AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Advanced UVM RAL

Neena Chandawale
(Host)

Nitin Chaudhary
(Presenter)

Agenda

- Verification Challenges
- UVM RAL
  - Overview
  - Access API
  - Constructing a Register Model
  - Integrating a Register Model
- Coverage Model
  - Customizing auto generated coverage
  - Cross Coverage
- UVM Callbacks
- Auto Mirroring
- UVM HDL PATHS
- Handling Constraints in UVM
- UVM IDS properties

Verification Challenges

- Managing Complexity
  - Reuse
    - Across industry – Standards
    - Across user groups – EDA tools
    - Across projects – IPs, VIPs
    - Across levels – Block, sub system, SoC
- Manual coding which causes:
  - errors
  - time consuming
  - resource consuming
- No standard way of capturing the information
  - Different way of capturing information by various teams
  - Various coding styles and design intent

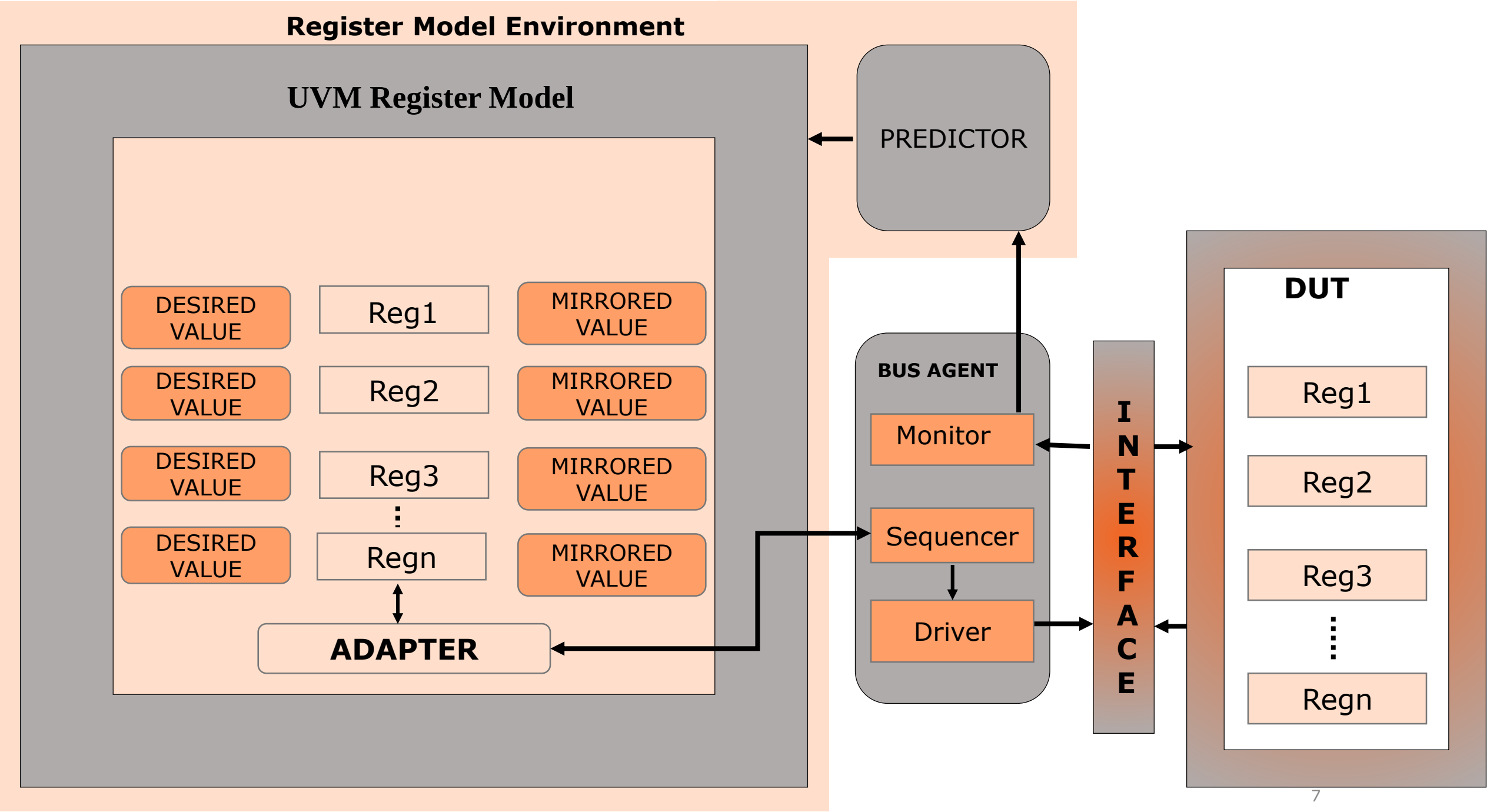
UVM RAL

Overview

- UVM RAL provides a standard base class libraries that enable users to implement the object-oriented model to access the DUT registers and memories
- UVM provides the best framework to achieve coverage-driven verification (CDV)
- UVM RAL provides a set of base classes and methods with a set of rules which easies the effort required for register access
- The UVM register layer classes are used to create a high-level, object-oriented model for memory-mapped registers and memories in a design under verification (DUV)

Access API

- **write()/read()**
  - Physical write/read transactions is executed on DUT
  - Mirrored value is then updated
- **poke()/peek()**
  - Write/read directly to the register, bypassing the physical interface
  - Mirrored value is then updated
- **set()/get()**
  - Write/read directly to the desired value, without accessing the DUT
- **update()**
  - If desired value is different from mirrored value, update method invokes write method
  - Hence the mirrored value is updated
- **mirror()**
  - mirror method invokes read() method to update mirrored value
  - Mirror can also compare the readback value with the current mirrored value before updating it


Register Model Environment
UVM Register Model
PREDICTOR
DESIRED VALUE
Reg1
MIRRORED VALUE
DESIRED VALUE
Reg2
MIRRORED VALUE
DESIRED VALUE
Reg3
MIRRORED VALUE
DESIRED VALUE
Regn
MIRRORED VALUE
ADAPTER
BUS AGENT
Monitor
Sequencer
Driver
INTERFACE
DUT
Reg1
Reg2
Reg3
Regn

