CONNECTING UVM WITH MIXED-SIGNAL DESIGN

Ivica Ignjić
Elsys Eastern Europe d.o.o.
Omladinskih brigada 88b
Belgrade, Serbia
ivica.ignjic@elsys-eastern.com

***Abstract*-With increased number of mixed-signal SoC designs and accordingly mixed-signal verification needs, UVM as a proven verification methodology for complex digital SoC is imposed as a solution. However, many mixed-signal UVM verification approaches are present, with no standardized methods of connecting UVM environment with the Mixed-Signal design. For these reasons, efficient Mixed-Signal design verification is becoming challenging and opens space for innovative verification solutions. This paper will show different ways to connect UVM environment with the Mixed-Signal design when Verilog-AMS models are used.**

I. INTRODUCTION

Mixed-Signal verification is very young, but as devices are getting more and more complex, the need for Mixed-Signal verification is increasing rapidly. There are a number of approaches present today, all of them with some advantages and flaws, and all of them have a common question: How should the analog modules be modeled? There are three major approaches:

1) Very high level of abstraction models (VHDL, Verilog),
2) Real Number Modeling that takes things more into "analog," but in the discrete time domain and using only digital simulator engine,
3) AMS modeling where we go almost entirely analog.

In the first case, we can connect UVM environment directly to our DUT as there are no new disciplines added. When using WREAL models, we can directly connect our UVM environment with DUT since WREAL is a discipline that allows us to define a real number as a port of the module. Finally, with AMS models we are not able to connect our UVM environment directly to DUT when we use the electrical port. Using AMS we need some component that can make this possible for us.
This paper will stick to Verilog-AMS as a most challenging approach in Mixed-Signal verification.

II. ENVIRONMENT

Usually, Mixed-Signal DUT has one big digital part and many analog modules for internal (such as bias or references) and external (such as power regulators, sensing, etc.) usage.

When we use digital approach, we write models in VHDL/Verilog language that are strictly behavioral with a high level of abstraction. In this case, we can connect our UVM environment directly to DUT since we use only digital signals. This approach is very simple and simulations are very fast. However, it has one big drawback, with this kind of models it is a high possibility to miss actual functional bugs. This method is useful for connectivity checking, and it does not require for the model creator to have a good knowledge of analog modules, he only needs to know the basic function of the module. Models are straightforward and easy to understand. If we want to include some module as TL (Transistor Level) schematic, then this method did not do well. Many automatic converters will be instantiated, and debugging them can be a very time-consuming task. The main issues are bidirectional ports, current ports, and different voltage levels that cause automatic converters to behave faultily. Most test cases that are written for VHDL model cannot be reused in case of the mixed-mode simulation, so the new test case needs to be created. For example, for LDO (Low Drop Out) we will set the output to 0V when disabled, 5V when enabled and any other value (0.01V…) in case some error occurred (such as wrong supply or bad combination of control signals). An example of VHDL LDO model is shown in Listing 1. We will then create checkers to be aligned with this model. When we include this LDO as TL schematic, then our checker would fail for sure! For this reason, we need to create different test case and checkers.

Listing 1: An example of LDOs output voltage modeled in VHDL

```
...

//LDO output voltage modeled in VHDL
VOUT <= 0.0 when EN = “0” else
        5.0 when EN = “1” else
        0.01;

...
```

We can improve this by using WREAL models. An advantage over VHDL models is that WREAL modeling allows us to use real numbers as ports. Because of this, we can now create the analogish behavior of the module. Now we can mimic the exact behavior of the module, but in the discrete time domain that gives us fast simulation with accurate results. It also has one big drawback, and it is due to a fact that with WREAL models it is not possible to model voltage and current at the same port at the same time, and we need to decide what is more important. For example, if we have an LDO output pin that has a reverse current limiter, then we are not able to model both voltage output and current input at the same pin. Usually, in this case, we model voltage functionality at the LDO output. In addition to that we create another mechanism to drive some variable in the model that will represent inverse current, and with this mechanism, we can verify that inverse current is limited by the limiter block and that proper flag is reported to dedicated status register. Also, in some cases, it is not possible to do a workaround. For example, if several modules are on the path of some current, we cannot correctly verify connectivity using only WREAL models. In this case, we must use TL schematics. WREAL models are also prone to automatic converter errors, especially when current port is in the path. The example of the usual configuration where this kind of issue can occur is in the Fig. 1. With this configuration, E2R converter will be created in between these two modules. This converter usually has an input impedance of a few hundred ohms. This low impedance will cause false current leakage. All of these converters can be manually fixed, but if we consider fixing every single automatic converter manually, then what is the point of having them? This process of debugging all possible converters and issues takes significant time from the verification engineer. Not to mention that with every new project, if we reuse the same model, the process of the debugging and setting up the environment is not skipped, but we need to do it all over again.

