2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION TAIWAN

Debugging RTL with Transactions

Small, simple changes enabling higher level understanding

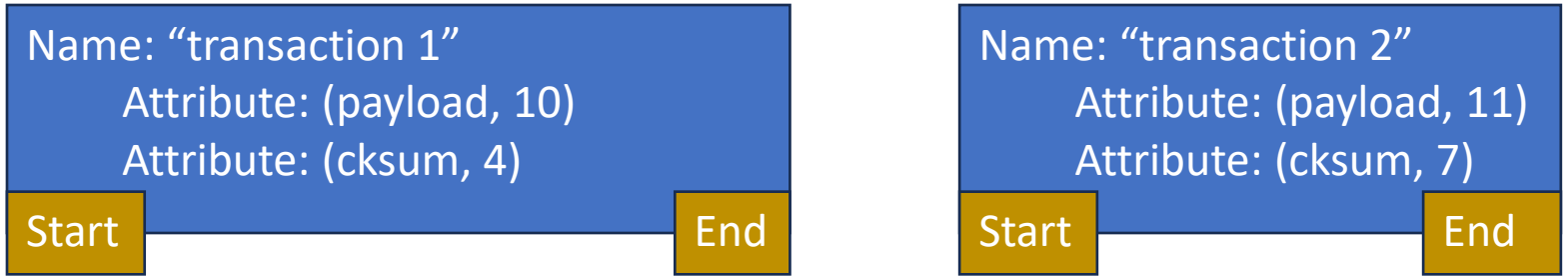
Rich Edelman, Siemens EDA, Fremont, CA US
Tsung-Yu Tsai, Siemens Taiwan, HsinChu City, Taiwan

