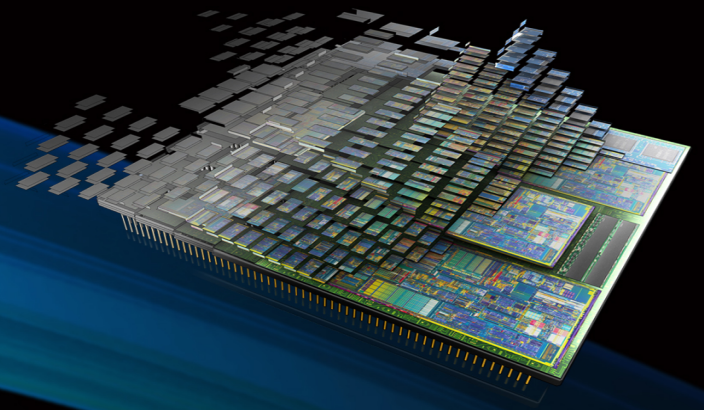
Verification Academy



Advanced UVM

Using the Register Layer

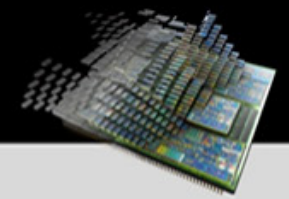
Tom Fitzpatrick

Verification Evangelist

info@verificationacademy.com | *www.verificationacademy.com*

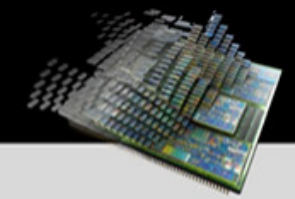
Mentor Graphics

UVM Register Use Models



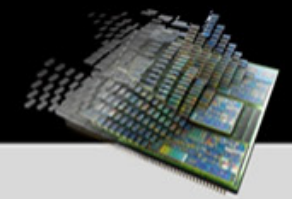
- **Stimulus Generation**
  - Firmware-level abstraction of stimulus:
    - i.e. Set this bit in this register rather than write x to address y
  - Stimulus reuse
    - If the bus agent changes, the stimulus still works
- **Configuration**
  - Register model reflects hardware programmable registers
  - Set up desired configuration in register model then dump to DUT
    - Randomization with configuration constraints
- **Analysis 'Mirror'**
  - Current state of the register model matches the DUT hardware
  - Useful for scoreboards and functional coverage monitors

UVM Register Class Access API



- **The register model has two register variables:**
  - **Desired value:** For when a field has been updated, but not the hardware
  - **Mirrored value:** Containing the latest known value
- **reg.read() and reg.write()**
  - Access the hardware register and update the register database
  - Front door access uses bus agent – takes time and may create side effects
  - Back door access is instant (via VPI) and does not cause side effects
  - Not used for individual fields
- **reg.peek() and reg.poke()**
  - For back door accesses, register model updated with result
  - Can be used for individual fields
- **reg.set() and reg.get()**
  - Access the desired value directly

Register Access Method Fields



Type	Name	Purpose
uvm_status_e	status	Indicates Access completed OK
uvm_reg_data_t	value	Data value transferred
uvm_path_e	***path***	Front or back door access
uvm_reg_map	***map***	Map to use for access
uvm_sequence_base	***parent***	Parent sequence
int	***prior***	Sequence priority on sequencer
uvm_object	***extension***	Transfer extension object
string	***fname***	Filename (For reporting)
int	***lineno***	Line number (For reporting)