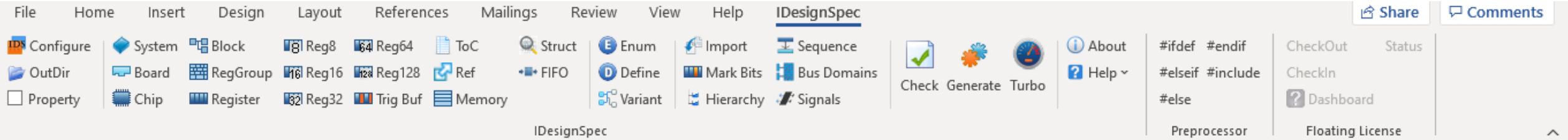
Features and Benefits of UVM RAL

- UVM RAL provides high-level abstraction for reading and writing DUT registers. i.e, registers can be accessed with its names
- It provides a register test sequence library containing predefined test cases these can be used to verify the registers and memories
- Register layer classes support front-door and back-door access
- Design registers can be accessed independently of the physical bus interface. i.e by calling read/write methods

UVM RAL using IDesignSpec™

- An instance of a register block is a register model, which may contain any number of registers, register files, memories, and other blocks
- Each register file contains any number of registers and other register files. Further, each register contains any number of fields, which mirror the values of the corresponding elements in hardware
- For each element in a register model—field, register, register file, memory or block—there is a class instance that abstracts the read and write operations on that element

IDesignSpec™ Register Specification



- Consider the following example of a register defined inside the block

IDSWord:

1 IDS_block		IDS_block		Block	0x0
offset		external		size	

1.1 IDS_reg_A		IDS_reg_A		Reg.	0x0
offset		external		size	32	default	0x0000000a

Sample register of block IDS_block

bits	name	s/w	h/w	default	description
31:0	IDS_fld	Rw	Rw	'hA	Field for the register IDS_reg_A with reset value of 'hA

RDL:

```
addrmap  IDS_block {
  reg IDS_reg_A  {
    desc = "Sample register  of block IDS_block";
    field {
      desc = "Field for the register IDS_reg_A with the reset
value'hA";
      hw = rw;
      sw = rw;
    }IDS_fld[31:0] = 32'hA ;
  };
  IDS_reg_A  IDS_reg_A;
};
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Register model hence generated

```
/*----------------------------------------------------------------------
Class       : IDS_block_block
----------------------------------------------------------------------*/
`ifndef CLASS_IDS_block_block
`define CLASS_IDS_block_block
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)

    rand IDS_block_IDS_reg_A IDS_reg_A;

    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction

    // Function : build
    virtual function void build();
        //IDS_REG_A
        IDS_reg_A   =   IDS_block_IDS_reg_A::type_id::create("IDS_reg_A");
        IDS_reg_A.configure(this, null, "IDS_reg_A");
        IDS_reg_A.build();

        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        default_map.add_reg( IDS_reg_A, 'h0, "RW");

        lock_model();
    endfunction

endclass : IDS_block_block
`endif
```

```
/*----------------------------------------------------------------------
Class       : IDS_block_IDS_reg_A

Description : Sample register of block IDS_block
----------------------------------------------------------------------*/

`ifndef CLASS_IDS_block_IDS_reg_A
`define CLASS_IDS_block_IDS_reg_A
class IDS_block_IDS_reg_A extends uvm_reg;
    `uvm_object_utils(IDS_block_IDS_reg_A)

    /*Field for the register IDS_reg_A with reset value of 'hA*/
    rand uvm_reg_field IDS_fld;

    // Function : new
    function new(string name = "IDS_block_IDS_reg_A");
        super.new(name, 32, build_coverage(UVM_NO_COVERAGE));
        add_coverage(build_coverage(UVM_NO_COVERAGE));
    endfunction

    // Function : build
    virtual function void build();
        this.IDS_fld = uvm_reg_field::type_id::create("IDS_fld");
        this.IDS_fld.configure(this, 32,  0, "RW", 0, 32'd10, 1, 1, 0);
    endfunction
endclass
`endif
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Constructing a Register Model

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Register

1.1.1 ids_reg	ids_reg	Reg.	0x00000, 0x00018...
offset	external	default	0x1

block[@name='block_name']/section[@name='reg_file']/reg[@name='ids_reg']

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	description
0	EVNTSTMP	rw	ro	1	An event stamp has occurred and put into the circular event buffer
1	SPDALRM	rw	ro	0x0	1= speed is faster then set speed, used to alert the rider
2	BTLOW	rw	ro	0x0	1= when battery level is below 10%
3	CLBRNDNE	rw	ro	0x0	1= when auto-calibration completed
4	CLBRNFLR	rw	ro	0x0	1= when auto-calibration cycle failed to calibrate
5	ABRTDNE	rw	ro	0x0	1=abort has completed and block is ready to set up again for the next task

```
class block_name_reg_file_ids_reg extends uvm_reg;
rand uvm_reg_field EVNTSTMP;
rand uvm_reg_field SPDALRM;
rand uvm_reg_field BTLOW;
rand uvm_reg_field CLBRNDNE;
rand uvm_reg_field CLBRNFLR;
rand uvm_reg_field ABRTDNE;
 function new(string name = "block_name_reg_file_ids_reg");
    super.new(name, 32, build_coverage(UVM_NO_COVERAGE));
add_coverage(build_coverage(UVM_NO_COVERAGE));
 endfunction
 virtual function void build();
   this.EVNTSTMP = uvm_reg_field::type_id::create("EVNTSTMP");
   this.SPDALRM = uvm_reg_field::type_id::create("SPDALRM");
   this.BTLOW = uvm_reg_field::type_id::create("BTLOW");
   this.CLBRNDNE = uvm_reg_field::type_id::create("CLBRNDNE");
   this.CLBRNFLR = uvm_reg_field::type_id::create("CLBRNFLR");
   this.ABRTDNE = uvm_reg_field::type_id::create("ABRTDNE");
   this.EVNTSTMP.configure(this, 1,  0, "RW", 0, 'd1, 1, 1, 0);
   this.SPDALRM.configure(this, 1,  1, "RW", 0, 'd0, 1, 1, 0);
   this.BTLOW.configure(this, 1,  2, "RW", 0, 'd0, 1, 1, 0);
   this.CLBRNDNE.configure(this, 1,  3, "RW", 0, 'd0, 1, 1, 0);
   this.CLBRNFLR.configure(this, 1,  4, "RW", 0, 'd0, 1, 1, 0);
   this.ABRTDNE.configure(this, 1,  5, "RW", 0, 'd0, 1, 1, 0);
 endfunction
`uvm_object_utils(block_name_reg_file_ids_reg)
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Register Group

1.2 reg_grp	reg_grp	RegGrp	0x4
offset	external	repeat	20	size

1.2.1 reg_array	reg_array	Reg.	0x4
offset	external	size	32	default	0x00000000

bits	name	s/w	h/w	default	description
7:0	Fld_1	Rw	Ro	'h0	

End RegGroup

RDL:

```
regfile {
    reg reg_array {
        regwidth = 32;
        field {
            sw = rw;
            hw = r;
        }Fld_1[7:0] = 8'h0 ;
    };
    reg_array  reg_array;
} reg_grp[20];
```

```
class IDS_block_reg_grp extends uvm_reg_file;
    `uvm_object_utils(IDS_block_reg_grp)

    rand IDS_block_reg_grp_reg_array reg_array;

    // Function : new
    function new(string name = "IDS_block_reg_grp");
        super.new(name);
    endfunction

    // Function : build
    virtual function void build();
        //create
        reg_array   =   IDS_block_reg_grp_reg_array::type_id::create("reg_array");

        //config
        reg_array.configure(get_block(), this, "reg_array");

        //build
        reg_array.build();
    endfunction

    virtual function void map(uvm_reg_map mp, uvm_reg_addr_t offset);
        //add reg and regfiles
        mp.add_reg(reg_array, offset +  'h0, "RW");
    endfunction

    virtual function void set_offset(uvm_reg_map mp, uvm_reg_addr_t offset);
        reg_array.set_offset(mp, offset +  'h0);
    endfunction
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Memory

1.2 Dma_mem				Dma_mem		Memory	0x2000, 0x2004...
offset	'h2000	depth	256	width	32	default	32'h0

```
class Blockl_Dma_mem_Dma_mem extends uvm_mem
    `uvm_object_utils(Blockl_Dma_mem_Dma_mem)
    // Function : new
    function new(string name = "Blockl_Dma_mem_Dma_mem");
        super.new(name, 'h100, 32, "RO", UVM_NO_COVERAGE);
    endfunction
endclass
class Blockl_block extends uvm_reg_block;
    `uvm_object_utils(Blockl_block)
    rand Blockl_Dma_mem_Dma_mem Dma_mem;
    // Function : new
    function new(string name = "Blockl_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction
    // Function : build
    virtual function void build();
        //DMA MEM
        Dma_mem    =    Blockl_Dma_mem_Dma_mem::type_id::create("Dma_mem");
        Dma_mem.configure(this, "Dma_mem");
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        default_map.add_mem(Dma_mem, 'h2000);
        Dma_mem.clear_hdl_path();
        Dma_mem.add_hdl_path_slice("Dma_mem_Dma_mem", 0, 32);
        lock_model();
    endfunction
endclass : Blockl_block
```

RDL:

```
addrmap  IDS_block {
    mem  Dma_mem {
        mementries = 256;
        memwidth = 32
    };
    Dma_mem Dma_mem @0x2000;
};
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Block

1 IDS_block		IDS_block		Block	0x0
offset		external		size	

```
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)
    rand IDS_block_reg_file reg_file[4];
    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction
    // Function : build
    virtual function void build();
        //REG_FILE
        foreach (reg_file[reg_file_i])
        begin
            reg_file[reg_file_i] = IDS_block_reg_file::type_id::create($sformatf("reg_file['h%0x]",
            reg_file_i));
            reg_file[reg_file_i].configure(this, null, $sformatf("reg_file['h%0x]", reg_file_i));
            reg_file[reg_file_i].build();
        end
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        foreach (reg_file[reg_file_i])
        begin
            reg_file[reg_file_i].map(default_map, 'h0 + reg_file_i * 'h54);
        end
        lock_model();
    endfunction
endclass : IDS_block_block
```

RDL:

```
addrmap IDS_block {
    regfile reg_file {
        reg Reg1 {
            field {
            }fld[31:0] = 31'h0;
        };
        Reg1 Reg1;
    };
    reg_file reg_file[4]';
};
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Testbench Environment Class

- The testbench environment class is architected to provide a flexible and extendable verification component.

```
class tb_env extends uvm_component;
IDS_blk_block regmodel;                    // IDS Generated UVM Register model
apb_agent     apb;                         // APB BUS (UVM already supports this bus interface)
uvm_reg_sequence seq;                      // UVM Sequence class
    . . . . . .
string hdl_root = "top.DUV";               // Top level HDL Path
        . . . . .
regmodel.set_hdl_path_root(hdl_root);      // Defining HDL Path to Reg-Model for Back-door access
end
    . . . . .
regmodel.default_map.set_sequencer(apb.sqr,reg2apb);      //Connecting the Bus Adapter
    . . . . . . .
virtual task run_phase();                                  // run Task
```

- After the testbench environment is setup, the testbench is created, where these classes are used. Testbench program collects all the files in the Environment.

```
`include "apb.sv"                          // Include APB Bus interface
`include "uvm_top_DUV.sv"                  // Include TOP level DUV
`include "uvm_seqlib.sv"                   // Include Register Sequence Classes
`include "ids/IDS_blk.regmem.sv"           // Include IDS generated Register Model
import IDS_blk_regmem_pkg::*;              // Import the Package containing Register Model Classes
program tb;                                // testbench "tb" program
. . . . .
`include "uvm_env.sv"                      //Include the Testbench Environment Class
. . . . . .
endprogram
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Coverage Model in UVM

Coverage Model in UVM

- UVM register library class do not include any coverage model, it provide necessary APIs to control instantiation and sampling of various coverage models

- Predefined Functional Coverage Type Identifiers

Identifier	Description
UVM_NO_COVERAGE	No coverage models
UVM_CVR_REG_BITS	Coverage models for the bits read or written in registers
UVM_CVR_ADDR_MAP	Coverage models for the addresses read or written in an address map
UVM_CVR_FIELD_VALS	Coverage models for the values of fields
UVM_CVR_ALL	All coverage models

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Coverage Model Sampling

- By default, coverage model are not included in a register model when it is instantiated

- To include use method ***uvm_reg::include_coverage()***
  - Example : *uvm_reg::include_coverage("*", UVM_CVR_REG_BITS + UVM_CVR_FIELD_VALS);*

- Also sampling for a coverage model is implicitly disabled by default. To turn the sampling on use:-
  - ***uvm_reg_block::set_coverage()*** - For Block
  - ***uvm_reg::set_coverage()*** - For register
  - ***uvm_mem::set_coverage()*** – For Memory

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Defining UVM Coverage

- UVM Register Model has its default coverage types i.e. fields, bits and address-map coverage types.
- User can also control, what type of coverage code should be generated for any particular register or block.
- IDesignSpec (IDS) automatically generate the coverage code for all the components inside the top-level block.
- The following covergroups are generated in the register model for different coverage types:

Coverage types	covergroups generated
a	cg_addr
b	rd_cg_bits
	wr_cg_bits
f	rd_cg_vals
	wr_cg_vals
on (is equivalent to"abf")	cg_addr (on block/memory) rd_cg_bits and/or wr_cg_bits(on register) rd_cg_vals and/or wr_cg_vals(on register)
off	-

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Register having ‘coverage=on/bf’

1 IDS_block	IDS_block	Block	0x0
offset	external	size	

1.1 IDS_reg	IDS_reg	Reg.	0x0
offset	external	size 32	default 0x00000000

{coverage=on}

bits	name	s/w	h/w	default	description
0:15	field_0	Rw	Ro	0	
16:31	field_1	Rw	Rw	0	

