidence to support the successful attack against this algorithm. The results from, National Institute of Standards and Technology (NIST) shows SHA-3 seems to be more secured than other versions of SHA-2 algorithm.

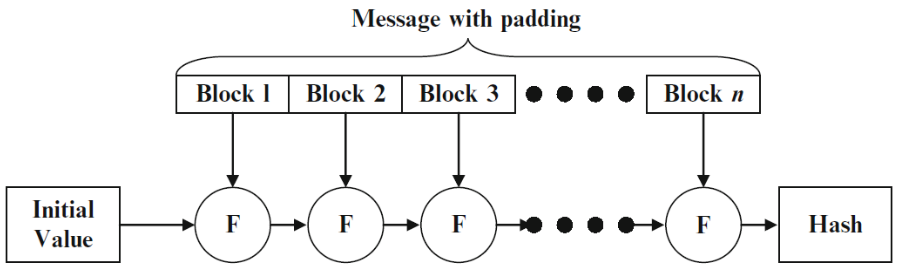

Figure 3.2: Merkle-Damgard model [1]

3.2.1 MD5

MD5 is an implementation of Merkle-Damgard. MD5 has 512 bits from M0-M15, sixteen 32-bit word. Internally, the hash state has state variables of four 32-bit word (A1, B1, C1, D1). In the stages of computation, the state variables are changed to new 32-bit value by the compression function F. MD5 has 64 stages of steps grouped in terms of 16 stages. First 16 stages are called Round one, second as Round two and so on. Every Round has unique compression function. The final hash message is the concatenation of state variables A1, B1, C1, D1 in last block of last Round.

3.2.2 SHA-256

SHA-2 can operate on four different modes, SHA-256, SHA-384, SHA-224,and SHA-512. This project uses SHA-256 and discussion is only based on SHA-256. SHA-256 also uses the Merkle-Damgard model and works similar to MD5. The message block size is 512 Bits like MD5, but the state variables are doubled to include 8 state variables. The operations performed on each stage are AND, OR, XOR.

3.3 APPLICATIONS OF HASH FUNCTIONS

3.3.1 Data Integrity

After receiving data, the hash message can be generated and compared with another hash message, which was generated from original data. Here, the original message should be sent through a secure channel. If both the hash message are equal, then data was not modified during the transmission. This is defined by the second preimage property of hash function.

3.3.2 Secured Digital Signature

Public-key algorithm combined with Hash functions evolves many applications, one such is Digital Signature. Digital Signature can be generated by combination of Hash message and Encryption from private-key. The Generated Text can be used as signature. The Signature can be verified by decrypting the signature using public-key and comparing it with the hash message.

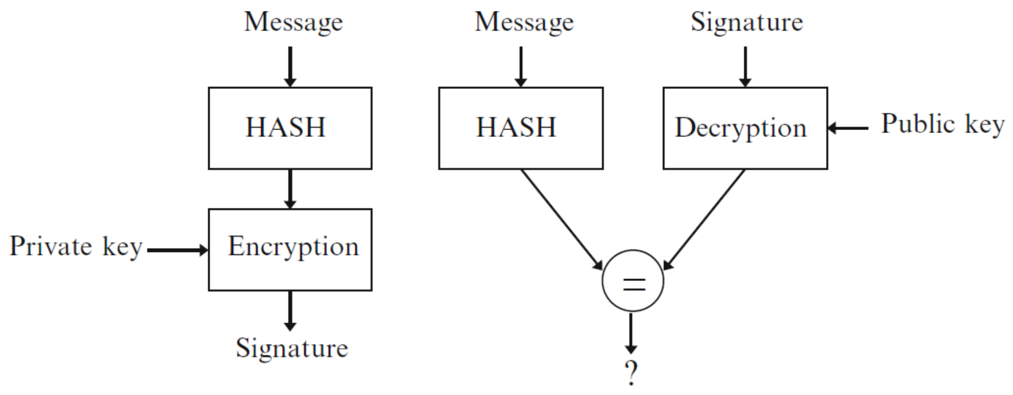

Figure 3.3: Digital Signature [1]

3.3.3 Authentication

The password of the user is hashed and sent to server, where it is compared by the server. Every time the password is hashed, it is concatenated with random values generated by the server. This allows high data-integrity.

Chapter 4

Overview of UVM

This section dives deep into UVM, and describes about different features, phases, transaction-level modeling (TLM), semantics, test bench components etc.

4.1 Evolution of UVM

The initial version of UVM was released on Feb 28, 2011, after which UVM became a stand-alone verification methodology. Since then UVM has undergone several changes to fix bugs and add features. A few important features of UVM 1.0 were:

1. End-of-test objection mechanism to enhance the ease of clean up at end of simulation.

2. Call-back mechanism that would alter the existing behavior.

3. Report catcher to enhance report handling.

4. Heartbeat Mechanism to monitor the life of UVM components.

5. Register Level Modeling to enhance the verification of memory unit.

6. TLM interface FIFO, resource database.

Around June 2014, UVM 2.0 was released. Further changes, bugs, features were improved; a few important features are:

1. uvm_sequence_base::starting_phase has been replaced by set_starting_phase and get_starting_phase. This change prevents modification of phase during run time.

2. uvm_sequence_base::set_automatic_phase_objection were added which avoids the necessity of calling raise_objection and drop_objection explicitly.

3. New methods like stop_request, stop_global_request, set_global_timeout, uvm_test_done, do_kill_all etc.

Today, SystemVerilog and UVM is the preferred environment to verify from IP, Block, SOC, chip level. Over years of practice, engineers have adapted themselves to the UVM environment. Even though UVM is sophisticated, it is highly flexible to provide the environment for complex verification projects. Even a few Verification IPs (VIPs) were introduced which became handy in creating UVM testbenchs.

4.2 UVM Scheduling Semantics

A single time slot (One clock period) is divided into multiple regions where several different events are scheduled. UVM time scheduling has been improved from Verilog scheduling semantics. This event scheduling provides clear and predictable interaction with DUT. Verification Engineer should be aware of event scheduling to understand the core of UVM. Following are the different region in event scheduling,

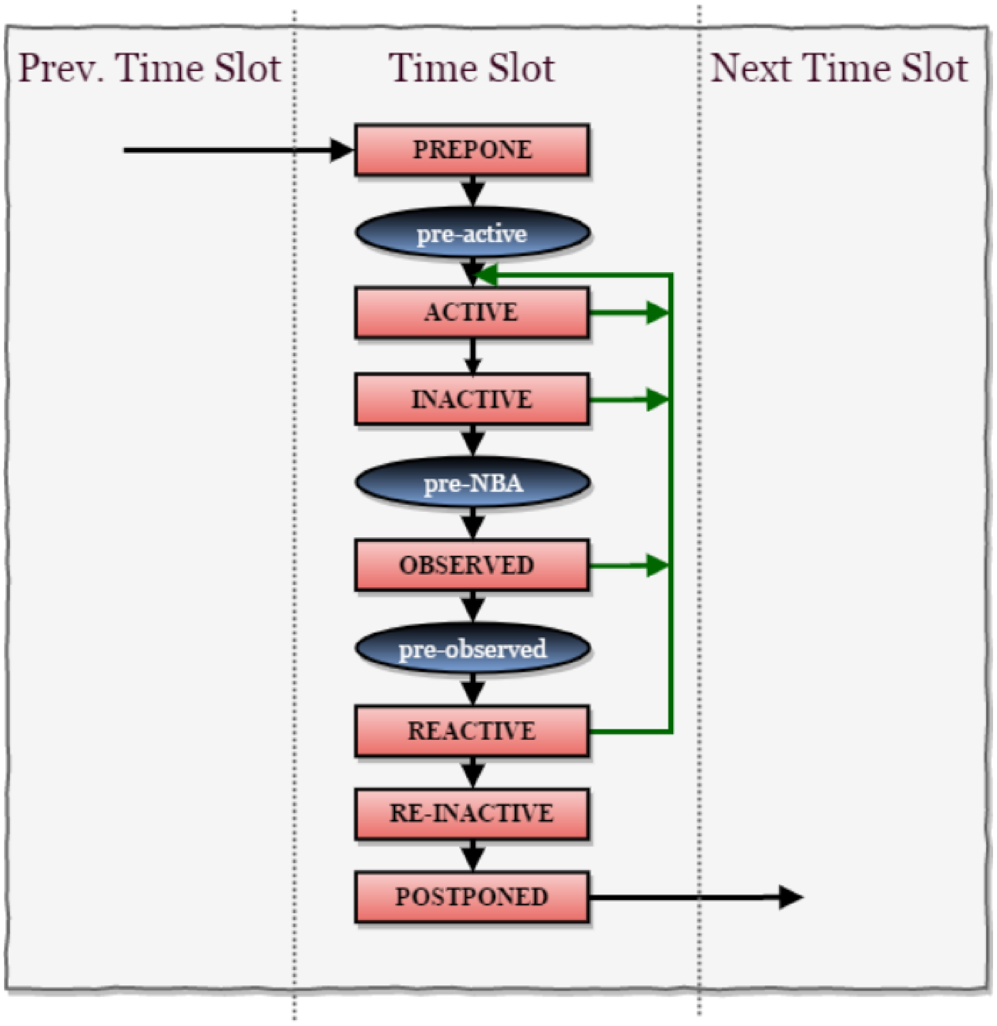

Figure 4.1: UVM Scheduling [2]

4.2.1 Preponed Region

The variables used in concurrent assertions are sampled in this region. The preponed region is evaluated only once, and it happens right after advancing simulation time.

4.2.2 Pre-Active Region

Pre-Active region is for the PLI (Programming Language Interface) callback mechanism. It allows user code to read and write values just before the active regions are evaluated.

4.2.3 Active Region

All the event that needs to be evaluated in corresponding clock period are placed in this region. The order of execution can be processed in any order.

1: Execute blocking assignments, continuous assignments.

2: Execute the right-hand side (RHS) of non-blocking assignments and schedule updates in the Non-blocking Assignment Events Region (NBA).

3: Execute $finish and $display commands.

4.2.4 Inactive Region

In this region all the events that needs to be evaluated after active events are placed. #0 blocking assignments are placed in this region.

4.2.5 Non-Blocking Assignment Region

The RHS of non-blocking assignments which were evaluated in active region, is updated to left-hand side (LHS) in this region. Since all the non-blocking assignments needs to be independent of each other, this region solves the issue.

4.2.6 Observed Region

The function of this region is to evaluate the concurrent assertions. The values required for concurrent assertions were sampled in preponed region. The reason for this region is, assertion should be evaluated once in every clock period. Depending on the result of assertion whether pass/fail, the corresponding event is scheduled in reactive region.

4.2.7 Reactive Region

This region is allocated for events in program block. The functionality if this region is to execute program block in any order. Since, this region is placed after active and non-active region this can be used to avoid race condition. Another way of avoiding race condition is using clocking block which would induce skew.

1. Execute program blocking assignments.
2. Execute pass/fail condition that were assigned in concurrent statements.
3. Execute program block continuous statements.
4. Update the LHS of non-blocking assignments in program block.

4.2.8 Postponed Region

$strobe and $monitor commands are executed in this region. The final updated value is reflected for the corresponding time slot. This region is also used to collect the functional coverage.

4.3 Phases of UVM

UVM phases were made to synchronize the behavior of testbench. This enables organized flow from start of testbench until the end of simulation. UVM phases are mainly divided in three categories build, run and clean up phases.

4.3.1 Build Phase

The build phase is responsible for two different functionalities. Building the phases and connecting the components. Components are constructed and connected at the start of simulation; this happens in zero simulation time. UVM builds components that are registered in the factory and connect them depending on how user defines the connection in connect phase. Once the components are built and connection, the elaboration is done.

4.3.2 Run Phase

Run phase is were simulation starts and ends. Entire simulation happens in the run phases. Run phase is time consuming and are usually defined in task. Run phase consists of resetting the DUT, setting up the DUT, generation of randomized value, driving the value, monitoring the value and comparing the results. Run phases is only terminated when the sequence is finished. Run phase is further divided as,

4.3.2.1 Reset

This phase is reserved for reset behavior. This would generate reset and put the interface into default state or send reset signals to DUT. The main motivation of having this phase ahead is to bring the DUT into initial condition.

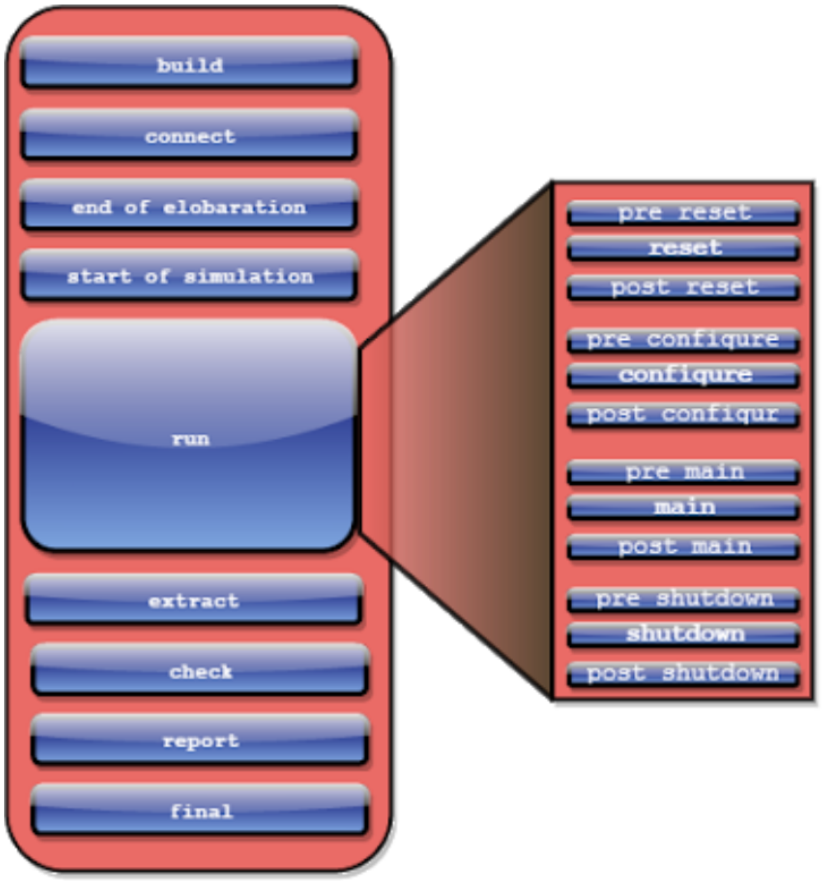

Figure 4.2: UVM_Phases [2]

4.3.2.2 Configure

This phase is used to configure the DUT. The configuration is done to make sure DUT is ready to accept any transactions which is about to be sent in near future.

4.3.2.3 Main

Main phase is where the stimulus is generated and sent to DUT. User can have N number of test cases or test sequence. Every test case is generating a random variable which is defined using “rand” or “randc” system function. Main Phases is responsible for generating the stimulus using random generator.

4.3.2.4 Shutdown

Shutdown is responsible to assure the stimulus generated in the main phases have propagated through the DUT. It also makes sure any resultant data have been drained away.

4.3.3 Clean-up Phase

This phase is initialized once the simulation is done. This phase is responsible to collect the data generated during the simulation run phase. It extracts the information such as reports, errors etc.

4.4 UVM factory

UVM factory is a repository where the components of test bench are manufactured and stored. Only one instance of factory is present in a simulation. All the testbench components needs to be registered and created in the factory. There are two types namely components and objects which can be registered and created in factory.

4.4.1 UVM_Objects

UVM objects are instance that is dynamic in nature and changes during the run time. uvm_seq_item and uvm_sequence are the dynamic objects in testbench environment. User can define and create objects for multiple sequence and transactions. They can be invoked depending on the user needs. UVM objects are registered using uvm_object_utilis method and then object is created for every single instance invoked.

4.4.2 UVM_component

UVM components are instance that is static in nature and does not change during the run time. UVM components are static and built during the build phase of UVM. These components can be considered as physical structure or building block of UVM environment. Driver, Sequencer, monitor and scoreboard are examples of components. UVM components are registered using uvm_component_utils method. Object are created every time when these components are invoked.

4.4.3 Type Override

The component are created using type_id::create() method instead of new(). Both the methods lets you create a object but new() does not let type override. When a object is created using new() methods, the type of the object can not be changed. This means, once an object is created and memory is allocated the behavior of the object cannot be altered. This is replaced using type_id::create() which lets user to override the behavior of the object type. Again, this is one enhancement that lets user to reuse the objects by changing its type. There are few UVM override methods that enables user to replace the object instead of type. UVM gets highly flexible here and provides multiple options to user.

Chapter 5

Custom UVM Backend Design

This chapter discusses the structure and design of the custom target-specific UVM framework. UVM is reusable, factor-based environment which can enhance the productivity of verification. UVM is generally used to verify at different level of abstraction from IP, block, chip level. Verification is usually bottom-top approach which provides level of confidence for verification engineers. Section 5.1 discusses the UVM Design for IP verification and Section 5.2 discusses the UVM Design for Block Level Verification.

5.1 UVM IP-Level Design

UVM is sophisticated, but on the other hand it is flexible for user requirements. This overall section describes how UVM environment could be developed for IP-Level Verification. IP level is the low level of abstraction in verification process. SHA-256 and MD5 are the two different IP used as DUT to verify its functionality. Next two sub-session describes the environment components of the UVM framework and the process flow between DUT/UVM/C model. For IP level Verification, Generic UVM framework was used. This consists of Multi-

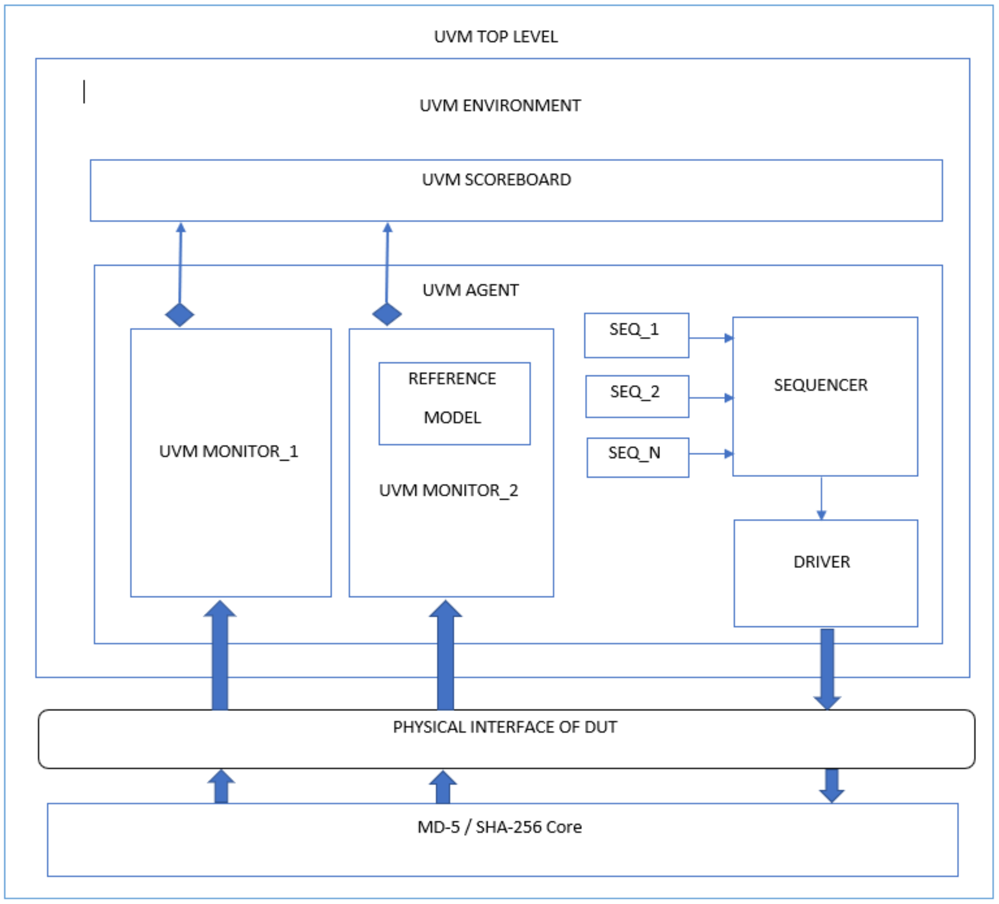

Figure 5.1: IP-Level Design

ple test sequence, sequencer, driver, couple of monitors, Scoreboard. The UVM continuously interacts with the DUT and Reference model during "Run Phase" and compares the results of DUT with reference model and updates the coverage. Phase objection were used to start the sequencer and end the UVM in appropriate way.