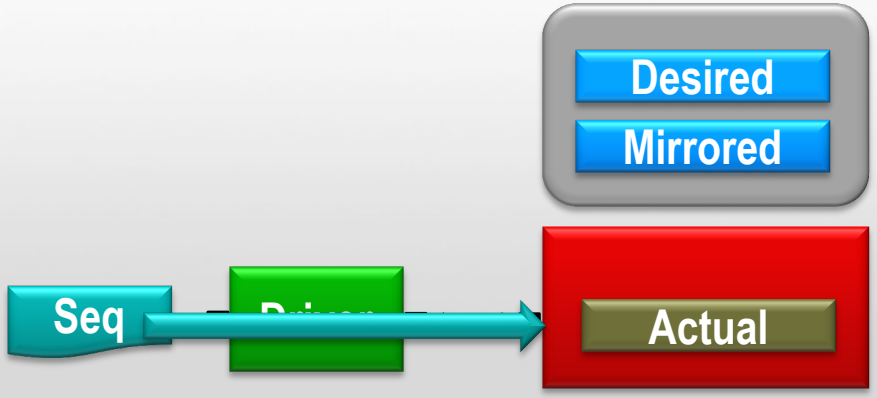
- **Good news – most of these fields have defaults!**
- **A typical register access only needs a few of these:**

```
spi_rm.ctrl.write(status, wdata, .parent(this));
```

Front-Door Access Modes

- **Consume time on the bus (default)**

```
spi_rm.ctrl.write(status, wdata, UVM_FRONTDOOR, .parent(this));
spi_rm.ctrl.read (status, rdata, UVM_FRONTDOOR, .parent(this));
```

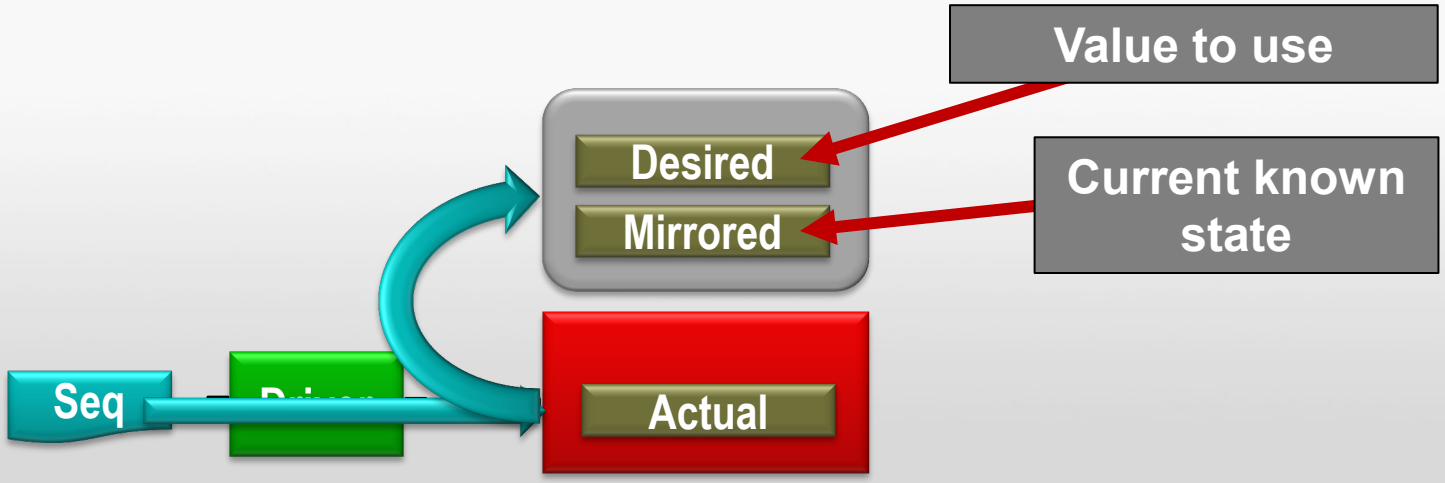


Front-Door Access Modes

- **Consume time on the bus (default)**

```
spi_rm.ctrl.write(status, wdata, UVM_FRONTDOOR, .parent(this));
spi_rm.ctrl.read (status, rdata, UVM_FRONTDOOR, .parent(this));
```

- **Desired and Mirrored values updated at end of transaction**
  - Based on transaction contents and field access mode