```
class IDS_block_ids_reg extends uvm_reg;
.   .   .
    local uvm_reg_data_t m_current;
    local uvm_reg_data_t m_data;
    local uvm_reg_data_t m_be;
    local bit            m_is_read;
    covergroup wr_cg_bits;
        field_0: coverpoint {m_current[0]
        field_1: coverpoint {m_current[1]
    endgroup
    covergroup rd_cg_bits;
        field_0: coverpoint {m_current[0]
        field_1: coverpoint {m_current[1]
    endgroup
    covergroup wr_cg_vals;
        Fld1: coverpoint Fld1.value[1:0];
    endgroup
    covergroup rd_cg_vals;
        Fld1: coverpoint Fld1.value[1:0];
    endgroup
```

```
// Function : new
function new(string
    super.new(name,
    add_coverage(bui
    if (has_coverage
        wr_cg_bits =
        rd_cg_bits =
    end
    if (has_coverage
        wr_cg_vals =
        rd_cg_vals =
    end
endfunction
```

```
protected virtual function void sample(uvm_reg_data_t data, uvm_reg_data_t byte_en,
bit is_read, uvm_reg_map map);
    super.sample(data, byte_en, is_read, map);
    if (get_coverage(UVM_CVR_REG_BITS)) begin
        m_current = get();
        m_data    = data;
        m_be      = byte_en;
        m_is_read = is_read;
        if(!is_read) begin
            wr_cg_bits.sample();
        end
        if(is_read) begin
            rd_cg_bits.sample();
        end
    end
    if (get_coverage(UVM_CVR_FIELD_VALS)) begin
        if(!is_read) begin
            wr_cg_vals.sample();
        end
        if(is_read) begin
            rd_cg_vals.sample();
        end
    end
endfunction
virtual function void sample_values();
    super.sample_values();
    if (get_coverage(UVM_CVR_FIELD_VALS)) begin
        wr_cg_vals.sample();
        rd_cg_vals.sample();
    end
endfunction
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Memory having ‘coverage=on/a’

1 IDS_block	IDS_block	Block	0x0
offset	external	size	

1.1 Mem1	Mem1	Memory	0x0, 0x4...
offset	depth 256	width 32	default 0
{coverage=on}			

```
class IDS_block_Mem1_Mem1 extends uvm_mem;
    `uvm_object_utils(IDS_block_Mem1_Mem1)
    local uvm_reg_addr_t m_offset;
    covergroup cg_addr;
        Mem1 : coverpoint m_offset
        {
            bins FIRST  = {[0:252]};
            bins SECOND = {[256:508]};
            bins THIRD  = {[512:764]};
            bins FOURTH = {[768:1020]};
        }
    endgroup
    // Function : new
    function new(string name = "IDS_block_Mem1_Mem1");
        super.new(name, 'h100, 32, "RO", build_coverage(UVM_CVR_ADDR_MAP
        ));
        if (has_coverage(UVM_CVR_ADDR_MAP)) begin
            cg_addr= new();
        end
    endfunction
    protected virtual function void sample(uvm_reg_addr_t offset, bit
    is_read, uvm_reg_map map);
        if (get_coverage(UVM_CVR_ADDR_MAP)) begin
            m_offset  = offset;
            cg_addr.sample();
        end
    endfunction
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Block having ‘coverage=on/a’

1 IDS_block	IDS_block	Block	0x0
offset	external	size	
{coverage=on}			

1.1 Mem1	Mem1	Memory	0x0, 0x4...
offset	depth 256	width 32	default 0

```
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)
    rand IDS_block_Mem1_Mem1 Mem1;
    local uvm_reg_addr_t m_offset;
    covergroup cg_addr;
        Mem1 : coverpoint m_offset
        {
            bins hit =  { ['h0 : 'h3FF] };
        }
    endgroup
    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, build_coverage(UVM_CVR_ADDR_MAP));
        if (has_coverage(UVM_CVR_ADDR_MAP)) begin
            cg_addr = new();
        end
    endfunction
    // Function : build
.   .   .
    protected virtual function void sample(uvm_reg_addr_t offset, bit
    is_read, uvm_reg_map map);
        if (get_coverage(UVM_CVR_ADDR_MAP))  begin
            m_offset  = offset;
            cg_addr.sample();
            if (this.get_parent() != null) begin
                this.get_parent().sample(m_offset+'h0,is_read,map);
            end
        end
    endfunction
endclass : IDS_block_block
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Cross Coverage

- Cross Coverage is specified between the cover points or variables. It is specified using the “cross” construct
- Verifying complex systems it is important that combination of functional points are verified
- A cross is defined to track the value of two or more coverpoints as a group:

  Syntax :

  CrossAB: **cross** a,b;

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Cross Coverage

RDL:

```
addrmap  IDS_block {
  coverage = "on";
  cross = "Reg1.Fld1:Reg2.Fld2";
  reg Reg1 {
    field {
      hw = rw;
      sw = rw;
    }Fld1[15:0] = 16'h0 ;
  };
  reg Reg2 {
    field {
      hw = rw;
      sw = rw;
    }Fld2[15:0] = 16'h0 ;
  };
  Reg1 Reg1 ;
  Reg2 Reg2 ;
};
```

```
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)

    rand IDS_block_Reg1 Reg1;

    rand IDS_block_Reg2 Reg2;

    local uvm_reg_addr_t m_offset;
    covergroup cg_addr;

        Reg1 : coverpoint m_offset
        {
            bins hit =  { 'h0};
        }
        Reg2 : coverpoint m_offset
        {
            bins hit =  { 'h4};
        }

    endgroup

    covergroup cross_covergroup;
        Reg1_Fld1: coverpoint Reg1.Fld1.value[15:0];
        Reg2_Fld2: coverpoint Reg2.Fld2.value[15:0];
        cross_Reg1_Fld1_Reg2_Fld2: cross Reg1_Fld1,Reg2_Fld2;
    endgroup

    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, build_coverage(UVM_CVR_ADDR_MAP));

        if (has_coverage(UVM_CVR_ADDR_MAP)) begin
            cg_addr = new();
            cross_covergroup = new();

        end
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Look-Up Table (LUT's)

- As LUT's contains important functional details of the design, creating cover points for their values help ascertain that all possible values are covered in the verification run.

1.1 status_reg			status_reg		Reg.
offset		external		size	32

{cross = current_state:error}

bits	name	s/w	h/w	default	description
2:0	current_state	Ro	Wo	0	{'b000:START,'b001:RINSE,'b010:SPIN,'b011:DRY,'b100:DONE}
3	error	Rc	Wo	0	{'b000:NO_ERROR,'b001:Imbalance}

