Embedded UVM

Opensource Emulation and Functional Verification

Puneet Goel <puneet@coverify.com>

COVERIFY

October 17, 2019

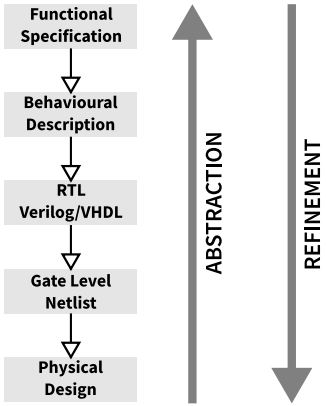
In this section …

Domain Knowledge

- SoC Design Flow
- Functional Verification
- Verification Trends and Challenges
- Introducing Embedded UVM
- Multicore Testbenches
- Productivity
- Interfacing with RTL Simulations
- Emulation
- The DUT

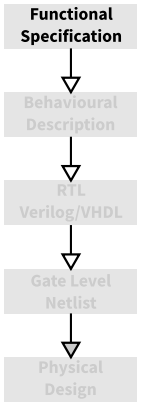
SoC Design Overview

- From Specification to Silicon
- Involves multiple steps
- Each step refines the output from previous step
- The whole process could take anything from 1 year to 3 or more years to complete



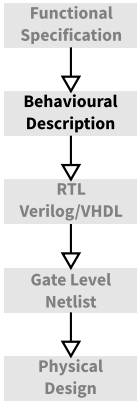
Functional Specification

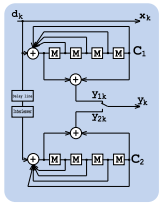
- Specifications are provided as a **Document**
- Specifications are prepared by the marketing team on basis of interaction with prospective customers
- As SoC design implementation takes multiple years to complete, the specifications may change



Behavioural Description

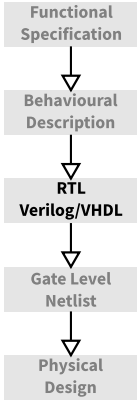
- The Specification to Behavioural Description flow is a manual process
- Done using Matlab or another System Design tool
  - Often only the Data Path is modeled
  - Matlab models functionality, with only a wide approximation of timing/performance
- When System Design performance is critical, SystemC is used
  - In some cases Virtual Platforms are used to model the whole system
  - Virtual Platforms enable early development of Software





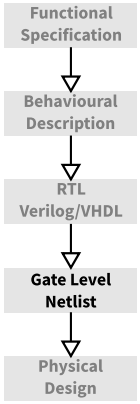
RTL

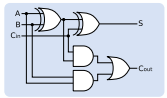
- More often, RTL is coded directly from Specification document
  - Behavioural compilers exist, but are not generic enough
- RTL is the most prominent of all the abstraction levels in multiple ways:
  - RTL is implementation agnostic – can map to multiple platforms – including ASIC, FPGA etc
  - RTL models complete functionality of the SoC in a cycle accurate fashion
  - All the steps beyond RTL abstraction are implementation dependent and are highly automated



Gate Level Netlist

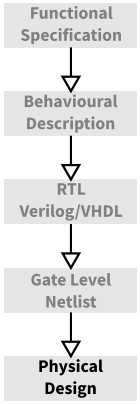
- RTL to Gate (synthesis) process is highly automated
- Gate Level Netlist is implementation technology dependent
- DFT (Design For Testability) and MBIST are introduced at this stage
- Gate Level Netlist is a purely structural Verilog model
- Gate Level Simulations are often an important criteria for tapeout
  - Gate Level Simulations allow timing back-annotation from PD

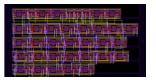




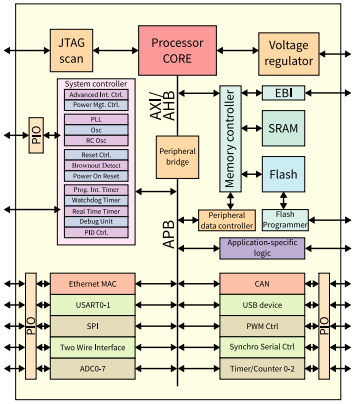
Physical Design

- Gates and Modules are placed and connected
- Highly automated, but involves multiple complex maneuvers:
  - Placement of cells/components
  - Clock Tree Synthesis
  - Routing, parasitic extraction
  - Static Timing Analysis
  - Layout Vs Schematic (LVS)



