Leveraging Virtual Platform ESL and TLM SystemVerification in RTL using UVM

Improving the Quality of RTL with Existing System Level Verification

Ashok Mehta
Senior Manager, Design Methodology Department
TSMC
San Jose, U.S.A.

Albert Chiang, Wei-Hua Han
Verification Group
Synopsys, Inc.
Mountain View, U.S.A.

Abstract— **The RTL design verification process is a fairly well understood one, but it is routinely tedious and laborious. One way to improve the efficiency of RTL design verification is to reuse as many pre-existing verification components as possible. One possible source of reuse is from the verification environment that was already deployed on a virtual platform. This paper will focus on reducing the effort in RTL design verification by reusing the verification environment created from the virtual platform.**

Keywords- ESL, UVM, TLM, RTL Verification, Virtual Platform, SystemVerilog, SystemC

I. Introduction

As software became a key product differentiator in system-on-chip (SOC) designs, virtual platforming has become a necessity to ensure that the complex interaction between both software and hardware can be validated and verified. Virtual platforming enables modeling a system-on-chip using a high abstraction language such as SystemC, so that building and simulation of the model can be done relatively quickly compared to using RTL code for hardware. Building a virtual platform involves piecing together SystemC TLM2 components that mimic their real world counterparts, followed by running simulations to validate if architectural assumptions were correct.

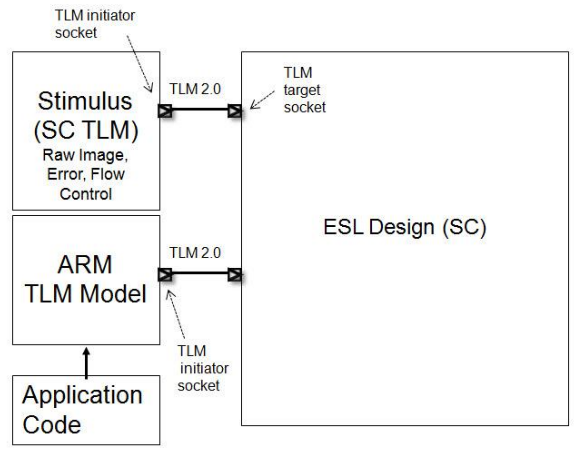
II. Virtual Platform

At the onset of designing a System-On-Chip (SOC), which is comprised of both hardware and software, an early prototype of the SOC is developed by the system or architecture team. This early prototype, or virtual platform, is modeled using high level language such as C, C++, SystemC. The benefit of using these high abstraction languages for moedeling is that development can be faster because details of the implementation can be bypassed for now. SystemC TLM 2.0 allows blocks within the virtual to communicate very quickly and in a standard fashion. 3rd party SystemC IP providers sell pre-made SystemC TLM 2.0 models to allow virtual platformer to focus on building SOC. Algorithmic explorations can be performed quickly to test out the latest de-compression algorithm. Architectural trades-offs between FIFO depths or lowering operating to reduce power consumption can be examined.

As the virtual platform, a high abstraction model of the hardware is finalized; the software team can start to integrate an OS, device drivers, firmware, and application code, months before the design is available as hardware. The verification team will also start to verify the virtual platform IPs, connectivity, and basic block to block system level tests. Let's look more closely at how the verification environment interacts with the virtual platform.

III. Virutal Platform Verification

Verification is an important step in the virtual platform environment. It ensures that the individual blocks within the virtual prototype work standalone. Once the virtual prototype is assembled using the pre-verified blocks, verification can focus on connectivity. Once connectivity is verified, verification can focus on real world system interations. Much effort is expended to understand the system architecture, the block functionality, and the interaction between the blocks at the system level. These virtual platform tests can be native application C/C++ code, with a processor model fetching and performing reads or writes to the system. The test can also be SystemC TLM2 transactions.



The software application based verification usually exercises the SOC in the normal manner. On the other hand, the hand crafted stimulus using SytemC TLM 2.0 transactions can provide interesting traffic not possible with C/C++ stimulus. For example, if the verification engineer want to recreate a scenario where the SOC is in the midst of a DMA transaction, and at the point the transfer buffer is about full, a critical interrupt needs to be fired to verify the latency of the response. This level of transaction granularity and control is possible with precise SystemC TLM 2.0 transaction.

Once these interesting tests employing SystemC 2.0 TLM are written for the virtual platform, would it be nice if these tests can be reused in the RTL, leaving time to write more complex RTL tests that might not have been written. But how can one reuse tests, originally targeted for virtual platform, on a RTL design?