```
class coverage_collector extends uvm_component;
    `uvm_component_utils(coverage_collector)
    washer_block rm;
    virtual controller_if washer_controller_if;

    covergroup statusreg_avg_analysis;
        currentstate: coverpoint rm.statusreg.currentstate.get{
        bins START = {0};
        bins RINSE = {1};
        bins SPIN = {2};
        bins DRY = {3};
        bins DONE = {4};}

        error: coverpoint rm.statusreg.error.get{
        bins START = {'b000};
        bins RINSE = {'b001};
        bins SPIN = {'b010};
        bins DRY = {'b011};
        bins DONE = {'b100};}

        cross_current_state_error : cross currentstate,error;
    endgroup
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Customized Auto-generated coverage

1 IDS_block	IDS_block	Block	0x0
offset	external	size	

1.1 reg_name	reg_name	Reg.	
offset	external	size 32	default 0

[uvm loc = coverpoint.b] ignore_bins ig_b={1,2}; [/uvm] {coverage=on}

bits	name	s/w	h/w	default	description
1:0	Fld_1	Rw	Rw	0	

```
covergroup wr_cg_bits;
    field_0: coverpoint {m_current[0],m_data[0]} iff (!m_is_read && m_be[0])
    {
        ignore_bins ig_b={1,2};
    }
    field_1: coverpoint {m_current[1],m_data[1]} iff (!m_is_read && m_be[0])
    {
        ignore_bins ig_b={1,2};
    }
endgroup
covergroup rd_cg_bits;

    field_0: coverpoint {m_current[0],m_data[0]} iff (m_is_read && m_be[0])
    {
        ignore_bins ig_b={1,2};
    }
    field_1: coverpoint {m_current[1],m_data[1]} iff (m_is_read && m_be[0])
    {
        ignore_bins ig_b={1,2};
    }
endgroup
covergroup wr_cg_vals;
    Fld_1: coverpoint Fld_1.value[1:0];
endgroup
covergroup rd_cg_vals;
    Fld_1: coverpoint Fld_1.value[1:0];
endgroup
```

To customize the generated coverage, user can use:
"[uvm] <custom code>[/uvm]" tags

Syntax:
[uvm loc = (coverpoint/covergroup).(abf/b/f/a).(rd/wr)] <u

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Callbacks

- Callback is a mechanism which is used for altering the behavior of component without modifying the component
- Developer of the component class defines a set of “hook” methods that enable users to customize certain behaviors of the component in a manner that is controlled by the component developer
- The integrity of the component’s overall behavior is intact, while still allowing certain customizable actions by the user
- The ***uvm_callbacks*** class provides a base class for implementing callbacks

UVM Callbacks

- Use Case : Aliased Register
  - Aliased registers are registers that are accessible from multiple addresses in the same address map
  - It is possible a register that contains a field that is RW when accessed via one address, but RO when accessed via another.
  - It would require two register types: one with a RW field and another one with a RO field

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Callbacks

RDL:

```
addrmap SpecialRegs {
  reg Reg1 {
    field {
      hw = rw;
      sw = rw;
    }Field1[31:0] = 32'h0 ;
  };
  Reg1 Reg1 ;
  alias Reg1 Reg1 Reg2;
  Reg2 ->sw = r;
};
```

```
class SpecialRegs_Reg1 extends uvm_reg;
    `uvm_object_utils(SpecialRegs_Reg1)

    rand uvm_reg_field Field1;
    ....

    // Function : build
    virtual function void build();
        this.Field1 = uvm_reg_field::type_id::create("Field1");

        this.Field1.configure(this, 32,  0, "RW", 0, 32'd0, 1, 1, 0);

    endfunction
endclass
```

```
class SpecialRegs_Reg2 extends uvm_reg;
    `uvm_object_utils(SpecialRegs_Reg2)

    rand uvm_reg_field Field1;
    ....

    // Function : build
    virtual function void build();
        this.Field1 = uvm_reg_field::type_id::create("Field1");

        this.Field1.configure(this, 32,  0, "RO", 0, 32'd0, 1, 1, 0);

    endfunction
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Callbacks

- The aliasing functionality must be provided in a block level class that links the two register type instances

```
class Alias_cb extends uvm_reg_cbs;
    local uvm_reg_field  m_toF;

    function new(string name, uvm_reg_field toF);
        super.new(name);
        m_toF = toF;
    endfunction

    virtual function void post_predict(input uvm_reg_field  fld,
        input uvm_reg_data_t previous,
        inout uvm_reg_data_t value,
        input uvm_predict_e kind,
        input uvm_path_e path,
        input uvm_reg_map map);
        if (kind == UVM_PREDICT_WRITE && path == UVM_FRONTDOOR) begin
            void'(m_toF.predict(value, -1, UVM_PREDICT_DIRECT, path, map));
        end
    endfunction
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Callbacks

- In Block class callback class is instantiated and registered with fields

```
class SpecialRegs_block extends uvm_reg_block;

    rand SpecialRegs_Reg1 Reg1;
    rand SpecialRegs_Reg2 Reg2;
    ....
    // Function : build
    virtual function void build();
        //REG1
        Reg1   =   SpecialRegs_Reg1::type_id::create("Reg1");
        Reg1.configure(this, null, "Reg1");
        Reg1.build();

        //REG2
        Reg2   =   SpecialRegs_Reg2::type_id::create("Reg2");
        Reg2.configure(this, null, "Reg2");
        Reg2.build();

        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        default_map.add_reg(Reg1, 'h0, "RW");
        default_map.add_reg(Reg2, 'h4, "RW");

        // Registering callback class instances with register fields
        begin
            Alias_cb Alias_SpecialRegs_Reg1_Field1;
            Alias_cb Alias_SpecialRegs_Reg2_Field1;

            Alias_SpecialRegs_Reg1_Field1 = new("Alias_SpecialRegs_Reg1_Field1", Reg1.Field1);
            uvm_reg_field_cb::add(Reg2.Field1, Alias_SpecialRegs_Reg1_Field1);

            Alias_SpecialRegs_Reg2_Field1 = new("Alias_SpecialRegs_Reg2_Field1", Reg2.Field1);
            uvm_reg_field_cb::add(Reg1.Field1, Alias_SpecialRegs_Reg2_Field1);
        end

    endfunction
endclass : SpecialRegs_block
```

