RTL-Agent Switch: Implementation and Applications

Aman Arora
Nathan Wooster
Pavan Mula
Rob Porter

NVIDIA
11001 Lakeline Boulevard, Ste # 100,
Austin, TX 78717
USA

www.nvidia.com

ABSTRACT

This paper presents a simple and useful technique for a testbench to dynamically choose, at runtime, either the RTL of a block or a behavioral agent, without modifying the design-under-test's (DUT) RTL and connectivity. The technique is implemented through a component called the RTL-Agent Switch. The paper will present this simple reusable component, explain how it can be implemented, and provide details on its applications. Applications where this technique has been successfully employed in real life designs include injecting traffic into SOC's, replacing RTL components that are not ready for test, and easing traffic generation on internal interfaces. Challenges encountered while implementing this technique in a real-life design will be presented.

Table of Contents

Table of Figures

1. Motivation

There is frequently a need to create or respond to traffic on an internal interface of a DUT (design under test) with a behavioral agent. This need can arise from many factors, such as lack of functionality in the DUT at earlier design phases, or difficulty in controlling stimulus for coverage closure. The two most common solutions for this are to:

1. Stub out one side of the interface (i.e. remove the RTL), or
2. Modify the design to expose the interface to the outside of the chip

These approaches both have significant drawbacks. For one, they require a new testbench variant to be created which has to be supported and maintained. Often there is organizational inertia against creating new testbenches given the many costs associated – build time, disk space, support staffing, and general verification environment complexity. A second problem (especially with solution b) is that these approaches involve modifying the DUT, creating a potentially less realistic verification environment. Some bugs might be missed, and some false bugs might be found. A third problem is that there is no way to switch between the actual RTL and the behavioral agent at runtime.

This paper describes a third solution, which is to attach an agent to an internal interface without modifying the design. We call the component that facilitates this an RTL-Agent Switch.

2. RTL-Agent Switch

Introduction

The RTL-Agent Switch allows us to attach an agent to an internal interface without modifying the design. This is shown in Figure 1.

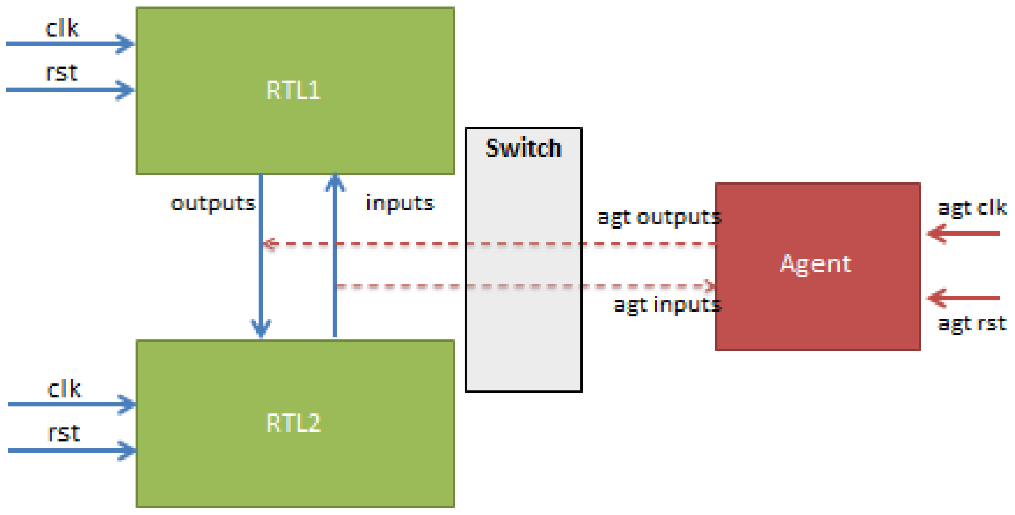

Figure 1 The switch in a testbench

The switch has several key properties:

a. The RTL on one side of the interface where the switch is attached should be powered down, clock gated, or in a state where it does not drive or respond to stimulus.
b. The agent connected to the switch can be either active or reactive.
c. The switch can change from allowing traffic from the agent to allowing traffic from the RTL at runtime. For this to happen there has to be a suitable transition point where the bus is quiet, all transactions are complete, and all credits are returned.

