Increasing Verification Productivity with VMM Applications

Dr. Ambar Sarkar
Chief Verification Technologist
Paradigm Works, Inc.

Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Extracting RALF Definitions Automatically
- Summary

Introduction –
ASIC and FPGA Technology and Development, FOSS

- **Founded in 2000, MA**
  - Michael Hoyt, CEO
  - Jim Crocker, VP
- **Design and Verification Services**
- **SW Products**
  - ReleaseWorks®
  - SystemVerilog FrameWorks™
  - RegWorks™

- **Highest value proposition in industry**
  - High quality engineering resources
  - Low cost structure
- **Blue Chip Client List**
- **Global Relationships**


Introduction

- ***All*** **designs have registers**
- **Register models are commonly used in verification environments**
  - Helps access and mirror register data
  - Provides clean API
    - i.e. WRITE(CFG_1,5) and READ(CFG_1) operations
  - Configure the chip and verify register data
  - Same registers may be used in multiple testbenches
  - In-house register model development could be expensive

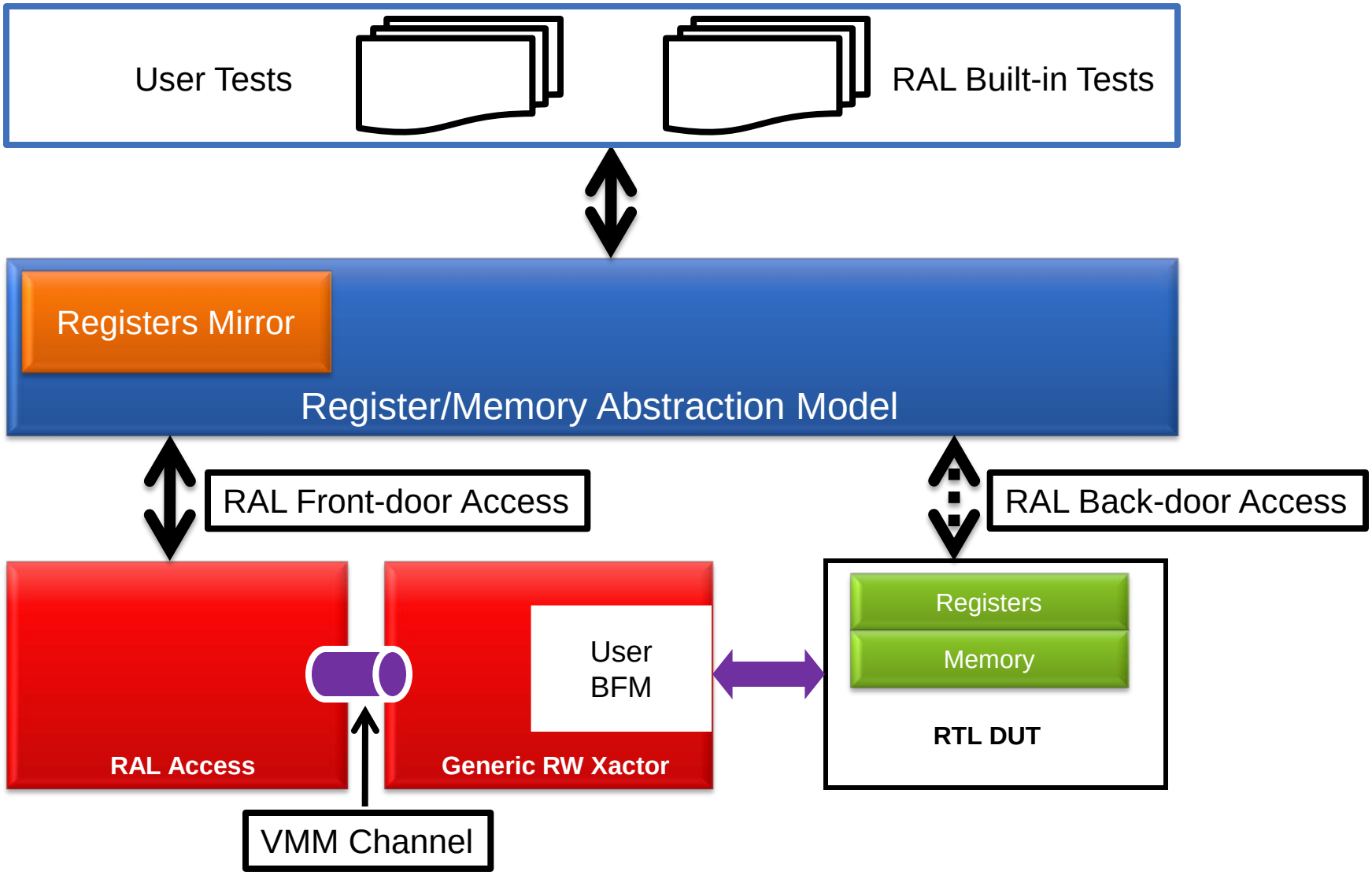
Challenges With Using Registers

- DUT register count is increasing
- Register definitions change – high maintenance
- Basic testing – labor intensive, tedious
- Backdoor – ad hoc approach, limits reuse
- Supporting many DUTs – model complexity

Solution: VMM Register Abstraction classes with built-in support for -

- Mirrored register state
- Field abstractions
- Built in tests cover the basics
- Automatic generation of backdoor methods
- Consistent, flexible, extensible API
- Reusable, hierarchical model

Typical RAL Based Environment



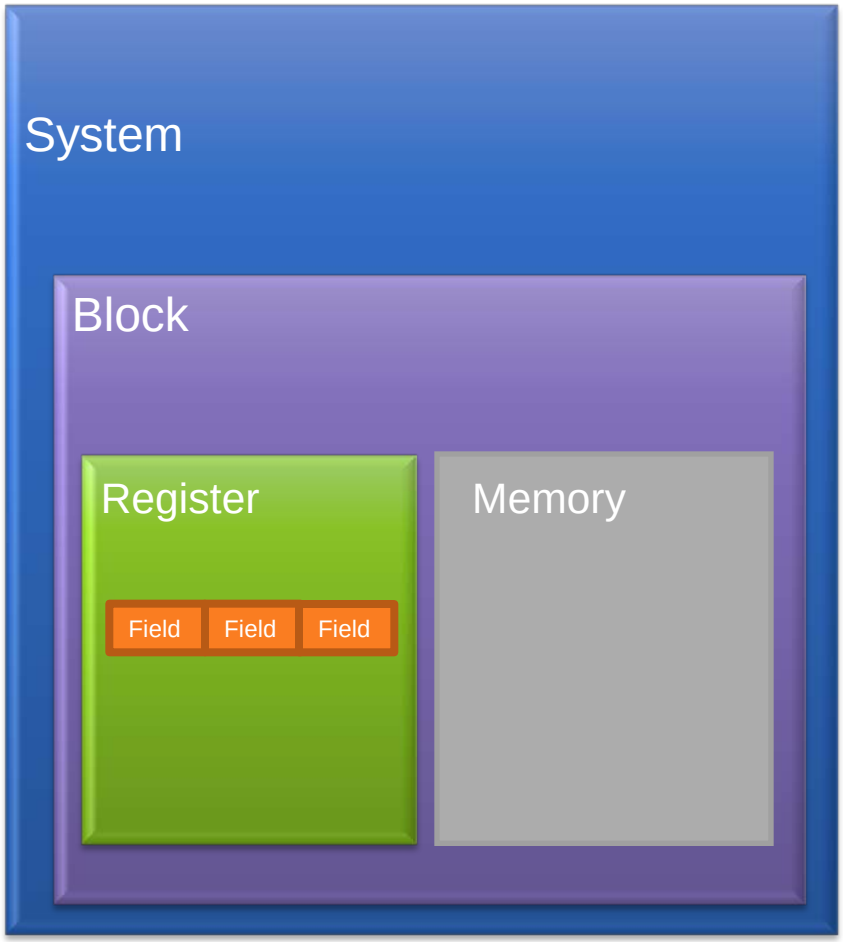
Mirror

- **RAL mirrors the DUT's registers**
  - Updated via register read/write
- **Mirroring Internal DUT register**
  - Updates status and counter registers
  - No write transaction
  - RAL provides methods to manually update mirror
- **Scoreboarding**
  - Mirror not a scoreboard
  - Possible to use RAL as a simple scoreboard
    - Can predict register value at specific known time
    - Testbench can update the mirror with predicted value

Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Extracting RALF Definitions Automatically
- Summary

Register and Memory Specification



- **A RALF file is used to describe registers & memories**
  - Uses Tcl 8.5 file format
- **Registers are made up of fields**
- **Blocks consist of registers and/or memories**
- **System is made up of blocks and/or systems**
- **Top level can be the system or block**

Register and Memory Specification

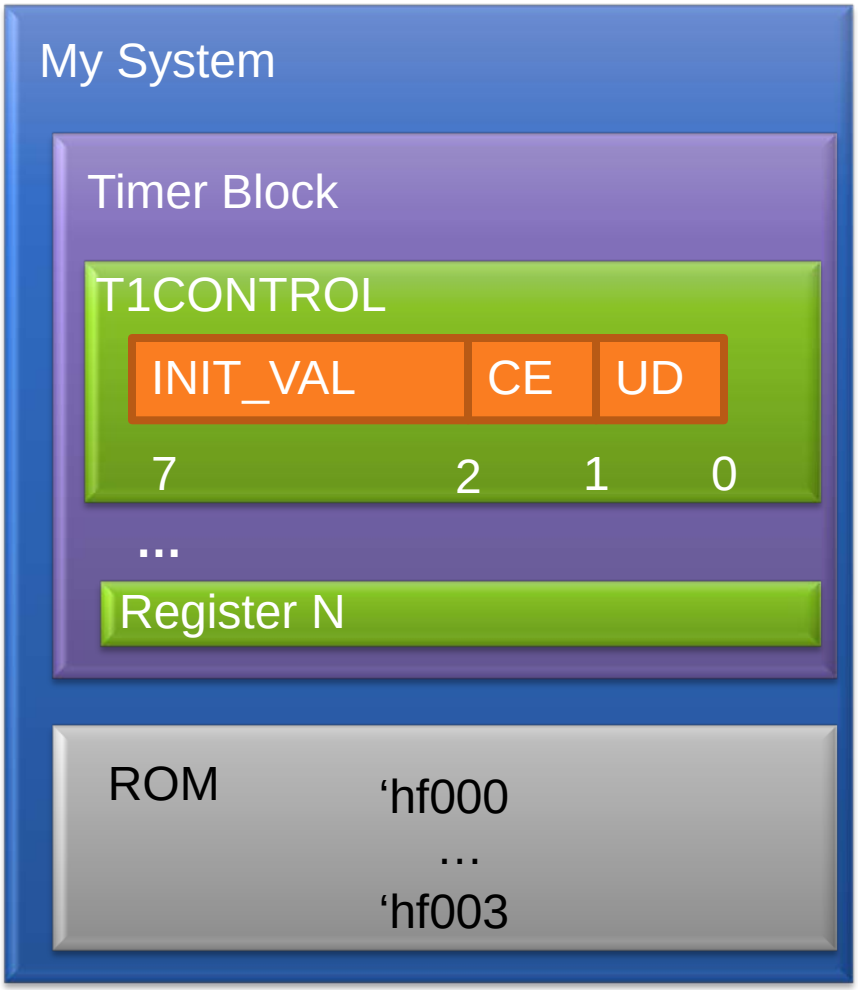
```
system My { bytes 4;
  block Timer {
    bytes 4;
    register T1CONTROL {
      field INIT_VAL {
        bits 6;
        reset 'h3f;
        constraint spec {
          value <= 6'h3;
        }
      }
      field CE { reset 'h1; }
      field UD { reset 'h1; }
    }
  }
  memory rom @'hf000 {
    size 4; bits 32; access ro;
  }
}
```

Reset Value

Constraint

ROM is Read Only

RALF File



Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Extracting RALF Definitions Automatically
- Summary

Generated Models - *ralgen*

- *ralgen* tool generates the RAL model

```
% ralgen -t topname -l sv {-I dir} filename.ralf
```

Where:

`-t` ***`topname`***

Top-most block or system in RALF file(s) for DUT.

`-l` ***`sv`***

Generate SystemVerilog code.

