With great power comes great responsibility:

A method to verify PMICs using UVM-MS

Dor Spigel
Mixed Signal Methodology Engineer
Microsemi POE Business Unit
1 Hanagar, Hod Hasharon Israel
dspigel@microsemi.com

Moshik Hershcovitch
Senior Verification Leader
Microsemi POE Business Unit
1 Hanagar, Hod Hasharon Israel
mhershcovitch@microsemi.com

***Abstract*— the increasing demand in "smart power" mobile applications has led many SOC (system on chip) designs towards advanced methods such as power gating, voltage changing and frequency scaling. A power management unit (PMU) is a mixed signal IC that controls and monitors the power consumption of peripherals in the SOC. However, the task of verifying this unit faces a challenge when trying to approach it with real number modeling (RNM) methodologies; which are up to X100,000 times faster than SPICE models. When using this methodology with VerilogAMS/wreal, the approach is to identify the main electrical property of an interface (i.e. either the current or the voltage) and to create a behavioral representation of it.**

The challenge in PMIC's modeling, using RNM, lies in the fact that the boundaries between "inputs" and "outputs" are unclear, when discussing powering interfaces. This is created due to the impedance mismatch between the load and the driver. This impedance mismatch cannot be overlooked when the design in question is a PMIC, since most of the simulations are performed using loads that stress the boundaries of the design.

Our new contribution to this field is to use the UVM configuration database to transfer information between the load and the driver. This eliminates the need of defined boundaries between the load and the driver, since the driver has all of the information. This method allows project to project reuse regarding tests and models. Only by having a totally independent set of driver/load API, can we have completely reusable verification components.

Our proposed methodology is to preconfigure the model with the load parameters in a reusable and scalable way that integrates well into the newest trends in AMS (Analog Mixed Signal) verification platforms (UVM-MS). The analog portion of the design is replaced with behavioral models, and instead of SPICE natives, the loads should be VerilogAMS blocks with local variables as their properties (resistance, capacitance etc). The loads are connected to the driver in such a way, that changing variables inside the loads would trigger the same change in the driver's variables. This provides the required separation between the load and the driver that allows the use of RNM.

Keywords – UVM_MS; RNM; PMIC; Load UVC

I. INTRODUCTION *TO* REAL NUMBER MODELING

When dealing with mixed signal power controllers the greatest challenge is simulating and verifying digital to analog interaction. For the past decade we have been developing ICs for POE power suppliers and dealing with this challenge. A single power management unit may include thousands of control bits and complex digital to analog handshake mechanisms. The AMS verification task was always a daunting one. The simulations took forever and critical bugs were always found near the Tapeout. The vast majority of bugs were integration related errors, such as control bits being hooked up incorrectly or driven the wrong way.

The simplest approach to AMS simulations is to try for the top level simulations ‘as is’ with SPICE models and RTL code. This approach is of course unrealistic in our industry, as these sorts of simulations may take weeks to complete depending on the device count. Until recently, we have been using the ‘Capture and Replay' approach [1]. This method involves recording digital stimuli using a digital simulator at certain time points of interest, and then driving them back to the analog design using a mixed signal simulator. This method proved to be effective in finding connectivity issues but was still very slow to simulate. It could only be done near the end of the design cycle, since both of the designs had to be finished in order to start these sorts of simulations. This method is good at detecting connectivity mistakes but doesn’t excel in verifying complex digital and analog loop back mechanisms.

Another option was to try behavioral analog modeling with VerilogAMS. This method showed some improvement in performance, but the modeling effort was not cost effective. Therefore, a new methodology had to be adopted to overcome our increasingly difficult verification challenge – the challenge of integration.

Analog simulations are based on the iterative SPICE solver. This analog solver is very accurate but too slow to cover all of the different scenarios required for sign off in a reasonable time, since a single top level simulation may take days or weeks to complete. A suggested way to approach this performance gap between the analog and digital simulator is to use behavioral models with real values at discrete time steps. These models were evaluated up to X100,000 times faster than SPICE models in the benchmarks we performed.