Implementation

An RTL-Agent Switch as described in the previous section can be implemented using the following facilities provided by System Verilog:

1. **The 'force' and 'release' statements** - 'force' and 'release' are continuous procedural statements. These statements have a similar effect to the 'assign'/'deassign' pair of statements, but a 'force' can be applied to nets as well as to registers. A 'force' statement overrides all assignments of a variable and all drivers of a net, until a 'release' statement is executed.
2. **The 'alias' statement** – An 'alias' statement declares multiple names for the same physical net, or bits within a net. Just like an 'assign' statement is a unidirectional assignment, bidirectional short circuit connections can be modelled using 'alias' statements.
3. **Command line inputs/plusargs** – Simulator behavior can be controlled by providing inputs on the command line. This is done using system function called $test$plusargs and $value$plusargs.

Let us assume that the DUT consists of two pieces of RTL, RTL1 and RTL2, which are connected by wires. As described above, we want to attach an agent to this internal interface between RTL1 and RTL2, without changing the connectivity between RTL1 and RTL2. Specifically, we want to replace RTL1 with an agent. As shown in Figure 2 below, this can be achieved by:

1. Forcing the output signals from the agent onto the inputs of RTL2 which connect to RTL1
2. Forcing the input signals of the agent from the outputs of RTL2 which connect to RTL1

This does not modify the connectivity of the DUT. And because these are procedural statements, they can be controlled by plusargs (and hence, the command line).

Now, when a plusarg is provided (ie. when the switch is enabled), the agent effectively replaces RTL2 in the DUT. When the switch is disabled, the normal functionality of the DUT is available.

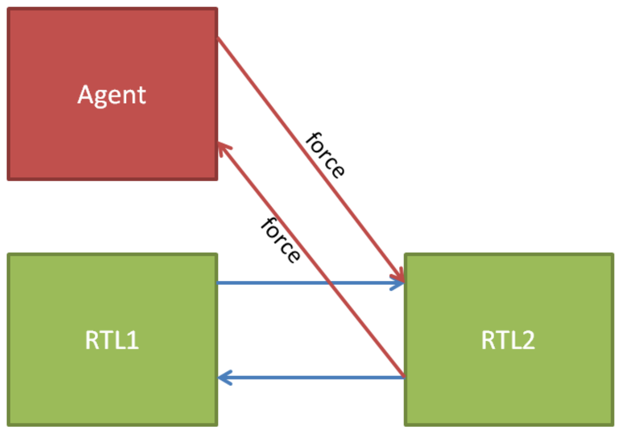

Figure 2 'force' being used to drive RTL from agent

To make this a reusable solution, the force statements and plusargs can be defined inside a module. We call this module the RTL-Agent switch. The switch module is slightly different than most other modules in a design: it cannot have any ports because declaring inputs and outputs means statically determining the direction of the signals and creating new drivers and sinks. Modifying the synthesized design's RTL and connectivity is antithetical to the goal of the switch. Because of this difference, the switch is not really a module in its traditional sense; it is just a component.

This creates a complication in connectivity. In the absence of a port list, simple port connectivity cannot be used. Also, 'assign' statements cannot be used because they are directional. The solution is to use 'alias' statements.