SIEMENS

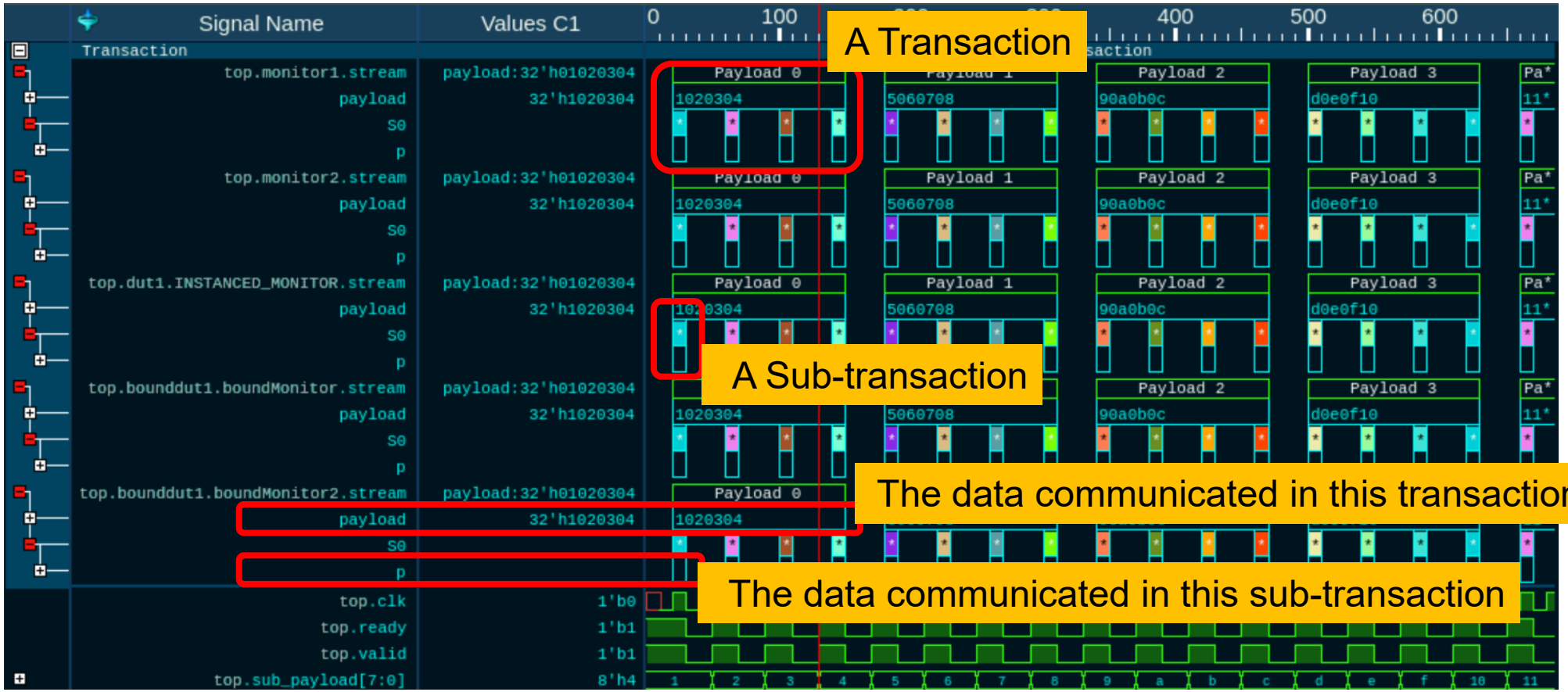
accellera SYSTEMS INITIATIVE

Transaction Data Model

- A transaction is “communication” of some kind
- In this paper – communication is a ready/valid pair with a clock and a payload.
- A transaction has a start and end.
- A transaction has “attribute” or “values-being-communicated”



Transaction Viewing Model



Simple Transaction Recording API

```
int stream;
int transaction;
int parent_transaction;

string kind, transaction_name, attribute_name, relationship_name;
time begin_time, end_time,

stream = $create_transaction_stream(stream_name, kind)

transaction = $begin_transaction(stream, transaction_name, begin_time, parent_transaction)
$end_transaction(transaction, end_time)
$free_transaction(transaction)

$add_attribute(transaction, value, attribute_name)
$add_relation(source_transaction, target_transaction, relationship_name)
$add_color(transaction, color)
$add_potential_link(transaction, signal)
```

Building a Simple Stream

Just Using the API

```
module simple();
  int stream_handle,
      full_tr_handle, sub_tr_handle;
  bit [31:0] full_payload;
  bit [7:0] payload;

    initial begin
      stream_handle = $create_transaction_stream("stream", "kind");
    end
```

Building a Simple Stream (continued)

Just Using the API

```
initial begin
  #20;
  forever begin
    full_tr_handle = $begin_transaction(stream_handle, $sformatf("Payload %0d", tr_count));
    full_payload = 0;
    for(int payload_count = 0; payload_count < 4; payload_count++) begin
      payload++;
      if (payload_count == 0) tr_count++;
      sub_tr_handle = $begin_transaction(stream_handle,
        $sformatf("tr%0d", payload_count), , full_tr_handle);
      $add_attribute(sub_tr_handle, payload, "p");
      full_payload = {full_payload, payload};
      #10;
      $end_transaction(sub_tr_handle);
      $free_transaction(sub_tr_handle);
       if (payload_count < 3) #30;
    end
    $add_attribute(full_tr_handle, full_payload, "payload");
    $end_transaction(full_tr_handle);
    $free_transaction(full_tr_handle);
    #30;
  end
end
endmodule
```

UVM Transactions – Field Automation

```
class transaction_using_fa extends uvm_sequence_item;
  rand bit [3:0] addr;
  rand bit [3:0] data;

  `uvm_object_utils_begin(transaction)
    `uvm_field_int(addr, UVM_DEFAULT)
    `uvm_field_int(data, UVM_DEFAULT)
  `uvm_object_utils_end
  ...
endclass
```

UVM Transactions – No Field Automation

```
class transaction extends uvm_sequence_item;
  `uvm_object_utils(transaction)
   rand bit [3:0] addr;
   rand bit [3:0] data;
   ...
  function void do_record(uvm_recorder recorder);
    super.do_record(recorder);
    if (data[0])
      // $add_color(recorder.get_handle(), "Pink"); // UVM 1.2 or greater
      $add_color(recorder.tr_handle, "Pink");
    else
      $add_color(recorder.tr_handle, "Purple");
    `uvm_record_field("name",  get_name());
    `uvm_record_field("addr",  addr);
    `uvm_record_field("data",  data);
  endfunction
endclass
```

Instrumenting a Verilog module

Building a Monitor

```
module VIP_MONITOR_MODULE(input clk, reg ready, reg valid, bit [7:0] payload);
// 8 bits at a time.
// 4 clocks of 8 bits = full payload (32 bits)
int stream_handle;
int tr_count;

initial begin
  stream_handle = $create_transaction_stream("stream", "kind");
  $add_potential_link(stream_handle, clk);
  $add_potential_link(stream_handle, ready);
  $add_potential_link(stream_handle, valid);
  $add_potential_link(stream_handle, payload);
end
```

