Introduction to SystemRDL (Part 2)

CURRENT STATE OF THE ART

Nikita Gulliya

PRESENTER
RND ENGINEER @ AGNISYS

Amanjyot Kaur

PRESENTER
RND ENGINEER @ AGNISYS

Christina Toole

HOST
MARKETING CONSULTANT @ AGNISYS

AGNISYS

Agenda

- Recap of Introduction to SystemRDL Webinar Part 1
- Special Register
  - Interrupt
  - Counters
- Verification Constructs
  - HDL PATH
  - Constraint
  - Structural Testing
- Properties
  - Intrinsic
  - UDPs
- Structures
- Perl preprocessor
- Extension of SystemRDL using UDPs
  - Extending various RTL functionality
  - Extension of UVM functionality
- SystemRDL Usage Methodology
- SystemRDL Alternatives (comparison of SystemRDL with other inputs such as IP-XACT)
- SystemRDL Editor

Recap of SystemRDL Webinar Part 1

- Introduction
  - SystemRDL is a textual representation of Hardware-Software interface consisting of addressable registers, interrupts, counters etc.
- Components
  - Field - lowest-level structural component
  - Register - set of one or more SystemRDL field instances
  - Register File - logical grouping of one or more register and register file instances
  - Address Map - contains registers, register files, memories, and/or other address maps
  - Memory - array of storage consisting of a number of entries of a given bit width
- Signals – create additional ports
- Enum - set of constant named integral values
- Parameters – creating a parameterized specification
- Expression – using complex expression in the specification
- Property Assignment – Dynamic, Standard, Default
- Address Allocation – offset, stride, alignment, fullalign, regalign, compact
- SystemRDL 1.0 Vs SystemRDL 2.0

AGNISYS

Interrupt

HANDLING EXTERNAL EVENTS

In this section …

- **Interrupt Introduction**
- **Example**
- **RTL**
- **UVM**

AGNISYS

Interrupt

Interrupt is a signal generated and sent to the processor by hardware or software indicating an event that needs attention

Keyword	Description
intr	Interrupt, part of interrupt logic for a register
posedge	Interrupt when next goes from low to high
negedge	Interrupt when next goes from high to low
bothedge	Interrupt when next changes value
level	Interrupt while the next value is asserted and maintained (the default)
nonsticky	Defines a non-sticky (hierarchical) interrupt (not locked)
enable	Defines an interrupt enable; i.e., which bits in an interrupt field are used to assert an interrupt
mask	Defines an interrupt mask ; i.e., which bits in an interrupt field are not used to assert an interrupt
haltenable	Defines a halt enable (the inverse of haltmask); i.e., which bits in an interrupt field are set to de-assert the halt out.
haltmask	Defines a halt mask (the inverse of haltenable); i.e., which bits in an interrupt field are set to assert the halt out
sticky	Defines the entire field as sticky; i.e., the value of the associated interrupt field shall be locked until cleared by software (write or clear on read)

```
addrmap block_name {
    reg Status1 {
       regwidth = 32;
        field {
          hw = rw;
          sw = rw;
          onread = r;
          onwrite = woclr;
          intr;
         } Fld[31:0] = 32'h0;
    };
    reg Status2 {
       regwidth = 32;
        field {
          hw = rw;
          sw = rw;
          onread = r;
          onwrite = woclr;
          intr;
         } Fld[31:0] = 32'h0;
    };
    reg Enable1 {
       regwidth = 32;
        field {
          hw = rw;
          sw = rw;
          onread = r;
          onwrite = w;
         } Fld[31:0] = 32'h0;
   };
reg Mask1 {
       regwidth = 32;
        field {
          hw = rw;
          sw = rw;
          onread = r;
          onwrite = w;
         } Fld[31:0] = 32'h0;
    };
    Status1 Status1 @0x0000;
    Status2 Status2 @0x0004;
    Enable1 Enable1 @0x0008;
    Mask1   Mask1   @0x000C;
    Status1.Fld -> enable = Enable1.Fld;
    Status2.Fld -> mask  = Mask1.Fld;
};
```

RTL

```
module blockname_ids(
 .  .  .  .  .
  irq,    // Output interrupt signal
 .  .  .  .  .
  output irq;
.  .  .  .  .
always @(posedge clk)
.  .  .  .  .
  else
   begin
    if (Status1_Fld_in_enb)   // hw driven interrupts

.        .        .
.        .
end
   end // always clk
.  .  .  .  .
 assign irq = ( |(Status1_Fld_q & Enable1_Fld_q)) | ( |(Status2_Fld_q & ~(Mask1_Fld_q))) ;
.  .  .  .  .
endmodule
```

UVM

```
/*-------------------------------------------------------
----------Class : blockname_Status1-------------------
-----------------------------------------*/
`ifndef CLASS_blockname_Status1
`define CLASS_blockname_Status1
 class blockname_Status1 extends uvm_reg;
`uvm_object_utils(blockname_Status1)
  rand intr_reg_field Fld;
. . . . .
/*-------------------------------------------------------
------------------Class : blockname_block-------------
----------------------------------------------*/
`ifndef CLASS_blockname_block
`define CLASS_blockname_block
 class blockname_block extends uvm_reg_block;
 . . . .
 begin
  intr_status_class status_Status1_Fld;
  status_Status1_Fld = new(.estatus(Status1.Fld),
  .pending(null),.enable(Enable1.Fld),.mask(null),
  .overflow(null));
uvm_reg_field_cb::add(Status1.Fld,status_Status1_Fld);
 end
 begin
  intr_status_class status_Status2_Fld;
  status_Status2_Fld = new(.estatus(Status2.Fld),
  .pending(null),.enable(null),.mask(Mask1.Fld),
  .overflow(null));
```

