2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Translating and Adapting to the "real" world: SerDes Mixed Signal Verification using UVM

Akhila Madhu Kumar
Intel of Canada, Ltd
250 Ferrand Drive, Suite 810
Toronto, ON – M3C 3G8, Canada
Email:akhila.m@intel.com

Karl Herterich
Intel of Canada, Ltd
250 Ferrand Drive, Suite 810
Toronto, ON – M3C 3G8, Canada
Email:karl.herterich@intel.com

Abstract- In the current trend of high speed interfaces, there are many IPs with critical design and verification challenges. A Serializer/Deserializer (SerDes) is one such example and is a commonly used block in high speed communication to compensate for limited input and output pins. It helps in providing data transmission over differential pair and converts between serial and parallel data in each direction. When talking about IP quality, verification closure time and reliability are of utmost significance and for mixed signal IPs like SerDes, there are additional challenges in improving verification content and reducing the time to market.

The focus of this paper is to address two main challenges in verifying mixed signal multi-lane designs like Physical Media Attachment (PMA) layer in SerDes: 1) Effectively handling real-type data across verification components while handling critical features like pre-emphasis. 2) Architecting a UVM sequence mechanism to generate meaningful stimulus across all lanes and collect corresponding coverage data

I. Introduction

When the analog "real world" and digital 1s and 0s come together, there are many considerations to be done while selecting the appropriate flow for the verification closure. A common trend is to use the analog behavioral models for the initial verification flows and gradually move towards an accurate analog model (with transistor level leaf cell information). Additionally, it is also recommended to run co-simulation with specific sub-blocks replaced by analog schematics. Throughout this flow, it is important to be able to reuse as many verification collaterals as possible like end to end data checkers, traffic transactors and UVM sequence generators. This also opens up opportunities for extending the verification collateral reuse to the SoC or sub-system level.

The two primary testbench components discussed in this paper are the translator blocks and the pre-emphasis driver. These are closely coupled to the design characteristics like the pre-emphasis finite impulse response (FIR) filter size and receiver equalization characteristics. Additionally, they also help in minimizing the number of components which need to handle the real-type data. By utilizing the inbuilt hierarchical sequence mechanism in UVM to develop a layered adapter sequence, the verification environment can help in building robust framework to drive multi-lane IPs like PMA to verification closure. We will talk briefly about generic PMA architecture to explain the common terminologies and also give some details on the commonly used sub-blocks. This solution can easily be extended to any mixed signal design with one or multiple data path lanes.

II. SerDes PMA IP Verification Framework Using Translator Blocks

The primary application of PMA is to transmit the parallel data coming from the higher layers (like Physical Coding Sublayer - PCS) to the receiver circuitry and also to the external serial interface (accounting for the channel losses and attenuation) such that the final data received at the receiver parallel interface is within the error tolerance limit. PMA has several digital and analog components adding to the complexity of proper feature modeling in testbench so that they are aligned to the design expectation. The channel between the transmitter and receiver circuitry is lossy and can attenuate the signal due to resistive loss and result in Inter Symbol Interference (ISI) caused by bandwidth limitation. This could inturn mean data loss and bit errors at the receiving end as the Rx data comparator thresholds will not be enough to detect the incoming pattern. This could result in a distorted eye with reduced eye width.