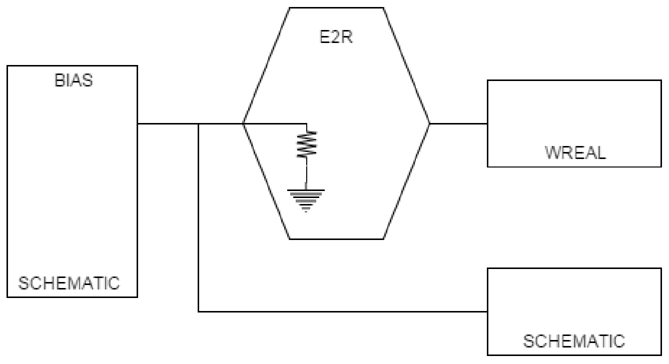


Figure 1: WREAL automatic converter issue

By using AMS models, we are able to model the analog behavior very accurately. We can also very easily switch between model and TL schematic since electrical discipline is used for analog pins and it can be connected directly to TL schematic ports. We don't need conversion elements when connecting electrical ports to TL schematics. However, in this case, we cannot connect directly to UVM, and we need some kind of connection module that will make this possible. Cadence, for example, provides automatic connection modules (L2E, E2L…) that can be used for this. In case we use only this kind of connections, we would need to have very complex UVM monitors that can process data from DUT. This type of monitor needs a clock and a complex discrete processing even for very simple tasks such as checking valid voltage on LDO output pin. When it comes to driving analog pins, we need to take care of these connection modules when thinking of rise and fall time. To mimic this, we would need to make a UVM driver to drive step-by-step voltage respecting step size, rise and fall time and all other analog parameters. This approach is making the verification engineer job much harder.

Instead of all this, we can create Verilog-AMS adapter that can do most of the job for us, avoiding usage of automatic converters, allowing top-level verification engineer to spend more time on the UVM side than on the analog side.

One example of UVM environment is shown in Fig. 2. It has all common UVM components with the addition of Verilog-AMS adapters for every UVC that is dedicated to the analog module, in this particular case, LDOs and SMPSs (Switched Mode Power Supply) as part of the power management device. In this case, we differentiated analog and digital pins of the device, meaning that we connect all digital pins directly to the UVM virtual interface, and analog pins through Verilog-AMS adapters. This image does not show all components (such as supply uVCs), but only the major ones.

Verification process had two stages:

1) Digital verification, where we instantiated digital part (digtop) as the DUT inside of our testbench and verified it as top-level. The biggest challenges were checkers, as we had to write them in a way that will allow us to reuse them in top-level verification. This stage was done in parallel with modeling.

2) Top-level verification, where we integrated all the models and reused as much as possible from the first stage. After verification with models had been done, we ran all tests in mixed-mode simulations where critical signal paths were used as TL schematic.