Registering callback class instances with register fields

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Auto Mirroring in UVM

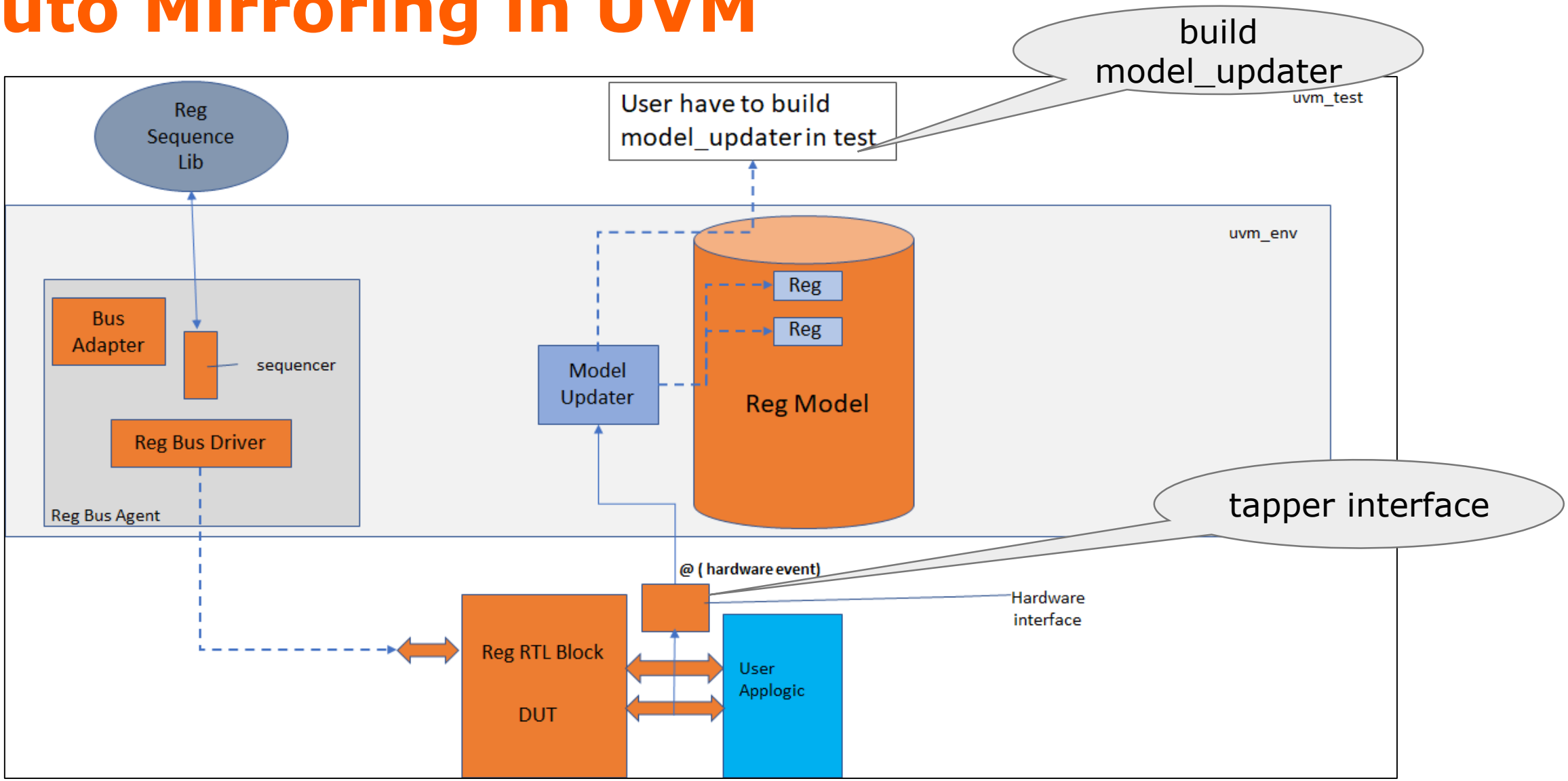
- Whenever a HW event occurs, the value on the HW interface needs to be updated on the regmap
  - To support this, the feature of **auto-mirroring** in IDS-generated UVM RAL is introduced.

- It updates the UVM register field with the value on HW interface when a HW event occurs on it
  - The required value is tapped from inside the HW register interface through a Block HW interface

- The feature of auto-mirroring is enabled using command line switch **"-auto_mirror"**

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Auto Mirroring in UVM



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM HDL Paths

HDL PATH

- To access a register or memory directly into the design, UVM register library can specify arbitrary hierarchical path components for blocks, register files, registers and memories that, when strung together, provide a unique hierarchical reference to a register or memory
- HDL_PATH is a mechanism by which each individual element in a UVM model is connected to the RTL model of the element.
- In IDS, the user can mention the hdl_path using a property named '**hdl_path**' with a value set to the hierarchical path.
- 'hdl_path' property can be added on the register, field, register array, register file, memory, block
- For gate level models, 'hdl_path_gate' property is used.

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Example

RDL:

```
addrmap  IDS_block {
   reg reg_name {
     hdl_path = "top.dut.r1";
     field {
       hw = rw;
       sw = rw;
     }Fld_1[31:0] = 2'h0 ;
   };
   reg_name reg_name ;
};
```

```
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)
    rand IDS_block_reg_name reg_name;
    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction
    // Function : build
    virtual function void build();
        //REG_NAME
        reg_name    =    IDS_block_reg_name::type_id::create("reg_name");
        reg_name.configure(this, null, "reg_name");
        reg_name.build();
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        default_map.add_reg( reg_name, 'h0, "RW");
        reg_name.clear_hdl_path();
        reg_name.add_hdl_path_slice("top.dut.r1", 0, 32);
        lock_model();
    endfunction
endclass : IDS_block_block
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

HDL Path Internal

- The structural hierarchy of registers and memories in the spec may vary from their actual design hierarchy.
- This property is used for defining the hdl paths of internal registers in a spec containing memories, external and internal registers.
- If specified on a block level, then the specified path will be prepended to the hdl paths of all registers within that block.

Example

1 Block1	Block1	Block	0x0
offset	external	size	
{hdl_path_internal=top.B1.}			

1.1 Reg1	Reg1	Reg.	0x0
offset	external	size	default
	1	32	0x00000000
{hdl_path=BB1.B2.reg_q}			

bits	name	s/w	h/w	default	description
1:0	Fld_1	Rw	Rw	0	

1.2 Reg2	Reg2	Reg.	0x4
offset	external	size	default
		32	0x00000000

bits	name	s/w	h/w	default	description
31:0	Fld_2	Rw	Rw	0	