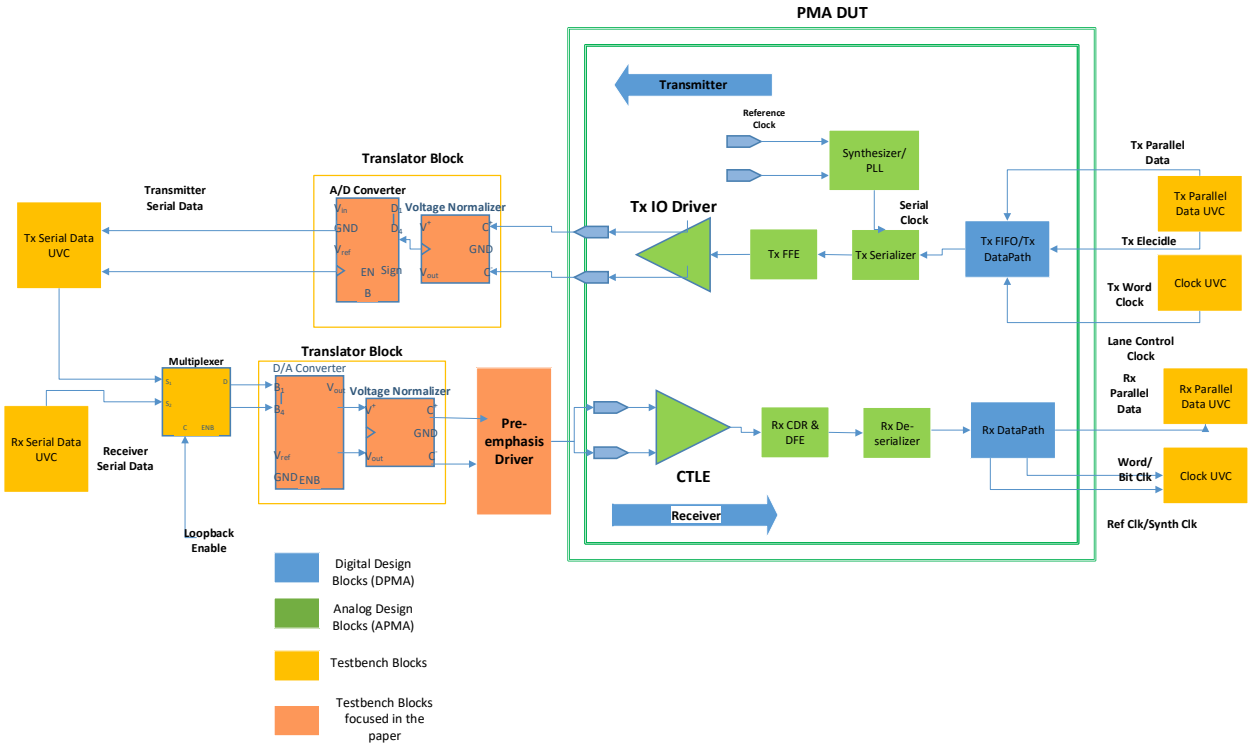

Fig.1 IP Verification framework of PMA

The Fig. 1 shows the block diagram of a conventional PMA Transmitter (Tx) and Receiver (Rx) data paths along with the testbench components around them. The testbench Tx components connect to the PMA Tx parallel interface to send the Tx parallel data and electrical idle information and at the serial interface to observe the serial differential outputs. In the transmitter path, the parallel data goes through the Tx FIFO in digital PMA (DPMA) which mangles the data and sends the appropriate word to the analog PMA (APMA). APMA has a set of serializer units which are architected to make the design more compact and ease timing closures. The mangling is done to send the parallel-to-serial (P2S) data to APMA in such a way that the serializers in the Tx path of APMA can decode the data efficiently. The P2S data feeds the serializer, the Tx Feed Forward Equalizer (FFE) and the Tx IO driver in the APMA. The IO driver is responsible for maintaining the output serial data at an idle state using the electrical idle signal, till the transmitted data pattern is received from DPMA. It also adds in the electrical information like the pre-emphasis voltage levels and converts the edge and data comparator output symbols to the final Tx serial data.

The Rx serial data can come from the Rx Universal Verification Component (UVC) or can be externally looped back from the Tx serial pins, if loopback is enabled. The Rx path receives the serial information through pads and once the clock data recovery (CDR) unit locks to the incoming data, its de-serialized and sent to the DPMA and finally to the output interface post data mangling/un-mangling. Prior to sending the data to CDR, it's reconstructed and equalized using continuous time linear equalizer (CTLE).
Equalization filters can be implemented in discrete-time or continuous-time at the Tx and/or Rx [1]. In this paper we focus on the verification challenges in using discrete time implementation at the Tx like transmitter pre-emphasis and continuous-time implementation at the Rx like Continuous Time Linear Equalizer (CTLE). Fig.2 shows a typical channel response which attenuates the data at higher frequency. A CTLE is a one tap continuous-time circuit with high-frequency gain boosting, whose transfer function can flatten the channel response, as shown in Fig. 2.Tx pre-emphasis will be discussed in the next subsection.

