UVM and UPF: an application of UPF Information Model

Amit Srivastava, Harsh Chilwal, Srivatsa Vasudevan

Agenda

- Introduction
- Need for UVM and UPF Integration
- Requirements of UVM Integration
- UPF Information Model
- UPF UVM Class Library
- Case Study: Bitcoin design
- Future Work
- Conclusion

Introduction

- UVM has been a popular verification methodology for long time
- Integrating UVM with Low Power Designs has been a challenge
  - Power Management is present outside HDL
  - Lack of interface with UPF
- IEEE 1801-2015 introduced UPF Information Model
  - SystemVerilog functions to access UPF Information
  - Insufficient but provided basic capabilities
- Class Library based on UPF Information Model that provides portable UPF and UVM Integration

UPF Information Model

- UPF Information Model enables access to UPF Information
  - Introduced in IEEE 1801-2015
- Provides a well defined, standard information model
- Simple APIs in to query the information
  - Tcl, SystemVerilog and VHDL
- Access to dynamic information provided in SystemVerilog and VHDL packages
- Read access provided for dynamic and static properties
- Write access provided for some dynamic properties

UPF Information Model APIs

- Functions defined in SystemVerilog UPF Package
- Accessing handle of UPF Objects
  - upf_get_handle_by_name
- Read access
  - upf_query_object_properties
- Continuous Read access
  - upf_create_object_mirror
- Write Access
  - set_power_state
  - supply_on
  - supply_off

UVM Integration

- UVM is based on Object Oriented concepts
- SystemVerilog classes
- Power management is separated from HDL
- UVM classes cant directly access power management information
- UPF Information Model provides complete access
- But its procedural in the form SystemVerilog functions
  - Similar to VPI

Requirements of UVM Integration

- Access Object Oriented Handles
  - Class objects
- Callbacks
  - Define callbacks on the states of UPF Objects
  - Trigger events whenever state changes
- Modifying State
  - Manipulate state of UPF Objects
  - Enables high level testbenches independent of pin level details

UPF UVM Class Library

- A Class library based on UPF Information Model
- UPF Classes
  - SV classes corresponding to UPF Information Model Objects
  - Wrapper over UPF handles
  - Contain important events corresponding to dynamic properties
  - Events enable UVM classes to attach callbacks