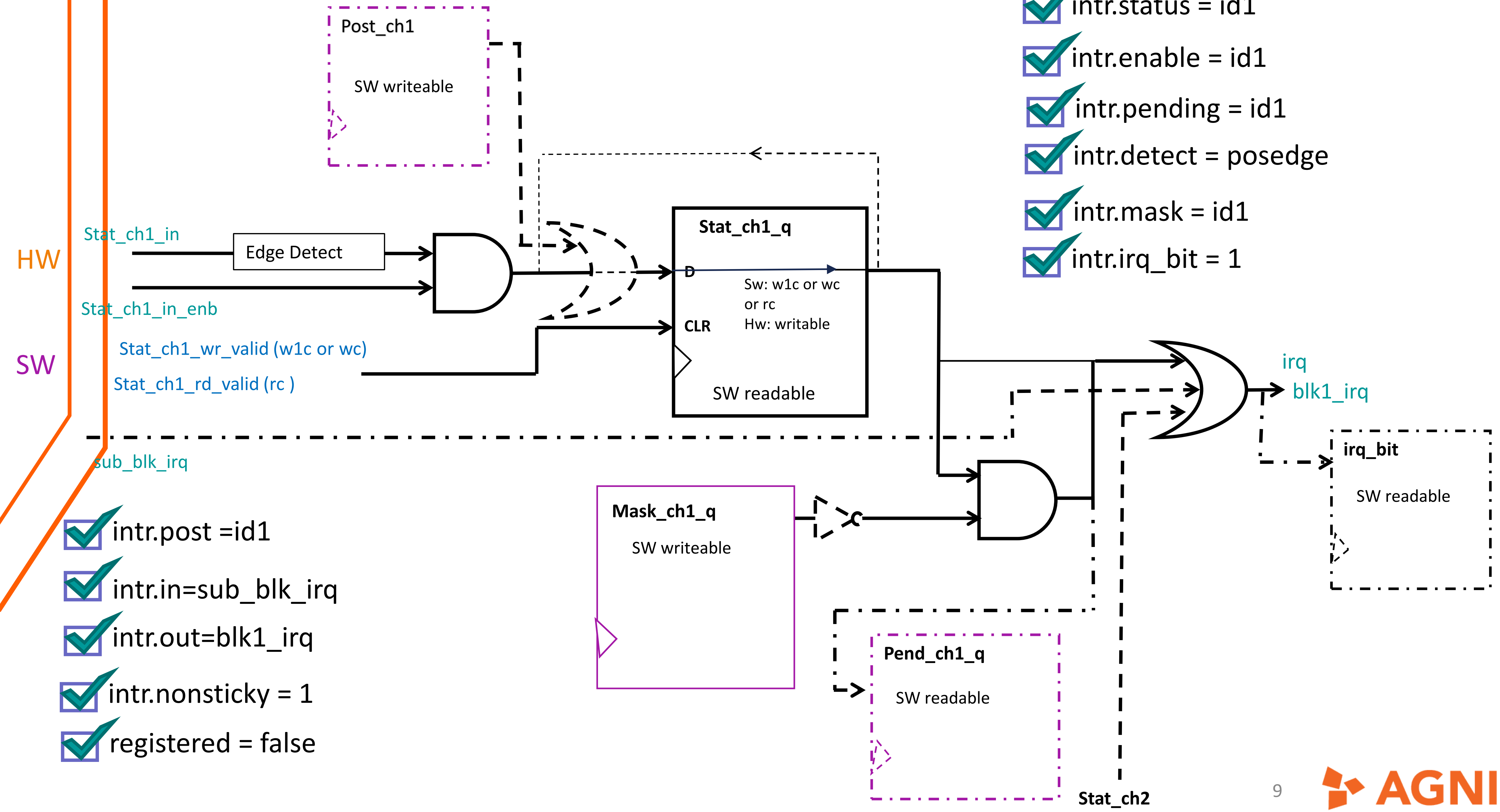
```
uvm_reg_field_cb::add(Status2.Fld,status_Status2_Fld);
 end
 lock_model();
endfunction
task interrupt_monitor;
  fork
  begin
    forever@(intr_hw.Status1_Fld_in)
  begin
   if(intr_hw.Status1_Fld_in_enb)
  begin
void'(Status1.Fld.predict(intr_hw.Status1_Fld_in
|Status1.Fld.get()));
  end
. . . . .
  join
 endtask
. . . . .

. . . . .
```

Callback registration

Interrupt (UDP) Properties

Property Name	Description
intr_irq_per_channel	To generate per channel Interrupt output
intr_in	It is used to specify name of one or multi bit interrupt input signal that needs to be simply ORed ,not registered in flip flops
intr_out	It specifies the name of the output interrupt signal,to translate it into RTL
intr_status	Identifies the status register for the interrupt logic
intr_enable	Identifies the enable register for the interrupt logic
intr_pending	Identifies the pending register for the interrupt logic
intr_irq_bit	The ORed value of all the interrupt channel after enable control logic can be registered and is stored in a field. Output interrupt signal specified in property intr.out is registered in this field
intr_detect	intr.detect is used to specify the detection circuitry for input interupt signal registered in any of register/field identified as status or pending
intr_post	intr.post is used to register software driven interrupts
intr_mask	intr.mask is used to specify the mask register for the interrupt logic
halt_enable	enables the halt signal to propagate to CPU
halt_mask	Halt mask bit corresponding to Status register bit decides that those halt signals will not be allowed to propagate to main halt signal
intr_nonsticky	intr.nonsticky property defines a nonsticky interrupt. The associated interrupt field shall not be locked.

Interrupt Circuit generated



- intr.status = id1
- intr.enable = id1
- intr.pending = id1
- intr.detect = posedge
- intr.mask = id1
- intr.irq_bit = 1

- intr.post =id1
- intr.in=sub_blk_irq
- intr.out=blk1_irq
- intr.nonsticky = 1
- registered = false

AGNISYS

In this section …

- **Counter Introduction**
- **Counter Keyword**
- **Example**
- **RTL**

Counter

COUNTING THE INCREMENTS AND DECREMENTS

Counter

A *counter* is a special purpose field which can be incremented or decremented by constants or dynamically specified values.