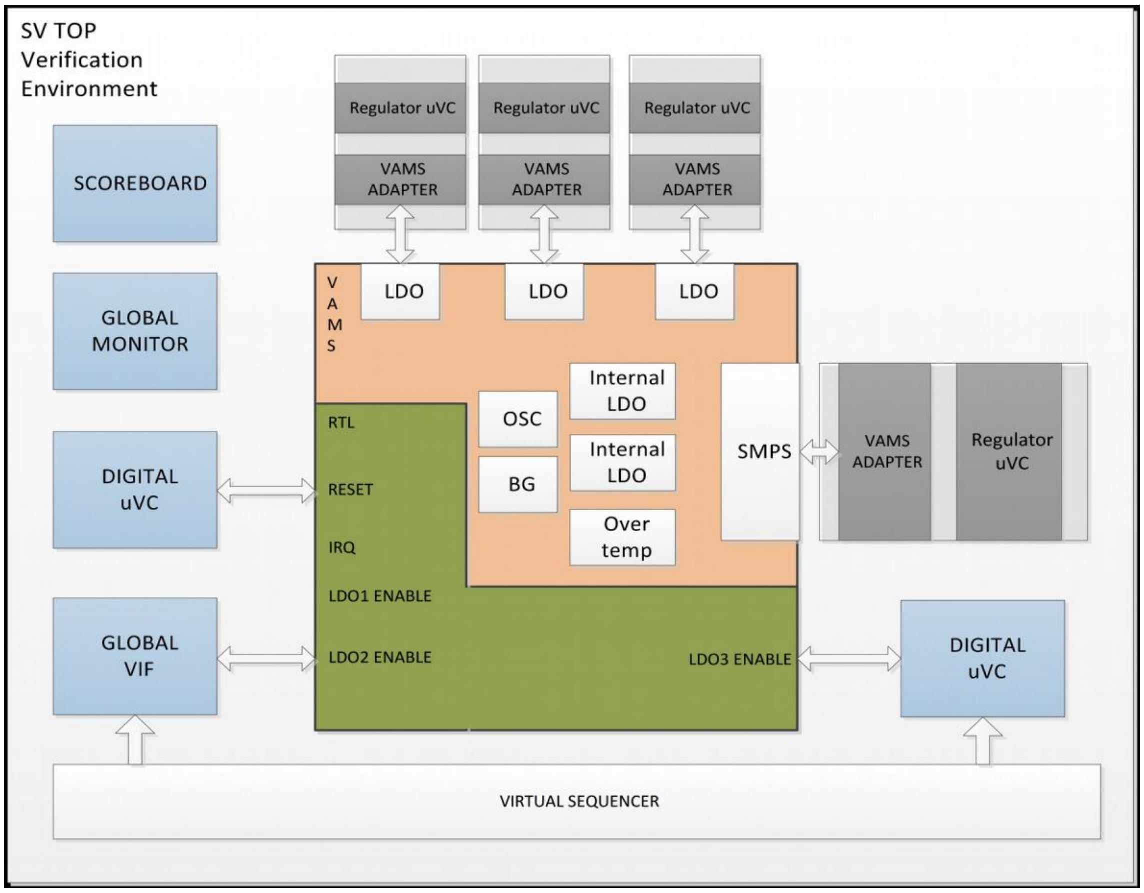


Figure 2: UVM environment example

III. VERILOG-AMS ADAPTERS

Verilog-AMS adapter is a custom connect module between the UVM environment and the AMS DUT. There are several essential elements that every adapter needs to contain for whatever module it is intended. An important aspect of the adapters is that they are reusable and can be considered as a part of uVC that adapter is created for, and thus it needs to be coded in a structured way. The main elements of any Verilog-AMS adapter are static configuration, dynamic configuration, drivers, monitors, harness. It proved to be the best way to use the adapter as interface even in cases when the port is modeled as logic. This way, in case we include TL schematic instead of the model where we have a logic port, we can instantiate needed drivers and monitors within the adapter to avoid using automatic converters. Connection and block diagram are shown in Fig. 3.