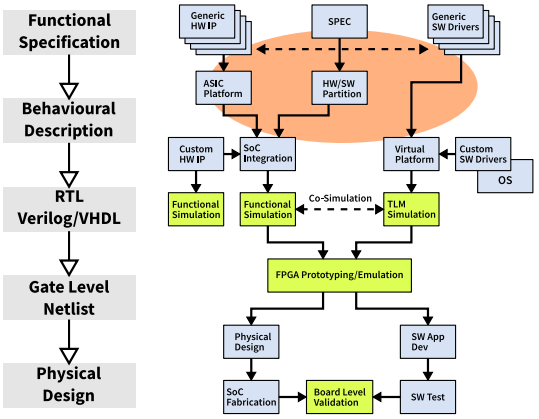


SoC Design



CVERIFY

SoC Design Flow

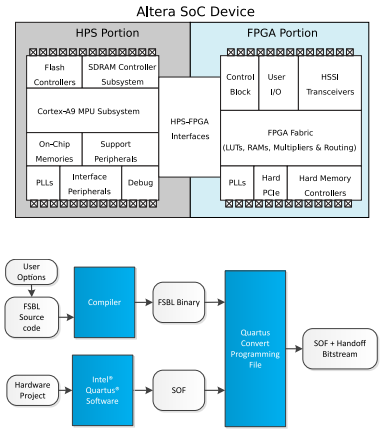


FRONTEND

CVERIFY

SoCFPGA and FPGA Accelerator Architecture and Design Flow

- No SoC integration – only the custom block is designed in RTL
- Integrates directly with Software via device-tree and SOF
- How would you verify an SoCFPGA device?



CVERIFY

The World needs more Verification

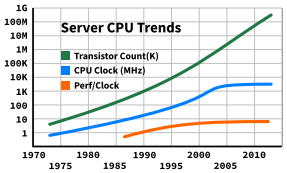
- Less than 40% chip designs achieve first pass success
  - About 80% of failures are due to undetected flaws in design
  - A re-spin costs a million dollar and months of delay
- In 1994, famous FDIV bug costed Intel $475 million
- In 2011, Sandy Bridge SATA Bug costed $700 million
- As complexity of design increases, more and more time is being spent on verification and validation, and yet bugs are escaping unnoticed to the customers

CVERIFY

FRONTEND

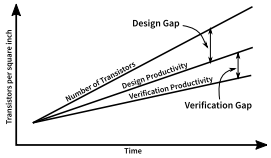
Moore's Law

The number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented.



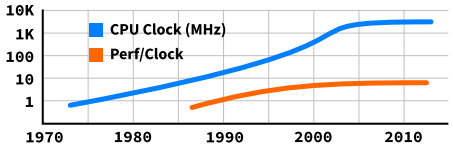
Verification Gap

- Unavailability of Verification Engineers
- Lack of Skills
- Verification Tool Limitations



CVERIFY

Moore's Law is Dead!



A corollary of Moore's law predicted that the Transistor Speed (CPU Clock Frequency) will double up every couple of years. Ever since year 2005, CPU clock frequency has not followed the trend.

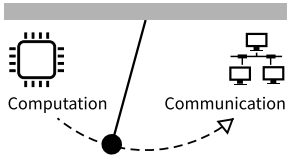
- Per Square inch, a CPU emits more heat than a room heater
- At 4GHz, light can travel only 7.5 cm
  - Electric current travels slower
  - Capacitive and Resistive effects make signal transfer much slower
- To enable multicore computing, CPUs in future would be running slower
- Until 2005, simulation server capacity increased every year thus offsetting the need for better tools to handle increased Logic

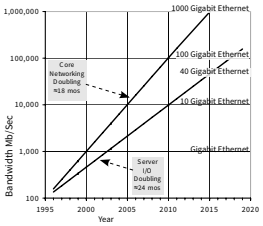
CVERIFY

He Who Rules The Data, Rules The World

Compute Performance and Moore's Law

- Until 2005, thanks to Moore's law, compute performance progressively matched throughput requirements
- But more recently, thanks to the demise of Moore's law, the pendulum has swung too far on the right (breaking the roof)





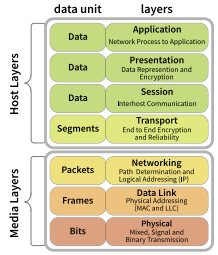
What Now?

