2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

A UVM Testbench for Checking the Global Convergence of Analog/Mixed-Signal Systems: An Adaptive Decision-Feedback Equalizer Example

Jaeha Kim, Seoul National University, Seoul, Korea (jaeha@snu.ac.kr)

***Abstract*— A UVM testbench capable of verifying the global convergence property of an analog/mixed-signal system is presented. For example, a sign-sign LMS adaptation algorithm for a decision-feedback equalizer (DFE) may converge to a false final state depending on the initial state. To detect the existence of such false final states, the testbench launches a sequence of trial runs, each starting from a random, unvisited initial state, until all possible states of the system are tried or traversed, or a problematic initial state is found. The simulation is run entirely in SystemVerilog by modeling the analog components of the high-speed wireline transceiver using the XMODEL primitives. To generate a sequence of trial runs based on the previous results and evaluate the termination conditions, the testbench utilizes a shared state coverage database and a global UVM event. The experimental results show that the testbench swiftly uncovers the false final states caused by high channel loss or insufficient constraints, and successfully confirms the global convergence of the adaptation loop when no such issues exist.**

Keywords—analog/mixed-signal verification; universal verification methodology (UVM); SystemVerilog; XMODEL; global convergence; decision-feedback equalizer (DFE).

I. INTRODUCTION

High-speed wireline transceivers are essential building blocks that enable high-bandwidth data communication between chips, boards, and systems. They have widespread use in computers, displays, network routers, data centers, automobiles, mobile devices, and Internet-of-Thing (IoT) devices. Since the exact characteristics of the communication channels are not known at the design time, their equalizers, which are needed to compensate for frequency-dependent loss in the channels, are typically made adjustable and often include self-calibrating adaptation loops. One key concern for such an adaptive equalizer is, however, whether the loop always converges to an optimal setting that can compensate for the channel loss. This paper aims to build a UVM testbench that can verify the global convergence property of a digital adaptation controller for a decision-feedback equalizer (DFE) of a high-speed wireline receiver. In other words, we aim to compose a testbench that can check if the DFE adaptation algorithm can settle to the desired optimally equalized state, starting from an arbitrary initial state.

Building such a UVM testbench presents several challenges. The first challenge is the need to simulate analog circuits (e.g., the high-speed transceiver circuits including the DFE) alongside digital systems (e.g., the DFE adaptation controller) on a platform that supports UVM, which is SystemVerilog. This requires the capability to model and simulate analog circuits in SystemVerilog, using methods such as Real-Number Modeling or XMODEL. The second challenge is the necessity to test every possible initial state of the DFE and verify their convergence to the same equalized state. For a 4-tap DFE, with each tap coefficient having a 6-bit value, a brute-force approach would require 2^{24} (≈16.8 million) trials. To address this, the presented UVM testbench leverages the fact that each trial run, starting from a given initial state, traverses through multiple intermediate states before reaching the final state. This means that one trial run can verify more than one initial state. To accomplish this, the testbench needs to keep the records of all previously visited states and initiate new trials with unvisited states until all possible states are marked as visited, or a problematic initial state is found.

The rest of the paper is organized as follows. Section II reviews the background of DFE and its sign-sign least-mean-squares (LMS) adaptation algorithm. Section III describes how these can be modeled in SystemVerilog using XMODEL. Section IV presents the UVM testbench that verifies the global convergence property of the adaptive DFE. Finally, Section V discusses the experimental results obtained with the UVM testbench, and Section VI concludes the paper.

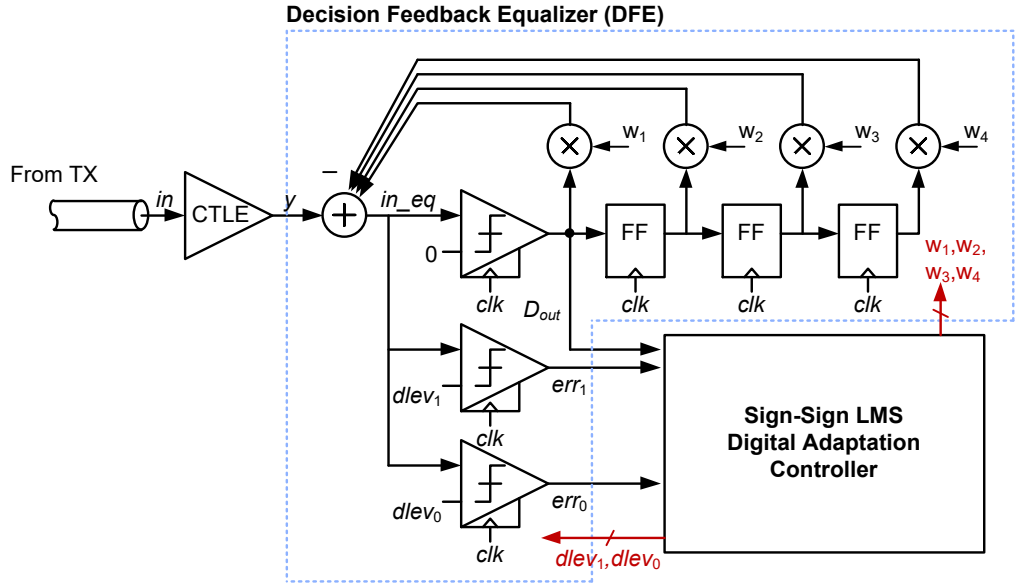

Figure 1. A high-speed wireline receiver with an adaptive decision-feedback equalizer (DFE), employing a sign-sign LMS adaptation algorithm.

II. Sign-Sign Least-Mean-Squares (LMS) Adaptation for Decision-Feedback Equalizers

Fig. 1 illustrates the front-end part of a high-speed wireline receiver with a 4-tap adaptive DFE. The receiver produces the output data $D_{out}[n]$ recovered from the incoming signal *in* by first filtering it with a continuous-time linear equalizer (CTLE), second subtracting the expected inter-symbol interference (ISI) caused by the previously-transmitted data $D_{out}[n\text{-}1]$, $D_{out}[n\text{-}2]$, $D_{out}[n\text{-}3]$, and $D_{out}[n\text{-}4]$, and finally slicing the result [1]. Here, the ISI is computed as a weight sum of $w_1 \cdot D_{out}[n\text{-}1] + w_2 \cdot D_{out}[n\text{-}2] + w_3 \cdot D_{out}[n\text{-}3] + w_4 \cdot D_{out}[n\text{-}4]$, where w_1, w_2, w_3, and w_4 are called the tap coefficients of the DFE. To support a wide range of communication channels with diverse characteristics, most advanced wireline receivers have the capability of adapting the DFE tap coefficients automatically, typically via a simplified form of the least-mean-squares (LMS) algorithm, called the sign-sign LMS algorithm [2]. The algorithm updates each tap coefficient value w_k using the update formula below:

$$\begin{aligned} w_{k,i+1} &= w_{k,i} - \frac{\mu}{2} \cdot \frac{\partial e^2[n]}{\partial w_k} = w_{k,i} - \mu \cdot e[n] \cdot y[n-k] \\ &\approx w_{k,i} - \mu \cdot sign(e[n]) \cdot sign(y[n-k]) \approx w_{k,i} - \mu \cdot sign(e[n]) \cdot D_{out}[n-k] \end{aligned} \quad (1)$$

where $y[n]$ is the actual input to the DFE sampled at time n, $e[n]$ is the difference between $y[n]$ and the desired input level ($=d_{lev} \cdot D_{out}[n]$), and μ is a scaling factor controlling the amount of change made with each observation. Basically, the algorithm adjusts the tap coefficients w_k's in a direction that can reduce the mean squared value of the error $e^2[n]$. To simplify the implementation, the sign-sign LMS algorithm detects only the polarity of the change necessary to each w_k, by approximating $sign(y[n\text{-}k])$ with $D_{out}[n\text{-}k]$.

