2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Register Testing – Exploring Tests, Register Model Libraries, Sequences and Backdoor Access

Rich Edelman, Siemens EDA

Fremont, CA

SIEMENS

accellera SYSTEMS INITIATIVE

Introduction

- Register Testing – Exploring Tests, Register Model Libraries, Sequences and Backdoor Access
- Testing efficiency, transparency and simplicity

- Explore writing code
- Modeling decisions
- Where to put the complexity?

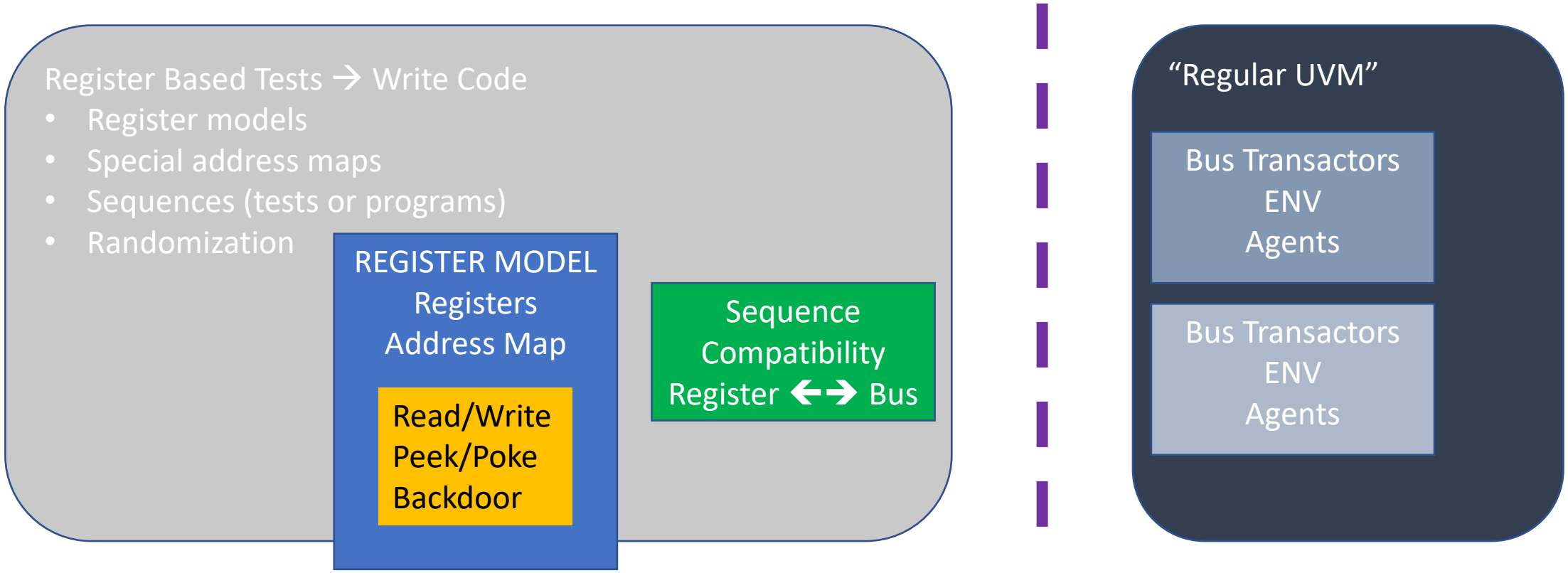
Background

- UVM Register package
  - Well used for many years
- Modeling decisions
  - Modeling fields (bits) with classes
- Complexity
  - 19k lines in total
  - 26 (27) files