- How to analyze data that is coming in at a rate far more than your compute server can handle?
- Does multi-core processor technology help here?

CVERIFY

More Intricate Hardware

- As speed requirements rise, more layers in a stack require hardware processing
- As hardware moves up the layers, verification becomes more intricate
- Until 2005, only the MAC layer was implemented in Hardware



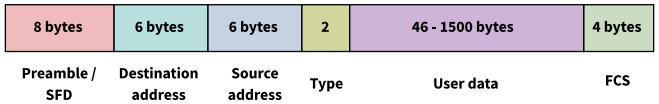
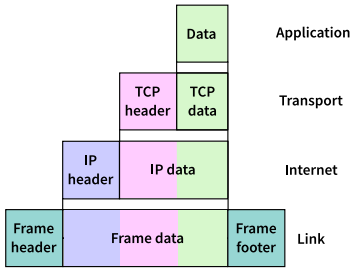


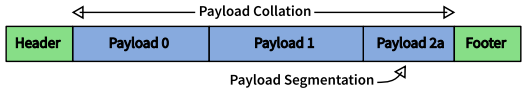
Figure: Ethernet Frame Structure

CVERIFY

Nested Packet Structures

- Layered Protocol Packet structures are Nested
  - Upper Layer packet frame is carried in the payload of the lower layer
- Hardware has to do deep packet analysis in order to decipher upper layer data
- Some protocols may require collation and/or segmentation of payload
- What impact would such hardware complexity have on Verification?

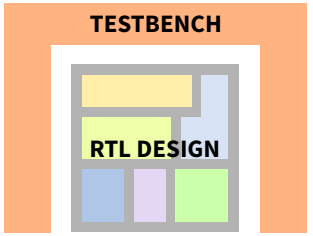




COVERIFY

Verification Tool Performance

- Intel Knights Landing processor has 72 cores and 288 threads
- AMD Threadripper has 32 cores and 64 threads
- To speed up testbenches we need to harness multicore



COVERIFY

Takeaways

Verification domain grows, wider (more transistors) and deeper (more complex algorithms) …

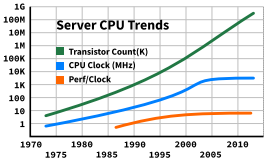
Wider	Deeper
Multicore Tools	Layered Testbenches
Emulation	Smarter Tools
More Engineers	Smarter Engineers
More Hardware	More Software

COVERIFY

Verification Performance

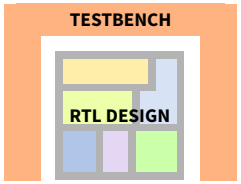
The Trend

- The Free Lunch is over
- The numbers of cores in server processors are projected to grow to hundreds in next 5 years



The Challenge

- Possible to partition RTL structurally because all variables are statically allocated
- No such automatic partioning possible for the dynamically allocated testbenches

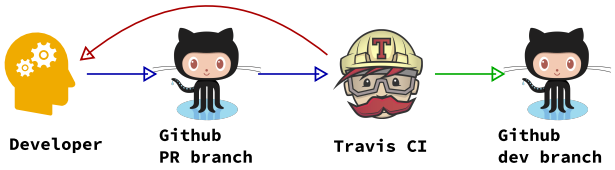


COVERIFY

FOSSI

Functional Verification as an RTL PR hand-off criteria

- Opensource RTL dev requires opensource EDA tools
  - Verification test-suit runs on CI cloud for each PR
  - A developer should be able to run tests on his end