```
// Function : build
virtual function void build();
    .   .   .
    Reg1.clear_hdl_path();
    Reg1.add_hdl_path_slice("BB1.B2.reg_q", 0, 32);
    Reg2.clear_hdl_path();
    Reg2.add_hdl_path_slice("top.B1.Reg2_Fld_2_q", 0, 32);

    lock_model();
endfunction
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Ignore_prop

- This property is used for ignoring hdl paths of internal registers in a spec, when "hdl_path_internal" property is used

1 Block1	Block1	Block	0x0
offset	external	size	
{hdl_path_internal=top.B1.}			

1.1 Reg1	Reg1	Reg.	0x0
offset	external	1	size	32	default	0x00000000
{hdl_path=BB1.B2.reg_q}						

bits	name	s/w	h/w	default	description
1:0	Fld_1	Rw	Rw	0	

1.2 Reg2	Reg2	Reg.	0x4
offset	external	size	32	default	0x00000000
{ignore_prop=hdl_path_internal}					

bits	name	s/w	h/w	default	description
31:0	Fld_2	Rw	Rw	0	

```
// Function : build
virtual function void build();
    .   .   .
    Reg1.clear_hdl_path();
    Reg1.add_hdl_path_slice("BB1.B2.reg_q", 0, 32);
    Reg2.clear_hdl_path();
    Reg2.add_hdl_path_slice("Reg2_Fld_2_q", 0, 32);
    lock_model();
endfunction
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Constraints

- Constraint Expression may consist of
  - Single statement Or
  - Statement following if condition Or
  - Statement following else of if condition
- The if condition may consist of
  - Single condition inside parenthesis
  - Multiple conditions separated by Logical OR and Logical And operator in any hierarchy
- Statement in any of the above case specified must have **value** keyword on LHS, which corresponds to the value of Field.
  - Operators that are used to assign values to **value** can be

    = - equal to

    != - not equal to

    < - less than

    <= - less than equal to

    >= - greater than equal to

    > - greater than

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Example

1.1 Crc_Error				Crc_Error		Reg.	0x0000000
offset		external		size	32	default	0xff

bits	name	s/w	h/w	default	description
7:0	Crc_err	rw	ro	0x0ff	{constraint= "value=10"}

Here the value corresponds to the value of field on which this property is applied. This will translate into coverpoints and constraints in UVM Regmodel

Constraints

```
constraint Crc_Error_Crc_err_constraint
  {
      Crc_err.value[7:0] == 'hA;
  }
```

Coverpoints

```
covergroup wr_cg_vals;
      Crc_Error_Crc_err: coverpoint Crc_err.value[7:0]
      {
          bins vals[1] = {'hA};
          illegal_bins err = default;
      }
  endgroup
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Complex Constraints

				Reg7		Reg.	
offset		external		size	32	default	0x00000000

bits	name	s/w	h/w	default	description
4:0	Fld1	Rw	Rw	0	{constraint=if(Fld2.value==9 && Fld3.value==9) then value = 3}
7:5	Fld2	Rw	Rw	0	{constraint= if(Fld3.value==9 || Fld3.value==9) then value = 3 else value =5}
15:8	Fld3	Rw	Rw	0	{constraint= if(Fld2.value==9 && Fld4.value==9) then value = 3,5,7,9}
20:16	Fld4	Rw	Rw	0	{constraint= if(Fld2.value==9 && Fld3.value==9) then value != 3}
26:21	Fld5	Rw	Rw	0	{constraint= if(Fld2.value==9 && Fld3.value==9) then value != 3,5,7,9}
28:27	Fld6	Rw	Rw	0	{constraint= if(Fld3.value==9 && Fld3.value==9) then value = 3 else value !=5}
31:29	Fld7	Rw	Rw	0	{constraint= if(Fld3.value==9 && Fld3.value==9) then value = 7,8,9 else value !=3,5,6,15}

```
constraint  Reg7_Fld1_constraint
  {
    if (((Fld2.value == 'h9) && (Fld3.value == 'h9)))
        Fld1.value[4:0] == 'h3;
  }
constraint  Reg7_Fld2_constraint
  {
    if (((Fld3.value == 'h9) || (Fld3.value == 'h9)))
        Fld2.value[2:0] == 'h3;
    else
        Fld2.value[2:0] == 'h5;
  }
constraint  Reg7_Fld3_constraint
  {
    if (((Fld2.value == 'h9) && (Fld4.value == 'h9)))
        Fld3.value[7:0] inside { 'h3,'h5,'h7,'h9 };
  }
constraint  Reg7_Fld4_constraint
  {
    if (((Fld2.value == 'h9) && (Fld3.value == 'h9)))
        Fld4.value[4:0] != 'h3;
  }
constraint  Reg7_Fld5_constraint
  {
    if (((Fld2.value == 'h9) && (Fld3.value == 'h9)))
        Fld5.value[5:0] inside { 'h3,'h5,'h7,'h9 };
  }
constraint  Reg7_Fld6_constraint
  {
    if (((Fld3.value == 'h9) && (Fld3.value == 'h9)))
        Fld6.value[1:0] == 'h3;
    else
        Fld6.value[1:0] != 'h5;
  }
constraint  Reg7_Fld7_constraint
  {
    if (((Fld3.value == 'h9) && (Fld3.value == 'h9)))
        Fld7.value[2:0] inside { 'h7,'h8,'h9 };
    else
        !(Fld7.value[2:0] inside { 'h3,'h5,'h6,'hF });
  }
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM IDS Properties

User Defined Classes

‘uvm_class’ property

1 IDS_block	IDS_block	Block	0x0
offset	external	size	

1.1 ids_reg	ids_reg				
offset	external	size	32	default	
{uvm_class=my_reg_class}					
bits	name	s/w	h/w	default	
31:0	Fld1	Rw	Ro	0	

```
class IDS_block_ids_reg extends my_reg_class;
    `uvm_object_utils(IDS_block_ids_reg)
    rand uvm_reg_field Fld1;
    // Function : new
    function new(string name = "IDS_block_ids_reg");
        super.new(name, 32, build_coverage(UVM_NO_COVERAGE));
        add_coverage(build_coverage(UVM_NO_COVERAGE));
    endfunction
    // Function : build
    virtual function void build();
        this.Fld1 = uvm_reg_field::type_id::create("Fld1");
        this.Fld1.configure(this, 32,  0, "RW", 0, 32'd0, 1, 1, 0);
    endfunction
endclass
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

