2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

SystemVerilog Transactions, UVM and C
Correlation in a functional verification environment

Tomoki Watanabe, Siemens Japan, Tokyo, Japan
Rich Edelman, Siemens EDA, Fremont, CA US

SIEMENS

accellera SYSTEMS INITIATIVE

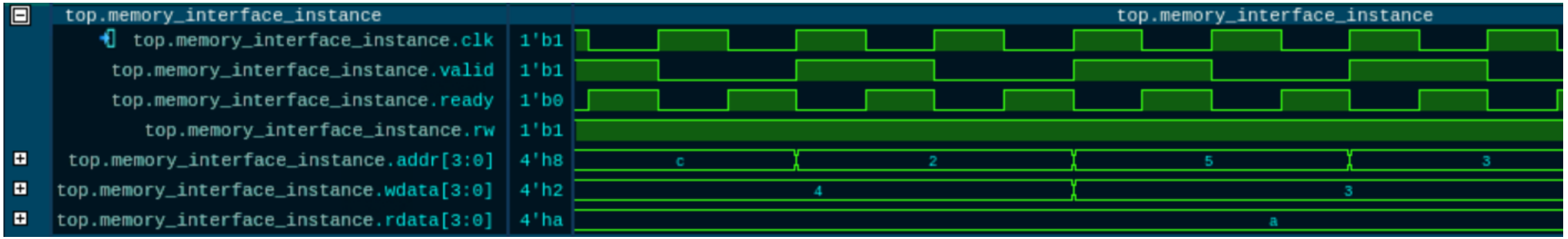
Verification Debugging - Correlation

- Waveform window – correlation by time
  - Waves
  - Transactions
  - Classes
  - C code
  - Verilog
  - VHDL

