A UVM Reactive Testbench for Jitter Tolerance Measurement of High-Speed Wireline Receivers

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

***Abstract*—This paper demonstrates a UVM testbench that can measure the jitter tolerance (JTOL) characteristics of a high-speed wireline receiver. This JTOL measurement requires an iterative search finding the maximum magnitude of the sinusoidal jitter (SJ) that keeps the bit-error rate (BER) of the receiver below the target rate (e.g., 10^{-12}). The presented UVM testbench adopts the reactive stimulus technique to perform the search for an analog/mixed-signal system, i.e., checking its BERs while varying the SJ frequencies and magnitudes. The UVM testbench also encapsulates all the analog/mixed-signal contents within the fixture module so that the rest of the testbench can be built using the standard UVM components. The model of the high-speed transceiver and the fixture instrumentations to apply stimuli and measure responses are composed with XMODEL primitives, which enable event-driven simulation of analog/mixed-signal circuits in SystemVerilog. A case with an example 16-Gb/s high-speed wireline receiver model shows that the presented testbench can measure the JTOL characteristics for 20 frequency points after performing 106 BER tests for a total duration of 41 minutes.**

I. Introduction

A jitter tolerance (JTOL) test measures the resilience of a high-speed wireline receiver to the sinusoidal jitter (SJ) added to the incoming data stream [1]. As illustrated in Fig. 1, the test setup adds SJ to the transmitter clock and checks if the receiver can still operate with a bit-error rate (BER) below the target rate (e.g., 10^{-12}). While the standards specify the minimum SJ magnitude that must be tolerated for each SJ frequency (called the JTOL mask), the actual JTOL test looks for the maximum SJ magnitude that can be tolerated for each SJ frequency.

Since each trial measuring the BER consumes considerable time, it is important to minimize the number of trials required to measure the whole JTOL characteristic curve. In other words, it is not feasible to measure the JTOL characteristics by blindly trying all SJ magnitude values for a given SJ frequency and later choosing the maximum value that meets the target BER. Instead, a reactive testbench that can judiciously pick the next trial point considering the outcomes of the previous trials is necessary.

This paper shows how one can write such a reactive testbench that can measure the JTOL characteristics of a high-speed wireline receiver in UVM. The presented UVM testbench adopts the reactive stimulus technique in [2] and follows the guidelines in [3], encapsulating all the analog/mixed-signal models and instrumentations within a fixture module and

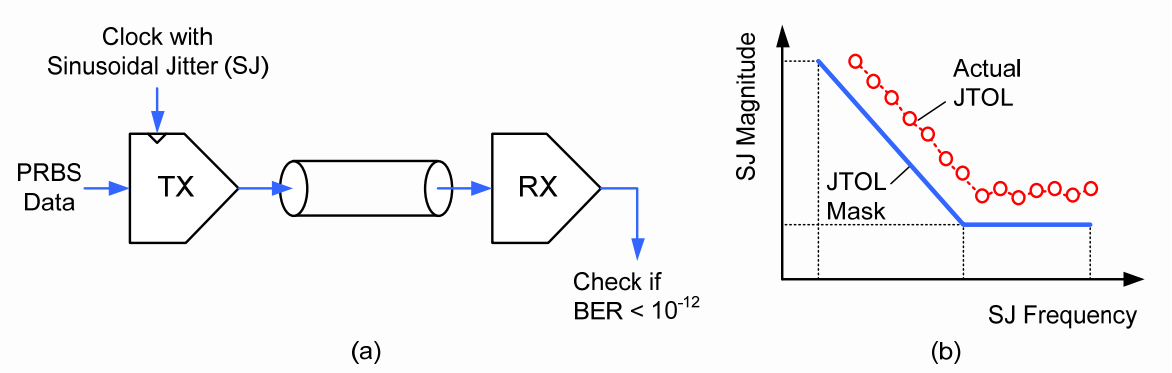


Figure 1. The jitter tolerance test of a high-speed receiver: (a) the test setup and (b) measured results.

composing the rest of the testbench with the standard UVM components only. With a 16-Gb/s high-speed wireline transceiver model and its fixture module described with XMODEL primitives [4], the UVM testbench can measure the JTOL characteristics for 20 frequency points with target BER of 10^{-12} after performing 106 BER tests for a total duration of 41 minutes.

The rest of the paper is organized as follows. Section II first describes the model of our system under verification, i.e., the model of a high-speed wireline transceiver. And Section III presents the UVM testbench that can measure its JTOL characteristics using a reactive stimulus technique. Finally, Section IV discusses the simulation results and Section V concludes the paper.

II. Modeling of High-Speed Wireline Transceiver

High-speed wireline transceivers achieve multi-Gb/s data rates transferring digital data from one IC chip to another via three key enabling elements: (1) impedance-controlled channel, (2) high-speed transmitter and receiver circuits, and (3) precise timing generation and recovery circuits. Fig. 2 illustrates the overall block diagram of an example 16-Gb/s high-speed wireline transceiver, employing these key enabling elements. First, a charge-pump phase-locked loop (*TX-PLL*) generates a 16-GHz clock (*tx_clk*) from a 2-GHz reference (*tx_ref_clk*). Second, 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. Third, on the receiver side, a continuous-time linear equalizer stage (*RX-EQ*) compensates the frequency-dependent loss in the channel, so that sampling its output can produce the output data stream (*Dout*). Fourth, the clock that triggers the data samplers (*rx_clk*) is recovered by a clock-and-data recovery loop (*RX-CDR*), which is in turn made of an internal PLL that generates a set of multi-phase clocks from a reference (*rx_ref_clk*) and a phase interpolator (*PI*) stage that synthesizes them into the clock with a desired phase, as directed by bang-bang phase detector (*PD*) and digital loop filter (*LF*) blocks.