User Defined Classes

‘uvm.inst_class’ property

1 IDS_block	IDS_block	Block	0x0
offset	external	size	

1.1 ids_reg	ids_reg	R	
offset	external	size	32 default

{uvm.inst_class=inst_class}

bits	name	s/w	h/w	default	d
31:0	Fld1	Rw	Ro	0	

```
class IDS_block_block extends uvm_reg_block;
    `uvm_object_utils(IDS_block_block)
    rand inst_class ids_reg;
    // Function : new
    function new(string name = "IDS_block_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction
    // Function : build
    virtual function void build();
        //IDS_REG
        ids_reg   =   inst_class::type_id::create("ids_reg");
        ids_reg.configure(this, null, "ids_reg");
        ids_reg.build();
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        default_map.add_reg( ids_reg, 'h0, "RW");
        lock_model();
    endfunction
endclass : IDS_block_block
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Uvm_add_regmap

This creates additional register map, ‘map1’ and ‘map2’ for the same bus domain

1 Top	Top	Block	0x0
offset	external	size	

{uvm_add_regmap=map1, map2}

1.1 foobar	foobar	Reg.	0
offset	external	size 32	default 0x00

bits	name	s/w	h/w	default	description
31:0	Control	Rw	Rw	0	

creates additional register map, ‘map1’ and ‘map2’ for the same bus domain

```
class Top_block extends uvm_reg_block;
    `uvm_object_utils(Top_block)
    rand Top_foobar foobar;
    uvm_reg_map map1;
    uvm_reg_map  map2;
    // Function : new
    function new(string name = "Top_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction
    // Function : build
    virtual function void build();
        //FOOBAR
        foobar    =    Top_foobar::type_id::create("foobar");
        foobar.configure(this, null, "foobar");
        foobar.build();
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);
        map1= create_map("map1", 'h0, 4, UVM_BIG_ENDIAN, 1);
        map2= create_map(" map2", 'h0, 4, UVM_BIG_ENDIAN, 1);
        map1.add_reg( foobar, 'h0, "RW");
        map2.add_reg( foobar, 'h0, "RW");
        default_map.add_reg( foobar, 'h0, "RW");
        foobar.clear_hdl_path();
        foobar.add_hdl_path_slice("foobar_Control_q", 0, 32);
        lock_model();
    endfunction
endclass : Top_block
```

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Properties

Name	Purpose
uvm.base_class	The class which is extended by the generated class
uvm.reg_class	The register classes will be extended by the class mentioned using this property
uvm.reg_access	If all fields in a register are software readable then the software access of register will be "RO", similarly for writeable fields. In case fields in a register are readable as well as writeable or have other special access, then register access will be "RW"
uvm_class	The name of the generated class
uvm.inst_class	The name of the variable that has the instance of the class
uvm.package	The name of the package that is being generated
uvm.user_coverage	Specify all the identifiers globally at the top level of the register specification in the block description. This will generate an enum of type “uvm_reg_cvr_t”
uvm.map	Changes the name of the default map in the UVM output.
is_rand	Specifies if the field is randomizable
is_acc	Specifies if the field can be individually accessible

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Properties

Name	Purpose
dontcompare	Eliminates the specified reg or field from verification. It adds compare (UVM_NO_CHECK) on the specified fields
Index_reg	Name of the index register
depth	Depth of the index register
hdl_path	Hierarchical path to the RTL storage of the element
hdl_path_gate	Hierarchical path to the RTL storage of the element
hdl_path_internal	To prepend the value in it, to the hdl_path. It prepend the value on internal registers only
ignore_prop	ignores the prepended hdl_path for registers
constraint	Specifies the constraint on the value
uvm.reset_constraint	Creates soft constraint reset for fields in register class.
volatile	used to set the volatility of the field
uvm.field_class	The field class name instance is replaced with the value in this property
vertical_reuse	the block level UVM classes will not be regenerated in the chip level register model file,
has_reset	A value of 1 indicates a hard reset, a value of 0 indicates that the reset is ignored by the UVM model

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

UVM Properties

Name	Purpose
uvm.handle_name_format	used to change handle's name
uvm.reg_name_format	used to change the register name
uvm.name_format	used to change the class name
uvm_add_regmap	creates additional register map for the same bus domain.
	creates additional register map with base addresses for the same bus domain.
uvm_global_param	It creates parameter at top in ".remem.sv" file and doesn't create parameterized class.
uvm_lock_model	used to remove lock model call function from generated output
uvm_opt	This property removes UVM factory registration macros. Eg: 'uvm_object_utis' which is used to register uvm objects and uvm components respectively with the factory.
uvm.alter_access	w1t/w1c access is converted to 'wo' access and callback is used to toggle/clear value.
regdef_pkg	supports package definition (<blockname>_regdef_pkg.sv) in case of third party UVM output
auto_volatile	If applied at top level module then all hardware writable fields in the hierarchy will become volatile
uvm_guard_band	If applied at top level module then it removes the default guard banding from the UVM classes

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

About Agnisys

- The EDA leader in solving complex design and verification problems associated with HW/SW interface
- Formed in 2007
- Privately held and Profitable
- Headquarters in Boston, MA
  - ~1000 users worldwide
  - ~50 customer companies
- Customer retention rate ~90%
- R&D centers (US and India)
- Support centers - Committed to ensure comprehensive support
  - Email : support@agnisys.com
  - Phone : 1-855-VERIFYY
  - Response time within one day; within hours in many cases
  - Time Zones (Boston MA, San Jose CA and Noida India)

IDESIGNSPEC™ (IDS) EXECUTABLE REGISTER SPECIFICATION

Automatically generate UVM models, Verilog/VHDL models, Coverage Model, Software model etc.

AUTOMATIC REGISTER VERIFICATION (ARV)

ARV-Sim™ : Create UVM Test Environment, Sequences, Verification Plans and instantly know the status of the verification project.

ARV-Formal™ : Create Formal Properties and Assertions, and Coverage Model from the specification.

ISEQUENCESPEC™ (ISS)

Create UVM sequences and Firmware routines from the specification.

DVinsight™ (DVi)

Smart Editor for SystemVerilog and UVM projects.

IDS – Next Generation™ (IDS-NG)

Comprehensive SoC/IP Spec Creation and Code Generation Tool

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

THANK YOU