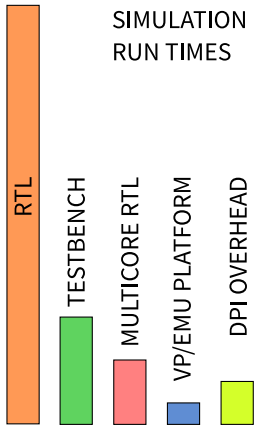
Testbenches for System Level Verification

The Trend

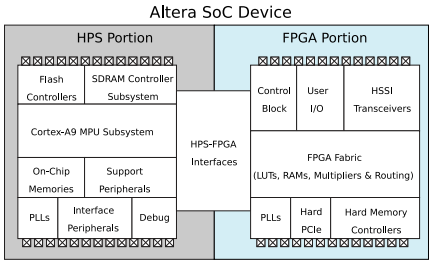
- More and more designs are being verified using Emulation Platforms
- Shorter product cycles are driving design teams to use Virtual Platforms

The Challenge

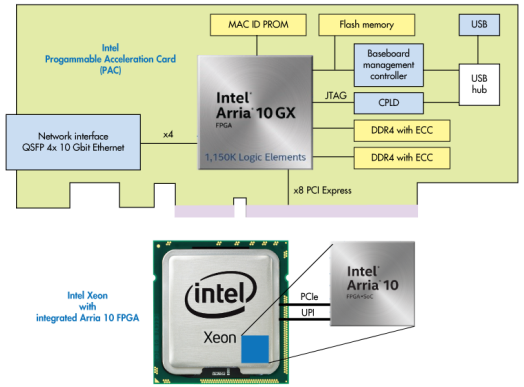
- Contemporary Verification Languages were drafted with RTL simulation in mind
- Simulator perfomance becomes a bottleneck when testbenching Emulation/ESL Platforms
- DPI interface overhead adds to testbench performance woes



Hardware Accelerators



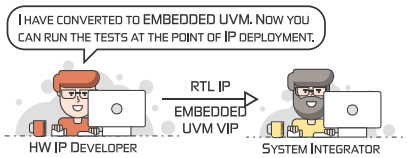
Hardware Accelerators – server end



Hardware Accelerators – Verification Perspective

COVERIFY

Hardware Accelerators Perspective – Embdded UVM



- Embedded UVM tests the point of deployment
- Allows System Level calls from withing a Testcase
  - For example DUT config part of testcase can be replaced by loading of device driver!!
- Failing tests can be taken back to RTL for debug
- Software testing is often Opensource

COVERIFY

SystemVerilog has limitations at System Level

Inspite of the SYSTEM moniker ~~System~~Verilog does not have any Systems Programming Feature!!

- You cannot write device drivers and/or operating system in ~~System~~Verilog
- No Library – Data-Structures? Algorithms??

For high-level applications programming to be effective and convenient, we need libraries. Using just the bare language features makes almost all programming quite painful. That's true for every general-purpose language. Conversely, given suitable libraries just about any programming task can be pleasant…

COVERIFY

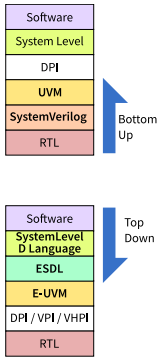
So, What is Embedded UVM?

- Complete port of IEEE UVM-1.0 (1800.2-2017) release including the reg-package
- Natively Compiled into:
  - Linux (and other OS) executables that can run on any processor
  - Shared libraries that can be loaded into any application (including Verilog/VHDL Simulations)
- Multicore Powered:
  - Multiple posix threads running concurrently
  - Allows multiple `uvm_root` instances and even mulitple simulators
- Systems Programming Enabled:
  - ABI compatibility with C/C++
  - LLVM based compiler - compiles to LLVM IR
- **Embedded UVM** is NOT **SystemVerilog**

COVERIFY

Top Down Verification

- HVLs are built on top of RTL – Software interaction is Week
- Embedded UVM is built on top of D Programming Language, A Systems Programming Laguage



COVERIFY

A UVM port that runs on Embedded Systems …

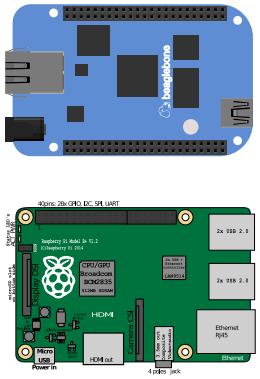
What will you verify?

