2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SystemRDL to PSS

BASIC TO PRO

Anupam Bakshi

Amanjyot Kaur

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

accellera SYSTEMS INITIATIVE

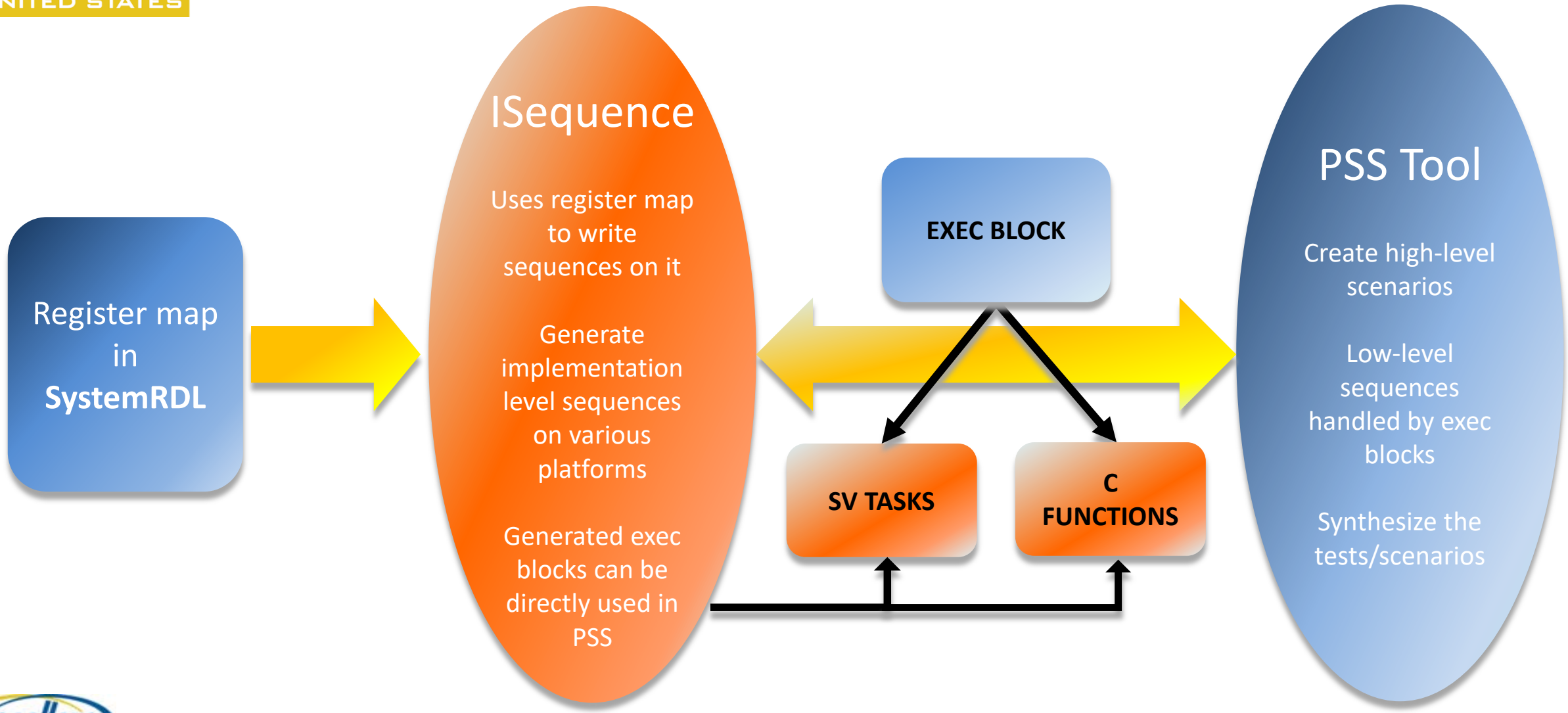
2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Introduction
- Components
  - Field
  - Register
  - Register File
  - Address Map
  - Memory
- Signals
- Address Allocation
- Enumerations
- Parameters
- Structures
- Property Assignment
- Special Register
  - Interrupt
  - Counters
- Verification Constructs
  - HDL PATH
  - Constraint
  - Structural Testing
- Perl preprocessor
- SystemRDL Usage Methodology
- SystemRDL Editor
- Introduction to Portable Standard Stimulus (PSS)
- SoC HW/SW Interface Layer (HSI)
- Example Sequence with HSI
- Introduction to Sequences
- Problems faced with sequences
- Proposed Solution
- ISS+PSS tool flow example: WISHBONE DMA
- Conclusion

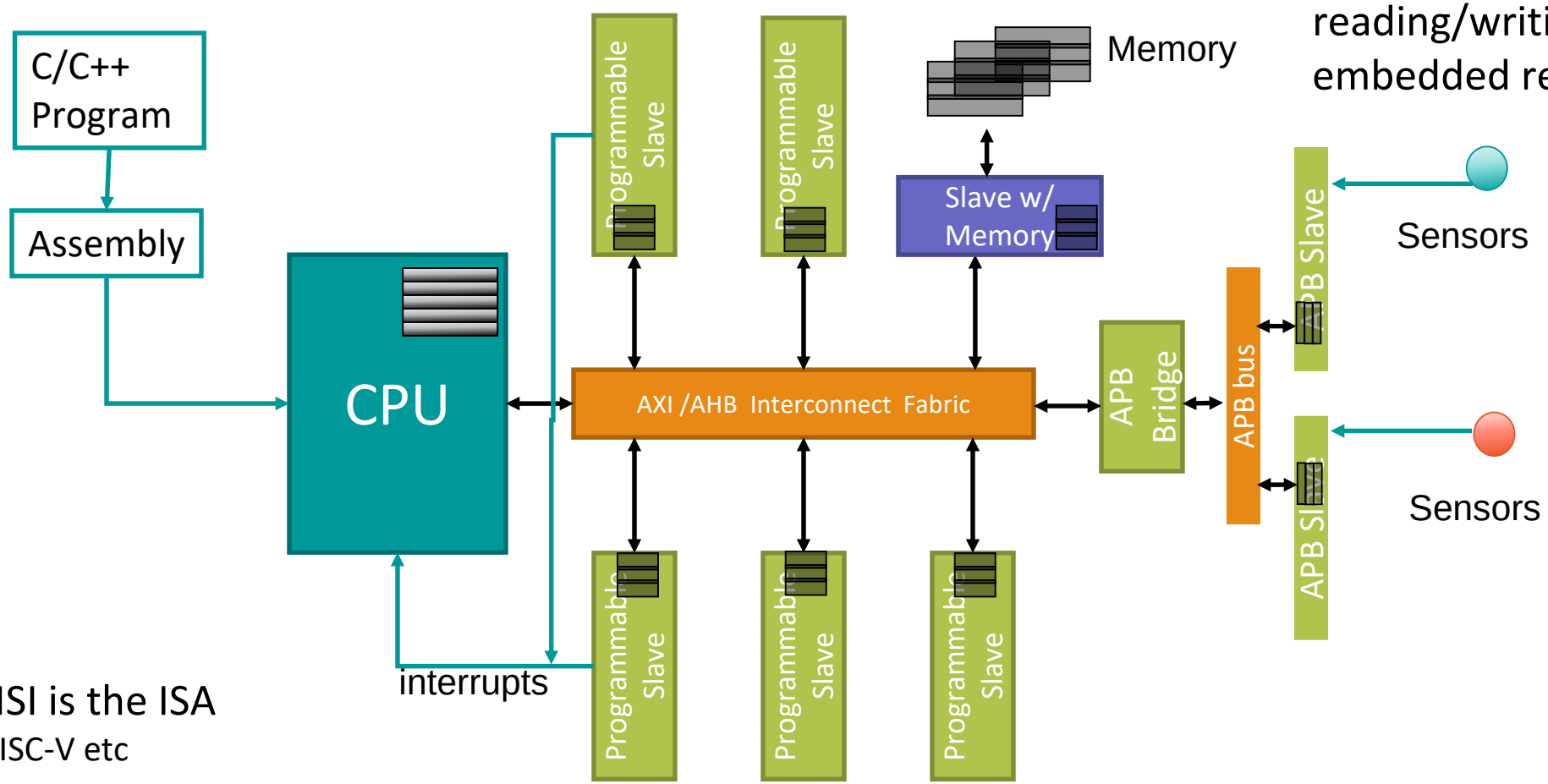
accellera SYSTEMS INITIATIVE

SystemRDL to PSS



CPU & IP Hardware Software Interface

The slaves are programmed by reading/writing to the embedded register



CPU HSI is the ISA

ARM, RISC-V etc

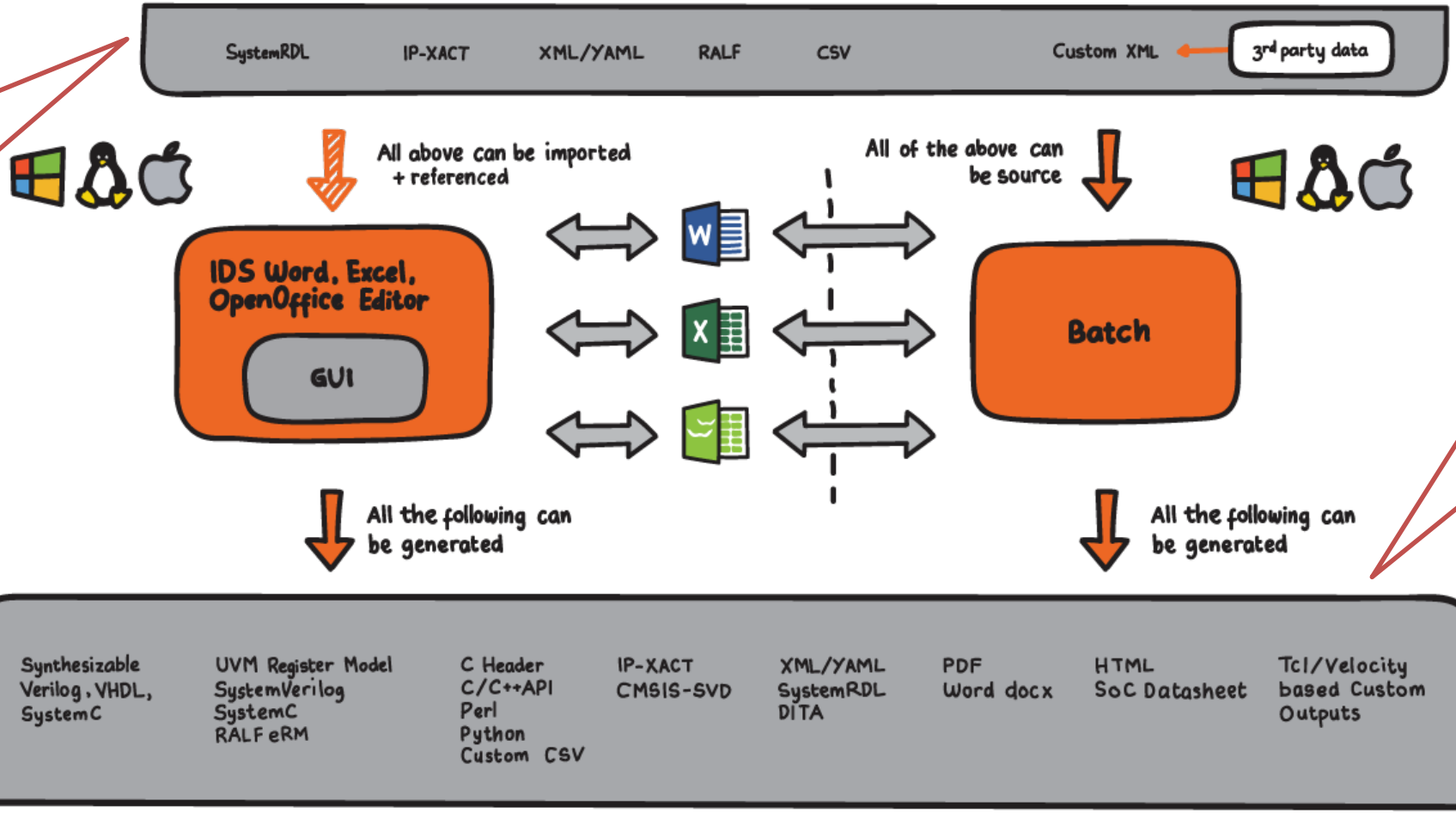
SystemRDL Importance and History

- An embedded system consists of Hardware and Software components.
- SystemRDL is a textual representation of Hardware-Software interface consisting of addressable registers, interrupts, counters etc.
- History
  - Created at Cisco, released as Accellera 1.0 standard.
  - Version 2.0 released in Jan 2018
    - Added Verification constructs, parameterization, data types etc.
    - Reference: https://www.accellera.org/images/downloads/standards/systemrdl/SystemRDL_2.0_Jan2018.pdf
- Support specification centric flow, automatically generate
  - RTL bus interface
  - Verification model
  - C header and API
  - Documentation

IDesignSpec™ – Centralize Register Design/Verification from a Golden Specification

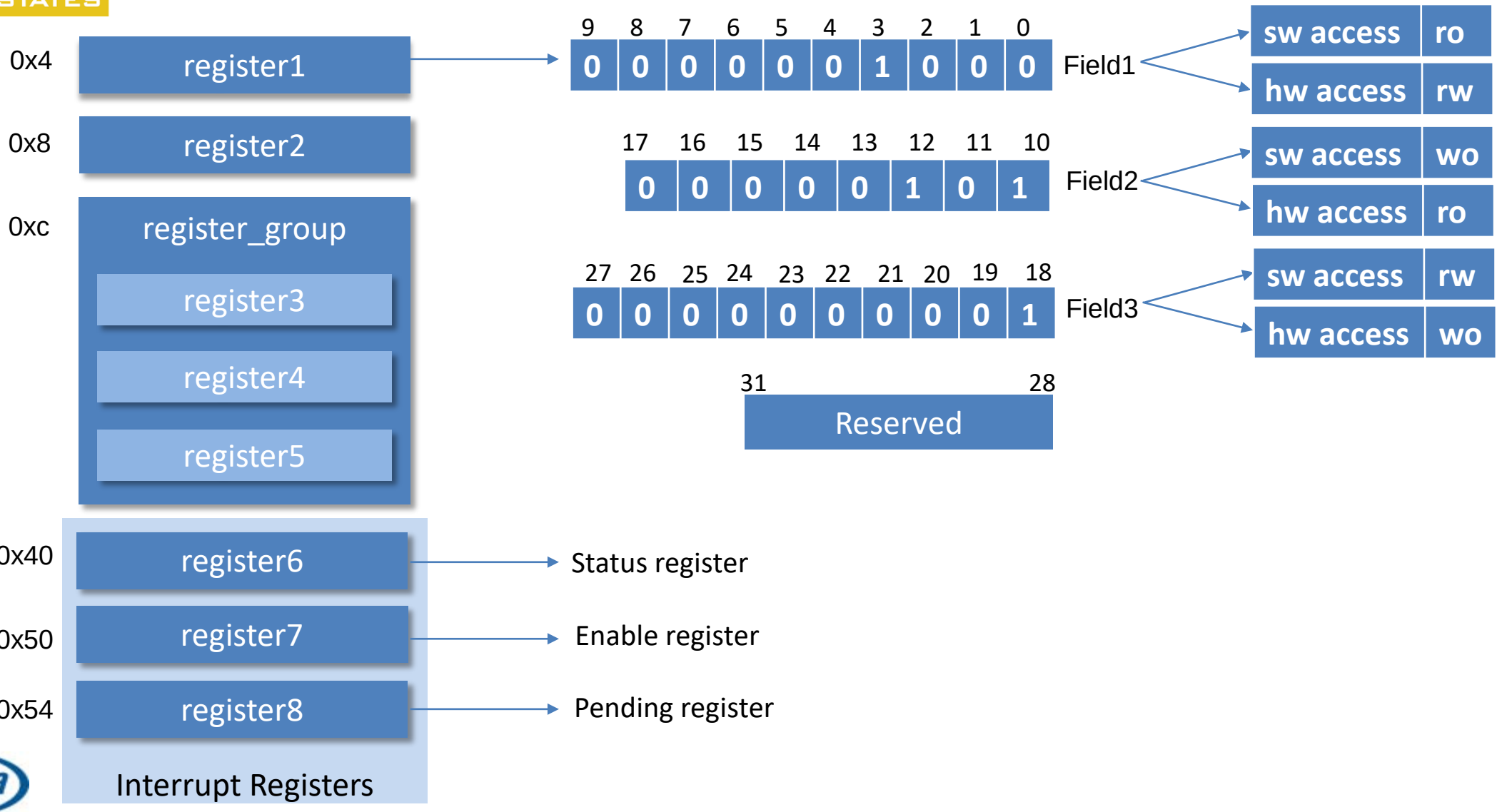
IDesignSpec™ helps IP/SoC Design architects and engineers to create an executable specification for registers and automatically generate output for SW and HW teams.