`-I` ***`dir`***

Optional list of directories to search for Tcl source file(s)

`filename.ralf`

File(s) containing the RALF description.

Generated Models - *ralgen*

"field" is basic unit

DUT.ralf

RALF File

```
register A {

   field Y {…}
   field Z {…}

}

block MY_BLK {
   register A;
}

system MY_SYS {

   block MY_BLK=BLK0 @'h1000;
   block MY_BLK=BLK1 @'h2000;
}
```

Multiple instances

RAL Model

```
class ral_reg_A extends
vmm_ral_reg;
   vmm_ral_field Y;
   vmm_ral_field Z;
endclass

class ral_block_MY_BLK extends
vmm_ral_block;
   ral_reg_REG_A REG_A;
   vmm_ral_field Y;
   vmm_ral_field Z;
   vmm_ral_field REG_A_Y;
   vmm_ral_field REG_A_Z;
endclass

class ral_sys_MY_SYS extends
vmm_ral_sys;
   ral_block_MY_BLK BLK0;
   ral_block_MY_BLK BLK1;
endclass
```

Fields and Registers

- Fields
  - Model register field access behavior
    - i.e. `RW, RO, W1C`
  - Field names should be unique within a block
- Registers
  - Class that is a container for fields
  - Methods to reset, update and view register content

RALF File

```
register REG_A {
   field Y {…}
   field unused {…}
   field Z {…}
}
```

RAL Model

```
class ral_reg_REG_A extends vmm_ral_reg;

   vmm_ral_field Y;
   vmm_ral_field Z;
   ...
endclass
```

Blocks and Systems

- Contain sets of registers and blocks
- Contain other blocks, or systems

RALF File

```
block BLK {
 bytes 4;
 endian little;
 register REG_A;
 register REG_B;
}

system SYS {
 bytes 4;
 block BLK=BLK_1 @'h10;
 block BLK=BLK_2 @'h20;
}
```

RAL Model

```
class ral_block_BLK extends
vmm_ral_block;
  rand ral_reg_REG_A REG_A;
  rand ral_reg_REG_B REG_B;
  rand vmm_ral_field Y;
  rand vmm_ral_field Z;
  rand vmm_ral_field REG_A_Y;
  rand vmm_ral_field REG_B_Z;
endclass

class ral_sys_MY_SYS extends
vmm_ral_sys;
  rand ral_block_MY_BLK BLK_1;
  rand ral_block_MY_BLK BLK_2;
endclass
```

Arrays and Regfiles

- Arrays allow iteration of registers at run-time
- Regfiles group registers

RALF File

```
register A {
 field X;
}

register B {
 field  Y{reset 1};
}

block MY_BLK {
 bytes 128;

 regfile reg[5] {
  register A;
  register B;
 }
}
```

RAL Model

```
class ral_regfile_MY_BLK_reg;
  rand ral_reg_A A;
  rand ral_reg_B B;
…
endclass

class ral_block_MY_BLK extends
vmm_ral_block;
 rand ral_regfile_MY_BLK_reg reg[5];
endclass
```

reg[0].A_REG, reg[0].B_REG,
reg[1].A_REG, reg[1].B_REG,
…

Address Map

Memories

- Sparse memory model
- Models memory behaviors such as RO, RW, and WO
- Methods to reset, update and view memory content
- Includes backdoor capabilities

RALF File

```
memory MEM {
 size 128; bits 64;
}

block MY_BLK {
 bytes 8;
 endian little;
 memory MEM () @ 0x0100;
}
```

RAL Model

```
class ral_mem_MEM extends
vmm_ral_mem;
…
endclass

class ral_block_MY_BLK extends
vmm_ral_block;
  rand ral_mem_MEM MEM;
endclass
```

Randomized Fields

- Random fields are specified with a constraint attribute
- An unconstrained field that can be randomized must include an empty constraint attribute.