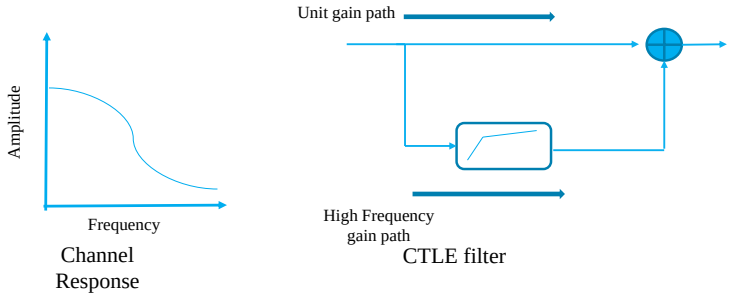

Fig.2: Channel response and corresponding CTLE implementation

It is a common practice to develop behavioral models for the analog components in-line with the analog schematics and use them for most of the pre-silicon verification. The models however, can chose the level of abstraction from being accurately modeled (netlisted down to leaf level) with all electrical information present or being an abstract model with only the functional aspects present. The possibility of having a bug free design depends largely on how close the behavioral models are to the real schematics and how well is all the electrical and functional information modelled. To represent the inherent analog properties in the behavioral models, the interface and the internal variables use "real" data-type. The flow recommended in this paper renders an effective way of re-using the Tx and Rx data transactors and checkers across different analog models by incorporating a translator block to handle the data types properly.

A. Voltage translator modules

In the beginning of the verification cycle the data transcators and end-to-end checkers can be developed to work with the abstracted analog models focusing on parallel data generation for various Built-In-Self Test(BIST) modes and data widths as per the specification. As the analog behavioral models become more accurate, the testbench end-to-end checkers and data transcators need to handle the "real" data type along with handling equalizer characteristics.

The prime functionality of translator module is to convert the "real" type data to "logic" type and vice versa, taking Tx equalization characteristics and the CTLE and DFE tap gain values into account. These are instantiated between the DUT and the serial data UVC as shown in Fig.1. The translator function is slightly different between the Tx and Rx datapath as can be seen in the flow chart in Fig.3. The translator module has two basic sub blocks: voltage normalizer and an analog-to-digital or a digital-to-analog converter.

1. Translator modules on Tx path: The Tx pad voltage values vary based on the Tx pre emphasis levels and the peak to peak voltage settings. The translator block normalizes the pad voltage and decodes it to determine the final data bit. The decoding is based on the pre-cursor, post cursor and the transition bit amplitudes of the IO driver and the type of data encoding (viz. Non-ReturnTo-Zero (NRZ), Pulse Amplitude Modulation (PAM4) etc.).
2. Translator modules on Rx path: On the Rx path, the input voltage amplitude to the translator block differs based on loopback enable. If the data is not being looped back the Serial Data UVC sends the 1/0 bit stream to the translator module which is then converted into voltage values based on the CTLE analog-to-digital converter thresholds. If the data is getting looped back, the incoming Tx pad voltage swing has to be normalized to cater to the Rx CTLE comparators so that they can decode the incoming pattern correctly.