Specifications can be written in **MS Word, MS Excel, LibreOffice** or text-based industry standard formats such as **SystemRDL, RALF or IP-XACT.**



IDesignSpec captures simple as well as **special registers, signals, interrupts, and generates synthesizable RTL, UVM model, C/C++ Headers, HTML or PDF.**

Example


0x4
register1
0x8
register2
0xc
register_group
register3
register4
register5
0x40
register6
0x50
register7
0x54
register8
Interrupt Registers
Status register
Enable register
Pending register
9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 1 0 0 0
Field1
sw access ro
hw access rw
17 16 15 14 13 12 11 10
0 0 0 0 0 1 0 1
Field2
sw access wo
hw access ro
27 26 25 24 23 22 21 20 19 18
0 0 0 0 0 0 0 0 0 1
Field3
sw access rw
hw access wo
31 28
Reserved

Defining Components

Definitive definition :

In definitive definition we instantiate the component in a separate statement. It is suitable for reuse.

```
addrmap top {
  regfile reggrp1 {
   reg r1 {
     regwidth = 32;
     field f1 {
       hw = rw;
       sw = rw;
     };
     f1 field1[31:0] = 31'b0;
   };
   r1 reg1[3] @0x100;
  };
  reggrp1 reggrp1;
};
```

regwidth

Bit information

Default value

Offset value

Repeat value

Anonymous definition:

In Anonymous definition we instantiate the component in the same statement. It is suitable for components that are used once.

```
addrmap top{
        regfile {
           reg {
            desc="Specify the register";
            field {} field1;
           } reg1;
    } reggrp1;
};
```

Describes the component's purpose.

Default regwidth = 32, fieldwidth=1, offset values = 0, default value =0 sw=rw, hw=rw, taken

Field

The **field** component is the lowest-level structural component, it stores the bit information of a register .

Field ordering in registers

Field ordering in registers	Syntax
lsb0	field_type field_instance [high:low]
msb0	field_type field_instance [low:high]

Definitive field definition:

field [**#(**field_parameter_instance [**,** field_parameter_instance]*)] field_instance_element [**,** field_instance_element]***;**

e.g. field f { };
f f1;

Anonymous field definition:

field {field_body} field_instance_element [,field_instance_element]*;

e.g. field { } f1, f2;

e.g.

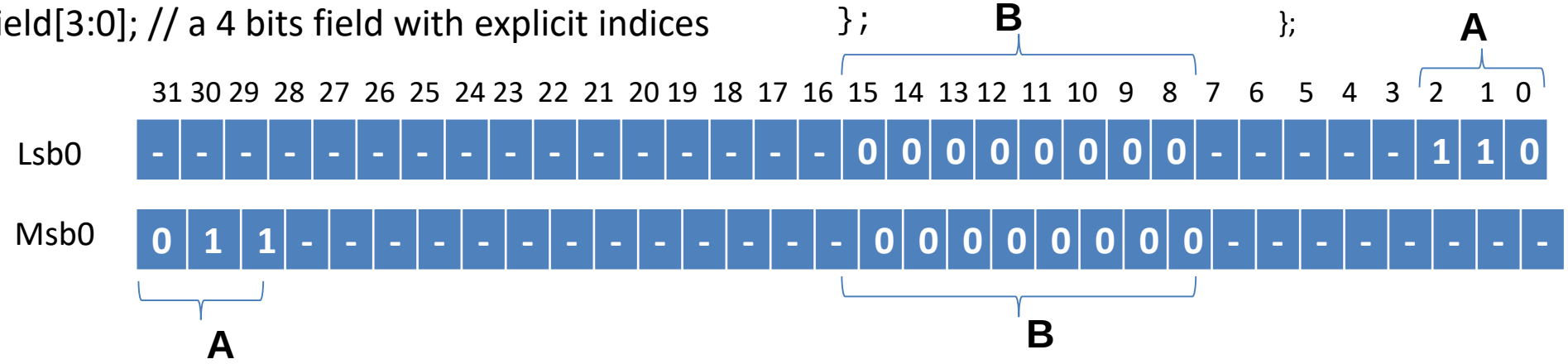
field { } singlebitfield; // 1 bit wide, not explicit about position
field { } somefield[4]; // 4 bits wide, not explicit about position
field { } somefield[3:0]; // a 4 bits field with explicit indices

```
addrmap top{
 lsb0;
 reg{
  field {} A[3]= 3'b110;
  field {} B[15:8];
 } regA;
};
```

3 bits from 2 down to 0

```
addrmap top{
 msb0;
 reg{
  field {} A[3]= 3'b110;
  field {} B[8:15];
 } regA;
};
```

3 bits from 29 down to 31



Software Access Properties

Properties	Description	Dynamic
rclr	Clear on read	Yes
rset	Set on read	Yes
onread	Read side-effect	Yes
woset	Write one to set	Yes
woclr	Write one to clear	Yes
onwrite	Write function	Yes
swwe	Software write-enable active high	Yes
swwel	Software write-enable active low	Yes
swmod	Assert when field is modified by software (written or read with a set or clear side effect)	Yes
swacc	Assert when field is software accessed	Yes
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	Yes

```
reg register1{
  field {} fld1,fld2;
  field {
   hw = rw;
   sw = rw;
   onread  = rclr;
   onwrite = woset;
   swacc;
  } fld3;
   field {
   hw = r;
   sw = w;
   singlepulse;
  } fld4;

};
addrmap myAmap{
 register1 reg1;
 reg1.fld1 -> swwel = true;
 reg1.fld2 -> swmod = true;
};
```

Hardware Access Properties

Property	Description	Dynamic
we	Write-enable (active high)	Yes
wel	Write-enable (active low)	Yes
anded	Logical AND of all bits in field	Yes
ored	Logical OR of all bits in field	Yes
xored	Logical XOR of all bits in field	Yes
fieldwidth	Determines the width of all instances of the field. This number shall be a numeric. The default value of fieldwidth is undefined	Yes
hwclr	Hardware clear. This field need not be declared as hardware-writable	Yes
hwset	Hardware set. This field need not be declared as hardware-writable	Yes
hwenable	Determines which bits may be updated after any write enables. Bits that are set to 1 will be updated	Yes
hwmask	Determines which bits may be updated after any write enables. Bits that are set to 1 will not be updated	Yes

```
reg register1{
  field {
    fieldwidth = 5;
  }fld1,fld2,fld3;
  field {}fld4;
  field {}fld5;

};
addrmap myAmap{
 register1 reg1,reg2;
 reg1.fld1 -> we  = true;
 reg1.fld2 -> wel = true;
 reg1.fld3 -> anded = true;
 reg2.fld1 -> hwenable =
reg1.fld1;
};
```

Register

A register is defined as a set of one or more SystemRDL field instances that are atomically accessible by software at a given address.

Definitive register definition

[**external**] *reg_name* [**#(***parameter_instance* [**,** *parameter_instance*]***)**]
reg_instance_element [**,** *reg_instance_element*]* **;**

Anonymous register definition

reg {[*reg_body*]**}**
[**external**] *reg_instance_element* [**,** *reg_instance_element*]***;**

Register Instantiation into three forms:

Register Instantiation forms	Description
internal	all register logic is created by the SystemRDL compiler for the instantiation (the default form)
external	the register/memory is implemented by the designer and the interface is inferred from instantiation
alias	Alias registers are used where designers want to allow alternate software access to registers. SystemRDL allows designers to specify alias registers for internal or external registers

```
  reg reg1 {
    field {
     hw=w;
     sw=rw;
    } field1;
  };
  reg some_intr {
    field {
     hw=w;
     sw=rw;
     onwrite = woclr;
    } field2;
  };
addrmap foo {
  some_intr  event1;
  external reg1 reg1;
  alias event1 some_intr
event1_for_dv;
};
```

Register Properties

Properties	Description	Dynamic
regwidth	Specifies the bit-width of the register (power of two)	No
accesswidth	Specifies the minimum software access width (power of two) operation that may be performed on the register	Yes
errextbus	The associated external register has error input	No
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	No
shared	Defines a register as being shared in different address maps	No

RDL:

```
addrmap top {
  reg reg1{
    errextbus = true;
    regwidth = 32;
    field {
      hw = rw;
      sw = rw;
    } fld;
  };
 external reg1 reg1 @0x0;
};
```

Memory Component

A *memory* is an array of storage consisting of a number of entries of a given bit width. The physical memory implementation is technology dependent and memories shall be **external**.

Definitive memory definition

external *mem_name* [**#(***parameter_instance* [**,** *parameter_instance*]***)**] *mem_instance_element* [**,** *mem_instance_element*]* **;**

Anonymous memory definition

mem {[*mem_body*]**} external** *mem_instance_element* [**,** *mem_instance_element*]* **;**

RDL

```
mem fixed_mem #(longint unsigned
word_size = 32, longint unsigned
memory_size = word_size * 4096) {
  mementries = memory_size/word_size ;
  memwidth = word_size ;
} ;
```

Properties	Description	Dynamic
mementries	The number of memory entries	No
memwidth	The memory entry bit width	No
sw	Programmer's ability to read/write a memory	Yes

Register File Components

- A *register file* is as a logical grouping of one or more register and register file instances.
- The only difference between the register file component (**regfile**) and the **addrmap** component is an **addrmap** defines an RTL implementation boundary where the **regfile** does not.

Definitive Register File Definition

[**external** | **internal**] *regfile_name* [**#(***parameter_instance* [**,** *parameter_instance*]***)**]
regfile_instance_element [**,** *regfile_instance_element*]* **;**

Anonymous Register File Definition

regfile {[*regfile_body*]**}**
[**external** | **internal**] *regfile_instance_element* [**,** *regfile_instance_element*]* **;**

Properties	Description	Dynamic
alignment	Specifies alignment of all instantiated components in the associated register file	No
sharedextbus	Forces all external registers to share a common bus	No
errextbus	For an external regfile, the associated regfile has an error input	No

```
regfile fifo_rfile {
   reg {field {} a;} a;
   reg {field {} a;} b;
};
regfile top_regfile {
  external fifo_rfile fifo_a;
  external fifo_rfile fifo_b[64];
  sharedextbus;
};

addrmap top{
   top_regfile top_regfile;
};
```

Addrmap

- An address component map (**addrmap**) contains registers, register files, memories, and/or other address maps and assigns a virtual address or final addresses.
- Specifies RTL module boundary

Definitive Definition

component new_component_name [**#(***parameter_definition* [**,** *parameter_definition*]***)**]
{[*component_body*]**}** [*instance_element* [**,** *instance_element*]***];**

Anonymous Definition

component **{**[*component_body*]**}** *instance_element* [**,** *instance_element*]***;**

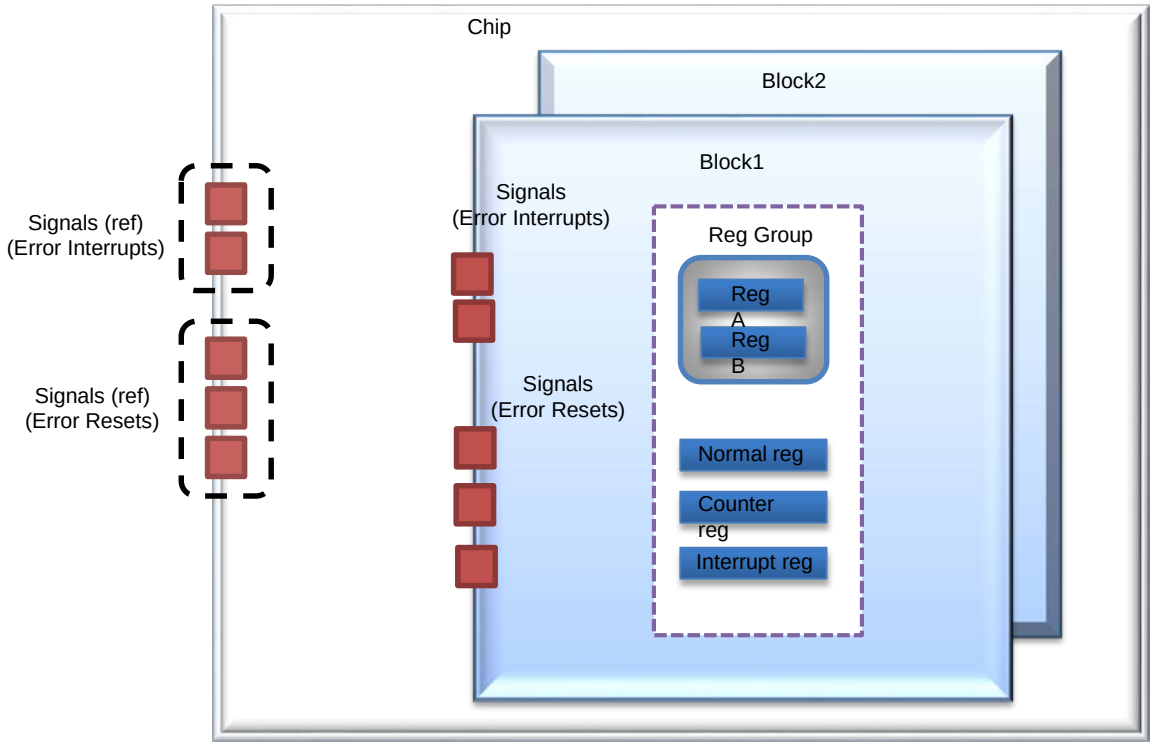
Properties	Description	Dynamic
alignment	Alignment of all instantiated components in the address map	No
sharedextbus	Forces all external registers to share a common bus	No
errextbus	The associated addrmap instance has an error input	No
littleendian	Uses little-endian architecture in the address map	Yes
addressing	Controls how addresses are computed in an address map	No
rsvdset	The read value of all fields not explicitly defined is set to 1 if rsvdset is True; otherwise, it is set to 0	No
rsvdsetx	The read value of all fields not explicitly defined is unknown if rsvd-setX is True	No
msb0	Specifies register bit-fields in an address map are defined as 0:N versus N:0	No
lsb0	Specifies register bit-fields in an address map are defined as N:0 versus N:0	No