Here is the code for the switch.

```
module switch();

    parameter num_agt_inputs  = 10;
    parameter num_agt_outputs = 10;
    parameter num_agt_clocks  = 1;
    parameter num_agt_resets  = 1;
    parameter num_rtl_inputs  = 10;
    parameter num_rtl_outputs = 10;
    parameter num_rtl_clocks  = 1;
    parameter num_rtl_resets  = 1;

    wire [num_agt_inputs-1:0]  agt_inputs;
    wire [num_agt_outputs-1:0] agt_outputs;
    wire [num_rtl_inputs-1:0]  rtl_inputs;
    wire [num_rtl_outputs-1:0] rtl_outputs;
    wire [num_agt_clocks-1:0]  agt_clocks;
    wire [num_agt_resets-1:0]  agt_resets;
    wire [num_rtl_clocks-1:0]  rtl_clocks;
    wire [num_rtl_resets-1:0]  rtl_resets;


    initial begin

        if ($test$plusargs("agt_only_mode")) begin
            $display("SWITCH : Running in AGT ONLY MODE");
            force rtl_outputs = agt_outputs;
            force agt_inputs = rtl_inputs;
            force rtl_resets = 0;
            force rtl_clocks = 0;
        end

        else begin
            $display("SWITCH : Running in RTL ONLY MODE");
            force agt_resets = 0;
            force agt_clocks = 0;
        end
    end

endmodule
```

The switch can be configured in various modes depending on plusargs:

1. **Agent-only mode** – In this mode, the agent replaces one piece of the RTL. Here, the switch is said to be ON. As can be seen from the code above, in this mode, the clocks and resets of the RTL being swapped out are forced to 0. Not only does this save simulation time and memory, this can also prevent side-effects from unnecessary toggling of signals inside the RTL.
2. **RTL-only mode** – In this mode, the agent is inactive. This is the default setting of the DUT. Here the switch is said to be OFF. In this mode, the agent's clocks and resets are forced to 0.

Operation

The following code segment shows an example instantiation of the switch:

```
    // ---------------------------
    // Switch
    // ---------------------------
    switch #(.num_agt_inputs(33),
             .num_agt_outputs(70),
             .num_agt_clocks(1),
             .num_agt_resets(1),
             .num_rtl_inputs(33),
             .num_rtl_outputs(70),
             .num_rtl_clocks(1),
             .num_rtl_resets(1))  u_switch();

    alias u_switch.agt_inputs = {
                                agt_if.rdata_data,   //32
                                agt_if.rdata_valid, //1
                                };

    alias u_switch.agt_outputs= {
                                agt_if.wdata_data,   //32
                                agt_if.wdata_valid, //1
                                agt_if.req_data,     //36
                                agt_if.req_valid     //1
                                };

    alias u_switch.agt_clocks = {
                                agt_if.clk
                                };

    alias u_switch.agt_resets = {
                                agt_if.resetn
                                };

    alias u_switch.rtl_inputs = {
                                u_rtl2.rdata_data,
                                u_rtl2.rdata_valid,
                                };

    alias u_switch.rtl_outputs= {
                                u_rtl2.wdata_data,
                                u_rtl2.wdata_valid,
                                u_rtl2.req_data,
                                u_rtl2.req_valid
                                };

    alias u_switch.rtl_clocks = {
                                u_rtl1.clk_port
                                };

    alias u_switch.rtl_resets = {
                                u_rtl1.rstn_port
                                };
```

The following diagram explains the default mode of operation of the DUT, i.e. when no plusarg is passed. This is the “RTL-only mode”, in which the agent is inactive.