The basic concept behind real number modeling is that the analog values are calculated based on events or in discrete time steps. This allows the use of solely the digital simulator. This method of behavioral modeling may not replace traditional

analog simulations, but it will in fact speed up mixed signal simulations up to the level of purely digital verification.

Real number modeling is already a well-established concept used by verification teams worldwide. However, for this modelling method to be applied, several assumptions must be taken [4]:

1. The concept behind the electrical signal can be described as data flow between components.
2. The output impedance should be zero and accordingly the input impedance should be infinite (ideal impedance relation).
3. The transfer function must be known explicitly.

When the DUT is a powering unit, assumptions 1) and 2) are not practical, for the simple reasons that most of the simulations are meant to test how the power supplier deals with overloads and underloads. The boundaries between inputs and outputs are unclear. For instance if the DUT is connected to a passive component, the DUT provides power to the load which in turn consumes current, all on the same wire. This fact makes it difficult to determine a dataflow direction between our components and it harms the modeling effort.

In this paper we will present a proposed modeling methodology for powering devices that will allow the use of quality real number models, even when the impedance relations are not ideal. This will be achieved by using our proposed load_uvc.

The outline of this paper will be as follows:

- In section III we will describe how the UVM configuration database was used to overcome the non-ideal impedance relations.
- In section IV we will show how analog events on powering interfaces can be quantized into UVM data flow packets that can be later used as UVM sequence items.
- In section V we will present a reference UVM environment containing load UVCs and modeled power amplifiers on a real life POE application

II. Assumptions

The examples in this paper are based on Accellera's Universal verification methodology version 1.1d. This paper assumes the reader is familiar with VerilogAMS real number modeling [3] as well as the basic UVM constructs and their typical usage [3]. Knowledge in Systemverilog 2009 is also required [3]. We are aware that Systemverilog 2012 (AKA sv-dc) is out and will soon be supported by most vendors. Please refer to the 'Future Work' section for ideas on what modifications should be applied when sv-dc vendor support widens.

III. Using the UVM configuration database to overcome the modelling challenges

First we shall address the issue of applying RNM to designs with *non-ideal* input/output impedance. Consider trying to model a simple voltage regulator with an output impedance 'Rs'. See description in figure 1:

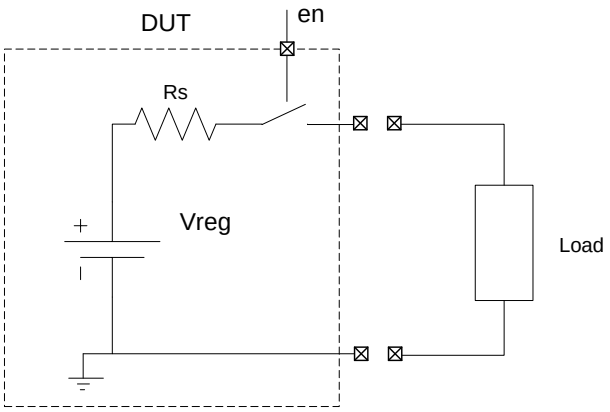

Figure 1: Simple voltage regulator

For the simplicity of the example we will assume for now that our load is a simple resistor 'Rload'. Assuming the ideal conditions of 'Rload' >> 'Rs' the model rather simple:

```
parameter Tstep = 100; // Depending on Timescale
// Note That by decreasing Tstep we can
// tradeoff performance with accuracy
always #Tstep
    begin
        vout_int = Vreg;
    End
assign vout = en ? vout_int : `wrealZState;
```

However, when the design in question is a power supplier, the majority of the simulations performed are done under loads that stress the regulator. Our regulator model has no way of calculating the output voltage without knowing anything about the load that's connected to it. Our suggested methodology allows the regulator model to be able to identify the load, without making any change to the interfaces defined by the analog designer.

For instance, if the parameters of load were available to the driving regulator, then the modelling effort would have boiled down to calculating a simple resistor divider. The UVM configuration database is a convenient tool used to set parameters for verification components in a way that is manageable and reusable. By using the configuration database we could have the load communicate with the driver model mid test without harming the boundaries of the model.