5.1.1 UVM Environment for IP-Level

1. **Sequence**: UVM_Sequence is the base class, which can be extended to define streams of test sequences for DUT. Here, 5 different test sequences are defined, however they

only have 1 sequencer to run. In other words, only one test sequence can be active at any point of simulation time. In order to run parallel test cases, DUT should be configured in pipelined fashion. "set_arbitration(UVM_SEQ_ARB_USER)" was used to arbitrate between the sequences. As mentioned, one sequence is selected among five sequences. The sequence is selected based on the priority given as argument. Once, the high priority sequence is completed, the next sequence is selected. In order to achieve this, Multiple sequences are defined extending UVM_sequence class. Top level virtual sequence is defined in which actual multiple sequences are instantiated and object are created respectively. This virtual sequence is connected to the sequencer.

2. 2**Sequencer**: As mentioned, one sequencer is defined. Sequencer is intermediate connection between the sequence and Driver. The connection between sequencer and driver is static, which is connected using uvm_seq_port. The connection between sequencer and sequence is dynamic and changes during run time depending on the logic inside the virtual sequence. In this case, Priority is used to switch between the multiple sequences in the virtual sequence. At any point of simulation time one sequence is connected to the driver through the sequencer.

3. **Driver**: Driver drives the DUT using the random data generated in the sequence. In other words, driver is heart of UVM environment which is responsible to extract the random data from sequence and drive the DUT. Every DUT has its own method to drive the Inputs and outputs. In MD5, data is sent, and output is collected with some reference signals. In SHA-256, the data should be placed in some loop fashion and output is collected with some reference signals. The communication between the Driver and the sequence happens through UVM_seq_port. There are few methods like get_next_item, which gets the next random value from sequence and item_done, which implies trans-

action is completed. There are few non-blocking methods like get(), put() which can be handy when more than one random value is required for 1 variable. The communication between the driver and the DUT happens through interface. Driver gets the access to physical interface using the virtual interface which can be passed through the uvm_config_db method. This is one of the excellent features added in UVM, which, unlike SystemVerilog, eliminates passing virtual interface through all hierarchical level until driver is reached.

4. **Monitor**: One monitor simply monitors the response from DUT. It collects the output from DUT for every valid transaction. This is where UVM becomes transaction level modeling ignoring low level abstraction noises from DUT. The collected value is sent to Scoreboard using Analysis_port, which is discussed in scoreboard section. Another monitor has reference model embedded in it. This reference model is simply C model which is used to generate expected values. This monitor monitors the input to DUT for every valid transaction and sends the same input to the reference model embedded inside it. The output from reference model is collected and sent to scoreboard for comparison.

5. **Agent**: Agent provides high level abstraction of Sequence, Sequencer, Driver and x2 Monitor. Agent provides the environment to wrap these components. The components are built and connected here. Agent holds analysis port which is connected to the analysis port of Monitor. The other end of agent's analysis port is connected to the scoreboard.

6. **Scoreboard**: Scoreboard is the component where results are compared and displayed. Monitor sends the value to scoreboard using analysis port through agent. Scoreboard gets these values for DUT and Reference model, compares them and updates the test results. In other words, scoreboard is simply a subscriber to the monitor. Inside scoreboard, there are 2 FIFOs that collects the value from DUT and reference model in each.

This eliminates the synchronization issues between the two results.

7. **Environment**: The environment is the highest-level abstraction of UVM components which wraps the Scoreboard and Agent. Components are built and connected in the environment. To access any component in UVM framework, it needs to be accessed through the environment.

5.1.2 Design Flow

It is obvious flow starts with randomization and ends at comparing the output with expected result. UVM works in top-bottom methodology, sequence is started on sequencer at UVM_test. There is multiple sequence, but they all have only sequencer on which they can run. Every sequence shares single sequencer and only one can run at a point of time. To control this, virtual sequence is used which decides what sequence will run when. Virtual sequence is configured using priority, each sequence is provided with some level of priority. Once, this is called sequence starts sending randomized value to driver. Driver and sequence are connected using peer to peer type connection called seq_item_port. They both work on system call protocols get_next_item, item_done, get, put. They are blocking methods and waits for transaction to be completed. Driver drives the DUT with the random value generated from sequence.

Monitor monitors the ports of DUT and captures every valid transaction. This eliminates unwanted low-level abstraction noises. One monitor is provided with access to reference model to generate expected result. This is done using DPI library which enables transaction between SystemVerilog and C language. Both the monitors send the DUT output and reference output to the analysis port. Scoreboard is connected to other end of the analysis port which samples these values. The overall flow is synchronized using system methods raise_objection and drop_objection. Once all the raised objections are dropped the simulation ends.

Assertions are defined in interface where direct access to input/output is available. Assertions run in parallel during Run phase during simulation time. This provides a level of confidence to verify the functionality of DUT. Cover group consists cover points (input and output) which monitors the toggling of bits. Cover group also provides code coverage which implies how much of code is being executed.

5.2 UVM Block Level Design

Sha-256 and MD5 were wrapped together as single block. They are instantiated and wired through a top-level unit. UVM framework was developed to test this block which consists of two independent IP's. Testbench was modified to drive two different IP's. One of the interesting things was these two IP's were driven, sampled, verified in parallel independent of each other. Systemverilog Fork comes handy to perform parallel task.

5.2.1 UVM Environment for Block-Level:

1. **Sequence**: UVM_Sequence is the base class, which can be extended to define streams of test sequences for DUT. Here, 5 different test sequences are defined, individually for SHA-256 and MD5. Each DUT has its own sequencer on which one sequence would run at a time. "set_arbitration(UVM_SEQ_ARB_USER)" was used to arbitrate between the sequences. one sequence is selected among five sequences. The sequence is selected based on the priority given as argument. Once, the high priority sequence is completed, the next sequence is selected. In order to achieve this, Multiple sequences are defined extending UVM_sequence class. Top level virtual sequence is defined in which actual multiple sequences are instantiated and object are created respectively. This virtual sequence is connected to the sequencer.

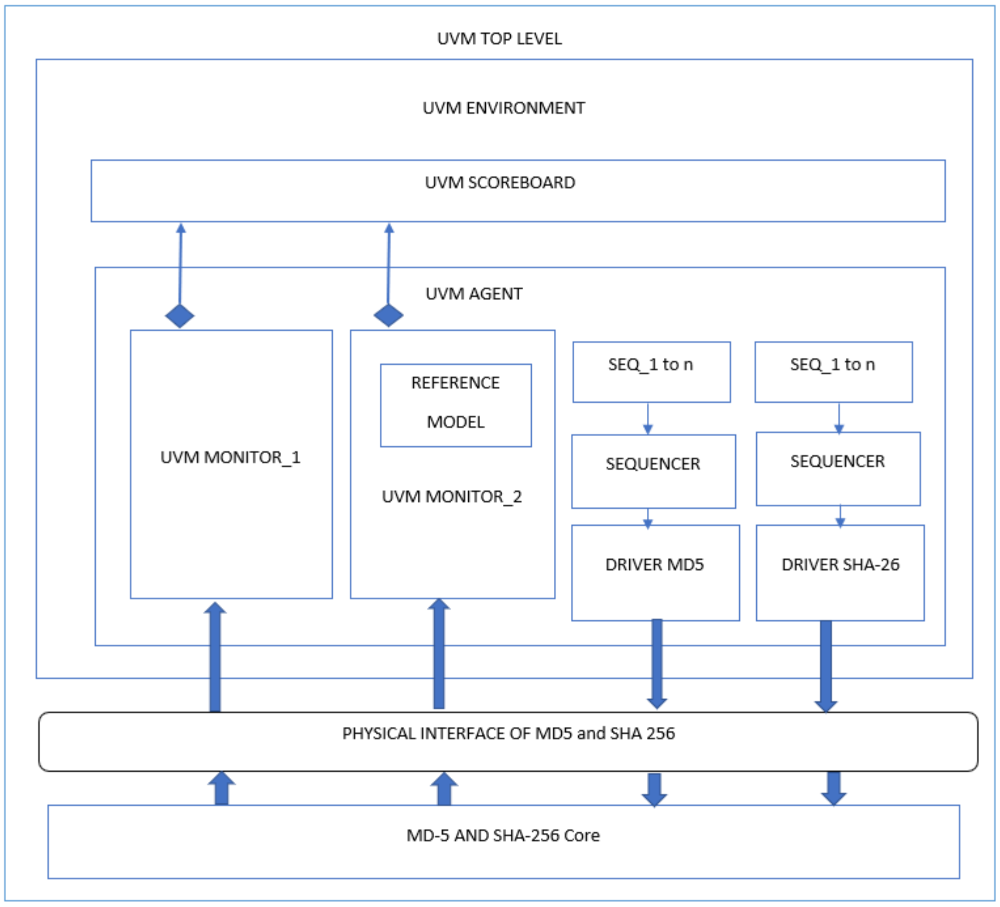

Figure 5.2: Block-Level Design

2. **Sequencer**: As mentioned, two sequencers were defined. Sequencer is intermediate connection between the sequence and Driver. The connection between sequencer and driver is static, which is connected using uvm_seq_port. The connection between sequencer and sequence is dynamic and changes during run time depending on the logic inside the virtual sequence. In this case, Priority is used to switch between the multiple sequences in the virtual sequence. At any point of simulation time one sequence is connected to the driver through the sequencer.

3. **Driver**: Two Drivers were used to drive two DUT in parallel. Driver drives the DUT using the random data generated in the sequence. In other words, driver is heart of UVM environment which is responsible to extract the random data from sequence and drive the DUT. In MD5, data is sent, and output is collected with some reference signals. In SHA-256, the data should be placed in some loop fashion and output is collected with some reference signals. The communication between the Driver and the sequence happens through UVM_seq_port. There are few methods like get_next_item, which gets the next random value from sequence and item_done, which implies transaction is completed. There are few non-blocking methods like get(), put() which can be handy when more than one random value is required for 1 variable. The communication between the driver and the DUT happens through interface. Driver gets the access to physical interface using the virtual interface which can be passed through UVM_Config_DB method. This is one of the excellent features added in UVM unlike SystemVerilog, which eliminates passing virtual interface through all hierarchical level until driver is reached.

4. **Monitor**: One monitor simply monitors the response from both MD5 and SHA-256. It collects the output from DUT for every valid transaction. This is where UVM becomes transaction level modelling ignoring low level abstraction noises from DUT. The col-

lected value is sent to Scoreboard using Analysis_port, which is discussed in scoreboard section. Another monitor has reference model embedded in it. This reference model is simply C model which was used to generate expected values. This monitor monitors the input to DUT for every valid transaction and sends the same input to the reference model embedded inside it. The output from reference model is collected and sent to scoreboard for comparison.

5. **Agent**: Agent provides the environment to wrap sequencer, driver and monitor. The components are built and connected here. Agent holds analysis port which is connected to the analysis port of Monitor. The other end of agent's analysis port is connected to the scoreboard.

6. **Scoreboard**: Scoreboard is the component where results are compared and displayed. Monitor sends the value to scoreboard using analysis port through agent. Scoreboard gets these values for DUT and Reference model, compares them and updates the test results. In other words, scoreboard is simply a subscriber to the monitor. Inside scoreboard, there are 4 FIFO that collects the value from DUT and reference model in each. This eliminates the synchronization issues between the two results.

7. **Environment**: The environment is the highest-level abstraction of UVM components which wraps the Scoreboard and Agent. Components are built and connected in the environment. To access any component in UVM framework, it needs to be accessed through the environment.

5.2.2 Verification Flow

The testbench starts with randomization and ends at comparing the output with expected result. However, this testbench drives two DUTs, the SHA-256 and MD5 parallel and irrespective of each other. UVM works in top-down methodology; the sequence is started on sequencer at uvm_test. There is multiple sequences for MD5 and SHA-256, but they all have only one sequencer on which they can run. Every sequence shares single sequencer and only one can run at a point of time. To control this, virtual sequence is used which decides what sequence will run when. Virtual sequence is configured using priority, each sequence is provided with some level of priority. Once, this is called sequence starts sending randomized value to driver. Driver and sequence are connected using peer to peer type connection called seq_item_port. They both work on system call protocols get_next_item, item_done, get, put. They are blocking methods and waits for transaction to be completed. Driver drives the DUT with the random value generated from sequence.

Monitor monitors the ports of DUT and captures every valid transaction. This eliminates unwanted low-level abstraction noises. One monitor is provided with access to reference model to generate expected result. This is done using DPI library which enables transaction between SystemVerilog and C language. Both the monitors send the DUT output and reference output to the analysis port. Scoreboard is connected to other end of the analysis port which samples these values. The overall flow is synchronized using system methods raise_objection and drop_objection. Once all the raised objections are dropped the simulation ends.

Assertions are defined in interface where direct access to input/output is available. Assertions run in parallel during run phase during simulation time. This provides a level of confidence to verify the functionality of DUT. Cover group consists cover points (input and output) which monitors the toggling of bits. Cover group also provides code coverage which implies how much of code is being executed.

Chapter 6

Results and Discussion

This chapter discusses the results of SHA-256 and MD5. IP's were successfully verified using UVM environment as independent IP's and as block level unit. MD5 takes around 900ns for every hash function and SHA-256 takes about 1400ns. Overall 3 testbench's were made and their source code is available in appendix.

6.1 Synthesis Report

The IP's were synthesized on different technology 180 nm, 65 nm and 32 nm. The results are compared and shown in the table6.16.2.

6.2 Simulation Report

SHA-256 and MD5 was verified in RTL level and Gate level simulation. The correctness of the logic matches in RTL level matches gate level. Gate level simulation is technology dependent and gives the exact timing results. Propagation delay of signals, and setup and hold time issues can be monitored in gate level simulation. The test stimulus was generated for different combination of input, which was monitored using coverage. The comparison between number of test cases and coverage is shown in table. Coverage increases as the number of test stimulus increases which is shown in graph. Assertions were monitoring the protocols that DUT needs to maintain at any point of time. Assertion is independent of test cases and is expected to be passing at any point of time. As these assertions do not monitor the output, they cannot verify the correctness of DUT. C model was used to generate expected results, which was used to verify the correctness of device. The IP's works as intended for different combination of stimulus.

Overall time spent in simulation is enormous. For a 32-bit adder in a processor, there are 4.2 billion test cases. Considering all other logic unit, verification of entire processor would cost months of simulation. The motivation behind the block level verification is to increase the simulation throughput. The block level IP verification environment drives both MD5 and SHA-256 parallelly. This method becomes handy when large block of SOC needs to be verified. However, this is limited to the resources available in work station. Here is where UVM becomes flexible as per user needs and stands out on top of verification methodology.

Table 6.1: Synthesis Results

SHA-256 Core	180 nm	32 nm	65 nm
Ports	1858	1858	1858
Nets	7134	6802	6727
Cells	4763	4210	4355
Total Area (um^2)	152585	16959	19623
Gate Count	15304	12909	11777
MD5 Core	180nm	32nm	65nm
Ports	665	661	661
Nets	5304	5534	4889
Cells	4564	4562	4157
Total Area (um^2)	98877	16508	13306
Gate Count	9917	10860	12909

Table 6.2: Timing, Power Results

	SHA-256 CORE	MD5 CORE
Timing @ 100 MHz		
Required Time	0.8	19.7
Arrival Time	- 0.29	- 3.2
Worst Slack	0.5	16.5
Power Consumption @ 100 MHz		
Internal Power	3.8 mW	311.03 uW
Switching Power	630.2 uW	133.8 uW
Total Power	4.43 mW	30.7 nW