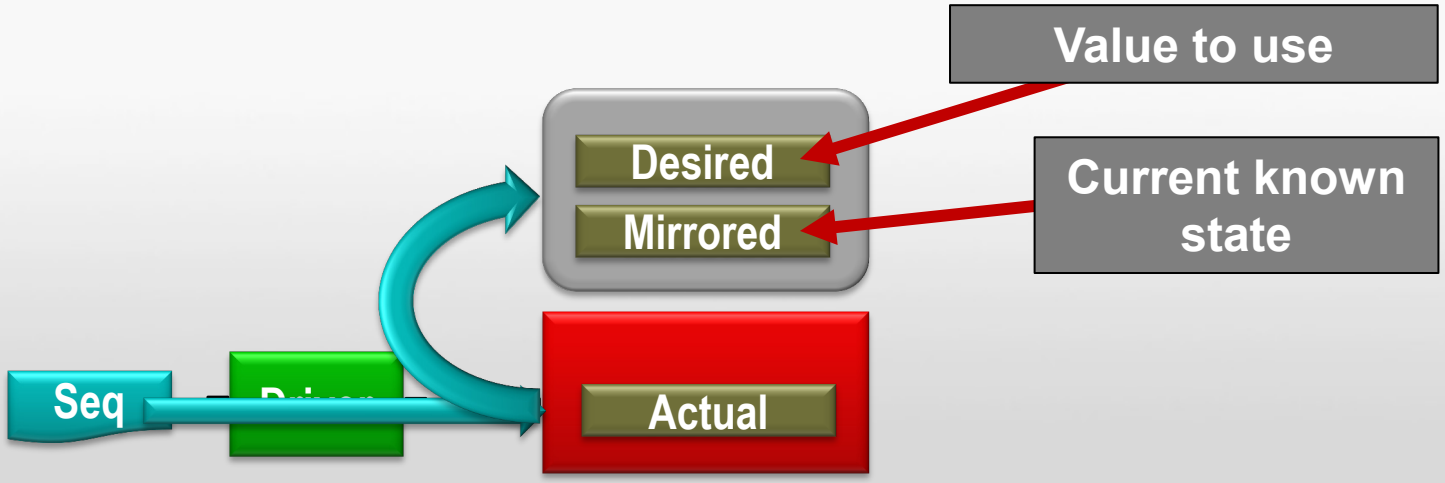


Front-Door Access Modes

- **Consume time on the bus (default)**

```
write_reg(model.ctrl, status, wdata, UVM_FRONTDOOR);
read_reg (model.ctrl, status, rdata, UVM_FRONTDOOR);
```

- **Desired and Mirrored values updated at end of transaction**
  - Based on transaction contents and field access mode