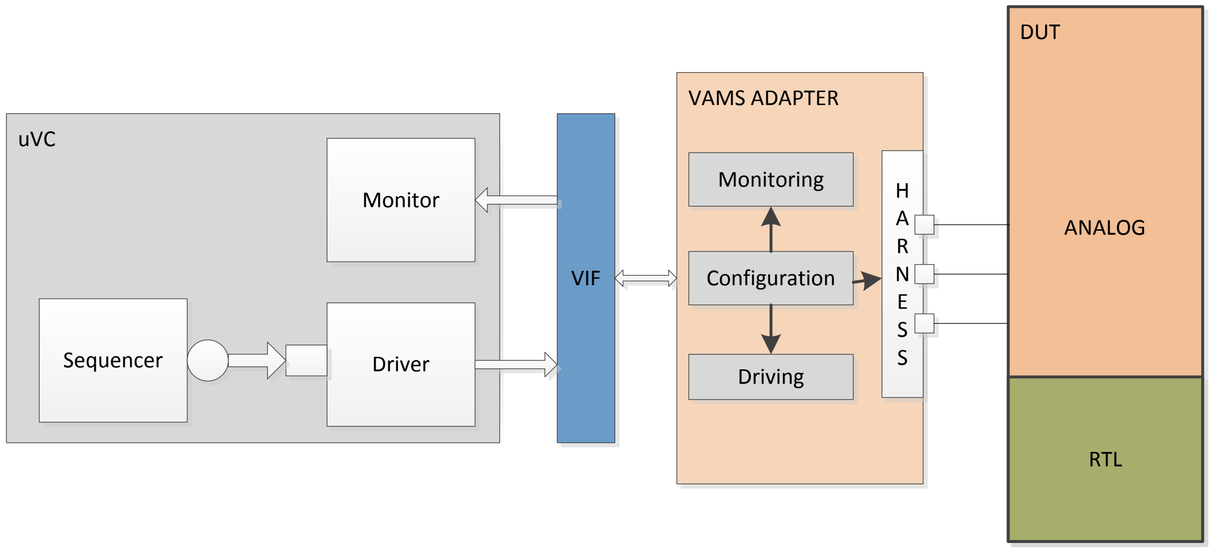


Figure 3: Verilog-AMS adapter connection with the uVC and the DUT

Static configuration is used to configure all parameters of the adapter at the beginning of the simulation. All the initial values of external components in harness, all initial values of the thresholds for monitoring comparators, all driver configurations. Static configuration is used only at the start of the simulation.

Dynamic configuration is used to change adapter parameters on the fly. Using dynamic configuration we can change resistor values to simulate load change, we can update monitor thresholds to align them with the register values inside the hardware.

In the example from Listing 2, we can see the simple dynamic configuration that reflects register over-voltage threshold taken from the specification. In this example, the UVM sequence would write the desired threshold to a configuration register inside the DUT. At the same time, it would send the same value to the example_adapter so that the adapter itself can set the same configuration. Now, we only need to wait for an over-voltage event.

Listing 2: Example of the over-voltage threshold dynamic configuration

```
module example_adapter(dyncfg_ov_th1, ... )

input [3:0] dyncfg_ov_th1;
...
//Over-voltage threshold dynamic configuration
real OV_THR;
always @(dyncfg_ov_th1) begin
  case (dyncfg_ov_th1)
        4'b0000: OV_THR = 4.0;
        4'b0001: OV_THR = 4.1;
        ...
        4'b1111: OV_THR = 6.0;
        default: OV_THR = 4.0;
  endcase
end
...
endmodule
```

Drivers are usually voltage or current sources, depending on the pin nature. Usual driver parameters are voltage/current value to be driven to a pin, rise and fall time, delay, output impedance, etc. In some cases, the driver can be more complex, i.e. when there is a need to switch from voltage to current driver. Driver can also be used to mimic the faults on the pins. An example of such driver is if we have a pin that has the capability to detect the over-voltage condition. In this case, we can create such topology that will allow us to drive over-voltage condition using the driver. Value to be driven to the DUT is sent from the UVM driver over the virtual interface. This external circuitry topology allows us to test an actual over-voltage event on the external port of the DUT even with the TL schematics without any further changes.

Listing 3 shows an example of the simple driver that is just a voltage source with configurable voltage and transition time. The UVC can now easily change the voltage value by sending a v_set value to the Verilog-AMS adapter. The Verilog-AMS adapter driver can be much more complicated, depending on the verification needs.

Listing 3: Example of driver code in Verilog-AMS adapter

```
module example_adapter(VIN, GND, ...)

output VIN;
input GND;
electrical VIN;
electrical GND;
...

//Use wreal if you need to connect through ports
real v_set;    //In testbench connect v_set to vif.
real t_trans;  //In testbench connect t_trans to vif
...

analog begin
  V(VIN, GND) <+ transition(v_set, 0.0, t_trans);
  ...
end
endmodule
```

Monitors are usually simple comparators with configurable thresholds. Comparators are used to return error flag to the UVM environment to be able to easily check if a signal reached the desired level, or some error occurred. As previously stated, we can drive over-voltage event, on the same pin we connect monitor and configure comparator threshold to be aligned with register values in the DUT. Monitor will capture if the over-voltage occurred, and send the information to the uVC. The uVC will capture that information and do the necessary checking.

Example code for monitoring block is shown in Listing 4, where simple over-voltage detection is shown. OV_THR variable is the same as in Listing 2, where it is shown how a dynamic configuration block is working. In this example, we use that configuration for the comparator. When VOUT goes above OV_THR comparator output will be asserted, and this flag will be collected by our uVC monitor. Detected over-voltage is shown in Fig. 4.