uvm-1.1d	uvm-1.2	1800.2-2020-1.1	
1016	1017	952	src/reg/uvm_mem_mam.svh
2409	2409	2039	src/reg/uvm_mem.svh
251	253	255	src/reg/uvm_reg_adapter.svh
347	347	272	src/reg/uvm_reg_backdoor.svh
2271	2271	2372	src/reg/uvm_reg_block.svh
529	529	367	src/reg/uvm_reg_cbs.svh
2004	2012	1661	src/reg/uvm_reg_field.svh
310	310	292	src/reg/uvm_reg_fifo.svh
498	500	408	src/reg/uvm_reg_file.svh
329	329	316	src/reg/uvm_reg_indirect.svh
315	315	558	src/reg/uvm_reg_item.svh
2166	2224	2225	src/reg/uvm_reg_map.svh
443	442	449	src/reg/uvm_reg_model.svh
260	264	270	src/reg/uvm_reg_predictor.svh
546	547	421	src/reg/uvm_reg_sequence.svh
3094	3101	2601	src/reg/uvm_reg.svh
1004	1004	826	src/reg/uvm_vreg_field.svh
1549	1553	1227	src/reg/uvm_vreg.svh
307	307	310	src/reg/sequences/uvm_mem_access_seq.svh
299	299	303	src/reg/sequences/uvm_mem_walk_seq.svh
360	365	372	src/reg/sequences/uvm_reg_access_seq.svh
300	302	306	src/reg/sequences/uvm_reg_bit_bash_seq.svh
147	171	186	src/reg/sequences/uvm_reg_hw_reset_seq.svh
138	138	143	src/reg/sequences/uvm_reg_mem_built_in_seq.svh
174	174	178	src/reg/sequences/uvm_reg_mem_hdl_paths_seq.svh
485	485	497	src/reg/sequences/uvm_reg_mem_shared_access_seq.svh
21551	21668	19806	