However, it is this approximation that can cause the global convergence issues. Note that $y[n\text{-}k]$ is the input to the DFE before the ISI is subtracted, whereas $D_{out}[n\text{-}k]$ is the data decision made on the signal after the ISI is subtracted. When the subtracted ISI term is small, the approximation holds well. However, when it is not, the adaptation can progress in a wrong direction, leading to a false convergence, i.e., the state that the DFE cannot properly recover the correct data. For example, the channel may have significant loss at high frequencies, demanding large DFE tap coefficient values. Or, the DFE tap coefficients may start with large values for some reason. In these cases, the computed ISI term can become large enough to cause global convergence failures.

III. MODELING OF HIGH-SPEED WIRELINE RECEIVER WITH ADAPTIVE DECISION-FEEDBACK EQUALIZER

Fig. 2 shows the overall block diagram of an example 16-Gb/s high-speed wireline transceiver model, including the adaptive DFE and its sign-sign LMS adaptation loop. This model is largely similar to the one presented in [3]. On the transmitter side, a charge-pump phase-locked loop (*TX-PLL*) generates a 16-GHz clock (*tx_clk*) from a 2-GHz reference (*tx_ref_clk*), and a differential current-mode transmitter with 1-tap de-emphasis (*TX-EQ*) drives the input data stream (*Din*) onto a pair of transmission lines with termination loads. On the receiver side, a continuous-time linear equalizer stage (*RX-CTLE*) followed by a 4-tap DFE stage perform equalization on the received signal before the data sampler makes decisions to produce the output data (*Dout*). Additionally, a phase-interpolator-based clock-and-data recovery loop (*RX-CDR*) recovers the clock (*rx_clk*) that triggers the data and edge samplers, as guided by a bang-bang phase detector (*PD*) and a digital loop filter (*LF*).

The analog components of this high-speed wireline transceiver are modeled in SystemVerilog using the primitives provided by XMODEL from Scientific Analog [4]. XMODEL introduces a signal type called *xreal*, which expresses a continuously-varying analog signal using an equation rather than a set of sample points. This allows XMODEL to perform truly event-driven simulation of analog circuits at fast speeds without sacrificing accuracy [5]. XMODEL has extended this event-driven simulation approach even to circuit-level models described as a network of circuit elements, including passive (e.g. resistors), active (e.g. transistors), and distributed elements (e.g. transmission lines). XMODEL has another signal type called *xbit*, which expresses digital signals with accurate timing, not limited to the SystemVerilog simulator's timestep.

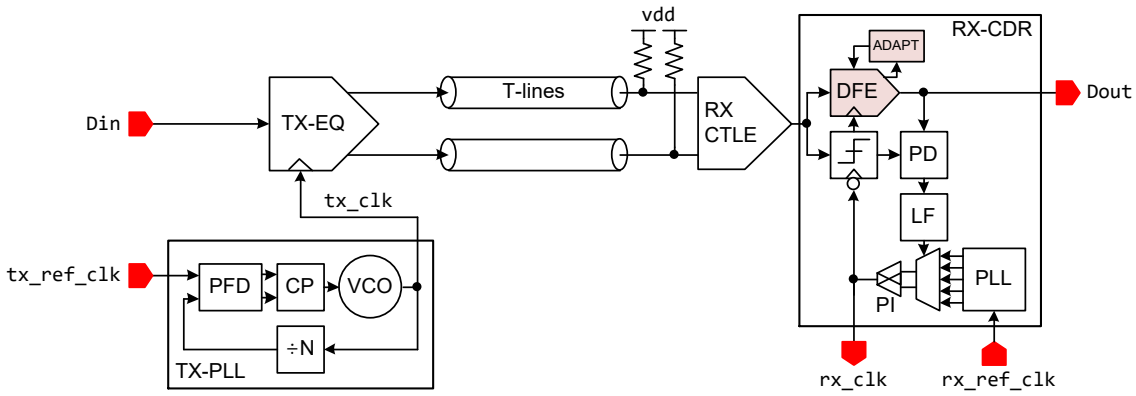

Figure 2. The overall block diagram of a high-speed wireline transceiver model with adaptive DFE.

Fig. 3 shows the detailed model of the 4-tap adaptive DFE and its sign-sign LMS adaptation loop in schematic form. All the block symbols, except for the green one labeled *eq_adapt*, are XMODEL primitives, such as *add*, *compare*, *dac*, *filter_disc_var*, and *inv_xbit*. For example, the *compare* primitive describes a clocked comparator, and *filter_disc_var* primitive describes a discrete-time filter with variable numerator and denominator coefficients in its z-domain transfer function. The other primitives perform the functions suggested by their names.

The model of this adaptive DFE receiver closely resembles that in Fig. 1, with a difference that it explicitly shows the crossings between the analog and digital domains. For the data sampler with DFE, the *compare* primitive converts the sampled analog signal *dfe_out* to a digital output *data* and the *dac* primitive on the feedback path converts *data* back to an analog signal *fb_in*, so that it can be subtracted from the incoming signal after being filtered by the *filter_disc_var* primitive. On the other hand, the error detector consists of two *compare* primitives that measure the polarity of the error between the equalized signal *dfe_out* and the desired levels *dlev*1 and *dlev*0. The sign-sign LMS adaptation is performed by a digital controller named *eq_adapt* based on the received data and error polarity values. The controller produces the desired level and four DFE tap coefficients in 6-bit digital values, which are then converted to analog values via a set of *dac* primitives.