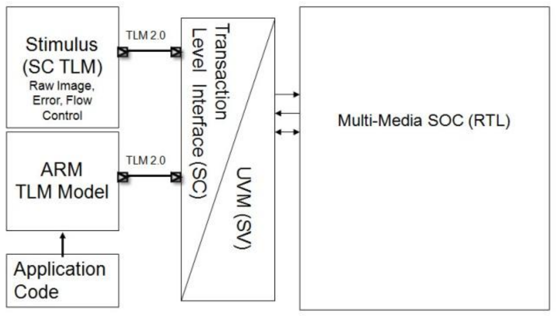
IV. Virtual Platform Verification Reuse Via UVM & TLI

In order to reuse the verification environment originally deployed on the virtual platform in the RTL environment, a mechanism is needed to translate the SystemC TLM 2.0 stimulus into pin level signals of an RTL design. This translation is achieved via the Transaction Level Interface (TLI) and Universal Verification Methodology (UVM).

Transaction Level Interface (TLI) is a mechanism in Synopsys VCS that allows SystemC and SystemVerilog to pass data to each other. In the ESL verification environment, the stimuli are TLM 2.0 generic payload transaction object, so TLI needs to be able to handle passing transaction objectrs. In addition to passing TLM 2.0 generic payload, the TLI mechanism also needs to support the various ways objets are cpassed, incliding TLM 2.0 concepts of blocking and non-blocking.

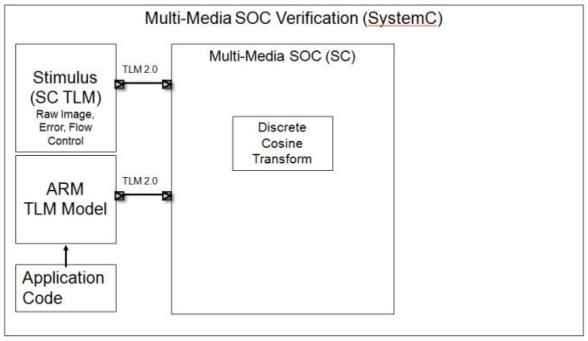
UVM was recently approved by Accellera, an electronics industry organization body comprised of industry experts, to create a common and open verification methodology. UVM will allow engineers to build scalable, extensible, and reusable verification environment that will improve productivity of the design verification community. Because UVM is written in SystemVerilog, UVM employs key SystemVerilog language features ideal for verification, including constrained random sequence generation, functional coverage, temporal assertions, and data structures such as class and smart queues. Of key interest is the reuse aspect of UVM to apply the same verification environment to both a SystemC and RTL version of the same design block, allow for all the verification effort on the SystemC verification to be reused on the RTL, thereby saving time and effort.

TLI, combined with UVM, allows for the ESL verification environment to be reused for RTL verification. Either the application code, or the SC TLM stimulus, can now be used on the RTL design The TLI and the UVM communicate via TLM sockets as well. In addition to allowing SystemC TLM 2.0 object to be passed to SystemVerilog through the standard TLM 2.0 socket methods such as blocking and non-blocking calls, TLI needs to also allow the SystemC TLM 2.0 initiator socket to be mapped to a SystemVerilog TLM 2.0 target socket. The TLI fully supports this and it will be demonstrated in a later example.



V. Multi-Media SOC Example

Image processing is a common component in most of today's consumer electronics. It is the hardware (and software) component that decompresses pictures (such as JPEG) to be displayed on your tablet or smart phone. Of interest in this example is the Discrete Cosine Transform (DCT) block. The multi-media SOC was prototyped virtually using SystemC, with other SystemC TLM 2.0 models. The SOC sits on an ARM AMBA bus as a slave with mastering capabilities to output the processed image to memory or display. In the system below, there are two masters that drive the SOC : an ARM model, and a stimulus block. The ARM model communicates with the SOC through SystemC TLM 2.0 interface. This interface allows for high abstraction, fast speed communication between the ARM model and the SOC.


Multi-Media SOC Verification (SystemC)
Stimulus (SC TLM)
Raw Image, Error, Flow Control
TLM 2.0
Multi-Media SOC (SC)
Discrete Cosine Transform
ARM TLM Model
TLM 2.0
Application Code

The stimulus block, written in SystemC, interfaces with the SOC via TLM 2.0 as well. The stimulus block can both data and control centric stimulus. For data, the stimulus block can read from an external file (such as a .jpg of Mona Lisa), to creating raw "patterned" images to create complex raw images, to purposely introducing noise into an image. Control stimulus that can be varied by the stimulus block includes overflow and underflow conditions.