register-report

Motivation – Use “registers” as part of tests



Abstract Thought Model

Write tests
- Write a register
- Read a register
- Compare/Check

Enabled by
- model of the register states
- address map

REGISTER MODEL
Registers
Address Map

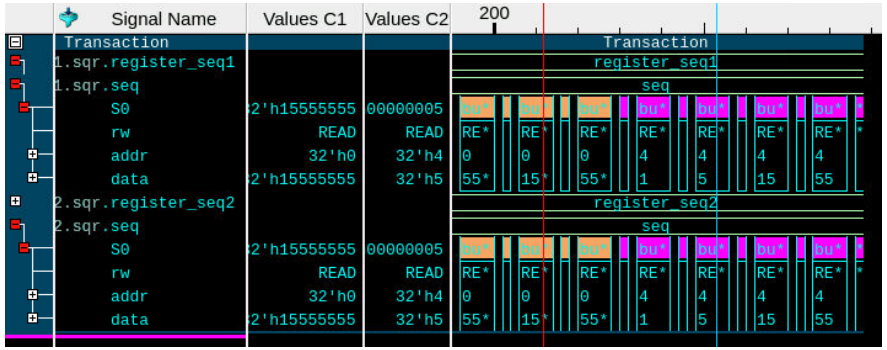
Read/Write
Peek/Poke
Backdoor

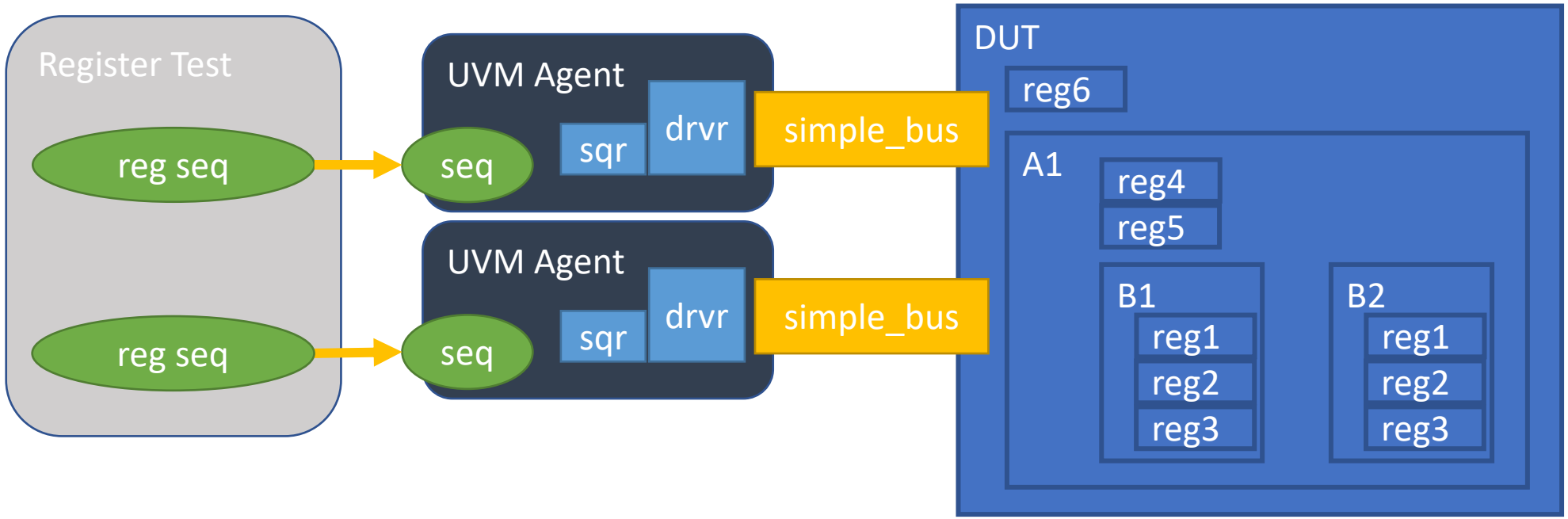
Sequence
Compatibility
Register ↔ Bus

Device Under Test

Many registers
Registers at addresses

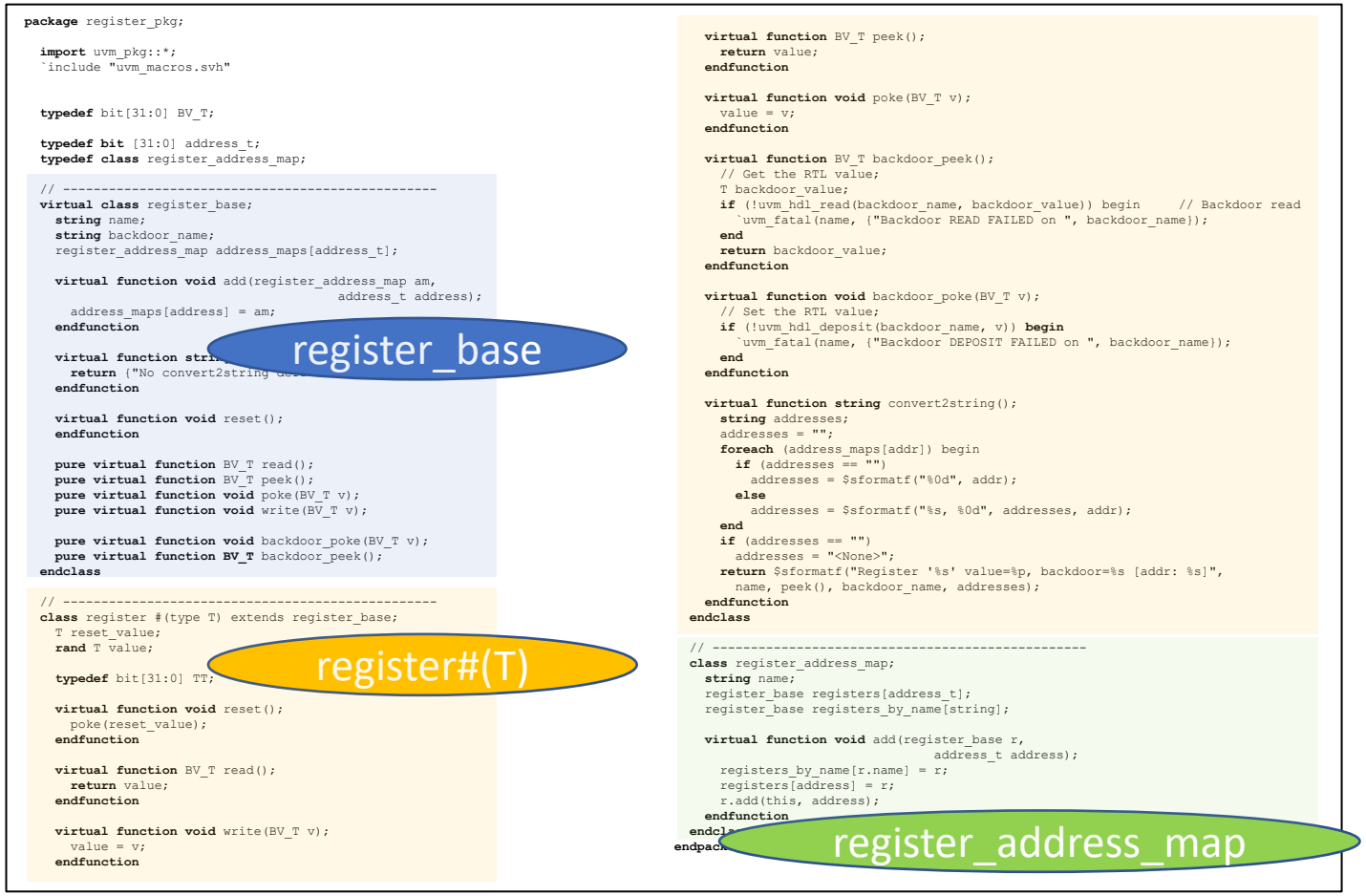
Our Verification Environment





The register_pkg

- 108 Lines of code



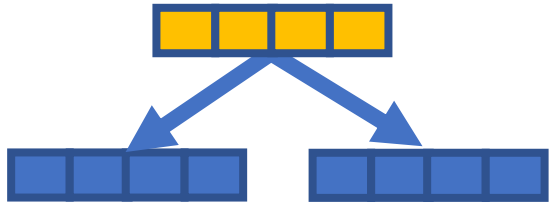
Clear on Read

- When READ, the value is returned
- And the register value is cleared

```
class clear_on_read_register #(type T) extends register#(T);

  function T read();
    T v;
    v = peek();
    poke('0);
    return v;
  endfunction
endclass
```

Broadcast Register



- Write to this register causes writes to “related” registers

```
class broadcast_register #(type T) extends register#(T);
  register#(T) targets[$];

  virtual function void add_target(register#(T) r);
    targets.push_back(r);
  endfunction

  function void write(T v);
    super.write(v);
    foreach (targets[i])
      targets[i].write(v);
  endfunction

  function void poke(T v);
    super.poke(v);
    foreach (targets[i])
      targets[i].poke(v);
  endfunction
endclass
```

Bus transactions – the way busses talk

- Read/Write
- Address
- Data

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

  bit [1:0] rw;    // WRITE: 0, READ: 1, IDLE: 3
  bit[31:0] addr;
  bit[31:0] data;

  function void do_record(uvm_recorder recorder);
    string rw_string;
    super.do_record(recorder);
    rw_string = (rw==0) ?"WRITE":"READ";
    `uvm_record_field("rw",   rw_string);
    `uvm_record_field("addr", addr);
    `uvm_record_field("data", data);
  endfunction
endclass
```

Register transactions – the way registers talk

- Read/Write
- Name
- Address
- Data
- Handle to a bus transaction