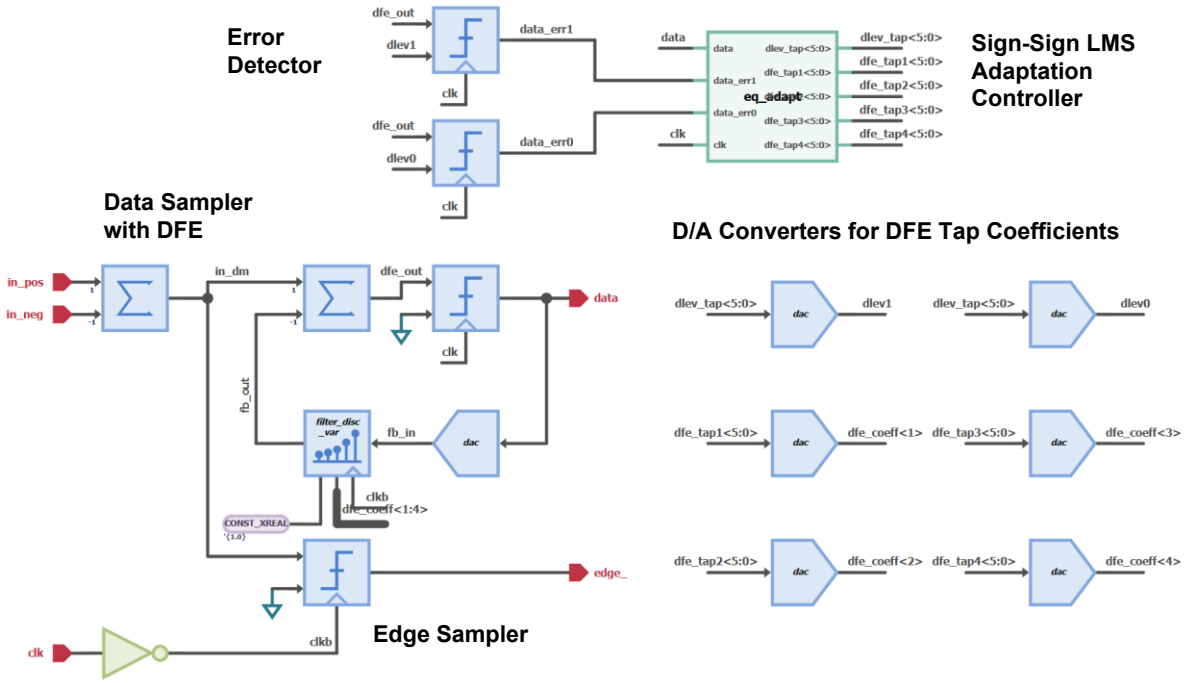

Figure 3. The model of the 4-tap adaptive DFE receiver with a sign-sign LMS adaptation loop.

Fig. 4 lists the Verilog code describing the sign-sign LMS adaptation controller (*eq_adapt*). It basically computes Eq. (1) to update the DFE tap coefficient values based on the product of the error polarity and the data. The time shift between the error e[n] and data applied before computing the product depends on the tap position. For example, w_1 is updated based on the product of $sign(e[n])$ and $D_{out}[n\text{-}1]$ and w_4 is updated based on the product of $sign(e[n])$ and $D_{out}[n\text{-}4]$. On the other hand, the desired level *dlev* is increased when the error polarity indicates that the equalized signal has the larger swing than *dlev* and vice versa. To avoid excessive dithering at the locked states, one update decision is made after accumulating 255 observations.

The code listed in Fig. 4 also contains additional components that facilitate verification by the UVM testbench to be presented shortly. First, it includes a task named *init*(), which initializes the DFE tap coefficient values and resets the internal accumulator states. Second, it triggers a SystemVerilog event named *updated* when the controller makes changes to the DFE tap coefficient values. Using these, the UVM testbench sets a new initial state and monitors the progression of the state over time.

```
module eq_adapt (
    output reg [5:0] dlev_tap, dfe_tap1, dfe_tap2, dfe_tap3, dfe_tap4,
    input data, data_err1, data_err0,
    input clk
);

reg signed [8:0] acc [0:4];
reg [7:0] count;
reg [1:4] d_buf;
reg data_err;
event updated;
int k;

always @(posedge clk) begin
    d_buf[1:4] <= {data, d_buf[1:3]};
end

always @(posedge clk) begin
    // sign-sign LMS operation
    data_err = (data) ? data_err1 : data_err0;
    acc[0] += ((data) ? data_err1 : ~data_err0) ? +1 : -1;
    acc[1] += (data_err ^ d_buf[1]) ? -1 : +1;
    acc[2] += (data_err ^ d_buf[2]) ? -1 : +1;
    acc[3] += (data_err ^ d_buf[3]) ? -1 : +1;
    acc[4] += (data_err ^ d_buf[4]) ? -1 : +1;
    count += 1;

    if (count == 255) begin
        // update DFE tap coefficients
        if (acc[0] > 8) dlev_tap <= (dlev_tap < 6'd63) ? dlev_tap + 1 : 6'd63;
        else if (acc[0] < -8) dlev_tap <= (dlev_tap > 6'd0) ? dlev_tap - 1 : 6'd0;
        if (acc[1] > 8) dfe_tap1 <= (dfe_tap1 < 6'd63) ? dfe_tap1 + 1 : 6'd63;
        else if (acc[1] < -8) dfe_tap1 <= (dfe_tap1 > 6'd0) ? dfe_tap1 - 1 : 6'd0;
        if (acc[2] > 8) dfe_tap2 <= (dfe_tap2 < 6'd63) ? dfe_tap2 + 1 : 6'd63;
        else if (acc[2] < -8) dfe_tap2 <= (dfe_tap2 > 6'd0) ? dfe_tap2 - 1 : 6'd0;
        if (acc[3] > 8) dfe_tap3 <= (dfe_tap3 < 6'd63) ? dfe_tap3 + 1 : 6'd63;
        else if (acc[3] < -8) dfe_tap3 <= (dfe_tap3 > 6'd0) ? dfe_tap3 - 1 : 6'd0;
        if (acc[4] > 8) dfe_tap4 <= (dfe_tap4 < 6'd63) ? dfe_tap4 + 1 : 6'd63;
        else if (acc[4] < -8) dfe_tap4 <= (dfe_tap4 > 6'd0) ? dfe_tap4 - 1 : 6'd0;

        for (k=0; k<=4; k++) acc[k] = 0;
        count = 0;

        // flag that the DFE tap coefficients are updated
        -> updated;
    end
end

// task that sets the initial values of the DFE tap coefficients
task init(
    input [5:0] init_dlev, init_dfe1, init_dfe2, init_dfe3, init_dfe4
);
    dlev_tap = init_dlev;
    dfe_tap1 = init_dfe1;  dfe_tap2 = init_dfe2;
    dfe_tap3 = init_dfe3;  dfe_tap4 = init_dfe4;

    d_buf = 0;
    for (k=0; k<=4; k++) acc[k] = 0;
    count = 0;
endtask

endmodule
```

Figure 4. The Verilog model of the sign-sign LMS adaptation controller for a 4-tap DFE receiver.

IV. UVM Testbench Verifying the Global Convergence of DFE Adaptation

The objective is to verify that the DFE tap coefficients consistently converge to the same values through the sign-sign LMS adaptation loop regardless of their initial values. To achieve this, the testbench needs to launch a series of trial runs, each starting from a different initial state—that is, a different set of DFE tap coefficient values—and check if they all converge to the same final state, i.e., the same set of tap coefficient values. During each trial run, if the adaptation loop traverses through intermediate states before reaching the final state, each of those intermediate states can be considered as a valid initial state leading to the same final state. Furthermore, some trial runs can be stopped early when they reach a state whose final state has already been verified. The verification concludes when all possible initial states have been visited or when a problematic initial state leading to a different final state is identified.