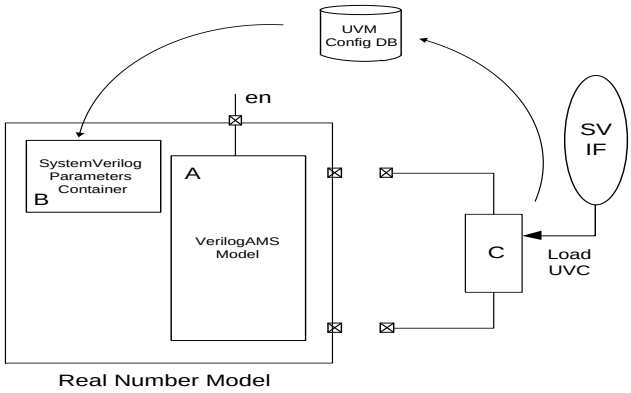

Figure 2: Non-ideal voltage regulator modeling architecture. This is a block diagram showing a proposed way to model the behavior of the circuit from Figure , using the UVM configuration database to pass parameters between the componnents.

Referring to Figure 2 we will now explain the different components and their purpose. Component A is our VerilogAMS driver model. It drives the output voltage based on the digital configuration and the load parameters obtained from the parameters container. Component B is another hierarchy that contains all of the load parameters such as capacitance, resistance, inductance etc. Here, another hierarchy is required in order to separate the VerilogAMS components from the Systemverilog parameter container, since these modules may use different engines and need to be separated. The parameter container module has an interface to the UVM configuration database from which the load parameters are obtained. Component C is the load UVC. This is simply an interface to the configuration database. Note that when the driver supplies voltage, that is calculated based on load parameters, the load doesn't have to be connected to it. However, the load UVC is written in a way that allows the combination of behavioral modeling as well as SPICE netlists, and therefore the load is physically connected (see section II.B)

A. Modelling the Driving regulator

This section refers to component A in Figure 2

First we need to understand the operation of our regulator and to decide which loads to support. For this example we shall assume that our regulator starts driving voltage when a digital 'en' signal is asserted, and that the loads we want to model would be resistors and capacitors. Since we have to know the explicit equation, the output would be:

$$V_{out} = V_0 + (V_{reg} - V_0)\left(1 - e^{-\frac{t}{RC}}\right)$$

Now the matching Verilog code would be:

```
always #Tstep
    begin
          // Get Parameters from Container
          // See A in Figure 2
          R = param_container_inst.rload;
          C = param_container_inst.cload;
        tau = R*C;
        vout_int = v0 + (vreg - v0)*
                        (1-exp(-dt/tau));
        dt = dt + time_step;
    end
assign vout = en ? vout_int : `wrealZState;
```

B. The Parameters Container

This section refers to component B in Figure 2

The parameter container is a Systemverilog module in charge of accessing the UVM configuration database and obtaining the load values for the VerilogAMS driver. The connection between the VerilogAMS driver and the container is done by OOMR (Out of Model Referencing), but the connection between the *load_uvc* and the container is done by the uvm configuration database. Note that since the container is a Systemverilog module, the access to the configuration database is done inside an initial block. Therefore a minimal delay has to be inserted in order to make sure the '*get(…)*' method is called after the test creates the object.

```
module load_parameters_container #(parameter
LOAD_INDEX = 0);

// UVM Config Object
load_config_object m_load_config_object;

// Load Variables
real rload;
real cload;

// Get UVM Config Object

initial
    begin
// to avoid null referencing create a  dummy object
    m_load_config_object = new();
// Minimal delay makes sure this happens after the top
creates the configuration item
    #1;
    uvm_config_db#(load_config_object)::get (…)
    end

// Update Module Variables from Config Object Vars
always @(m_load_config_object.rload)
    rload = m_load_config_object.rload;

always @(m_load_config_object.cload)
    cload = m_load_config_object.cload;


endmodule
```

The driving regulator model could then be verified by comparing its outputs to the ones of the actual design.

C. Creating the Load UVC

This section refers to component C in Figure 2

Please see Figure 3 - the proposed load UVC architecture. Note that the load UVC only serves as a gasket between the module based verilogAMS domain and the class based Systemverilog domain. However, we will show that by using compiler directives, we can use the load UVC both for analog simulations with SPICE netlists still using the benefits of UVM.