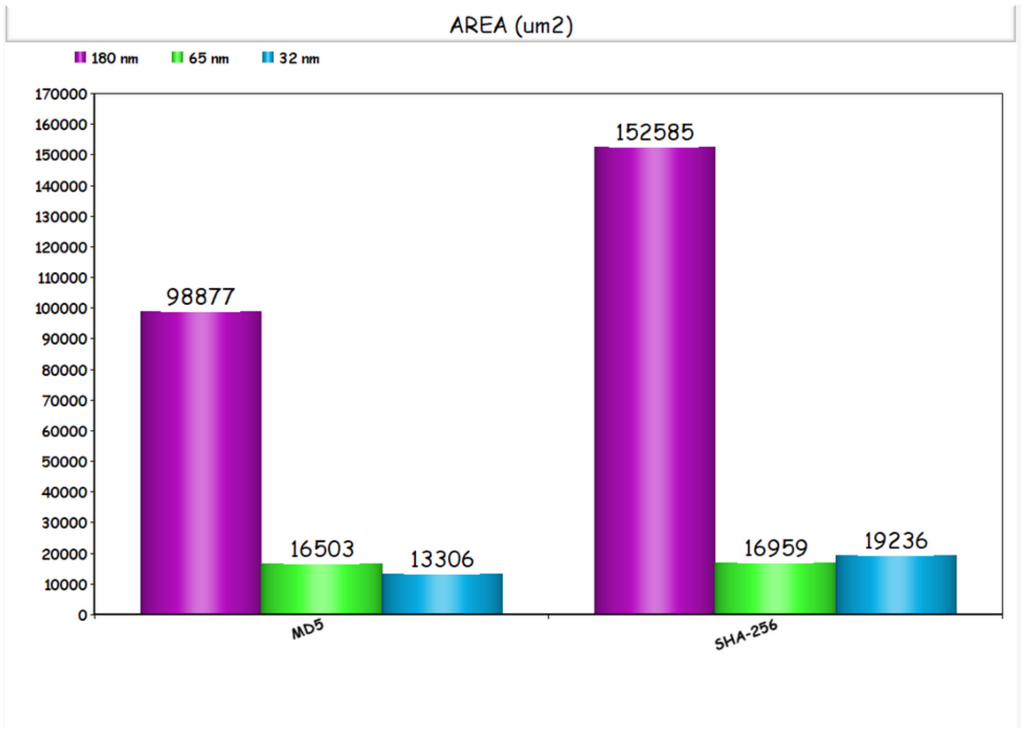

Figure 6.1: MD5 and SHA-256 Area over Different technology

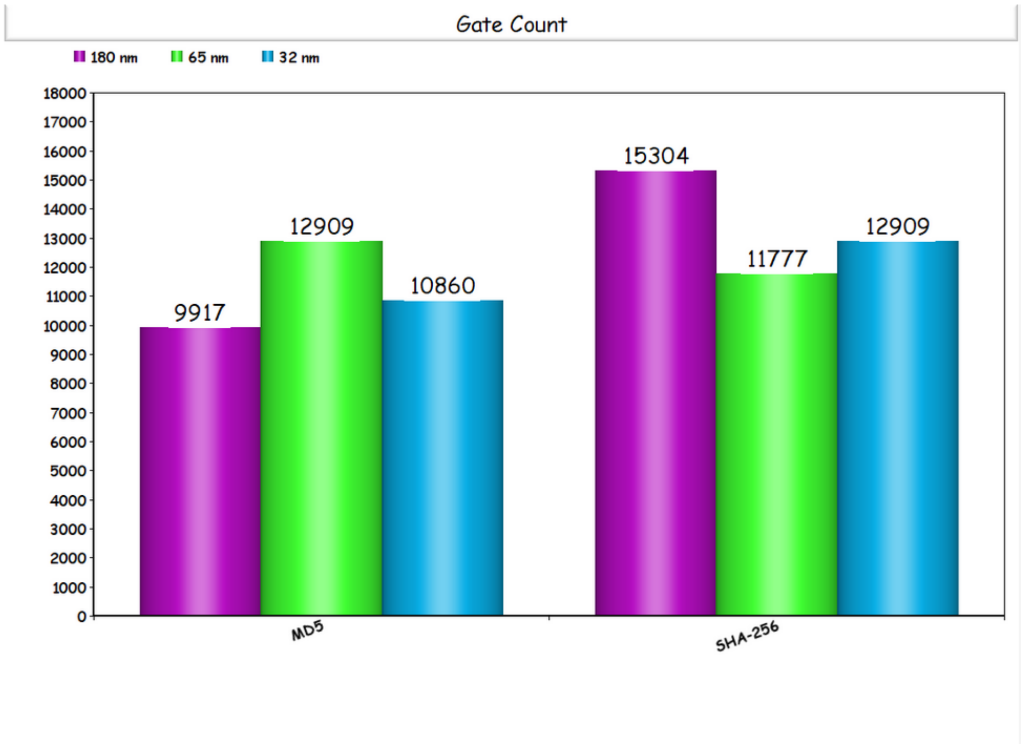

Figure 6.2: MD5 and SHA-256 Gate Count over Different technology

Table 6.3: Coverage across different test cases

Test Cases	SHA- 256 Core		
	Code Coverage %	Functional Coverage %	Assertion Coverage %
50	93.1	34.3	100
100	93.8	44.5	100
1000	93.87	73.82	100
10000	94.9	96.12	100
100000	95.12	97.1	100
1000000	98.23	100	100
	MD5 Core		
50	89.77	8.4	100
100	90.4	55	100
1000	90.44	73.2	100
10000	90.49	96.2	100
100000	90.55	100	100
1000000	94.2	100	100
	MD5 and SHA-256 Combined Core		
50	92.5	23.9	100
100	93	62.15	100
1000	93.2	82.1	100
10000	93.2	98.5	100
100000	93.2	100	100
1000000	93.2	100	100

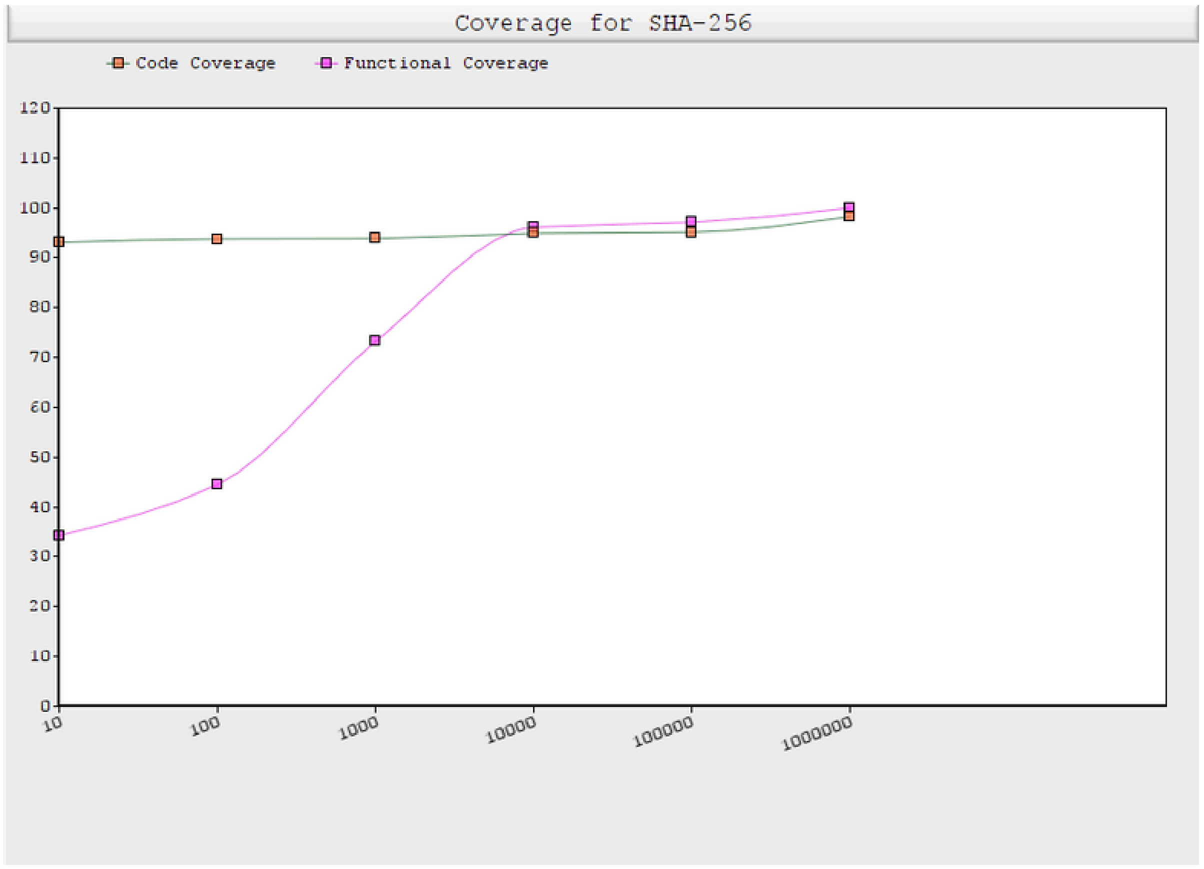

Figure 6.3: SHA-256 Coverage

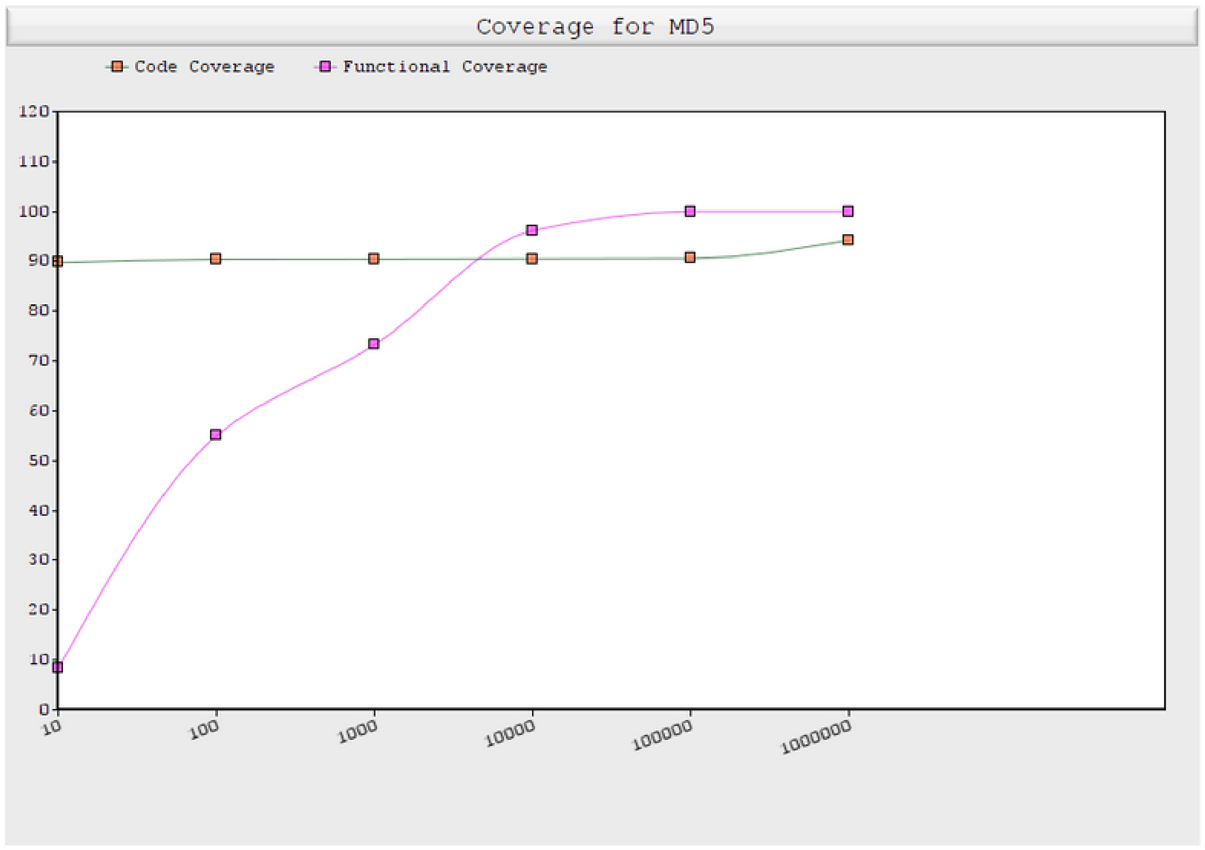

Figure 6.4: MD5 Coverage

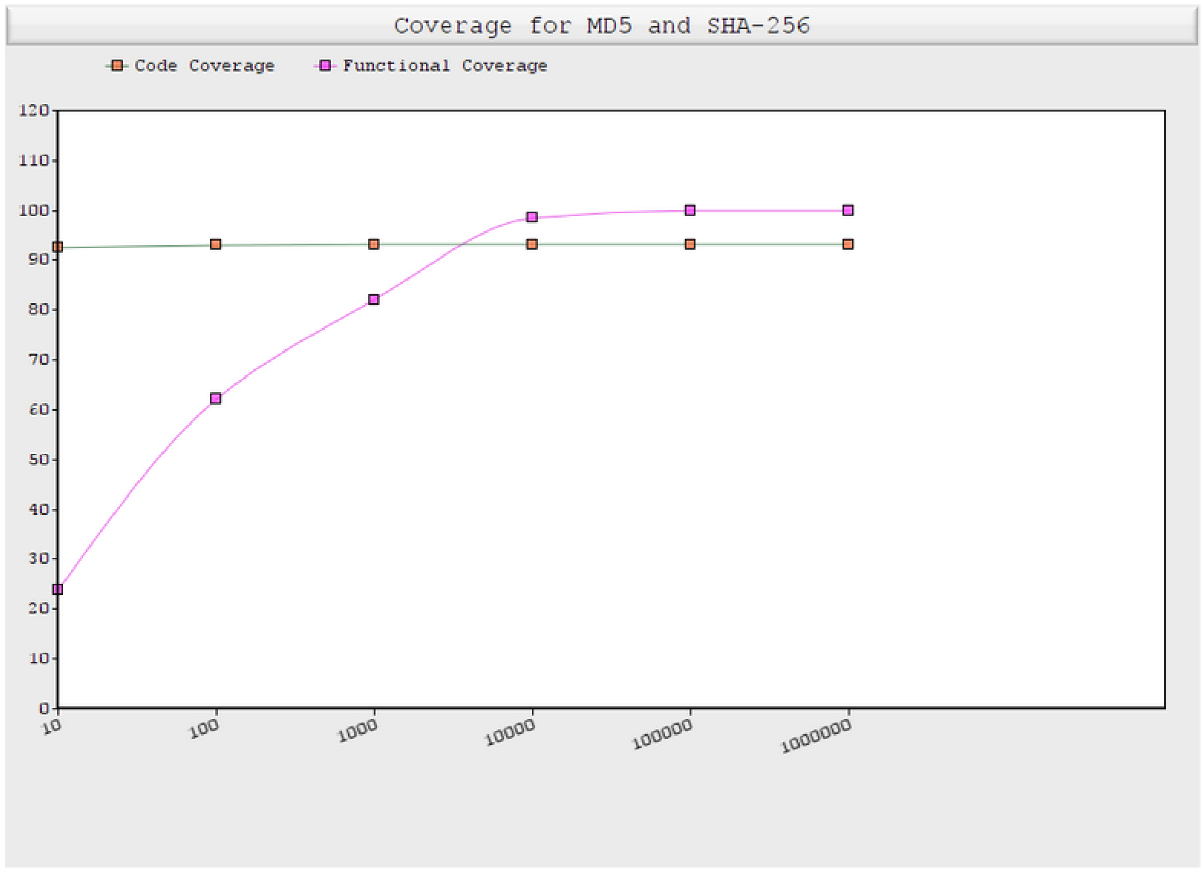

Figure 6.5: MD5 and SHA-256 Coverage

Chapter 7

Conclusion

MD5 and SHA-256 RTL cores were successfully verified using UVM and SystemVerilog. A reference model in C language was used to generate expected values and imported into the UVM framework. DUT output data was compared with expected model values to verify the correctness of the each core. Assertions were used to verify the protocol over which each DUT communicates. Additional assertions were used to check the DUT misbehavior. This provided a level of confidence and ensured each RTL core functions properly.

The MD5 Core takes 32-bit data which can contain up to 4.2 billion different combination values. Randomizing without any constraints might end up not hitting several possible regions. For this purpose, the randomization is divided in 5 regions (4.2 billion / 5), each covering a region. Each region is embedded in a dedicated sequence and randomized several times to ensure high coverage. This method increases the probability of finding corner cases. Coverage metrics were used to keep a track on randomization. A similar method was followed to verify the SHA-256 core. Both cores achieved 100% functional coverage at 1 million input combinations in every region. Note however code coverage remained close to 95%.

7.1 Future work

- From code coverage metrics, we can analyze that the code is not compact, therefore certain RTL blocks can be rewritten to make the code more compact.
- MD5 takes around 800 ns and SHA-256 takes 1400 ns for processing one input message. The RTL could be optimized to improved to reduce these times.
- MD5 and SHA-256 can process only one input at any point of time. Pipelining can be introduced which would allow the cores to take streams of input data.
- The timing on few paths are not very great, therefore certain nets can be remodeled.

References

[1] Bart Preneel. CRYPTOGRAPHIC HASH FUNCTIONS: AN OVERVIEW. In *Proceedings of the 6th International Computer Security and Virus Conference (ICSVC 1993)*, volume 9., pages 461 – 479, 1993. URL: `https://www.esat.kuleuven.be/cosic/publications/article-289.pdf`.

[2] K. Jarvinen, M. Tommiska and J. Skytta. Hardware Implementation Analysis of the MD5 Hash Algorithm. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Big Island, HI, USA*, 2005.

[3] IEEE Standard for Universal Verification Methodology Language Reference Manual, 2017.

[4] K. Salah. A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities. In *2014 9th International Design and Test Symposium (IDT)*, pages 94–99, December 2014. `doi:10.1109/IDT.2014.7038594`.

[5] J. Bromley. If SystemVerilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language. In *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, pages 1–7, September 2013.

[6] Ray Salemi. *The UVM Primer: A Step-by-Step Introduction to the Universal Verification Methodology*. Boston Light Press, 2013.

[7] B. Min and G. Choi. RTL functional verification using excitation and observation coverage,. *Sixth IEEE International High-Level Design Validation and Test Workshop, Monterey, CA, USA*, 2001.

[8] C. Kuznik and W. Muller. Aspect enhanced functional coverage driven verification in the SystemC HDVL. *International SoC Design Conference*, 2011.

[9] P. Ma, Q. Zhao, Y. Fan, M. Liu, and K. Li. The design and verification of packet processing engine model using SystemC. In *2011 International Conference on Electronics, Communications and Control (ICECC)*, pages 1099–1101, September 2011. `doi:10.1109/ICECC.2011.6066395`.

[10] P. D. Mulani. SoC Level Verification Using System Verilog,. *Second International Conference on Emerging Trends in Engineering & Technology, Nagpur*, 2009.

[11] Jordan Cote Zhijie Shi, ChujiaoMa and Bing Wang. *Hardware Implementation Of Hash Functions*. Springer Science+Business Media, LLC, 2012. `doi:10.1007/978-1-4419-8080-9_2`.

[12] Y. W. Hau M.Khalil, M.Nazrin. Implementation of SHA-2 Hash Function for a Digital Signature System-on-Chip in FPGA. *VLSI-eCAD Research Laboratory (VeCAD) Faculty of Electrical Engineering Universiti Teknologi Malaysia (UTM)*, 2008.

[13] Dengguo Feng XiaoyunWang and Hongbo Yu Xuejia Lai. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, 2004. URL: `http://eprint.iacr.org/2004/199`.

[14] D. Lee. Hash Function Vulnerability Index and Hash Chain Attacks. In *2007 3rd IEEE Workshop on Secure Network Protocols*, pages 1–6, October 2007. `doi:10.1109/NPSEC.2007.4371616`.

[15] M. Wang and Y. Li. Hash Function with Variable Output Length. *International Conference on Network and Information Systems for Computers, Wuhan*, 2015.

[16] M. Singh and D. Garg. Choosing Best Hashing Strategies and Hash Functions. In *2009 IEEE International Advance Computing Conference*, pages 50–55, March 2009. `doi:10.1109/IADCC.2009.4808979`.

[17] X. Qiuyun, H. Ligang, L. Qiming, G. Shuqin, and W. Jinhui. The Verification of SHA-256 IP using a semi-automatic UVM platform. In *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, pages 111–115, October 2017. `doi:10.1109/ICEMI.2017.8265733`.

[18] W. Ni and J. Zhang. Research of reusability based on UVM verification. *IEEE 11th International Conference on ASIC*, 2015.

[19] R. Madan, N. Kumar and S. Deb. Pragmatic approaches to implement self-checking mechanism in UVM based TestBench,. *International Conference on Advances in Computer Engineering and Applications, Ghaziabad*, 2015.

[20] R. P. McEvoy, F. M. Crowe, C. C. Murphy and W. P. Marnane. Optimisation of the SHA-2 family of hash functions on FPGAs. *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06), Karlsruhe*, 2006.