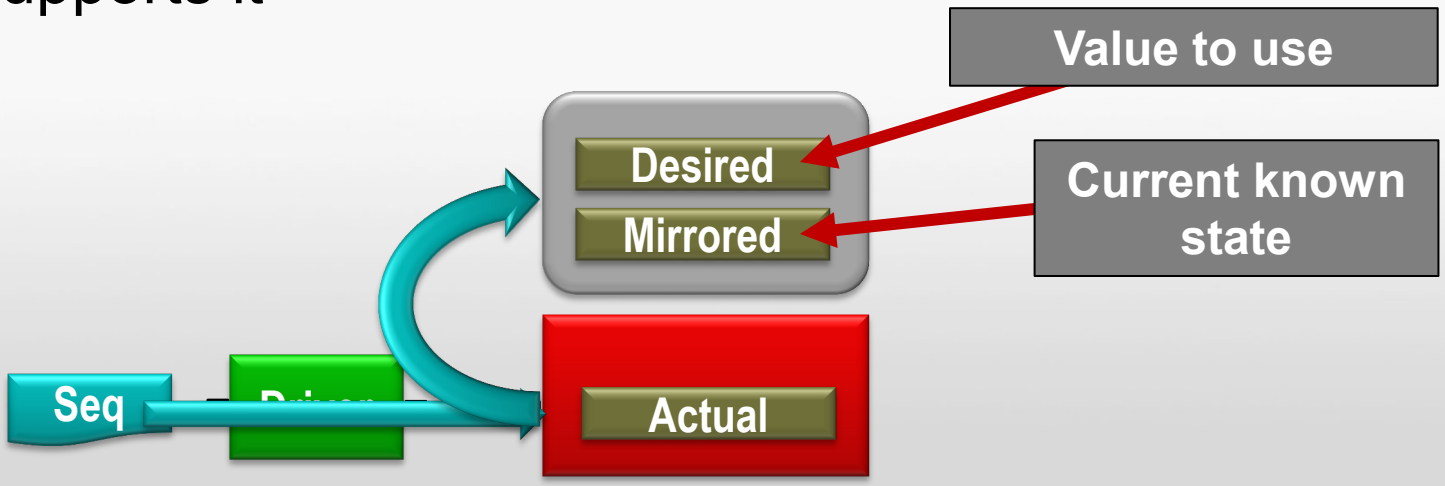


Front-Door Access Modes

- **Consume time on the bus (default)**

```
write_reg(model.ctrl, status, wdata, UVM_FRONTDOOR);
read_reg (model.ctrl, status, rdata, UVM_FRONTDOOR);
```

- **Desired and Mirrored values updated at end of transaction**
  - Based on transaction contents and field access mode
- **Can access individual fields**
  - Only if hardware supports it
    - Field = byte lane

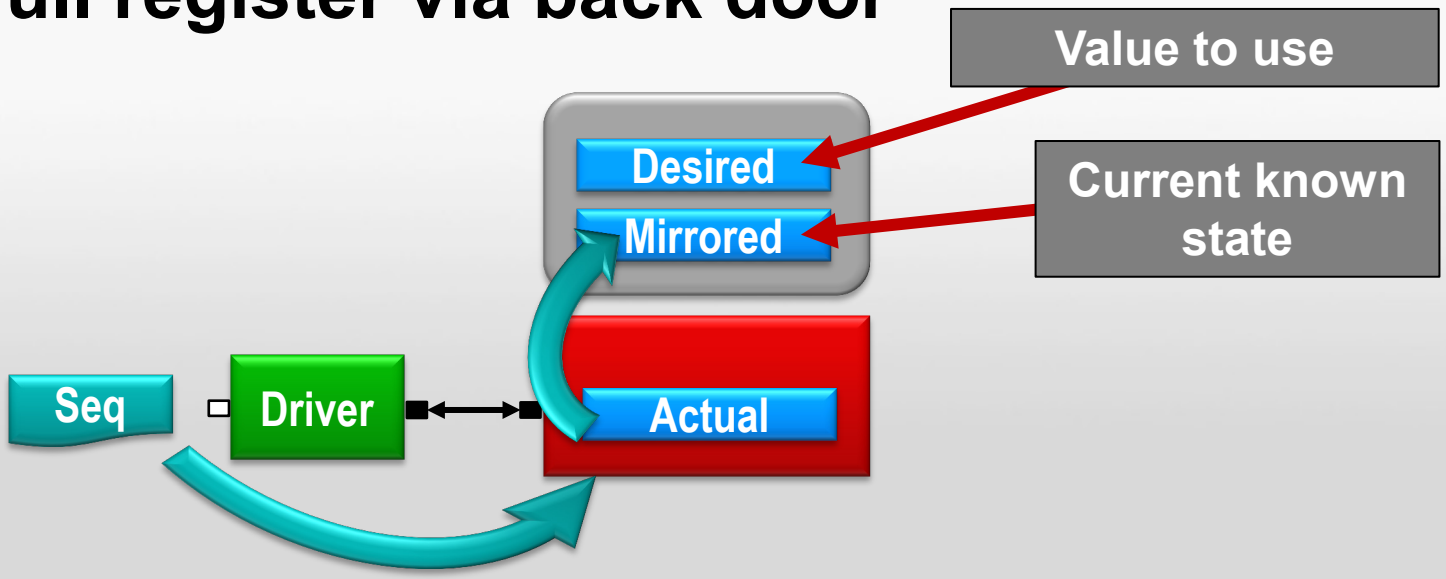


Back-Door Access Modes

- **Consume no time on the bus**

```
write_reg(model.ctrl, status, wdata, UVM_BACKDOOR);
read_reg (model.ctrl, status, rdata, UVM_BACKDOOR);
```