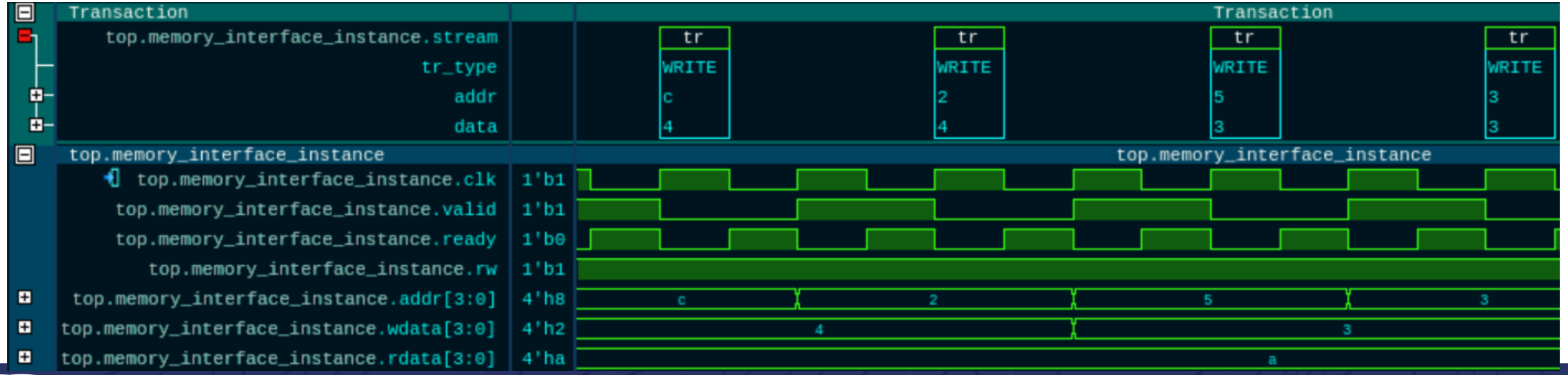
Correlation

Waves

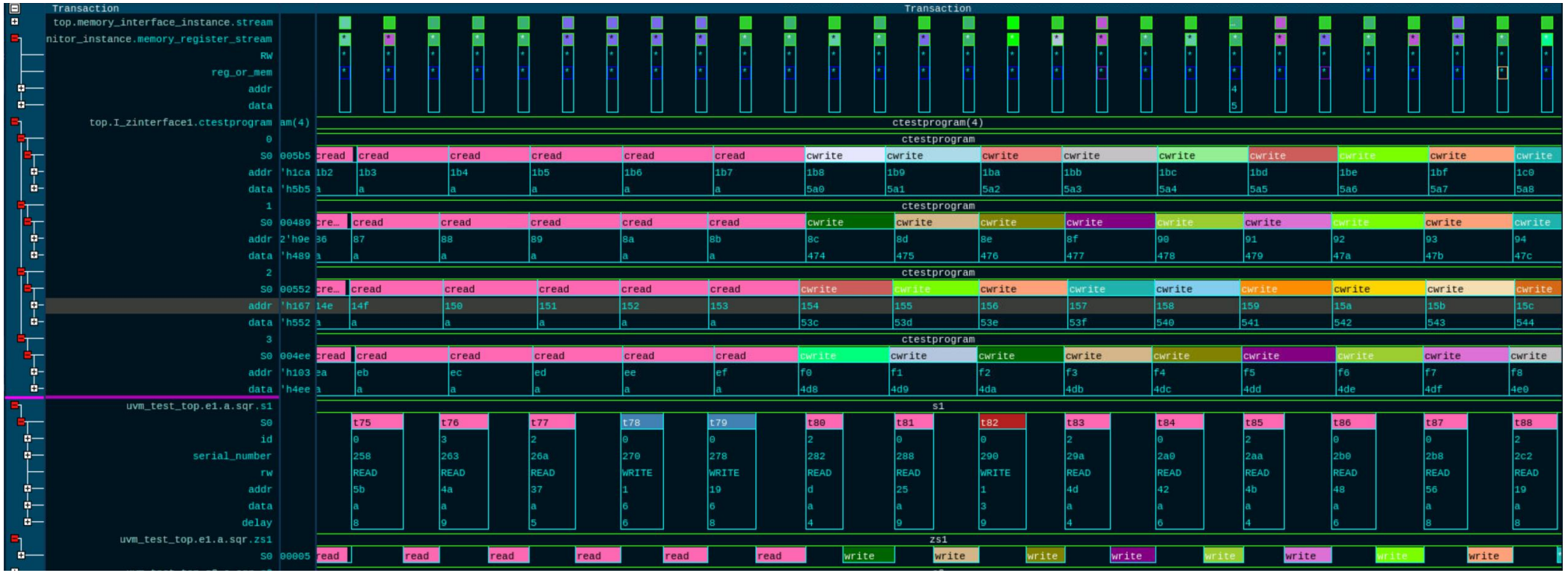
- add wave …



- add transactions …



Correlation – Coloring for “data”



How to Log Transactions – The PLI way

- $create_transaction_stream
- $begin_transaction
- $end_transaction
- $free_transaction
- $add_attribute

Generate some transactions

```
module M(input clk);
  int stream;

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

Simple Use of PLI Transaction Recording

```
module sub_channel(input clk, input int
inflight, int stream);
  always @(posedge clk) begin: LOOP
    ...
    begin
      req_rsp = REQ;
      tr = $begin_transaction(stream, ...);
      $add_attribute(tr, req_rsp, "req_rsp");
      $add_attribute(tr, addr, "addr");
      payload = 0;
      $add_attribute(tr, payload, "payload");
      $add_attribute(tr, id, "id");
      $add_attribute(tr, serial_number,
       "serial number");
      #10;
      $end_transaction(tr);
      $free_transaction(tr);
    end
```

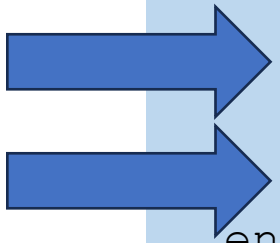
```
begin
  req_rsp = RSP;
  tr = $begin_transaction(stream, ...);
  $add_attribute(tr, req_rsp, "req_rsp");
  $add_attribute(tr, addr, "addr");
  payload = $random() % 1024;
  $add_attribute(tr, payload, "payload");
  $add_attribute(tr, id, "id");
  $add_attribute(tr, serial_number,
    "serial_number");
  delay = 10;
  #delay;
  #10;
  $end_transaction(tr);
  $free_transaction(tr);
end
end
endmodule
```

Just the REQ side

```
req_rsp = REQ;
tr = $begin_transaction(stream, ...);
$add_attribute(tr, req_rsp, "req_rsp");
$add_attribute(tr, addr, "addr");
payload = 0;
$add_attribute(tr, payload, "payload");
$add_attribute(tr, id, "id");
$add_attribute(tr, serial_number, "serial_number");
#10;
$end_transaction(tr);
$free_transaction(tr);
```

Building a Monitor