RALF File

```
register r {
 field f1 { bits 2; }
 field f2 { bits 8;
  constraint empty {};
 }
 field f3 { bits 8;
  constraint spec {
   value <= 8'h80;
  }
 }
 constraint consistency {
  f3.value == f2.value;
 }
}
```

RAL Model

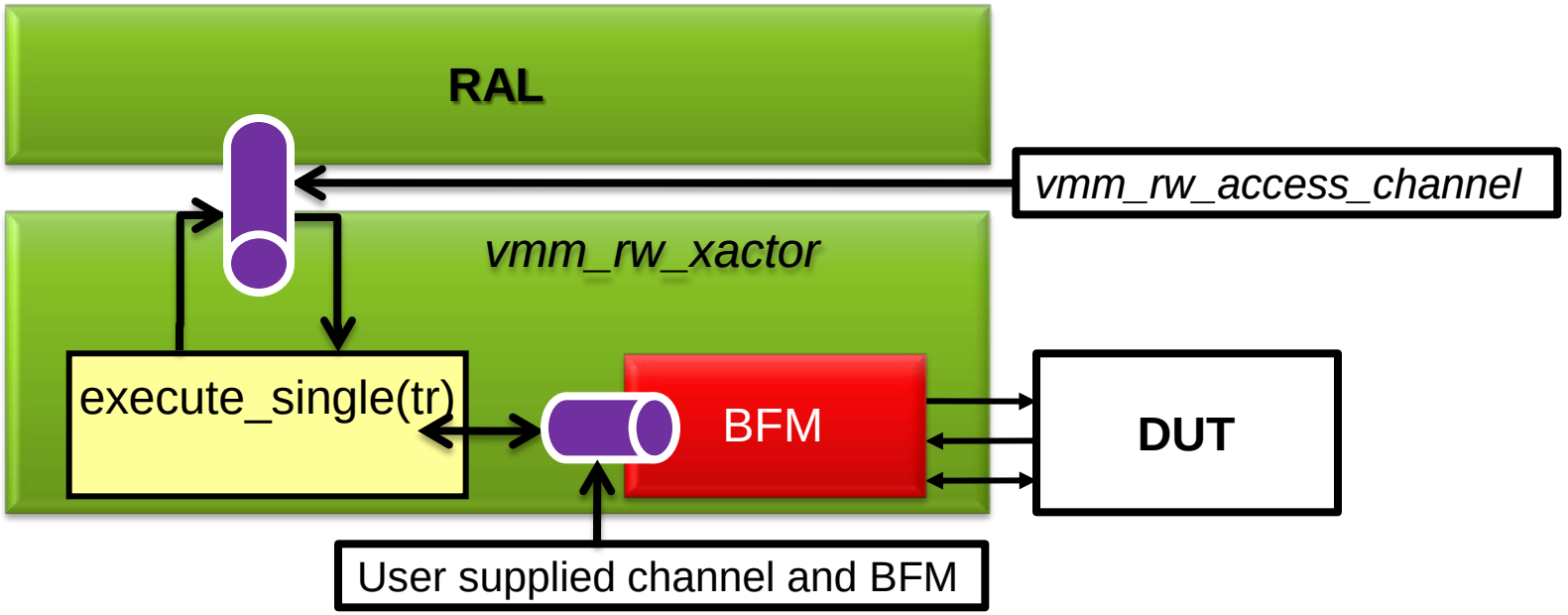
```
class ral_reg_r extends vmm_ral_reg;
   rand vmm_ral_field f1;
   rand vmm_ral_field f2;
   rand vmm_ral_field f3;

   constraint f2_empty {};
   constraint f3_spec {
      f3.value <= 8'h80;
   }
   constraint consistency {
      f3.value == f2.value;
   }
endclass : ral_reg_r
```

Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Extracting RALF Definitions Automatically
- Summary

Hooking Up RAL

- RAL is a high-level abstraction mechanism
- RAL and DUT communicate via Generic Transactor
- Generic Transactor executes RAL transactions



Hooking Up RAL

vmm_env -> vmm_ral_env

- Must derive from the *vmm_ral_env* class
- *vmm_ral_env*
  - Simple extension of *vmm_env*
  - Serves to introduce two predefined elements:
    - An instance of the *vmm_ral_access* transactor
    - Reset task to reset *DUT* repeatedly during simulation

```
class vmm_ral_env extends vmm_env;
  virtual task reset_dut();
     hw_reset();
endclass
```