[21] S. binti Suhaili and T. Watanabe. Design of high-throughput SHA-256 hash function based on FPGA. *6th International Conference on Electrical Engineering and Informatics (ICEEI), Langkawi*, 2017.

[22] A. T. Hoang, K. Yamazaki and S. Oyanagi. Multi-stage Pipelining MD5 Implementations on FPGA with Data Forwarding. *6th International Symposium on Field-Programmable Custom Computing Machines, Palo Alto, CA*, 2008.

[23] H. M. Heys J. Deepakumara and R. Venkatesan. FPGA implementation of MD5 hash algorithm. In *Canadian Conference on Electrical and Computer Engineering 2001. Conference Proceedings (Cat. No.01TH8555)*, volume 2, 2001.

[24] Chris Spear. *SystemVerilog for Verification*. Springer Science+Business Media, LLC, 2012.

[25] Accellera Systems Initiative (Accellera). *Universal Verification Methodology (UVM) 1.2 User's Guide*, October 2015.

[26] Verification Guide, 2016. SystemVerilog and UVM Resources. URL: `https://www.verificationguide.com/p/home.html`.

[27] *SHA 256 Verilog source code*. URL: `https://github.com/secworks/sha256/tree/master/src/rtl`.

[28] *SHA 256 Reference Model in C*. URL: `https://github.com/B-Con/crypto-algorithms/blob/master/sha256.c`.

[29] *MD5 Verilog Source Code*. URL: `https://github.com/stass/md5_core`.

[30] *MD5 Reference Model in C*. URL: `https://openwall.info/wiki/people/solar/software/public-domain-source-code/md5`.

Appendix A

Source Code for MD5

A.1 MD5 Interface

```
interface md5_interface ;

// ports
logic clk ;
logic reset ;
logic rdy_i ;
logic [31:0] msg_i ;
logic [127:0] hash_o ;
logic rdy_o ;
logic busy_o ;
logic [31:0] syn;
```

```
//—— Assertion 1 : Reset and Rdy signals ———//
// Rdy is followed by reset signal
sequence assertreset ;
(rdy_o == 1'b1) ;
endsequence
property assert_reset ;
@(posedge clk)
(reset == 1'b1 ) |-> ##3 assertreset ;
endproperty
assertion_reset : assert property ( assert_reset) else
   $display ("Data sending failed" ) ;

//—— Assertion 2 : Rdy output and busy signals ——//
// rdy_o is never asserted when busy_o is high
sequence rdybusy ;
(rdy_o == 1'b0) ;
endsequence
property rdy_busy ;
@(posedge clk)
(busy_o == 1'b1 ) |-> rdybusy ;
endproperty
assertion_rdy_busy : assert property ( rdy_busy) else $display
    (" Ready and Busy asserted concurrently" ) ;
```

```
//——— Assertion 3 : hash_o ————//
// Hash_o is never 0 at any point of time
hash0 : assert property  ( @(posedge clk) ## 1 hash_o != 128'
   H0) else $display (" Ready and Busy asserted concurrently"
   ) ;
endinterface : md5_interface
```

A.2 MD5 Sequence and Sequencer

```
class md5_seq_item extends uvm_sequence_item ;
      virtual md5_interface vif;

//———properties of sequence ——————//
logic [127:0] hash_o ;
bit rdy_o;
string modelout ;
bit rdy_i;

// upper and lower case with numbers
rand bit [7:0] num ;
rand bit [7:0] upper ;
rand bit [7:0] lower ;
rand bit [31:0] msg1 ;
rand bit [31:0] msg2 ;
rand bit [7:0] msg3 ;
rand bit [7:0] msg4 ;
rand bit [31:0] msg_i ;
bit [31:0] msg ;
bit [31:0] sequ  ;

constraint new_con { msg1[7:0] inside { [48:57], [65:90],
   [97:122] }; }
```

```
constraint new_con1 { msg1[15:8] inside { [48:57], [65:90],
   [97:122] }; }
constraint new_con2 { msg1[23:16] inside { [48:57], [65:90],
   [97:122] }; }
constraint new_con3 { msg1[31:24] inside { [48:57], [65:90],
   [97:122] }; }



rand byte unsigned temp [];
constraint str_len {temp.size() == 4; }
constraint temp_str_ascii { foreach (temp[i]) temp[i] inside {
    [65:90], [97:122] }; }

// ———————— constructor ——————————————
function new(string name="");
super.new(name);
endfunction: new
//—————————— utility and macros——————————
`uvm_object_utils_begin (md5_seq_item)
`uvm_field_int(hash_o, UVM_ALL_ON)
`uvm_field_int(rdy_o, UVM_ALL_ON)
`uvm_field_int(rdy_i, UVM_ALL_ON)
`uvm_field_int(msg_i, UVM_ALL_ON)

`uvm_object_utils_end
```

```
endclass: md5_seq_item


// ——— sequence1  ———//
class md5_sequence1 extends uvm_sequence #(md5_seq_item);


`uvm_object_utils(md5_sequence1)
//————————— constructor—————————————————
function new(string name="");
super.new(name);
endfunction: new
//——————————randomize and sending to driver ———————————


task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body


endclass : md5_sequence1
```

```
// ----- sequence2 ----//
class md5_sequence2 extends uvm_sequence #(md5_seq_item);

`uvm_object_utils(md5_sequence2)
//--------------- constructor ------------------------
function new(string name="");
super.new(name);
endfunction: new

//-------randomize and sending to driver ----//

task body();
          integer i ;
          md5_seq_item req;
          req = md5_seq_item::type_id::create(.name("req"), .
             contxt(get_full_name()));
          start_item(req);
          assert(req.randomize());
          finish_item(req);
endtask :body

endclass : md5_sequence2

// ----- sequence3 ----//
class md5_sequence3 extends uvm_sequence #(md5_seq_item);
```

```
`uvm_object_utils(md5_sequence3)
//———————— constructor ————————————
function new(string name="");
super.new(name);
endfunction: new

//——————————randomize and sending to driver ——————————

task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body

endclass : md5_sequence3

// ——— sequence4  ————//
class md5_sequence2 extends uvm_sequence #(md5_seq_item);

`uvm_object_utils(md5_sequence4)
```

```
//————————— constructor ———————————————
function new(string name="");
super.new(name);
endfunction: new

//———————randomize and sending to driver ———————————

task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body

endclass : md5_sequence4

// ————Top level Virtual sequence  ——————//
class md5_sequence extends uvm_sequence #(md5_seq_item);

          `uvm_object_utils(md5_sequence)
//————————— constructor ———————————————
function new(string name="md5_sequence");
```

```
        super.new(name);
endfunction: new

md5_sequence1 seq1 ;
md5_sequence2 seq2 ;
md5_sequence2 seq3 ;
md5_sequence2 seq4 ;
md5_sequence2 seq5 ;
//————————randomize and sending to driver ——————————

task body();

seq1 = md5_sequence1::type_id::create("seq1");
seq2 = md5_sequence2::type_id::create("seq2");
seq3 = md5_sequence3::type_id::create("seq3");
seq4 = md5_sequence4::type_id::create("seq4");
seq5 = md5_sequence5::type_id::create("seq5");

m_sequencer.set_arbitration(UVM_SEQ_ARB_USER);
fork
begin
        repeat(4) begin
        seq1.start(m_sequencer, this, 100);
end
end
```

```
begin
        repeat(4) begin
        seq2.start(m_sequencer, this, 200);
end
end
begin
        repeat(4) begin
        seq3.start(m_sequencer, this, 300);
end
end
begin
        repeat(4) begin
        seq4.start(m_sequencer, this, 400);
end
end
begin
        repeat(4) begin
        seq5.start(m_sequencer, this, 500);
end
end
join
                        endtask :body
endclass : md5_sequence


// ---- sequencer   -------//
```

```
typedef uvm_sequencer#(md5_seq_item) md5_sequencer;
```

A.3 MD5 Driver

```
import "DPI-C"  context function string string_sv2c (input
   string str );


class md5_driver extends uvm_driver #(md5_seq_item);
bit [31:0] infake ;
bit [127:0] outfake ;
`uvm_component_utils(md5_driver)
//----------------------------------------------------------
// Virtual Interface
//----------------------------------------------------------
virtual md5_interface vif;
md5_seq_item req ;
//----------------------------------------------------------
// Functional coverage handler
//----------------------------------------------------------
md5_seq_item req_cg ;


//----------------------------------------------------------
// Functional coverage
//----------------------------------------------------------
covergroup md5_cg ;
msg_in:      coverpoint req.msg_i;
msg_in_valid : coverpoint vif.hash_o ;
```

```
cross req.msg_i, vif.hash_o ;
endgroup: md5_cg
//————————————————————————————————————————
// Functional coverage display class
//————————————————————————————————————————
function void display ();
$display ("[%tns] input = %h", $time, req_cg.msg_i);
endfunction : display
//————————————————————————————————————————
// Constructor
//————————————————————————————————————————
function new (string name, uvm_component parent);
super.new(name, parent);
md5_cg = new;
endfunction : new
//————————————————————————————————————————
// build phase
//————————————————————————————————————————
function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        void'(uvm_resource_db#(virtual md5_interface)::
           read_by_name
        (.scope("ifs"), .name("md5_interface"), .val(vif)));
        endfunction: build_phase
        // generating random string
        function string get_str ();
        string str ;
        foreach (req.temp[i])
        str = { str , string '(req.temp[i]) } ;
        return str;
endfunction
//--------------------------------------------
// run phase
//--------------------------------------------
task run_phase(uvm_phase phase);
drive();
endtask : run_phase
virtual task drive();
string randomi ;
integer counter = 0 , state = 0 ;
integer j ;
string modelin ;
string modelout ;
string rtloutstring ;
bit [127:0] modeloutb;
string dummy = "hello";
logic [127:0] rtlout ;

bit [7:0] check1 = 8'H61;
string check2 ;
```

```
bit [127:0] checksum = 128'Habcd1234abcd1234abcd1234abcd1234;
//md5_cg = new();

forever begin

if (counter == 0 ) begin
vif.reset <= 1 ;
#10;
vif.reset <= 0 ;
#10;
vif.syn <= 1 ;
seq_item_port.get_next_item(req);
state = 1 ;
end
@(posedge vif.clk)
begin
case (state)
1: begin
vif.syn = 0 ;

vif.rdy_i = 1'b1 ;
for (j = 0; j < 16; j = j + 1) begin
if (j == 0)
begin
req.msg = {req.msg1, req.msg2 , req.msg3 , req.msg4} ;
```

```
infake = req.msg ;
outfake = vif.hash_o ;
modelin = req.msg1 ;
vif.msg_i = req.msg1 ;
string_sv2c(modelin);
modelout = myscript(dummy);
//$display (" output from model %s", modelout);
end
else if (j == 1)
vif.msg_i = 1<<31;
else if (j == 14)
vif.msg_i = 32'h20000000;
else
vif.msg_i = 0;
#10;
end
counter = counter + 1 ;
if (counter == 1 ) state = 2 ;
end
2: begin
vif.rdy_i = 1'b0 ;
counter = counter + 1 ;
if(vif.rdy_o)
begin
rtlout = {vif.hash_o} ;
```

```
rtloutstring =myscript2(rtlout);
end
if (counter == 70 ) state  = 3 ;
end
3: begin
myscript3 (modelout,rtloutstring , modelin);
req_cg = req ;
//randomi = get_str();
//$display ( " the randomized char is : %0s ", randomi  );
md5_cg.sample();
state = 0;
counter = 0 ;
seq_item_port.item_done();
end
endcase
end
end
endtask : drive
endclass : md5_driver

function string myscript ( string data) ;
string str3;
string str4;
static integer i = 0 ;
integer j ;
```

```
j = i % 2  ;
if (j == 0) str3 = data ;
if ( j == 1) str4 = data ;
i = i + 1 ;
return str3 ;
endfunction : myscript
export "DPI-C"  function  myscript;

function string myscript2 (bit [127:0] data);
reg [3:0]  in [32];
bit [3:0] extract ;
static string out ="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
string out1 ;
static integer i =0 ;
reg [127:0] indata;
indata = data ;
in [31] =  indata [3:0] ;
in [30] =  indata [7:4] ;
in [29] =  indata [11:8] ;
in [28] =  indata [15:12] ;
in [27] =  indata [19:16] ;
in [26] =  indata [23:20] ;
in [25] =  indata [27:24] ;
in [24] =  indata [31:28] ;
in [23] =  indata [35:32] ;
in [22] = indata [39:36] ;
in [21] = indata [43:40] ;
in [20] = indata [47:44] ;
in [19] = indata [51:48] ;
in [18] = indata [55:52] ;
in [17] = indata [59:56] ;
in [16] = indata [63:60] ;
in [15] = indata [67:64] ;
in [14] = indata [71:68] ;
in [13] = indata [75:72] ;
in [12] = indata [79:76] ;
in [11] = indata [83:80] ;
in [10] = indata [87:84] ;
in [9] = indata [91:88] ;
in [8] = indata [95:92] ;
in [7] = indata [99:96] ;
in [6] = indata [103:100] ;
in [5] = indata [107:104] ;
in [4] = indata [111:108] ;
in [3] = indata [115:112] ;
in [2] = indata [119:116] ;
in [1] = indata [123:120] ;
in [0] = indata [127:124] ;
for ( i = 0 ; i < 32 ; i = i + 1 )
begin
```

```
        if (in[i] == 4'b0000) out.putc(i,"0");
        if (in[i] == 4'b0001) out.putc(i,"1");
        if (in[i] == 4'b0010) out.putc(i,"2");
        if (in[i] == 4'b0011) out.putc(i,"3");
        if (in[i] == 4'b0100) out.putc(i,"4");
        if (in[i] == 4'b0101) out.putc(i,"5");
        if (in[i] == 4'b0110) out.putc(i,"6");
        if (in[i] == 4'b0111) out.putc(i,"7");

        if (in[i] == 4'b1000) out.putc(i,"8");
        if (in[i]== 4'b1001) out.putc(i,"9");
        if (in[i] == 4'b1010) out.putc(i,"a");
        if (in[i]== 4'b1011) out.putc(i,"b");
        if (in[i]== 4'b1100) out.putc(i,"c");
        if (in[i] == 4'b1101) out.putc(i,"d");
        if (in[i] == 4'b1110) out.putc(i,"e");
        if (in[i] == 4'b1111) out.putc(i,"f");

end
return out;
endfunction : myscript2

function void myscript3 (string model,rtl , in) ;
static integer i = 0 ;
static integer j = 0 ;
```

```
if (model == rtl )
begin
        i = i + 1 ;
        $display ( "———TEST COUNT:%d     time: %0t          INPUT
            CHARACTER: %s                RTL OUTPUT is: %S
                  MODEL OUTPUT is: %s———> 'TEST PASS'",i,
           $time, in,rtl, model);

end

else
begin
        j = j + 1 ;
        $display ( "——————————————— TEST SEQUENCE fail
           count: %d      Performed at time :%0t            RTL
           OUTPUT is: %S                    MODEL OUTPUT is: %s
           ————————————> Test fail", i, $time, rtl, model);
end
endfunction : myscript3
```

A.4 MD5 Monitor

```
import "DPI-C" function string_sv2c (input string str, output
    string modelcheck);
import "DPI-C"  context function string string_sv2c (input
    string str );
class md5_monitor_before extends uvm_monitor;
  `uvm_component_utils(md5_monitor_before)

logic  [15:0] yome [10] ;
string str , str1 , str2 , str3 , str4 , str5 , str6;
bit [31:0] check ;
bit modelcheck ;
string classin ;

virtual md5_interface vif;
uvm_analysis_port #(md5_seq_item) mon_ap_before;
// constructor
function new ();
super.new(name, parent);
endfunction : new
function void build_phase(uvm_phase phase);
super.build_phase(phase);

void '(uvm_resource_db#(virtual md5_interface)::read_by_name
(.scope("ifs"), .name("md5_interface"), .val(vif)));
mon_ap_before = new(.name("mon_ap_before"), .parent(this));
endfunction: build_phase

_______________________________________________

  // run_phase - convert the signal level activity to
     transaction level.
  // i.e, sample the values on interface signal and assigns to
      transaction class fields
  //-----------------------------------------------
task run_phase(uvm_phase phase);

   md5_seq_item req ;
   req = md5_seq_item::type_id::create
                  (.name("req"), .contxt(get_full_name()));
str5 = "hello from class " ;
forever begin
 @ (posedge vif.clk , vif.rdy_o);
begin
 if (vif.rdy_o )   begin

          if ( !vif.busy_o)begin
                  req.hash_o = vif.hash_o;
                  req.modelout = " hello from scoreboard" ;
```

```
                    //$display (" The value from monitor %h", req.
                       hash_o);
                    mon_ap_before.write(req);
                    //yome = main();
                    //$display ( "%s ", yome);
          end
end
end
end
endtask : run_phase

endclass : md5_monitor_before

//—————————————————————————————————————————//
//——————————————————————————————————————————//
class md5_monitor_after extends uvm_monitor;
  `uvm_component_utils(md5_monitor_after)
virtual md5_interface vif;

//———————————————————————————————————————————————
// analysis port, to send the transaction to scoreboard
//———————————————————————————————————————————————
uvm_analysis_port #(md5_seq_item) mon_ap_after;
//———————————————————————————————————————————————
// Handler // coverage
```

```
//——————————————————————————————————————————
// define sequence handler
md5_seq_item req ;
// sequence handler for coverage
md5_seq_item req_cg ;
//——————————————————————————————————————————
// new – constructor
//——————————————————————————————————————————
function new (string name, uvm_component parent);
super.new(name, parent);
//  md5_cg = new ;
endfunction : new
// run_phase – convert the signal level activity to
   transaction level.
// i.e, sample the values on interface signal and assigns to
   transaction class fields
task run_phase(uvm_phase phase);
        bit [31:0] a ;
        integer counter = 0 , state = 0 ;
   req = md5_seq_item::type_id::create
                (.name("req"), .contxt(get_full_name()));
forever begin
@(posedge vif.clk);
begin
        if(vif.syn == 1'b1 )
        begin
                        state = 1 ;
        end
        if(state == 1 )
        begin
                //req.msg1 = vif.msg_i [31:24];
                counter = counter + 1 ;
                a = req.sequ;
        end
        if (counter == 1 )
        begin
                state = 0 ;
                //a = {req.msg1, req.msg2, req.msg3, req.msg4
                   } ;
                //$display ("The value from the monitor after
                   phase is %H" , a );
                counter = 0 ;
                mon_ap_after.write(req);
        end
end
end
endtask: run_phase
endclass : md5_monitor_after
```