```
reg valid;
reg ready;
reg rw;
reg [3:0] addr;
reg [3:0] wdata;
reg [3:0] rdata;
```



```
forever begin
  @(posedge clk);
  if ((ready == 1) && (valid == 1)) begin
    @(posedge clk);
    if (rw == READ) // READ
      $display("MONITOR: %m: READ  addr=%0d, data=%0d", addr, wdata);
    else // WRITE
      $display("MONITOR: %m: WRITE addr=%0d, data=%0d", addr, rdata);
  end
end
```

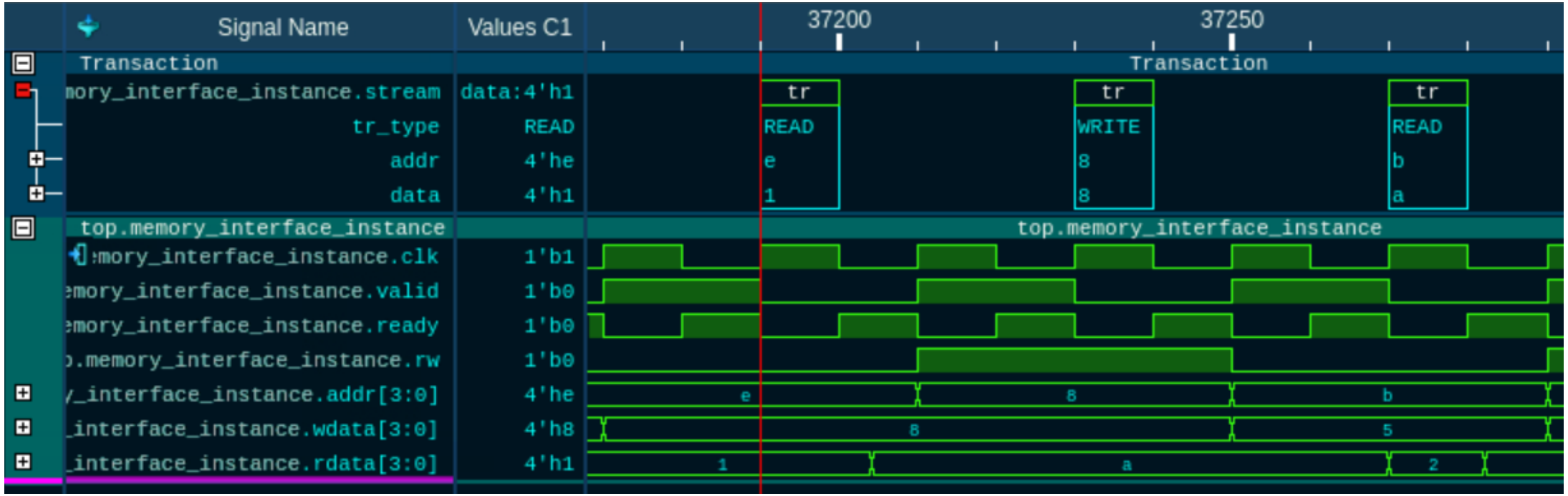
Building a Monitor

Signals on the BUS

```
reg valid;
reg ready;
reg rw;
reg [3:0] addr;
reg [3:0] wdata;
reg [3:0] rdata;
```