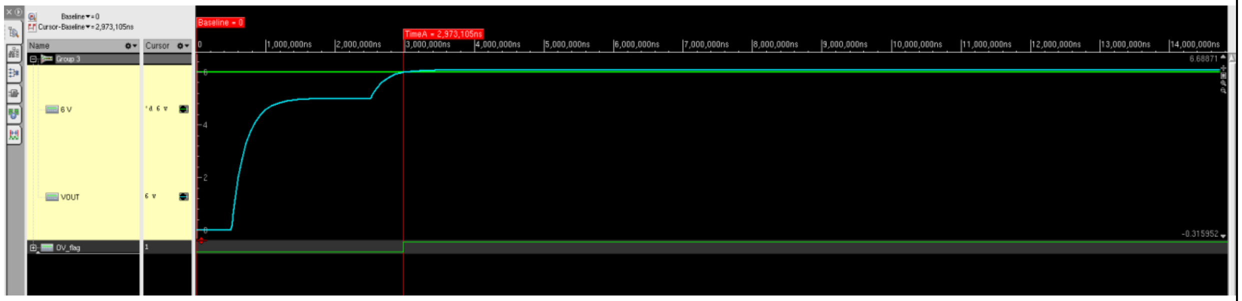
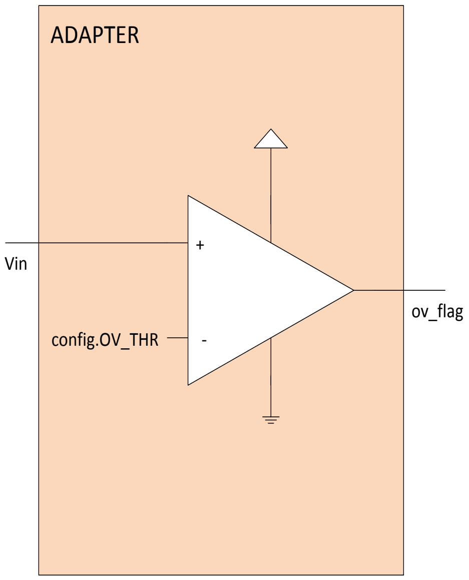


Figure 4: Over-voltage detection diagram

Listing 4: Monitoring block example and comparator diagram

```
module example_adapter(OUT, GND, ov_flag_o, ... )

input VOUT;           //DUT output voltage
input GND;         //Ground
output ov_flag_o;  //Over-voltage flag output

electrical VOUT;
electrical GND;
...

//Over-voltage monitor
reg ov_flag_s;
always @(above(V(OUT, GND) - OV_THR)) begin
  ov_flag_s = 1;
end
always @(above(OV_THR - V(OUT, GND))) begin
  ov_flag_s = 0;
end
assign ov_flag_o = ov_flag_s;

...

endmodule
```



A harness is a block that contains all external components needed for analog modules to work, i.e. resistors, capacitors, inductors, external transistors, etc. Harness topology cannot be changed dynamically, so it needs to have a mechanism to provide a way for error injection (such as over-voltage, over-current, etc.). This error injection is the most challenging part of creating a harness. There are situations where this might not be possible at all, but some other kind of error injection would be needed. For example, if it is not possible to inject error from the harness because of the nature and complexity of the functionality, then it can be done by forcing internal values inside the DUT.

Following this structure, we are now able to trigger faults on pins. Also, it allows us to create very simple and easy uVCs and monitors, keeping them almost the same as for digital approach, since adapter will return only the error flag. This means that the uVC will catch the error flag and check if it is expected or not. Also, adapters provide us ways to easily drive signals without going into complex uVC driver coding. If needed, the adapter is capable of returning the voltage or the current value back to the uVC, for example, if we want to print error message with voltage information as shown in Listing 5. If OV event is not expected (desired by the test sequence), then UVM_ERROR will be reported with a message containing the voltage value of Vout.

Listing 5: uVC monitor assertion example

```
ov_unexpected: assert ((transaction.ov_flag===1'b1)&&(transaction.set_ov_flag===1'b1)) else
  `uvm_error("OV_unexpected", $sformatf("OV occured unexpectedly. Vout=%.2f",
vout_voltage.sample()))
```

Sample method will read the current voltage value from the dedicated Verilog-AMS adapter. This means that adapter needs to have a freeze frame mechanism. In this case, the adapter needs to sample voltage when the overvoltage error occur and store it in the variable as shown in Listing 6. When uVC monitor calls the sample() method, it will actually collect value from the adapter freeze frame. Freeze frames are not needed for the basic functionality of the adapter, but they are useful for message reporting and easier debugging.