Addrmap - Contd..

```
addrmap top{
  errextbus;
  rsvdset;
  reg reg1 {
    field {
    } fld1[31:20];
    field {
    } fld2[7:5];
  };
  reg reg2 {
    field {
    } fld1[32];
  };
reg1 reg1 @0x0;
external reg2 reg2 @0x4;
};
```

Signals

- "Signals" creates ports, at the block or chip level, and connect certain internal design signals to the external world.
- User can choose what gets connected to these signals and where these signals are used in the generated RTL using properties



Keyword	Description	Dynamic
signalwidth	Width of the signal	No
sync	Synchronous to the clock of the component	Yes
async	Asynchronous to the clock of the component	Yes
cpuif_reset	Default signal to use for resetting the software interface logic. This parameter only controls the CPU interface of a generated slave	Yes
field_reset	Default signal to use for resetting field implementations	Yes
active low	Signal is active low (state of 0 means ON)	Yes
active high	Signal is active high (state of 1 means ON)	Yes
resetsignal	Reference to the signal used to reset the field	Yes

Signals - Contd..

```
addrmap top {
signal{activelow;async;field_reset;} pci_soft_reset;
signal{async;activelow;cpuif_reset;} pci_hard_reset;

  reg PCIE_REG_BIST {
    regwidth = 8;
      field {
        hw = rw;
        sw = r;
        fieldwidth = 4;
      } cplCode [3:0];
      field {
        hw = rw;
        sw = rw;
        fieldwidth = 1;
        resetsignal = pci_hard_reset;
      } capable [7:7]=0;
  };
 PCIE_REG_BIST PCIE_REG_BIST @0x0;
};
```

Instance address allocation

Instance Alignment

Address allocation operators

a) **@** *expression :* Specifies the address for the component instance.

b) **+=** *expression :* Specifies the address stride when instantiating an array of components (controls the spacing of the components).

c) **%=** *expression :* Specifies the alignment of the next address when instantiating a component (controls the alignment of the components).

Addressing Modes

Addressing Modes:

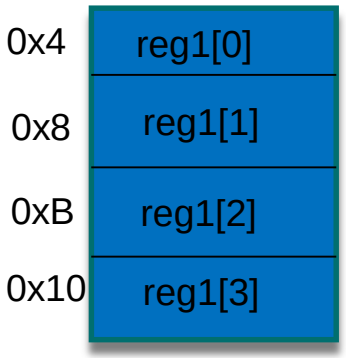
a) Compact : Specifies the components are packed tightly together while still being aligned to the accesswidth parameter

b) Regalign : Specifies the components are packed so each component's start address is a multiple of its size

c) fullalign : The assigning of addresses is similar regalign, except for arrays.

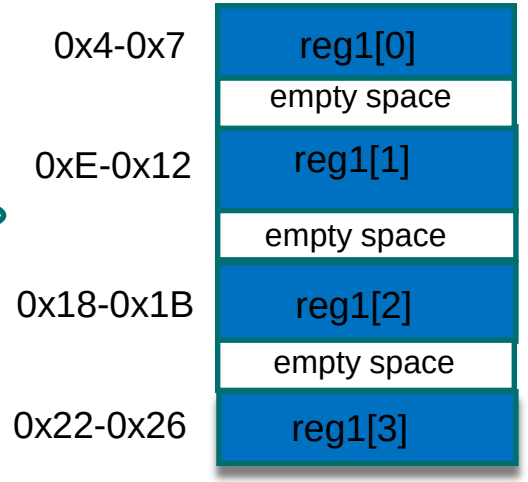
Offset (@)

```
addrmap top {
  reg r1 {
   field { } f1[3:0];
  };
  r1 reg1[4] @0x4;
};
```



Stride (+=)

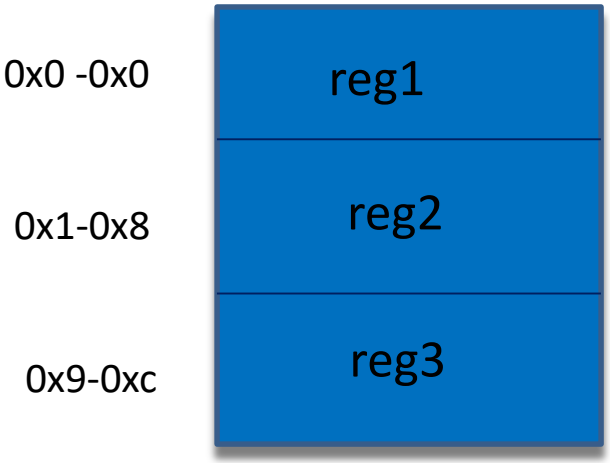
```
addrmap top {
  reg r1 {
   field { } f1[3:0];
  };
  r1 reg1[4] @0x4 += 10 ;
};
```



Compact

- It specifies the components are packed tightly together.

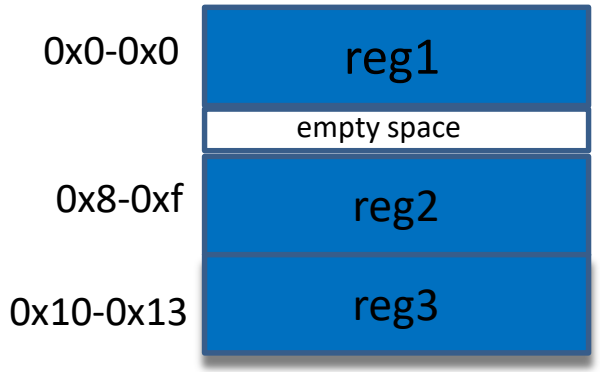
```
addrmap b1{
  addressing = compact;
  reg {
    regwidth = 8;
    field {
    } fld[7:0];
  } reg1;
  reg {
    regwidth=64;
    field {
    } fld1[63:0];
  } reg2;
  reg {
    regwidth = 32;
    field {
    } fld2[31:0];
  } reg3[20];
};
```



Regalign

- It specifies the components are packed so each component's start address is a multiple of its size

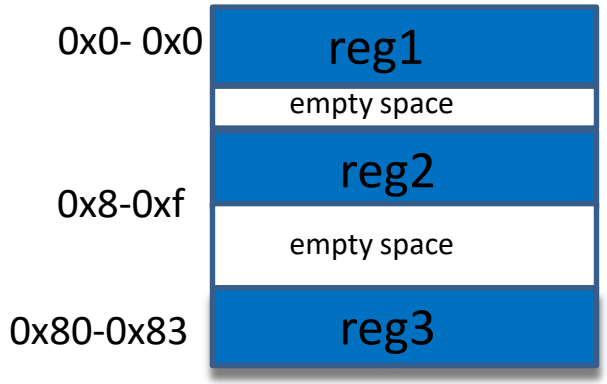
```
addrmap b1{
  addressing = regalign;
  reg {
    regwidth = 8;
    field {
    } fld[7:0];
  } reg1;
  reg {
    regwidth=64;
    field {
    } fld1[63:0];
  } reg2;
  reg {
    regwidth = 32;
    field {
    } fld2[31:0];
  } reg3[20];
};
```



Fullalign

- The assigning of addresses is similar regalign, except for arrays.
- The alignment value for the first element in an array is the size in bytes of the whole array (i.e., the size of an array element multiplied by the number of elements), rounded up to nearest power of two.

```
addrmap b1{
  addressing = fullalign;
  reg {
    regwidth = 8;
    field {
    } fld[7:0];
  } reg1;
  reg {
    regwidth=64;
    field {
    } fld1[63:0];
  } reg2;
  reg {
    regwidth = 32;
    field {
    } fld2[31:0];
  } reg3[20];
};
```



Enumerations

- It encloses a set of constant named integral values into the enumeration's scope

Syntax: An *enum component definition* appears as follows.

enum *enum_name* **{** *encoding***;** [*encoding***;**]* **};**

Enumerator references shall be prefixed with their enumerated type name and two colons (::), e.g., MyEnumeration::MyValue.

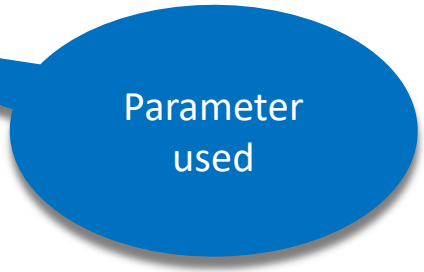
Keyword	Description	Dynamic
enum	It encloses a set of constant named integral values into the enumeration's scope	no
encode	Binds an enumeration to a field.	Yes

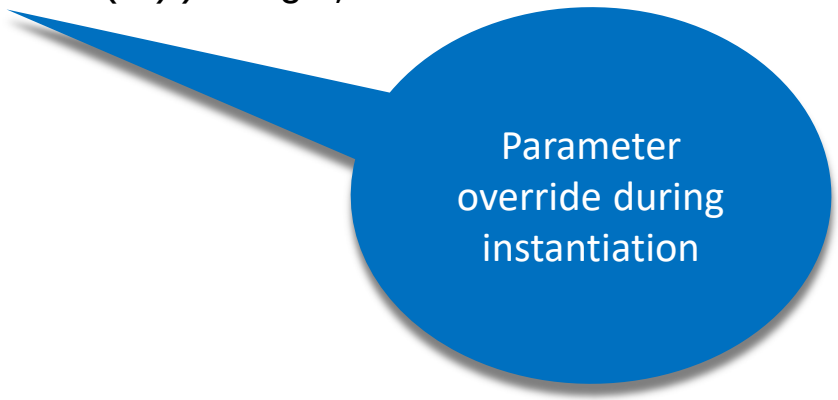
```
enum Enum1 {
 VAL1 = 3'h0 ;
 VAL2 = 3'h1 ;
} ;
enum Enum2 {
 VAL11 = 3'h0 ;
 VAL22 = 3'h1 ;
 VAL33 = 3'h2 ;
} ;
property MyUDP { component = addrmap ; type = Enum1;};
addrmap top {
  reg some_reg { field {} a[3] ; } ;
    addrmap {
      MyUDP = Enum1::VAL1 ; // Allowed
      some_reg regA ;
      regA.a -> reset = Enum1::VAL2 + Enum2::VAL33;
    } submap1 ;
    addrmap {
      reg {
       field {
        hwclr=longint'(Enum1::VAL1) ==
longint'(Enum2::VAL11);
       } b;
      } other_shared_reg ;
    } submap2 ;
};
```

Defining component parameters

- All definitive component types, except enumerations and constraints, may be parameterized using Verilog-style parameters.

```
reg myReg #(longint unsigned SIZE =32){
  regwidth = SIZE;
  field {
  } data[SIZE – 1];
};
addrmap myAmap {
  myReg reg32;
  myReg reg32_arr[8];
  myReg #(.SIZE(16)) reg16;
  myReg #(.SIZE(8)) reg8;
};
```





Struct

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

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

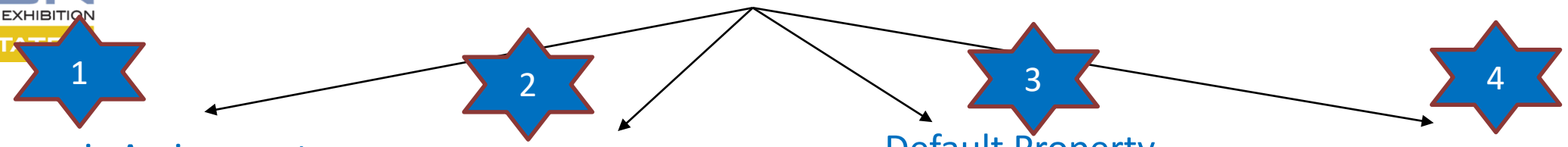
Deriving structures

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;
};
```

Property Assignment



Dynamic Assignment

- When a property is assigned after the component is instantiated, the assignment itself is referred to as a *dynamic assignment*.

Syntax:

instance_name ->
property_name [= value];

Property Assignment

- A specific property shall only be set once per scope.

Syntax:

property_name[=expression];

Default Property Assignment

- A specific property **default** value shall only be set once per scope.

Syntax

default property_name [= value];

SystemRDL Default Value for Property type

- Property takes its default value

```
reg {
 default name =”def
 name”;
 field f_type {
  name = “other name”;
 };
 f_type f1;
 f1->name = “Dynamic
Assignment”;
} some_reg;
```

```
reg {
 default name =”def
 name”;
 field f_type {
  name = “other name”;
 };
 f_type f1;
 f1->name = “Dynamic
Assignment”;
} some_reg;
```

```
reg {
 default name =”def
 name”;
 field f_type {
  name = “other name”;
 };
 f_type f1;
 f1->name = “Dynamic
Assignment”;
} some_reg;
```

```
reg {
 default name =”def
 name”;
 field f_type {
  name = “other name”;
  we;
 };
 f_type f1;
 f1->name = “Dynamic
Assignment”;
} some_reg;
```

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;
};
```

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;
};
```

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" ;
};
```

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;
};
```

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)
.       .       .
.       .
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);
uvm_resource_db#(bit)::set({"REG::", this.get_full_name()},
"NO_REG_TESTS", 1, this);
```

dontcompare

donttest

SystemRDL with Embedded Perl

- 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];
};
```

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;
};
```