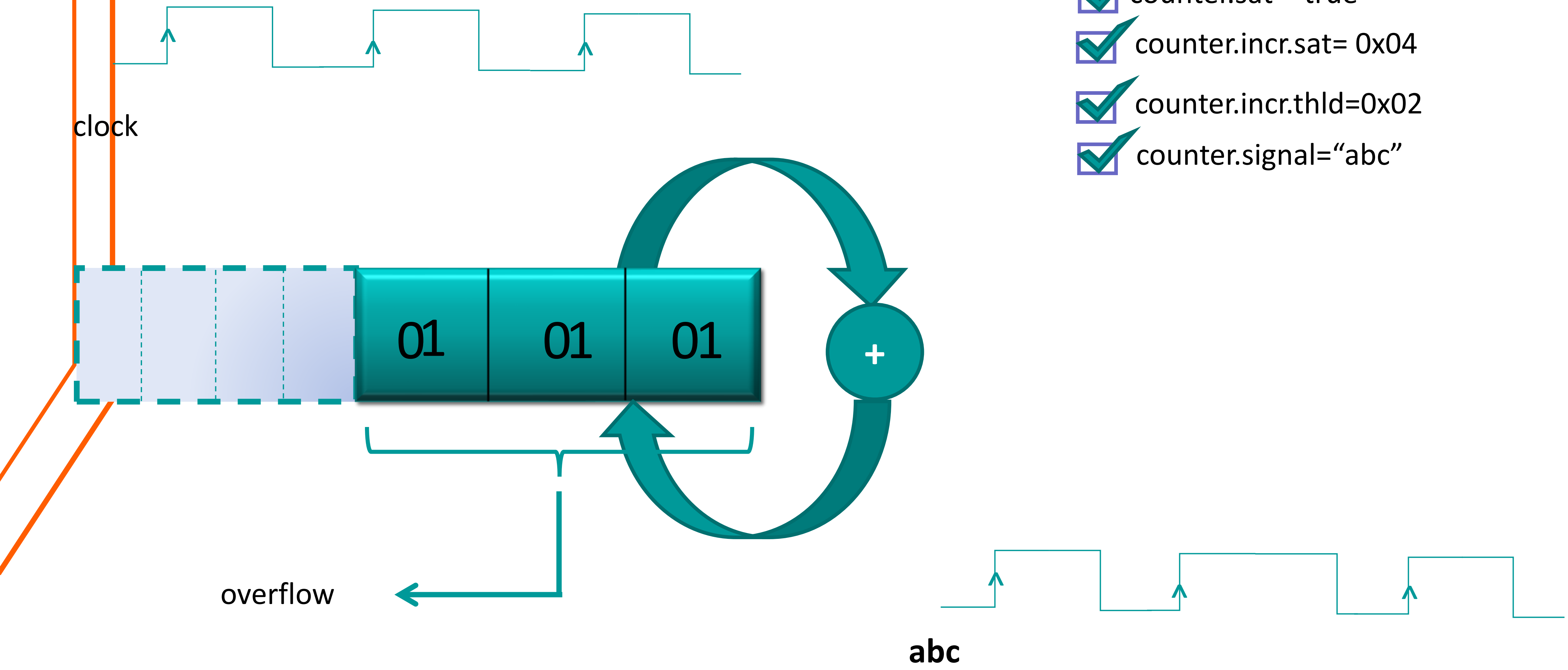
Keyword	Description
counter	Field implemented as a counter.
incrvalue	Increment counter by specified value.
decrvalue	Decrement counter by specified value.
incrsaturate	Indicates the counter saturates in the incrementing direction.
decrsaturate	Indicates the counter saturates in the decrementing direction.
Incrthreshold	Indicates the counter has a threshold in the incrementing direction.
decrthreshold	Indicates the counter has a threshold in the decrementing direction.
decrwidth	Width of the interface to hardware to control decrementing the counter externally.
incrwidth	Width of the interface to hardware to control incrementing the counter externally.
threshold	This is an alias of incrthreshold.
saturate	This is an alias of incrsaturate.
underflow	Underflow signal asserted when counter underflows or wraps.
overflow	Overflow signal asserted when counter overflows or wraps.
incr	The counter increment is controlled by another component or signal (active high).
decr	The counter decrement is controlled by another component or signal (active high).

```
addrmap block_name {
    reg incr_reg {
       regwidth = 32;
        field {
          hw = na;
          sw = rw;
          counter;
          incrvalue = 2;
          incrsaturate = 15;
          incrthreshold = 10;
        } Fld[31:0] = 32'h0;
    };
   reg decr_reg {
       regwidth = 32;
        field {
          hw = na;
          sw = rw;
          counter;
          decrvalue = 2;
          decrthreshold = 10;
          decrsaturate = 5;
        } Fld[31:0] = 32'h0;
    };
 incr_reg incr_reg @0x0000;
 decr_reg decr_reg @0x0004;
};
```


AGNISYS

Counter Circuit

- counter = incr
- counter.incr.val = 0x02
- counter.sat = true
- counter.incr.sat= 0x04
- counter.incr.thld=0x02
- counter.signal="abc"



Property Name	Description
counter_precedence	This property is used to change the sequence of precedence for all three type of counters
counter_sw_wr = incr/decr [,value/"wdata"/field]	This specifies that the counter is incrementing/decrementing for write operation from SW interface. 2nd optional argument to this property is the incrementing/decrementing value which could a- •**positive integer** or •**wdata** for wr_data from bus interface to be the increment/decrement value or •**value in any register field** within the design block.
counter_sw_rd = incr/decr [,positive integer]	This specifies that the counter is incrementing/decrementing for read operation from SW interface. 2nd optional argument is the constant value by which the counter would increment/decrement.
counter_signal { type = string; component = field ; };	This is used to control the increment and decrement events of a counter. It is an Active-High event
counter_hw_enb	To specify enable signal for HW write
counter_sw_wr_enb	To specify enable signal for SW write
counter_sw_rd_enb	To specify enable signal for SW read
counter.both.enb	To generate individual enable for both increment and decrement counter.

In this section …

- **HDL path**
- **Constraint**
- **Structural Testing**

Verification Constructs

TESTING THE DESIGN

HDL PATH

By specifying an HDL path, the verification environment can have direct access to memory, register, and field implementation nets in a Design Under Test (DUT).

An **hdl_path_slice** or **hdl_path_gate_slice** can be put on a **field** or **mem** component. It can be used when the corresponding RTL or gate-level netlist is not contiguous.

Syntax:

hdl_path = "*path*";

hdl_path_gate = "*path*";

hdl_path_slice = '{"*path*" [, "*path*"]*};

hdl_path_gate_slice = '{"*path*" [, "*path*"]*};

Property	Description	Dynamic
hdl_path	Assigns the RTL hdl_path for an addrmap, reg, or regfile	Yes
hdl_path_slice	Assigns a list of RTL hdl_path for a field or mem	Yes
hdl_path_gate	Assigns the gate-level hdl_path for an addrmap, reg, or regfile	Yes
hdl_path_gate_slice	Assigns a list of gate-level hdl_path for a field or mem	Yes

```
addrmap blk_def #(string ext_hdl_path = "ext_block"){
   hdl_path = "int_block" ;
   reg {
     hdl_path = { ext_hdl_path, ".externl_reg" } ;
       field {
         hdl_path_slice = '{ "field1" } ;
       } f1 ;
   } external external_reg ;
   reg {
     hdl_path = "int_reg" ;
       field {
         hdl_path_slice = '{ "field1" } ;
       } f1 ;
   } internal_reg ;
} ;
addrmap top {
  hdl_path = "TOP" ;
  blk_def #( .ext_hdl_path("ext_block0")) int_block0 ;
  int_block0 -> hdl_path = "int0" ;
  blk_def #( .ext_hdl_path("ext_block1")) int_block1 ;
  int_block1 -> hdl_path = "int1" ;
};
```


AGNISYS

UVM

```
. . . . .
. . . . .
this.clear_hdl_path();
this.add_hdl_path("TOP");
intblock0.clear_hdl_path();
intblock0.add_hdl_path("");
intblock0.externalreg.clear_hdl_path();
intblock0.externalreg.add_hdl_path_slice("ext_block0.externl_regfield1", 0, 1);
intblock0.internalreg.clear_hdl_path();
intblock0.internalreg.add_hdl_path_slice("int_regfield1", 0, 1);
intblock1.clear_hdl_path();
intblock1.add_hdl_path("");
intblock1.externalreg.clear_hdl_path();
intblock1.externalreg.add_hdl_path_slice("ext_block1.externl_regfield1", 0, 1);
intblock1.internalreg.clear_hdl_path();
intblock1.internalreg.add_hdl_path_slice("int_regfield1", 0, 1);
lock_model();
endfunction
endclass : top_block
`endif
```

hdl_path

hdl_path_slice

Constraint

A *constraint* is a value-based condition on one or more components; e.g., constraint-driven test generation allows users to automatically generate tests for functional verification.