XMODEL from Scientific Analog [4] provides a set of primitives that can model various analog/mixed-signal circuits and simulate them efficiently in SystemVerilog. Especially when modeling and simulating a high-speed wireline transceiver in SystemVerilog, XMODEL provides many advantages compared to alternative solutions, such as real-number modeling (RNM). First, XMODEL is capable of fast, event-driven simulation of analog signals propagating through the transmission line channels, based on its equation-based signal type called *xreal* [5]. Second, with another signal type called *xbit*, it can express the precise timing of the PLL and CDR clocks without being limited to the simulator's timestep. Third, XMODEL can describe analog circuits directly as a network of circuit elements and simulate their voltages and currents. For instance, one can describe the differential current-mode transmitter with 1-tap de-emphasis directly as two differential pairs with shorted outputs, each steering its own bias current based on the current-cycle input and previous-cycle input, respectively. Also, the transmission lines modeled with *tline* primitives can have frequency-dependent losses as well as reflections due to impedance mismatch. Fourth, XMODEL can propagate probability information through some of its primitives, allowing efficient statistical simulation

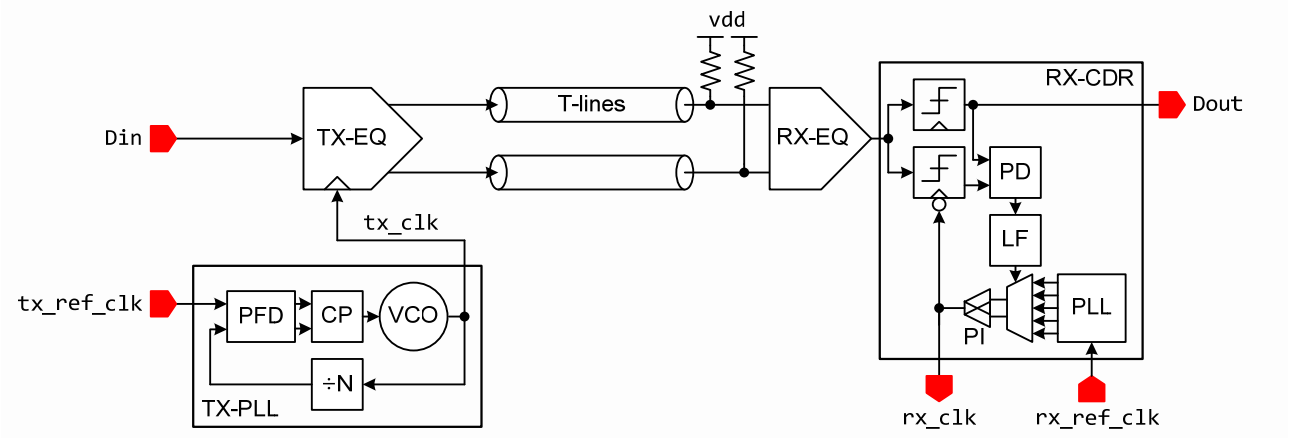
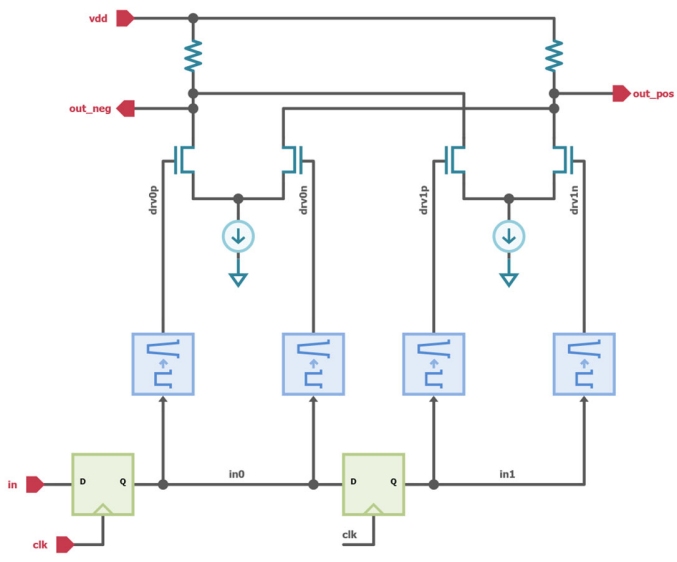


Figure 2. Overall block diagram of a high-speed wireline transceiver model.

that can estimate the BER of a high-speed wireline transceiver as low as 10^{-20} in a few minutes. This last capability is particularly critical to our JTOL testbench, since validating a BER less than 10^{-12} via a plain Monte-Carlo simulation would take at least ~3 years even when the simulator can run 10,000 bits per second.

Fig. 3 illustrates the case of modeling the *TX-EQ* block in Fig. 2, i.e., the differential current transmitter with 1-tap de-emphasis using the XMODEL primitives. Note that a model with XMODEL primitives can be described either as a schematic (*left*) or as a structural model in SystemVerilog (*right*). Functionality-wise, this *TX-EQ* block drives its two outputs with a pair of pull-down currents, whose net difference is equal to the input filtered by the discrete-time transfer function $H(z) = 1-\alpha z^{-1}$. Component-wise, the block is made of two differential pairs sharing the output nodes. Each differential pair is driven by the current-cycle input and previous-cycle input retimed to the clock's rising edge, produced by a series of two D flip-flops registering the input signal, respectively. The ratio between the tail currents of the two differential pairs sets the value of α. The two resistors connected between the outputs and supply voltage serve both as load resistances and as terminations matched to the characteristic impedance of the transmission lines.