A.5 MD5 Agent

```
class md5_agent extends uvm_agent;
  `uvm_component_utils(md5_agent)
        uvm_analysis_port#(md5_seq_item) agent_ap_before;
        uvm_analysis_port#(md5_seq_item) agent_ap_after;
  md5_sequencer sequencer;
  md5_driver     driver;
  md5_monitor_before md5_mon_before;
  md5_monitor_after  md5_mon_after ;
  //———————————————————————————————————————
  // constructor
  //———————————————————————————————————————
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  //———————————————————————————————————————
  // build_phase
  //———————————————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
```

```
agent_ap_before  = new(.name("agent_ap_before"), .parent(this))
    ;
agent_ap_after   = new(.name("agent_ap_after"), .parent(this));
sequencer        = md5_sequencer::type_id::create(.name("
    sequencer"), .parent(this));
driver           = md5_driver::type_id::create(.name("driver"),
     .parent(this));
md5_mon_before   = md5_monitor_before::type_id::create(.name("
    md5_mon_before"), .parent(this));
md5_mon_after    = md5_monitor_after::type_id::create(.name("
    md5_mon_after"), .parent(this));
endfunction: build_phase
  //———————————————————————————————————————
  // connect_phase - connecting the driver and sequencer port
  //———————————————————————————————————————
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
driver.seq_item_port.connect(sequencer.seq_item_export);
md5_mon_before.mon_ap_before.connect(agent_ap_before);
md5_mon_after.mon_ap_after.connect(agent_ap_after);
endfunction: connect_phase
endclass : md5_agent
```

A.6 MD5 Scoreboard

```
`uvm_analysis_imp_decl(_before)
`uvm_analysis_imp_decl(_after)
//export "DPI-C" function  my_sv_function;



class md5_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(md5_scoreboard)

uvm_analysis_export #(md5_seq_item) sb_export_before;
uvm_analysis_export #(md5_seq_item) sb_export_after;

uvm_tlm_analysis_fifo #(md5_seq_item) before_fifo;
uvm_tlm_analysis_fifo #(md5_seq_item) after_fifo;

md5_seq_item transaction_before;
md5_seq_item transaction_after;



//--------------------------------------------------
// REgistering in Factory
//--------------------------------------------------

function new(string name, uvm_component parent);
```

```
        super.new(name, parent);

        transaction_before      = new("transaction_before");
        transaction_after       = new("transaction_after");
endfunction: new



//————————————————————————————————————————
// constructing
//————————————————————————————————————————
function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        sb_export_before        = new("sb_export_before", this
           );
        sb_export_after         = new("sb_export_after", this)
           ;

        before_fifo             = new("before_fifo", this);
        after_fifo              = new("after_fifo", this);
endfunction: build_phase



//————————————————————————————————————————
// connect phase
```

```
//――――――――――――――――――――――――――――――――――――――――――――――――
function void connect_phase(uvm_phase phase);
        sb_export_before.connect(before_fifo.analysis_export);
        sb_export_after.connect(after_fifo.analysis_export);
endfunction: connect_phase



//――――――――――――――――――――――――――――――――――――――――――――――――
// Run task
//――――――――――――――――――――――――――――――――――――――――――――――――

task run();
forever begin
        before_fifo.get(transaction_before);
        after_fifo.get(transaction_after);
        //$display (" from the scoreboard %H " ,
           transaction_before.hash_o) ;
        //$display ( " %s " , transaction_before.modelout);
        compare();
end
endtask: run




virtual function void compare();
```

```
`uvm_info (" code works until SB" , UVM_LOW) ;
if(transaction_before.out == transaction_after.out) begin
        `uvm_info("compare", {"Test: OK!"}, UVM_LOW);
end else begin
        `uvm_info("compare", {"Test: Fail!"}, UVM_LOW);
end
endfunction: compare
endclass: md5_scoreboard
```

A.7 MD5 Environment

```
class md5_env extends uvm_env;
          `uvm_component_utils(md5_env)

//————————————————————————————————————————
// agent and scoreboard instance
//————————————————————————————————————————
md5_agent        agent;
md5_scoreboard sb;

//————————————————————————————————————————
// constructor
//————————————————————————————————————————
function new(string name, uvm_component parent);
super.new(name, parent);
endfunction : new

//————————————————————————————————————————
// build_phase - create the components
//————————————————————————————————————————

function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        agent    = md5_agent::type_id::create(.name("agent"), .
           parent(this));
        sb       = md5_scoreboard::type_id::create(.name("sb"),
            .parent(this));
endfunction: build_phase

//------------------------------------------
// connect_phase - connecting monitor and scoreboard port
//------------------------------------------
function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        agent.agent_ap_before.connect(sb.sb_export_before);
        agent.agent_ap_after.connect(sb.sb_export_after);
endfunction: connect_phase

endclass : md5_env
```

A.8 MD5 UVM Package

```
package md5_pkg;
        import uvm_pkg::*;

`include "md5_sequencer.sv"
`include "md5_monitor.sv"
`include "md5_driver.sv"
`include "md5_agent.sv"
`include "md5_scoreboard.sv"
`include "md5_config.sv"
`include "md5_env.sv"
`include "md5_ctl_test.sv"
endpackage: md5_pkg
```

A.9 MD5 Test

```
class md5_ctl_test extends uvm_test;
`uvm_component_utils(md5_ctl_test)

//----------------------------------------------//
// sequence / env instance
//--------------------------------------------//
md5_env env;
//---------------------------------------------------------------------
// constructor
//---------------------------------------------------------------------
function new(string name = "md5_test",uvm_component parent=
    null);
super.new(name,parent);
endfunction : new
//---------------------------------------------------------------------
// build_phase
//---------------------------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);

// Create the sequence/ env
env = md5_env::type_id::create(.name("env"), .parent(this));
```

```
endfunction : build_phase
//----------------------------------------------------
// run_phase - starting the test
//----------------------------------------------------
task run_phase(uvm_phase phase);
md5_sequence md5_seq;

phase.raise_objection(.obj(this));
md5_seq = md5_sequence::type_id::create(.name("md5_seq"), .
    contxt(get_full_name()));
assert(md5_seq.randomize());
md5_seq.start(env.agent.sequencer);
phase.drop_objection(.obj(this));
endtask: run_phase

endclass : md5_ctl_test
```

A.10 MD5 Top

```
`include "uvm_macros.svh"
`include "md5_pkg.sv"
`include "md5_interface.sv"
module test;
import uvm_pkg::*;
import md5_pkg::*;
md5_interface vif () ;
md5_ctl top (vif.clk, vif.rdy_i, vif.msg_i, vif.reset, vif.
   hash_o, vif.rdy_o, vif.busy_o );

initial
begin
uvm_resource_db#(virtual md5_interface)::set
(.scope("ifs"), .name("md5_interface"), .val(vif));
$display ( "-------- Test Started-------------") ;
run_test();
$display ( "----------end test------ ") ;
end
initial
begin
vif.clk = 1'b0 ;
end
always
```

```
begin
#5 vif.clk = ~vif.clk ;
end

initial
begin
$timeformat(-9,2,"ns", 16);
$set_coverage_db_name("md5_ctl");
`ifdef SDFSCAN
$sdf_annotate("sdf/md5_ctl_scan.sdf", test.top);
`endif
end

endmodule
```

Appendix B

Source Code for SHA-256

B.1 SHA-256 Interface

```
interface sha256_interface ;

// ports
logic clk ;
logic reset ;
logic [31:0] text_i ;
logic [31:0] text_o ;
logic [2:0] cmd_i ;
logic cmd_w_i ;
logic [3:0] cmd_o;

//--- Assertion 1 : reset / output ---//
```

```
// Rdy is followed by reset signal
sequence resetreset ;
(text_o  == 32'b0) ;
endsequence
property reset_reset ;
@(posedge clk)
        (reset == 1'b1 ) |-> ##1 resetreset ;
endproperty
assertion_reset_reset : assert property ( reset_reset   ) else
    $display (" Text_o misbehaviour with respect to reset" ) ;


//--- Assertion 2 : reset / cmd_o ---//
// Rdy is followed by reset signal
sequence resetresetcmd ;
(cmd_o   == 3'b0) ;
endsequence
property reset_reset_cmd ;
@(posedge clk)
(reset == 1'b1 ) |-> ##1 resetresetcmd ;
endproperty
assertion_reset_reset_cmd : assert property ( reset_reset_cmd
      ) else $display ("cmd_o misbehaviour with respect to
   reset" ) ;
```

```
endinterface : sha256_interface
```

B.2 SHA-256 Sequence and Sequencer

```
// —— sequence item ———//
class sha256_seq_item extends uvm_sequence_item ;
virtual sha256_interface vif;
rand bit [447:0] cga ;
rand bit [447:0] cgb ;
rand bit [7:0] msg4 ;
rand bit [31:0] msg5 ;
// Generating random string

rand byte unsigned a [];
constraint str_len {a.size() == 56; }
constraint temp_str_ascii { foreach (a[i]) a[i] inside {
   [48:57], [65:90], [97:122] }; }

// ———————— constructor ————————————
function new(string name="");
super.new(name);
endfunction: new

//———————— utility and macros ————————
`uvm_object_utils_begin (sha256_seq_item)
`uvm_field_int(cga, UVM_ALL_ON)
`uvm_field_int(cgb, UVM_ALL_ON)
```

```
`uvm_field_int(msg4, UVM_ALL_ON)
`uvm_field_int(msg5, UVM_ALL_ON)
`uvm_object_utils_end
endclass: sha256_seq_item


// --- sequence1  ---//
class sha256_sequence1 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence1)
//---------------- constructor -------------------------
function new(string name="");
super.new(name);
endfunction: new
//-------------randomize and sending to driver ----------------
task body();
integer i ;
sha256_seq_item req;
// -------- change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence1


// —— sequence2  ———//
class sha256_sequence2 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence2)
//———————— constructor ————————————
function new(string name="");
super.new(name);
endfunction: new
//——————randomize and sending to driver ————————
task body();
integer i ;
sha256_seq_item req;
// ———— change value of I as per test cases requires
    —————//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence2


// --- sequence3  ---//
class sha256_sequence3 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence3)
//--------------- constructor -------------------------
function new(string name="");
super.new(name);
endfunction: new
//-----------randomize and sending to driver ---------------
task body();
integer i ;
sha256_seq_item req;
// ------- change value of I as per test cases requires
   -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence3

// --- sequence4  ---//
class sha256_sequence4 extends uvm_sequence #(sha256_seq_item)
   ;
`uvm_object_utils(sha256_sequence4)
//---------------- constructor -------------------------
function new(string name="");
super.new(name);
endfunction: new
//------------randomize and sending to driver ----------------
task body();
integer i ;
sha256_seq_item req;
// ------- change value of I as per test cases requires
   -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
   get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence4

// --- sequence5  ---//
class sha256_sequence5 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence5)
//------------- constructor --------------------
function new(string name="");
super.new(name);
endfunction: new
//----------randomize and sending to driver -------------
task body();
integer i ;
sha256_seq_item req;
// ------ change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence5

// --- Top level sequence --//
class sha256_sequence extends uvm_sequence #(md5_seq_item);
`uvm_object_utils(sha256_sequence)
//--------------- constructor ------------------------
function new(string name="md5_sequence");
super.new(name);
endfunction: new

sha256_sequence1 seq1 ;
sha256_sequence2 seq2 ;
sha256_sequence3 seq3 ;
sha256_sequence4 seq4 ;
sha256_sequence5 seq5 ;

//------------randomize and sending to driver ---------------
task body();
seq1 = sha256_sequence1::type_id::create("seq1");
// seq1.seq_no = 1;
seq2 = sha256_sequence2::type_id::create("seq2");
// seq2.seq_no = 2;
```

```
seq3 = sha256_sequence3::type_id::create("seq3");
// seq2.seq_no = 3;
seq4 = sha256_sequence4::type_id::create("seq4");
// seq2.seq_no = 4;
seq5 = sha256_sequence5::type_id::create("seq5");
// seq2.seq_no = 5;

m_sequencer.set_arbitration(UVM_SEQ_ARB_USER);
fork

begin
repeat(200000) begin
seq1.start(m_sequencer, this, 100); ///Highest priority
end
end

begin
repeat(200000) begin
seq2.start(m_sequencer, this, 200); ///Highest priority
end
end

begin
repeat(200000) begin
seq3.start(m_sequencer, this, 300); ///Highest priority
```

```
end
end

begin
repeat(200000) begin
seq4.start(m_sequencer, this, 400); ///Highest priority
end
end

begin
repeat(200000) begin
seq5.start(m_sequencer, this, 500); ///Highest priority
end
end

join
endtask :body
endclass : sha256_sequence

// ———————————————————————————— sequencer
   ——————————————————————————//
typedef uvm_sequencer#(sha256_seq_item) sha256_sequencer;
```

B.3 SHA-256 Driver

```
import "DPI-C"  context function string cscript (input string
   str , output string );
import "DPI-C"  function void hello ();
`define SHA256_TEST              "
   abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
`define SHA256_TEST_PADDING      {1'b1,63'b0,448'b0,64'd448}
        // 448 bit

class sha256_driver extends uvm_driver #(sha256_seq_item);
bit [447:0] infake ;
bit [127:0] outfake ;
`uvm_component_utils(sha256_driver)
//--------------------------------------------
// Virtual Interface
//--------------------------------------------
virtual sha256_interface vif;
sha256_seq_item req ;
//--------------------------------------------
// Functional coverage handler
//--------------------------------------------
sha256_seq_item req_cg ;
//--------------------------------------------
// Functional coverage
```

```
//------------------------------------------------
covergroup sha256_cg   ;
x :     coverpoint req.cga ;
y :     coverpoint vif.text_o ;
cross req.cga, vif.text_o ;
endgroup : sha256_cg
//------------------------------------------------
// Constructor
//------------------------------------------------
function new (string name, uvm_component parent);
super.new(name, parent);
sha256_cg = new;
endfunction : new
//------------------------------------------------
// build phase
//------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);
void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
endfunction: build_phase
//------------------------------------------------
// string randomize function
//------------------------------------------------
function string get_str ();
```

```
string str ;
foreach (req.a[i])
str = { str , string '(req.a[i]) } ;
return str;
endfunction
//———————————————————————
// run phase
//———————————————————————
task run_phase(uvm_phase phase);
drive();
endtask : run_phase
//———————————————————————
// drive – transaction level to signal level
// drives the value's from seq_item to interface signals
//———————————————————————
virtual task drive();
// properties of task
integer i;
reg [1023:0] all_message;
reg [511:0] tmp_i;
logic [255:0] tmp_o;
integer counter = 0 , state = 0 ;
bit [447:0] randomvalue ;
string modelin ;
string modelout;
string rtlstring;
string dpistring ;

forever begin
//hello () ;
if (counter == 0 ) begin
vif.reset <= 1 ;
#10;
vif.reset <= 0 ;
#10;
seq_item_port.get_next_item(req);
state = 1 ;
end

@(posedge vif.clk)
begin
case (state)
1: begin
vif.cmd_i = 3'b000;
vif.cmd_w_i = 1'b0;
#100;
modelin = get_str ();
infake = modelin ;
randomvalue = modelin;
```

```
//$display (" The random string generated is %s and in bit is
   %h", get_str(),  randomvalue);
//all_message = {randomvalue ,1'b1,63'b0,448'b0,64'd448};
all_message = {randomvalue,`SHA256_TEST_PADDING};
//$display (" all messages %h " , all_message);
tmp_i = all_message[1023:512];

vif.cmd_w_i = 1'b1;
@(posedge vif.clk);
vif.cmd_i = 3'b010;
for (i=0;i<16;i=i+1)
begin
@(posedge vif.clk);
vif.cmd_w_i = 1'b0;
vif.text_i = tmp_i[16*32-1:15*32];
tmp_i = tmp_i << 32;

end
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
while (vif.cmd_o[3])
@(posedge vif.clk);
```

```
#100;
counter = counter + 1 ;
state = 2 ;
end

2: begin
tmp_i = all_message[511:0];
@(posedge vif.clk);
vif.cmd_i = 3'b110;
vif.cmd_w_i = 1'b1;
for (i=0;i<16;i=i+1)
begin
@(posedge vif.clk);
vif.cmd_w_i = 1'b0;
vif.text_i = tmp_i[16*32-1:15*32];
tmp_i = tmp_i << 32;
end
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
while (vif.cmd_o[3])
@(posedge vif.clk);
@(posedge vif.clk);
```

```
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
 vif.cmd_i = 3'b001;
 vif.cmd_w_i = 1'b1;
@(posedge vif.clk);
 vif.cmd_w_i = 1'b0;
 for (i=0;i<8;i=i+1)
 begin
@(posedge vif.clk);
 #1;
 tmp_o[31:0] = vif.text_o ;
 //$display ( " the output is %h " , vif.text_o);
 if ( i < 7 ) tmp_o = tmp_o << 32 ;
 end
 //$display ( " The final output is %h " , tmp_o);
 rtlstring = myscript2 (tmp_o);
 cscript ( modelin, dpistring);
 modelout = myscript1 ("hello");
 //$display ( " the final output from RTL is %s" , rtlstring )
     ;
 //$display ( " the final output from model is %s" , dpistring
     ) ;
 state = 3 ;
```

```
end

3: begin
// calling coverage
sha256_cg.sample() ;
script3 (modelout, rtlstring , modelin);
state = 0 ;
counter = 0 ;
seq_item_port.item_done();
end
endcase
end
end
endtask : drive
endclass : sha256_driver

//----script that exports the value from c model to UVM----//
function string myscript1 ( string data) ;
string str3;
string str4;
static integer i = 0 ;
integer j ;
j = i % 2  ;
if (j == 0) str3 = data ;
if ( j == 1) str4 = data ;
```

```
i = i + 1 ;
return str3 ;
endfunction : myscript1
export "DPI-C" function myscript1;

function string myscript2 ( bit [255:0] hexvalue ) ;
static string out ="
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   ";
bit [3:0] temp;
integer i ;
//$display (" the initial value as string is %h ", hexvalue);
for (i = 0 ; i < 64 ; i = i + 1 )
begin
temp [3:0] = hexvalue [255:252] ;
hexvalue[255:0] = {hexvalue[251:0], 4'b0};
if (temp    == 4'b0000) out.putc( i,"0");
if (temp    == 4'b0001) out.putc( i,"1");
if (temp    == 4'b0010) out.putc( i,"2");
if (temp    == 4'b0011) out.putc( i,"3");
if (temp    == 4'b0100) out.putc( i,"4");
if (temp    == 4'b0101) out.putc( i,"5");
if (temp    == 4'b0110) out.putc( i,"6");
if (temp    == 4'b0111) out.putc( i,"7");
if (temp    == 4'b1000) out.putc( i,"8");
if (temp   == 4'b1001) out.putc( i,"9");
if (temp    == 4'b1010) out.putc( i,"a");
if (temp   == 4'b1011) out.putc( i,"b");
if (temp   == 4'b1100) out.putc( i,"c");
if (temp    == 4'b1101) out.putc( i,"d");
if (temp    == 4'b1110) out.putc( i,"e");
if (temp    == 4'b1111) out.putc( i,"f");
end
return out;
endfunction: myscript2

function void script3 (string model,rtl , in) ;
static integer i = 0 ;
static integer j = 0 ;
$display ( " \n " ) ;
if (model == rtl )
begin
        i = i + 1 ;
        $display ( "———INPUT:%s |RTL OUTPUT is:%S |MODEL
          OUTPUT is: %s———> 'TEST PASS'" ,in,rtl , model);
end

else
begin
```

```
        j = j + 1 ;
        $display ( "—— TEST SEQUENCE fail count: %d
           Performed at time :%0t          RTL OUTPUT is: %S
                          MODEL OUTPUT is: %s——————————>
           Test fail", i, $time, rtl, model);
end

endfunction : script3
```

