2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Register Modeling – Exploring Fields, Registers and Address Maps

A Journey through Object Oriented Programming and Back

Rich Edelman

Siemens EDA, Fremont, CA US

SIEMENS

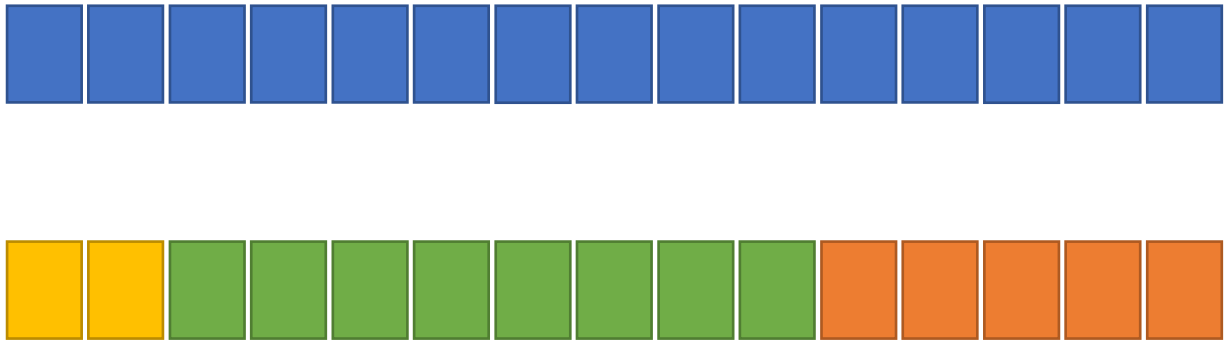
accellera SYSTEMS INITIATIVE

Introduction

- Goal
  - Explore SystemVerilog modeling using classes
  - Use a register, field and address map as our model
  - Try to build a “complete” model
  - Keep it simple
  - What You See Is What You Get

What’s a Register and a Field?

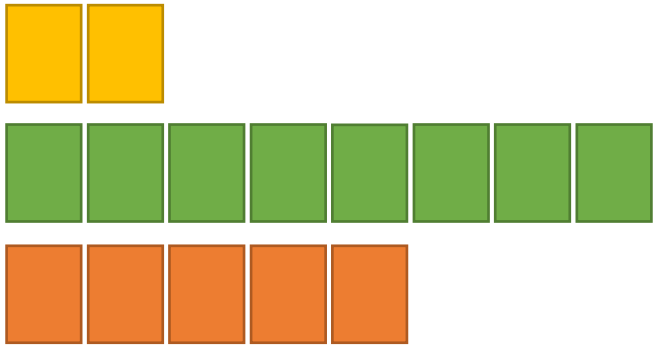
- Register has-a
  - Name
  - Backdoor name
  - Reset value
  - Value
  - Fields