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

  bit [1:0]rw;
  string register_name;
  bit[31:0] addr;
  bit[31:0] data;

  uvm_sequence_item bus_tr;

  function string convert2string();
    return $sformatf("Register: %s : rw=%0b, addr=%3d, data=%b",
      register_name, rw, addr, data);
  endfunction
endclass
```

Register Base – The API

```
virtual class register_base;
   string name, backdoor_name;
   register_address_map address_maps[address_t];

   virtual function void add(register_address_map am, address_t address);
     address_maps[address] = am;
   endfunction

   virtual function string convert2string(); ...
   virtual function void reset(); ...

   pure virtual function BV_T read();
   pure virtual function BV_T peek();
   pure virtual function void poke(BV_T v);
   pure virtual function void write(BV_T v);

   pure virtual function void backdoor_poke(BV_T v);
   pure virtual function BV_T backdoor_peek();
 endclass
```

Register – class member variables + API

```
class register #(type T) extends register_base;
   T reset_value;
   rand T value;

   virtual function void reset();
     poke(reset_value);
   endfunction

   virtual function BV_T read();
     return value;
   endfunction

   virtual function void write(BV_T v);
     value = v;
   endfunction
```

```
virtual function BV_T peek();
  return value;
endfunction

virtual function void poke(BV_T v);
  value = v;
endfunction
```

Register – remaining API

```
virtual function BV_T backdoor_peek();
  T backdoor_value;
  uvm_hdl_read(backdoor_name, backdoor_value);
  return backdoor_value;
endfunction

virtual function void backdoor_poke(BV_T v);
  uvm_hdl_deposit(backdoor_name, v);
endfunction

virtual function string convert2string();
  string addresses;
  addresses = "";
  foreach (address_maps[addr])
    addresses = $sformatf("%s, %0d", addresses, addr);
  return $sformatf("Register '%s' value=%p, backdoor=%s [addr: %s]",
       name, peek(), backdoor_name, addresses);
endfunction
```