```
int stream;
int tr;
...
string tr_type;
stream = $create_transaction_stream("stream", "kind");
forever begin
  @(posedge clk);
  if ((ready == 1) && (valid == 1)) begin
    @(posedge clk);
    tr = $begin_transaction(stream, "tr");
    if (rw == READ) begin // READ
      tr_type = "READ";
      $add_attribute(tr, tr_type, "tr_type");
      $add_attribute(tr, addr, "addr");
      $add_attribute(tr, rdata, "data");
    end
    else begin // WRITE
      tr_type = "WRITE";
      $add_attribute(tr, tr_type, "tr_type");
      $add_attribute(tr, addr, "addr");
      $add_attribute(tr, wdata, "data");
    end
    @(negedge clk);
    $end_transaction(tr);
    $free_transaction(tr);
  end
end
```

A Monitored Set of Transactions



UVM

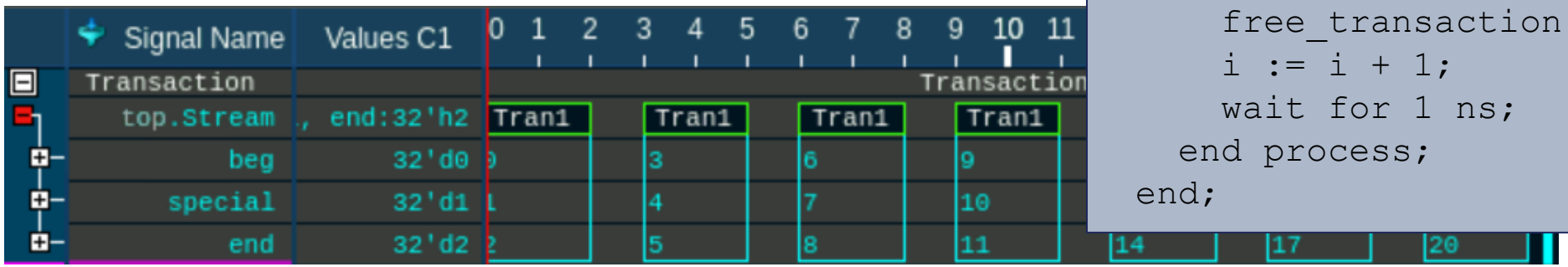
```
class transaction extends uvm_sequence_item;
  `uvm_object_utils(transaction)

  bit  [2:0] id; // 0 to 7
  bit [31:0] serial_number;

  rand int delay;

  rand RW_T rw;
  rand bit [31:0] addr;
  rand bit [31:0] data;

  function void do_record(uvm_recorder recorder);
    super.do_record(recorder);
    `uvm_record_field("id", id);
    `uvm_record_field("serial_number", serial_number);
    `uvm_record_field("rw", rw.name());
    `uvm_record_field("addr", addr);
    `uvm_record_field("data", data);
    `uvm_record_field("delay", delay);
  endfunction
endclass
```

Backdoor access

```
uvm_hdl_deposit("top.memory_instance.regA", 1);
uvm_hdl_deposit("top.memory_instance.regB", 2);
uvm_hdl_deposit("top.memory_instance.regC", 3);
#10;
uvm_hdl_deposit("top.memory_instance.regA", 2);
uvm_hdl_deposit("top.memory_instance.regB", 4);
uvm_hdl_deposit("top.memory_instance.regC", 6);
```

VHDL Transactions

```
entity top is
end;

library IEEE;
    use IEEE.std_logic_1164.all;

library modelsim_lib;
    use modelsim_lib.transactions.all;
```




```
architecture arch of top is
begin
  process
    variable stream : TrStream :=
       create_transaction_stream("Stream");
    variable tr: TrTransaction := 0;

    variable i : integer := 0;
  begin
    tr := begin_transaction(stream,"Tran1");
    add_attribute(tr, i, "beg");
    i := i + 1;
    wait for 1 ns;
    add_attribute(tr, i, "special");
    i := i + 1;
    wait for 1 ns;
    add_attribute(tr, i, "end");
    end_transaction(tr);
    free_transaction(tr);
    i := i + 1;
    wait for 1 ns;
  end process;
end;
```

Gate Level


```
module MAJ(output d, input a, b, c);
  and #4 ab_and(ab, a, b);
  and #4 bc_and(bc, b, c);
  and #4 ac_and(ac, a, c);
  or  #4   d_or(d , ab, bc, ac);
endmodule
```

```
module MAJ_monitor(input reg d, a, b, c, ab, bc, ac);
  int stream, tr;
  initial
    stream = $create_transaction_stream("gate_stream", "kind");
  always @(d) begin
    tr = $begin_transaction(stream, "MAJ_tr");
    $add_attribute(tr, {d, a, b, c}, "inputs_and_output");
    $add_attribute(tr, {ab, bc, ac}, "internal");
    #5;
    $end_transaction(tr);
    $free_transaction(tr);
  end
endmodule
```

```
module bind_module();
  bind MAJ MAJ_monitor monitor_instance(d, a, b, c, ab, bc, ac);
endmodule
```

Recording from C