Definitive definition

constraint *constraint_component_name* **{**[*constraint_body*]**};**

constraint_component_name constraint_inst;

Anonymous definition

constraint {[*constraint_body*]**}**

constraint_component_name;

Property	Description	Dynamic
constraint_disable	Specifies whether to disable (true) or enable (false) constraints	Yes

```
constraint max_value { this < 256; };
enum color {
 red = 0 { desc = " color red ";};
 green = 1 { desc = " color green ";};
};
reg register1 {
  field {
  } limit[0:2]= 0;
  field {
    max_value max1;
  } f1[3:9]= 3;
  field {
    encode=color;
    constraint{this inside{color::red,color::green};}rg1;
  } f2[10:31];
};
addrmap constraint_component_example {
  register1 reg1;
  register1 reg2;
  reg2.f2.rg1->constraint_disable = true;
};
```

UVM

```
`ifndef CLASS_constraintcomponentexample_reg1
`define CLASS_constraintcomponentexample_reg1
class constraintcomponentexample_reg1 extends uvm_reg;
`uvm_object_utils(constraintcomponentexample_reg1)
      typedef enum {
        reg1_f2_e_red   =  0,  //   color red
        reg1_f2_e_green  =  1  //   color green
      } reg1_f2_e_color;

     rand uvm_reg_field limit;
     rand uvm_reg_field f1;
     rand uvm_reg_field f2;
    constraint reg1_f1_constraint
    {
        f1.value[6:0] < 'h100;
    }
    constraint reg1_f2_constraint
    {
       f2.value[21:0] inside   {'h0,'h1};
    }
.  .  .  .  .  .

.  .  .  .  .  .

endclass
`endif
```

Constraint

Structural Testing

1) dontcompare : This is testing property indicates the components read data shall be discarded and not compared against expected results.

2) donttest : This testing property indicates the component is not included in structural testing.

```
addrmap top{
  reg r1{
   dontcompare;
   field{
   } fld1;
  };
  reg r2{
   donttest;
   field{
   } fld1;
  };
 r1 r1 @0x0;
 r2 r2 @0x8;
};
```

```
`ifndef CLASS_top_r1
`define CLASS_top_r1
class top_r1 extends uvm_reg;
`uvm_object_utils(top_r1)
.        .        .
.        .
virtual function void build();
this.fld1 = uvm_reg_field::type_id::create("fld1");
this.fld1.configure(this, 1,  0, "RW", 0, 1'd0, 1, 1, 0);
this.fld1.set_compare(UVM_NO_CHECK);
.        .        .
.        .
class top_r2 extends uvm_reg;
`uvm_object_utils(top_r2)
.        .        .
.        .
irtual function void build();
this.fld1 = uvm_reg_field::type_id::create("fld1");
this.fld1.configure(this, 1,  0, "RW", 0, 1'd0, 1, 1, 0);
uvm_resource_db#(bit)::set({"REG::", this.get_full_name()},
"NO_REG_TESTS", 1, this);
```

dontcompare

donttest

In this section …

- **Intrinsic Properties**
- **User Defined Properties**

Properties

DESCRIBING THE BEHAVIOR

Intrinsic Properties

Properties	Description
swwe	Software write-enable active high
swwel	Software write-enable active low
swmod	Assert when field is modified by software (written or read with a set or clear side effect)
swacc	Assert when field is software accessed
singlepulse	The field asserts for one cycle when written 1 and then clears back to 0 on the next cycle. This creates a single-cycle pulse on the hardware interface
we	Write-enable (active high)
wel	Write-enable (active low)
anded	Logical AND of all bits in field
ored	Logical OR of all bits in field
xored	Logical XOR of all bits in field
fieldwidth	Determines the width of all instances of the field. This number shall be a numeric. The default value of fieldwidth is undefined
hwclr	Hardware clear. This field need not be declared as hardware-writable
hwset	Hardware set. This field need not be declared as hardware-writable
hwenable	Determines which bits may be updated after any write enables has been performed. Bits that are set to 1 will be updated
hwmask	Determines which bits may be updated after any write enables has been performed. Bits that are set to 1 will not be updated
regwidth	Specifies the bit-width of the register (power of two)
accesswidth	Specifies the minimum software access width (power of two) operation that may be performed on the register

Properties	Description
errextbus	The associated external register has error input
intr	Represents the inclusive OR of all the interrupt bits in a register after any field enable and/or field mask logic has been applied
halt	Represents the inclusive OR of all the interrupt bits in a register after any field haltenable and/or field haltmask logic has been applied
shared	Defines a register as being shared in different address maps
mementries	The number of memory entries
memwidth	The memory entry bit width
sw	Programmer’s ability to read/write a memory
alignment	Specifies alignment of all instantiated components in the associated register file
sharedextbus	Forces all external registers to share a common bus
errextbus	For an external regfile, the associated regfile has an error input
alignment	Alignment of all instantiated components in the address map
sharedextbus	Forces all external registers to share a common bus
errextbus	The associated addrmap instance has an error input
littleendian	Uses little-endian architecture in the address map
addressing	Controls how addresses are computed in an address map
rsvdset	The read value of all fields not explicitly defined is set to 1 if rsvdset is True; otherwise, it is set to 0
rsvdsetx	The read value of all fields not explicitly defined is unknown if rsvd-setX is True

Properties	Description
msb0	Specifies register bit-fields in an address map are defined as 0:N versus N:0
lsb0	Specifies register bit-fields in an address map are defined as N:0 versus N:0
enum	It encloses a set of constant named integral values into the enumeration's scope
encode	Binds an enumeration to a field
signalwidth	Width of the signal
sync	Synchronous to the clock of the component
async	Asynchronous to the clock of the component
cpuif_reset	Default signal to use for resetting the software interface logic. This parameter only controls the CPU interface of a generated slave
field_reset	Default signal to use for resetting field implementations
active low	Signal is active low (state of 0 means ON)
active high	Signal is active high (state of 1 means ON)
resetsignal	Reference to the signal used to reset the field
intr	Interrupt, part of interrupt logic for a register
posedge	Interrupt when next goes from low to high
negedge	Interrupt when next goes from high to low
bothedge	Interrupt when next changes value
level	Interrupt while the next value is asserted and maintained (the default)
nonsticky	Defines a non-sticky (hierarchical) interrupt (not locked)
enable	Defines an interrupt enable; i.e., which bits in an interrupt field are used to assert an interrupt