As listed in the SystemVerilog structural model, the *TX-EQ* block is modeled using the logic primitives like *dff_xbit*, function primitives like *transition*, and circuit primitives like *isource*, *nmosfet*, and *resistor*. The *transition* primitive is analogous to the *transition*() operator in Verilog-A, converting the digital input to analog output with finite rise/fall transition times. For the part modeled with the circuit primitives, XMODEL computes all the node voltages and branch currents as governed by the Kirchhoff circuit laws. Nonetheless, its simulation does not invoke a SPICE engine. The XMODEL engine interfacing via SystemVerilog DPI can simulate all the analog waveforms in an event-driven manner using the equation-based, *xreal*-type representation.




```
module tx_eq (
    output xreal out_neg,
    output xreal out_pos,
    input xbit clk,
    input xbit in,
    input xreal vdd
);

// signal declarations
xreal drv0p, drv0n, drv1p, drv1n;
xreal tail0, tail1;
xbit in0, in1;

// instance declarations
isource     #(.mode("dc"), .dc(0.00875))
            I0 (.neg(`ground), .pos(tail0), .in(`ground));
isource     #(.mode("dc"), .dc(0.00125))
            I1 (.neg(`ground), .pos(tail1), .in(`ground));
nmosfet     #(.W(2e-05), .L(4e-08), .Kp(2e-05), .Vth(0.5))
            M0 (.g(drv0p), .d(out_neg), .b(`ground), .s(tail0));
nmosfet     #(.W(2e-05), .L(4e-08), .Kp(2e-05), .Vth(0.5))
            M1 (.g(drv0n), .d(out_pos), .b(`ground), .s(tail0));
nmosfet     #(.W(2e-05), .L(4e-08), .Kp(2e-05), .Vth(0.5))
            M2 (.g(drv1p), .d(out_neg), .b(`ground), .s(tail1));
nmosfet     #(.W(2e-05), .L(4e-08), .Kp(2e-05), .Vth(0.5))
            M3 (.g(drv1n), .d(out_pos), .b(`ground), .s(tail1));
resistor    #(.R(50)) R0 (.neg(out_pos), .pos(vdd));
resistor    #(.R(50)) R1 (.neg(out_neg), .pos(vdd));
transition  #(.value0(0.0), .value1(1.0)) XP0 (.out(drv0p), .in(in0));
transition  #(.value0(1.0), .value1(0.0)) XP1 (.out(drv0n), .in(in0));
transition  #(.value0(1.0), .value1(0.0)) XP2 (.out(drv1p), .in(in1));
transition #(.value0(0.0), .value1(1.0)) XP3 (.out(drv1n), .in(in1));
dff_xbit    XP4 (.clk(clk), .d(in), .q(in0));
dff_xbit    XP5 (.clk(clk), .d(in0), .q(in1));

endmodule   // tx_eq
```

Figure 3. SystemVerilog model of the differential current-mode transmitter with 1-tap de-emphasis equalization (TX-EQ) using XMODEL primitives.

Fig. 4 shows the simulated eye diagrams of the described high-speed wireline transceiver model. An eye diagram is constructed by overlapping the received signal waveforms of each bit period and can visually indicate how good a separation is maintained between the levels of symbol 0 and symbol 1 and between the symbols adjacent in time. Fig. 4(a) shows the eye diagram without equalization, which is completely closed due to the frequency-dependent loss in the channel. Fig. 4(b) is the eye diagram with the transmitter-side equalization (*TX-EQ*) enabled, showing that it can open the eye to some degree by compensating the loss. Fig. 4(c) shows that the eye opening can be further improved with the receiver-side equalization (*RX-EQ*) enabled as well. These results demonstrate that with XMODEL primitives, one can compose a SystemVerilog model for a fully-functional high-speed wireline transceiver, which can serve as a device under verification (DUV) in the UVM testbench described next.

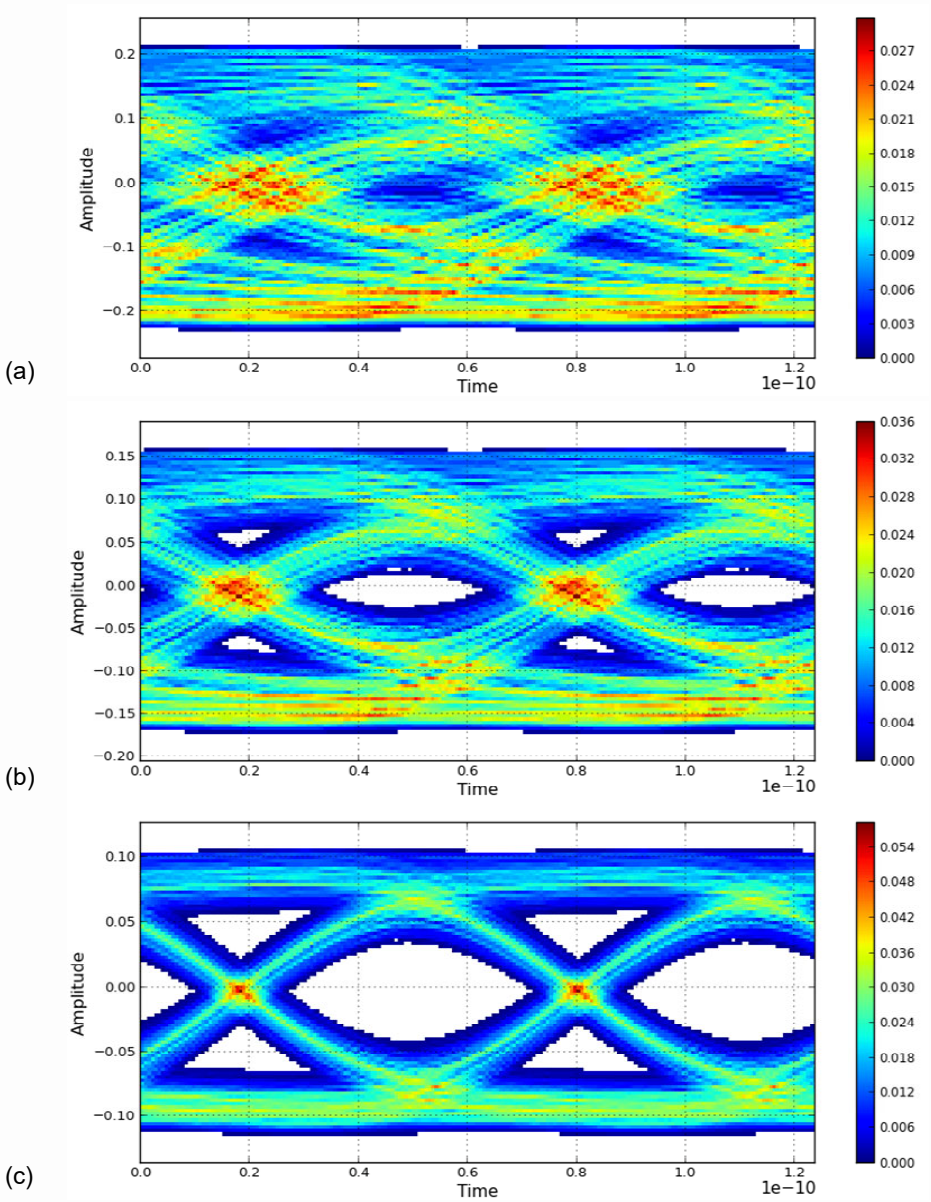


Figure 4. Simulated eye diagrams of the high-speed transceiver model: (a) with both TX-EQ and RX-EQ disabled, (b) with TX-EQ enabled, (c) with both TX-EQ and RX-EQ enabled.

