Improving analog design verification using UVM

Mike Bartley - March 23, 2015

As technology becomes more integrated into our everyday life, our chips need to better communicate with the analog world. Most modern system on chip (SoC) designs therefore contain analog and mixed-signal (AMS) elements integrated with digital components. According to Sandip Ray of Intel, AMS elements currently consume about 40% of the design effort, and an estimated 50% of errors in recent chips that require a redesign are due to bugs in the AMS portion of the design [1].

This increase in AMS content in silicon creates several verification challenges: how do we verify the analog design itself, its integration with the digital, and whether the combination achieves the intended overall function? However, there is no standard or even widely adopted approach to this despite the continuous increase in analog content. One potential route is via an efficient, reusable AMS verification approach using the Universal Verification Methodology (UVM) outlined below:

To demonstrate this approach, let’s use the AMS design shown in Figure 1. This is a very simplified version of a power management design where the LDO output voltage is a function of the control signals from the digital sub-chip, the bias currents from the bias generator, and the reference voltage.

![Figure 1: A simple power management AMS design](image-url)
This approach to verification is multi-layered:

1. We must verify the analog IP design itself using traditional SPICE simulations

2. We must verify the integration of the analog with the digital:
   i) At block level
   ii) And at SoC level

Regarding point 2 above, the first major consideration is the speed of simulation. SPICE simulations are notoriously slow (even on the lowest accuracy) and so many engineers develop a model of the analog block to enable faster simulation speeds. The model will often use real numbers or electrical signals. However, this creates a third verification task:

3. Verification of the accuracy of the model

In this outline I will focus on point 2 above and we will consider how to do this using a constrained random approach based on UVM. An outline of a UVM test bench is shown in Figure 2 below.

![Figure 2: UVM test bench](image)

Some explanation of Figure 2 is required first:

- The UVM Driver and Monitor are standard UVM test bench elements that perform all of the communication with the AMS test bench. They model AMS signals as discrete-time real value numbers.
- The “RVM to Electrical Modelling” block within the “AMS TB” allows us to convert real numbers to
electrical values (assuming we have used electrical discipline in the “VAMS model”).
- The UVM environment can now use real values in our key elements of the test bench: constraints, assertions, and coverage. An example code fragment is shown below:

```verilog
real vbat;

covergroup comparator_cg @(posedge vco_if.clock);
    vbatvalues : coverpoint vbat {
        bins power_cut       = {[0.1:0.5]};
        bins low             = {[0.6:1.5]};
        bins full_power      = {[1.6:5.0]};
    }
    Comp_cross : cross vbat_trim, vbatvalues;
endgroup
```

Figure 3: Example coverage code fragment

Any checker has to recognise that analog signals can change continuously rather than just on clock edges as in synchronous digital verification. For example, we might want to have a self-checking mechanism for the peak and trough values on a VCO output, which is a continuous analog waveform, as shown in Figure 4.

![Sampling peaks and troughs](image)

Figure 4: Sampling peaks and troughs

Existing language constructs do not allow us to have a continuous frequency check assertion on an analog signal. The solution is to convert the value to digital and perform some computations on the values (as in the sample code below).
So, we are able to bring the full power of UVM to a test bench that mixes digital design with models of the analog parts of the chip. This allows us to better verify both the digital-analog interfaces and also the combined functionality of the digital and analog. We still need to verify the analog SPICE netlist as well as the model used in the UVM test bench above, but those are topics for the future.

This blog is based on a presentation that Mike Bartley and Jeganath Gandhi of TVS delivered at DVCon Europe in October 2014.

References:

1. *Verification of SoC Designs with Integrated Analog/Mixed-Signal Components*, Sandip Ray, Strategic CAD Labs, Intel Corporation

---

```verilog
defined virtual task clock_monitor();
  var time prev_time;
  var real ldo_output;
  var real peak = 1.0;
  var real trough = 0.0;
  var real cur_sample;
  var real prev_sample;
  var real prev_2_sample;
  var bit first_peak = 0;

  @(posedge vco_if.clock)
  forever begin
    @(posedge vco_if.clock)
    cur_sample = vco_if.ldo_out;
    if(cur_sample == peak) Begin
      first_peak = 1;
      end else if(first_peak == 1) begin
        // Frequency Monitor
        if(cur_sample < prev_sample && prev_sample > prev_2_sample) begin
          peak_q.push_back(prev_time);
        end
        if(cur_sample > prev_sample && prev_sample < prev_2_sample)
          trough_q.push_back(prev_time);

  prev_2_sample = prev_sample;
  prev_sample = cur_sample;
  prev_time = $time;

  if(peak_q.size() != 0 && trough_q.size != 0)
    begin
      clk_period = (trough_q[0] > peak_q[0]) ?
        (trough_q[0] - peak_q[0]) : (peak_q[0] - trough_q[0]);
      if(!(clk_period > exp_period - err_margin) &&
        (clk_period < exp_period + err_margin))
        uvm_report_error("ERR_PERIOD",
        "Expected : $t", clk_period, exp_period));
    end
  end
endtask : clock_monitor
```

Figure 5: Checker Code Sample