In SystemC, the stimulus generator will create instances of "tlm_generic_payload" and populate it with image data such as one from a picture or noise. In this example, the SystemC class "producer" will create an instance of a SystemC TLM generic payload "trans" and populate it with the desired transaction; in this case, the populated command will be to perform an AXI write transaction. Once the "trans" is populated, the SystemC TLM generic payload will be passed over to SystemVerilog-UVM via the "init_socket->nb_transport_fw" call. How is SystemC TLM socket "init_socket" connected to a SV port "producer_sc_inst_export" ? They are connected via TLI SystemC system method called "tli_tlm_bind_initiator" to bind the SystemC socket "init_socket" to the SystemVerilog TLM socket "producer_sc_inst_export".

1. Passing TLM 2.0 generic payload from SC to SV

```
class ScVerif : public sc_module,
public
tlm::tlm_bw_transport_if<my_payload_typ
es>
{
   void main {
      trans = new generic_payload;
      trans->addr = dma_config_reg;
      initiator_socket-
>b_transport(*axiTrans, delay);
     if (trans->response != 1) {
fail(); }
   } // main
};
```

2. SV UVM socket receiving TLM 2.0 generic payload

```
class sv_uvm extends uvm_component;
   task b_transport(payload t,
uvm_tlm_time delay);
      // decode SC TLM payload
      // drive RTL
   endtask
endclass
```

Once the sockets are established on the SystemC verification environment as the TLM 2.0 initiator socket and UVM SystemVerilog as the TLM 2.0 target socket, the TLI need to be able to bind the two connected (map) these two sockets. These is done from a top level file from both SystemC and SystemVerilog.

1. Example of binding a SC TLM2 initiator socket to a SV-UVM target socket

```
class ScVerif : public_sc_module {
   public:
   SC_CTOR(ScVerif) … {

uvm_tlm2_bind_sc_initiator(i.initiator_
socket.
      UVM_TLM_B, "sc_2_sv_socket",
true);
   }
};
```

2. Example of binding a SC TLM2 initiator socket to a SV-UVM target socket

```
class SocEnv extends uvm_env;

  function void connect_phase(uvm_phase
phase);

uvm_tlm2_sv_bind#(payload)::connect(tar
get1.socket,
         UVM_TLM_B_INITIATOR,
         "sc_2_sv_socket")
       endfunction

endclass // SocEnv
```

Once these are bound, the SystemC verification environment is ready to send TLM 2.0 transactions. In our SOC, the main interface bus is an ARM AXI3 bus, so the transactions will be address, data, and command.

1. Stimulus from SystemC using TLM 2.0:

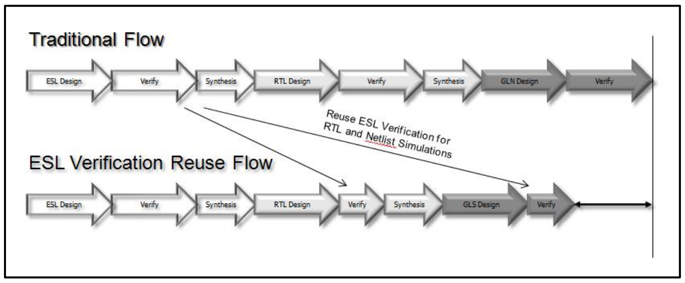
```
class producer: public
sc_core::sc_module, …{
    tlm_tlm_initiator_socket<>
init_socket;
  tlm::tlm_generic_payload* trans;
  producer::initial_thread() {
     trans->set_command(WRITE); //
create stimulu
     trans->set_address(addr);
     ret=init_socket-
>nb_transport_fw(*trans,phase,delay);
     }
};
```

2. Component in SystemVerilog-UVM

```
class        sv_uvm_comp        extends
uvm_component;
   task      b_transport(payload      t,
uvm_tlme_time delay);
      //  decode  SC  TLM  payload  drive
RTL
      @(posedge intf.mclk);
      intf.maddr = t.addr;
   endtask
endclass
```

VI. Results

TLI coverts the SystemC TLM 2.0 transactions into SystemVerilog TLM 2.0 tranactions, and UVM decodes the transactions into RTL signal. Using the the TLI and UVM mechanisms, we were able to reuse original C/C++/SystemC image processing verification environment on the RTL. These tests produced ARM AXI TLM 2.0 transactions, and through TLI and UVM, the transactions were converted to RTL pin signals. Additionally, application code written for the ARM processor were also reusable on the RTL as well. The application code eventually culminated in a typical configure-write-read-compare sequences, which were translated from TLM 2.0 transactions to ultimately RTL pins signals as well.



The verification already built for the virtual platform is reused on RTL and gate level netlist simulation. The time saved on

VII. Conclusion

With TLI and UVM, SystemC ESL verification environment can be reused on RTL simulations. Reusing tests that were already passing in SystemC ESL on the RTL verification increases confidence of the RTL design. As a bonus, RTL verification centric tools such as coverage, planning, and debug can augment the ESL tests to gain higher confidence and further insights into the not only the completeness of RTL verification, but on the ESL verification as well.