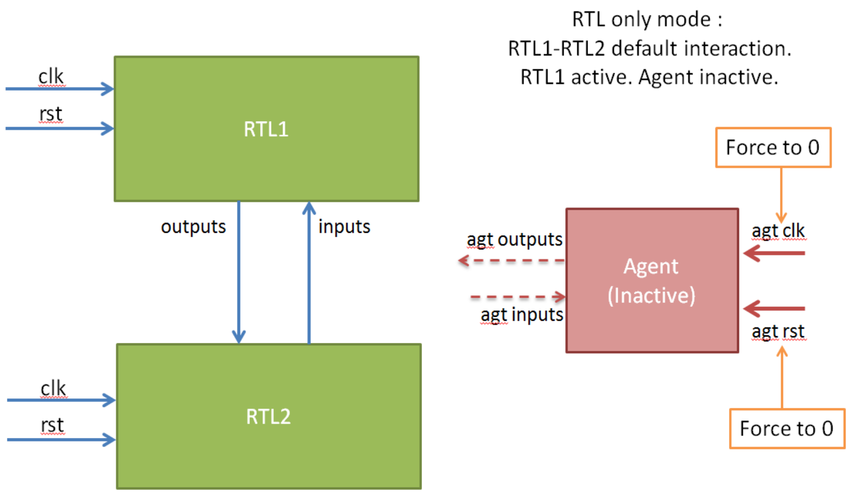

Figure 3 RTL-only mode

The following diagram explains what happens when the plusarg “agt_only_mode” is passed on the command line:

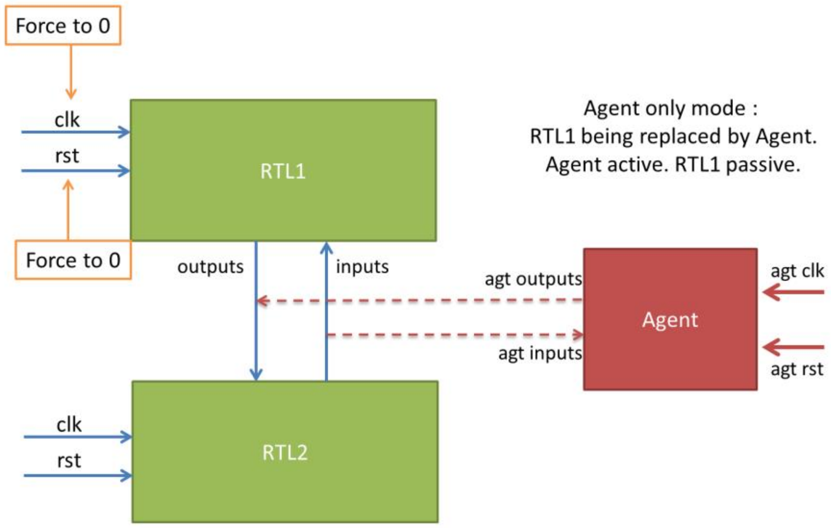

Figure 4 Agent-only mode

3. Applications

Injecting traffic into SOCs

With growing complexity of SOC designs, doing full-chip simulations for all SOC level testing has become prohibitive because of extremely high run times. One solution to this problem is to do most of the testing with tests running on the host machine and injecting transactions into the DUT using an agent. But when doing this, generally a MUX is introduced in the RTL which selects whether the transactions will come from the host machine or the CPU RTL. This is easy to implement, but is not ideal, because it means modifying the tape-out RTL. This newly added MUX has to be pulled out of the design when doing synthesis.

The following diagram shows such a modification:

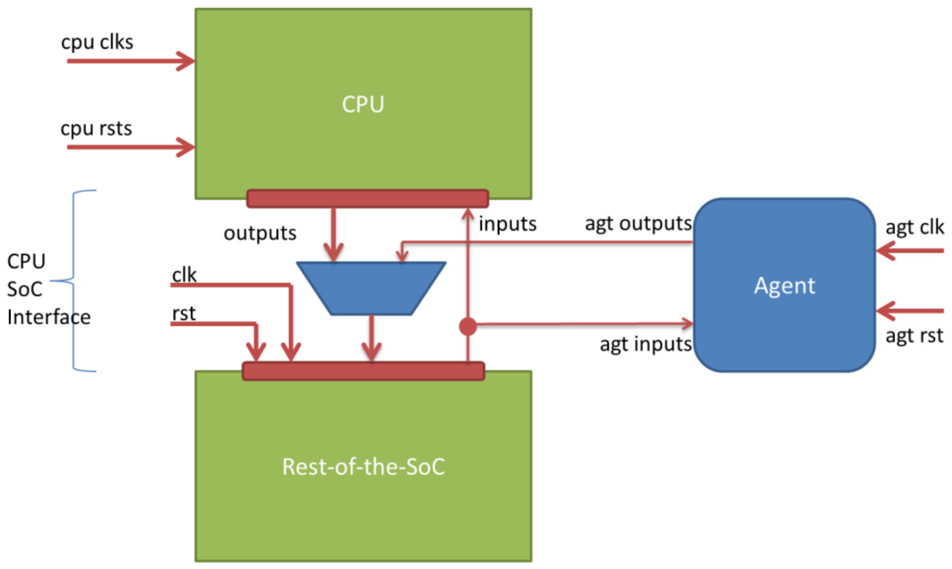

Figure 5 Using a mux to drive the SOC with an agent

However, if the RTL-Agent switch is used here, the addition of the MUX can be avoided, without changing any functionality. The agent can be used to drive the interface of the rest-of-the-soc, which the CPU normally drives. The agent can execute the test and push in transactions into the rest-of-the-soc, thereby improving simulation times.

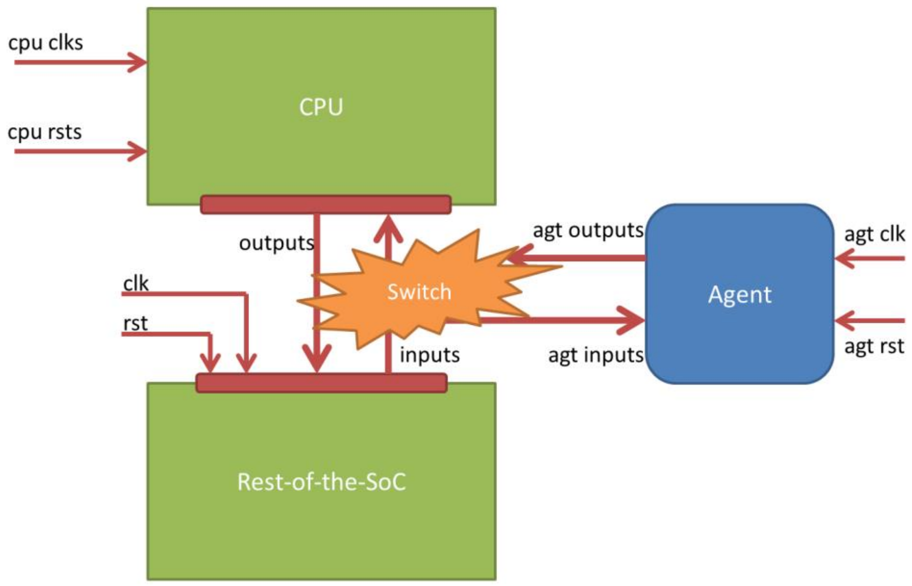

Figure 6 Using the switch to drive the SOC with an agent

RTL components not ready

With the shrinking cycle times of today's SOC designs, it is not uncommon to have several pieces of IP arrive later in the cycle than the DUT itself. And it is desirable to continue top-level test and testbench development without being gated by such deliverables. This cannot be done in the presence of a mere stub of the late IP. One solution is to be able to swap out the late-coming IP's stub with a port-accurate agent. But when doing this, care needs to be taken to not modify the existing connectivity. A switch-like component which can dynamically swap the IP's RTL with the Agent is desirable.

The following diagram shows an SOC in which IP2 is late in its schedule.