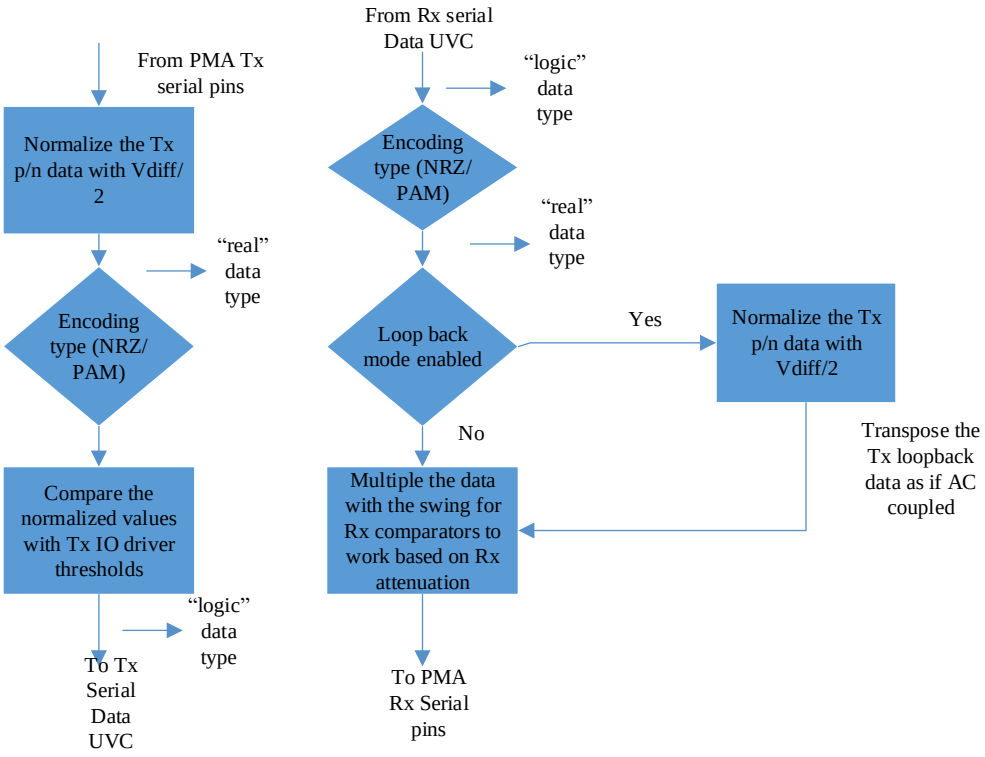

Fig.3 Tx and Rx Translator function flow chart

The translator modules were implemented using UVM callbacks and took the following parameters from UVM configuration database, making them reusable across future IP variants:

1. The Tx IO driver voltage levels which determine the swing at the Tx serial pins
2. Rx CTLE attenuation modelling and the ADC's thresholds as the incoming data has to be scaled before sending to CDR
3. Data encoding type: NRZ/PAM4
4. Loop back mode enable to determine if the data has to be normalized before sending to CTLE

III. PRE-EMPHASIS DRIVER

When talking about PCIe in specific, the channels have large attenuation at high frequencies and hence result in spreading of the transmitted signal over multiple signals, generating Inter Symbol Interferences (ISI) and bit errors at the receiver [2]. This effect can be mitigated by adding Tx equalization (also known as Tx Pre-emphasis) using a configurable FIR filter as shown in Fig.4. Pre-emphasis is a way to boost only the signal's high-frequency components, while leaving the low-frequency components in their original state as explained in [3]. In PMA, the Tx IO driver implements the pre-emphasis and the levels can be dynamically programmed, depending on the number of FIR filter stages. Here a 3 stage FIR filter with coefficients C_{-1}, C_0 and C_{+1} is shown. The coefficient are programmable and determine the voltage levels which the Tx output will generate. Hence with pre-emphasis enabled, the translator block has to handle 4 voltage levels (considering NRZ encoding with pre-cursor and post-cursor enabled for pre-emphasis).

Taking the FIR filter in Fig.3 for example, the output voltage can be derived as:

$$V_{outp} = |C_{-1}| * x[n-1] + |C_0| * x[n] + |C_0| * x[n+1] \quad (1)$$

On the Tx path, the translator blocks explained in the section above can handle the pre-emphasized data and convert them into "logic" variables for the transactors and checkers to work on. However, the Rx path becomes bit more complicated as the voltage value now has to scale in accordance with the CTLE comparator range, especially when the Tx pre-emphasized data is being looped back.

This is handled by the pre-emphasis driver on the Rx path. The driver gets the pre-emphasis level (input to PMA) and the output from the Rx translator blocks. The pre-emphasis levels are ideally set by PCS and hence can be constrained randomized in the UVM testbench and set as part of the configuration database. Table I shows the decoding of each pre-emphasis voltage value based on the current(x[n]), previous(x[n-1]) and the next(x[n+1]) serial bits.