Fig. 5 illustrates the organization of the proposed UVM testbench to verify the global convergence property of the adaptive DFE. Following the approach described in [6],[7], all the analog-specific details are encapsulated within a fixture module, allowing the rest of the testbench to be built using standard UVM components. In this particular case, the stimuli and responses of the device-under-test (DUT) enclosed by the fixture module are all digital. Specifically, the sequencer and driver components provide the fixture module with the next initial tap coefficient values to try, and the monitor component observes the tap coefficient values traversed by the adaptation loop and checks if they have converged to their final values.

While one method for the sequencer component to choose a next initial state that has not been tried or traversed is to use a reactive stimulus technique [3],[8], it was not straightforward to implement it in this particular case because a single stimulus can generate multiple responses. Instead, a common database containing the state coverage information is shared among the sequencer, monitor, and scoreboard components via the UVM configuration database (*uvm_config_db*). In this approach, the monitor component updates this coverage database with the observed state values, and the sequencer component selects the next initial state by querying it. Furthermore, when the monitor component determines that a trial run has reached a new final state or one of the previously verified states with a known final state, it triggers a UVM event named *LOCKED*, which is stored in the global *uvm_event_pool*. This event allows the sequencer component to initiate a new trial run.

The following subsections provide detailed descriptions of each component within this UVM testbench.

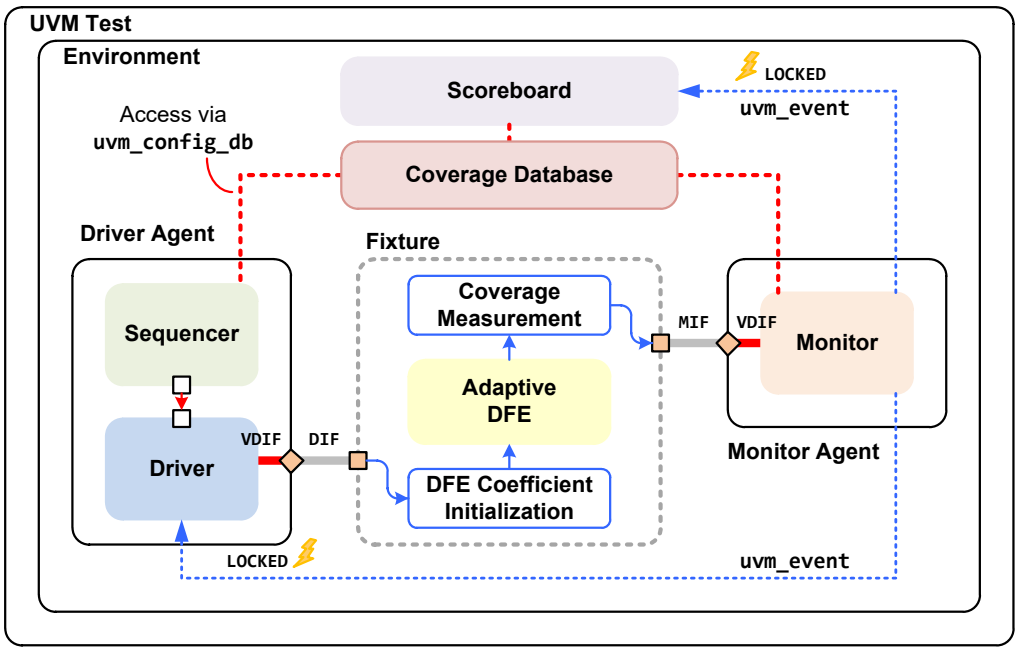

Figure 5. Conceptual diagram of the UVM testbench performing the global convergence checks on the adaptive decision-feedback equalizer (DFE) of a high-speed wireline receiver.

A. Coverage Database

Fig. 6 presents the code defining a class named *COVERAGE*, maintaining a list of previously traversed states and their corresponding final states using two member variables: *visited* and *locks*. First, *visited* is a SystemVerilog associative array mapping each 24-bit state value, comprising four 6-bit tap coefficient values, to an integer-valued index of its final state, with valid values starting from 1. States not stored in *visited* have default mapped values of 0 and are considered not visited yet. On the other hand, *locks* is a SystemVerilog queue keeping the list of final states discovered so far. Therefore, the goal of this UVM testbench is to populate *visited* with all possible initial states and verify that only one final state is registered in *locks*. The *COVERAGE* class also includes additional member variables such as *num_trials*, which tracks the number of trial runs executed so far, and *size_full*, defining the size of the array *visited* when it is full.

The *COVERAGE* class also defines a set of member functions, such as *new*(), *check_constraint*(), and *calc_coeff*(), to handle cases where the initial state space needs to be constrained. During the construction of a new instance, the *visited* array is populated with the states for which the *check_constraint*() function returns 0, with these states mapped to a value of -1. The *calc_coeff*() function helps define the state-excluding conditions within the *check_constraint*() function by converting the digital tap coefficient value to analog. For example, the *check_constraint*() function listed in Fig. 6 constrains the initial state space to the tap coefficient values satisfying $|w_1|+|w_2|+|w_3|+|w_4| \le 0.05$, $|w_1| > |w_2| > |w_3|$, and $|w_2| > |w_4|$, which occupies only 0.014% of the total state space.

An instance of the *COVERAGE* class named *CVG* is created within the top-level module *UVM_TB* and shared globally with the UVM components, using the *uvm_config_db*::*set*() and *uvm_config_db*::*get*() methods. The UVM_TB module calls *uvm_config_db*::*set*() to register the handle to the *CVG* instance in the UVM configuration database, and each UVM component calls the *uvm_config_db*::*get*() to retrieve it and access the CVG instance's contents.

```
`define SIZE_STATE        24

typedef bit [`SIZE_STATE-1:0] DATA_t;
typedef virtual IF_t VIF_t;

class COVERAGE;
    shortint visited[DATA_t];
    DATA_t locks[$];
    int num_trials = 0;
    int size_full = (1 << `SIZE_STATE);

    function new();
        DATA_t v = 0;
        for (int i=0; i<size_full; i++) begin
            if (!check_constraint(v)) visited[v] = -1;
            v++;
        end
    endfunction: new

    function int check_constraint(DATA_t value);
        real c1 = calc_coeff(value[23:18]);
        real c2 = calc_coeff(value[17:12]);
        real c3 = calc_coeff(value[11:6]);
        real c4 = calc_coeff(value[5:0]);
        return (`fabs(c1) + `fabs(c2) + `fabs(c3) + `fabs(c4) <= 0.05 &&
                `fabs(c1) > `fabs(c2) && `fabs(c2) > `fabs(c3) && `fabs(c2) > `fabs(c4));
    endfunction: check_constraint

    function real calc_coeff(bit [5:0] v);
        real scale = 0.1;
        return scale * (v*2.0/63 - 1.0);
    endfunction: calc_coeff
endclass: COVERAGE
```

Figure 6. The coverage database class for maintaining a list of traversed states and their corresponding final states.