III. UVM Testbench for JTOL Measurement

Fig. 5 illustrates the organization of the proposed UVM testbench that measures the JTOL characteristics of a high-speed wireline receiver. Following the approach described in [3], all the analog-specific details are encapsulated within a fixture module. The fixture module contains the models of the high-speed wireline transmitter-receiver pair and the instrumentations to apply the SJ with the prescribed frequency and magnitude and measure the resulting BER. The transceiver models and instrumentations composed with XMODEL primitives enable a seamless simulation of such an analog/mixed-signal system in SystemVerilog and also a fast estimation of BERs as low as 10^{-12} with only ~30,000 samples.

Besides the fixture module, the rest of the testbench are built with standard UVM components. The key difference from the testbench in [3] is that the driver agent also serves as the monitor agent, as it must observe the response to choose the next stimulus. The sequence and driver are the main components performing the iterative search using the reactive stimulus technique [2]. The sequencer has two-way communication with the driver, sending the next packet to drive and receiving the response from the fixture. The packet contains the SJ frequency and magnitude value to try, and the response contains the corresponding BER value. All the BER trial results are broadcast to the scoreboard which book-keeps the largest SJ magnitude that meets the target BER for each SJ frequency. When all the trials are completed, the scoreboard reports the final JTOL results.

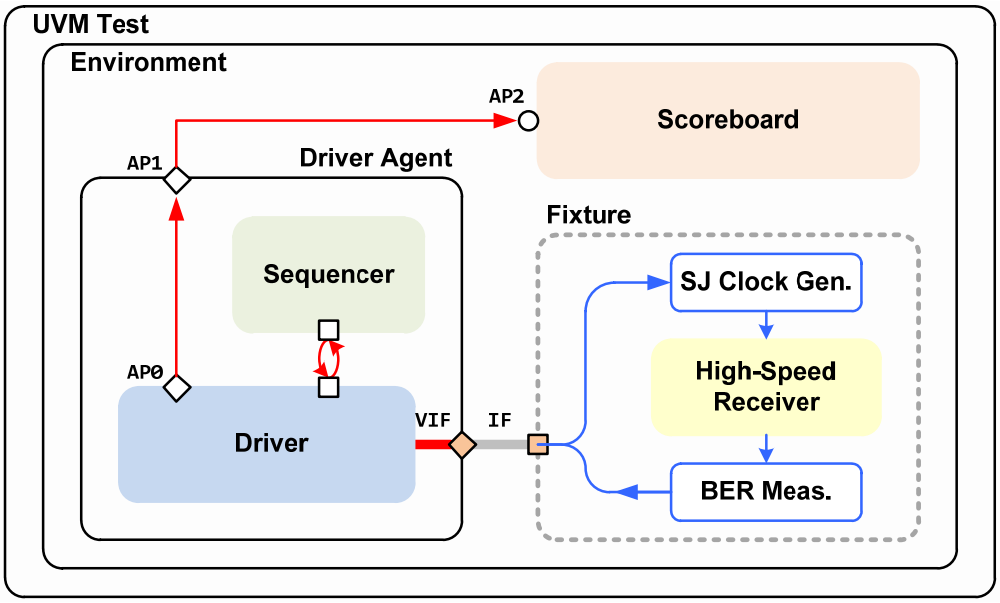


Figure 5. Conceptual diagram of the UVM testbench performing JTOL tests with reactive stimulus technique.

A. Fixture Module

The fixture module instantiates the high-speed wireline transceiver model and also contains the instrumentations to measure its JTOL characteristics. Also, it is the only module that contains the XMODEL primitives. Fig. 6 lists the code of the fixture module.

The fixture module handshakes with the driver component using the *START* and *DONE* signals to perform multiple BER measurements within a single simulation run. That is, when the driver wants to initiate a new BER measurement with a set of SJ frequency and magnitude values (*SJ_freq* and *SJ_mag*), it asserts *START* to 1. Then, the fixture module carries out the test by applying the corresponding SJ to the transmitter clock (*tx_clk*), feeding the pseudo-random bit sequence (PRBS) to the data input (*Din*), and supplying the receiver reference clock (*rx_ref_clk*) and supply voltage (*vdd*). It initially waits for a period of time prescribed by *t_lock* and starts collecting the bit error probabilities for a period of time prescribed by *t_meas*. After completing the measurement, it sends the resulting BER value back to the driver and asserts *DONE* to 1.

```
interface IF_t (input bit RST);
    bit START, DONE;                // handshaking signals
    real SJ_freq, SJ_mag;           // stimulus parameters
    real BER;                       // measurement results
endinterface: IF_t

module FIXTURE (IF_t IF);
    parameter real data_freq = 16.0e9;      // data rate
    parameter real ref_freq = 2.0e9;        // RX reference clock frequency
    parameter real t_lock = 200e-9;         // initial lock time
    parameter real t_meas = 2e-6;           // measurement time

    xbit tx_ref_clk, rx_ref_clk, rx_clk;
    xbit Din, Dout;
    xreal vdd;
    xreal n1, n2, n3, freq;

    real SJ_freq, SJ_amp;
    longint num_bit, num_err;
    real prb_err, sum_prb;
    wire bit_err;
    bit init;

    // DUT
    hslink_jtol     hslink(.tx_clk, .rx_ref_clk, .rx_clk, .Din, .Dout, .vdd);

    // interface handshaking
    always begin: LOOP
        @(posedge IF.START);
        IF.DONE = 0;
        SJ_freq = IF.SJ_freq;
        SJ_amp = M_PI*IF.SJ_mag;

        init = 1;
        #($xmodel_reltime(t_lock));
        init = 0;
        #($xmodel_reltime(t_meas));

        IF.BER = sum_prb/num_bit;
        IF.DONE = 1;
    end: LOOP

    // transmitter clock with sinusoidal jitter
    real_to_xreal   U1 (.in(SJ_freq), .out(freq));
    integ_mod       #(.scale(2*M_PI), .modulus(2*M_PI)) U2 (.in(freq), .out(n1));
    sin_func        U3 (.in(n1), .out(n2));
    scale_var       U4 (.in(n2), .out(n3), .scale(SJ_amp));
    phase_to_clk    #(.freq(data_freq)) U5 (.in(n3), .out(tx_clk));

    // clock, data, and supply sources
    prbs_gen        U6 (.trig(tx_clk), .out(Din));
    clk_gen         #(.freq(ref_freq)) U7 (rx_ref_clk);
    dc_gen          #(.value(1.2)) U8 (vdd);

    // BER measurement
    meas_ber        U9 (.bit_err(bit_err), .prb_err(prb_err),
                        .in(Dout), .in_ref(Dout), .clk(rx_clk));

    always @(posedge `value(rx_clk) or init) begin
        if (init) begin
            num_bit = 0;
            num_err = 0;
            sum_prb = 0.0;
        end
        else begin
            num_bit += 1;
            num_err += bit_err;
            sum_prb += prb_err;
        end
    end

endmodule: FIXTURE
```