Instrumenting a Verilog module

Building a Monitor

```
always @(posedge clk) begin
  int full_tr_handle;
  int sub_tr_handle;

  bit [31:0] full_payload;
  bit [7:0] payload_count;

  if ((ready == 1) && (valid == 1)) begin
    // This is a "transaction"
    full_tr_handle = $begin_transaction(stream_handle,
      $sformatf("Payload %0d", tr_count++));
     full_payload = 0;
    payload_count = 0;
    forever begin // 1, 2, 3, 4
      if ((ready == 1) && (valid == 1)) begin
        payload_count++;
        sub_tr_handle = $begin_transaction(
          stream_handle,
          $sformatf("tr%0d", payload_count),,
          full_tr_handle);
        $add_attribute(sub_tr_handle, payload, "p");
        full_payload = {full_payload, payload};
        @(negedge clk);
        $end_transaction(sub_tr_handle);
        $free_transaction(sub_tr_handle);
      end
      if (payload_count >= 4) break;
      @(posedge clk);
    end
    $add_attribute(full_tr_handle,
      full_payload, "payload");
    $end_transaction(full_tr_handle);
    $free_transaction(full_tr_handle);
  end
end
```

Instrumenting interface (in a task)

Building a Monitor

```
interface VIP_INTERFACE(input clk, reg ready, reg valid, bit [7:0] payload);
  int stream_handle;
  int tr_count;

  initial begin
    stream_handle = $create_transaction_stream("stream", "kind");
    $add_potential_link(stream_handle, clk);
    $add_potential_link(stream_handle, ready);
    $add_potential_link(stream_handle, valid);
    $add_potential_link(stream_handle, payload);

    fork
      monitor(); // Start the thread
    join_none
  end
```

Instrumenting interface (in a task)

Building a Monitor

```
task monitor();
   int full_tr_handle, sub_tr_handle;
   bit [31:0] full_payload;
   bit [7:0] payload_count;

   forever @(posedge clk) begin
     if ((ready == 1) && (valid == 1)) begin
       // This is a "transaction"
       full_tr_handle = $begin_transaction(
         stream_handle, $sformatf("Payload %0d",
           tr_count++));
       full_payload = 0;
       payload_count = 0;
       forever begin
         if ((ready == 1) && (valid == 1)) begin
           payload_count++;
           sub_tr_handle = $begin_transaction(
             stream_handle,
             $sformatf("tr%0d", payload_count),,
             full_tr_handle);
           $add_attribute(sub_tr_handle,
             payload, "p");
           full_payload = {full_payload,
             payload};
           @(negedge clk);
           $end_transaction(sub_tr_handle);
           $free_transaction(sub_tr_handle);
         end
         if (payload_count >= 4) break;
         @(posedge clk);
       end
       $add_attribute(full_tr_handle,
         full_payload, "payload");
       $end_transaction(full_tr_handle);
       $free_transaction(full_tr_handle);
     end
   end
endtask
```

Instancing a Monitor

Using a Monitor

```
module DUT(input clk, reg ready, reg valid, bit [7:0] payload);
  VIP_INTERFACE INSTANCED_MONITOR(clk, ready, valid, payload);

  always @(posedge clk) begin
    if ((ready == 1) && (valid == 1)) begin
      if (payload == 8'h10) begin
        $display("Bingo!");
      end
    end
  end
endmodule
```

Using “bind”

Using a Monitor

```
bind boundDUT VIP_INTERFACE boundMonitor(clk, ready, valid, payload);
```

- This bind statement will cause an instance named “boundMonitor” to be created inside all instances of the boundDUT module. That boundMonitor instance will be of type “VIP_MODULE_MONITOR”

Binding a Monitor

Using a Monitor

```
module boundDUT(input clk, reg ready, reg valid, bit [7:0] payload);

  always @(posedge clk) begin
    if ((ready == 1) && (valid == 1)) begin
      if (payload == 8'h10) begin
        $display("Bingo!");
      end
    end
  end
endmodule
```

A Simulation Transaction Recording

- Instrumented
  - AES Gate-level
  - VHDL
  - Verilog

Questions

- All source available from the authors