Reuse of SV-UVM based IP Verification Environment at SoC – Challenges Involved

Vinod Paparaju

22/08/2012

Outline

- Introduction and Problem Description
- Proposed Methodology
- Analysis
- Conclusion

Introduction

- SoC design verification involves checking the integration as well as simulating some use case scenarios
- Latter involves a Driver/BFM inside the test bench
- Developing such a TB model per module is no mean task
- One of the prevalent noble approach is to reuse the IP verification environment at the SoC level

Problem Description

- Driven by the capabilities of System Verilog, UVM is fast becoming a popular environment for verifying standalone modules (IP)
- Currently no proper guidelines are defined to reuse the UVM based IP environment at SoC level
- This work is aimed at addressing this problem

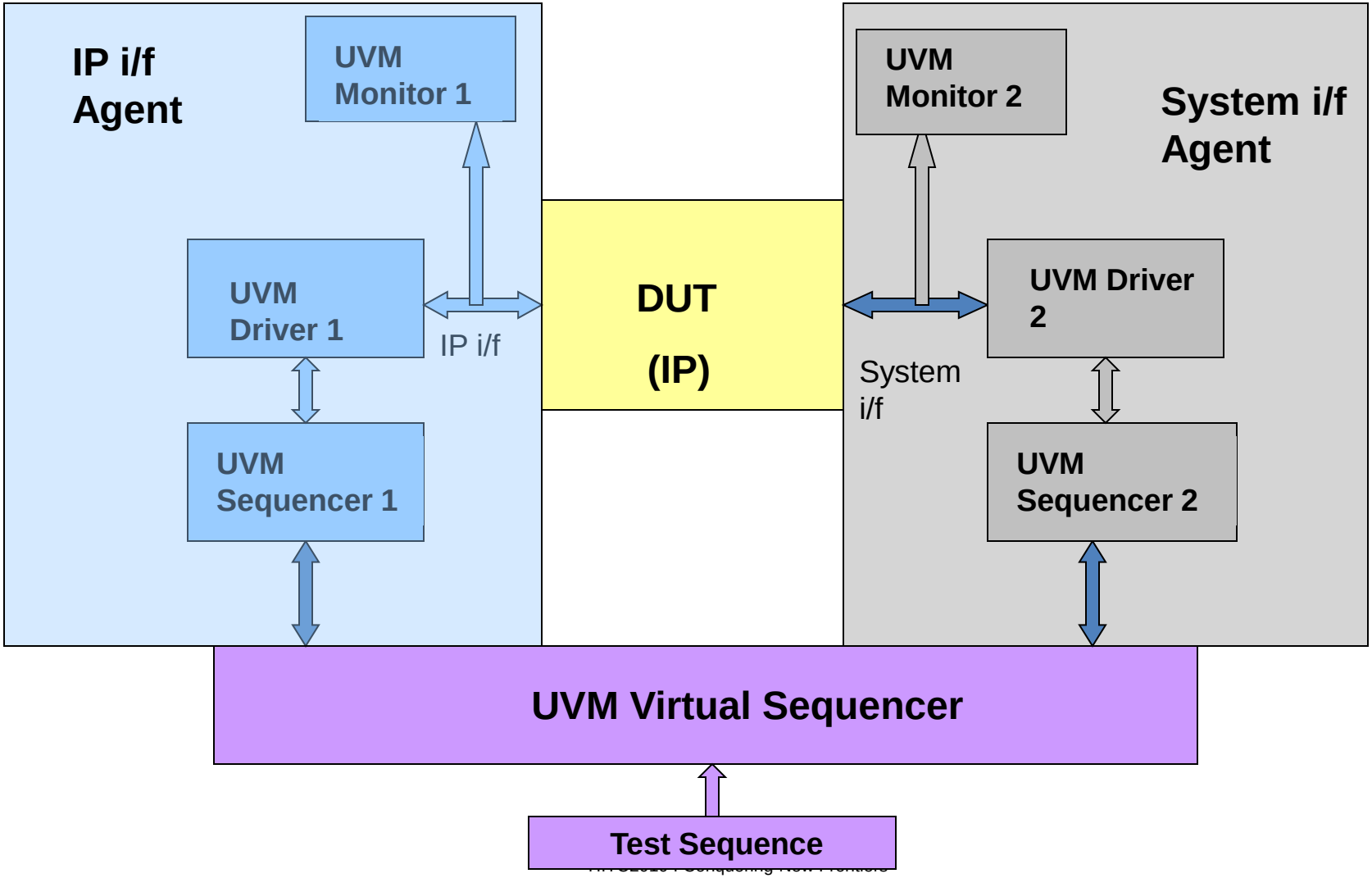
Proposed Work

- Challenges/Solutions in designing a SoC reusable SV-UVM based IP verification environment (env)
  - Connecting the IP env at SoC
  - Synchronizing the IP env operation with that of CPU at SoC
  - Simulating test scenarios at SoC using the IP env

System Verilog based UVM (SV-UVM)

- System Verilog
  - SV is a unified hardware design, specification and verification language
  - All the HDL simulators support it thus avoiding any additional overhead
- Universal Verification Methodology
  - A open-source verification methodology based on a library of classes
  - Uses an architecture for building the verification environment that enhances reusability