B.4 SHA-256 Monitor

```
import "DPI-C" function string_sv2c (input string str, output
   string modelcheck);
import "DPI-C"  context function string string_sv2c (input
   string str );

class sha256_monitor_before extends uvm_monitor;
`uvm_component_utils(sha256_monitor_before)
logic  [15:0] yome [10] ;
string str, str1, str2, str3, str4, str5, str6;
bit [31:0] check ;
bit modelcheck ;
string classin ;
//-------------------------------------------------
// Virtual Interface
//-------------------------------------------------
virtual sha256_interface vif;
//-------------------------------------------------
// analysis port, to send the transaction to scoreboard
//-------------------------------------------------
uvm_analysis_port #(sha256_seq_item) mon_ap_before;
//-------------------------------------------------
// new - constructor
//-------------------------------------------------
```

```
function new (string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//—————————————————————————————
// build_phase - getting the interface handle
//—————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);

void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
mon_ap_before = new(.name("mon_ap_before"), .parent(this));
endfunction: build_phase
//—————————————————————————————
// run_phase - convert the signal level activity to
   transaction level.
// i.e, sample the values on interface signal and assigns to
   transaction class fields
//—————————————————————————————
task run_phase(uvm_phase phase);
sha256_seq_item req ;
req = sha256_seq_item::type_id::create
(.name("req"), .contxt(get_full_name()));
str5 = "hello from class " ;
endtask : run_phase
```

```
endclass : sha256_monitor_before



// ——— monitor after/ checksum————//
class sha256_monitor_after extends uvm_monitor;
`uvm_component_utils(sha256_monitor_after)
//———————————————————————————
// Virtual Interface
//———————————————————————————
virtual sha256_interface vif;
//———————————————————————————
// analysis port, to send the transaction to scoreboard
//———————————————————————————
uvm_analysis_port #(sha256_seq_item) mon_ap_after;
//———————————————————————————
// Handler // coverage
//———————————————————————————
// define sequence handler
sha256_seq_item req ;
// sequence handler for coverage
  md5_seq_item req_cg ;
//———————————————————————————
// new - constructor
//———————————————————————————
function new (string name, uvm_component parent);
```

```
super.new(name, parent);
// sha256_cg = new ;
endfunction : new
//------------------------------------------------------------------
// build_phase - getting the interface handle
//------------------------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);
void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
mon_ap_after= new(.name("mon_ap_after"), .parent(this));
endfunction: build_phase
//------------------------------------------------------------------
// run_phase - convert the signal level activity to
    transaction level.
// i.e, sample the values on interface signal and assigns to
    transaction class fields
task run_phase(uvm_phase phase);
bit [31:0] a ;
integer counter = 0 , state = 0 ;
req = sha256_seq_item::type_id::create
(.name("req"), .contxt(get_full_name()));
endtask: run_phase
endclass : sha256_monitor_after
```

B.5 SHA-256 Agent

```
class sha256_agent extends uvm_agent;
`uvm_component_utils(sha256_agent)
//—————————————————————————————————————————
// agent instances
//—————————————————————————————————————————
uvm_analysis_port#(sha256_seq_item) agent_ap_before;
uvm_analysis_port#(sha256_seq_item) agent_ap_after;
//—————————————————————————————————————————
// component instances
//—————————————————————————————————————————
sha256_sequencer sequencer;
sha256_driver    driver;
sha256_monitor_before sha256_mon_before;
sha256_monitor_after  sha256_mon_after ;
//—————————————————————————————————————————
// constructor
//—————————————————————————————————————————
function new (string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//—————————————————————————————————————————
// build_phase
//—————————————————————————————————————————
```

```
function void build_phase(uvm_phase phase);
super.build_phase(phase);
agent_ap_before = new(.name("agent_ap_before"), .parent(this))
    ;
agent_ap_after  = new(.name("agent_ap_after"), .parent(this));
sequencer       = sha256_sequencer::type_id::create(.name("
    sequencer"), .parent(this));
driver          = sha256_driver::type_id::create(.name("driver
    "), .parent(this));
sha256_mon_before        = sha256_monitor_before::type_id::
    create(.name("sha256_mon_before"), .parent(this));
sha256_mon_after         = sha256_monitor_after::type_id::
    create(.name("sha256_mon_after"), .parent(this));
endfunction: build_phase
//――――――――――――――――――――――――――――――――――――――
// connect_phase - connecting the driver and sequencer port
//――――――――――――――――――――――――――――――――――――――
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
driver.seq_item_port.connect(sequencer.seq_item_export);
sha256_mon_before.mon_ap_before.connect(agent_ap_before);
sha256_mon_after.mon_ap_after.connect(agent_ap_after);
endfunction: connect_phase
endclass : sha256_agent
```

B.6 SHA-256 Scoreboard

```
`uvm_analysis_imp_decl(_before)
`uvm_analysis_imp_decl(_after)
export "DPI-C" function  my_sv_function;

class sha256_scoreboard extends uvm_scoreboard;
`uvm_component_utils(sha256_scoreboard)

uvm_analysis_export #(sha256_seq_item) sb_export_before;
uvm_analysis_export #(sha256_seq_item) sb_export_after;
uvm_tlm_analysis_fifo #(sha256_seq_item) before_fifo;
uvm_tlm_analysis_fifo #(sha256_seq_item) after_fifo;
sha256_seq_item transaction_before;
sha256_seq_item transaction_after;

function new(string name, uvm_component parent);
super.new(name, parent);
transaction_before      = new("transaction_before");
transaction_after       = new("transaction_after");
endfunction: new

function void build_phase(uvm_phase phase);
super.build_phase(phase);
sb_export_before        = new("sb_export_before", this);
```

```
sb_export_after          = new("sb_export_after", this);
before_fifo              = new("before_fifo", this);
after_fifo               = new("after_fifo", this);
endfunction: build_phase

function void connect_phase(uvm_phase phase);
sb_export_before.connect(before_fifo.analysis_export);
sb_export_after.connect(after_fifo.analysis_export);
endfunction: connect_phase

task run();
forever begin
before_fifo.get(transaction_before);
after_fifo.get(transaction_after);
compare();
end
endtask: run

virtual function void compare();
`uvm_info (" code works until SB" , UVM_LOW) ;
if(transaction_before.out == transaction_after.out) begin
`uvm_info("compare", {"Test: OK!"}, UVM_LOW);
end else begin
`uvm_info("compare", {"Test: Fail!"}, UVM_LOW);
end
```

```
    endfunction: compare
endclass: sha256_scoreboard
```

B.7 SHA-256 Environment

```
class sha256_env extends uvm_env;
`uvm_component_utils(sha256_env)
//—————————————————————————————
// agent and scoreboard instance
//—————————————————————————————
sha256_agent      agent;
sha256_scoreboard sb;
//—————————————————————————————
// constructor
//—————————————————————————————
function new(string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//—————————————————————————————
// build_phase - create the components
//—————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
agent   = sha256_agent::type_id::create(.name("agent"), .
   parent(this));
sb      = sha256_scoreboard::type_id::create(.name("sb"), .
   parent(this));
```

```
endfunction: build_phase
//———————————————————————————————————————————
// connect_phase – connecting monitor and scoreboard port
//———————————————————————————————————————————
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
agent.agent_ap_before.connect(sb.sb_export_before);
agent.agent_ap_after.connect(sb.sb_export_after);
endfunction: connect_phase
endclass : sha256_env
```

B.8 SHA-256 Package

```
package sha256_pkg;
        import uvm_pkg::*;

`include "sha256_sequencer.sv"
`include "sha256_monitor.sv"
`include "sha256_driver.sv"
`include "sha256_agent.sv"
`include "sha256_scoreboard.sv"
`include "sha256_config.sv"
`include "sha256_env.sv"
`include "sha256_test.sv"

endpackage: sha256_pkg
```

B.9 SHA-256 Test

```
class sha256_test extends uvm_test;
`uvm_component_utils(sha256_test)
//----------------------------------------------------------
// sequence / env instance
//----------------------------------------------------------
sha256_env env;
//----------------------------------------------------------
// constructor
//----------------------------------------------------------
function new(string name = "sha256_test",uvm_component parent=
    null);
super.new(name,parent);
endfunction : new
//----------------------------------------------------------
// build_phase
//----------------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);
// Create the sequence/ env
env = sha256_env::type_id::create(.name("env"), .parent(this))
    ;
endfunction : build_phase
```

```
//——————————————————————————————————
// run_phase – starting the test
//——————————————————————————————————
task run_phase(uvm_phase phase);
sha256_sequence sha256_seq;
phase.raise_objection(.obj(this));
sha256_seq = sha256_sequence::type_id::create(.name("
    sha256_seq"), .contxt(get_full_name()));
assert(sha256_seq.randomize());
sha256_seq.start(env.agent.sequencer);
phase.drop_objection(.obj(this));
endtask: run_phase
endclass : sha256_test
```

B.10 SHA-256 Top

```
`include "uvm_macros.svh"
`include "sha256_pkg.sv"
`include "sha256_interface.sv"
import "DPI-C" function void main();
module test;
import uvm_pkg::*;
import sha256_pkg::*;
sha256_interface vif () ;
sha256 top (vif.clk, vif.reset, vif.text_i, vif.text_o, vif.
    cmd_i, vif.cmd_w_i, vif.cmd_o );
initial
begin
uvm_resource_db#(virtual sha256_interface)::set
(.scope("ifs"), .name("sha256_interface"), .val(vif));
$display ( "--SHA-256 Test Started----------------") ;
run_test();
$display ( " -----end test------- ") ;
end

initial
begin
vif.clk = 1'b0 ;
vif.reset = 1'b0;
```

```
end
always
begin
#5 vif.clk = ~vif.clk ;
end
initial
begin
$timeformat(-9,2,"ns", 16);
$set_coverage_db_name("sha256");
`ifdef SDFSCAN
$sdf_annotate("sdf/md5_ctl_scan.sdf", test.top);
`endif
end
endmodule
```

Appendix C

Source Code for Combined MD5 and SHA-256

C.1 MD5 and SHA-256 Interface

```
interface md5_interface ;
// ports for MD5
logic clk ;
logic reset ;
logic rdy_i ;
logic [31:0] msg_i ;
logic [127:0] hash_o ;
logic rdy_o ;
logic busy_o ;
logic [31:0] syn;
```

```
// ports for SHA256
logic s_clk ;
logic s_reset ;
logic [31:0] text_i ;
logic [31:0] text_o ;
logic [2:0] cmd_i ;
logic cmd_w_i ;
logic [3:0] cmd_o;
logic   scan_in0;
logic   scan_en;
logic   test_mode;
logic  scan_out0;

//-- Assertion 1 : Reset and Rdy signals --//
// Rdy is followed by reset signal
sequence assertreset ;
        (rdy_o == 1'b1) ;
endsequence

property assert_reset ;
        @(posedge clk)
                        (reset == 1'b1 ) |-> ##3 assertreset ;
endproperty
```

```
assertion_reset : assert property ( assert_reset) else
   $display ("Data sending failed" ) ;

//—— Assertion 2 : Rdy output and busy signals ——//
// rdy_o is never asserted when busy_o is high
sequence rdybusy ;
        (rdy_o == 1'b0) ;
endsequence

property rdy_busy ;
        @(posedge clk)
                        (busy_o == 1'b1 ) |—> rdybusy ;
endproperty

assertion_rdy_busy : assert property ( rdy_busy) else $display
    (" Ready and Busy asserted concurrently" ) ;

//—— Assertion 3 : hash_o——//
// Hash_o is never 0 at any point of time
hash0 : assert property  ( @(posedge clk) ## 1 hash_o != 128'
   H0) else $display (" Ready and Busy asserted concurrently"
   ) ;

//—— Assertion 4 SHA : reset / output ——//
// Rdy is followed by reset signal
```

```
sequence resetreset ;
        (text_o  == 32'b0) ;
endsequence

property reset_reset ;
        @(posedge s_clk)
                        (reset == 1'b1 ) |-> ##1 resetreset ;
endproperty

assertion_reset_reset : assert property ( reset_reset   ) else
    $display (" Text_o misbehaviour with respect to reset" ) ;

//-- Assertion 5 SHA : reset / cmd_o -//
// Rdy is followed by reset signal
sequence resetresetcmd ;
        (cmd_o   == 3'b0) ;
endsequence

property reset_reset_cmd ;
        @(posedge s_clk)
                        (reset == 1'b1 ) |-> ##1 resetresetcmd
                            ;
endproperty
```

```
assertion_reset_reset_cmd : assert property ( reset_reset_cmd
      ) else $display ("cmd_o misbehaviour with respect to
   reset" ) ;

//------ Assertion 6 : Reset and Rdy signals --------//
// Rdy is followed by reset signal
sequence assertreset ;
(rdy_o == 1'b1) ;
endsequence
property assert_reset ;
@(posedge clk)
(reset == 1'b1 ) |-> ##3 assertreset ;
endproperty
assertion_reset : assert property ( assert_reset) else
   $display ("Data sending failed" ) ;

//------ Assertion 7 : Rdy output and busy signals ----//
// rdy_o is never asserted when busy_o is high
sequence rdybusy ;
(rdy_o == 1'b0) ;
endsequence
property rdy_busy ;
@(posedge clk)
(busy_o == 1'b1 ) |-> rdybusy ;
endproperty
```

```
assertion_rdy_busy : assert property ( rdy_busy) else $display
    (" Ready and Busy asserted concurrently" ) ;

//——— Assertion 8 : hash_o————//
// Hash_o is never 0 at any point of time
hash0 : assert property  ( @(posedge clk) ## 1 hash_o != 128'
   H0) else $display (" Ready and Busy asserted concurrently"
   ) ;

endinterface : md5_interface
```

C.2 MD5 Sequence and Sequencer

```
class md5_seq_item extends uvm_sequence_item ;
      virtual md5_interface vif;

//———properties of sequence ——————//
logic [127:0] hash_o ;
bit rdy_o;
string modelout ;
bit rdy_i;

// upper and lower case with numbers
rand bit [7:0] num ;
rand bit [7:0] upper ;
rand bit [7:0] lower ;
rand bit [31:0] msg1 ;
rand bit [31:0] msg2 ;
rand bit [7:0] msg3 ;
rand bit [7:0] msg4 ;
rand bit [31:0] msg_i ;
bit [31:0] msg ;
bit [31:0] sequ  ;

constraint new_con { msg1[7:0] inside { [48:57], [65:90],
   [97:122] }; }
```

```
constraint new_con1 { msg1[15:8] inside { [48:57], [65:90],
   [97:122] }; }
constraint new_con2 { msg1[23:16] inside { [48:57], [65:90],
   [97:122] }; }
constraint new_con3 { msg1[31:24] inside { [48:57], [65:90],
   [97:122] }; }



rand byte unsigned temp [];
constraint str_len {temp.size() == 4; }
constraint temp_str_ascii { foreach (temp[i]) temp[i] inside {
    [65:90], [97:122] }; }

// ———————— constructor ————————————
function new(string name="");
super.new(name);
endfunction: new
//———————— utility and macros————————
`uvm_object_utils_begin (md5_seq_item)
`uvm_field_int(hash_o, UVM_ALL_ON)
`uvm_field_int(rdy_o, UVM_ALL_ON)
`uvm_field_int(rdy_i, UVM_ALL_ON)
`uvm_field_int(msg_i, UVM_ALL_ON)

`uvm_object_utils_end
```

```
endclass: md5_seq_item


// ——— sequence1  ----//
class md5_sequence1 extends uvm_sequence #(md5_seq_item);


`uvm_object_utils(md5_sequence1)
//—————— constructor ——————————
function new(string name="");
super.new(name);
endfunction: new
//—————randomize and sending to driver ——————


task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body


endclass : md5_sequence1
```

```
// ——— sequence2  ———-//
class md5_sequence2 extends uvm_sequence #(md5_seq_item);

`uvm_object_utils(md5_sequence2)
//————————— constructor ——————————————
function new(string name="");
super.new(name);
endfunction: new

//———–randomize and sending to driver ———-//

task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body

endclass : md5_sequence2

// ——— sequence3  ———-//
class md5_sequence3 extends uvm_sequence #(md5_seq_item);
```

```
`uvm_object_utils(md5_sequence3)
//————————— constructor ——————————————
function new(string name="");
super.new(name);
endfunction: new

//———————randomize and sending to driver —————————

task body();
        integer i ;
        md5_seq_item req;
        req = md5_seq_item::type_id::create(.name("req"), .
           contxt(get_full_name()));
        start_item(req);
        assert(req.randomize());
        finish_item(req);
endtask :body

endclass : md5_sequence3

// ——— sequence4  ————//
class md5_sequence2 extends uvm_sequence #(md5_seq_item);

`uvm_object_utils(md5_sequence4)
```

```
//————————— constructor ————————————————
function new(string name="");
super.new(name);
endfunction: new

//——————————randomize and sending to driver —————————————


task body();
         integer i ;
         md5_seq_item req;
         req = md5_seq_item::type_id::create(.name("req"), .
            contxt(get_full_name()));
         start_item(req);
         assert(req.randomize());
         finish_item(req);
endtask :body

endclass : md5_sequence4

// —————Top level Virtual sequence ——————//
class md5_sequence extends uvm_sequence #(md5_seq_item);

            `uvm_object_utils(md5_sequence)
//————————— constructor ————————————————
function new(string name="md5_sequence");
```

```
        super.new(name);
endfunction: new

md5_sequence1 seq1 ;
md5_sequence2 seq2 ;
md5_sequence2 seq3 ;
md5_sequence2 seq4 ;
md5_sequence2 seq5 ;
//-----------randomize and sending to driver ---------------

task body();

seq1 = md5_sequence1::type_id::create("seq1");
seq2 = md5_sequence2::type_id::create("seq2");
seq3 = md5_sequence3::type_id::create("seq3");
seq4 = md5_sequence4::type_id::create("seq4");
seq5 = md5_sequence5::type_id::create("seq5");

m_sequencer.set_arbitration(UVM_SEQ_ARB_USER);
fork
begin
        repeat(4) begin
        seq1.start(m_sequencer, this, 100);
end
end
```

```
begin
        repeat(4) begin
        seq2.start(m_sequencer, this, 200);
end
end
begin
        repeat(4) begin
        seq3.start(m_sequencer, this, 300);
end
end
begin
        repeat(4) begin
        seq4.start(m_sequencer, this, 400);
end
end
begin
        repeat(4) begin
        seq5.start(m_sequencer, this, 500);
end
end
join
                        endtask :body
endclass : md5_sequence


// ---- sequencer   -------//
```

```
typedef uvm_sequencer#(md5_seq_item) md5_sequencer;
```