```
export "DPI-C" function create_transaction_stream;
export "DPI-C" function begin_transaction;
export "DPI-C" function end_transaction;
export "DPI-C" function free_transaction;
export "DPI-C" function add_attribute_int;
```

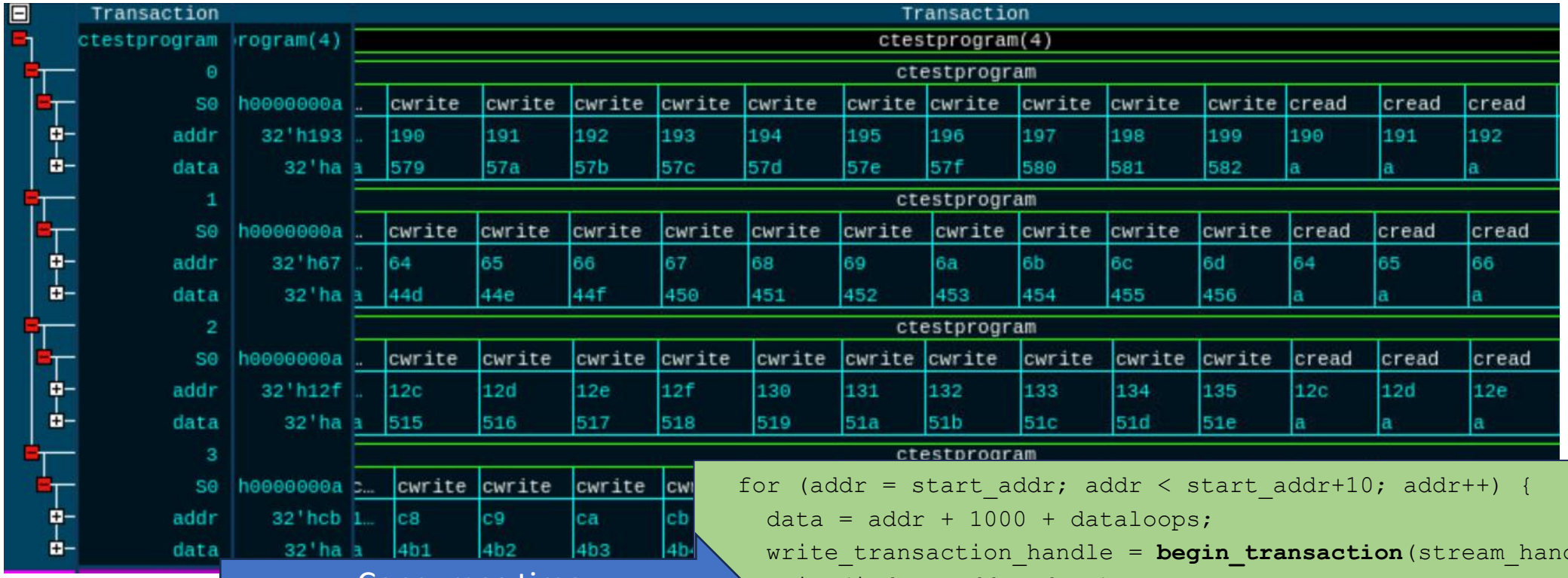
```
function int create_transaction_stream(string name, string kind);
  int stream_handle;
  stream_handle = $create_transaction_stream(name, kind);
  return stream_handle;
endfunction

function int begin_transaction(int stream_handle, string name, ...);
  int tr_handle;
  tr_handle = $begin_transaction(stream_handle, name);
  ...
  return tr_handle;
endfunction
```

```
function void end_transaction(int tr_handle);
  $end_transaction(tr_handle);
endfunction

function void free_transaction(int tr_handle);
  $free_transaction(tr_handle);
endfunction

function int add_attribute_int(int tr_handle, int value, string attribute_name);
  $add_attribute(tr_handle, value, attribute_name);
endfunction
```

C Test Program

```
for (addr = start_addr; addr < start_addr+10; addr++) {
  data = addr + 1000 + dataloops;
  write_transaction_handle = begin_transaction(stream_handle, "cwrite, ...);
  write(index, addr, data);
  add_attribute_int(write_transaction_handle, addr, "addr");
  add_attribute_int(write_transaction_handle, data, "data");
  end_transaction(write_transaction_handle);
  free_transaction(write_transaction_handle);
}
```

The C Test Transactions



Consumes time

```
for (addr = start_addr; addr < start_addr+10; addr++) {
  data = addr + 1000 + dataloops;
  write_transaction_handle = begin_transaction(stream_handle, "cwrite, ...);
  write(index, addr, data);
  add_attribute_int(write_transaction_handle, addr, "addr");
  add_attribute_int(write_transaction_handle, data, "data");
  end_transaction(write_transaction_handle);
  free_transaction(write_transaction_handle);
}
```

A Monitor for Registers and Memory