To better explain the algorithm here, we will be selecting values for the coefficients based on the PCIe protocol defined constraints [2]. For a Tx voltage swing of 0 to 0.9V $V_{txswing}$ = 0.45, let the values be C_0 = 0.4285, C_{-1} = -0.242, C_{+1} = -0.242. With no pre-emphasis the Rx translator output for a CTLE Rx range of V_{rxn} to V_{rxp} (-0.171 to 0.171 for the example here) can be represented by (2). This directly corresponds to the Rx translator function from the flowchart in Fig.3. t_{xp} is the Tx output voltage .

$$rx_{xlator_p} = \frac{(tx_p - Vtxswing)}{Vtxswing} * Vrx_p \quad (2)$$

The highlighted value in column 1 of the table below is the serialized pattern being sent out from Tx and the intended input to the APMA Rx serial interface, post the pre-emphasis driver. It can be seen that a simple comparator is not enough to decode the final normalized output as a value of -0.131 should be decoded as 1 whereas +0.131 should be decoded as 0. The pre-emphasis driver does the intended conversion using a twostep process:

1. Run a reverse function of scaling and normalizing done in the Rx translator block. This is done using (3):

$$rx_{premph_p} = \frac{rx_{xlator_p}}{Vrx_p} * Vtxswing + Vtxswing \quad (3)$$

2. If the output from step 1 is a factor of C_0, then the Rx output bit value has to be 1. This output is used to send "real" data to APMA based on the expected V_{rxp} and V_{rxn} values. A sample code snippet for these 2 steps is shown in Fig 5. The pre-emphasis level, Tx voltage swing and Rx p/n values can be passed using uvm_config_db or can be preset in the virtual interface. V_{rxp} will be treated as bit "1" and V_{rxp} as bit "0" and hence the Rx data will match the Tx data from column 1.

Table I: Truth table for pre-emphasized Tx output and scaled Rx input

X[n-1],X[n], X[n+1]	Pre-emphasized Tx serial output	Normalized Voltage on Rx input as per equation 1 (b) (to pre-emphasis driver)	Pre-emphasis driver output based on CTLE comparator threshold
000	0	-0.171	V_{rxn} (-0.171)
001	C_{+1} = 0.242	-0.0789	V_{rxn} (-0.171)
010	C_0 = 0.4285	-0.13161	V_{rxp} (0.171)
011	$C_0 + C_{+1}$ = 0.670	0.0789	V_{rxp} (0.171)
100	C_{-1} = 0.242	-0.0789	V_{rxn} (-0.171)
101	$C_{-1} + C_{+1}$ = 0.484	0.013161	V_{rxn} (-0.171)
110	$C_{-1} + C_0$ = 0.670	0.0789	V_{rxp} (0.171)
111	$C_{-1} + C_0 + C_{+1}$ = 0.9285	0.171	V_{rxp} (0.171)

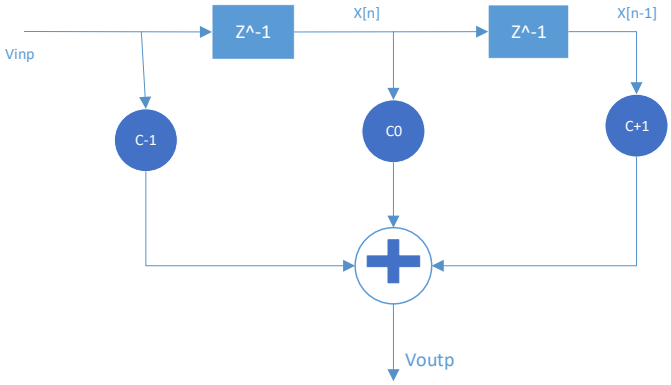

Fig.4 Tx Pre-emphasis 3-stage FIR filter

```
begin
            `ifdef APMA_AMS
            forever begin
                @(vif.rx_xlator_p or vif.rx_xlator_n);
                //Reverse function of scaling and normalizing done as under
                tx_pre_emp_p = ((rx_xlator_p/pma_cfg.apma_tx_volt_norm)*(pma_cfg.apma_rx_volt_p))
                             + (pma_cfg.apma_rx_volt_p);
                tx_pre_emp_n = ((rx_xlator_n/pma_cfg.apma_tx_volt_norm)*(pma_cfg.apma_rx_volt_p))
                             + (pma_cfg.apma_rx_volt_p);
                //If the tx_pre_emp value is a factor of C0, output is 1
                if((tx_pre_emp_p % vif.C0) == 0) begin
                    vif_rx_p = pma_cfg.apma_rx_volt_p;
                end
                else begin
                    vif_rx_p = pma_cfg.apma_rx_volt_n;
                end
                if((tx_pre_emp_n % vif.C0) == 0) begin
                    vif_rx_n = pma_cfg.apma_rx_volt_n;
                end
                else begin
                    vif_rx_n = pma_cfg.apma_rx_volt_p;
                end
            end
            `endif
    end
```