Hooking Up RAL

- **Generic RW transactions**
  - Part of the RAL base class
  - Transaction sent via *vmm_rw_access_channel*
  - Sent to Generic Transactor's overloaded virtual task
  - User translates and sends transactions to physical interface of DUT

```
class vmm_rw_access extends vmm_data;
  rand bit [63:0] addr;
  rand bit [63:0] data;
  rand vmm_rw::kind_e kind;
  rand vmm_rw::status_e status;
endclass
```

READ or WRITE

Return status
(IS_OK, ERROR, RETRY)

Hooking Up RAL

Extending the Generic Transactor

```
class ahb_rw_xlate extends vmm_rw_xactor;
  ahb_master ahb;

  function new(string inst, ahb_master ahb);
    super.new("AHB RAL Master", inst, ahb.stream_id);
    this.ahb = ahb;
  endfunction

  task start_xactor();
    super.start_xactor();
    ahb.start_xactor();
  endtask

  …
endclass ahb_rw_xlate
```

Extends from *vmm_rw_xactor*

BFM connected to DUT

Hooking Up RAL

Extending the Generic Transactor

```
class ahb_rw_xlate extends vmm_rw_xactor;
…
  virtual task execute_single(vmm_rw_access tr);
    ahb_tr xtr = new;

    xtr.kind = (tr.kind == vmm_rw::WRITE) ?
      ahb_tr::WRITE : ahb_tr::READ;
    xtr.addr = tr.addr << 2;
    xtr.data = tr.data[31:0];

    this.ahb.in_chan.put(xtr);
    // wait for BFM to complete fully
    if (tr.kind == vmm_rw::READ) tr.data = xtr.data;
    tr.status = (xtr.status == ahb_tr::NO_ERR) ?
      vmm_rw::IS_OK : vmm_rw::ERROR;
  endtask: execute_single
endclass ahb_rw_xlate
```

Translate generic R/W transaction

Execute to completion on DUT

Translate response

Hooking Up RAL

- Construct instance of the generated RAL model
- Register the RAL model

ral_env.svh

```
`include "ral_env.sv"
`define RAL_TB_ENV my_env
```

```
class my_env extends vmm_ral_env;
ral_block_uart ral_model;
...
function new();
       this.ral_model = new;
       this.ral.set_model(this.ral_model);
endfunction: new
...
endclass: tb_env
```

The RAL model name is generated by the RALF description

Register the RAL model

Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Extracting RALF Definitions Automatically
- Summary

Predefined Tests

- **General**
  - Tests assume DUT is completely idle
  - Bits of mode *vmm_ral::OTHER* and *vmm_ral::USER* excluded except in hw_reset
- ***hw_reset***
  - Applies hardware reset to the design
  - Reads all registers in the design
  - Verifies that the value read for each register corresponds to the specified reset value
- ***bit_bash***
  - Verifies that all bits operate as specified (rw, ro, w1c etc)

Predefined Tests

- ***reg_access***
  - Exercises all registers with a backdoor access available using the following process:
    - Write ~reset via front door
    - Check register content using backdoor
    - Write reset value via backdoor
    - Check register content using front door
- ***mem_walk***
  - Walks through all addresses in vmm_ral::RW memories using the following process for address k:
    - Write ~k at address k
    - If k>0, read address k-1 and expect ~(k-1)
    - If k>0, write k-1 at address k-1
    - If last address, read address k and expect ~k
- ***mem_access, shared_access***
  - Similar to *reg_access*