B. *Sequencer and Driver Components*

Fig. 7 lists the code of the sequencer and driver components that launches a sequence of trial runs. The sequencer randomly selects the next initial tap coefficient values that have not been tried or traversed by finding the state value not registered in the coverage database (*CVG*) using the constrained randomization solver of SystemVerilog. The driver then feeds this value to the fixture module via the driver-side interface bus (*VDIF*) and initiates a new trial run by triggering its member event named *TRIG*. Each trial run concludes when the monitor component finds that a termination condition is met and triggers the global UVM event named *LOCKED*. The sequencer keeps launching new trial runs until the *CVG.visited* array is full or the *CVG.locks* queue has more than one entry.

```
class PACKET extends uvm_sequence_item;
    `uvm_object_utils(PACKET)

    rand DATA_t DATA;
    COVERAGE CVG;

    constraint EXCLUDE_con { !CVG.visited.exists(DATA); }
    ...
endclass: PACKET


class SEQ_EQADAPT extends uvm_sequence #(PACKET);
    `uvm_object_utils(SEQ_EQADAPT)

    COVERAGE CVG;
    PACKET PKT;
    DATA_t init_state;

    task body();
        void'(uvm_config_db #(COVERAGE)::get(null, "uvm_test_top", "Key_CVG", CVG));
        PKT = PACKET::type_id::create("PKT");
        PKT.CVG = CVG;
        while (CVG.visited.size() < CVG.size_full && CVG.locks.size() <= 1) begin: LOOP
            start_item(PKT);
            if (CVG.num_trials == 0) PKT.DATA = init_state;
            else void'(PKT.randomize());
            CVG.num_trials++;
            finish_item(PKT);
        end: LOOP
    endtask: body
    ...
endclass: SEQ_EQADAPT


class DRIVER extends uvm_driver #(PACKET);
    `uvm_component_utils(DRIVER)

    VIF_t VDIF;
    uvm_event LOCKED;
    PACKET PKT;
    ...

    function void build_phase(uvm_phase phase);
        void'(uvm_config_db #(VIF_t)::get(null, "uvm_test_top", "Key_VDIF", VDIF));
    endfunction: build_phase

    task run_phase(uvm_phase phase);
        LOCKED = uvm_event_pool::get_global("LOCKED");
        wait(!VDIF.RST);
        forever begin: LOOP
            // apply a new initial state
            seq_item_port.get_next_item(PKT);
            VDIF.DATA = PKT.DATA;
            -> VDIF.TRIG;
            `uvm_info("DRV", $sformatf("\n   | DRV #%0d: trying new initial state: %b", PKT.CVG.num_trials,
VDIF.DATA), UVM_HIGH);

            // wait until a lock is reached
            LOCKED.wait_trigger();
            #(1ns);
            seq_item_port.item_done();
        end: LOOP
    endtask: run_phase
endclass: DRIVER
```

Figure 7. The sequencer and driver components launching a sequence of trial runs with randomly selected initial tap coefficient values.

C. *Fixture Module*

The fixture module, shown in Fig. 8, instantiates the model of the high-speed wireline transceiver described in Section III, including the 4-tap DFE and its sign-sign LMS adaptation controller. It also includes the necessary instrumentations to apply new initial tap coefficient values to the adaptation controller and observe the tap coefficient values being traversed by the adaptation controller afterwards.

Specifically, when the *TRIG* event of the driver-side interface bus (DIF) is triggered, the fixture module calls the *init*() task of the *eq_adapt* module instance to set its tap coefficients to the values provided by the sequencer component (*DIF.DATA*). Additionally, when the *updated* event of the *eq_adapt* module instance is triggered, indicating a change in the tap coefficient values, the fixture module forwards the values to the monitor component via the monitor-side interface bus (*MIF*) and triggers its *TRIG* event.

```
`define DUT_EQADAPT       DUT.IRXCDR.IRXEQ.IEQADAPT

interface IF_t (input bit RST);
    DATA_t DATA;
    event TRIG;
endinterface: IF_t

module FIXTURE (IF_t DIF, IF_t MIF);
    parameter real data_freq = 16.0e9;      // data rate
    parameter real ref_freq = 2.0e9;        // RX reference clock frequency
    parameter real ref_RJ = 1e-12;          // RX reference clock jitter

    xbit ref_txclk, ref_rxclk, tx_clk, rx_clk;
    xbit Din, Dout, Dout_os;
    xreal delay_txclk, vdd;
    bit [5:0] init_dfe1, init_dfe2, init_dfe3, init_dfe4;

    // DUT instantiation
    hslink          #(.channel_noise(0.001), .rx_noise(0.001))
                    DUT (.ref_txclk, .ref_rxclk, .tx_clk, .rx_clk, .Din, .Dout, .Dout_os, .delay_txclk, .vdd);

    // clock, data, and supply sources
    clk_gen         #(.freq(ref_freq), .RJ_rms(ref_RJ)) U1 (ref_txclk);
    clk_gen         #(.freq(ref_freq), .RJ_rms(ref_RJ)) U2 (ref_rxclk);
    prbs_gen        #(.length(15)) U3 (.trig(tx_clk), .out(Din));
    dc_gen          #(.value(0.0)) U4 (delay_txclk);
    dc_gen          #(.value(1.2)) U5 (vdd);

    // interfaces with driver & monitor
    always @(DIF.TRIG) begin
        // initialize DFE coefficients
        {init_dfe1, init_dfe2, init_dfe3, init_dfe4} = DIF.DATA;
        `DUT_EQADAPT.init(
            .init_dlev(6'b010110),  // NOTE: fixing dlev at 6'b010110
            .init_dfe1(init_dfe1), .init_dfe2(init_dfe2), .init_dfe3(init_dfe3), .init_dfe4(init_dfe4)
        );
    end

    always @(`DUT_EQADAPT.updated) begin
        MIF.DATA = {`DUT_EQADAPT.dfe_tap1, `DUT_EQADAPT.dfe_tap2, `DUT_EQADAPT.dfe_tap3, `DUT_EQADAPT.dfe_tap4};
        -> MIF.TRIG;
    end

endmodule: FIXTURE
```

Figure 8. The fixture module instantiating the high-speed wireline transceiver model and facilitating the trial runs by setting new initial tap coefficient values and observing their traversal afterwards.