Note that the load UVC was designed to allow multiple loads connected in parallel to the same interface. This is useful when you have multiple drivers on the same power line and each one handles different load types.

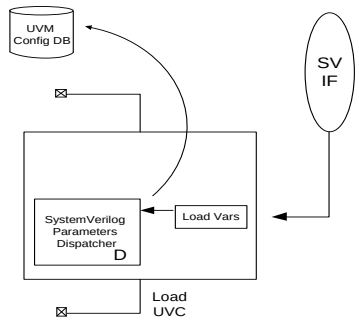

Figure 3: Load UVC Block Architecture

One may ask, if the driver has all of the information on the load then why do we need to actually connect something on the other end? The answer is that while the majority of the simulations would be done using behavioral models, some tests require more accuracy. For these sorts of tests a mixture of SPICE netlists would be used instead of some model blocks. During these scenarios an actual load is required. For these simulations the load uvc acts as a VerilogA model and behaves like a traditional load as well as remaining an interface to the UVM configuration database.

```
module load_uvc #(

                    parameter      LOAD_INDEX = 0,

                    parameter real R_NOMINAL  = 25e3,

                    parameter real C_NOMINAL  = 1e-9

                  )

                  (v_load_p,v_load_n,enable);

input enable;
```

```
// Define Electrical Mode When your driving cell

// is a spice netlist or VerilogA

`ifdef LOAD_UVC_ELECTRICAL_MODE

    inout      v_load_p;

    inout      v_load_n;

    electrical v_load_p;

    electrical v_load_n;

`else //LOAD_UVC_BEHAVIORAL_MODE

    inout v_load_p;

    inout v_load_n;

    wreal v_load_p;

    wreal v_load_n;

`endif
```

Note that the interface may be used as behavioral real numbers (wreal) or as VerilogA *electrical*. The next part contains the parameter dispatcher module instance; this module is similar to the parameter container module, but instead of drawing from the configuration object, it changes the variables that are inside of it.

```
// Load Variables

real r_load = R_NOMINAL;

real c_load = C_NOMINAL;

// Parameter Dispatcher into UVM Database

load_parameter_dispatcher
#(.LOAD_INDEX(LOAD_INDEX))
load_parameter_dispatcher_inst();

// Hookup with Dispatcher

always @(r_load)

    load_parameter_dispatcher_inst.rload =
r_load;

always @(c_load)

    load_parameter_dispatcher_inst.cload =
c_load;
```

The last part of the load UVC contains VerilogA code that allows the load to behave like an analog device, still maintaining its connection to the UVM environment.

```
// Electrical Load Behavior

`ifdef LOAD_UVC_ELECTRICAL_MODE

    // Resistive Value during en == 1'b0

    parameter real ROFF = 10e6; // 10Mohm

    // Declare Branches

    branch (v_load_p,v_load_n) res,cap,load;

    // Analog

    analog begin

        // Res

        I(res) <+ V(res)*transition (enable ?
1/r_load : 1/ROFF , 0 /*Delay*/ , 20p /*Rise
Time*/ , 20p /*Rise Time*/);

        // Cap

        I(cap) <+  ddt(V(cap))*transition
(enable ? c_load : 0 , 0 /*Delay*/ , 20p /*Rise
Time*/ , 20p /*Rise Time*/);

    end

`endif
```

IV. Breaking down Power consumption events into UVM data flow packets

In the previous part we covered the measures that can be taken to overcome the challenge in modeling analog components with non-ideal impedance relations. The second challenge is to quantize the analog events on a power line, as if data was flowing from the load to the powering device.

