SoC Design and Practice

Jungrae Kim (dale40@skku.edu)
Semiconductor Systems Engineering

Register Automation

Recap

- Pipelining Priciples
- Pipeline Control

Outline

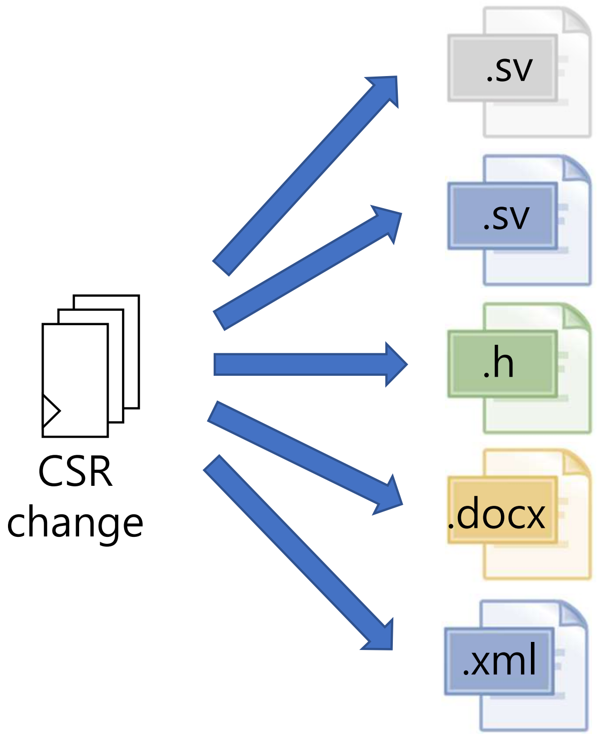
- Why Register Automation?
- Register Description Languages
  - SystemRDL
- RDL Tools
  - PeakRDL

SoC Designs Today

- Ever increasing design complexity
  - Especially on CSRs (Control & Status Registers)
    - A custom co-processor designed at Microsoft has
      - 335,996 CSRs comprised of 898,463 fields
      - 585,809 lines of Verilog RTL
      - 5% of total die area
- IP reuse
  - With a new chip
    - Need to modify hardware interfaces, software interface, and documentation
- SW/HW co-design
  - Frequent changes on CSRs

Changing a CSR

- Requires changes in multiple specification formats



Hardware designer has to modify RTL design

Verification engineer has to modify Testbenches

Software engineer has to modify Firmware

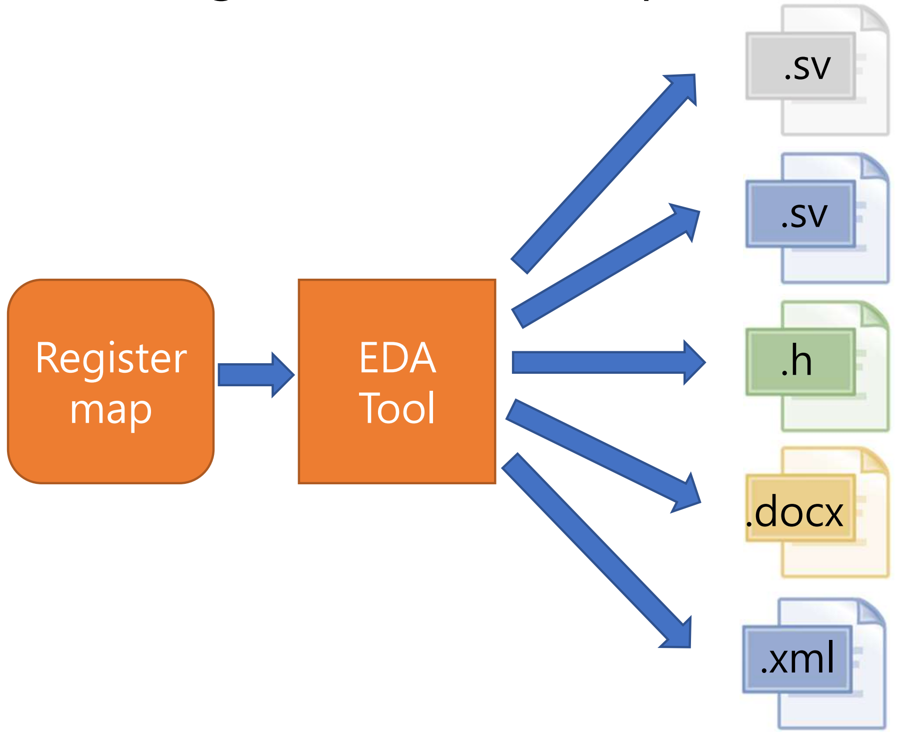
Technical writer has to modify User Guide

Integration engineer has to modify Connection/Testbench

- Manually keeping all these files consistently up-to-date is inefficient, costly, tedious, and error-prone