Figure 6. The fixture module including the analog/mixed-signal instrumentations for measuring the JTOL characteristics of a high-speed wireline transceiver.

Fig. 7 illustrates the instrumentation with XMODEL primitives to generate a clock with SJ in a schematic form. First, the *real*-type *SJ_freq* input is converted to an *xreal*-type signal using a *real_to_xreal* primitive. This signal then drives an *integ_mod* primitive, which performs the integral and modulo operations and generates a sawtooth waveform ranging between 0 and 2π with a frequency equal to *SJ_freq*. The following *sin_func* and *scale_var* primitives convert this sawtooth waveform to a sinusoidal waveform with the frequency equal to *SJ_freq* and magnitude equal to *SJ_amp*. The *sin_func* primitive computes the sine function of the input and the *scale_var* primitive scales the input by a variable factor. Finally, a *phase_to_clk* primitive takes the resulting signal as its input and produces a fixed-frequency output clock (*tx_clk*) whose phase changes as the sinusoidal function of time with a frequency equal to *SJ_freq* and magnitude equal to *SJ_amp*.

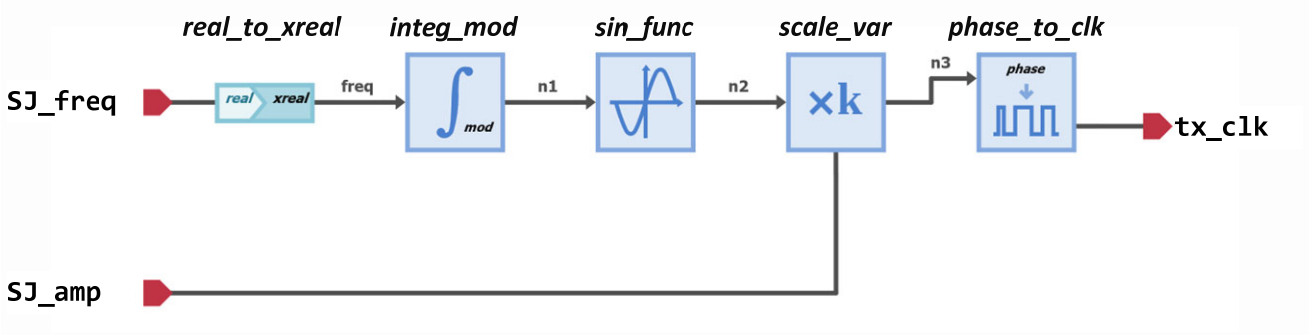


Figure 7. The fixture instrumentations with XMODEL primitives to generate a clock with sinusoidal jitter (SJ).

B. Driver Component

The driver component in this UVM testbench relays the information between the sequencer component and fixture module, and also broadcasts the BER measurement results to the scoreboard component. Fig. 8 lists the code of the driver component.

Once the reset signal (*RST*) is de-asserted, the driver initiates a new BER measurement whenever it receives a packet (i.e., a sequence item) from the sequencer. That is, when it retrieves a packet sent by the sequencer using the *get_next_item*() method, it drives the *SJ_freq* and *SJ_mag* signals to the fixture module using the values prescribed in the packet and asserts *START* to 1. Then, it waits until the fixture module responds with *DONE* asserted to 1, indicating that it has completed the measurement. It then prepares a response packet with the BER measurement result and sends it back to the sequencer using the *item_done*() method [2]. The driver also sends the result to the scoreboard by writing the response packet to the analysis port *AP0*, which forwards the packet to the analysis port *AP1* of the driver agent and then to *AP2* of the scoreboard component.

```
typedef virtual IF_t VIF_t;

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

    uvm_analysis_port #(PACKET) AP0;
    VIF_t VIF;
    PACKET PKT, RSP;

    function new(string INAME, uvm_component PARENT);
        super.new(INAME, PARENT);
    endfunction: new

    function void build_phase(uvm_phase phase);
        AP0 = new("AP0", this);
        uvm_config_db #(VIF_t)::get(this, "", "Key_VIF", VIF);
    endfunction: build_phase

    task run_phase(uvm_phase phase);
        VIF.START = 0;
        wait(!VIF.RST);

        forever begin: LOOP
```

```
            // apply stimuli
            seq_item_port.get_next_item(PKT);
            if (!$cast(RSP, PKT.clone())) `uvm_fatal("DRIVER", "failed casting PKT to RSP");
            RSP.set_id_info(PKT);

            VIF.SJ_freq = PKT.SJ_freq;
            VIF.SJ_mag = PKT.SJ_mag;
            VIF.START = 1;

            // collect responses
            @(posedge VIF.DONE);
            RSP.BER = VIF.BER;
            VIF.START = 0;
            seq_item_port.item_done(RSP);
            AP0.write(RSP);

            uvm_report_info("RUN",
                $sformatf("\n  | #%3d SJ freq=%e Hz, mag=%e UIpp --> BER=%e", PKT.tag, PKT.SJ_freq, PKT.SJ_mag, VIF.BER),
                UVM_HIGH
            );

            // intermission time
            #(1ns);
        end:  LOOP
    endtask: run_phase

endclass: DRIVER
```

Figure 8. The driver component initiating a new BER measurement when it receives a packet from the sequencer and sending the BER result back to the sequencer when the fixture module completes the measurement.