  - must be specified explicitly
- **Desired and Mirrored values updated at end of transaction**
  - Based on transaction contents and field access mode
- **Can only access full register via back door**

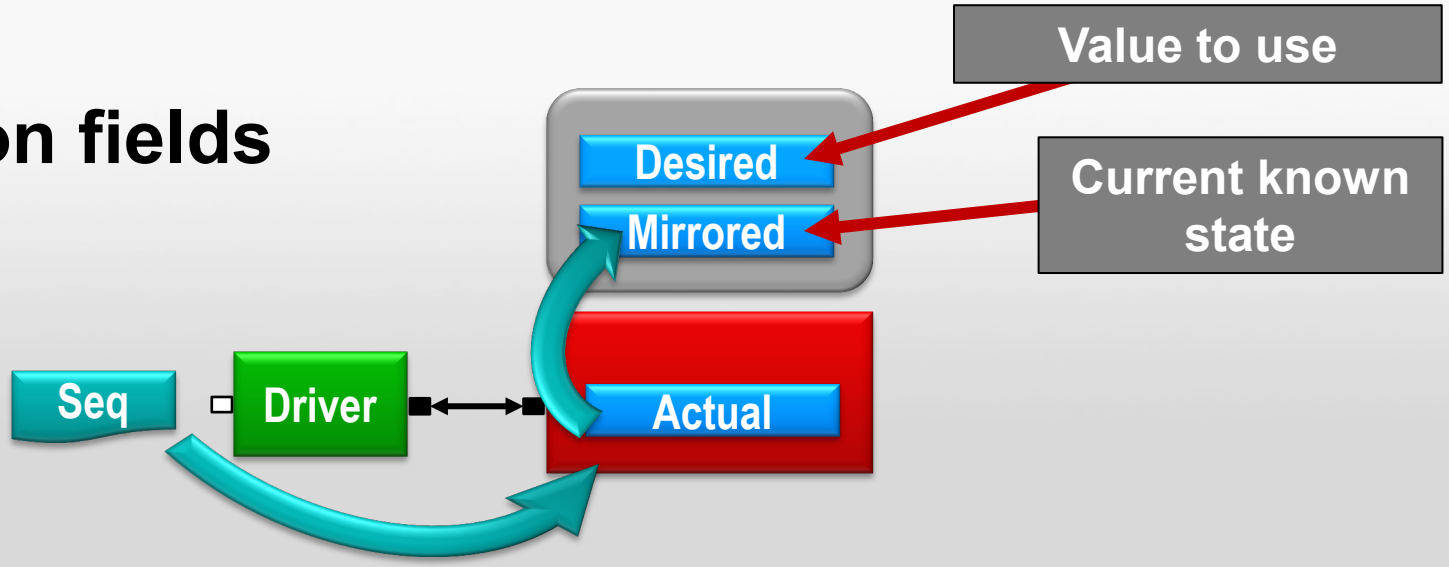


Back-Door Access Modes

- **Consume no time on the bus**

```
poke_reg (model.ctrl, status, wdata);
peek_reg (model.ctrl, status, rdata);
```

- **Desired and Mirrored values updated directly at end of transaction**
  - Poke sets the actual register value
  - Peek samples the actual value, which is written to model
- **Peek/Poke work on fields**