COVERIFY

What will you verify?

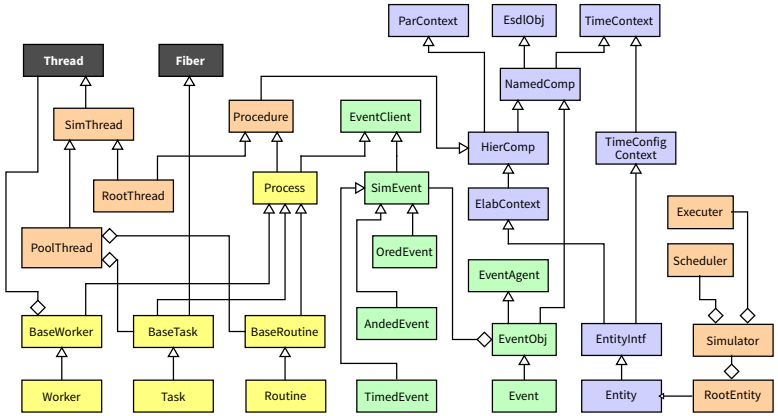
Run your UVM testbench on a RaspverryPi or a Beaglebone and Generate

- Constrained Random Ethernet packets that go out on a real ethernet port
- Constrained Random USB packets that emerge from USB connectors of the board
- Test a Beaglebone Cape or a Raspberry Shield that you designed in your Garage
- Stimulate GPIO pins and connect these pins to an Arty FPGA board



COVERIFY

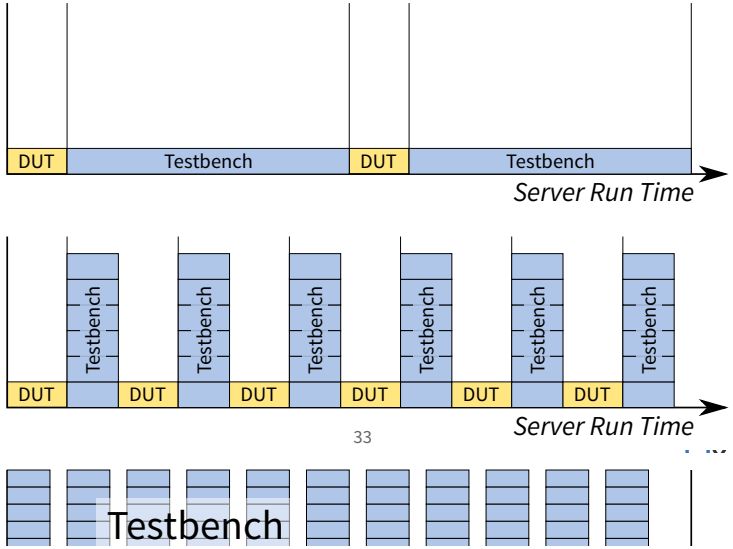
Embedded UVM Core Infrastructure



COVERIFY

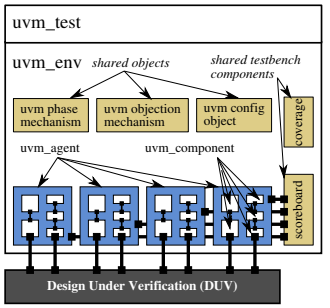
HVLs Are Essentially Single Threaded

- Both SystemVerilog and SystemC (as of now) run on a single OS thread
  - SV/SystemC use Cooperative Threading for Fork/Spawn
- Testbench can be made efficient by invoking concurrent threads
- And even more efficient by running the testbench in parallel



Embedded UVM Innovation – Multi Core UVM

- Most System Level Designs have multiple (TLM) Interfaces
- Each (TLM) Interface requires a VIP (or a `uvm_agent`)
  - Most VPIs have no interaction with other VIPs
  - This provides the right opportunity for parallelism
  - Embedded UVM implementation runs `uvm_agent` threads parallelly