Fig.5 Sample implementation of pre-emphasis driver

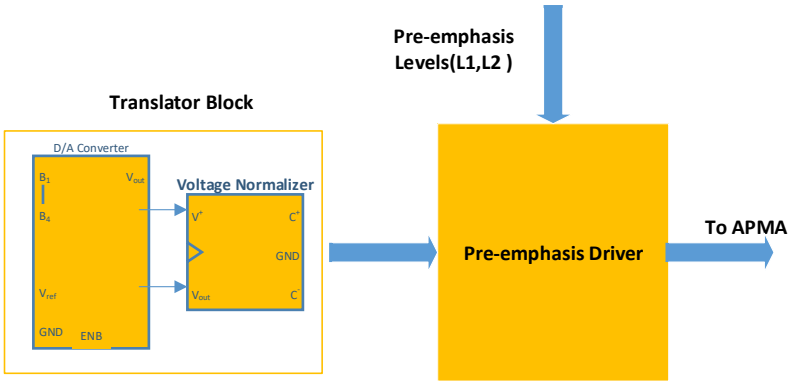

Fig.6 Pre-emphasized output sent to the APMA using pre-emphasis driver

IV. LAYERED UVM "ADAPTER" SEQUENCE

Using UVM methodology enables creating layered sequences using virtual sequences and sequencers. But for IPs like PMA, which can be configured to have multiple data lanes (1, 2, 4, 8 and 16 being most common

configurations), having a central layer for transaction ordering and control helps in generating effective lane specific traffic.

Each transaction takes in the direction (Tx or Rx) as well as the lane number as a parameter which can be set from the test specific virtual sequence. The sequence library has certain data sequences which can generate transactions for each lane with a pre-defined amount of delay between them. This delay is seen by the parallel Data UVC which brings the lane out of electrical idle accordingly. Similar is the case when we send data on the Rx pads from the Serial Data UVC. The data integrity is checked by doing a bitwise comparison between the serial and the parallel data for each lane.

Fig.7 shows the proposed sequence architecture for a testbench using the adapter sequence which runs on the control sequencer and converts the jtag or memory bus transactions into lane specific transactions viz. request/response, control/data and clock/debug. The packet created by every test goes through this layer and becomes an abstract version of the transaction. Fig.10 shows a high level overview of how the complete verification environment with adapter sequence and translator blocks look. Here the sub-sequencers represent all the sequencers shown in Fig.7. A sample code snippet of adapter sequence handling a force power-up (force_pup) request to the PLLs is shown in Fig.9. Some of the advantages of adopting such an architecture are as under:

a. Helps in streamlining the checkers based on the control or data transaction, per lane per feature.
b. Gives additional control on determining the test stimulus injection level: pin or register overrides
c. Having one central control of various random transaction field results in effective coverage collection and consistent test-sequence interface

The coverage models focused on the feature coverage (like Rx margining offsets, pre-emphasis levels etc.) and made use of the adapter sequence to sample the transaction fields being sent on each lane. Additionally, the MSV control knobs present in the analog models were tuned to get a better coverage for analog properties like DFE edge and data offsets, calibration offsets for voltage controlled oscillators etc.