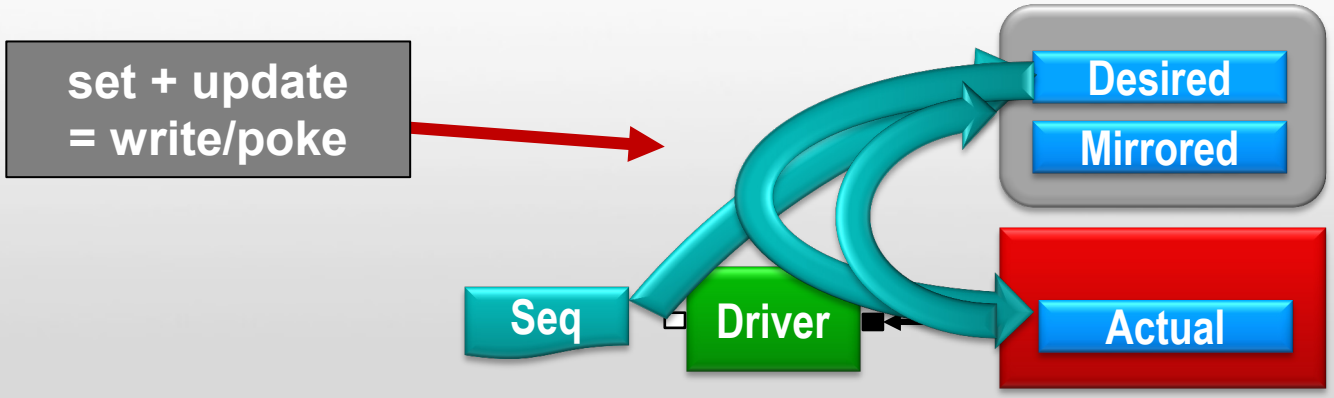
Direct Access Modes

- **Consume no time on the bus**
- **Access the Desired Value directly**

```
model.ctrl.set(wdata);
rdata = model.ctrl.get();
```

```
model.ctrl.randomize();
```

- **Use `update()` method to update actual value**
  - via frontdoor: `update_reg(model.ctrl, status, UVM_FRONTDOOR);`
  - or backdoor: `update_reg(model.ctrl, status, UVM_BACKDOOR);`



Mirror Method

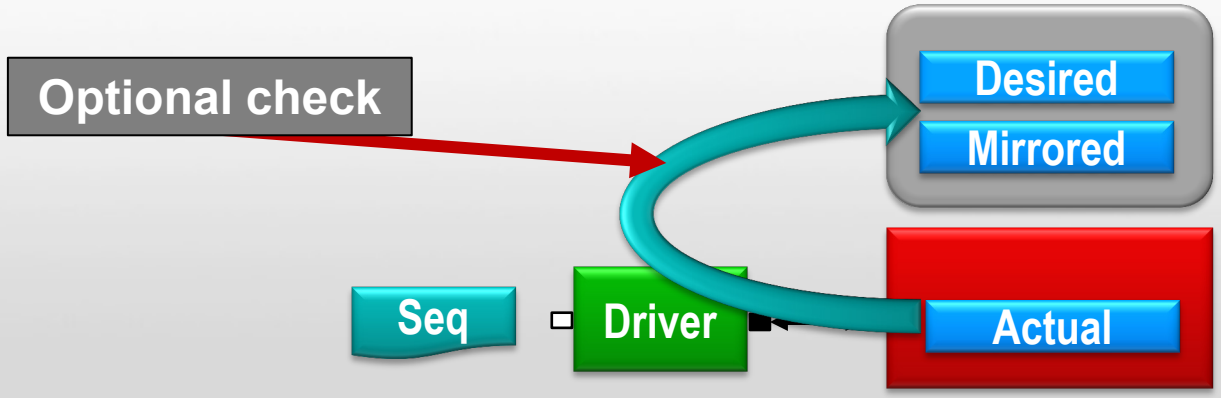
- **Read register and update/check mirror value**
  - via frontdoor

```
mirror_reg(model.ctrl, status, UVM_CHECK, UVM_FRONTDOOR);
```

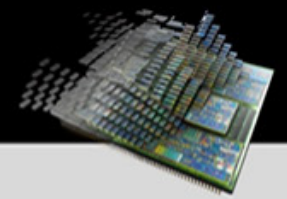
  - or backdoor

```
mirror_reg(model.ctrl, status, UVM_CHECK, UVM_BACKDOOR);
```