Keyword	Description
mask	Defines an interrupt mask ; i.e., which bits in an interrupt field are not used to assert an interrupt
haltenable	Defines a halt enable (the inverse of haltmask); i.e., which bits in an interrupt field are set to de-assert the halt out
haltmask	Defines a halt mask (the inverse of haltenable); i.e., which bits in an interrupt field are set to assert the halt out
sticky	Defines the entire field as sticky; i.e., the value of the associated interrupt field shall be locked until cleared by software (write or clear on read)
counter	Field implemented as a counter
incrvalue	Increment counter by specified value
decrvalue	Decrement counter by specified value
incrsaturate	Indicates the counter saturates in the incrementing direction
decrsaturate	Indicates the counter saturates in the decrementing direction
Incrthreshold	Indicates the counter has a threshold in the incrementing direction
decrthreshold	Indicates the counter has a threshold in the decrementing direction
decrwidth	Width of the interface to hardware to control decrementing the counter externally
incrwidth	Width of the interface to hardware to control incrementing the counter externally
threshold	This is an alias of incrthreshold
saturate	This is an alias of incrsaturate
underflow	Underflow signal asserted when counter underflows or wraps
overflow	Overflow signal asserted when counter overflows or wraps
incr	The counter increment is controlled by another component or signal (active high)
decr	The counter decrement is controlled by another component or signal (active high)

Property	Description
hdl_path	Assigns the RTL hdl_path for an addrmap, reg, or regfile
hdl_path_slice	Assigns a list of RTL hdl_path for a field or mem
hdl_path_gate	Assigns the gate-level hdl_path for an addrmap, reg, or regfile
hdl_path_gate_slice	Assigns a list of gate-level hdl_path for a field or mem
constraint_disable	Specifies whether to disable (true) or enable (false) constraints
donttest	This testing property indicates the component is not included in structural testing
dontcompare	This is testing property indicates the components read data shall be discarded and not compared against expected results

User-defined properties (UDP)

User-defined properties enable the creation of custom properties that extend the structural component types in a SystemRDL design.

Syntax

property *property_name* **{***attribute***;** [*attribute***;**]***};**

e.g. property property_name { type = boolean|string|number|reference ; component = addrmap|regfile|reg|field;};

Note: UDP doesn't contain any keyword of the SystemRDL.

General Properties

Property Name	Description
explicit_name	Property to create hierarchical name signals.
byte_addressing	Support accessing individual bytes of data.
count	Works same as repeat.
lock	Locks the software access of a register depending on the value
output_file_name	To change the generated output file name with the specified property value
doc	Use to add description
reg_sw	Specify the sw access for register
reg_hw	Specify the hw access for register
reg_default	Specify the default value for register

RTL Properties (UDP)

Property Name	Description
clock_edge	Specifies the clock edge used for implementing registers
reset_type	Specifies the type of reset for generated registers
reset_level	Specifies the level of the reset for the registers
rtl_precedence	To prioritize HW/SW for write operation
registered	Indicates whether to register the signal coming into the generated module is from the hardware side. Hardware access must be set to writeable
module_name	Changes the name of the module in the verilog output
vhdl_entity	Changes the name of the entity in the VHDL output
vhdl_package	Changes the name of the generated VHDL package
rtl_name_format	Required for formating of signal.
rtl_hw_w1p	A pulse is generated whenever 1 is written to a field through the bus interface
rtl_hw_w0p	A pulse is generated whenever 0 is written to a field through the bus interface
rtl_hw_wp	A pulse is generated whenever a write happens to a field through the bus interface
rtl_hw_rp	A pulse is generated whenever a field is read through the bus interface
rtl_hw_clear	A field can be cleared whenever a signal from the application logic or hardware interface is asserted high
buffer_trig	Used to specify a field of any register as a trigger
clock_name	This specifies the clock name on which the register exists
endianness	Endianness to select the trigger address

RTL Properties (UDP) Contd..

Property Name	Description
rtl_hw_enb	Create an input enable signal on the hardware interface when the field is HW writable
rtl_reg_enb	Create an output signal '<reg>_enb' on the hardware interface if SW access of the register is writable irrespective of the HW access
default_clock_name	To customize clock name
default_reset_name	To customize hard reset name
virtual	To specify an indirect address map
indirect_map	To specify an indirect address map used with property "virtual=true"
datagroup	Four 8-bit registers (Data0, Data1, Data2, Data3) are grouped together by specifying the property “datagroup = A” on each of them. Where the value “A” is the group name.
hw_rd_trigger	A write/read to the “Index” register (8bit) triggers read from the indirect register (32bit)
hw_wr_trigger	A write/read to the “Data3” register (8bit) triggers write to the indirect register (32bit)
rtl_default	To customize the register reset value to be driven by external signal
reg_wprot	To protect the register from the SW (AXI bus) side
rtl_axi4prot	For '0' assignment to protection signals in IDS generated RTL
reg_rprot	To protect the register to read its value from the SW (AXI bus) side
reg_prot	To protect the register from the SW (AMBA and AMBA3AHBLITE bus) side

RTL Properties Example

RDL

```
property module_name {type = string; component = addrmap;};
property clock_edge {type = string; component = addrmap|reg;};
property reg_prot {type = string; component = reg;  };
property reg_wprot {type = string; component = reg;  };

addrmap block_name {
   module_name = "BLOCK1" ;
   clock_edge = "negedge" ;
    reg reg_name {
       regwidth = 32;
       reg_prot = "priv,secure";
        field {
        } Fld[31:0] = 32'h0;
    };
    reg reg_name1 {
       regwidth = 32;
       reg_wprot = "priv,secure";
        field {
        } Fld[31:0] = 32'h0;
    };
 reg_name   reg_name @0x0000;
 reg_name1 reg_name1 @0x0004;
};
```