D. Monitor Component

The monitor component in Fig. 9 plays an important role in this UVM testbench by collecting a trace of the tap coefficient values traversed by the adaptation controller and updating the coverage database when one of the termination conditions is met. Specifically, the monitor continues collecting the trace until either a new final locked state is reached or a previously-visited state is revisited. Depending on which termination condition occurs, the monitor records the states included in the trace in the coverage database with a new final state or an existing final state, respectively. Note that the determination of whether the adaptation controller has reached a final locked state is based on checking if the controller revisits a state that was recorded in the trace of the current trial run 8 or more update cycles earlier.

```
class MONITOR extends uvm_monitor;
    `uvm_component_utils(MONITOR)

    VIF_t VMIF;
    COVERAGE CVG;
    uvm_event LOCKED;
    ...
    function void build_phase(uvm_phase phase);
        void'(uvm_config_db #(VIF_t)::get(null, "uvm_test_top", "Key_VMIF", VMIF));
        void'(uvm_config_db #(COVERAGE)::get(null, "uvm_test_top", "Key_CVG", CVG));
    endfunction: build_phase

    task run_phase(uvm_phase phase);
        DATA_t queue[$];
        shortint index_lock;
        int result[$];
        LOCKED = uvm_event_pool::get_global("LOCKED");
        wait(!VMIF.RST);
        forever begin:LOOP
            @(VMIF.TRIG);

            // collect a trace of states until a lock is reached
            if (CVG.visited.exists(VMIF.DATA) && CVG.visited[VMIF.DATA] > 0)
                index_lock = CVG.visited[VMIF.DATA];
            else begin
                result = queue.find_first_index with (item == VMIF.DATA);
                if (result.size() != 0 && result[0] < queue.size() - 8)
                    index_lock = -1;
                else begin
                    queue.push_back(VMIF.DATA);
                    index_lock = 0;
                end
            end

            // put the trace into the coverage database
            if (index_lock != 0) begin
                if (index_lock < 0) begin
                    CVG.locks.push_back(queue[$]);
                    index_lock = CVG.locks.size();
                end
                `uvm_info("MON", $sformatf("\n  | MON #%0d: reaching %b (final state #%0d: %b)", CVG.num_trials,
queue[$], index_lock, CVG.locks[index_lock-1]), UVM_HIGH);
                foreach (queue[i]) CVG.visited[queue[i]] = index_lock;
                queue.delete();

                // trigger LOCKED to initiate a new search
                LOCKED.trigger();
            end
        end: LOOP
    endtask: run_phase
endclass: MONITOR
```

Figure 9. The monitor component observing the tap coefficient values traversed by the adaptation controller and updating the coverage database depending on whether a new final locked state is reached or a previously-visited state is revisited.

E. Scoreboard Component

The scoreboard component in this UVM testbench simply reports the pass/fail result after the sequence of trial runs is completed. As the code listed in Fig. 10 shows, it determines whether the global convergence property of the sign-sign LMS adaptation controller is verified as true or false, based on the number of final locked states registered in the *locks* queue of the coverage database (*CVG.locks*). If *CVG.locks* has only one entry, it implies that the adaptation consistently converges to the same final state for all possible initial states. Otherwise, if *CVG.locks* has multiple entries, it suggests that there are some initial states that lead to different final states than others, which requires further examination.

```
class SCOREBOARD extends uvm_scoreboard;
    `uvm_component_utils(SCOREBOARD)

    COVERAGE CVG;
    uvm_event LOCKED;
    ...

    function void build_phase(uvm_phase phase);
        void'(uvm_config_db #(COVERAGE)::get(null, "uvm_test_top", "Key_CVG", CVG));
    endfunction: build_phase

    function void report_phase(uvm_phase phase);
        if (CVG.locks.size() == 1) begin
            `uvm_info("SCB", "\n  | SCB: [PASS] all tested initial states lead to the same locked state.", UVM_HIGH);
        end
        else begin
            if (CVG.locks.size() >= 2) begin
                `uvm_info("SCB", $sformatf("\n  | SCB: [FAIL] more than one locked states are found:\n    #1: %b\n
#2: %b\n", CVG.locks[0], CVG.locks[1]), UVM_HIGH);
            end
            else begin
                `uvm_info("SCB", "\n  | SCB: [FAIL] no locked state is found.", UVM_HIGH);
            end
        end
        `uvm_info("SCB", $sformatf("\n    | SCB: number of trials = %0d, final coverage = %g (%0d/%0d)",
CVG.num_trials, real'(CVG.visited.size())/CVG.size_full, CVG.visited.size(), CVG.size_full), UVM_HIGH);
    endfunction: report_phase

endclass: SCOREBOARD
```

Figure 10. The scoreboard component reporting the pass/fail result of the simulation.

V. EXPERIMENTAL RESULTS

This section discusses the simulation results obtained using the presented UVM testbench to check the global convergence property of the sign-sign LMS adaptation controller for the 4-tap DFE of the 16-Gb/s high-speed wireline transceiver model described in Section III. The simulations are run with Cadence Xcelium and Scientific Analog's XMODEL, and the reported runtimes are measured on a 64-bit Linux machine with 2.3-GHz 4-core Intel Core i7 processor and 8-GB of memory. The following subsections present the results with multiple scenarios: the case with high channel loss, the case with unconstrained tap coefficients, and the case with constrained tap coefficients. Based on the discussions in Section II, the sign-sign LMS adaptation loop is expected to encounter difficulties with global convergence when the tap coefficient values become large, either due to high channel loss or unconstrained initialization.

A. Case with High Channel Loss

First, the simulation is run with a channel having very high loss, such as a -45dB loss at the Nyquist rate of 8GHz. Fig. 11 shows the simulation log generated by the UVM testbench. After running 7 trials with randomized initial tap coefficient values, the testbench identified two final locked states that the DFE adaptation loop could converge to. Since the simulation was aborted as soon as the second locked state was found, the total runtime was only 85 seconds.

```
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 500.000ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #1: trying new initial state: 10000010000100000100000
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1137.467ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #1: reaching 10011100100100001100000 (final state #1: 1001111001001000011000000)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 1138.467ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #2: trying new initial state: 01100101101010000101111O
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1823.684ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #2: reaching 10100110011010000100000 (final state #1: 1001111001001000011000000)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 1824.684ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #3: trying new initial state: 10011110010110000000111O
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 2159.451ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #3: reaching 10101010011110001010000O (final state #1: 1001111001001000011000000)
  ...
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 3071.904ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #7: trying new initial state: 01101001110010001011111O
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 3422.325ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #7: reaching 01011101101010000001110O (final state #2: 0101110110101000000111OO)

UVM_INFO /PATH/UVM_eqadapt/uvm_tb/SCB_PKG.sv(45) @ 3423.325ns: uvm_test_top.E.SCB [SCB]
  | SCB: [FAIL] more than one locked states are found:
    #1: 1001111001001000011000000
    #2: 0101110110101000000111OO
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/SCB_PKG.sv(51) @ 3423.325ns: uvm_test_top.E.SCB [SCB]
  | SCB: number of trials = 7, final coverage = 0.999862 (16774909/16777216)
```

Figure 11. The UVM simulation log reporting a global convergence failure when the channel has a high loss of -45-dB at 8GHz.