Register Address Map

By Address

Address	Register Base
2	@reg@1
4	@reg@2
6	@reg@3

By Name

Name	Register Base
reg1	@reg@1
reg2	@reg@2
reg3	@reg@3

```
class register_address_map;
   string name;
   register_base registers[address_t];
   register_base registers_by_name[string];

   virtual function void add(register_base r, address_t address);
     registers_by_name[r.name] = r;
     registers[address] = r;
     r.add(this, address);
   endfunction
 endclass
```

Register Sequence to Bus Sequence

- It's just a sequence – a gasket sequence
  - get from register talk to bus talk
- It has a special behavior
  - Start it, and it stays "running" – it is "open for business"
- Other sequences get a handle to it and "send transactions"
  - start_reg_item ➔ Like 'start_item'
  - finish_reg_item ➔ Like 'finish_item'

Each bus type has it's own

Register Sequence to Bus Sequence

```
class REGISTERtoBUS_rw_sequence extends uvm_sequence#(bus_transaction);
  `uvm_object_utils(REGISTERtoBUS_rw_sequence)

  bit all_done;

  task body();
    `uvm_info(get_type_name(), "...running", UVM_MEDIUM)
    all_done = 0;
    wait (all_done == 1);
  endtask
```

body() acts as a "Translator Thread"

Register Sequence to Bus Sequence

```
task start_reg_item(register_transaction register_tr);
  bus_transaction bus_tr;
  bus_tr = new("bus_tr");
  bus_tr.rw = register_tr.rw;
  bus_tr.addr = register_tr.addr;
  bus_tr.data = register_tr.data;
  start_item(bus_tr);
  register_tr.bus_tr = bus_tr; // Save it for later.
endtask

task finish_reg_item(register_transaction register_tr);
  bus_transaction bus_tr;
  $cast(bus_tr, register_tr.bus_tr);
  bus_tr.rw = register_tr.rw;
  bus_tr.addr = register_tr.addr;
  bus_tr.data = register_tr.data;
  finish_item(bus_tr);
  register_tr.data = bus_tr.data;
endtask
```

Translate to BUS

Translate back to Register

Translate to BUS

Translate back to Register

Register Sequence – how to program registers

```
class REGISTER_SEQUENCE#(type SEQUENCE = REGISTERtoBUS_rw_sequence)
        extends uvm_sequence#(uvm_sequence_item);
    `uvm_object_utils(REGISTER_SEQUENCE)

  register_address_map address_map;
  SEQUENCE seq;

  task startup();
    seq = new("seq");
    fork
      seq.start(m_sequencer);
    join_none
    #0;
  endtask
```

- Starts a “Register Sequence to Bus Sequence” for itself to use.
- Then it does whatever it wants.
  - Go get an address map – read and write registers
  - Check results

Register Sequence – the body

- The body(). It's just a sequence

```
task body();
  register_base reg_base;
  typedef bit[31:0] T;
  register #(T) reg_handle;

  register_transaction register_tr;

  string path;
  uvm_hdl_data_t backdoor_value;
  bit [31:0] register_model_value;

  bit [31:0] bv;
  bit [31:0] addr;
  int i;

  startup();
```

Register Sequence

- Address map
- Register handles
- Reading/Writing
- It's a "test"

```
typedef struct packed {
   bit [15:0] a;
   bit [15:0] b;
 } split_word_T;
```

This is a Register Test

```
register_address_map rm1;
register#(split_word_T) r; // Exact type
register_base b;
b = address_map.registers_by_name["reg1"];
$cast(r, b);
```

```
split_word_T test_values;
register_tr = new($sformatf("register_tr"));
register_tr.register_name = r.name;
register_tr.addr = addr;
r.value.a = 15;
r.value.b = 7;

test_values.a = 15;
test_values.b = 7;
r.poke(test_values);
r.write(test_values);

register_tr.data = b.peek();
seq.start_reg_item(register_tr);
seq.finish_reg_item(register_tr);
```