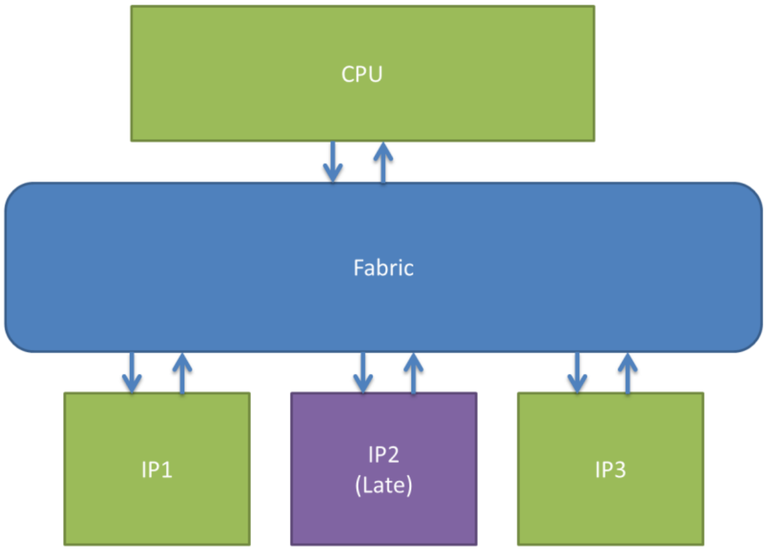

Figure 7 IP2 is late in schedule thereby hampering overall test development

Use of the RTL-Agent Switch to replace IP2 with an agent can ungate test development and hence, keep schedule timelines.

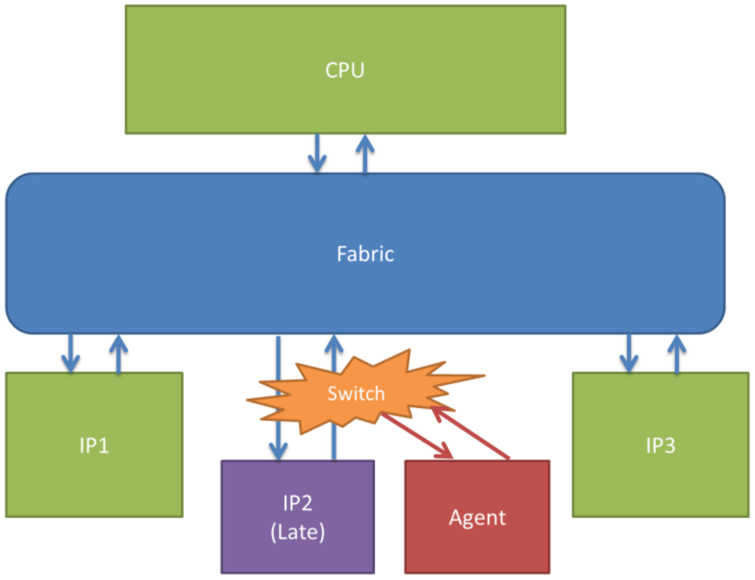

Figure 8 IP2 being replaced with an agent using the switch

Ease of traffic generation

Sometimes it is difficult to generate traffic at the interface of an RTL block. This creates coverage holes, which are difficult to fill.