Listing 6: Freeze frame example

```
module example_adapter( ... )

...

//Freeze frame block
real freeze_OUT;
always @(posedge ov_flag_s) begin
  freeze_OUT = V(OUT, GND);
end

...

endmodule
```

IV. PROS AND CONS

Like any other methodology, using Verilog-AMS adapters as an interface has its advantages and pitfalls. It is worth to mention that, from our experience, this proved to be the best method for various types of DUTs, including power management, sensor applications, etc.

The main reason why this method is good is that the UVM verification engineer will have much more time to spend on actual verification tasks, rather than coding complicated drivers and monitors. Instead of writing monitor that will work on some flying clock and that needs to sample voltage values and store them into an array, then doing some processing on that array so you can simply detect the fault event on some pin, with this methodology it is only needed to write a simple checker that is triggered by a flag returned from the adapter. The drawback is that analog engineer that wrote the Verilog-AMS model, will need to write adapter too. Even though adapters are usually very simple, this task can be time-consuming depending on needed functionality to be covered.

Another benefit is that with this approach, we don't have to alter anything when running mixed mode simulations. Exactly the same testbench setup is used when the TL schematic is selected instead of the AMS model. Automatic connect rules are sometimes causing false results because they have their default parameters. When using them we need to be very careful, especially when the bidirectional port is used. Using Verilog-AMS adapters, we no longer need to take care of them, as the adapter itself is actually a connection module.

In case we need to use WREAL models, only a few small changes are needed to make this work.

The biggest pitfall is with signals such as sine waves. It is very difficult to monitor and check this kind of signals in a simple and easy way like with the adapters. The main reason for that is that it is very hard to write a monitor that is capable of tracking sine wave. Amplitude and frequency are quite easy to monitor, by using zero crossing and max functions, but the trouble here is that we cannot easily know if this signal is the actual sine wave. Fig. 3 shows two different waves, one sine wave and one saw wave. The adapter can monitor if signals are in good range or it has gone over-voltage, it can monitor if the frequency is correct, but it cannot differentiate between these two signals. If we run sine and saw wave with same amplitudes, same frequencies, the adapter will not be able to tell what wave it is. This is something that needs to be implemented inside the UVC monitor using FFT (detect highest harmonic) or a similar technique. This is probably something that creates space for the biggest improvements in the methodology.

Another big drawback is that it can take significant time until we are able to run the first simulation on top-level. This is because we need to model analog module and adapter together. This task is by default very time consuming, and they are critical regarding verification. If the models and adapters are done correctly, top-level verification tasks are really easy, leaving space for exhaustive verification of all device features.

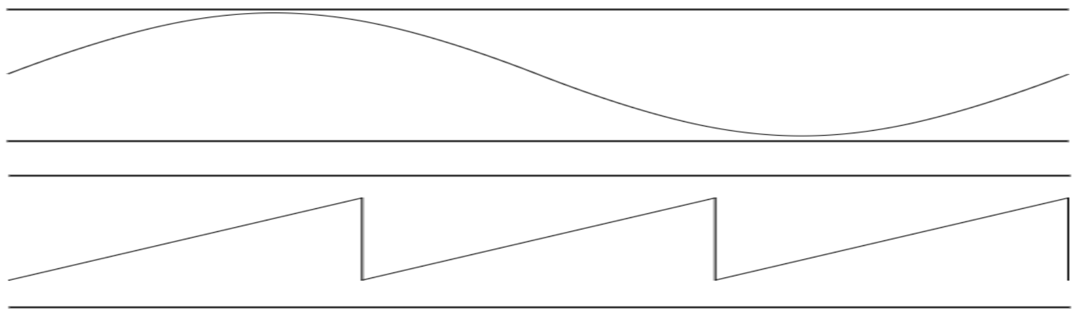


Figure 5: Main drawback

V. Conclusion

This methodology is quite young and it has plenty of space for improvements. As seen in section IV, there is a number of drawbacks, and all of them require additional effort to be resolved. However, despite this additional effort, this proved to be a good method of finding bugs, and not just the bugs in digital part, but also the bugs in the analog part of the device. It is known that using a digital approach we are able to verify mostly the connectivity between analog and the digital part. This method allows us to model all features of the device, including analog loops and feedback. This method proved to be the best method when we need to verify how analog modules behave when they are integrated into top-level. We also have been able to find lots of hard to find bugs that are usually found on the silicon in the lab. These are the bugs like leakage in the analog module, impedance incompatibility and similar. Finding this kind of bugs is very important because it significantly improves lab results, and of course, increase chances to have fewer re-spins of the same chip.