Setting the register fields directly

Setting a struct

Poking or writing the struct

Register Sequence – check all registers

- Iterate the address map
- Bit set
- Frontdoor
- Backdoor
- Check

```
foreach (address_map.registers[addr]) begin
  reg_base = address_map.registers[addr];
  register_tr = new($sformatf("register_tr%0d", i++));
  register_tr.register_name = reg_base.name;
  register_tr.addr = addr;
  for (int j = 0; j < 32; j++) begin
    register_tr.data[j++] = 1;                    // bit set
    register_tr.rw = 0;                           // WRITE
    reg_base.write(register_tr.data);             // write() => model
    seq.start_reg_item(register_tr);              // Frontdoor write
    seq.finish_reg_item(register_tr);
    register_model_value = reg_base.peek(); // peek() the model
    uvm_hdl_read(path, backdoor_value);           // Backdoor read
    if (backdoor_value != register_model_value) ... //Check
  end
```

This is a Register Test

Register Sequence - backdoor

- Backdoor poke(), peek() and compare

```
BV_T rbv, wbv;

// peek
rbv = reg_base.backdoor_peek();

wbv = rbv+2;

// poke
reg_base.backdoor_poke(wbv);

// Re-peek
rbv = reg_base.backdoor_peek();

if (rbv != wbv)
  `uvm_error(...
```

```
`uvm_info(get_type_name(), $sformatf("BACKDOOR READ (%s) = %b", path, wbv), UVM_MEDIUM)
```

```
# UVM_INFO tb_pkg.sv(204) @ 20200: uvm_test_top.e2.sqr@@register_seq2
    [REGISTER_SEQUENCE] BACKDOOR READ (top.dutA.A1.B1.reg3) = 0101010101010101010...0
```

This is a Register Model

A Register Model

- Just a class with class members
  - registers and address maps
- The constructor
  - Builds a register, sets its name, sets its backdoor name
  - Builds the address maps. Puts the register handles into the address map at the right address

```
class register_model;
  register#(split_word_T) reg1_1;

  simple_register reg1_2;
  simple_register reg1_3;
  simple_register reg4;
  simple_register reg5;
  simple_register reg6;
  register#(split_word_T) reg2_1;
  simple_register reg2_2;
  simple_register reg2_3;

  register_address_map address_map;

  register_base base_array[string];
  register_base b;
```

```
 typedef struct packed {
    bit [15:0] a;
    bit [15:0] b;
  } split_word_T;
typedef register#(bit[31:0]) simple_register;
```

A Register Model - constructor

- Constructor: create, set name, set backdoor name

```
function new();
  reg1_1 = new(); reg1_1.name = "reg1"; reg1_1.backdoor_name = "top.dutA.A1.B1.reg1";
  reg1_2 = new(); reg1_2.name = "reg2"; reg1_2.backdoor_name = "top.dutA.A1.B1.reg2";
  reg1_3 = new(); reg1_3.name = "reg3"; reg1_3.backdoor_name = "top.dutA.A1.B1.reg3";

  reg4   = new(); reg4.name   = "reg4";   reg4.backdoor_name = "top.dutA.A1.reg4";
  reg5   = new(); reg5.name   = "reg5";   reg5.backdoor_name = "top.dutA.A1.reg5";
  reg6   = new(); reg6.name   = "reg6";   reg6.backdoor_name = "top.dutA.reg6";

  reg2_1 = new(); reg2_1.name = "reg1"; reg2_1.backdoor_name = "top.dutA.A1.B2.reg1";
  reg2_2 = new(); reg2_2.name = "reg2"; reg2_2.backdoor_name = "top.dutA.A1.B2.reg2";
  reg2_3 = new(); reg2_3.name = "reg3"; reg2_3.backdoor_name = "top.dutA.A1.B2.reg3";
```

A Register Model – more constructor

- More Constructor
  - Create address map(s)
  - Set the address name name
  - Add the registers into the address map at the correct address

```
    address_map = new();
    address_map.name = "address_map";

    address_map.add(reg1_1,  0);
    address_map.add(reg1_2,  4);
    address_map.add(reg1_3,  8);
    address_map.add(reg4,   12);
    address_map.add(reg5,   16);
    address_map.add(reg1_3, 20);
    address_map.add(reg1_3, 24);
    address_map.add(reg1_3, 28);
    address_map.add(reg6,   32);
  endfunction
endclass
```

A Test – class members

- Two envs
- Two register sequences
- A register model
- That's it

```
class test extends uvm_test;
  `uvm_component_utils(test)

  env e1, e2;
  REGISTER_SEQUENCE register_seq1, register_seq2;

  register_model reg_model;
```