- **Can be called on field, reg or block**



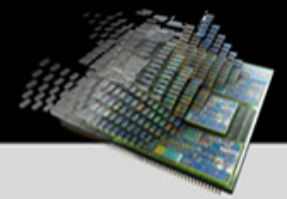
Register Sequence Base Class



```
class uvm_reg_sequence#(type BASE=uvm_sequence#(uvm_reg_item))
                       extends BASE;
  `uvm_object_param_utils(uvm_reg_sequence #(BASE))

  uvm_reg_block model;
  …
endclass
```

Default item type

Register Sequence Base Class



```
class blk_R_test_seq
                        extends uvm_reg_sequence;
  `uvm_object_utils(blk_R_test_seq)

  reg_block_B   model;

  function new(string name = "blk_R_test_seq");
    super.new(name);
  endfunction: new


  virtual task body();
    uvm_status_e status;
    uvm_reg_data_t data, rd_data;

    write_reg(model.R, status, data);
    read_reg (model.R, status, rd_data);
    …
  endtask
endclass
```

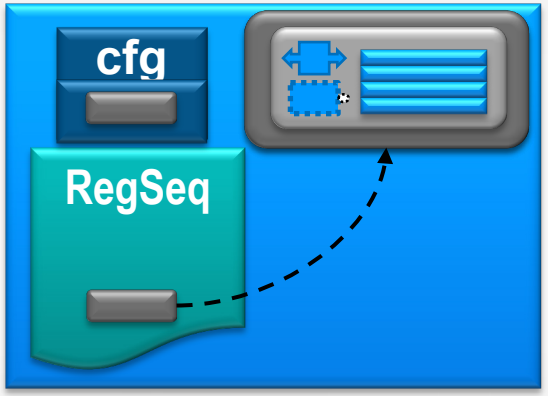
Set correct type

Provides convenience methods for reading/writing registers and memories

Register Stimulus: Base Class

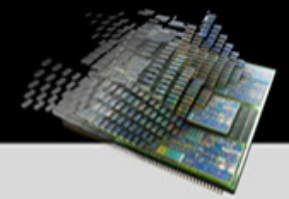
```
class spi_bus_base_seq extends uvm_reg_sequence;

  `uvm_object_utils(spi_bus_base_seq)

  spi_rm model;
  // SPI env config object (contains register model handle)
  spi_env_config m_cfg;

  // Properties used by the register access methods:
  rand  uvm_reg_data_t data; // For passing data
  uvm_status_e status;       // Return status

  task body;
    m_cfg = uvm_config_db #(spi_env_config)::get(null, get_full_name(),
                                                 "spi_env_config", m_cfg)
    model = m_cfg.spi_rm;
  endtask: body

endclass: spi_bus_base_seq
```

Base sequence contains variables common to all register sequences



Register model passed in via config_db
Can also be set directly

Register Stimulus: Building on the Base



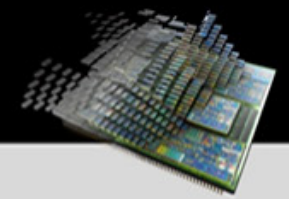
```
class div_load_seq  extends spi_bus_base_seq;

`uvm_object_utils(div_load_seq)

  // Interesting divisor values:
  constraint div_values {data[15:0] inside {16'h0, 16'h1, 16'h2,
                                             16'h4, 16'h8, 16'h10,
                                             16'h20, 16'h40, 16'h80};}

  task body;
    super.body;
    // Randomize the local data value
    assert(this.randomize());
    // Write to the divider register
    write_reg(model.divider_reg, status, data);
    assert(status == UVM_IS_OK);
  endtask: body

endclass: div_load_seq
```