C. Sequencer Component

The sequencer component is the key element in this UVM testbench, as it performs the iterative search to find the maximum SJ magnitude for which the high-speed wireline receiver can achieve a satisfactory BER (e.g., < 10^{-12}) for each SJ frequency. Fig. 9 lists the code of the sequencer component.

The sequencer launches a series of BER measurement tests to iteratively find the maximum tolerable SJ magnitude. Each BER test is initiated by writing the SJ frequency and magnitude values (*SJ_freq* and *SJ_max*) to the packet *PKT*, sending the packet to the driver using the *start_item*() and *finish_item*() methods, and waiting for the driver's response using the *get_response*() method. The response packet *RSP* contains the measured BER for the given SJ conditions. Ref. [2] explains this reactive stimulus technique in detail.

To minimize the number of BER tests required to find the JTOL characteristics, the sequencer uses an algorithm that combines a coarse-resolution linear search and fine-resolution binary search. The search for the maximum SJ magnitude starts from the highest SJ frequency value and progresses towards the lowest SJ frequency value. A general JTOL characteristic curve has a shape shown in Fig. 1(b), where at the highest SJ frequency, the maximum tolerable SJ magnitude is bounded to below $1.0UI_{pp}$ and it gradually increases as the SJ frequency decreases. To exploit this smoothness of the JTOL characteristic curve, the search at the highest SJ frequency uses $0.5UI_{pp}$ as the initial SJ magnitude, and the subsequent searches at the lower SJ frequencies use the previously-found maximum tolerance SJ magnitude as the initial SJ magnitude.

First, the coarse-resolution linear search aims to identify a range of SJ magnitude that encloses the maximum tolerable SJ magnitude. In other words, it uses a linear search algorithm with a fixed increment *mag_inc* set to 20% of the initial SJ magnitude value and finds a range [*mag_min*, *mag_max*] where the BER is satisfactory for SJ magnitude equal to *mag_min* and not satisfactory for SJ magnitude equal to *mag_max*, assuming that the BER monotonically decreases as the SJ magnitude increases. To do so, the algorithm increments the SJ magnitude in one direction until it finds a transition from a satisfactory BER to an unsatisfactory BER, or a transition from an unsatisfactory BER to a satisfactory BER. The search direction and polarity of the sought transition depends on the BER outcome of the first trial.

Next, the fine-resolution binary search aims to narrow down the SJ magnitude range [*mag_min*, *mag_max*] until the ratio of *mag_max*/*mag_min* becomes less than 1.05. At each iteration, it selects the geometric mean of *mag_min* and *mag_max* as the new trial value and updates the lower or upper limit of the current range depending on its BER outcome. The geometric mean

instead of the arithmetic mean is used considering that the JTOL characteristic curve is plotted in log scale. When the criteria is met, the sequencer concludes the search for the current SJ frequency value and moves on to the next SJ frequency value.

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

    // parameters from command-line options
    real freq_min = 5e6;        // minimum value of JTOL frequency
    real freq_max = 5e9;        // maximum value of JTOL frequency
    int freq_numpt = 20;        // number of JTOL frequency points
    real BER_tol = 1e-12;       // BER tolerance

    PACKET PKT, RSP;
    real freq_ratio, mag_max, mag_min, mag_inc;
    int flag;

    function new(string INAME="SEQ_JTOL");
        super.new(INAME);
    endfunction: new

    task body();
        PKT = PACKET::type_id::create("PKT");
        PKT.BER_tol = BER_tol;
        freq_ratio = $pow(freq_max/freq_min, 1.0/(freq_numpt-1));

        for (int i=0; i<freq_numpt; i++) begin:LOOP
            PKT.tag = freq_numpt-i;
            PKT.SJ_freq = (i == 0) ? freq_max : PKT.SJ_freq/freq_ratio;
            PKT.SJ_mag = (i == 0) ? 0.5 : PKT.SJ_mag;
            mag_max = 1.0;
            mag_min = 0.01;
            mag_inc = 0.2*PKT.SJ_mag;
            flag = 0;

            // phase 1: linear search to find a failing point
            while (1) begin
                start_item(PKT);
                finish_item(PKT);
                get_response(RSP);

                if (RSP.BER < RSP.BER_tol) begin
                    mag_min = PKT.SJ_mag;
                    PKT.SJ_mag += mag_inc;
                    if (flag == -1) break;
                    else flag = 1;
                end
                else begin
                    mag_max = PKT.SJ_mag;
                    PKT.SJ_mag -= mag_inc;
                    if (flag == 1) break;
                    else flag = -1;
                end
            end

            // phase 2: binary search to find the pass/fail boundary
            while (mag_max/mag_min > 1.05) begin
                PKT.SJ_mag = $sqrt(mag_max * mag_min);
                start_item(PKT);
                finish_item(PKT);
                get_response(RSP);

                if (RSP.BER < RSP.BER_tol) mag_min = PKT.SJ_mag;
                else mag_max = PKT.SJ_mag;
            end
        end:  LOOP
    endtask: body