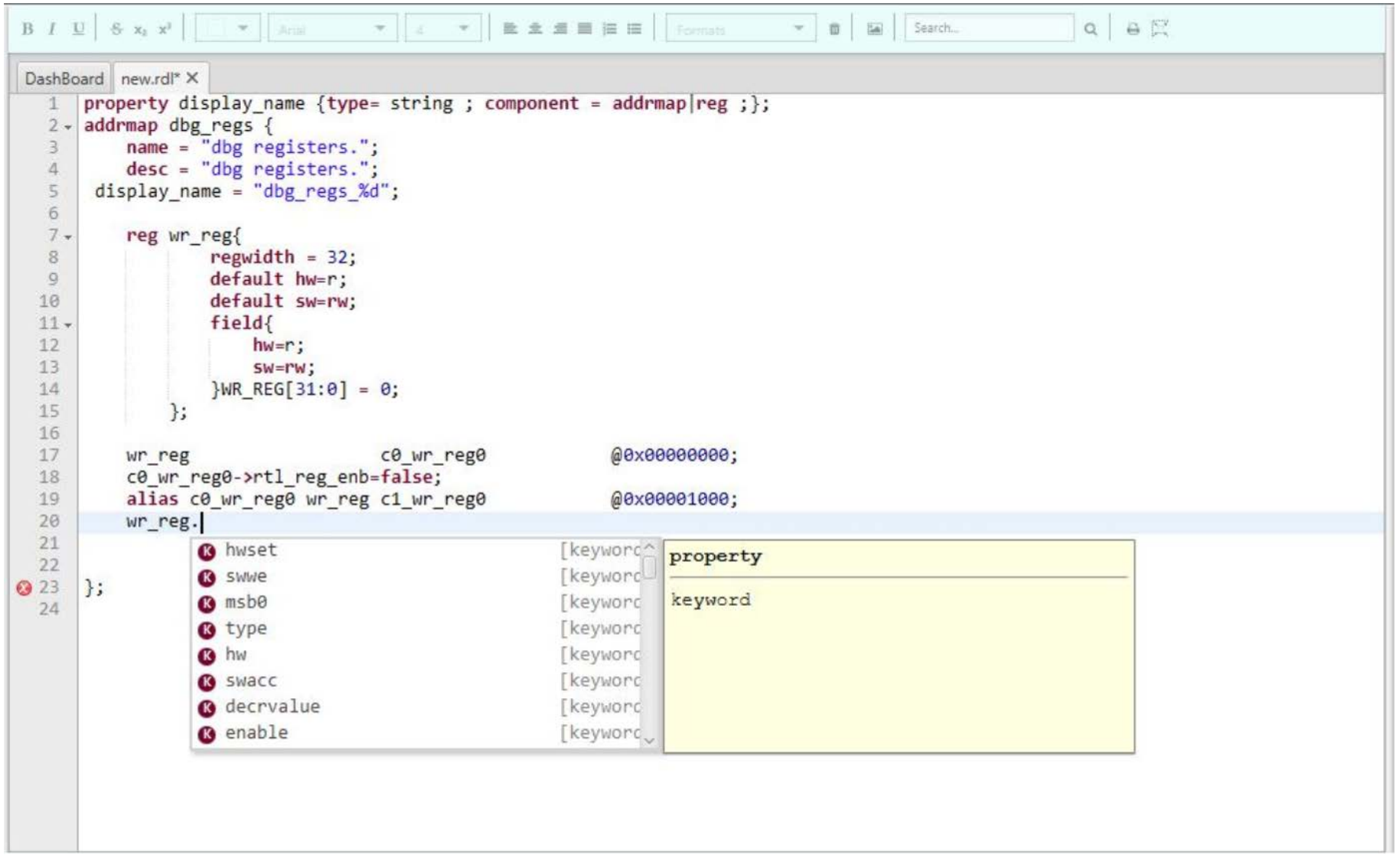
IDSBatch Output:

```
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
```

SystemRDL Editor

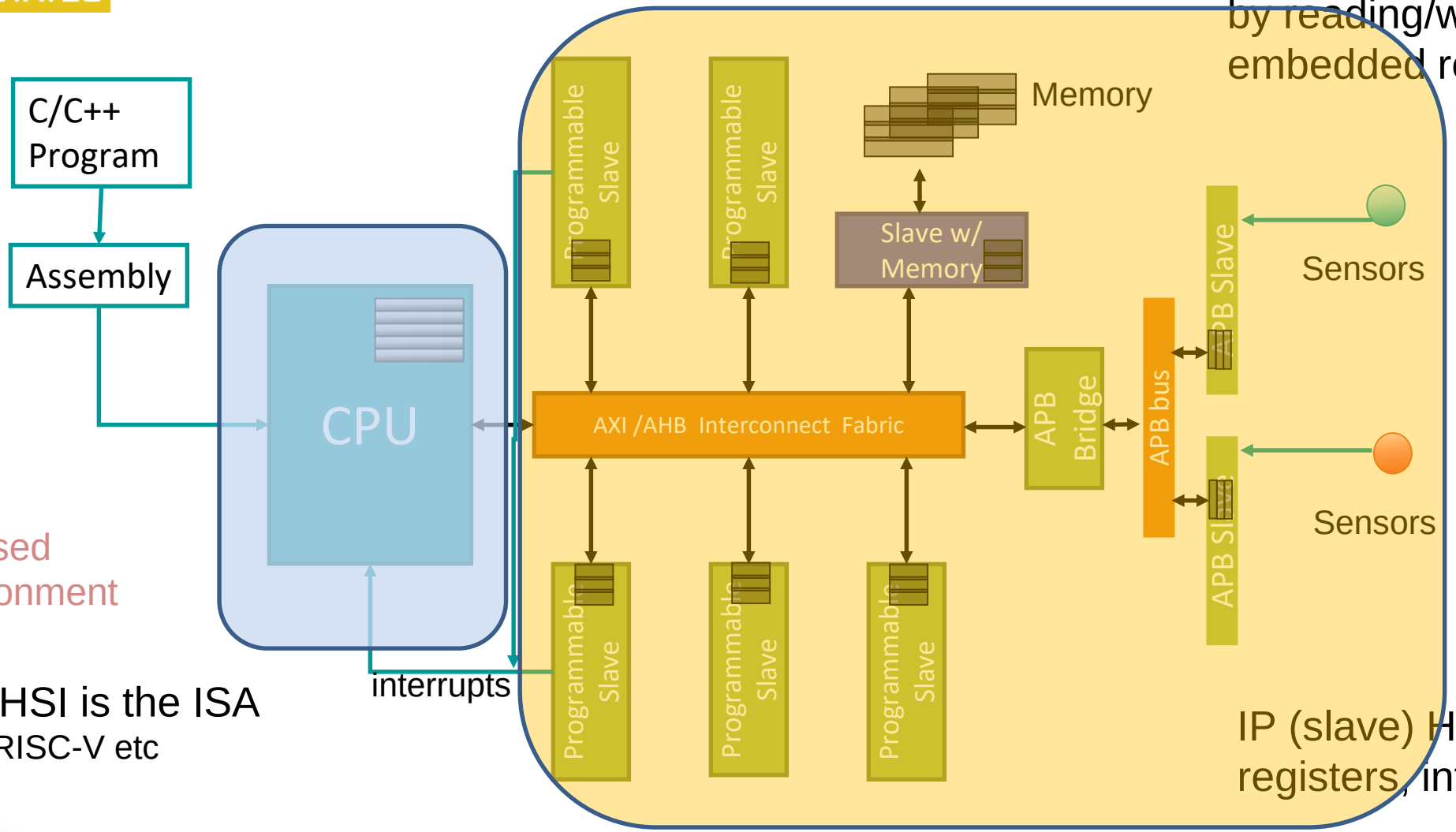
- SystemRDL editor is available as a part of IDS-NG
- User can write input SystemRDL file in the editor
- Keywords are highlighted which makes effective code visibility
- Auto completion of components is also possible (e.g. bracket, semicolon completion)
- The tool indicates syntax error for every line, 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 – Contd..



SoC HW/SW Interface Layer

The slaves are programmed by reading/writing to the embedded register



SV-UVM environment

C based environment

CPU HSI is the ISA
ARM, RISC-V etc

IP (slave) HSI are the registers, interrupts

Example Sequence with HSI

- As an example, the code below is a SV task that is manually coded by the user. It shows that HSI is a critical part of a sequence to achieve a certain behavior in the target device.

```
task xmit( int noOfTxTrans);

        for ( int count = 0 ; count < noOfTxTrans;count++ )
        begin

            if (1 && count == LineRate && rdValue == ClockFreq)
                begin
                    lvar = InitialWriteData + count;

                    rm.TXDATA.write(status, lvar, .parent(this));

                    rm.CONTROL.TXEN.write(status, uartControl[1], .parent(this));
                end
                while  (rdValue == 0)
                begin

                    rm.STATUS.TXDONE.read(status, STATUS_TXDONE , .parent(this));

                    rdValue=STATUS_TXDONE;
                end
            end

endtask
```

Writing a Register

Writing a Field

Reading a Field

Introduction to Sequences

- Sequences are built on registers, memories, pins
- Sequences contain
  - Register / Field Writes
  - Register / Field Reads
  - Pin Manipulation Commands
  - Wait / Function calls, sub sequence calls
- Information about Registers/Memories can be in any format
  - IP-XACT
  - SystemRDL
  - Word / Excel
  - Text files

The Problem:
How to write sequences once, run anywhere?

- Designer Creates HW design with a certain sequence
- Verification engineer reads from a spec or from designer's mind and creates SV/UVM sequences
- Firmware engineer repeats step 2 but this time in his own environment typically C/C++
- Lap debug may have a C based environment or even TCL/Python based environment
- Repeat the process for validation

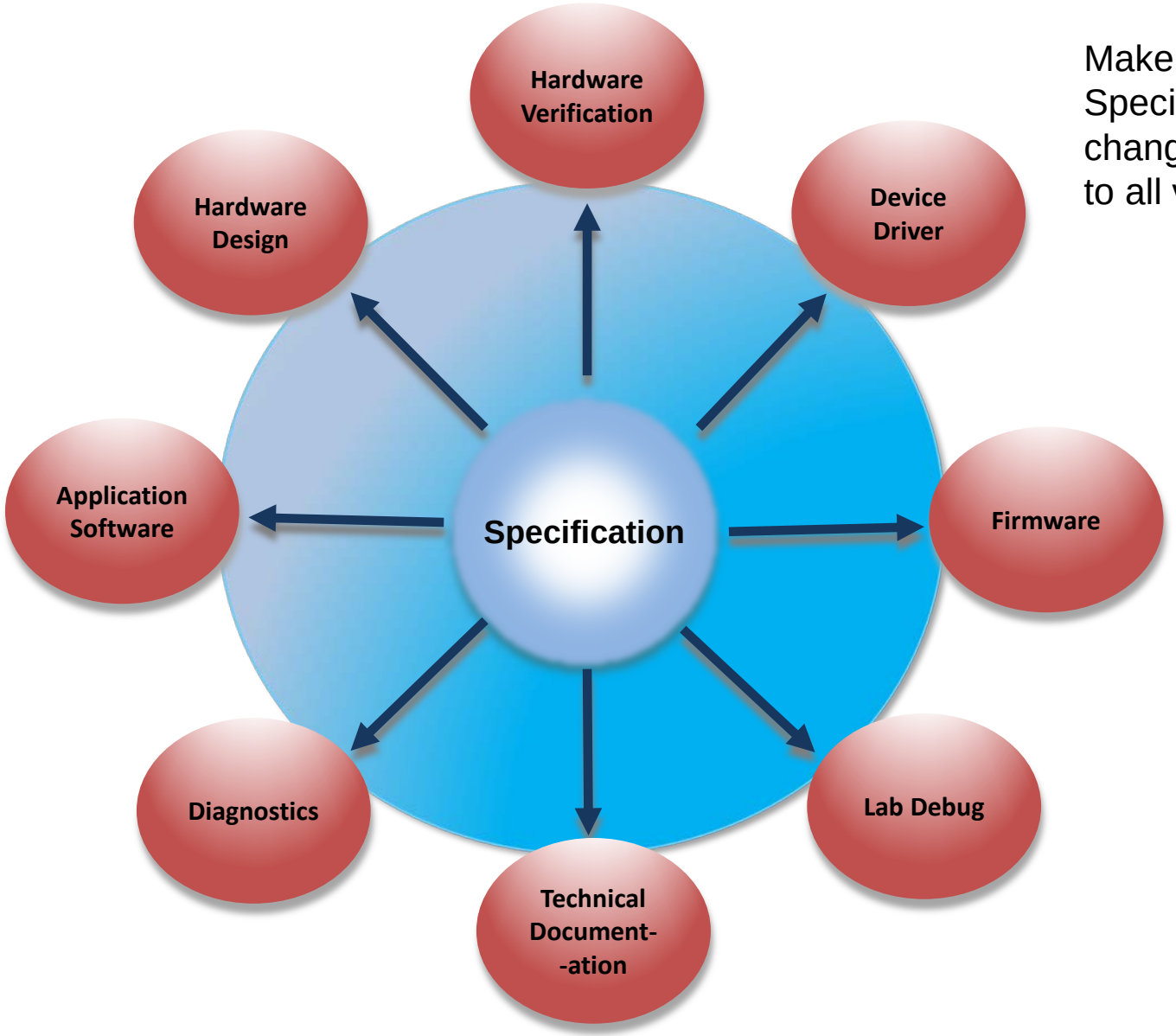
Why repeat the same algorithm over and over again in the various stages of the development?