```
module memory_register_monitor(input clk, input valid, input ready, input rw,
    input bit [3:0] addr, input [3:0] wdata, input [3:0] rdata);

  rw_t rw_as_enum;
  reg [3:0] data;
  string rw_name;
  string name;
  int stream;
  int tr;

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

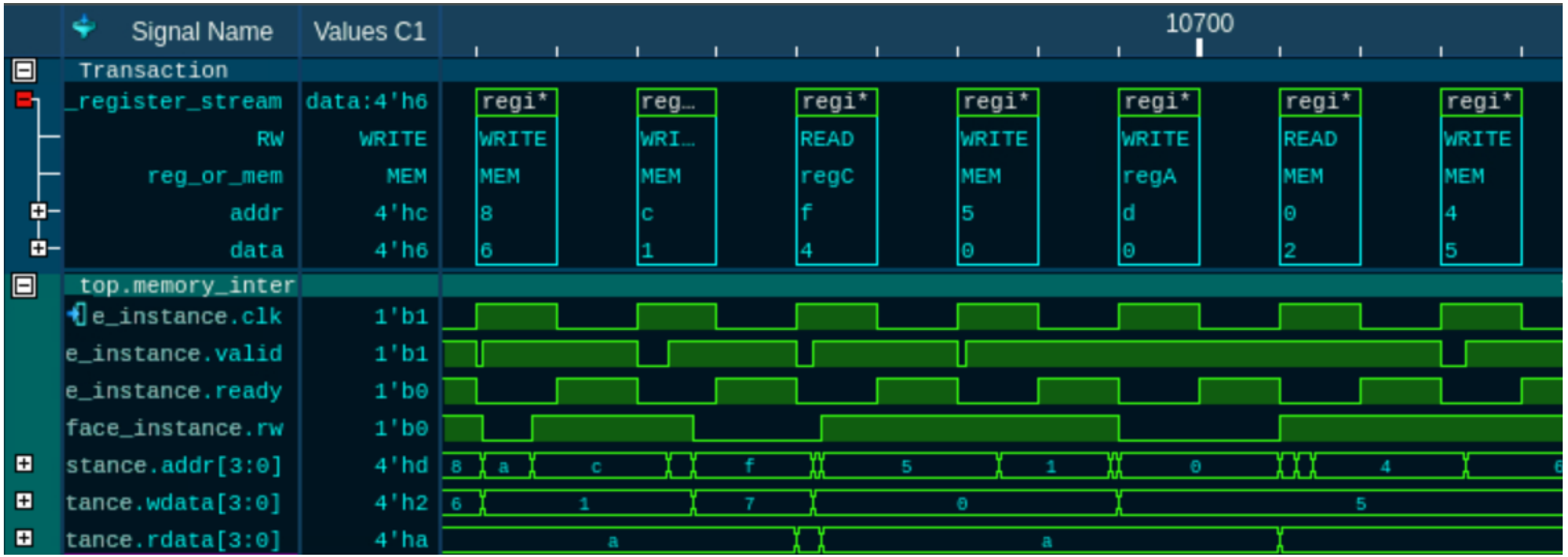
The READ or the WRITE phase

```
always @(posedge clk) begin
  rw_as_enum = rw_t'(rw);
  if ((ready == 1) && (valid ==1 )) begin
    tr = $begin_transaction(stream, "register");
    case (rw)
    READ: begin
      rw_name = "READ";
      data = rdata;
      case (addr)
        REGA: name = "regA";
        REGB: name = "regB";
        REGC: name = "regC";
        default: name = "MEM";
      endcase
    end
```

```
    WRITE: begin
      rw_name = "WRITE";
      data = wdata;
      case (addr)
        REGA: name = "regA";
        REGB: name = "regB";
        REGC: name = "regC";
        default: name = "MEM";
      endcase
    end
    endcase
    $add_attribute(tr, rw_name, "RW");
    $add_attribute(tr, name, "reg_or_mem");
    $add_attribute(tr, addr, "addr");
    $add_attribute(tr, data, "data");
    @(negedge clk);
    $end_transaction(tr);
    $free_transaction(tr);
    end
  end
```

Transactions for Registers and Memory

- Reads and Writes



Conclusion

- Correlation is a powerful debug technique.
- Correlation for time-based data is easy in the wave window.
- Transactions help abstract the details across interface
- C code can be recorded
- VHDL can be recorded
- Verilog can be recorded

Questions

- Source code available form the authors