Two ways to think about Fields

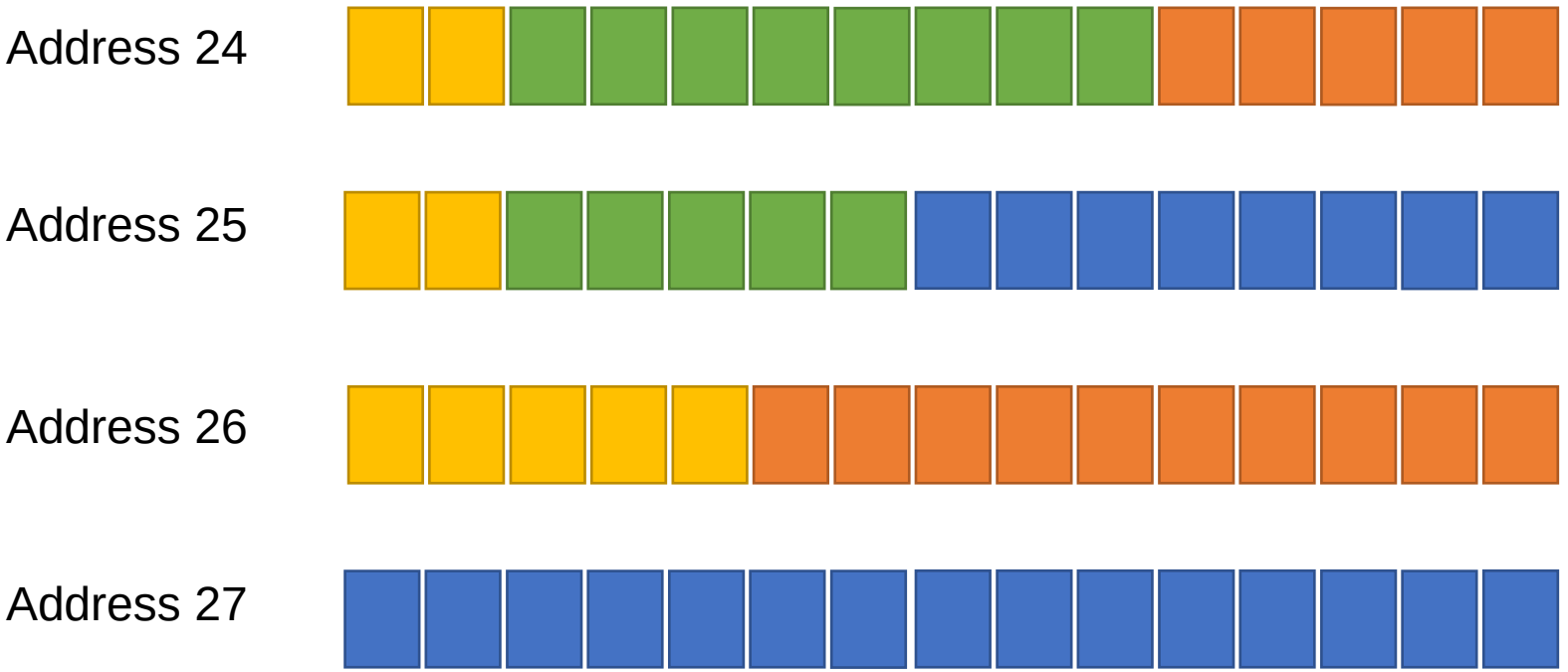
- A field is a piece of a register

- A field is important and will be used by-name
  - This is just a “small” register
  - Treat it like a register

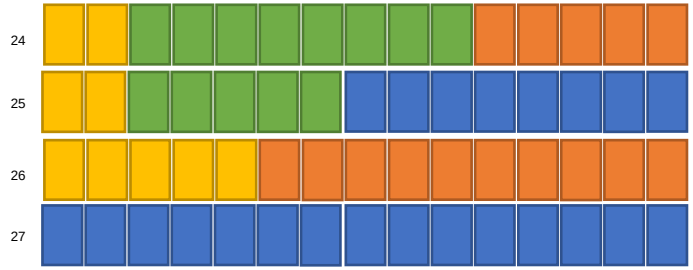


What's An Address Map?

- Define relationship between an address and a register



Testing a Register

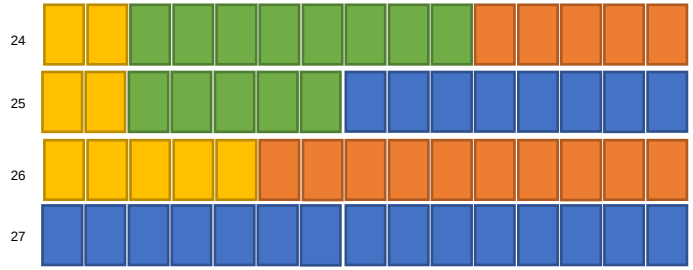


- Can it be written?
- Can it be read?
- Was every bit a 1?
- Was every bit a 0?
- Check field relationships
  - Was Field A = STP when Field B = STP
    - Check that BOTH A and B have the value STP at the same time

Testing an Address Map

- Does every legal address result in a register?
- Generate a list of all the registers in the address map
- Using the list of registers in the address map
  - Reset all the registers
  - Print the current values of the registers

Coverage in a Register



- Was it read?
- Was it written?
- Did each bit have the value 0 and 1?
- Were all fields read and written?
  - Once
  - More than once
- Did interesting combinations of fields occur?