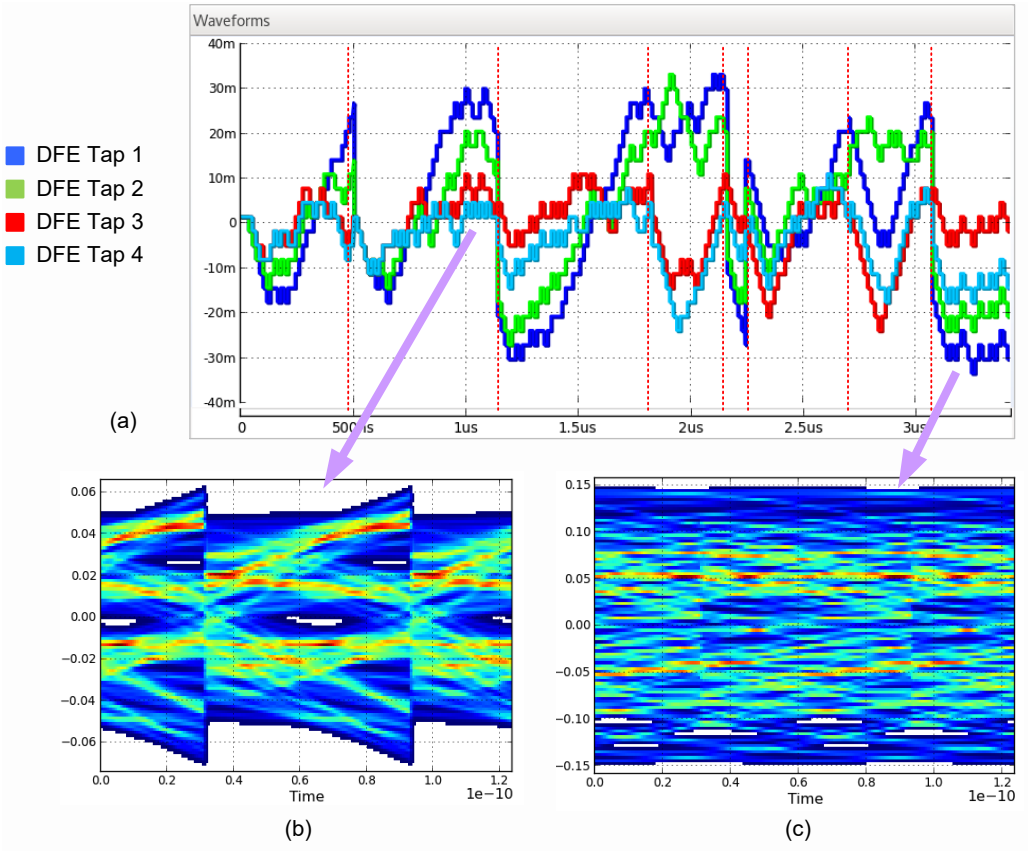

Figure 12. (a) The trajectories of the DFE tap coefficient values during the simulation with a -45dB channel loss; (b) the equalized eye diagram at the first locked state, and (c) the equalized eye diagram at the second locked state.

Fig. 12(a) plots the trajectories of the DFE tap coefficient values traversed during the entire simulation. The time points where the tap coefficient values have abrupt changes indicate when the testbench initiates a new trial run with a newly generated set of tap coefficient values. Fig. 12(b) and 12(c) show the equalized eye diagrams using the two sets of tap coefficients identified by the simulation. The first set seems adequate, although the eye opening is still small due to the high uncompensated loss of the channel. The second set clearly represents a false locked state, yielding no eye opening at all. In this second set, the tap coefficients have relatively large values: w_1=010111 (-27mV), w_2=011010 (-17mV), w_3=100000 (1.6mV), and w_4=011100 (-11mV). These values cause the DFE receiver to produce an alternating data pattern of 10101010 regardless of the actual input to the receiver.

B. Case with Unconstrained Tap Coefficients

Next, the simulation is run with a channel exhibiting a moderate loss of -20-dB at 8GHz and no constraints on the tap coefficient values, other than the minimum and maximum bounds of -0.1 and +0.1V, respectively. Fig. 13 shows the simulation log generated by the UVM testbench for this case. After running just 6 trials in 25 seconds, the testbench identified two final locked states that the DFE adaptation loop could converge to.

Fig. 14(a) plots the trajectories of the DFE tap coefficient values traversed during the entire simulation. And Fig. 14(b) and 14(c) show the equalized eye diagrams using the two sets of tap coefficients identified by the simulation. The first set is clearly the desired one, yielding a wide eye opening of 58mV$_{pp,diff}$. In contrast, the second set produces a very strange-looking eye diagram. Similar to the case with the high-loss channel, the DFE tap coefficients have large values that can force the decision solely based on the previous outputs, regardless of the current input. The DFE receiver in this case also produces an alternating data pattern of 10101010.

```
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 500.000ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #1: trying new initial state: 10000010000100000100000
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 818.731ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #1: reaching 10011001111110001011111 (final state #1: 10011001111110001011111)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 819.731ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #2: trying new initial state: 01101001001011110011101
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1361.606ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #2: reaching 10011010000100011100000 (final state #1: 10011001111110001011111)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 1362.606ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #3: trying new initial state: 00001101000101111111000
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1952.230ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #3: reaching 10000110000110000110001 (final state #1: 10011001111110001011111)
  ...
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 3580.856ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #6: trying new initial state: 11100100100010111001011
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 3883.683ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #6: reaching 10111101001010000101010 (final state #2: 10111010010100001010101)

UVM_INFO /PATH/UVM_eqadapt/uvm_tb/SCB_PKG.sv(45) @ 3884.683ns: uvm_test_top.E.SCB [SCB]
  | SCB: [FAIL] more than one locked states are found:
    #1: 10011001111110001011111
    #2: 10111010010100001010101
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/SCB_PKG.sv(51) @ 3884.683ns: uvm_test_top.E.SCB [SCB]
  | SCB: number of trials = 6, final coverage = 1.15037e-05 (193/16777216)
```

Figure 13. The UVM simulation log reporting a global convergence failure when the initial tap coefficient space is not constrained.