endclass: SEQ_JTOL
```

Figure 9. The sequencer component performing iterative search to find the maximum SJ magnitude that keeps the BER below the target rate.

D. Scoreboard Component

The scoreboard component receives all the BER measurement results broadcast by the driver component and keeps track of the largest SJ magnitude that satisfies the target BER for each SJ frequency. Fig. 10 lists the code of the scoreboard component.

The scoreboard component uses an instance of a custom UVM object named *SCORECARD* to do the book-keeping. This instance named *SCD* contains an associative array of struct-type elements called *DATA*, which stores the largest SJ magnitude that so far satisfied the target BER for each SJ frequency. For convenience, this associative array *DATA* is indexed by an integer value (*tag*), each value of which corresponds to a unique SJ frequency value.

The scoreboard contains a FIFO that buffers the response packets broadcast by the driver component and whenever it finds a new packet arrived, the scoreboard updates the content of *SCD.DATA*. That is, if the response packet contains a result with a satisfactory BER using the larger SJ magnitude for the given SJ frequency, the scoreboard updates the largest SJ magnitude value stored in *SCD.DATA*. As a result, when all the searches are over, *SCD.DATA* will have the full JTOL characteristics of the high-speed wireline receiver under verification.

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

    SCORECARD SCD;
    PACKET PKT;

    uvm_analysis_export #(PACKET) AP2;
    uvm_tlm_analysis_fifo #(PACKET) FIFO;

    function new(string INAME, uvm_component PARENT);
        super.new(INAME, PARENT);
        SCD = new("SCD");
    endfunction: new

    function void build_phase(uvm_phase phase);
        AP2 = new("AP2", this);
        FIFO = new("FIFO", this);
    endfunction: build_phase

    function void connect_phase(uvm_phase phase);
        super.connect_phase(phase);
        AP2.connect(FIFO.analysis_export);
    endfunction: connect_phase

    task run_phase(uvm_phase phase);
        int N;
        forever begin: SCORING
            FIFO.get(PKT);
            N = PKT.tag;

            if (SCD.DATA.exists(N)) begin
                if (PKT.BER < PKT.BER_tol && SCD.DATA[N].mag < PKT.SJ_mag)
                    SCD.DATA[N].mag = PKT.SJ_mag;
            end
            else begin
                SCD.DATA[N].freq = PKT.SJ_freq;
                SCD.DATA[N].mag = (PKT.BER < PKT.BER_tol) ? PKT.SJ_mag : 0.0;
            end
            SCD.num_trials++;
        end: SCORING
    endtask: run_phase

    function void report_phase(uvm_phase phase);
        SCD.print(SCD.printer);
    endfunction: report_phase

endclass: SCOREBOARD
```

Figure 10. The scoreboard component keeping track of the largest SJ magnitude that satisfies the target BER for each SJ frequency.

For completeness, Fig. 11 lists the code of the *SCORECARD* object class. It also defines a custom printing method via *do_print*() function, which prints out the measured JTOL characteristics in a table format when all the BER trial tests are finished.

```
typedef struct {
    real freq;              // JTOL frequency
    real mag;               // JTOL magnitude
} SCD_t;

class SCORECARD extends uvm_object;

    SCD_t DATA[int];
    int num_trials = 0;
    uvm_table_printer printer;

    function new(string INAME);
        super.new(INAME);
        printer = new();
    endfunction: new

    function void do_print(uvm_printer printer);
        string RULING;
        int N;

        // printer knob settings
        printer.knobs.header     = 1'b0;
        printer.knobs.identifier = 1'b1;
        printer.knobs.full_name  = 1'b0;
        printer.knobs.type_name  = 1'b0;
        printer.knobs.size       = 1'b0;
        printer.knobs.prefix     = "";

        // print jitter tolerance results
        RULING = {48{"-"}};
        printer.print_generic("", "", 0, RULING);
        printer.print_generic("", "", 0, "JITTER TOLERANCE (JTOL)");
        printer.print_generic("", "", 0, "INDEX     FREQUENCY(Hz)   MAGNITUDE(UIpp)");
        printer.print_generic("", "", 0, RULING);

        foreach (DATA[N]) begin: PRINT_LINE
            printer.print_generic("", "", 0, $sformatf("%-8d  %.4e      %.4f", N, DATA[N].freq, DATA[N].mag));
        end: PRINT_LINE

        printer.print_generic("", "", 0, RULING);
        printer.print_generic("", "", 0, $sformatf("TOTAL NUMBER OF TRIALS: %-8d", num_trials));
        printer.print_generic("", "", 0, {RULING, "\n"});
        printer.print_array_footer();
    endfunction: do_print

endclass: SCORECARD
```

Figure 11. The scorecard class for book-keeping the BER measurement results and printing the final JTOL characteristics.

IV. Simulation Results

The described UVM testbench along with the 16-Gb/s high-speed wireline transceiver model is run with Synopsys' VCS and Scientific Analog's XMODEL. Fig. 12 shows the simulation log printed by UVM, listing the results of the 106 BER measurement tests performed during the iterative search process. On a 64-bit Linux machine with 2.3-GHz 4-core Intel Core-i7 processor and 8-GB memory, it took a total of 41 minutes to measure the JTOL characteristics over 20 SJ frequency points ranging from 5-MHz to 5-GHz. It means that each BER test took about 23 seconds in average. The assumed initial locking time (*t_lock*) was 200ns and measurement time (*t_meas*) was 2μs. Thanks to the statistical simulation capability of XMODEL, it was possible to measure the BER as low as 10^{-9}~10^{-16} by running only 32,000 bit samples.

```
UVM_INFO @ 2210.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=5.000000e-01 UIpp --> BER=6.033595e-04
UVM_INFO @ 4411.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=4.000000e-01 UIpp --> BER=4.253071e-06
UVM_INFO @ 6612.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=3.000000e-01 UIpp --> BER=4.559507e-09
UVM_INFO @ 8813.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=2.000000e-01 UIpp --> BER=4.824476e-13
UVM_INFO @ 11014.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=2.449490e-01 UIpp --> BER=3.271928e-11
UVM_INFO @ 13215.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=2.213364e-01 UIpp --> BER=3.781672e-12
UVM_INFO @ 15416.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=2.103979e-01 UIpp --> BER=1.263946e-12
UVM_INFO @ 17617.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 20 SJ freq=5.000000e+09 Hz, mag=2.051331e-01 UIpp --> BER=7.470467e-13
UVM_INFO @ 19818.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 19 SJ freq=3.475964e+09 Hz, mag=2.051331e-01 UIpp --> BER=2.923724e-15
UVM_INFO @ 22019.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 19 SJ freq=3.475964e+09 Hz, mag=2.461597e-01 UIpp --> BER=8.445417e-14
UVM_INFO @ 24220.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 19 SJ freq=3.475964e+09 Hz, mag=2.871863e-01 UIpp --> BER=2.007209e-12
UVM_INFO @ 26421.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 19 SJ freq=3.475964e+09 Hz, mag=2.658829e-01 UIpp --> BER=4.560037e-13
UVM_INFO @ 28622.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | # 19 SJ freq=3.475964e+09 Hz, mag=2.763294e-01 UIpp --> BER=7.860811e-13

(... omitted for brevity ...)

  | #  2 SJ freq=7.192249e+06 Hz, mag=4.179040e+00 UIpp --> BER=2.324675e-19
UVM_INFO @ 213506.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  2 SJ freq=7.192249e+06 Hz, mag=5.014848e+00 UIpp --> BER=1.745598e-17
UVM_INFO @ 215707.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  2 SJ freq=7.192249e+06 Hz, mag=5.850657e+00 UIpp --> BER=1.286969e-12
UVM_INFO @ 217908.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  2 SJ freq=7.192249e+06 Hz, mag=5.416655e+00 UIpp --> BER=3.667517e-16
UVM_INFO @ 220109.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  2 SJ freq=7.192249e+06 Hz, mag=5.629475e+00 UIpp --> BER=4.189074e-15
UVM_INFO @ 222310.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=5.629475e+00 UIpp --> BER=1.387735e-20
UVM_INFO @ 224511.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=6.755370e+00 UIpp --> BER=4.093819e-19
UVM_INFO @ 226712.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=7.881265e+00 UIpp --> BER=1.972978e-15
UVM_INFO @ 228913.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=9.007160e+00 UIpp --> BER=2.232116e-03
UVM_INFO @ 231114.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=8.425427e+00 UIpp --> BER=2.148739e-11
UVM_INFO @ 233315.000ns: uvm_test_top.E.AGNT.DRV [RUN]
  | #  1 SJ freq=5.000000e+06 Hz, mag=8.148805e+00 UIpp --> BER=8.002244e-15
```