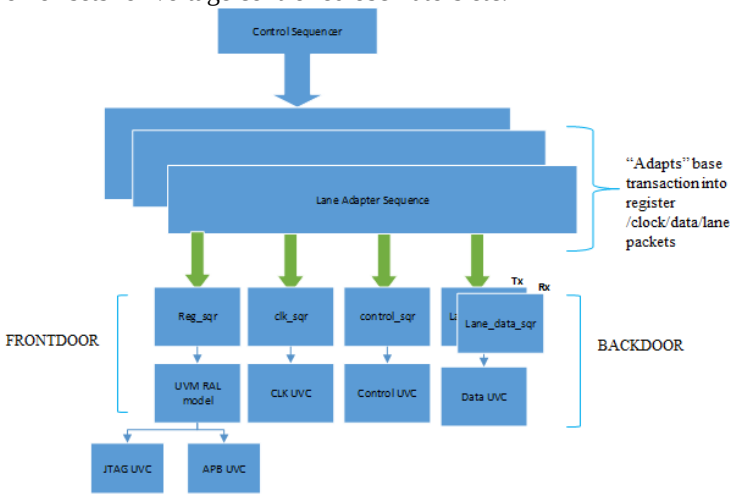

Fig.7: Layered UVM Adapter sequence

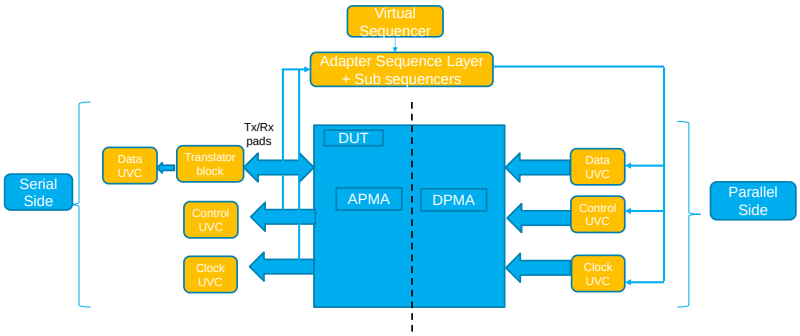

Fig.8: Top level PMA verification framework with Adapter Sequence and Translator Block

```
class pma_lane_adapter_seq extends uvm_sequence#(pma_transaction);
  pma_transaction pma_req;
  pma_transaction req_copy;
  parallel_data_cfg par_drv_cfg;
  `uvm_object_utils(pma_lane_adapter_seq)
  `uvm_declare_p_sequencer(pma_lane_sequencer)

  // All Register Sequence's object created
  pma_init_state_reg_seq              init_state_reg_seq;
  pma_set_spec_reg_seq                set_spec_reg_seq;
  pma_configure_lane_reg_seq      configure_lane_reg_seq;
//<some common functions skipped in the snippet>
  virtual task body();
    forever begin
      p_sequencer.get_next_item(pma_req);
      if (pma_req.primitive_layer == pma_transaction::CTRL) begin
        drive_ctrl();
      end else if (pma_req.primitive_layer == pma_transaction::DATA) begin
        drive_data();
      end else begin
        drive_debug();
      end
      p_sequencer.item_done();
    end
  endtask : body

  task drive_ctrl();
    case (pma_req.primitive_name)
      pma_transaction::FORCE_PUP :
        begin
          if (pma_req.pma_cfg.ctrl_path == UVM_BACKDOOR) begin
            $cast(req_copy, pma_req.clone());
            req_copy.set_item_context(this, p_sequencer.cmn_ctrl_seqr);
            start_item(req_copy);
            finish_item(req_copy);
           return_response = 0;
          end else begin
            `uvm_do_on_with(force_pup_reg_seq,
                    p_sequencer.reg_seqr, { block == pma_req.primitive_target; enable == pma_req.power_on; });
          end
        end
      endtask
```

Virtual sequence sets the primitive layer type based on the feature being validated

Ctrl_path is chosen randomly for features which can be configured through pin (UVM_BACKDOOR) as well as registers

Fig.9: Sample UVM Adapter sequence code

V. Replacing Sub-Blocks with SPICE netlist

The flow presented in this paper can be seamlessly extended to co-simulation (mixed signal simulation) where APMA submodules are replaced by the SPICE netlist. The paper does not talk about details of enabling this flow as

this is a work in progress, but listed below are few common steps and consideration to be done to enable this (the exact command/steps are simulator specific).

1. Create a configuration file specifying the netlist path, module name to be used in SPICE view and the relevant include files. Simulator specific commands like use_spice can be called to replace a module with its netlist, per instantiation.
2. It is important to model the A2D and D2A thresholds for proper "real" to "logic" conversions in the simulations. Tools have commands to specify the low, mid and high thresholds.
3. Provide correct settings for interface elements which need to model electrical to real or vice versa conversions.

VI. RESULTS

The translator blocks along with the pre-emphasis driver rendered a reusable and configurable UVM verification environment, across various stages of the analog model development, as we moved from behavioral to accurate analog models and eventually to the analog schematics (using co-simulation). The Fig.10 shows an example of how having a configurable translator and pre-emphasis model helped in modeling the serial data correctly in a loop back mode.