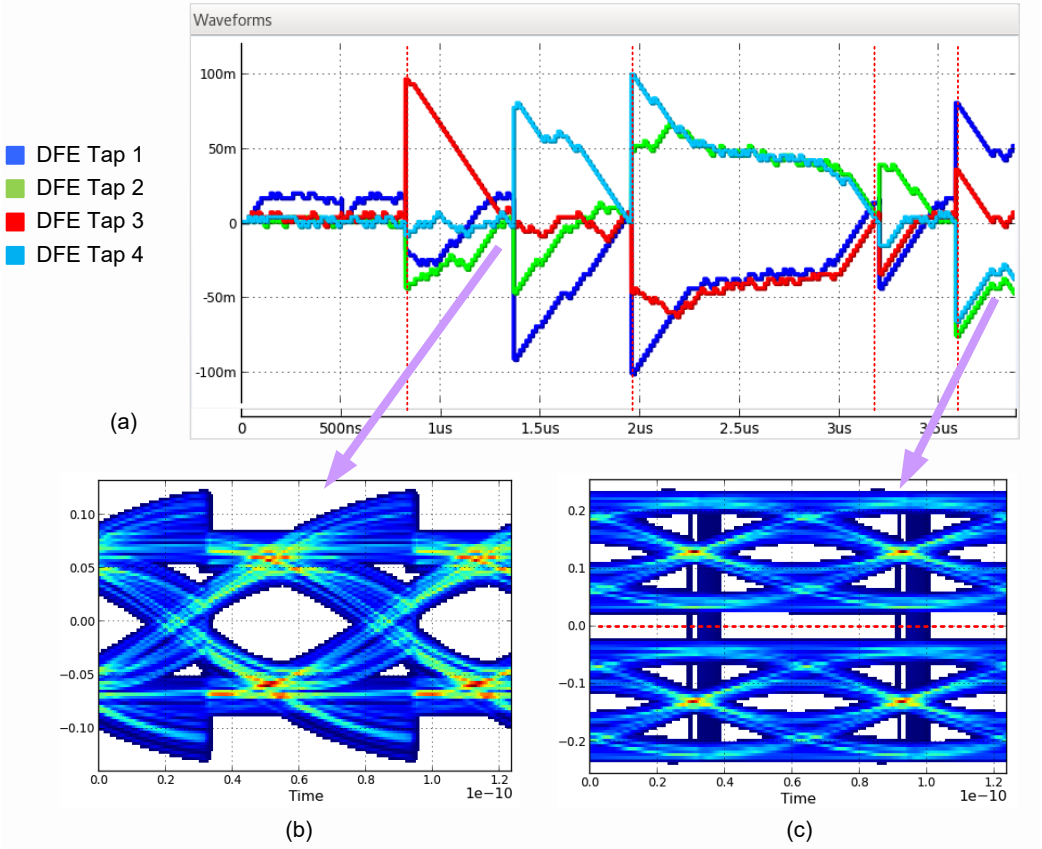

Figure 14. (a) The trajectories of the DFE tap coefficient values during the simulation when the state space is unconstrained; (b) the equalized eye diagram at the first locked state, and (c) the equalized eye diagram at the second locked state.

C. Case with Constrained Tap Coefficients

Finally, the simulation is run with the moderate-loss channel and the constraints discussed in Section IV-A, namely, $|w_1|+|w_2|+|w_3|+|w_4| \le 0.05$, $|w_1| > |w_2| > |w_3|$, and $|w_2| > |w_4|$. These constraints exclude the problematic initial states identified in the previous subsection and make the simulation feasible by reducing the state space.

The simulation log shown in Fig. 15 reports a successful global convergence after running 1,721 trials for 5 hours and 12 minutes. The simulation verified a total of 2,347 states, achieving an effective 26.7% reduction in the number of trial runs required. Further improvement may be possible by adding more guidance to the random selection of the next initial state, so that each trial run can traverse as many intermediate states as possible.

```
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 500.000ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #1: trying new initial state: 10000010000100000100000
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 882.482ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #1: reaching 10010010000100001100001 (final state #1: 10010010000100001100001)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 883.482ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #2: trying new initial state: 01100101101010000101111O
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1074.667ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #2: reaching 10001110000100010100000 (final state #1: 10010010000100001100001)
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 1075.667ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #3: trying new initial state: 10011110010001111100000O
UVM_INFO /PATH/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 1155.357ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #3: reaching 10011010000100001100001 (final state #1: 10010010000100001100001)
  ...
UVM_INFO /users/jaeha/projects/UVM_eqadapt/uvm_tb/DRV_PKG.sv(40) @ 83992.481ns: uvm_test_top.E.AGNTD.DRV [DRV]
  | DRV #1721: trying new initial state: 10010101101101110011100
UVM_INFO /users/jaeha/projects/UVM_eqadapt/uvm_tb/MON_PKG.sv(62) @ 84024.354ns: uvm_test_top.E.AGNTM.MON [MON]
  | MON #1721: reaching 10010101101101110011100 (final state #1: 10010010000100001100001)

UVM_INFO /users/jaeha/projects/UVM_eqadapt/uvm_tb/SCB_PKG.sv(41) @ 84025.354ns: uvm_test_top.E.SCB [SCB]
  | SCB: [PASS] all tested initial states lead to the same locked state.
UVM_INFO /users/jaeha/projects/UVM_eqadapt/uvm_tb/SCB_PKG.sv(51) @ 84025.354ns: uvm_test_top.E.SCB [SCB]
  | SCB: number of trials = 1721, final coverage = 1.0 (16777216/16777216)
```

Figure 15. The UVM simulation log reporting a successful global convergence success when the initial tap coefficient space is constrained with $|w_1|+|w_2|+|w_3|+|w_4| \leq 0.05$, $|w_1| > |w_2| > |w_3|$, and $|w_2| > |w_4|$.

VI. CONCLUSION

This work demonstrated that the power of UVM can be harnessed to verify the global convergence property of analog/mixed-signal systems. Specifically, it presented a UVM testbench capable of checking whether a sign-sign LMS adaptation controller for a high-speed wireline DFE receiver can reach the desired equalized state regardless of its initial state conditions. To achieve this, the proposed testbench launches a sequence of trial runs with different initial states with an objective of exploring all possible states in the system. Thus, the proposed testbench generates a reactive stimulus, but its stimulus-response pattern does not conform to the standard UVM framework described in [8]. Instead, the testbench utilizes a state coverage database shared via the UVM configuration database and a UVM event maintained by the global event pool. Further directions may include improving the efficiency of state exploration and verifying the global convergence property of other analog/mixed-signal systems as well.

VII. ACKNOWLEDGMENT

The EDA tools used in this work were supported by the IC Design Education Center (IDEC), Korea and Scientific Analog, Inc, Palo Alto, U.S.A.

VIII. REFERENCES

[1] R. W. Lucky, "Techniques for Adaptive Equalization of Digital Communication Systems," The Bell Systems Technical Journal, Feb. 1966.

[2] V. Stojanovic, et al., “Autonomous Dual-Mode (PAM2/PAM4) Serial Link Transceiver with Adaptive Equalization and Data Recovery,” IEEE J. Solid-State Circuits, April 2005.

[3] J. Kim, "A UVM Reactive Testbench for Jitter Tolerance Measurement of High-Speed Wireline Receivers," *Design and Verification Conference and Exhibition* (*DVCON*) *U.S.*, Mar. 2023.

[4] Scientific Analog, Inc. XMODEL. [Online]. Available at: https://www.scianalog.com/xmodel.

[5] J. E. Jang, et al., “True Event-Driven Simulation of Analog/Mixed-Signal Behaviors in SystemVerilog: A Decision-Feedback Equalizing (DFE) Receiver Example,” *IEEE Custom Integrated Circuits Conf.* (*CICC*), Sept. 2012.

[6] C. Dancak, "A UVM SystemVerilog Testbench for Analog/Mixed-Signal Verification: A Digitally-Programmable Analog Filter Example," *Design and Verification Conference and Exhibition* (*DVCON*) *U.S.*, Mar. 2021.

[7] C. Dancak, "A UVM SystemVerilog Testbench for Directed & Random Testing of an AMS LDO Voltage Regulator," *Design and Verification Conference and Exhibition* (*DVCON*) *U.S.*, Mar. 2024.

[8] C. E. Cummings, et al., “UVM Reactive Stimulus Techniques,” *Design and Verification Conference and Exhibition* (*DVCON*) *U.S.*, Mar. 2020.