Consider some events that a typical power supplier might monitor:

- Over Current
- Short Circuit
- Under Current

All of these events start and end at a specific time point, but their nature is continuous and difficult to capture using discrete time events. However, today's PMICs use digital controllers to monitor and react to these events. Therefore, a good starting point would be to model these events at time steps that are at least as fast as the fastest clock cycle in the system. We could even go further with this idea. For instance, if the digital block monitors the current and the voltage through an ADC, then one could decide to model a "short circuit packet" for the duration of a single ADC measurement. For example, consider the following code snippet showing the definition of a load UVM sequence item. This sequence item simply lets the driver configure the parameters for the load under constraints specific for that event (short circuit, underload, etc.)

```
class load_seq_item extends uvm_sequence_item;

    rand LOAD_ERROR_ENUM_TYPE         load_error_e;

    rand real                         load_res;

    rand real                         load_cap;

    // Constraints

    Constraint load_res_c {

              if (load_error_e == SHORT_CIRCUIT)

                     { load_res inside …}

              else if (load_error_e == UNDER_CURRENT)
                     { load_res inside …}

              else if (load_error_e == UNDER_CURRENT)
                      { load_res inside …}

        }
```

The matching load driver only has to assert the load properties to the load UVC and the connection to the behavioral model is done through the dispatcher and the container.

```
class load_driver extends uvm_driver
#(load_seq_item);

...

    task main_phase(uvm_phase phase);

            load_seq_item req;

            forever begin:loop

           // Get Item From SQR

                seq_item_port.get_next_item(req);

           // Wait For the Digital monitoring event


                wait(m_adc_svif.start_measuring_ev);

           // Assert Load Properties to the load UVC

                m_load_svif.R = req.load_res;

                m_load_svif.C = req.load_cap;

            // Finish

                seq_item_port.item_done();

            end:loop

    endtask

endclass
```

V. Building the environment - Application to a real life design

This reference topology was successfully used to Tapeout our latest generation of POE PSE's (power supply equipment).

A. Background

Power over Ethernet is a technology that allows network devices such as IP telephones, WLAN access points, security network cameras and such, to receive power parallel to the data over existing CAT-5 Ethernet infrastructure without making any modification to it.

The POE standard [8] specifies that before powering up any device, some actions must be taken first. Refer to figure 5 for the timing and voltage levels of the required actions before operating voltage is applied.

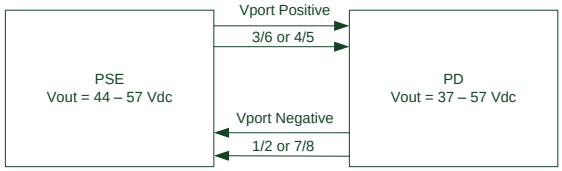

Figure 4: PSE and PD Interconnection; the numbers indicate the Ethernet pin number

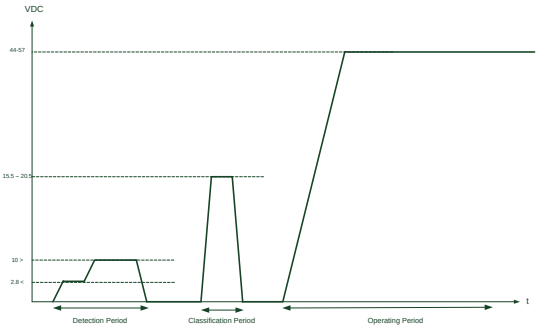

Figure 5: POE Detection to Power up Waveform

Detection Period: before power is applied, safety dictates that a valid PD connected to the PSE, must first be ensured. This process is referred to as 'line detection' and it involves the PSE seeking a specific 25KΩ resistor [9].

Classification Period: after a successful detection, the PSE may optionally proceed to classification. During the classification period a voltage of 15.5 – 20.5v is applied to the PD. The current consumed by the PD indicates its power class [9].

Startup and Operation: after the detection and optional classification, the PSE switches to its full voltage capacity and precedes to monitors the current consumption for power management and responding to overloads [9].

B. Verification Environment Architecture

In this example there are three different analog drivers that are operating on the same line. For this reason, three different load UVCs will be used (actually we could do with one, however it was decided to separate to 3 in order to increase project to project reuse).

The top level of the testbench should contain the digital portion of the design, the analog portion (SPICE netlist or any other behavioral model) and the load UVCs. Refer to figure 5 for the block level diagram of the testbench.