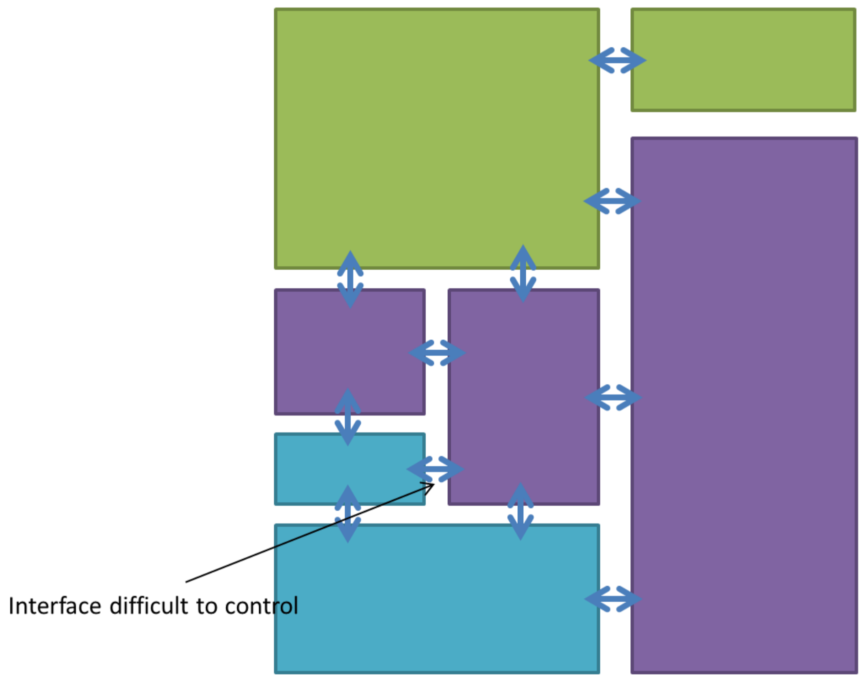

Figure 9 An interface between two RTL pieces is difficult to control

Replacing the RTL block with a model provides easy controllability of the interface and hence coverage closure becomes easy. It must be noted however, that replacing the RTL module with a model should be done when:

1. The verification concerns are well understood. Complete sign-off has to be given only when the tape-out RTL is in place.
2. A separate testbench creation effort is not warranted.

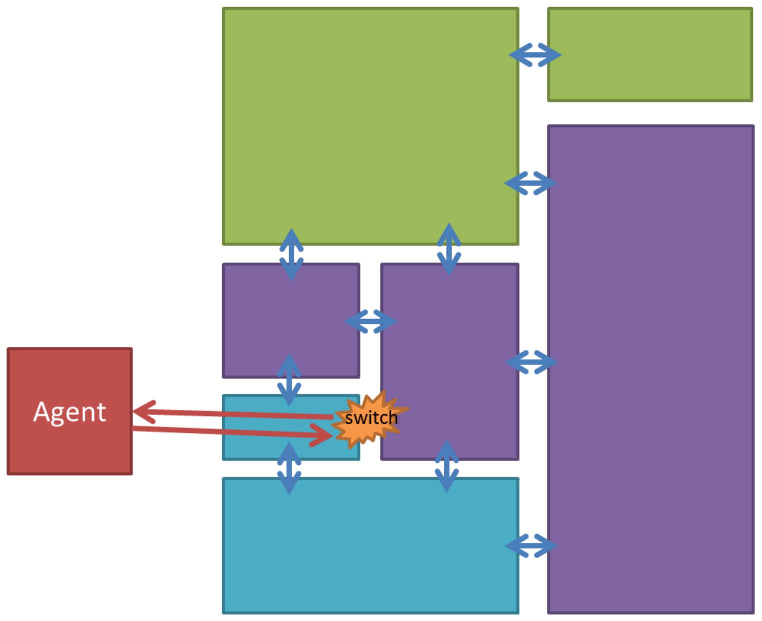

Figure 10 An agent being used to control the interface using a switch

4. Discussion

Results

The RTL-Agent Switch has been used in real designs for the applications mentioned in the previous section. It was found to be of immense help.

In the first application, a CPU's RTL was replaced with an AXI master agent to drive the stimulus into the DUT. Simulation time improved a lot, because the HOST machine was pushing transactions into the DUT, instead of the CPU RTL. Build time was only slightly increased, because only the AXI agent and switch were added to the compilation.

In the second application, a RAM wrapper which was delivered late was replaced by an APB slave agent. This prevented any schedule slips because test development continued unhindered. Only a few days of extra effort were involved setting up and flow-flushing the switch. When the actual RTL was available, a simple change in the command line was needed to get the tests working with the new 'complete' RTL.

In the third application, GPU RTL was replaced with a custom model of the GPU's interface to the rest of the DUT. This was done because it was difficult to control the interface from the GPU's RTL to obtain coverage. Because it was easy to control the model, coverage closure was done easily. Basic tests were still run with the full GPU RTL in place, to validate the tape-out RTL.