- The first group is the pre-emphasized output from APMA Tx serial pads which has multiple levels based on the input data transition.
- The second group shows the Tx translated output from transactor which predicts the 0s and 1s correctly and uses for data comparisons and checks.
- Third group shows the output of Rx translator blocks which uses the Tx pad data and scales based on translator Rx path algorithm
- Fourth group shows the final output from pre-emphasis driver which corrects the swing based on CTLE requirements (0 translated to -0.171 and 1 translated to 0.171 in the example shown)

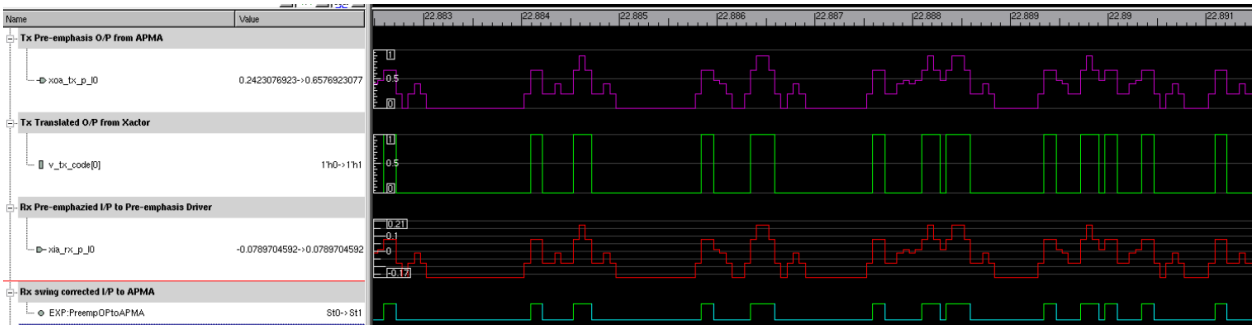

Fig.10: Layered UVM Adapter sequence

Table II summarizes the benefits of using a UVM adapter sequence in a multi-lane IP like PMA.

Table II: Key benefits of using adapter sequence for an IP with multiple data lane.

Features of Adapter Sequence	**Remarks**
Lane specific traffic generation and routing	Sending traffic across lanes, wait for CDR lock and checking lane-to-lane interactions (skew, latency etc.)
Demarcation between control, data and debug transfers	Setting up beacon, sending burst, checking clocks or bit errors
Centralized coverage sampling across model types	Coverage packet can be sent to coverage class or can be sampled here directly for control and data transfers
Passing packet to scoreboard/monitors	Each transaction from register or control sequencer comes to lane adapter and hence the packet contents can be used to add checks in scoreboard or monitor specification adherence

VII. CONCLUSION

The approaches mentioned in this paper can render build a verification framework which can model critical design features and help in focused verification of the same. Building a configurable translator module and the UVM layered structure can help reuse the blocks across IP variants and also across various analog models.

The analog models themselves go through a formal equivalence check with the schematics at the sub-block level. The flow runs static checks and catches bugs with regards to connectivity and basic behavior in the models. This approach helps reduce the gap between the real schematics and the model being used for the majority of pre-silicon verification. When these models get integrated and interact with DPMA at different data rates and widths, other critical bugs get caught in the simulations with the help of translator blocks and checkers, as discussed in this paper. The higher simulation time and limited debug capabilities with mixed signal simulation (discussed in section V) makes it even more critical to have robust verification components like the pre-emphasis driver.

Using the adapter sequences rendered creating a vast set of sequence library for each data and lane configurations as per the specification. Some of the additional benefits with this flow were:

- Catching critical bugs in the analog accurate model with regards to electrical idle modelling and CTLE tunings. Effective modeling of pre-emphasis and translator block helped catch critical bugs in Tx IO driver and DPMA –APMA interactions.
- Consistent code coverage across the models
- All the testcases were reused across behavioral and accurately netlisted analog models

REFERENCES

[1] Rangel-Patiño, Francisco E. “Transmitter and Receiver Equalizers Optimization Methodologies for High-Speed”, 2018-07
[2] Logic truit white paper on PCIe equalization
[3] Signal Integrity Engineer's Companion, A: Real-Time Test and Measurement and Design Simulation
[4] SerDes system CTLE Basics: By John Baprawski, March 22 2012