Proposed Solution

Create a Golden Spec for Implementation-Level Sequences and Auto-Generate the Code

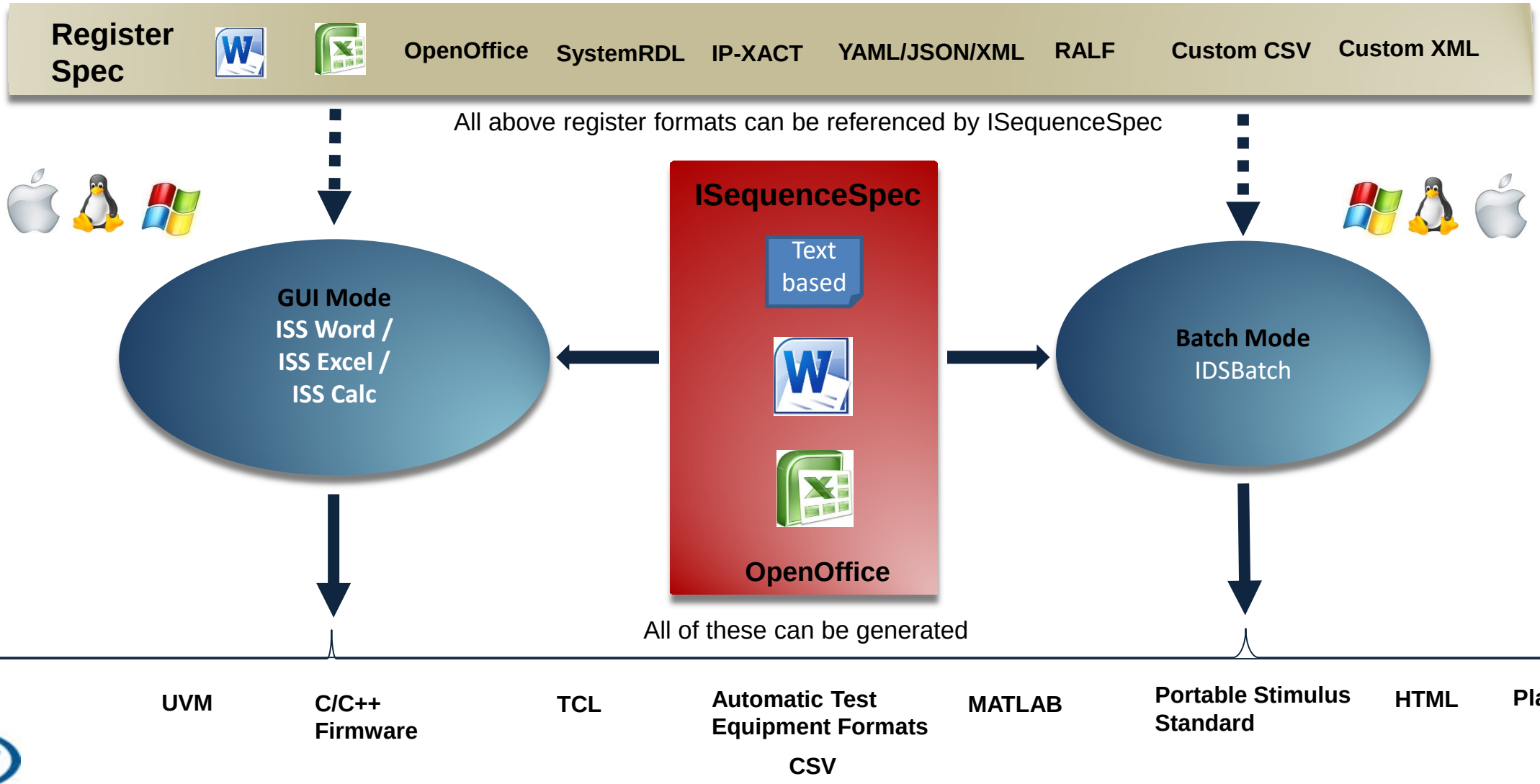
- Capturing the golden specification for sequences will need the following capabilities:
  - Control flow
  - High level of abstraction devoid of implementation detail
  - Access to hierarchical register data for SoC, Subsystem and IPs
  - Access to pins, signals and interfaces
  - High level execution of arbitrary transactions
  - Deal with timing differently based on the target
  - Hierarchy of sequence and base address of the DUT

Specification Driven Development



Make changes to the Specification and have the change automatically permeate to all views

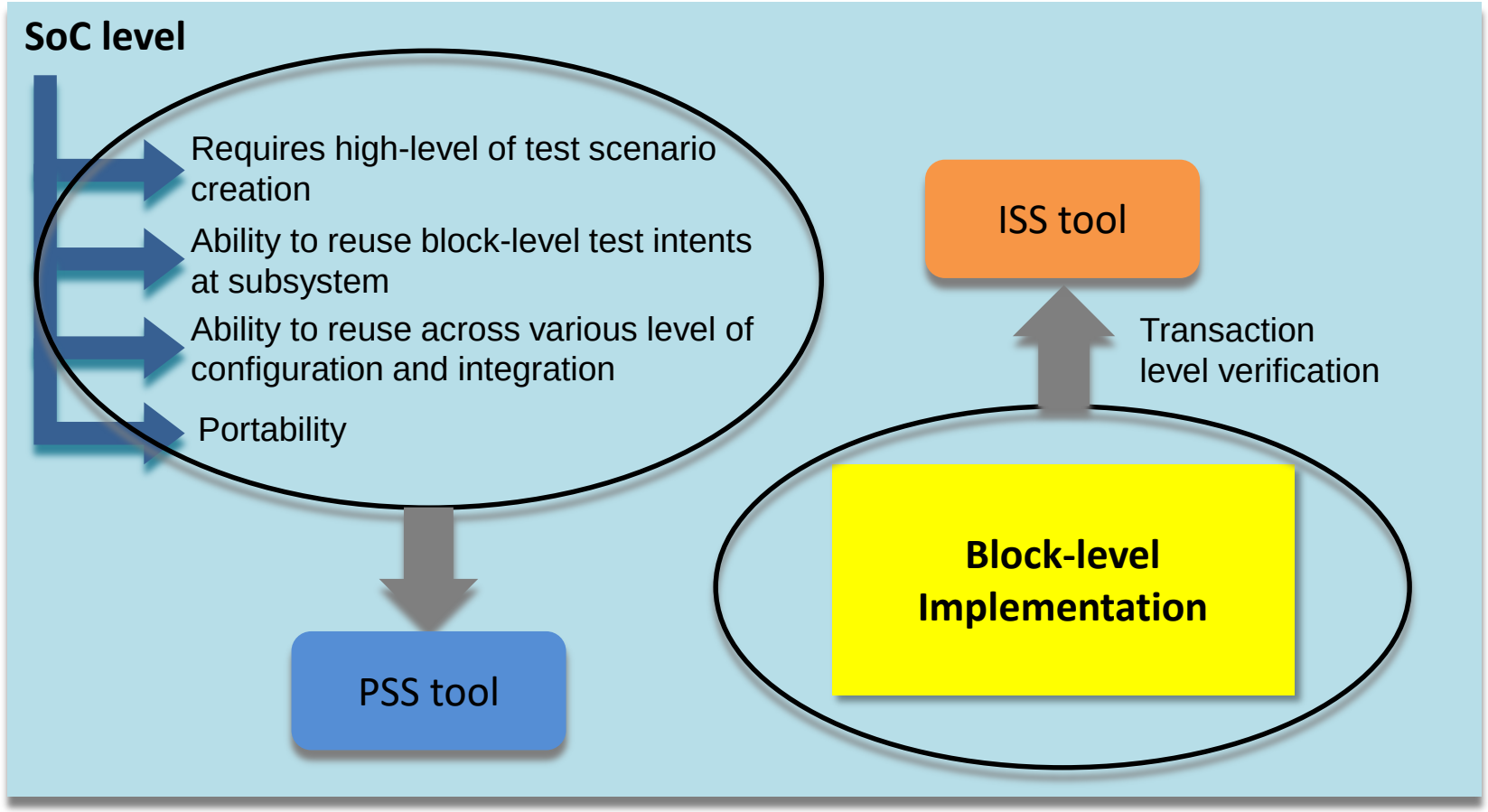
ISequenceSpec™ Suite



Portable Stimulus Standard

- PSS helps automate the testing process, thereby reducing the time to generate complex use-case scenarios
- It can generate tests, 10x faster than hand coding
- Portability from IP to sub-system to SoC level, including hardware-aware software can be achieved

Portable Stimulus Standard – Contd..



Portable Stimulus Standard – Contd..

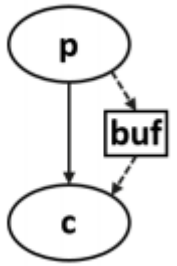
- PSS 1.0 Standard was released in June 2018

> **1.1 Purpose**
>
> The Portable Test and Stimulus Standard defines a specification for creating a single representation of stimulus and test scenarios, usable by a variety of users across different levels of integration under different configurations, enabling the generation of different implementations of a scenario that run on a variety of execution platforms, including, but not necessarily limited to, simulation, emulation, FPGA prototyping, and post-Silicon. With this standard, users can specify a set of behaviors once, from which multiple implementations may be derived.

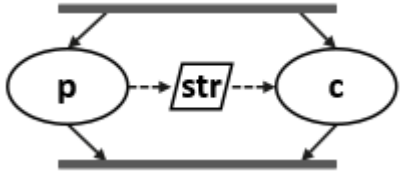
- Powerful concepts of PSS: Abstraction and Reuse
- PSS is useful for high-level test scenario creation
  - Modeling Data flow
  - Modeling Behavior
  - Constraints, Randomization, Coverage
- Actions are a key abstraction unit – can model the scenarios and include exec blocks
- The implementation-level tests are handled by “exec blocks”

PSS Data Flow Object Types

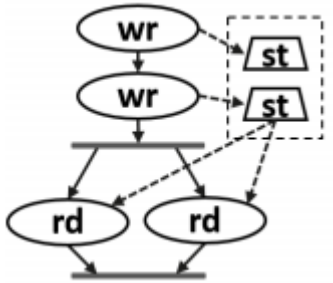
- **Buffers**: A buffer represents persistent data that can be written (output by a producing action) and may be read (input) by any number of consuming actions.
- **Streams**: The stream flow object type represents transient data shared between actions. The semantics of the stream flow object requires that the producing and consuming actions execute in parallel (i.e., both activities shall begin execution when the same preceding action(s) complete.
- **States**: The state flow object represents the state of some element in the DUT or test environment at a given time. Multiple actions may read or write the state object, but only one write action may execute at a time.
- **Data Object Pools**: Data flow objects are grouped into pools, which can be used to limit the set of actions that can communicate using objects of a given type. For buffer and stream types, the pool will contain the number of objects of the given type needed to support the communication between actions sharing the pool. For state objects, the pool will only contain a single object of the state type at any given time.



a. Buffers



b. Streams



c. States

PSS Language Constructs

- **Component**: A structural entity, defined per type and instantiated under other components.

```
component component_identifier {
        <component_body_item>
};
```

overrides_declaration
action_declaration
object_bind_stmt
exec_block
package_body_item
component_field_declaration
attr_group

- **Action**: An element of behavior.

```
action action_identifier {
        <action_body_item>
};
```

overrides_declaration
activity_declaration
constraint_declaration
exec_block_stmt
symbol_declaration
action_field_declaration
covergroup_declaration

```
action write {
        output data_buf data;
        rand int size;
        ...ls
action write_compound {
        write w1, w2;
        activity{
                w1;
                w2;
        }
};
```

  - **Atomic action**: An action that corresponds directly to operations of the underlying system under test (SUT) and test environment.
  - **Compound action**: An action which is defined in terms of one or more sub-actions.

- **Activity**: An abstract, partial specification of a scenario that is used in a compound action to determine the high-level intent and leaves all other details open.

PSS Language Constructs – Contd..

- **Exec block**: Specifies the mapping of PSS scenario entities to its non-PSS implementation.

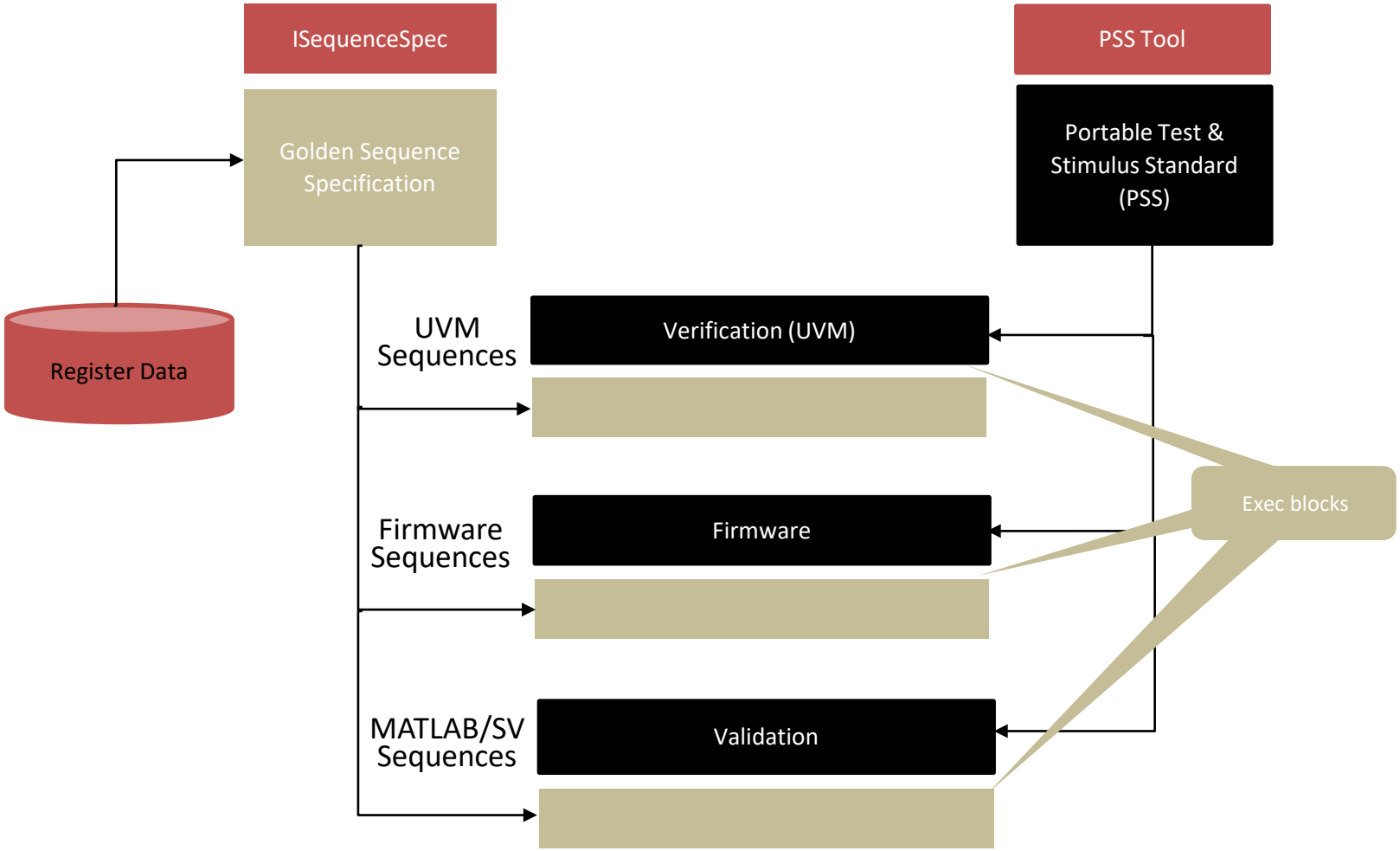
```
exec exec_kind_identifier {
        <exec_body_stmt>
};
```

exec_kind_identifier:

- pre_solve
- post_solve
- body
- header
- declaration
- run_start
- run_end
- init