Productivity Features of Embedded UVM

- Pointer-less programming at application level
  - Automatic Garbage Collection
  - Arrays are first class data objects
  - Pointers are available for low-level programming
  - `std.allocator` module allows alternate allocation ways
- Fast Compilation – ZERO Elaboration time
  - Effective for Agile Programming

```
class Foo: uvm_object {
  mixin uvm_object_utils;
  @rand!8 byte[] foo;
  @rand Logic!12 baz;
}
class Bar: Foo {
  mixin uvm_object_utils;
  @rand ubyte[8] bar;
  Constraint! q{
    foo.length > 2; // array
    baz[0..8] == 16;
  } cstFooLength;
  Constraint! q{
    foreach(i, f; bar) f <= i;
    foreach(i, f; foo) {
      if(i > 4) /*condition*/
        f + i < 32 && f > 16;
    }
  } cstFoo;
}
void main() {
```


Modelling Hardware and Software

- How do you code an Array of Associative Arrays in C++?
- Here is how you do it in D …

```
#include <vector>
#include <map>
#include <string>

void foo () {
  std::vector
    <std::map
     <std::string, int> > myVect;
  std::map<std::string,int> entry1;
  std::map<std::string,int> entry2;
  entry1["ABC"] = 1;
  entry1["DEF"] = 2;
  myVect.push_back(entry1);
  entry2["ABC"] = 5;
  entry2["RKD"] = 9;
  myVect.push_back(entry2);
}
void foo() {
  int[string][] myVect =
    [["ABC": 1, "DEF": 2],
     ["ABC": 5, "RKD": 9]];
}
```

Step Locked Co-Simulations

- Tight binding, Reflective Signals
- Fine grained time locking
  - Verilog Simulator yields for Embedded UVM simulator at every time step
  - Even Delta time steps
- Signal Level Binding
  - Each and every Testbench signal is bound with corresponding RTL signal
  - A change driven by RTL is visible in testbench
  - A change driven by testbench gets reflected in RTL
- Impedes Parallelism
  - Note that Verilog Simulator stops when Embedded UVM Simulator is running

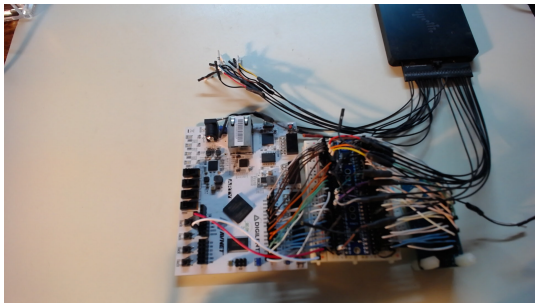
Transaction Level Co-Simulations

Transaction Level Co-Simulations

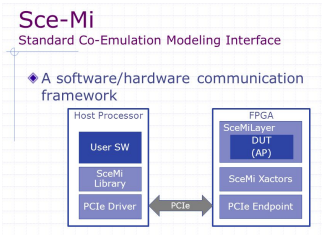
- Verilog Simulation asks for a transaction (sequence item) when it needs one
- When a transaction is complete, the Verilog Simulator pushes back the status/result
- At every data exchange event, Embedded UVM gets an update on siulation time
- Transaction Level exchange happens via asynchronous TLM Fifos
  - Leads to a much better opportunity for parallelism

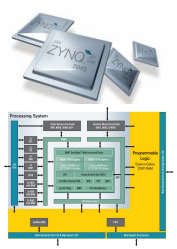
Embedded UVM Powered Emulation

- Embedded UVM is ABI compatible with C
- Integrates seamlessly with MRAA Library
  - MRAA library provides user friendly and efficient interface to embedded processor's GPIO pins



Embedded Emulation





Embedded Emulation

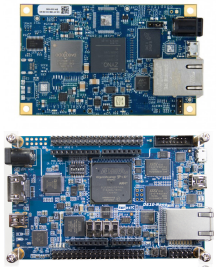
- There is a need to Provide a viable Verification/Emulation platform
  - Existing emulation platforms are overly costly
- To ease debug, it should be possible to drive Emulation and Verification by the same Stimulus
- Embedded UVM, being natively compiled, has a small memory footprint
- And therefor can be run on embedded processors with limited RAM
- Embedded UVM, being multicore enabled, scales well on multicore embedded processors