RTL

```
module BLOCK1(
 . . . .
 // Write/Read Protection signals
 wire reg_name_wprot ;
 wire reg_name_rprot ;
 . . . .
 // Register Protection : reg_name
 // Write Protection : Privileged and Secure Access
   assign reg_name_wprot =  & ~hprot_i[1]hprot_i[1];
 // Read Protection :
   assign reg_name_rprot =  & ~hprot_i[1]hprot_i[1]
// Register Protection : reg_name1
// Write Protection : Privileged and Secure Access
   assign reg_name1_wprot =  & ~hprot_i[1]hprot_i[1];
  . . . .
 always @(negedge clk)
  . . . .
   assign reg_name_write_error = reg_name_decode &&
      wr_stb && ~reg_name_wprot;
   assign reg_name_read_error = reg_name_decode &&
      rd_stb && ~reg_name_rprot;
. . . .
```

module_name

clock_edge

reg_prot

UVM Properties

Property Name	Description
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”
index_reg	Name of the index register
depth	Depth of the index register
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
uvm_reset_constraint	Creates soft constraint reset for fields in register class.
volatile	use to set the volatility of the field
auto_volatile	If applied at top level module then all hardware writable fields in the hierarchy will become volatile.
config_hdl_path	When this property is applied at top with 'false' value then IDS will not add HDL path in configure function
coverage	Generates appropriate coverage code in the output. Note :- with coverage=off property at top,the build_coverage and add_coverage functions can be removed from the generated uvm regmodel

Struct

CREATE GENERIC SPECIFICATION

In this section …

- **Structure Introduction**
- **Output Based**

Struct

It enable the creation of structured properties for more complex extension of component types.

Syntax: A **struct** definition appears as follows.

[**abstract**] **struct** *struct_name* [**:** *base_struct_name*]
{{*member_type member_name*;}***};**

Deriving structures

A **struct** declaration may *derive* from another **struct** by specifying the base **struct**'s name after a colon (:),

```
struct base_struct {
  bit foo ;
} ;

struct derived_struct : base_struct {
  longint unsigned bar ;
} ;

struct final_struct : derived_struct {
 // final_struct's members are foo, bar, and baz.
  string baz ;
} ;
```

```
 struct configIP {
   boolean Reg1_is_present;
   boolean Reg2_is_present;
 };
 struct configTop {
   configIP IP1;
   configIP IP2;
 };
addrmap ip #(configTop t){
  reg r1 {
    ispresent = t.IP1.Reg1_is_present;
    field {}f1;
  };
  reg r2{
    ispresent = t.IP2.Reg2_is_present;
    field {}f1;
  };
  r1 r1;
  r2 r2;
};
addrmap top {
  ip #(.t(configTop'{IP1:configIP'{Reg1_is_present:true},
                     IP2:configIP'{Reg2_is_present:false} } ) ) ip1;
  ip #(.t( configTop'{IP1:configIP'{Reg1_is_present:false},
                      IP2:configIP'{Reg2_is_present:true} } ) ) ip2;
};
```

RTL

```
module top_ids(
 // REGISTER : REG1 PORT SIGNAL
    ip1_reg1_enb,
    ip1_reg1_f1_in,
    ip1_reg1_f1_in_enb,
    ip1_reg1_f1_r,
 // REGISTER : REG2 PORT SIGNAL
    ip2_reg2_enb,
    ip2_reg2_f1_in,
    ip2_reg2_f1_in_enb,
    ip2_reg2_f1_r,
 . . . . .
 always @(posedge clk)
   begin
    if (!reset_l)
     begin
      ip1_reg1_f1_q <= 1'd0;
     end
. . . . .
end // always clk
. . . . .
always @(posedge clk)
   begin
    if (!reset_l)
     begin
      ip2_reg2_f1_q <= 1'd0;
. . . . .
end // always clk
. . . . .
```

UVM

```
/*--------------------------------------------------------
--------Class : top_ip2---------------------------------
---------------------------*/
`ifndef CLASS_top_ip2
`define CLASS_top_ip2
 class top_ip2 extends uvm_reg_file;
  `uvm_object_utils(top_ip2)
   rand top_ip2_reg2 reg2;
. . . . .
. . . . .

endclass
`endif

/*--------------------------------------------------------
----Class : top_ip1-------------------------------------
---------------------------*/
`ifndef CLASS_top_ip1
`define CLASS_top_ip1
 class top_ip1 extends uvm_reg_file;
  `uvm_object_utils(top_ip1)
   rand top_ip1_reg1 reg1;
. . . . .
. . . . .

endclass
`endif
```

In this section …

- Embedded Perl Preprocessor
- Example
- Verilog-style preprocessor directives

Preprocessor Directives

FILE INCLUSION AND TEXT SUBSTITUTION

Perl Preprocessor

- Perl snippets shall begin with **<%** and be terminated by **%>**; between these markers any valid Perl syntax may be used.
- Any SystemRDL code outside of the Perl snippet markers is equivalent to the Perl print 'RDL code' and the resulting code is printed directly to the post-processed output.
- **<%=$VARIABLE%>** (no whitespace is allowed) is equivalent to the Perl print $VARIABLE.
- The resulting Perl code is interpreted, and the result is sent to the traditional Verilog-style preprocessor.

Directive	Defining standard	Description
`define	SystemVerilog	Text macro definition
`if	Verilog	Conditional compilation
`else	Verilog	Conditional compilation
`elsif	Verilog	Conditional compilation
`endif	Verilog	Conditional compilation
`ifdef	Verilog	Conditional compilation
`ifndef	Verilog	Conditional compilation
`include	Verilog	File inclusion
`line	Verilog	Source filename and number
`undef	Verilog	Undefine text macro

```
reg myReg { <% for( $i = 0; $i < 6; $i += 2 ) {
%> myField data<%=$i%> [<%=$i+1%>:<%=$i%>]; <% } %>
};
```

```
reg myReg {
 myField data0 [1:0];
 myField data2 [3:2];
 myField data4 [5:4];
};
```

SystemRDL Usage Methodology

HANDLING SYSTEMRDL PROJECT

Including Multiple File

```
addrmap BlockA {
  name = "BlockA Address Map";


  reg RegA1 {
     regwidth = 32;
    field {
      hw = rw;
      sw = rw;


      desc = "Hi I am field of register A";
    } FA1[31:0] = 32'h0;
```

```
addrmap BlockB {
  name = "BlockB Address Map";


  reg RegB1 {

  desc = "I am register B1.";
     regwidth = 32;
    field {
      hw = rw;
      sw = rw;

    } FB1[31:0] = 32'h0;

  };
```

```
addrmap BlockC {
  name = "BlockC Address Map";


  reg RegC1 {
     regwidth = 32;
    field {
      hw = rw;
      sw = rw;

    } FC1[31:0] = 32'h0;

  };
```

```
`include "BlockA.rdl"
`include "BlockB.rdl"
`include "BlockC.rdl"


addrmap top {
  name = "top Address Map";

  BlockA BlockA   @0x000;
  BlockB BlockB   @0x500;
  BlockC BlockC  @0x1000;
};
```