C.3 SHA-256 Sequence and Sequencer

```
// ——  sequence item ———//
class sha256_seq_item extends uvm_sequence_item ;
virtual sha256_interface vif;
rand bit [447:0] cga ;
rand bit [447:0] cgb ;
rand bit [7:0] msg4 ;
rand bit [31:0] msg5 ;
// Generating random string

rand byte unsigned a [];
constraint str_len {a.size() == 56; }
constraint temp_str_ascii { foreach (a[i]) a[i] inside {
   [48:57],  [65:90], [97:122] }; }

// ——————— constructor ————————————
function new(string name="");
super.new(name);
endfunction: new

//———————— utility and macros————————
`uvm_object_utils_begin (sha256_seq_item)
`uvm_field_int(cga, UVM_ALL_ON)
`uvm_field_int(cgb, UVM_ALL_ON)
```

```
`uvm_field_int(msg4, UVM_ALL_ON)
`uvm_field_int(msg5, UVM_ALL_ON)
`uvm_object_utils_end
endclass: sha256_seq_item

// --- sequence1 ---//
class sha256_sequence1 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence1)
//---------------- constructor ------------------------
function new(string name="");
super.new(name);
endfunction: new
//------------randomize and sending to driver ----------------
task body();
integer i ;
sha256_seq_item req;
// -------- change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence1


// --- sequence2  ---//
class sha256_sequence2 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence2)
//--------------- constructor -----------------------
function new(string name="");
super.new(name);
endfunction: new
//------------randomize and sending to driver ---------------
task body();
integer i ;
sha256_seq_item req;
// ------- change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence2


// --- sequence3  ---//
class sha256_sequence3 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence3)
//---------------- constructor -------------------------
function new(string name="");
super.new(name);
endfunction: new
//------------randomize and sending to driver ------------
task body();
integer i ;
sha256_seq_item req;
// -------- change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence3

// --- sequence4  ---//
class sha256_sequence4 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence4)
//------------- constructor --------------------
function new(string name="");
super.new(name);
endfunction: new
//----------randomize and sending to driver -------------
task body();
integer i ;
sha256_seq_item req;
// ------ change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence4


// --- sequence5  ---//
class sha256_sequence5 extends uvm_sequence #(sha256_seq_item)
    ;
`uvm_object_utils(sha256_sequence5)
//------------- constructor -----------------------
function new(string name="");
super.new(name);
endfunction: new
//----------randomize and sending to driver --------------
task body();
integer i ;
sha256_seq_item req;
// ------ change value of I as per test cases requires
    -----//
i = 5;
repeat(i) begin
req = sha256_seq_item::type_id::create(.name("req"), .contxt(
    get_full_name()));
start_item(req);
assert(req.randomize());
```

```
finish_item(req);
end
endtask :body
endclass : sha256_sequence5

// --- Top level sequence  --//
class sha256_sequence extends uvm_sequence #(md5_seq_item);
`uvm_object_utils(sha256_sequence)
//---------------- constructor ------------------------
function new(string name="md5_sequence");
super.new(name);
endfunction: new

sha256_sequence1 seq1 ;
sha256_sequence2 seq2 ;
sha256_sequence3 seq3 ;
sha256_sequence4 seq4 ;
sha256_sequence5 seq5 ;

//------------randomize and sending to driver ---------------
task body();
seq1 = sha256_sequence1::type_id::create("seq1");
// seq1.seq_no = 1;
seq2 = sha256_sequence2::type_id::create("seq2");
// seq2.seq_no = 2;
```

```
seq3 = sha256_sequence3::type_id::create("seq3");
// seq2.seq_no = 3;
seq4 = sha256_sequence4::type_id::create("seq4");
// seq2.seq_no = 4;
seq5 = sha256_sequence5::type_id::create("seq5");
// seq2.seq_no = 5;

m_sequencer.set_arbitration(UVM_SEQ_ARB_USER);
fork

begin
repeat(200000) begin
seq1.start(m_sequencer, this, 100); ///Highest priority
end
end

begin
repeat(200000) begin
seq2.start(m_sequencer, this, 200); ///Highest priority
end
end

begin
repeat(200000) begin
seq3.start(m_sequencer, this, 300); ///Highest priority
```

```
end
end

begin
repeat(200000) begin
seq4.start(m_sequencer, this, 400); ///Highest priority
end
end

begin
repeat(200000) begin
seq5.start(m_sequencer, this, 500); ///Highest priority
end
end

join
endtask :body
endclass : sha256_sequence

// ———————————————————————— sequencer
   ————————————————————————//
typedef uvm_sequencer#(sha256_seq_item) sha256_sequencer;
```

C.4 MD5 Driver

```
import "DPI-C"  context function string string_sv2c (input
    string str );

class md5_driver extends uvm_driver #(md5_seq_item);
bit [31:0] infake ;
bit [127:0] outfake ;
`uvm_component_utils(md5_driver)
//----------------------------------------------
// Virtual Interface
//----------------------------------------------
virtual md5_interface vif;
md5_seq_item req ;
//----------------------------------------------
// Functional coverage handler
//----------------------------------------------
md5_seq_item req_cg ;

//----------------------------------------------
// Functional coverage
//----------------------------------------------
covergroup md5_cg ;
msg_in:       coverpoint req.msg_i;
msg_in_valid : coverpoint vif.hash_o ;
```

```
cross req.msg_i, vif.hash_o ;
endgroup: md5_cg
//------------------------------------------------------------
// Functional coverage display class
//------------------------------------------------------------
function void display ();
$display ("[%tns] input = %h", $time, req_cg.msg_i);
endfunction : display
//------------------------------------------------------------
// Constructor
//------------------------------------------------------------
function new (string name, uvm_component parent);
super.new(name, parent);
md5_cg = new;
endfunction : new
//------------------------------------------------------------
// build phase
//------------------------------------------------------------
function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        void'(uvm_resource_db#(virtual md5_interface)::
           read_by_name
        (.scope("ifs"), .name("md5_interface"), .val(vif)));
        endfunction: build_phase
        // generating random string
        function string get_str ();
        string str ;
        foreach (req.temp[i])
        str = { str , string '(req.temp[i]) } ;
        return str;
endfunction
//————————————————————————
// run phase
//————————————————————————
task run_phase(uvm_phase phase);
drive();
endtask : run_phase
//————————————————————————
// drive — transaction level to signal level
// drives the value's from seq_item to interface signals
//————————————————————————
virtual task drive();
string randomi ;
integer counter = 0 , state = 0 ;
integer j ;
string modelin ;
string modelout ;
string rtloutstring ;
bit [127:0] modeloutb;
string dummy = "hello";
```

```
logic [127:0] rtlout ;

bit [7:0] check1 = 8'H61;
string check2 ;
bit [127:0] checksum = 128'Habcd1234abcd1234abcd1234abcd1234;
//md5_cg = new();

forever begin

if (counter == 0 ) begin
vif.reset <= 1 ;
#10;
vif.reset <= 0 ;
#10;
vif.syn <= 1 ;
seq_item_port.get_next_item(req);
state = 1 ;
end
@(posedge vif.clk)
begin
case (state)
1: begin
vif.syn = 0 ;

vif.rdy_i = 1'b1 ;
```

```
for (j = 0; j < 16; j = j + 1) begin
if (j == 0)
begin
req.msg = {req.msg1, req.msg2 , req.msg3 , req.msg4} ;
infake = req.msg ;
outfake = vif.hash_o ;
modelin = req.msg1 ;
vif.msg_i = req.msg1 ;
string_sv2c(modelin);
modelout = myscript(dummy);
//$display (" output from model %s", modelout);
end
else if (j == 1)
vif.msg_i = 1<<31;
else if (j == 14)
vif.msg_i = 32'h20000000;
else
vif.msg_i = 0;
#10;
end
counter = counter + 1 ;
if (counter == 1 ) state = 2 ;
end
2: begin
vif.rdy_i = 1'b0 ;
```

```
counter = counter +  1 ;
if(vif.rdy_o)
begin
rtlout = {vif.hash_o} ;
rtloutstring =myscript2(rtlout);
end
if (counter == 70 ) state  = 3 ;
end
3: begin
myscript3 (modelout,rtloutstring , modelin);
req_cg = req ;
//randomi = get_str();
//$display ( " the randomized char is : %0s ", randomi  );
md5_cg.sample();
state = 0;
counter = 0 ;
seq_item_port.item_done();
end
endcase
end
end
endtask : drive
endclass : md5_driver


//———script that exports the value from c model to UVM———//
```

```
function string myscript ( string data) ;
string str3;
string str4;
static integer i = 0 ;
integer j ;
j = i % 2  ;
if (j == 0) str3 = data ;
if ( j == 1) str4 = data ;
i = i + 1 ;
return str3 ;
endfunction : myscript
export "DPI-C"  function  myscript;

function string myscript2 (bit [127:0] data);
reg [3:0]  in [32];
bit [3:0] extract ;
static string out ="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
string out1 ;
static integer i =0 ;
reg [127:0] indata;
indata = data ;
in [31] =  indata [3:0] ;
in [30] =  indata [7:4] ;
in [29] =  indata [11:8] ;
in [28] =  indata [15:12] ;
in [27] =  indata [19:16] ;
in [26] =  indata [23:20] ;
in [25] =  indata [27:24] ;
in [24] =  indata [31:28] ;
in [23] =  indata [35:32] ;
in [22] =  indata [39:36] ;
in [21] =  indata [43:40] ;
in [20] =  indata [47:44] ;
in [19] =  indata [51:48] ;
in [18] =  indata [55:52] ;
in [17] =  indata [59:56] ;
in [16] =  indata [63:60] ;
in [15] =  indata [67:64] ;
in [14] =  indata [71:68] ;
in [13] =  indata [75:72] ;
in [12] =  indata [79:76] ;
in [11] =  indata [83:80] ;
in [10] =  indata [87:84] ;
in [9] =  indata [91:88] ;
in [8] =  indata [95:92] ;
in [7] =  indata [99:96] ;
in [6] =  indata [103:100] ;
in [5] =  indata [107:104] ;
in [4] =  indata [111:108] ;
in [3] =  indata [115:112] ;
in [2] =  indata [119:116] ;
in [1] =  indata [123:120] ;
in [0] =  indata [127:124] ;
for ( i = 0 ; i < 32 ; i = i + 1 )
begin
        if (in[i] == 4'b0000) out.putc(i,"0");
        if (in[i] == 4'b0001) out.putc(i,"1");
        if (in[i] == 4'b0010) out.putc(i,"2");
        if (in[i] == 4'b0011) out.putc(i,"3");
        if (in[i] == 4'b0100) out.putc(i,"4");
        if (in[i] == 4'b0101) out.putc(i,"5");
        if (in[i] == 4'b0110) out.putc(i,"6");
        if (in[i] == 4'b0111) out.putc(i,"7");

        if (in[i] == 4'b1000) out.putc(i,"8");
        if (in[i]== 4'b1001) out.putc(i,"9");
        if (in[i] == 4'b1010) out.putc(i,"a");
        if (in[i]== 4'b1011) out.putc(i,"b");
        if (in[i]== 4'b1100) out.putc(i,"c");
        if (in[i] == 4'b1101) out.putc(i,"d");
        if (in[i] == 4'b1110) out.putc(i,"e");
        if (in[i] == 4'b1111) out.putc(i,"f");

end
return out;
```

```
endfunction : myscript2

function void myscript3 (string model,rtl , in) ;
static integer i = 0 ;
static integer j = 0 ;

if (model == rtl )
begin
        i = i + 1 ;
        $display ( "------TEST COUNT:%d     time: %0t          INPUT
            CHARACTER: %s               RTL OUTPUT is: %S
                   MODEL OUTPUT is: %s---> 'TEST PASS'",i,
           $time , in,rtl , model);

end

else
begin
        j = j + 1 ;
        $display ( "-------------------------- TEST SEQUENCE fail
           count: %d     Performed at time :%0t           RTL
           OUTPUT is: %S                   MODEL OUTPUT is: %s
           ----------------> Test fail", i, $time, rtl , model);
end
endfunction : myscript3
```

C.5 SHA-256 Driver

```
import "DPI-C"  context function string cscript (input string
   str , output string );
import "DPI-C"  function void hello ();
`define SHA256_TEST              "
   abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
`define SHA256_TEST_PADDING      {1'b1,63'b0,448'b0,64'd448}
        // 448 bit

class sha256_driver extends uvm_driver #(sha256_seq_item);
bit [447:0] infake ;
bit [127:0] outfake ;
`uvm_component_utils(sha256_driver)
//------------------------------------------------------------
// Virtual Interface
//------------------------------------------------------------
virtual sha256_interface vif;
sha256_seq_item req ;
//------------------------------------------------------------
// Functional coverage handler
//------------------------------------------------------------
sha256_seq_item req_cg ;
//------------------------------------------------------------
// Functional coverage
```

```
//------------------------------------------------------------
covergroup sha256_cg   ;
x :     coverpoint req.cga ;
y :     coverpoint vif.text_o ;
cross req.cga, vif.text_o ;
endgroup : sha256_cg
//------------------------------------------------------------
// Constructor
//------------------------------------------------------------
function new (string name, uvm_component parent);
super.new(name, parent);
sha256_cg = new;
endfunction : new
//------------------------------------------------------------
// build phase
//------------------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);
void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
endfunction: build_phase
//------------------------------------------------------------
// string randomize function
//------------------------------------------------------------
function string get_str ();
```

```
string str ;
foreach (req.a[i])
str = { str , string '(req.a[i]) } ;
return str;
endfunction
//----------------------------------------------------------
// run phase
//----------------------------------------------------------
task run_phase(uvm_phase phase);
drive();
endtask : run_phase
//----------------------------------------------------------
// drive - transaction level to signal level
// drives the value's from seq_item to interface signals
//----------------------------------------------------------
virtual task drive();
// properties of task
integer i;
reg [1023:0] all_message;
reg [511:0] tmp_i;
logic [255:0] tmp_o;
integer counter = 0 , state = 0 ;
bit [447:0] randomvalue ;
string modelin ;
string modelout;
string rtlstring;
string dpistring ;

forever begin
//hello () ;
if (counter == 0 ) begin
vif.reset <= 1 ;
#10;
vif.reset <= 0 ;
#10;
seq_item_port.get_next_item(req);
state = 1 ;
end

@(posedge vif.clk)
begin
case (state)
1: begin
vif.cmd_i = 3'b000;
vif.cmd_w_i = 1'b0;
#100;
modelin = get_str ();
infake = modelin ;
randomvalue = modelin;
```

```
//$display (" The random string generated is %s and in bit is
   %h", get_str(),  randomvalue);
//all_message = {randomvalue ,1'b1,63'b0,448'b0,64'd448};
all_message = {randomvalue ,`SHA256_TEST_PADDING};
//$display (" all messages %h " , all_message);
tmp_i = all_message[1023:512];

vif.cmd_w_i = 1'b1;
@(posedge vif.clk);
vif.cmd_i = 3'b010;
for (i=0;i<16;i=i+1)
begin
@(posedge vif.clk);
vif.cmd_w_i = 1'b0;
vif.text_i = tmp_i[16*32-1:15*32];
tmp_i = tmp_i << 32;

end
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
while (vif.cmd_o[3])
@(posedge vif.clk);
```

```
#100;
counter = counter + 1 ;
state = 2 ;
end

2: begin
tmp_i = all_message[511:0];
@(posedge vif.clk);
vif.cmd_i = 3'b110;
vif.cmd_w_i = 1'b1;
for (i=0;i<16;i=i+1)
begin
@(posedge vif.clk);
vif.cmd_w_i = 1'b0;
vif.text_i = tmp_i[16*32-1:15*32];
tmp_i = tmp_i << 32;
end
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
while (vif.cmd_o[3])
@(posedge vif.clk);
@(posedge vif.clk);
```

```
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
@(posedge vif.clk);
vif.cmd_i = 3'b001;
vif.cmd_w_i = 1'b1;
@(posedge vif.clk);
vif.cmd_w_i = 1'b0;
for (i=0;i<8;i=i+1)
begin
@(posedge vif.clk);
#1;
tmp_o[31:0] = vif.text_o ;
//$display ( " the output is %h " , vif.text_o);
if ( i < 7 ) tmp_o = tmp_o << 32 ;
end
//$display ( " The final output is %h " , tmp_o);
rtlstring = myscript2 (tmp_o);
cscript ( modelin, dpistring);
modelout = myscript1 ("hello");
//$display ( " the final output from RTL is %s" , rtlstring )
    ;
//$display ( " the final output from model is %s" , dpistring
    ) ;
state = 3 ;
```

```
end


3: begin
// calling coverage
sha256_cg.sample() ;
script3 (modelout, rtlstring , modelin);
state = 0 ;
counter = 0 ;
seq_item_port.item_done();
end
endcase
end
end
endtask : drive
endclass : sha256_driver


//——script that exports the value from c model to UVM———//
function string myscript1 ( string data) ;
string str3;
string str4;
static integer i = 0 ;
integer j ;
j = i % 2  ;
if (j == 0) str3 = data ;
if ( j == 1) str4 = data ;
```

```
i = i + 1 ;
return str3 ;
endfunction : myscript1
export "DPI-C" function myscript1;

function string myscript2 ( bit [255:0] hexvalue ) ;
static string out ="
   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
   ";
bit [3:0] temp;
integer i ;
//$display (" the initial value as string is %h ", hexvalue);
for (i = 0 ; i < 64 ; i = i + 1 )
begin
temp [3:0] = hexvalue [255:252] ;
hexvalue[255:0] = {hexvalue[251:0], 4'b0};
if (temp    == 4'b0000) out.putc( i,"0");
if (temp    == 4'b0001) out.putc( i,"1");
if (temp    == 4'b0010) out.putc( i,"2");
if (temp    == 4'b0011) out.putc( i,"3");
if (temp    == 4'b0100) out.putc( i,"4");
if (temp    == 4'b0101) out.putc( i,"5");
if (temp    == 4'b0110) out.putc( i,"6");
if (temp    == 4'b0111) out.putc( i,"7");
if (temp   == 4'b1000) out.putc( i,"8");
if (temp  == 4'b1001) out.putc( i,"9");
if (temp   == 4'b1010) out.putc( i,"a");
if (temp  == 4'b1011) out.putc( i,"b");
if (temp  == 4'b1100) out.putc( i,"c");
if (temp   == 4'b1101) out.putc( i,"d");
if (temp   == 4'b1110) out.putc( i,"e");
if (temp   == 4'b1111) out.putc( i,"f");
end
return out;
endfunction: myscript2

function void script3 (string model,rtl , in) ;
static integer i = 0 ;
static integer j = 0 ;
$display ( " \n " ) ;
if (model == rtl )
begin
        i = i + 1 ;
        $display ( "-----INPUT:%s |RTL OUTPUT is:%S |MODEL
          OUTPUT is: %s---> 'TEST PASS'" ,in,rtl , model);
end

else
begin
```

```
        j = j + 1 ;
        $display ( "—— TEST SEQUENCE fail count: %d
           Performed at time :%0t          RTL OUTPUT is: %S
                          MODEL OUTPUT is: %s—————————————>
           Test fail", i, $time, rtl , model);
end

endfunction : script3
```