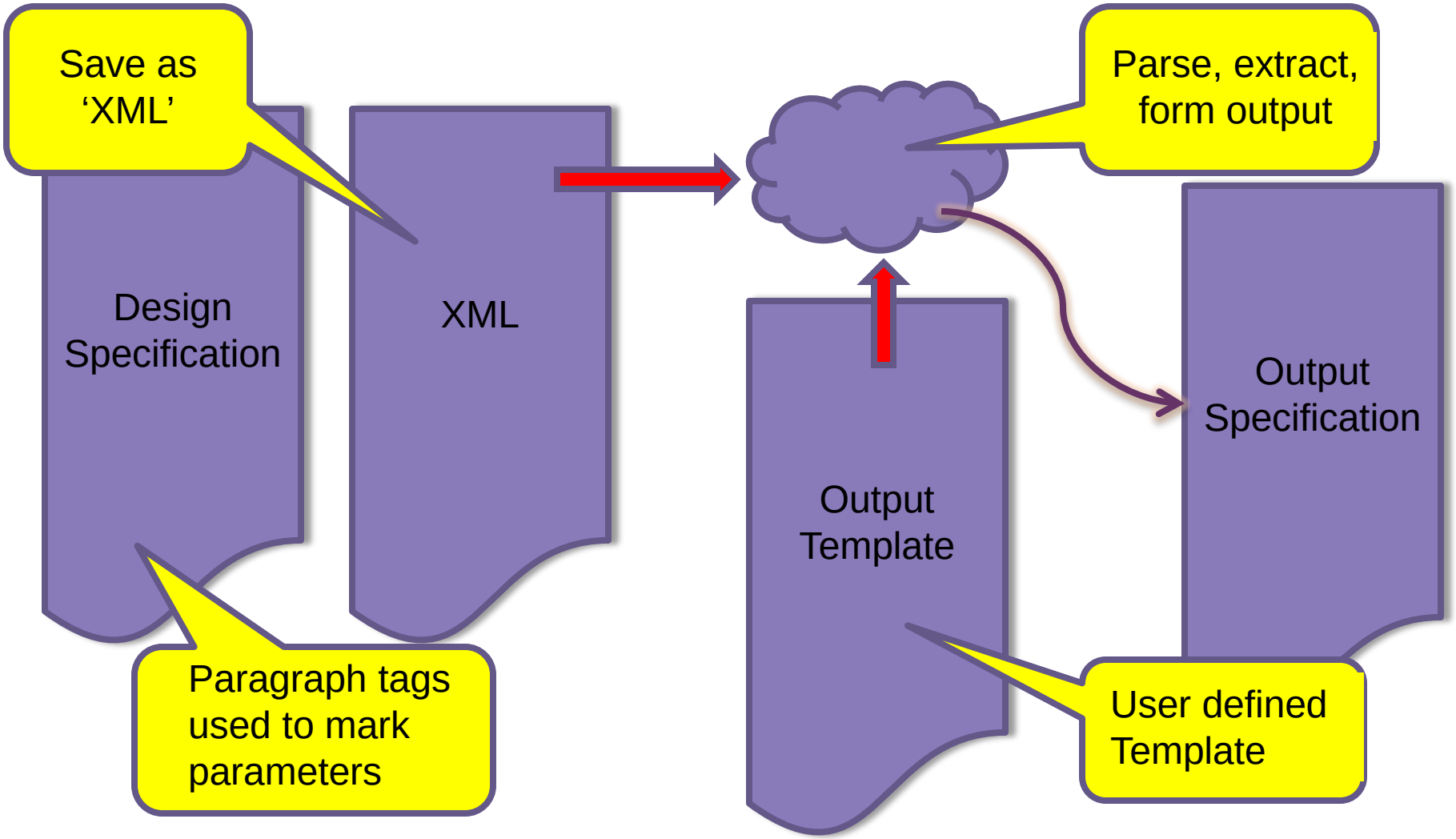
Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Functional Coverage
- Extracting RALF Definitions Automatically
- Summary

RegWorks
Extracting RALF Definitions Automatically

- **Free Open Source Software (FOSS)**
  - Available on SourceForge (search RegWorks)
- **Developed by Paradigm-Works**
- **Generates RALF formatted register definition**
  - From MS-Word specifications
  - No scripting knowledge required
- **Provides pre-defined MS-Word templates**
  - To generate register and memories definitions
- **Complete customization of generated code**

RegWorks - How it works



Agenda

- Introduction
- Specifying Registers and Memories
- Generated Models
- Hooking up RAL
- Pre-defined Tests
- Functional Coverage
- Extracting RALF Definitions Automatically
- Summary

Register Abstraction Layer
Summary

- RAL delivers powerful application for handling registers and memories
- Supports automated testing
- Highly reusable
- Supports 3rd party applications
  - Enables a complete end-to-end flow for register related verification