Parallella

- 650MHz ARM Cortex A9
- Xilinx 7020
- USD 250

DE10-Nano

- 800MHz ARM Cortex A9
- Altera/Intel Cyclone V
- USD 130



Design Under Test - SHA3

- An encryption standard, SSH uses SHA2
- Efficient hardware implementation one of the goals of standard
- An opensource IP core
  - Both RTL and SW are opensource implementations
  - RTL is small, only about 200 lines of code

DUT (SHA3 Core) – Functional Details

- The DUT implements Sponge Construct, as defined in the SHA3 standard
- Control and Status registers

Offset	Register
0x0000	NAME0
0x0004	NAME1
0x0008	VERSION
0x0020	CONTROL
0x0024	STATUS

- NAME0, NAME1, and VERSION are read-only registers

SHA3 Core – Control and Status

- Upper 30 bits of the CONTROL register are DONT CARE
  - Write 0x01 for signalling INIT (input frame init)
  - Write 0x10 for signalling NEXT
- Upper 30 bits of the STATUS register are DONT CARE as well
  - bit 1 (asserted) signifies that the output to be VALID
  - bit 0 always reads 1

SHA3 Core – Data Map

- DATA Address Space

Offset	BANK
0x0200	BLOCK
0x0300	STATE

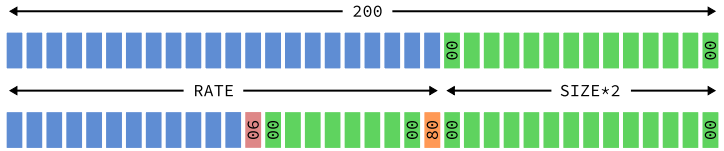
- Length of each bank is 200 bytes
- The first bank is RW, and the second RO

SHA3 Core – Output

- The SHA3 Sponge works in 4 modes, and can output
  - 224 bit hash (28 bytes)
  - 256 bit hash (32 bytes)
  - 384 bit hash (48 bytes)
  - 512 bit hash (64 bytes)

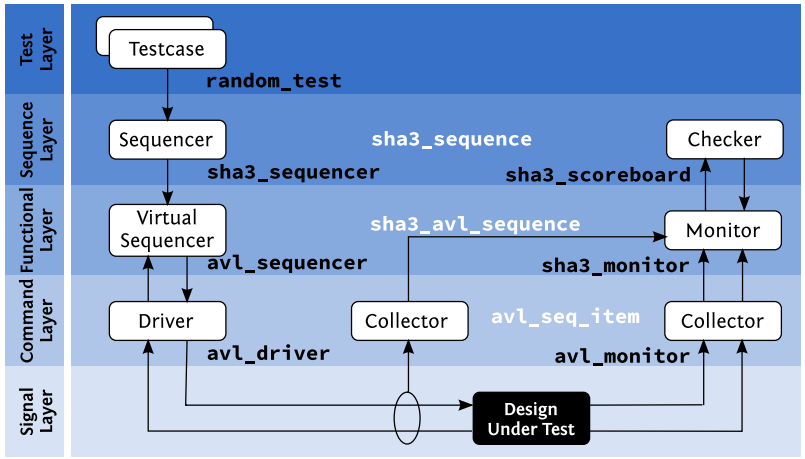
SHA3 Core – Input

- SHA3 input can be a string of bytes of arbitrary size
- Long SHA3 inputs need to be broken into multiple frames
- SHA3 input frames are a bit tricky, depending on the mode, the frame needs to have a ZERO padding of twice the size the output
- The last frame may need extra ZERO padding, since the input string may not fill the frame
  - The ZERO padding in the last frame always starts with 0x06 and ends with 0x80
  - 0x86 is a special case



COVERIFY

Testbench Architecture



COVERIFY

Thank You for your attention!

COVERIFY