Challenges

One of the most important challenges while designing the switch module and connecting it to the simulation environment was to handle the 'force' statements. This is because forcing a wire overrides all drivers of all the fanouts of this wire. This is a bit obvious for normal wires, but even for ports the same holds true. Forcing an instance's port not only overrides the drivers of the port's fanout inside the module, but also overrides the drivers of any wire connected to ports going to other instances. Here is an example:

```
      ip1 u_ip1(.port1(wire));
      ip2 u_ip2(.port2(wire));
```

If `u_ip1.port1` is forced, that 'force' affects `u_ip2.port2` as well.

Basically this necessitates the existence of an 'assign' statement, which is unidirectional.

```
      assign wire1 = wire;
      assign wire2 = wire;
      ip1 u_ip1(.port1(wire1));
      ip2 u_ip2(.port2(wire2));
```

Now if `u_ip1.port` is forced, `u_ip2.port2` will not be affected.

Therefore, care must be taken to understand the connectivity of your design when connecting signals to the switch using 'alias' statements.

Using alias statements creates another complication – the widths of the signals being aliased have to match. Thus proper padding is necessary in case the agent's ports differ in width. This should be done with caution so as not to change the behavior of the model, potentially causing verification holes.

Limitations and Drawbacks

- A limitation of this method is that we can use only port-accurate models. Non-port-accurate models can be used, but only by adding a port-accurate wrapper over the model and then using the switch.

- Note that even during "Agent-only mode" the RTL that was swapped out still exists in the design, although it is not functional. Keeping the RTL in place has some impact on build time, simulation time and memory footprint. In comparison, an approach in which the RTL is stubbed out doesn't have this drawback.

Enhancements

- During the deployment of the switch, we ended up having multiple instances in the entire simulation. We needed the freedom to have each switch select either of "RTL-only mode" or "Agent-only mode". But in the implementation discussed so far, we have used plusargs which are global. So, a nice feature is to add code to uniquify plusargs per instance of the switch. This can be done by taking a string as a parameter and using that string as a part of the plusarg.

```
parameter unique_string = "";
string agt_only_mode = $psprintf("%s_agt_only_mode",
unique_string);
if ($test$plusargs(agt_only_mode)) begin
end
```

- At times, it was desirable to have the capability of switching the mode of operation of the switch mid-way during the simulation. For example, "Agent-only mode" enabled in the beginning and then "RTL-only mode" to take over when a specific event is encountered. This can be achieved by adding something called a "mix mode". This cannot be done per transaction but only at specific barriers. i.e. we can't interleave transactions between these two sources – RTL and the Agent. The example usecase would be to use "Agent-only mode" for the boot and switch back to "RTL-only mode" later when the test starts executing main phase.

```
if ($test$plusargs(agt_only_mode)) begin
  //code for agent only mode
  force a = b;
  force c = d;
  //wait for event
  @(posedge change_mode);
  //code for rtl only mode
  release a;
  release c;
end
```

Future Work

- The way the switch is currently deployed, we have one switch replacing one interface on a piece of RTL. We can extend this approach and add switches to multiple interfaces and potentially all interfaces in the DUT. We can then have the capability of swapping in and out models of various blocks and do simulations such as:
    1. One piece of the DUT is RTL and the rest is agents.
    2. Architectural level simulations if all RTLs are swapped with agents.

- Though this document talks about switching RTL with agents, this approach is not limited to only replacing RTL. We can replace gate-level design blocks with agents as well.

Also, the “agents” can be System Verilog models or System C models or even architectural models.

5. Summary

The RTL-Agent switch is a useful tool that can help an engineer add useful functionality to an existing testbench without modifying RTL or overall testbench topology. By removing the various costs associated with build-time testbench variants, it is easier for the engineer to accelerate simulations or replace functionality of RTL with a behavioral agent. As a final word of caution, it is important to note that the switch helps dynamically switch between RTL and agent, thereby improving and fastening verification flows, but for final verification sign-off, the switch must of be turned ’off ’ and the DUT must be all-RTL, because that is what will be fabricated.