Figure 12. The UVM simulation log listing the BER tests being performed during the iterative search process.

Finally, Fig. 13 shows the scoreboard report listing the maximum tolerance SJ magnitudes collected for 20 SJ frequencies ranging from 5-MHz to 5-GHz and the plot of the corresponding JTOL characteristics. The measurement results show that the high-speed wireline receiver has the worst-case timing margin of $0.1894UI_{pp}$, and tracking bandwidth of ~200MHz. The former is indicated by the worst-case JTOL in the high-frequency range and the latter is by the knee point of the JTOL curve. For a bang-bang controlled CDR, a common cause of insufficient timing margin is the excessive dithering jitter, which can be mitigated by either reducing the phase step (e.g., enhancing the phase interpolator resolution) or using a decimation filter at the phase detector output. However, these remedies can reduce the tracking bandwidth of the CDR. A judicious design striking the balance between these two competing objectives is a key to the high-speed wireline receiver design.

```
-------------------------------------------
JITTER TOLERANCE (JTOL)
INDEX     FREQUENCY(Hz)   MAGNITUDE(UIpp)
-------------------------------------------
1         5.0000e+06      8.1488
2         7.1922e+06      5.6295
3         1.0346e+07      4.0418
4         1.4882e+07      2.8870
5         2.1407e+07      2.0728
6         3.0792e+07      1.4806
7         4.4293e+07      1.0576
8         6.3714e+07      0.7851
9         9.1649e+07      0.5858
10        1.3183e+08      0.4184
11        1.8963e+08      0.3106
12        2.7278e+08      0.2709
13        3.9238e+08      0.2834
14        5.6442e+08      0.2362
15        8.1189e+08      0.1894
16        1.1679e+09      0.2756
17        1.6799e+09      0.2996
18        2.4165e+09      0.3168
19        3.4760e+09      0.2763
20        5.0000e+09      0.2051
-------------------------------------------
TOTAL NUMBER OF TRIALS: 106
-------------------------------------------
```

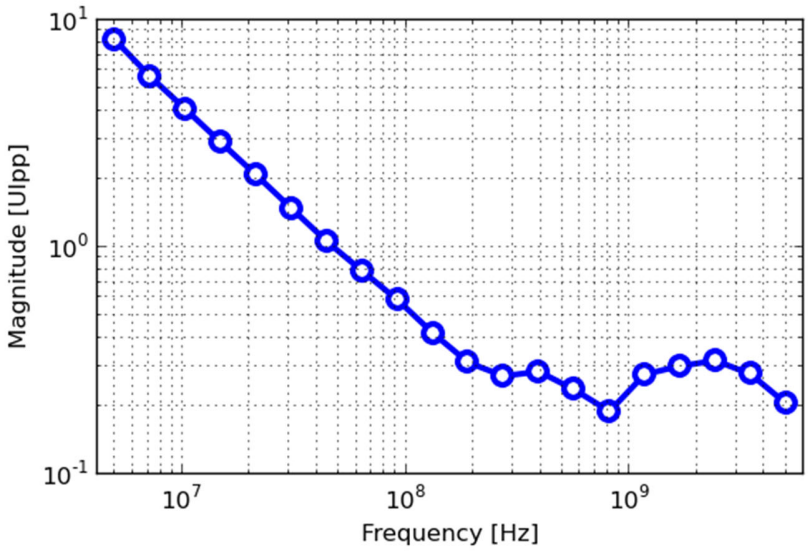


Figure 13. (a) The scoreboard report listing the maximum tolerable SJ magnitudes and (b) the plot of the corresponding JTOL characteristics.

V. Conclusion

This work demonstrated that the reactive stimulus technique of UVM can be extended to analog/mixed-signal verification, using an example of measuring the JTOL characteristics of a high-speed wireline receiver. The proposed UVM testbench effectively combines the advantage of UVM and XMODEL, by drawing a clear line between the two using the fixture module. The UVM part uses the reactive stimulus technique to find the maximum SJ magnitude meeting the target BER via an iterative search algorithm. And the XMODEL part enclosed within the fixture module uses the event-driven analog simulation algorithm and statistical simulation algorithm to estimate the BER of the high-speed wireline receiver model in SystemVerilog. As a result, the presented UVM testbench was able to measure the 20-point JTOL characteristics in only 41 minutes, while performing a total of 106 BER measurements. We are confident that this approach can be extended to other analog/mixed-signal verification cases as well.

Acknowledgment

This work was supported by Samsung Electronics Co., Ltd. The EDA tools were supported by the IC Design Education Center (IDEC), Korea and Scientific Analog, Inc.

References

[1] D. Derickson and M. Muller, *Ed.*, *Digital Communications Test and Measurement: High-Speed Physical Layer Characterization*, Prentice Hall, 2007.

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

[3] C. Dančak, "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.

[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.