  - Uses TLM (transaction) interfaces for connecting verification environment blocks
  - Provides added features using which objects in the environment can be modified during the run time

IP Verification Env with SV-UVM



Operation of SV-UVM Env

- Test sequence will contain a mix of IP i/f sequences and System i/f sequences
- Virtual sequencer will channel the sequences to their respective sequencers
- Sequencer will execute the sequence and pass the stimulus to the driver through TLM port
- Driver will drive the stimulus in the respective protocol
- Monitor will monitor the incoming data on the i/f

Making the Env Reusable at SoC – Issues and Proposed Solution

- Not all the components in the IP env are required for SoC reuse
  - Pass on a parameter that controls the construction of the agents
    - Each component in the SV-UVM env is a dynamic object
    - Use a conditional assignment in the env
  - Keeping the condition always true will ensure no effect on IP verification
- Simulating a complex scenario at SoC requires a communication between CPU and the Sequencer
  - Define a set of triggers in the IP i/f
  - Keep them unconnected for IP verification
  - Define a handle to the i/f (a virtual interface) as a virtual task in the virtual sequencer
    - Unless called through a test sequence this virtual i/f will never be assigned

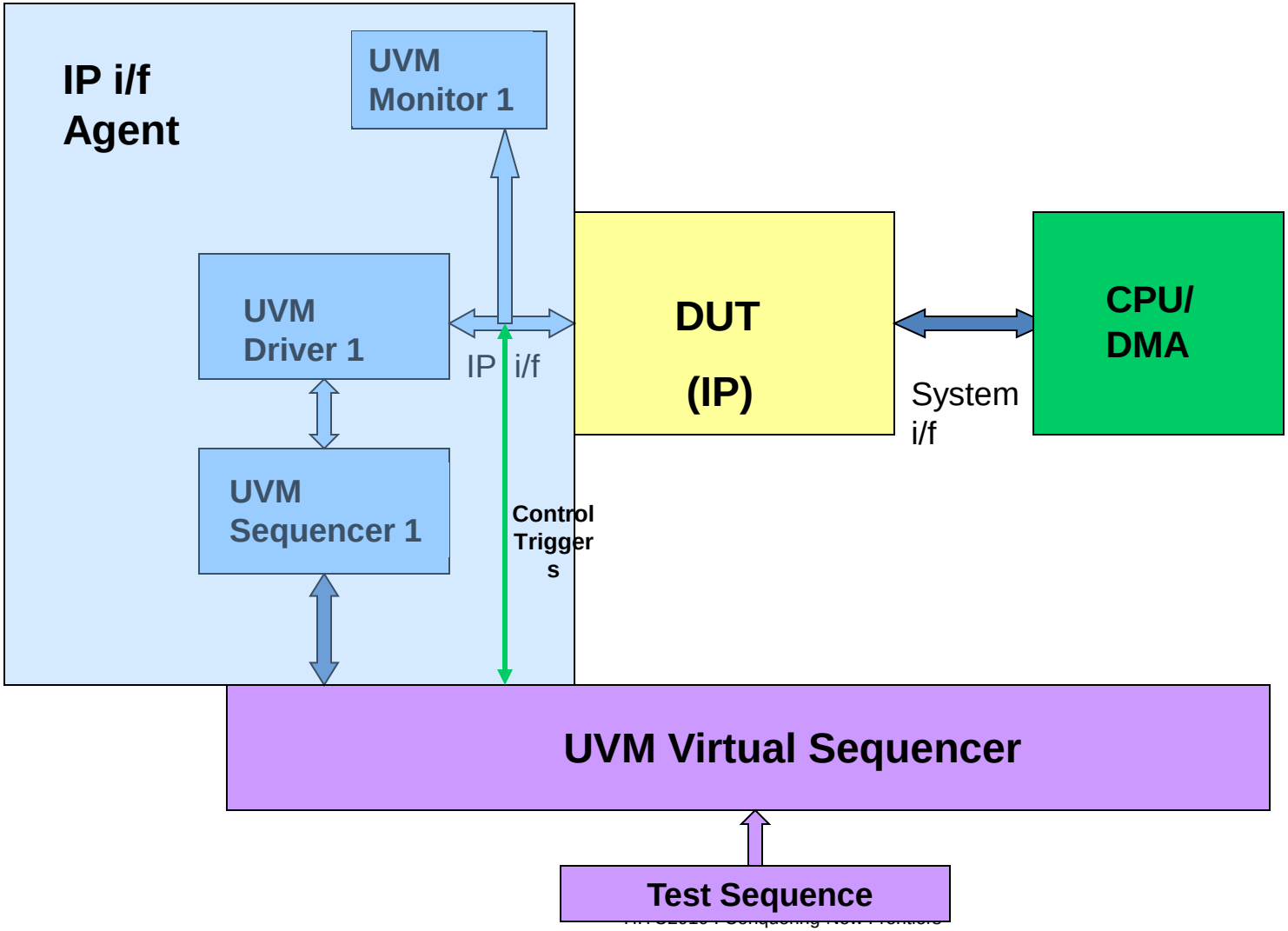
Connecting the Env at SoC – Issues and Proposed Solutions