Extends base sequence

Randomizes data value with specific constraint

Write data to divider register

Register Sequence: TX Data Load



```
class data_load_seq extends spi_bus_base_seq;

  `uvm_object_utils(data_load_seq)

  uvm_reg data_regs[]; // Array of registers

  task body;
    super.body;
    // Set up the data register handle array
    data_regs = '{spi_rm.rxtx0_reg, spi_rm.rxtx1_reg,
                  spi_rm.rxtx2_reg, spi_rm.rxtx3_reg};
    // Randomize order
    data_regs.shuffle();
    foreach(data_regs[i]) begin
      assert(data_regs[i].randomize());
      update_reg(data_regs[i], status, UVM_FRONTDOOR);
    end
  endtask: body
endclass: data_load_seq
```

Extends base sequence

Get an array of register handles

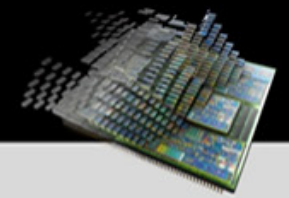
Randomize the array index order

Foreach reg in the array

Randomize the content

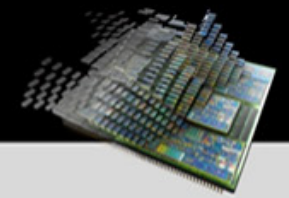
Update the register

Built-In Sequences



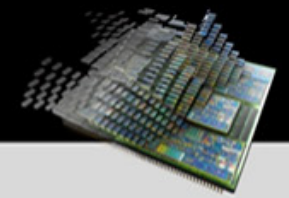
- **Sequences are easy to run**
  - Low overhead to use
  - Useful for initial sanity checks on bus connectivity
- **Access modes are respected**
  - e.g. Read only registers are not bit bashed
  - Read only memories are not tested
- **Memories, Registers or Fields can be opted out of a test**
  - e.g. Clock enable bit
  - Mechanism is to use the uvm_config_db to set an attribute for the register

Register Built-In Sequences



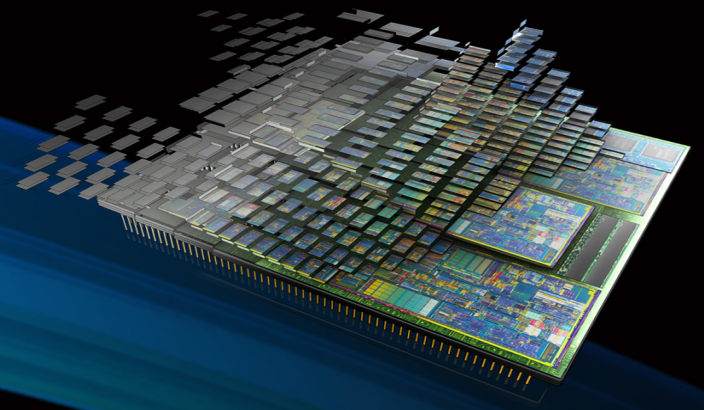
Sequence Name	Description
uvm_reg_hw_reset_seq	Checks register reset values
uvm_reg_single_bit_bash_seq	Checks R/W path to each register bit in a register
uvm_reg_bit_bash_seq	Runs single_bit_bash_seq on a register block
uvm_reg_single_access_seq	Checks that both front and back door accesses work correctly for a register
uvm_reg_access_seq	Runs single_access_seq on a register block
uvm_reg_shared_access_seq	If a register is in multiple maps, checks that accesses can be made from each map

UVM Register Summary



- **Register model follows hardware structure**
  - Fields, Registers, Blocks, Maps
  - Internal access – get(), set() etc.
    - Sets up desired value
  - External access – Front and Backdoor
- **Access layered via model**
  - Generic sequences adapted to target bus sequences
  - Sequence reuse straight-forward
- **Use the convenience API**
  - Extend uvm_reg_sequence
  - write_reg()/read_reg() vs. write()/read()
  - Don’t have to worry about .parent() argument

Verification Academy



Advanced UVM

Using the Register Layer

Tom Fitzpatrick

Verification Evangelist

info@verificationacademy.com | *www.verificationacademy.com*

Mentor Graphics