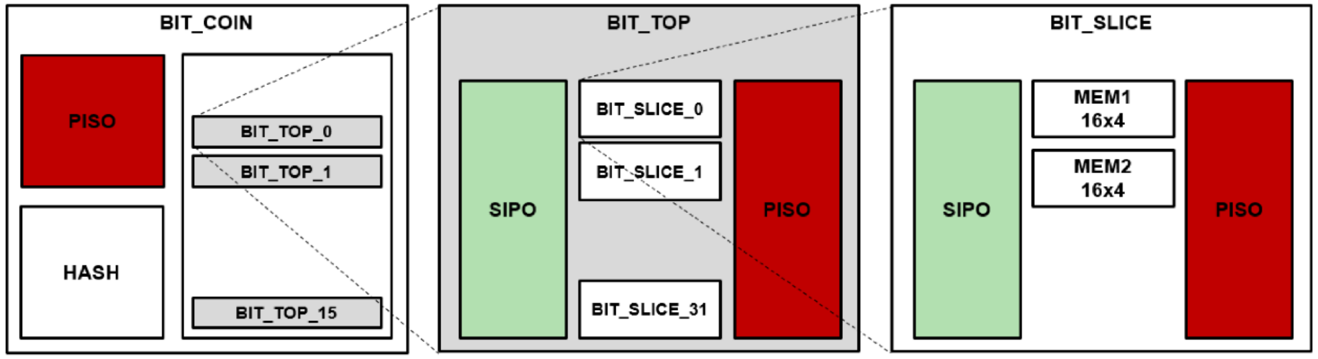
UPF Mirror Utilities

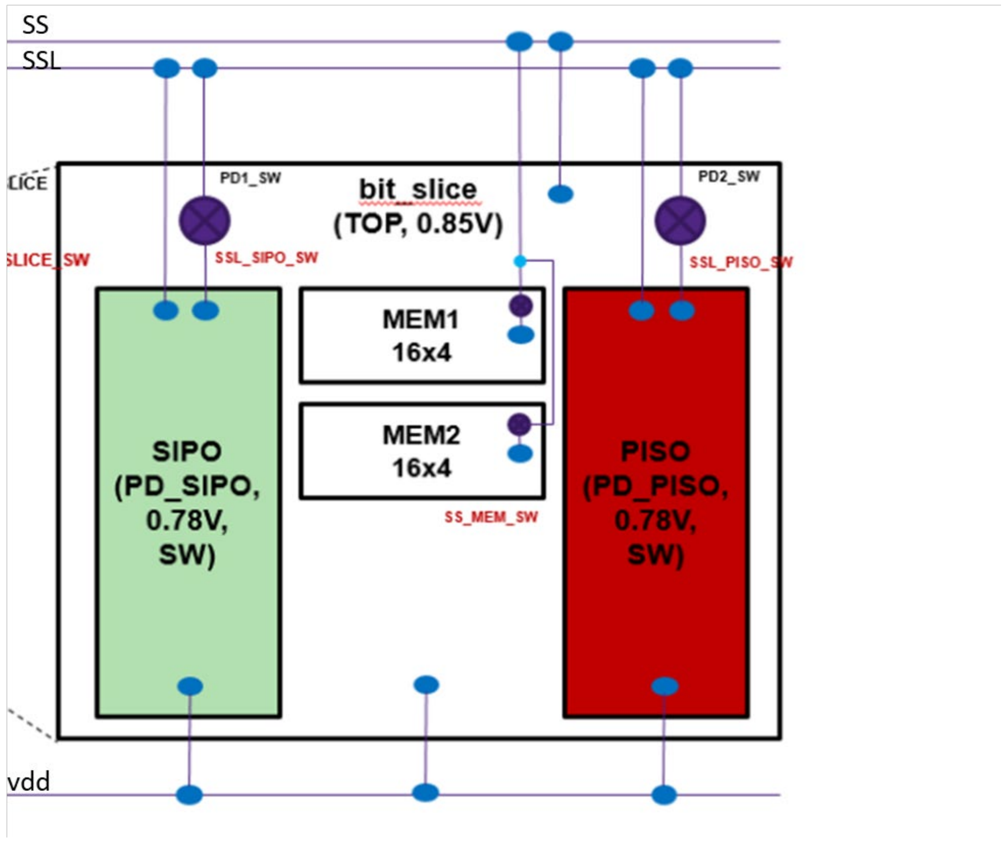
- Utility functions to establish mirror relationships between Class objects and UPF IM objects
- Enables class objects to be in sync with UPF objects
- Portability across tools
- Can be replaced by native implementation to boost performance

Low Power Coverage

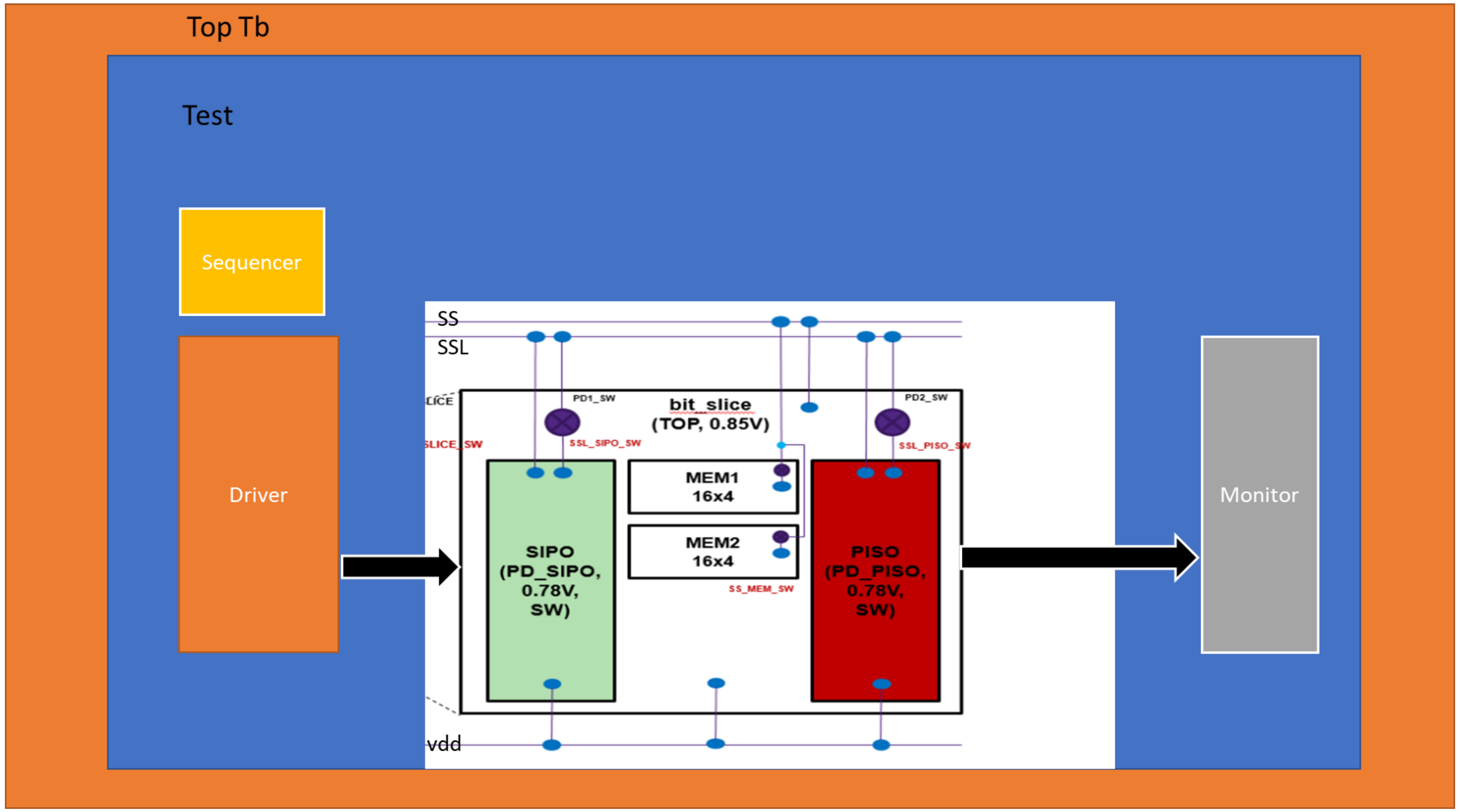
- Enables capturing coverage metrics for UPF objects
- SV covergroups to capture states of Power Domain
- Class can be extended to include custom coverage mentrics
- Sampling is done inside forever blocks