IDSBatch Output:

```
17-Sep-19  05:54 PM    <DIR>          .
17-Sep-19  05:54 PM    <DIR>          ..
17-Sep-19  05:54 PM            15,481 BlockA.v
17-Sep-19  05:54 PM            10,525 BlockB.v
17-Sep-19  05:54 PM            10,510 BlockC.v
17-Sep-19  05:54 PM            10,202 ids_top_amba_aggregation.v
17-Sep-19  05:54 PM             9,855 top.v
               5 File(s)         56,573 bytes
               2 Dir(s)  334,284,574,720 bytes free
```

In this section …

- IP-XACT
- YAML
- IDSWord
- IDSExcel

SystemRDL Alternatives

FINDING OUT SOME MORE OPTIONS

IP-XACT

- This standard provides the EDA vendors, IP providers, and SoC design communities with a well-defined and unified specification for the meta-data that represents the components and designs within an electronic system.
- IP-XACT can also describe components for memory and register maps, but quite limited in this area, especially as new types of register designs are needed to meet new requirements of next-generation SoCs.
- To address these limitations, SystemRDL was also formulated by the same industry.

SystemRDL	IP-XACT 2014
addrmap	<ipxact:memoryMap>
addrmap	<ipxact:addressBlock>
<block name> <block_instance> @offset	<ipxact:addressOffset>
external	none(can be handled with the help of vendor extension)
regwidth	<ipxact:width>
regfile	<ipxact:registerFile>
<regroup name> <regroup_instance>@offset	<ipxact:addressOffset>
reg	<ipxact:register>
reg <register_name>	<ipxact:name>
<reg name> <reg_instance>@offset	<ipxact:addressOffset>
hw=wo/rw	<ipxact:volatile>
default	<ipxact:reset>
desc	<ipxact:description>
field	<ipxact:field>
[Lsb]	<ipxact:bitOffset>
[Msb : Lsb]	<ipxact:bitWidth>
sw	<ipxact:access>
[<repeat_value>]	<ipxact:dim>
User Defined Properties	Vendor Extensions

RDL

```
addrmap block_name {
  addrmap reg_group {
    reg Reg1 {
        regwidth = 32;
        field {
        } Fld1[31:0] = 32'h0;
    };
    Reg1 Reg1 @0x0;
  };
  reg Reg2 {
     regwidth = 32;
     field {
     } Fld2[31:0] = 32'h0;
  };
   reg_group  reg_group @0x0;
   Reg2 Reg2 @0x4;
};
```

YAML

```
node:
     type: block
     name: block_name
     node:
     -
         type: section
         name: reg_group
         node:
         -
             type: reg
             name: Reg1
             offset:
             field:
             -
                  name: Fld1
                  offset: 31:0
                  defaultVal: 0
                  sw_access: rw
                  hw_access: rw
     -
          type: reg
          name: Reg2
          offset:
          field:
          -
               name: Fld2
               offset: 31:0
               defaultVal: 0
               sw_access: rw
               hw_access: rw
```

IDSWord

1 block_name		block_name		Block	0x0
offset		external		size	

1.1 reg_group			reg_group		RegGrp		0x0
offset		external		repeat		size	

1.1.1 Reg1			Reg1		Reg.		0x0
offset		external		size	32	default	0x00000000

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

End RegGroup

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

bits	name	s/w	h/w	default	description
31:0	Fld2	Rw	Rw	0	

IDSExcel

block	section name	register	width	description	field	sw access	hw access	field default	bits
block_name									
	reg_group								
		Reg1	32						
					Fld1	rw	rw	0	[31:0]
	end section								
		Reg2	32						
					Fld2	rw	rw	0	[31:0]

IDSBatch- SystemRDL Compiler

- IDesignSpec can be used in Batch mode to input SystemRDL and automatically generate various outputs from it such as :
  - RTL
  - UVM
  - Headers
  - Documentation – HTML, PDF, Word
- A simple IDSBatch command line can be used to generate such outputs with various additional switches to modify the generated output such as selection of bus protocols (AMBA-AHB, APB, AXI, Avalon, OCB, proprietary , I2C, SPI etc.)
- SystemRDL Compiler:
  - Fast and robust
  - Strictly compliant to SystemRDL standard
  - Error and Warnings
- Lots of satisfied customers who are using the compiler

Agnisys' Solutions

IDesignSpec (IDS)

Create Models

ARV-Sim

Create Test Sequences & Environment

ARV-C

Create Test Sequences & Environment in C

ARV-Formal

Create Formal Properties and Assertions

ISequenceSpec

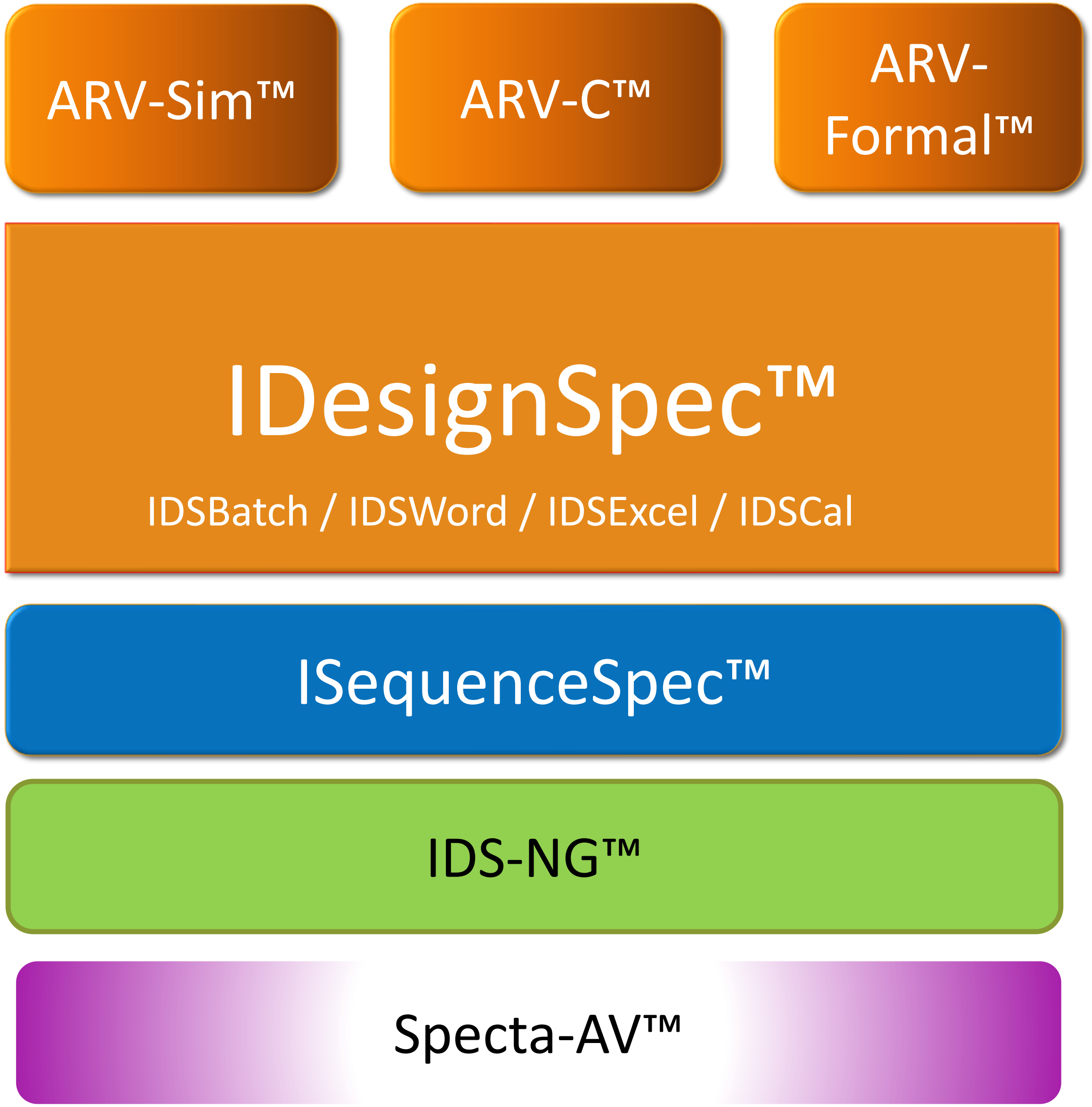
Create UVM sequences and Firmware routines from the specification