A Register Model

- Modeling bits with classes is expensive
- Classes support lots of functionality
- Want a base class

```
pure virtual class register;
  rand T value;
  pure virtual function T read();
  pure virtual function   write(T v);
  pure virtual function T peek();
  pure virtual function   poke(T v);
  pure virtual function T backdoor_peek();
  pure virtual function   backdoor_poke(T v);
endclass
```

Operation	What it does
Read	Get the register **value**, execute any "special behavior"
Write	Set the register **value**, execute any "special behavior"
Peek	Get the register **value**, no special behavior
Poke	Set the register **value**, no special behavior
Backdoor Peek	Get the DUT register **value** using DPI/VPI/PLI
Backdoor Poke	Set the DUT register **value** using DPI/VPI/PLI

A Register Model w/Base Class

- Base class
- Type specific in extended class

```
virtual class register_base;
  string name;
  string backdoor_name;
  virtual function string convert2string();
  virtual function void reset();
endclass
```

```
class register #(type T)
      extends register_base;
  T reset_value;
  rand T value;
  virtual function void reset();
  virtual function T read();
  virtual function void write(T v);
```

```
  virtual function T peek();
  virtual function void poke(T v);
  virtual function T backdoor_peek();
  virtual function void backdoor_poke(T v);
  virtual function string convert2string();
endclass
```

The Register Base Class

```
typedef bit [31:0] address_t;

virtual class register_base;
  string name;
  string 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();
    return {"No convert2string defined for ", name};
  endfunction

  virtual function void reset();
  endfunction
endclass
```

Register ➔ **@r1**

name = “r1”

backdoor_name = “top.....r1”

Address Map Handle
@am1
@am2

The Register Class

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

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

  virtual function T read();
    return value;
  endfunction

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

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

  virtual function void poke(T v);
    value = v;
  endfunction

  virtual function T backdoor_peek();
  endfunction

  virtual function void backdoor_poke(T v);
  endfunction

  virtual function string convert2string();
    return $sformatf("%x", peek());
  endfunction
endclass
```

The Address Map Class

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

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

Address Map ➔ @am1
name = address_map1

Register Handle	Address
@r1	24
@r1	42
@r2	25

Address Map ➔ @am2
name = address_map2

Register Handle	Address
@r1	12

Access Modes or Permissions

- Just write (generate) the code
- A register with a READ-ONLY field
  - Means – “it cannot be written”

```
typedef struct {
  reg [1:0] status;
  reg [7:0] error_count;
  reg [4:0] active_tr; // READ-ONLY
} csr_t;

class csr_RO_active_tr extends register#(csr_t);
  function void write(T v);
    value.status = v.status;
    value.error_count = v.error_count;
  endfunction
endclass
```

Building a Register

```
typedef register#(bit[7:0]) simple_register;

typedef struct packed {
  reg [1:0] status;
  reg [7:0] error_count;
  reg [4:0] active_tr;
} csr_t;

simple_register       r1;
register #(bit[7:0]) r2;
register #(csr_t)    b1;
```

```
r1 = new();
r1.name = "r1";
r1.backdoor_name = "top.dut.blockA.blockB.r1";

r2 = new();
r2.name = "r2";
r2.backdoor_name = "top.dut.blockA.r2";

b1 = new();
b1.name = "b1";
b1.backdoor_name = "top.dut.blockA.blockB.b1";
```

Using a Register

- Test 1
  - Write a value
  - Read a value
  - Compare
- Test 2
  - Poke a value
  - Peek a value
  - Print the results
- Test 3
  - Reset the register
  - Peek a value
  - Print the results

```
r1.write(bv);
rbv = r1.read();
if (bv != rbv) // Automatic checking
  ...

// Check reset
r1.poke(bv);
$display("reg=%p, bv=%x", r1, bv); // Logging
bv = r1.peek();
$display("reg=%p, bv=%x", r1, bv);

r1.reset();
bv = r1.peek();
$display("reg=%p, bv=%x", r1, bv);
```

Printing a Register

```
$display("INFO: Register %s = %p", r1.name, r1.convert2string());
$display("INFO: Register %s = %p", r2.name, r2.convert2string());
$display("INFO: Register %s = %p", b1.name, b1.convert2string());
```

```
# INFO: Register r1 = "0"
# INFO: Register r2 = "0"
# INFO: Register b1 = "'{status:x, error_count:x, active_tr:x}"
```