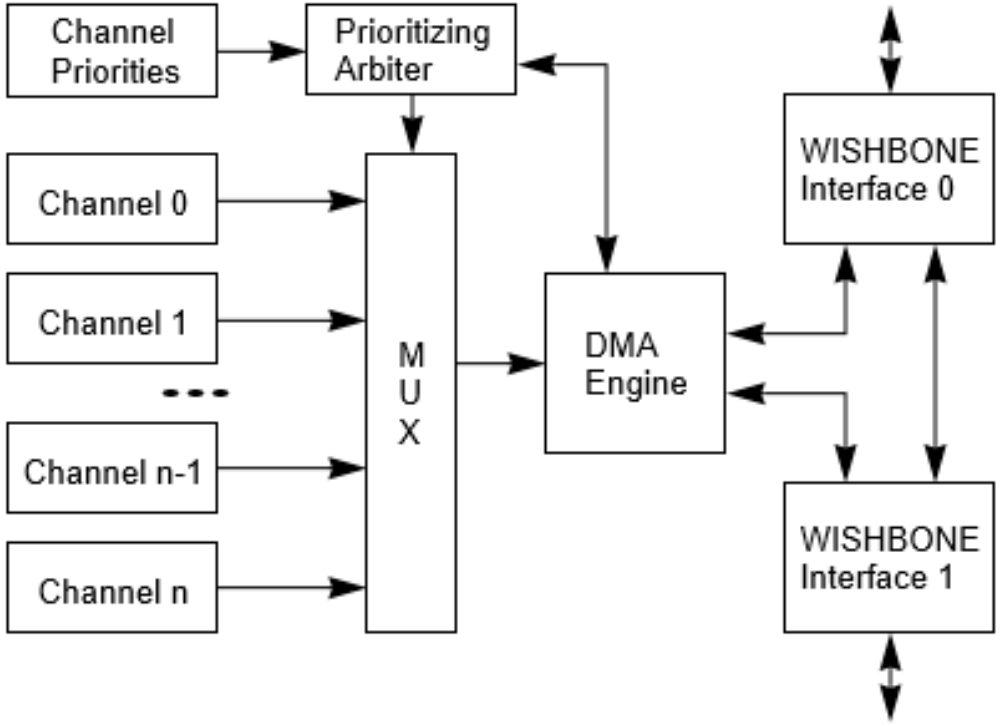
ISequenceSpec + PSS Proposed Tool Flow

- Capture sequences in pseudo-code in the golden spec (spreadsheet or text)
- Generate sequences in multiple formats (C, System Verilog, UVM)
- PSS tool user creates the test scenarios and calls the exec blocks generated by ISS
- PSS tool user synthesizes the tests/scenarios and generates the required files for the target platform



WISHBONE DMA

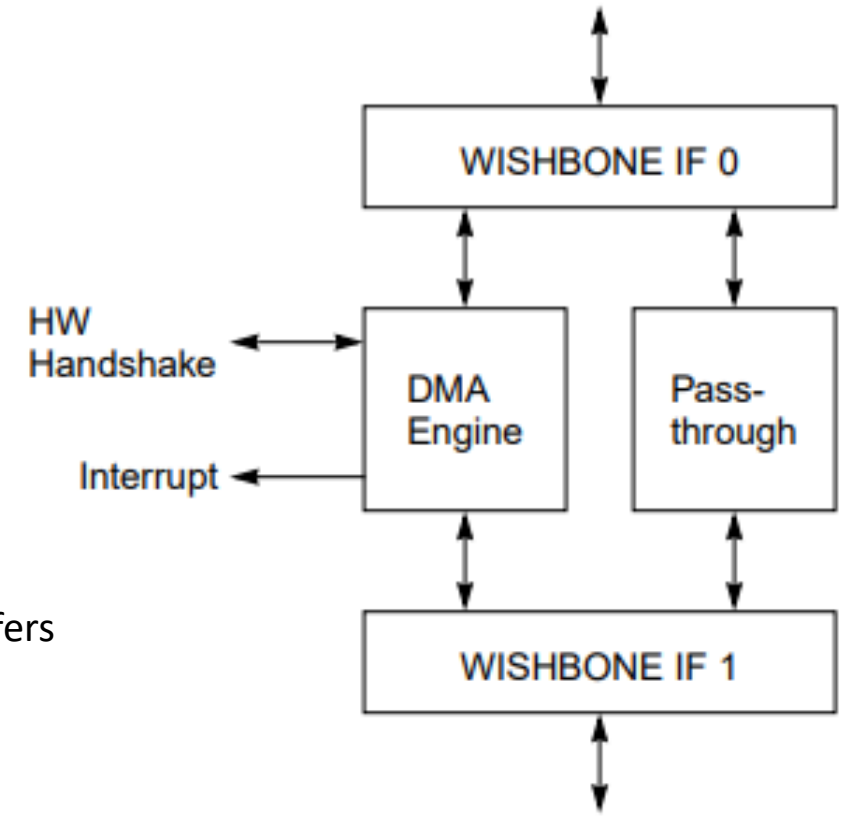
- WISHBONE DMA has been contributed by OpenCores. This core provides DMA transfers between two WISHBONE interfaces. Transfers can also be performed on the same WISHBONE interface.
- Following block diagram depicts the DMA Engine and its functional blocks.



Source: https://opencores.org/websvn/filedetails?repname=wb_dma&path=%2Fwb_dma%2Ftrunk%2Fdoc%2Fdma_doc.pdf

WISHBONE DMA – Contd..

- It consists of 3 building blocks:
  - 2 WISHBONE interfaces
  - DMA Engine
  - Pass-through logic
- **WISHBONE interface :**
  - DMA Bridge/Core has two master and slave capable WISHBONE interfaces.
  - Both interfaces are WISHBONE SoC bus specification Rev. B compliant.
  - This implementation implements a 32-bit bus width.
- **DMA Engine:**
  - The DMA engine is a up to 31 channel DMA engine that supports transfers between the two interfaces as well as transfers on the same interface.
  - Each channel can be programmed to have a different priority.
- **Pass-through logic:**
  - This block performs the bridging operation between the two WISHBONE interfaces.
  - It includes a two entry deep write buffer in each direction. The write buffer can be disabled if desired.



Source: https://opencores.org/websvn/filedetails?repname=wb_dma&path=%2Fwb_dma%2Ftrunk%2Fdoc%2Fdma_doc.pdf

WISHBONE DMA – Contd..

- This implementation is designed to work with two WISHBONE interfaces running at the same clock.

- The WISBONE specification and additional information about WISHBONE SoC can be found at: http://www.opencores.org/wishbone/

- The Main features of the DMA/Bridge are:
  - Up to 31 DMA Channels
  - 2, 4 or 8 priority levels
  - Linked List Descriptors Support
  - Circular Buffer Support
  - FIFO buffer support
  - Hardware handshake support

Source: https://opencores.org/websvn/filedetails?repname=wb_dma&path=%2Fwb_dma%2Ftrunk%2Fdoc%2Fdma_doc.pdf

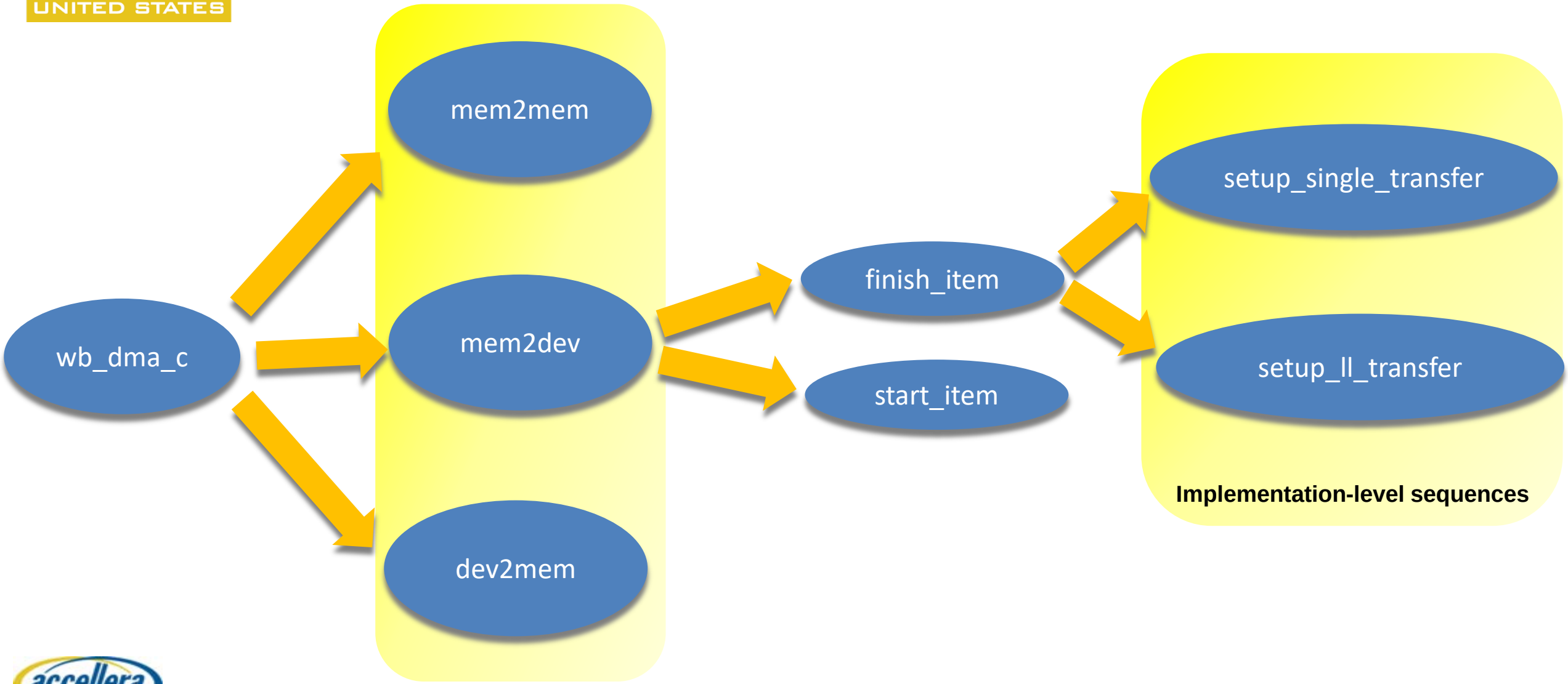
DMA Register Map

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

  reg CSR {
     regwidth = 32;
    field {
      hw = rw;
      sw = rw;
    } PAUSE[0:0] = 1'h0;
  };

  reg int_msk_a {
     regwidth = 32;
    field {
      hw = r;
      sw = rw;
    } MASK[30:0] = 31'h0;
  };

  reg int_msk_b {
     regwidth = 32;
    field {
      hw = r;
      sw = rw;
    } MASK[30:0] = 31'h0;
  };

  .          .          .
```

```
addrmap ch {
  name  = "ch Address Map";
  reg csr {
     regwidth = 32;
    field {
      hw = r;
      sw = r;
    } I_CHK_DONE[22:22] = 1'h0;
    field {
      hw = r;
      sw = r;
    } I_DONE[21:21] = 1'h0;
    .      .        .
  };
  reg SZ {
     regwidth = 32;
    field {
      hw = rw;
      sw = rw;
    } CHK_SZ[24:16] = 9'h0;
    field {
      hw = rw;
      sw = rw;
    } TOT_SZ[11:0] = 12'h0;
  };
   .      .        .
  };
  csr csr @0x000;
  SZ SZ @0x004;
  A0 A0 @0x008;
  AM0 AM0 @0x00C;
  A1 A1 @0x010;
  AM1 AM1 @0x014;
  DESC DESC @0x018;
};
```

```
  CSR CSR @0x000;
  int_msk_a int_msk_a @0x004;
  int_msk_b int_msk_b @0x008;
  int_src_a int_src_a @0x00C;
  int_src_b int_src_b @0x010;
  ch  ch[32] @0x014;
};
```

DMA Verification Intent



Descriptors in DMA

Descriptors for DMA

```
class wb_dma_descriptor extends uvm_sequence_item;
    `uvm_object_utils(wb_dma_descriptor)
    rand bit[5:0]               channel;
    rand bit                    mode;
    rand bit                    inc_src;
    rand bit                    inc_dst;
    rand bit                    src_sel;
    rand bit                    dst_sel;
    rand bit[11:0]              tot_sz;
    rand bit[2:0]               trn_sz;
    rand bit[8:0]               chk_sz;
    rand bit[31:0]              src_addr;
    rand bit[31:0]              dst_addr;
    bit                         rand_addr;
    constraint trn_sz_c {
        trn_sz inside {1, 2, 4};
    }
    constraint addr_incr_c {
        inc_src || inc_dst;
    }
    constraint channel_c {
        channel inside {[0:7]};
    }
    constraint tot_sz_c {
        tot_sz > 0;
    }
    constraint chk_sz_c {
        chk_sz > 0;
    }
    constraint rand_addr_c {
        (rand_addr == 0) -> src_addr == 0;
        (rand_addr == 0) -> dst_addr == 0;
    }
endclass
```

PSS code

PSS component containing actions

```
component wb_dma_c {
  import pvm_types_pkg::*;

  action wb_dma_a {
    // The channel this transfer runs on
    rand bit[3]              channel;

    // Total transfers to perform
    rand bit[16]             tot_sz;

    // Bytes to transfer at a time (1, 2, 4)
    rand bit[4] in [1,2,4]   trn_sz;
  }

  action mem2mem_a : wb_dma_a {
    input data_mem_b           dat_i;
    output data_mem_b          dat_o;

    constraint {
        dat_i.sz == dat_o.sz;
    }
  }

  action mem2dev_a : wb_dma_a {
    input data_mem_b           dat_i;
    output data_ref_mem_s      dev_dat_o;

    constraint {
        dat_i.sz == dev_dat_o.sz;
    }
  }

  action dev2mem_a : wb_dma_a {
    input data_mem_b           dat_o;
    output data_ref_mem_s      dev_dat_i;

    constraint {
        dat_o.sz == dev_dat_i.sz;
    }
  }
}
```

PSS code – Contd..

PSS action blocks containing the Exec Block

“exec” blocks generated by ISequenceSpec containing SV tasks.

```
extend action wb_dma_c::mem2mem_a {

    exec body {
        mem2mem(channel, dat_i.addr, dat_o.addr, tot_sz, trn_sz);
    }

    exec body SV = """
        mem2mem({{channel}}, {{dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}}, {{trn_sz}});
    """;
    exec body C = """
        mem2mem({{channel}}, {{dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}}, {{trn_sz}});
    """;

}

extend action wb_dma_c::dev2mem_a {
    exec body {
        dev2mem(channel, dev_dat_i.addr, dat_o.addr, tot_sz, trn_sz);
    }

    exec body SV = """
        dev2mem({{channel}}, {{dev_dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}},
        {{trn_sz}});
    """;
    exec body C = """
        dev2mem({{channel}}, {{dev_dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}},
        {{trn_sz}});
    """;
}
```

PSS code – Contd..