A Test - build

- build_phase
  - Two environments
  - Get the virtual interface connections
  - Build the register model

```
function void build_phase(uvm_phase phase);
  e1 = new("e1", this);
  e2 = new("e2", this);
  uvm_config_db#(virtual simple_bus)::get(this, "*", "busA", e1.bus)) ...
  uvm_config_db#(virtual simple_bus)::get(this, "*", "busB", e2.bus)) ...
  reg_model = new();
endfunction
```

A Test - run

- Start 2 threads
- In each thread
  - Construct a sequence
  - Pass the address map to the sequence
  - Run (start) the sequence

```
task run_phase(uvm_phase phase);
  phase.raise_objection(this);
  fork
    begin
      register_seq1 = new("register_seq1");
      register_seq1.address_map = reg_model.address_map;
      register_seq1.start(e1.sqr);
    end
    begin
      register_seq2 = new("register_seq2");
      register_seq2.address_map = reg_model.address_map;
      register_seq2.start(e2.sqr);
    end
  join
  phase.drop_objection(this);
endtask
```

The Top

```
interface simple_bus (input clk);
  reg [1:0] rw;
  reg [31:0] addr;
  reg [31:0] wdata;
  reg [31:0] rdata;
endinterface
```

```
module top();
  reg clk;

  simple_bus busA(clk);
  simple_bus busB(clk);

  DUT dutA(.clk(busA.clk),
    .rw1(busA.rw), .addr1(busA.addr), .wdata1(busA.wdata), .rdata1(busA.rdata),
    .rw2(busB.rw), .addr2(busB.addr), .wdata2(busB.wdata), .rdata2(busB.rdata)
  );

  initial begin
    uvm_config_db#(virtual simple_bus)::set(null, "*", "busA", busA);
    uvm_config_db#(virtual simple_bus)::set(null, "*", "busB", busB);
    run_test("test");
  end
endmodule
```

Register Modeling- ExploringFields, Registers and Address Maps

- DVCON Japan 2022
- Available from rich.edelman@siemens.com

Questions & Conclusions

Introduction

- Register Testing – Exploring Tests, Register Model Libraries, Sequences and Backdoor Access
- Testing efficiency, transparency and simplicity

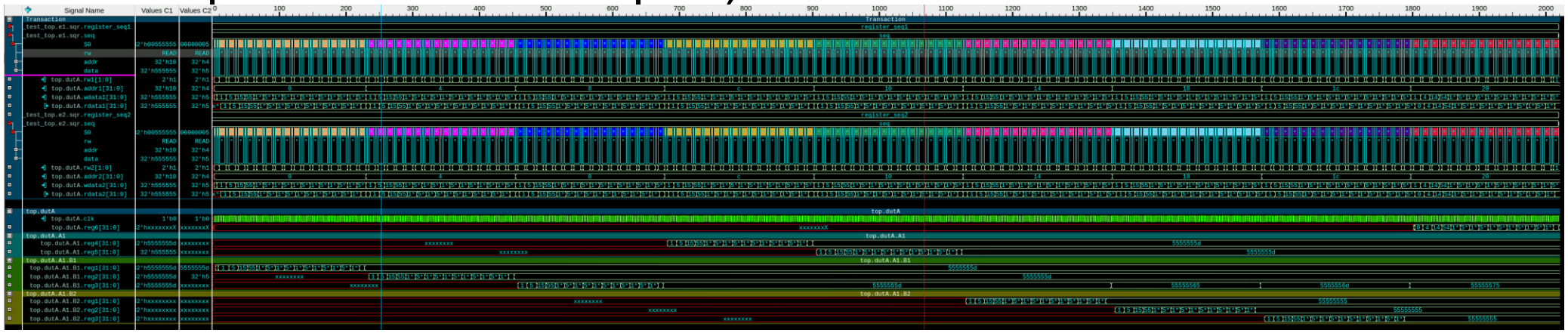
- Explore writing code
- Modeling decisions
- Where to put the complexity?

Simple code - Easy to understand models

Doesn’t replace all the UVM Register functionality

Avoided most complexity

- “translator sequence” is not complex, but it takes a minute to think about



Souce code available – Apache license – rich.edelman@siemens.com