- The SoC pins will be different
  - Create a verilog wrapper around the SV-UVM env
  - Use this wrapper to map the IP i/f to the SoC pins
- A CPU/DMA will replace the System i/f agent
  - Pass a parameter during the compilation stage that will gate the build of System i/f agent
    - Disables the build of all components within it
  - Keep the System i/f pins unconnected
    - This i/f will not be even assigned to the env

Simulating a Scenario at SoC – Issues and Proposed Solutions

- There should be proper communication between the CPU and the IP sequencer
  - Assign the virtual interface in the Sequencer to the IP i/f through a task call from the test sequence
  - Connect the control triggers in the IP i/f to General Purpose IOs (GPIO) of the SoC interface
  - Let the CPU and the Sequencer control one trigger each and monitor the other
  - Synchronize the operation of CPU and the IP Sequencer through the trigger control
  - Include the trigger control only in the test sequence and retain all the IP i/f sequences
    - The test sequence will be the only difference from IP to SoC

Schematic of the Proposal



Example test sequence at SoC

- Test Sequence

```
uvm_report_info(get_type_name(), "Starting soc_endpoint_test_seq
 Sequence...", UVM_HIGH);
 p_sequencer.connect_if(); // Connects the interface to virtual
 sequencer
uvm_report_info(get_type_name(), "executing soc_test_seq..",
 UVM_HIGH);
wait(p_sequencer.IP _sequencer_if.soc_trigger === 1'b1); //Waits until
 CPU sends a trigger
< IP  Host sequences >
```

- Virtual Sequencer

```
virtual task connect_if ();
IP_host_sequencer_if  =
 IP_host_sequencer_if_wrapper.virtual_IP_host_if;
endtask :
```

Analysis

- The Proposal is implemented on an IP that has a AHB (System) i/f and a Slave (IP) i/f
- Able to use the same SV-UVM environment for both IP and SoC verification
- It is only the test sequences that differ
- Able to run different scenarios at SoC using this environment
- Effort spent on making the environment reusable is minimal (<2% of the total env development effort)

Conclusion

- By following the proposed methodology, any IP env that uses SV-UVM can be ported to the SoC with minimal effort

- Limitations of the proposal
  - The UVM libraries can not be directly ported to FPGA or Palladium boards

- Scope for further work
  - Define a methodology for reuse of the env for real time simulations