```
component wb_dma_par4_xfer_c {
  enum xfer_type_e {
    mem2mem,
    mem2dev,
    dev2mem
  }
  action xfer_a {
    rand bit[3]                 channel;
    rand xfer_type_e            xt;
    wb_dma_c::mem2mem_a         m2m;
    wb_dma_c::dev2mem_a         d2m;
    wb_dma_c::mem2dev_a         m2d;
    dma_helper_c::gendata_a     gd;
    dma_helper_c::checkdata_a   cd;
    dma_helper_c::dev_sink_a    dsi;
    dma_helper_c::dev_src_a     dsr;
    constraint channel_c {
        m2m.channel == channel;
        m2d.channel == channel;
        d2m.channel == channel;
    }
    activity {
        match (xt) {
            [mem2mem]: {gd ; m2m; cd; }
            [dev2mem]: {
                parallel { d2m; dsr; }
                cd;
            }
            [mem2dev]: {
                gd;
                parallel { m2d; dsi; }
            }
        }
    }
}
```

```
action scenario_top_a {
  action wb_dma_par4_xfer_c::xfer_type_e  xt_1, xt_2, xt_3, xt_4;
  action bit[3]                           ch_1, ch_2, ch_3, ch_4;
  xfer_a x1, x2, x3, x4;
  constraint channel_unique_c {
      unique {x1.channel, x2.channel, x3.channel, x4.channel};
  }
  activity {
      repeat (40) {
          xt_1; xt_2; xt_3; xt_4;
          ch_1; ch_2; ch_3; ch_4;

          parallel {
              x1;
              x2;
              x3;
              x4;
          }}}
  constraint xt_cov_c {
      xt_1 == x1.xt; ch_1 == x1.channel;
      xt_2 == x2.xt; ch_2 == x2.channel;
      xt_3 == x3.xt; ch_3 == x3.channel;
      xt_4 == x4.xt; ch_4 == x4.channel;}
  covergroup {
      xt1_cp : coverpoint xt_1; // Transfer type for x1
      xt2_cp : coverpoint xt_2;
      xt3_cp : coverpoint xt_3;
      xt4_cp : coverpoint xt_4;
      c1_cp : coverpoint ch_1; // Channel for x1
      c2_cp : coverpoint ch_2;
      c3_cp : coverpoint ch_3;
      c4_cp : coverpoint ch_4;
      xt_c_1_cross : cross xt1_cp, c1_cp;
      xt_c_2_cross : cross xt2_cp, c2_cp;
      xt_c_3_cross : cross xt3_cp, c3_cp;
      xt_c_4_cross : cross xt4_cp, c4_cp;
  } cg; }    }
```

DMA sequences

```
task mem2mem(
    bit[31:0]       channel,
    bit[31:0]       src,
    bit[31:0]       dst,
    bit[31:0]       tot_sz,
    bit[31:0]       trn_sz);
    wb_dma_descriptor       desc = wb_dma_descriptor::
    type_id::create();
    $display("--> mem2mem");
    desc.mode = 0;
    desc.inc_src = 1;
    desc.inc_dst = 1;
    desc.src_sel = 0;
    desc.dst_sel = 1;
    desc.tot_sz = tot_sz;
    desc.trn_sz = trn_sz;
    desc.chk_sz = 16;
    desc.src_addr = src;
    desc.dst_addr = dst;

    start_item(desc);
    finish_item(desc);
    $display("<-- mem2mem");
endtask
```

```
task dev2mem(
    bit[31:0]       channel,
    bit[31:0]       src,
    bit[31:0]       dst,
    bit[31:0]       tot_sz,
    bit[31:0]       trn_sz);
    wb_dma_descriptor       desc = wb_dma_descriptor::
    type_id::create();
    $display("--> dev2mem");
    desc.mode = 0;
    desc.inc_src = 0;
    desc.inc_dst = 1;
    desc.src_sel = 0;
    desc.dst_sel = 1;
    desc.tot_sz = tot_sz;
    desc.trn_sz = trn_sz;
    desc.chk_sz = 16;
    desc.src_addr = src;
    desc.dst_addr = dst;

    start_item(desc);
    finish_item(desc);
    $display("<-- dev2mem");
endtask
```

```
task mem2dev(
    bit[31:0]       channel,
    bit[31:0]       src,
    bit[31:0]       dst,
    bit[31:0]       tot_sz,
    bit[31:0]       trn_sz);
    wb_dma_descriptor       desc = wb_dma_descriptor::
    type_id::create();
    $display("--> mem2dev");
    desc.mode = 0;
    desc.inc_src = 0;
    desc.inc_dst = 1;
    desc.src_sel = 0;
    desc.dst_sel = 1;
    desc.tot_sz = tot_sz;
    desc.trn_sz = trn_sz;
    desc.chk_sz = 16;
    desc.src_addr = src;
    desc.dst_addr = dst;

    start_item(desc);
    finish_item(desc);
    $display("<-- mem2dev");
endtask
```

DMA sequences– Contd..

```
virtual task finish_item(
    input uvm_sequence_item item,
    input int set_priority=-1);
    uvm_reg_data_t value;
    uvm_status_e status;
    wb_dma_descriptor desc;
    wb_dma_ll_descriptor ll_desc;
    wb_dma_ch ch;
    bit[31:0]       addresses[$];
    if (!$cast(desc, item)) begin
        `uvm_fatal(get_name(), "Failed to cast item to
        wb_dma_descriptor");
    end
    $display("--> Finish Item: channel=%0d", desc.channel);
    m_reg_sem.get(1);
    // Setup appropriate channel
    ch = m_regs.ch[desc.channel];
    if ($cast(ll_desc, desc)) begin
        setup_ll_transfer(ll_desc, addresses);
    end else begin
        setup_single_transfer(desc, addresses);
    end
    m_reg_sem.put(1);
    $display("  SRC ADDR='h%08h", desc.src_addr);
    $display("  DST ADDR='h%08h", desc.dst_addr);
    $display("  SRC IFC=%0d", desc.src_sel);
    $display("  DST IFC=%0d", desc.dst_sel);
    // Now, wait completion
    repeat(1000) begin
        #10us;
        m_reg_sem.get(1);
        ch.CSR.read(status, value);
        m_reg_sem.put(1);

        if (value[11]) begin
            $display("== DONE  CSR='h%08h ==", value);
            if (m_done_ap != null) begin
                m_done_ap.write(desc);
            end
            break;
        end
    end
    m_reg_sem.get(1);
    ch.CSR.read(status, value);
    if (!value[11]) begin
        `uvm_fatal(get_name(), "DMA transfer failed to
        terminate");
    end
    ch.CSR.read(status, value);
    value[0] = 0;
    ch.CSR.write(status, value);
    m_reg_sem.put(1);
    foreach (addresses[i]) begin
        m_mem_mgr.free(addresses[i]);
    end
    addresses = {};
    $display("<-- Finish Item %0d", desc.channel);
endtask
```

Capture Sequences in Golden Spec

Setting configuration for each output in text based format

```
configure = {
        'output' : {
                'uvm' : {
                        'name_format' : '%s',
                        'max_nesting':'1',
                        'Timemultiplier': '100',
                        'consolidated_write':'true',
                        'mout': 'false',
                        'default data-type':{
                                        'arguments':'int',
                                        'constant':'int',
                                        'variable':'int',
                                        'returntype':'int'
                                },
                        'regmodel_template' :{
                                'read' : 'rread(status, %lhs, .parent(this))',
                                'write': 'write(status, %d, .parent(this))'
                                }
                        },
                'firmware' : {
                        'name_format' : '%s',
                        'max_nesting':'1',
                        'Timemultiplier': '100',
                        'structure':'0',
                        'consolidated_write':'true',
                        'consolidated_r':'true',
                        .       .       .       .
        }
```

Capture Sequences in Golden Spec – Contd..

```
from regMap.registermap.registermap import ISequenceSpec as iss
from regMap.block import block

block =block()
iss = iss()

class wb_dma_descriptor(structure):
    def __init__(self,type = 'bit',line_length = '100'):
        self.channel = iss.struct_elem(6,'')
        self.mode = iss.struct_elem(1,'')
        self.inc_src = iss.struct_elem(1,'')
        self.inc_dst = iss.struct_elem(1,'')
        self.src_sel = iss.struct_elem(1,'')
        self.dst_sel = iss.struct_elem(1,'')
        self.tot_sz = iss.struct_elem(12,'')
        self.trn_sz = iss.struct_elem(3,'')
        self.chk_sz = iss.struct_elem(9,'')
        self.src_addr = iss.struct_elem(32,'')
        self.dst_addr = iss.struct_elem(32,'')
        self.rand_addr = iss.struct_elem(1,'')
```

Structure class

Arguments of sequences

```
class sequences:
    def setup_single_transfer(self,ip = 'WBA_DMA.rdl',desc = '{uvm
sk=true; uvm.regmodel=wb_dma_reg_block; uvm.base_class=wb_dma_tran
_seq; uvm.handle_name_format=m_regs}'):

    desc = iss.argument(wb_dma_descriptor(),'')

    clear_val=iss.constant(0,'clear_val','')

    read_val=iss.variable(0,'read_val','')
    val=iss.variable([0,0],'val','')
    inc_dst=iss.variable(0,'inc_dst','')
    src_sel=iss.variable(0,'src_sel','')
    dst_sel=iss.variable(0,'dst_sel','')
    inc_src=iss.variable(0,'inc_src','')
    channel=iss.variable(0,'channel','')
    write_st=iss.variable('0','write_st','')
    chk_sz=iss.variable(0,'chk_sz','')
    tot_sz=iss.variable(0,'tot_sz','')
    value=iss.variable('0','value','')
```

Declaration of Constants

Declaration of Variables

Capture Sequences in Golden Spec – Contd..

Steps of sequences

```
iss.read(desc.channel,write_st,'')
iss.read(ch[write_st].CSR,value,'')
iss.write(value,value & 0xFFFFFFFE,'')
iss.write(ch[write_st].CSR, value,'')
iss.read(ch[write_st].A0,value,'')
iss.read(ch[write_st].A1,value,'')
iss.write(ch[write_st].A0,desc.src_addr,'')
iss.write(ch[write_st].A1,desc.dst_addr,'')
iss.write(ch[write_st].AM0,0xFFFFFFFC,'')
iss.write(ch[write_st].AM1,0xFFFFFFFC,'')
iss.read(ch[write_st].SZ,value,'')
iss.write(value,value & 0xFE00C000,'')
iss.write(val[0],'(desc.trn_sz == 4)?0:(desc.trn_sz==2)?0:1','')
iss.write(val[1],'(desc.trn_sz == 4)?0:(desc.trn_sz==2)?1:0','')
iss.write(val[0], val[0]<<12,'')
iss.write(val[1], val[1]<<13,'')
iss.write(chk_sz, desc.chk_sz<<16 , '')
iss.write(tot_sz, desc.tot_sz)
iss.write(read_val, tot_sz|chk_sz|val[1]|val[0],'')
iss.write(value, read_val | value,'')
iss.write(ch[write_st].SZ,value,'')
iss.read(ch[write_st].CSR,value,'')
iss.write(value, value & 0xFFFE1EE0,'')
iss.write(channel, desc.channel<<13,'')
iss.write(src_sel, desc.src_sel<<2,'')
iss.write(inc_src, desc.inc_src<<4,'')
iss.write(dst_Sel, desc.dst_sel<<1,'')
iss.write(inc_dst, desc.inc_dst<<3,'')
iss.write(read_val, channel|src_sel|dst_sel|inc_dst|0x1|0x100|0x1
0000|inc_src,'')
iss.write(value, value | read_val, '')
self.write_analysis_port(desc)
iss.write(ch[write_st].CSR, value)
```

Generated Sequences in the Target Format

```
task wb_dma_transfer_seq::setup_single_transfer (
    wb_dma_descriptor  desc
    );
    uvm_status_e status;
    wb_dma_reg_block rm = m_regs ;
    const int clear_val = 0 ;
    bit [31:0]read_val = 0 ;
    bit  [2:0]chnl = 0;
    bit val[2] = {0,0} ;
    int write_st = 0 ;
    int value = 0 ;
    int lvar;
    uvm_reg_data_t ch_A0 ;
    uvm_reg_data_t ch_A1 ;
    uvm_reg_data_t ch_SZ ;
    uvm_reg_data_t ch_CSR ;
    if (rm == null)  begin
        `uvm_error("wb_dma_reg_block_block", "No register model sp
        sequence on, you should specify regmodel by using property "u
        in the sequence")
        return;
    end
    write_st=desc.channel;
    rm.ch[write_st].CSR.read(status, ch_CSR  );
    value=ch_CSR ;
    lvar = value & 'hfffffffe;
    value=lvar;
    rm.ch[write_st].CSR.write(status, value );
    rm.ch[write_st].A0.read(status, ch_A0  );
    value=ch_A0 ;
    rm.ch[write_st].A1.read(status, ch_A1  );
    value=ch_A1 ;
    rm.ch[write_st].A0.write(status, desc.src_addr );
    rm.ch[write_st].A1.write(status, desc.dst_addr );
    rm.ch[write_st].AM0.write(status, 'hFFFFFFFC );
    rm.ch[write_st].AM1.write(status, 'hFFFFFFFC );
    rm.ch[write_st].SZ.read(status, ch_SZ  );
    value=ch_SZ ;
    value = value & 'hFE00C000 ;
    lvar = desc.trn_sz==4 ? 0 : (desc.trn_sz==2 ? 0 : 1);
    val[0]=lvar;
    lvar = desc.trn_sz==4 ? 0 : (desc.trn_sz==2 ? 1 : 0);
    val[1]=lvar;
    ead_val = {1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,desc.chk_sz,1'b0,1'b0,val[1],
    [0],desc.tot_sz};
    r = read_val | value;
    ue=lvar;
    ch[write_st].SZ.write(status, value );
    .ch[write_st].CSR.read(status, ch_CSR  );
    value=ch_CSR ;
    value = value & 'hFFFE1EE0 ;
    chnl = desc.channel ;
    read_val = {1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
    1'b0,1'b0,1'b1,chnl,1'b0,1'b0,1'b0,1'b0,1'b1,1'b0,1'b0,1'b0,desc.inc_src,desc.
    inc_dst,desc.src_sel,desc.dst_sel,1'b1};
    lvar = value | read_val;
    value=lvar;
    `uvm_info("ISS", $sformatf("== DMA Transfer %0d ==/n   SRC: 'h%08h (%0d)
    inc=%0d/n   DST: 'h%08h (%0d) inc=%0d/n   SZ:  %0d/n        ", desc.channel,
    desc.src_addr, desc.src_sel, desc.inc_src, desc.dst_addr, desc.dst_sel, desc.
    inc_dst, desc.tot_sz), UVM_LOW);
    write_analyis_port(desc);
    rm.ch[write_st].CSR.write(status, value );

endtask
```

UVM generated sequences

Portability and connection to implementation level sequences

```
extend action wb_dma_c :: mem2mem_a {
        exec body SV = """
                mem2mem({{channel}}, {{dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}}, {{trn_sz}});
""";
}
```