C.6 MD5 Monitor

```
 //import "DPI-C" function string_sv2c (input string str,
    output string modelcheck);
//import "DPI-C"  context function string string_sv2c (input
   string str );
class md5_monitor_before extends uvm_monitor;
  `uvm_component_utils(md5_monitor_before)


logic  [15:0] yome [10] ;
string str, str1, str2, str3, str4, str5, str6;
bit [31:0] check ;
bit modelcheck ;
string classin ;


//---------------------------------------------
// Virtual Interface
//---------------------------------------------
virtual md5_interface vif;


//---------------------------------------------
// analysis port, to send the transaction to scoreboard
//---------------------------------------------
uvm_analysis_port #(md5_seq_item) mon_ap_before;
```

```
//—————————————————————————————————————————
// new – constructor
//—————————————————————————————————————————
function new (string name, uvm_component parent);
super.new(name, parent);
endfunction : new

//—————————————————————————————————————————
// build_phase – getting the interface handle
//—————————————————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);

void'(uvm_resource_db#(virtual md5_interface)::read_by_name
(.scope("ifs"), .name("md5_interface"), .val(vif)));
mon_ap_before = new(.name("mon_ap_before"), .parent(this));
endfunction: build_phase
```

———————————————————————

```
  // run_phase – convert the signal level activity to
     transaction level.
  // i.e, sample the values on interface signal and assigns to
      transaction class fields
```

```
   //———————————————————————————————————————
task run_phase(uvm_phase phase);

    md5_seq_item req ;
    req = md5_seq_item::type_id::create
                    (.name("req"), .contxt(get_full_name()));
str5 = "hello from class " ;
forever begin
 @ (posedge vif.clk , vif.rdy_o);
begin
 if (vif.rdy_o )   begin

          if ( !vif.busy_o)begin
                    req.hash_o = vif.hash_o;
                    req.modelout = " hello from scoreboard" ;
                    //$display (" The value from monitor %h", req.
                       hash_o);
                    mon_ap_before.write(req);
                    //yome = main();
                    //$display ( "%s ", yome);
          end
end
end
end
endtask : run_phase
```

```
endclass : md5_monitor_before

//—————————————————————————————————//
// ——— monitor after/ checksum————//
//——————————————————————————————————//
class md5_monitor_after extends uvm_monitor;
  `uvm_component_utils(md5_monitor_after)

//————————————————————————————————————————
// Virtual Interface
//————————————————————————————————————————
virtual md5_interface vif;

//————————————————————————————————————————
// analysis port, to send the transaction to scoreboard
//————————————————————————————————————————
uvm_analysis_port #(md5_seq_item) mon_ap_after;

//————————————————————————————————————————
// Handler // coverage
//————————————————————————————————————————

// define sequence handler
md5_seq_item req ;
```

```
// sequence handler for coverage

md5_seq_item req_cg ;

//—————————————————————————————————————
// new - constructor
//—————————————————————————————————————
function new (string name, uvm_component parent);
super.new(name, parent);
//  md5_cg = new ;
endfunction : new

//—————————————————————————————————————
// build_phase - getting the interface handle
//—————————————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
void '(uvm_resource_db#(virtual md5_interface)::read_by_name
                (.scope("ifs"), .name("md5_interface"), .val(
                    vif)));
mon_ap_after= new(.name("mon_ap_after"), .parent(this));
endfunction: build_phase

//—————————————————————————————————————
```

```
// run_phase - convert the signal level activity to
   transaction level.
// i.e, sample the values on interface signal and assigns to
   transaction class fields


task run_phase(uvm_phase phase);
        bit [31:0] a ;
        integer counter = 0 , state = 0 ;
   req = md5_seq_item::type_id::create
                (.name("req"), .contxt(get_full_name()));

forever begin
@(posedge vif.clk);
begin

        if(vif.syn == 1'b1 )
        begin
                        state = 1 ;
        end


        if(state == 1 )
        begin
                //req.msg1 = vif.msg_i [31:24];
```

```
            counter = counter + 1 ;
            a = req.sequ;

      end

      if (counter == 1 )
      begin
            state = 0 ;
            //a = {req.msg1, req.msg2, req.msg3, req.msg4
               } ;
            //$display ("The value from the monitor after
               phase is %H" , a );
            counter = 0 ;

            mon_ap_after.write(req);
      end
end
end
endtask: run_phase
endclass : md5_monitor_after
```

C.7 SHA-256 Monitor

```
import "DPI-C" function string_sv2c (input string str, output
   string modelcheck);
import "DPI-C"  context function string string_sv2c (input
   string str );

class sha256_monitor_before extends uvm_monitor;
`uvm_component_utils(sha256_monitor_before)
logic  [15:0] yome [10] ;
string str, str1, str2, str3, str4, str5, str6;
bit [31:0] check ;
bit modelcheck ;
string classin ;
//---------------------------------------------------------------
// Virtual Interface
//---------------------------------------------------------------
virtual sha256_interface vif;
//---------------------------------------------------------------
// analysis port, to send the transaction to scoreboard
//---------------------------------------------------------------
uvm_analysis_port #(sha256_seq_item) mon_ap_before;
//---------------------------------------------------------------
// new - constructor
//---------------------------------------------------------------
```

```
function new (string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//---------------------------------------------
// build_phase - getting the interface handle
//---------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);

void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
mon_ap_before = new(.name("mon_ap_before"), .parent(this));
endfunction: build_phase
//---------------------------------------------
// run_phase - convert the signal level activity to
   transaction level.
// i.e, sample the values on interface signal and assigns to
   transaction class fields
//---------------------------------------------
task run_phase(uvm_phase phase);
sha256_seq_item req ;
req = sha256_seq_item::type_id::create
(.name("req"), .contxt(get_full_name()));
str5 = "hello from class " ;
endtask : run_phase
```

```
endclass : sha256_monitor_before



// ——— monitor after/ checksum————//
class sha256_monitor_after extends uvm_monitor;
`uvm_component_utils(sha256_monitor_after)
//———————————————————————————————
// Virtual Interface
//———————————————————————————————
virtual sha256_interface vif;
//———————————————————————————————
// analysis port, to send the transaction to scoreboard
//———————————————————————————————
uvm_analysis_port #(sha256_seq_item) mon_ap_after;
//———————————————————————————————
// Handler // coverage
//———————————————————————————————
// define sequence handler
sha256_seq_item req ;
// sequence handler for coverage
  md5_seq_item req_cg ;
//———————————————————————————————
// new - constructor
//———————————————————————————————
function new (string name, uvm_component parent);
```

```
super.new(name, parent);
// sha256_cg = new ;
endfunction : new
//—————————————————————————————
// build_phase – getting the interface handle
//—————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
void'(uvm_resource_db#(virtual sha256_interface)::read_by_name
(.scope("ifs"), .name("sha256_interface"), .val(vif)));
mon_ap_after= new(.name("mon_ap_after"), .parent(this));
endfunction: build_phase
//—————————————————————————————
// run_phase – convert the signal level activity to
   transaction level.
// i.e, sample the values on interface signal and assigns to
   transaction class fields
task run_phase(uvm_phase phase);
bit [31:0] a ;
integer counter = 0 , state = 0 ;
req = sha256_seq_item::type_id::create
(.name("req"), .contxt(get_full_name()));s
endtask: run_phase
endclass : sha256_monitor_after
```

C.8 MD5 and SHA-256 Agent

```
class md5_agent extends uvm_agent;
  `uvm_component_utils(md5_agent)
//———————————————————————————————
// agent instances
//———————————————————————————————
uvm_analysis_port#(md5_seq_item) agent_ap_before;
uvm_analysis_port#(md5_seq_item) agent_ap_after;
uvm_analysis_port#(sha256_seq_item) sha_agent_ap_before;
uvm_analysis_port#(sha256_seq_item) sha_agent_ap_after;
//———————————————————————————————
// component instances
//———————————————————————————————
md5_sequencer sequencer;
md5_driver    driver;
sha256_sequencer sha_seqr ;
sha256_driver sha_driver ;
md5_monitor_before md5_mon_before;
md5_monitor_after  md5_mon_after ;
sha256_monitor_before sha256_mon_before;
sha256_monitor_after  sha256_mon_after ;
//———————————————————————————————
// constructor
//———————————————————————————————
```

```
function new (string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//------------------------------------------------
// build_phase
//------------------------------------------------
function void build_phase(uvm_phase phase);
super.build_phase(phase);
agent_ap_before = new(.name("agent_ap_before"), .parent(this))
   ;
agent_ap_after  = new(.name("agent_ap_after"), .parent(this));
sha_agent_ap_before     = new(.name("sha_agent_ap_before"), .
   parent(this));
sha_agent_ap_after      = new(.name("sha_agent_ap_after"), .
   parent(this));
// MD5 driver and seqr
sequencer        = md5_sequencer::type_id::create(.name("
   sequencer"), .parent(this));
driver           = md5_driver::type_id::create(.name("driver"),
    .parent(this));
// SHA256 driver and seqr
sha_seqr         = sha256_sequencer::type_id::create(.name("
   sha_seqr"), .parent(this));
sha_driver       = sha256_driver::type_id::create(.name("
   sha_driver"), .parent(this));
```

```
// MD5 monitors
md5_mon_before   = md5_monitor_before::type_id::create(.name("
   md5_mon_before"), .parent(this));
md5_mon_after    = md5_monitor_after::type_id::create(.name("
   md5_mon_after"), .parent(this));
// sha 256 monitors
sha256_mon_before        = sha256_monitor_before::type_id::
   create(.name("sha256_mon_before"), .parent(this));
sha256_mon_after         = sha256_monitor_after::type_id::
   create(.name("sha256_mon_after"), .parent(this));
endfunction: build_phase
//------------------------------------------------------
// connect_phase - connecting the driver and sequencer port
//------------------------------------------------------
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
// MD5 driver and seqr connection
driver.seq_item_port.connect(sequencer.seq_item_export);
// SHA256 driver and seqr connection
sha_driver.seq_item_port.connect(sha_seqr.seq_item_export);

md5_mon_before.mon_ap_before.connect(agent_ap_before);
md5_mon_after.mon_ap_after.connect(agent_ap_after);
sha256_mon_before.mon_ap_before.connect(sha_agent_ap_before);
sha256_mon_after.mon_ap_after.connect(sha_agent_ap_after);
```

```
endfunction: connect_phase

endclass : md5_agent
```

C.9 MD5 Scoreboard

```
`uvm_analysis_imp_decl(_before)
`uvm_analysis_imp_decl(_after)
//export "DPI-C" function  my_sv_function;
class md5_scoreboard extends uvm_scoreboard;
`uvm_component_utils(md5_scoreboard)
uvm_analysis_export #(md5_seq_item) sb_export_before;
uvm_analysis_export #(md5_seq_item) sb_export_after;
uvm_tlm_analysis_fifo #(md5_seq_item) before_fifo;
uvm_tlm_analysis_fifo #(md5_seq_item) after_fifo;
md5_seq_item transaction_before;
md5_seq_item transaction_after;
//-------------------------------------------------------
// REgistering in Factory
//-------------------------------------------------------
function new(string name, uvm_component parent);
super.new(name, parent);
transaction_before      = new("transaction_before");
transaction_after       = new("transaction_after");
endfunction: new
//-------------------------------------------------------
// constructing
//-------------------------------------------------------
function void build_phase(uvm_phase phase);
```

```
super.build_phase(phase);
sb_export_before        = new("sb_export_before", this);
sb_export_after         = new("sb_export_after", this);
before_fifo             = new("before_fifo", this);
after_fifo              = new("after_fifo", this);
endfunction: build_phase
//---------------------------------------------------
// connect phase
//---------------------------------------------------
function void connect_phase(uvm_phase phase);
sb_export_before.connect(before_fifo.analysis_export);
sb_export_after.connect(after_fifo.analysis_export);
endfunction: connect_phase
//---------------------------------------------------
// Run task
//---------------------------------------------------
task run();
forever begin
before_fifo.get(transaction_before);
after_fifo.get(transaction_after);
//$display (" from the scoreboard %H " , transaction_before.
   hash_o) ;
//$display ( " %s " , transaction_before.modelout);
compare();
end
```

```
endtask: run
virtual function void compare();
`uvm_info (" code works until SB" , UVM_LOW) ;
if(transaction_before.out == transaction_after.out) begin
`uvm_info("compare", {"Test: OK!"}, UVM_LOW);
end else begin
`uvm_info("compare", {"Test: Fail!"}, UVM_LOW);
end
endfunction: compare
endclass: md5_scoreboard
```

C.10 SHA-256 Scoreboard

```
`uvm_analysis_imp_decl(_before)
`uvm_analysis_imp_decl(_after)
export "DPI-C" function  my_sv_function;
class sha256_scoreboard extends uvm_scoreboard;
        `uvm_component_utils(sha256_scoreboard)

        uvm_analysis_export #(sha256_seq_item)
           sb_export_before;
        uvm_analysis_export #(sha256_seq_item) sb_export_after
           ;

        uvm_tlm_analysis_fifo #(sha256_seq_item) before_fifo;
        uvm_tlm_analysis_fifo #(sha256_seq_item) after_fifo;

        sha256_seq_item transaction_before;
        sha256_seq_item transaction_after;

        function new(string name, uvm_component parent);
                super.new(name, parent);

                transaction_before      = new("
                   transaction_before");
                transaction_after         = new("
                   transaction_after");
        endfunction: new

        function void build_phase(uvm_phase phase);
                super.build_phase(phase);

                sb_export_before          = new("
                   sb_export_before", this);
                sb_export_after           = new("sb_export_after
                   ", this);

                before_fifo               = new("before_fifo",
                   this);
                after_fifo                = new("after_fifo",
                   this);
        endfunction: build_phase

        function void connect_phase(uvm_phase phase);
                sb_export_before.connect(before_fifo.
                   analysis_export);
                sb_export_after.connect(after_fifo.
                   analysis_export);
        endfunction: connect_phase
```

```
        task run();
                forever begin
                        before_fifo.get(transaction_before);
                        after_fifo.get(transaction_after);

                        compare();
                end
        endtask: run
        virtual function void compare();

                `uvm_info (" code works until SB" , UVM_LOW) ;
                if(transaction_before.out == transaction_after
                   .out) begin
                        `uvm_info("compare", {"Test: OK!"},
                          UVM_LOW);
                end else begin
                        `uvm_info("compare", {"Test: Fail!"},
                          UVM_LOW);
                end
        endfunction: compare
endclass: sha256_scoreboard
```

C.11 MD5 and SHA-256 Environment

```
class md5_env extends uvm_env;
`uvm_component_utils(md5_env)
//————————————————————————————————————————
// agent and scoreboard instance
//————————————————————————————————————————
md5_agent       agent;
md5_scoreboard  sb;
sha256_scoreboard sha256_sb ;
//————————————————————————————————————————
// constructor
//————————————————————————————————————————
function new(string name, uvm_component parent);
super.new(name, parent);
endfunction : new
//————————————————————————————————————————
// build_phase - create the components
//————————————————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
agent   = md5_agent::type_id::create(.name("agent"), .parent(
   this));
sb      = md5_scoreboard::type_id::create(.name("sb"), .parent
   (this));
```

```
sha256_sb      = sha256_scoreboard::type_id::create(.name("
    sha256_sb"), .parent(this));
endfunction: build_phase
//----------------------------------------------------------------
// connect_phase - connecting monitor and scoreboard port
//----------------------------------------------------------------
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
agent.agent_ap_before.connect(sb.sb_export_before);
agent.agent_ap_after.connect(sb.sb_export_after);
agent.sha_agent_ap_before.connect(sha256_sb.sb_export_before);
agent.sha_agent_ap_after.connect(sha256_sb.sb_export_after);
endfunction: connect_phase
endclass : md5_env
```

C.12 MD5 and SHA-256 UVM Package

```
package md5sha256_pkg;
        import uvm_pkg::*;



`include "md5_sequencer.sv"
`include "sha256_sequencer.sv"
`include "sha256_driver.sv"
`include "md5_driver.sv"







`include "md5_monitor.sv"
`include "sha256_monitor.sv"
`include "md5_agent.sv"

`include "md5_scoreboard.sv"
`include "sha256_scoreboard.sv"
`include "md5_config.sv"
`include "md5_env.sv"
`include "md5_sha256_test.sv"
```

```
endpackage: md5sha256_pkg
```

C.13 MD5 and SHA-256 Test

```
class md5_sha256_test extends uvm_test;
`uvm_component_utils(md5_sha256_test)

// sequence / env instance
//————————————————————————————————————————
md5_env env;
// constructor
function new(string name = "md5_test",uvm_component parent=
   null);
super.new(name,parent);
endfunction : new
//————————————————————————————————————————
// build_phase
//————————————————————————————————————————
function void build_phase(uvm_phase phase);
super.build_phase(phase);
// Create the sequence/ env
env = md5_env::type_id::create(.name("env"), .parent(this));
endfunction : build_phase
//————————————————————————————————————————
// run_phase – starting the test
//————————————————————————————————————————
```

```
task run_phase(uvm_phase phase);
md5_sequence md5_seq;
sha256_sequence sha256_seq ;
phase.raise_objection(.obj(this));
md5_seq = md5_sequence::type_id::create(.name("md5_seq"), .
   contxt(get_full_name()));
sha256_seq = sha256_sequence::type_id::create(.name("
   sha256_seq"), .contxt(get_full_name()));
assert(md5_seq.randomize());
assert(sha256_seq.randomize());

//Running two driver in parallel
fork
begin
sha256_seq.start(env.agent.sha_seqr) ;
end

begin
md5_seq.start(env.agent.sequencer);
end
join

phase.drop_objection(.obj(this));
endtask: run_phase
endclass :md5_sha256_test
```

C.14 MD5 and SHA-256 Top

```
`include "uvm_macros.svh"
`include "md5sha256_pkg.sv"
`include "md5_interface.sv"
import "DPI-C" function void main();
module test;
import uvm_pkg::*;
import md5sha256_pkg::*;
md5_interface vif () ;
//md5_ctl top (vif.clk, vif.rdy_i, vif.msg_i, vif.reset, vif.
   hash_o, vif.rdy_o, vif.busy_o );
//sha256 top1 (vif.s_clk, vif.s_reset, vif.text_i, vif.text_o,
    vif.cmd_i, vif.cmd_w_i, vif.cmd_o );
md5_sha256 top (vif.clk, vif.reset, vif.s_clk, vif.s_reset,
   vif.text_i, vif.text_o, vif.cmd_i, vif.cmd_w_i, vif.cmd_o,
   vif.scan_in0, vif.scan_en, vif.test_mode, vif.scan_out0,
   vif.rdy_i, vif.msg_i,  vif.hash_o, vif.rdy_o, vif.busy_o );

initial
begin
uvm_resource_db#(virtual md5_interface)::set
(.scope("ifs"), .name("md5_interface"), .val(vif));
$display ( "--- Test Started------") ;
//my_sv_function();
```

```
run_test();
$display ( " ---end test--- ") ;
end

initial
begin
vif.clk = 1'b0 ;
vif.s_clk = 1'b0 ;
end
always
begin

#5 vif.s_clk = ~vif.s_clk ;
end
always
begin
#5 vif.clk = ~vif.clk ;
end

initial
begin

$timeformat(-9,2,"ns", 16);
$set_coverage_db_name("md5_sha256");
`ifdef SDFSCAN
```

```
$sdf_annotate("sdf/md5_sha256_scan.sdf", test.top);
`endif
end
endmodule
```