IDS-NextGen

Cross-platform HSI Layer Specification

Specta-AV

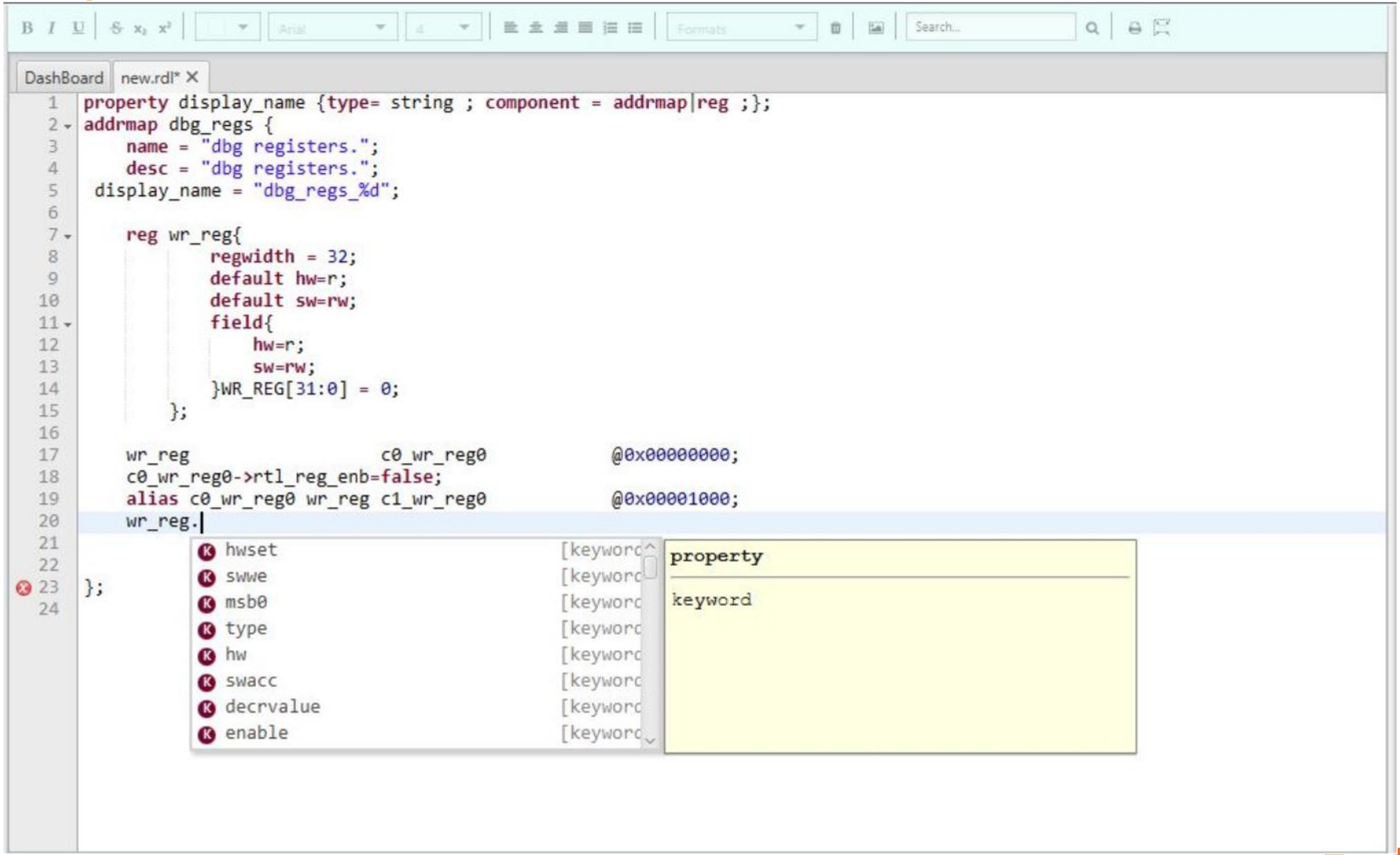
Automatic Verification



SystemRDL Editor

- SystemRDL editor is available as a part of IDS-NG
- User can write input SystemRDL file in the editor
- Keywords are highlighted which makes effective code visibility
- Auto completion of components is also possible (e.g. bracket, semicolon completion)
- The tool indicates syntax error for every line, simultaneously, while writing the spec
- The tool also provides keyword hinting, and it can also hint to the component names used within the file during instantiation or dynamic assignment.
- At the end, the entire input file can be checked for compilation and syntax errors
- Suggestions for error resolution are also provided
- User can check and generate the input file as well from the tool
- Evaluation Request : support@agnisys.com

SystemRDL Editor:



Agnisys, Inc.

www.agnisys.com

support@agnisys.com

PH: 1 (855) VERIFYY

1 (855) 837-4399

- IDesignSpec™
- ISequenceSpec™
- IDS NextGen™
- ARV™
- DVInsight-Pro™