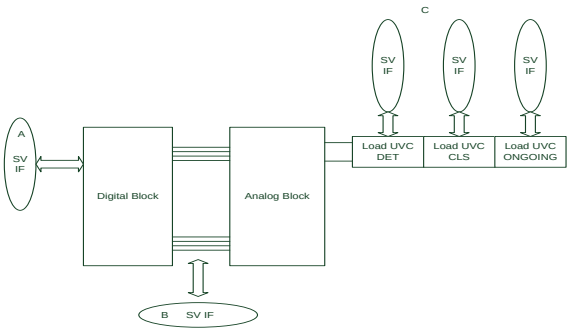

Figure 6: Top level testbench overview

Now that we have this setup, we can simply use the interfaces connected to each load UVC, and control the behavior of our loads during the runtime of our tests. The next steps would be to create UVM agents for the different functionalities of the system, add passive monitors to cover digital to analog interfaces, and create the register abstraction layer.

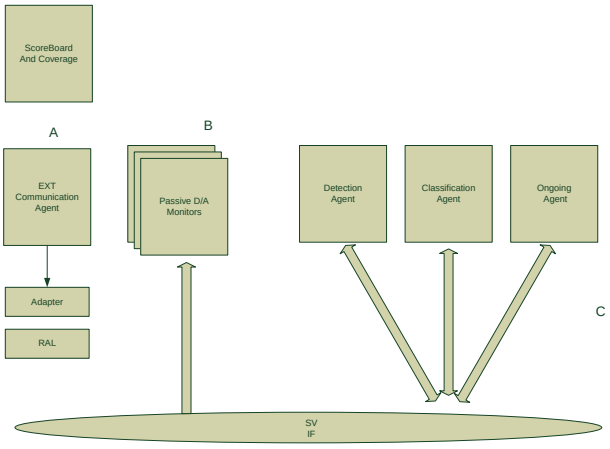

Figure 7: UVM Enviornement Block Diagram. Note that the symbols A,B,C are connected to matching interfaces in Figure 6.

Attached below is a snapshot from our tests simulating the POE detection to Power UP flow using our methodology.

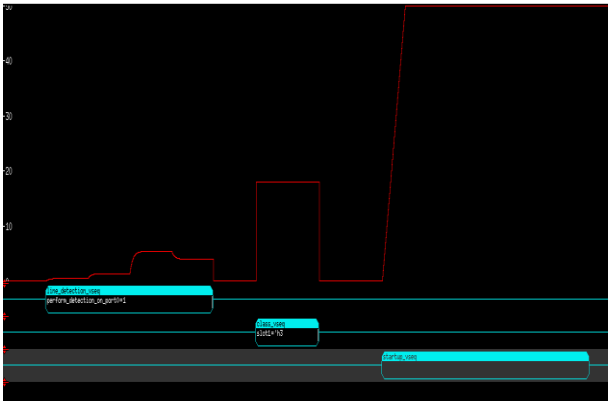

Figure 8: Simulation Snapshot of POE Power Up Cycle with UVM sequences

When enough of the Vplan is covered, we will proceed to perform simulations with actual SPICE models. The motivations behind this, is to make sure nothing was missed in the modeling process and to verify that no analog bugs were created by digital to analog transient events. For instance, checking the full data path from the load to the analog front end through the ADC and finally to the digital controller. Running full top level AMS simulations may not be completed in reasonable times. Several actions may be taken to reduce simulation times even with SPICE models. Since the load UVC could be compiled as verilogA with electrical nets, it could be set to electrical mode and use actual analog designs to drive our configurable load. This will allow us to reuse all of our automatic checkers and coverage collectors, thus allowing the simulation to run in batch mode. However most of the times this won't be enough to bring the simulation up to speed. For that reason what could be done is to decide on data paths that are critical for the verification process and only have those blocks modeled.