```
extend action wb_dma_c :: mem2mem_a {
        exec body C = """
                mem2mem({{channel}}, {{dat_i.addr}}, {{dat_o.addr}}, {{tot_sz}}, {{trn_sz}});
""";
}
```

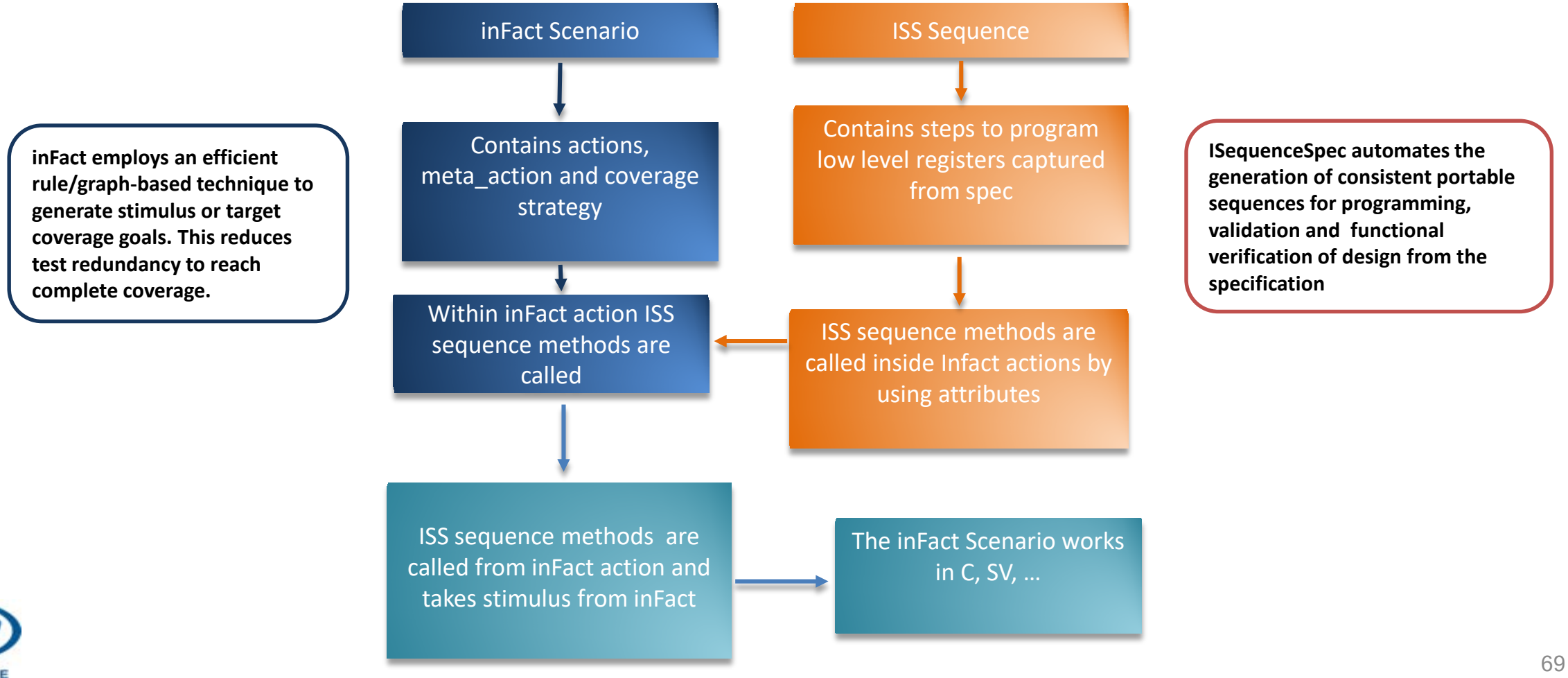
Generated PSS code

Creation of high-level test scenario

```
class wb_dma_par4_seq extends wb_dma_transfer_seq;
  // Register sequence with UVM factory
  `uvm_object_utils(wb_dma_par4_seq)
  typedef class functor;
  typedef class pcCallBack;
  typedef class covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg;
  // pss_exec class
  class pss_exec;
  // pss_exec_block class
  class pss_exec_block extends pss_exec;
  // Execute PSS exec bodies in sequence
  class pss_exec_seq extends pss_exec;
  // Execute PSS exec bodies in parallel
  class pss_exec_par extends pss_exec;
  // pss_exec_thread class
  class pss_exec_thread;
  // fields of class wb_dma_par4_seq
  TestEngine m_te;
  // Graph name
  string m_graph_name = "PSS_wb_dma_par4_xfer_c_scenario_top_a";
  // Control for coverage strategy creation message
  int m_gt_opt = 1;
  .     .     .
  // Coverage strategy covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg
  covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg
  m_covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg;
  // Controls creation of coverage strategy
  bit m_create_covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg = 1;
  // Dynamic array size(s)
  static bit registered = 0;
  int unsigned m_values[string];
  .     .     .
  // Constructor (wb_dma_par4_seq)
  function new(string name = "");
  // Function delete()
  virtual function void delete();
  // Initialization function called from constructor
  function void initialize(string name);
  function void registeraut();
  // Action tasks
  virtual task actionTask(int id);
  // Signed meta action tasks
  virtual task signedMetaActionTask(
  // Unsigned meta action tasks
  virtual task unsignedMetaActionTask(
  // Unsigned meta action import functions
  virtual function longint unsigned unsignedMetaActionImportFunction(int id);
  // Trace entry function
  virtual function void infact_trace_entry(int unsigned id);
  // Trace exit function
  virtual function void infact_trace_exit(int unsigned id);
  // Returns the cumulative coverage report of all coverages
  function string getCoverageReport();
  // Returns a true when all the individual coverage goals have been met
  function bit allCoverageGoalsHaveBeenMet();
  // Callback closure
  class functor extends FunctorBase;
  // Coverage strategy class covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg
  class covergroup_action_wb_dma_par4_xfer_c_scenario_top_a_cg;
  class pcCallBack extends PathCoverageCallback;
  function int unsigned get_val(string vname);
  // Called at the beginning of a parallel block
  task parallel_start();
  // Called at the beginning of each parallel branch
  task parallel_branch_start();
  // Called at the end of each parallel branch
  task parallel_branch_end();
  task parallel_end();
  // exec_block tasks
  task exec_body_action_wb_dma_c_mem2mem_a(string ctxt);
  task exec_body_action_wb_dma_c_mem2dev_a(string ctxt);
  task exec_body_action_wb_dma_c_dev2mem_a(string ctxt);
  virtual task body();
endclass : wb_dma_par4_seq
```

Flow Diagram

ISequenceSpec uses the stimulus created by Questa inFact test engine by calling ISS generated sequence methods inside inFact actions.



Conclusion

- Questa inFact™ is useful for SoC high-level test scenario creation; the IP level details are currently handled using “exec blocks”
- Currently, users have to manually write long sequences that deal with the registers and pin manipulation commands
- Every scenario can not be covered by manual sequences which can have low coverage.
- This limitation is removed by PSS where all scenarios are covered which finally provide the maximum coverage.
- Also ISequenceSpec™ augments PSS tools and includes:
  - Capturing sequences in a golden spec
  - Generate implementation-level SV/UVM/C sequences that enable register R/W and pin manipulation commands

Thanks to Matt @MentorGraphics for his help with PSS, inFact and DMA example

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

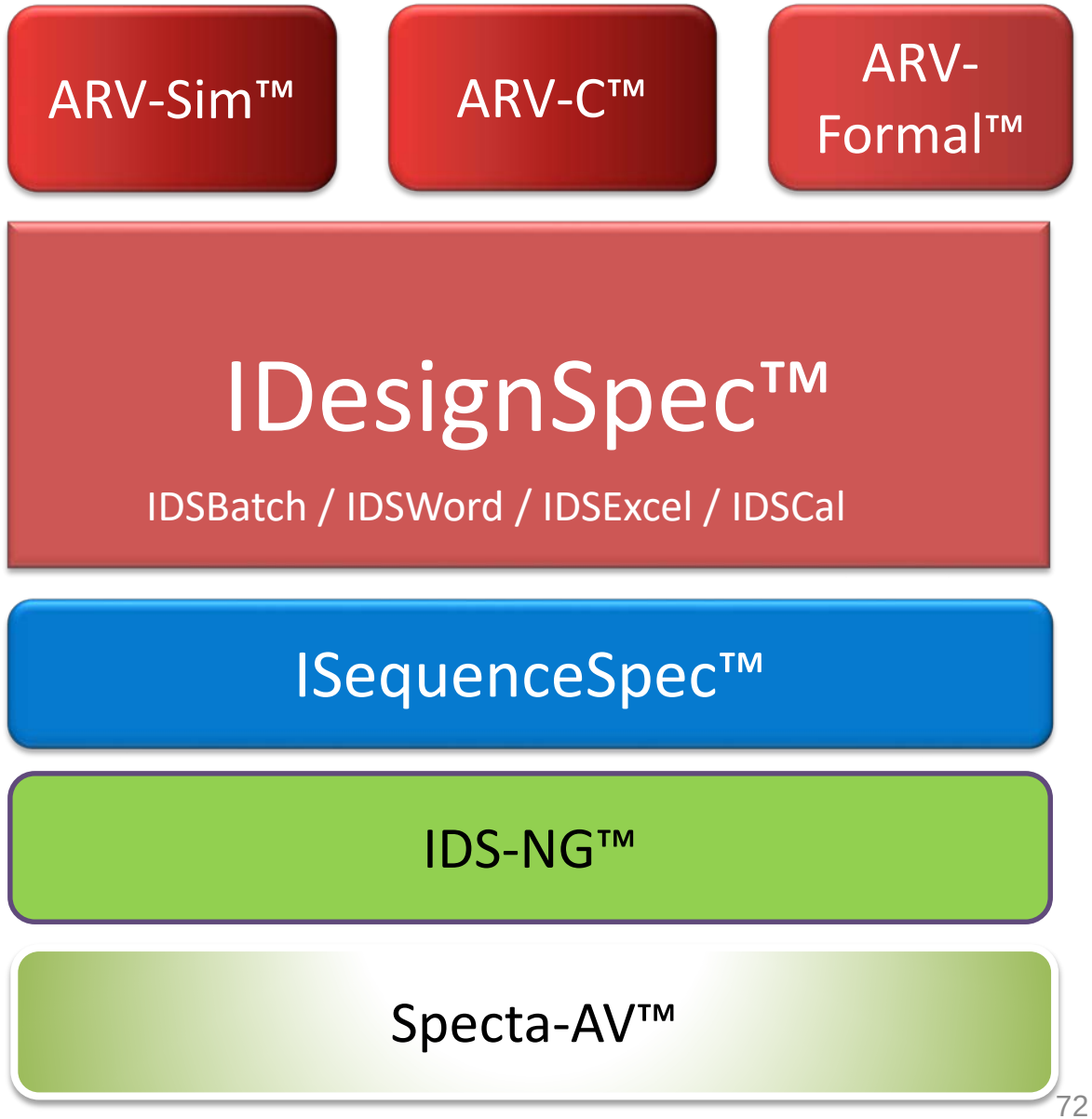
Create UVM sequences and Firmware routines from the specification

IDS-NextGen

Cross-platform HSI Layer Specification

Specta-AV

Automatic Verification



Hands-on Instruction

- Go to the website www.agnisys.com
- Go to MEDIA ROOM > Events > DVCon US 2020
- Click on ‘To download eval, click here’ link
- Fill the requisite information and the license will be sent to the email mentioned in the form
- Install the software
- Set the path for license file

Agnisys, Inc.

www.agnisys.com

support@agnisys.com

PH: 1 (855) VERIFYY

1 (855) 837-4399

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

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

APPENDIX

accellera SYSTEMS INITIATIVE

Software Access

Access	UVM	SystemRDL 1.0	SystemRDL 2.0
Read Only	RO	sw = r	sw = r onread=r
Read Clear	RC	sw=r rclr	sw=r onread =rclr
Read Set	RS	sw = r rset	sw=r onread =rset
Write Only	WO	sw = w	sw=w onwrite =w
Write One to Clear	-	sw = w woclr	sw=w onwrite =woclr
Write one to set	-	sw = w woset	sw=w onwrite =woset
Write one to toggle	-	-	sw=w onwrite =wot
Write zero to clear	-	-	sw=w onwrite =wzc
Write zero to set	-	-	sw=w onwrite =wzs

Software Access – Contd..

Access	UVM	SystemRDl 1.0	SystemRDL 2.0
Write zero to toggle	-	-	sw=w onwrite =wzt
Write clear	WOC	-	sw=w onwrite =wclr
Write set	WOS	-	sw=w onwrite =wset
Read Write	RW	sw = rw	sw=rw onread =r onwrite =w
Read / Write one to clear	W1C	sw = rw; woclr	sw=rw onread =r onwrite =woclr
Read /Write one to set	W1S	sw = rw; woset	sw = rw; onread =r onwrite =woset
Read /Write one to toggle	W1T	-	sw=rw onread =r onwrite =wot
Read /Write zero to clear	W0C	-	sw=rw onread =r onwrite =wzc

Software Access – Contd..

Access	UVM	SystemRDL 1.0	SystemRDL 2.0
Read /Write zero to set	W0S	-	sw=rw onread =r onwrite =wzs
Read / Write zero to toggle	W0T	-	sw=rw onread =r onwrite =wzt
Read/ Write clear	WC	-	sw=rw onread =r onwrite =wclr
Read / Write set	WS	-	sw=rw onread =r onwrite =wset
Write/Read clear	WRC	sw = rw rclr	sw=rw onread =rclr onwrite =w
Read Clear / write one to clear	-	sw =rw rclr woclr	sw=rw onread =rclr onwrite =woclr
Read clear / Write one to set	W1SRC	sw = rw rclr woset	sw=rw onread =rclr onwrite =woset
Read clear / Write one to toggle	-	-	sw=rw onread =rclr onwrite =wot

Software Access – Contd..

Access	UVM	SystemRDL 1.0	SystemRDL 2.0
Read clear / Write zero to clear	-	-	sw=rw onread =rclr onwrite =wzc
Read clear / Write zero to set	W0SRC	-	sw=rw onread =rclr onwrite =wzs
Read clear / Write zero to toggle	-	-	sw=rw onread =rclr onwrite =wzt
Read write clear	-	-	sw=rw onread =rclr onwrite =wclr
Read clear / write set	WSRC	-	sw=rw onread =rclr onwrite =wset
Read set / Write	WRS	sw = rw rset	sw=rw onread = rset onwrite =w
Read set / Write one to clear	W1CRS	sw = rw rset woclr	sw=rw onread = rset onwrite =woclr
Read set / Write one to set	-	sw = rw rset woset	sw=rw onread = rset onwrite =woset

Software Access – Contd..

Access	UVM	SystemRDL 1.0	SystemRDL 2.0
Read set / Write zero to clear	W0CRS	-	sw=rw onread = rset onwrite =wzc
Read set / Write zero to set	-	-	sw=rw onread = rset onwrite =wzs
Read set / Write zero to toggle	-	-	sw=rw onread = rset onwrite =wzt
Read set / Write clear	WCRS	-	sw=rw onread = rset onwrite =wclr
Read set / Write set	-	-	sw=rw onread = rset onwrite =wset
Read set / Write one to set	-	-	sw=rw onread = rset onwrite =woset
Read set / write zero to toggle	-	-	sw=rw onread = rset onwrite =wot
Read set / Write set	-	-	sw=rw onread = rset onwrite =wset
Read set / write zero to toggle	-	-	sw=rw onread = rset onwrite =wot