```
virtual function string convert2string();
  return $sformatf("%p", peek());
endfunction
```

Id Register

- On each READ, return an integer from a list

```
// ---------------------------------------------
// Id Register
// Each time read() is called, return the next item in the list of Id's
// This register is read only
class id_register #(type T) extends register #(T);
  T values[$];
  int index; // -1 means empty. Otherwise points
             //    at the next to read.
             // 0 is the first
             // .size() is that last+1

  function new();
    super.new();
    index = -1;
  endfunction
```

Id Register (continued)

```
  function void add_id(T v);
    values.push_back(v);
    index=0;
  endfunction

  function T read();
    T value;
    value = values[index];
    index++;
    if (index >= values.size())
      index = 0;
    return value;
  endfunction

  function void write(T v);
  endfunction
```

```
  function T peek();
    return values[index];
  endfunction

  function void poke(T v);
  endfunction

  function T backdoor_peek();
    return super.backdoor_peek();
  endfunction

  function void backdoor_poke(T v);
  endfunction

  function string convert2string();
    return $sformatf("values=%p, index=%0d",
      values, index);
  endfunction
endclass
```

Id Register Tests

```
id_register #(int) r3;
...
initial begin
  r3 = new();
  r3.name = "r3";
  r3.backdoor_name = "top.dut.blockA.blockB.r3";
  r3.add_id(1);
  r3.add_id(2);
  r3.add_id(3);
  r3.add_id(4);

  begin : id_register_test
    int rv;
    repeat (10) begin
      rv = r3.read();
      $display("Id Register=%0d", rv);
    end
  end
```

Broadcast Register

- Write to this register causes writes to “related” registers
- This register cannot be READ

```
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

  function string convert2string();
    return $sformatf("targets=%p", targets);
  endfunction
endclass
```

Broadcast Register Tests

```
broadcast_register #(csr_t) r4;
          register #(csr_t) b1;
          register #(csr_t) b2;
          register #(csr_t) b3;
...
initial begin
  r4 = new();
  r4.name = "r4";
  r4.backdoor_name =
    "top.dut.blockA.blockB.r4";
  r4.add_target(b1);
  r4.add_target(b2);
  r4.add_target(b3);
```

```
begin : broadcast_test
    csr_t v;
    csr_t rv;
    v.status = 3;
    v.error_count = 12;
    v. active_tr = '1;
    b1.reset();
    b2.reset();
    b3.reset();

    r4.write(v);
    rv = b1.read();
    if (rv != v)
      ...
    rv = b2.read();
    if (rv != v)
      ...
    rv = b3.read();
    if (rv != v)
      ...
  end
```

Clear On Read Register

- 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
```

Size and Space and Complexity

	This Package	UVM Register
Lines	<100	>5000
Number of files	1	3
RAM Consumed	1x	10x
Time to build 1M	Seconds	Minutes
Simplicity	Simple	Complex
Completeness	Incomplete → NEW	Complete?

Modeling Choices

- Access routines
  - Some, but not 100%
  - But all variables can be directly accessed
- Separation of Concerns
  - The register model models registers and fields and address maps
  - It doesn’t know or care about sequences and bus adapters
- What You See Is What You Get
  - Simple
  - Transparent
  - Easy to read. Easy to change/update/enhance
- All register models get auto-generated – coding made easy

Future Work

- Complete the Register/Field/Adddress Map models
  - Using them in tests will help fill out any missing modeling
- Sequences
- Tests
  - For example
    - Write a value using the backdoor
    - Read the value using the frontdoor
    - Compare
- Backdoor implementations
  - Adapt/Adopt the UVM Register DPI/VPI/PLI
    - It is a general purpose “name lookup” – has nothing to do with registers

Summary

- Goal
  - ☑ Explore SystemVerilog modeling using classes
  - ☑ Use a register, field and address map as our model
  - ☑ Try to build a “complete” model
  - ☑ Keep it simple
  - ☑ What You See Is What You Get
- This Package
  - Very few lines of code. Easy to read
  - Fast
  - Small memory footprint
- Incomplete ➔ coming soon

Questions

- Contact the author – rich.edelman@siemens.com