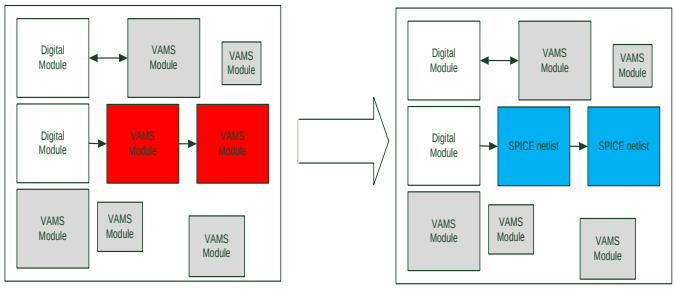

Figure 9: Block diagram of a mixed signal design. Marked in red are the blocks that are in the datapath that needs to be verified ,and in blue the blocks that were replaced by their SPICE netlists. Marked in gray are the other analog blocks in our design.

Attached below is our benchmark for the same simulation shown in Figure 8 for a fully modeled design up to a full SPICE netlisted top level simulation

Actual Schematics in the DUT (from a total of 15 blocks)	Runtime
Fully Modeled	8 seconds
Fully Modeled Design with 1 Cell (small) as schematic	20 minutes
Fully Modeled Design with 1 Cell (Large) as schematic	5 hours
Fully Modeled Design with 3 Cells as schematic	24 hours
Full Chip	1.5 weeks

VI. Summary

This paper introduces ideas for mixed signal power supply ICs simulations using digital verification methodologies. This is the first time an idea for a UVM configurable load is presented, to our knowledge. Our new contribution eliminates the need to separate the load from the driver by including load values inside the driver model. This method allows project to project reuse regarding tests and models.

We are aware that the Universal Verification Methodology has a stiff learning curve; however we believe that the overall cost could be minimized by using the load UVCs, since it may be easily reused for different kinds of projects.

This methodology was implemented by us on our latest generation of POE controllers. We are glad to say that this project was a success. Mo bugs were traced post silicon at the system level.

One of our conclusions is that the incorporation of UVM constructs in analog models implies that the models preferably be written by verification engineers rather than analog designers. Our recommendation is to include the verification engineers during the analog planning phase, in order to increase the model writer's understanding of the analog blocks.

To summarize, we believe that this methodology along with the load UVC could be used to help power supplier design teams overcome their upcoming verification challenges.

VII. Future Work

We are aware of the fact that when Systemverilog DC (2012) is more widely accepted by vendors, the issue of impedance mismatching could be solved using the new user defined resolution functions. Our suggested UVM environment architecture should remain identical even if you choose to migrate to SVDC, this is because our methodology states that the UVM components interact only with the load component.

Meaning that by adopting our suggested methodology only the modelling concept should be modified if you choose to migrate to SV-DC. All of the other UVM components may remain unchanged.

ACKNOWLEDGMENT

We would like to acknowledge the insights and ideas provided by Nadav Barnea to our team effort. We would also like to thank Tamar Lanir for the time spent proof reading this paper.

REFERENCES

[1] Neyaz Khan and Yaron Kashai " From Spec to Verification Closure: a case study of applying UVM-MS for first pass success to a complex Mixed-Signal SoC design", DVCon 2012

[2] Rosenberg, S. and Meade, K., 2010, A Practical Guide to Adopting the Universal Verification Methodology (UVM), Cadence Design Systems.

[3] Online Resources from http://www.accellera.org/downloads/standards

[4] Online Resources from https://www.semiwiki.com/forum/content/3181-speeding-up-ams-verification-modeling-real-numbers.html

[5] Jess Chen, Michael Henrie, Monte F. Mar Ph.D. and Mladen Nizic "Mixed-Signal Methodology Guide Advanced Methodology for AMS IP and SOC Design, Verification and Implementation" Cadence Design Systems.

[6] Bishnupriya B., John D., Gary H., Nick H., Yaron K., Neyaz K., Zeev K., Efrat S., 2012, Advanced Verification Topics, Cadence Design Systems,

[7] Kenneth S.Kundert and Olaf Zinke " The Designer's Guide to Verilog AMS"

[8] IEEE Std 802.3at™-2009

[9] Galit Mendelson "All You Need To Know About Power over Ethernet (PoE) and the IEEE 802.3af Standard" http://www.microsemi.com/documents/powerdsine/whitepapers/PoE_and_IEEE802_3af.pdf