Case Study: Bitcoin design



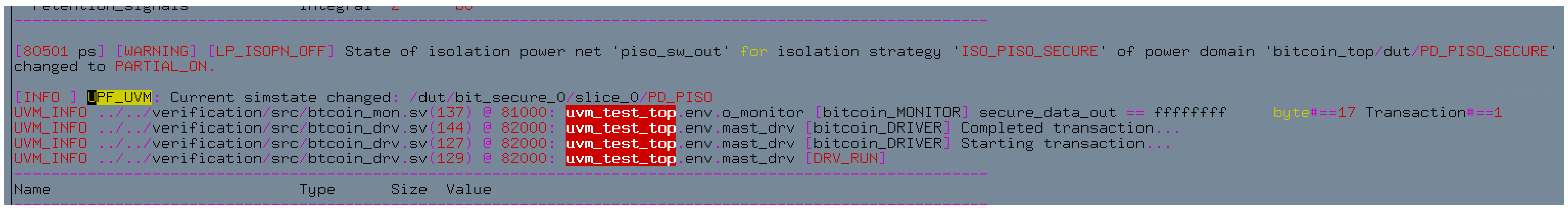


UVM and UPF Integration



Sample code and Results

- Monitor gets handle to event from config_db.
- The DUT power switches are wired to the module DUT ports and are connected to the driver and monitor interfaces.
- The testbench environment is designed to produce stimulus via sequences and the driver which then drives them into the DUT. The driver also drives the switches for low power operation. Specific sequences have been created to power up/power down portions of the bit_slice module.



```
--------------------------------------------------------------------------------

[80501 ps] [WARNING] [LP_ISOPN_OFF] State of isolation power net 'piso_sw_out' for isolation strategy 'ISO_PISO_SECURE' of power domain 'bitcoin_top/dut/PD_PISO_SECURE'
changed to PARTIAL_ON.

[INFO ] UPF_UVM: Current simstate changed: /dut/bit_secure_0/slice_0/PD_PISO
UVM_INFO ../../verification/src/btcoin_mon.sv(137) @ 81000: uvm_test_top.env.o_monitor [bitcoin_MONITOR] secure_data_out == ffffffff     byte#==17 Transaction#==1
UVM_INFO ../../verification/src/btcoin_drv.sv(144) @ 82000: uvm_test_top.env.mast_drv [bitcoin_DRIVER] Completed transaction...
UVM_INFO ../../verification/src/btcoin_drv.sv(127) @ 82000: uvm_test_top.env.mast_drv [bitcoin_DRIVER] Starting transaction...
UVM_INFO ../../verification/src/btcoin_drv.sv(129) @ 82000: uvm_test_top.env.mast_drv [DRV_RUN]
--------------------------------------------------------------------------------
Name                           Type       Size  Value
--------------------------------------------------------------------------------
```

Future Work

- Improve performance by adding native implementation of mirroring package
- Add more capability
  - Write methods
- Power domain manager
  - Communicate effectively with components in UVM

Conclusion

- Interoperability between UVM and UPF has been missing for more than a decade
- Introduction of UPF Information Model opened the doors for standard interface
- The class library enables a standard and portable interface of UPF with UVM
- The application allows easy to use and extensible SystemVerilog classes which bridge the gap between UPF and UVM

THANK YOU