Register Automation

- Single Source, Multiple Outputs



Auto-generated CSR RTL design

Auto-generated register model for testbenches

Auto-generated C/C++ macros/structs for registers

Auto-generated register tables

Auto-generated integration info

Benefits of Register Automation

- Fast
- Consistent
- Correct by construction
- Standardized Verilog and C++ codes
- Complete, in-sync documentation
- Automatic register read/write tests
- High reusability

Register Description Language (RDL)

- Text-based description language that focuses exclusively on CSRs
  - Can describe and implement a wide variety of CSRs
  - Can automatically generate and synchronize views for specification, hardware design, verification, software development, and documentation

- RDL formats
  - SystemRDL
  - CSRSpec
  - IP-XACT
  - UVM Register Abstraction Layer File

SystemRDL

- Created at Cisco
- Released as Accellera 1.0 standard
- Version 2.0 released in Jan 2018
- Support specification-centric flows
  - Automatically generate
    - RTL bus interface
    - Verification model
    - C header and API
    - Documentation

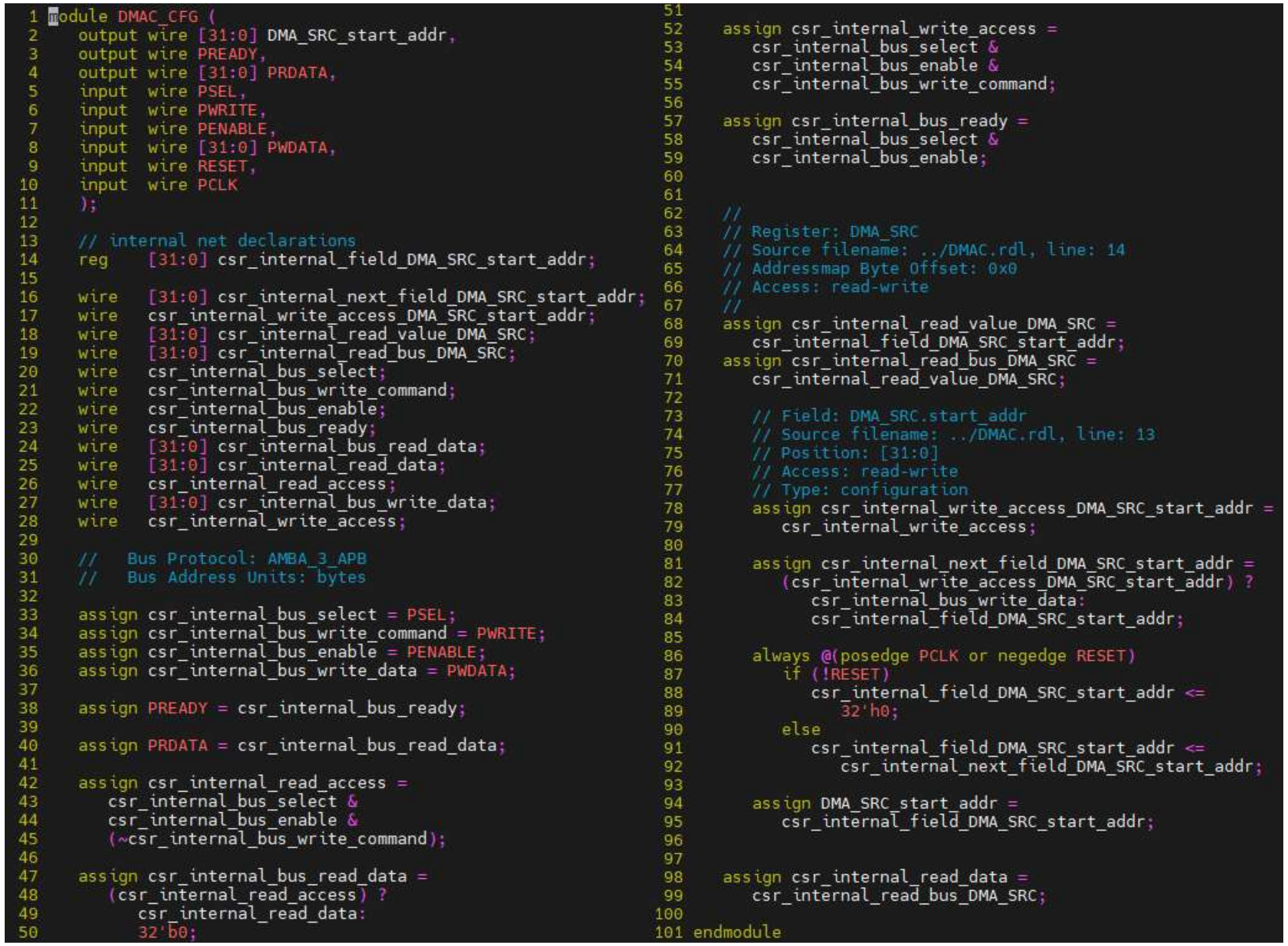
PeakRDL

- Open-source RDL automation tool
  - https://peakrdl.readthedocs.io/

Generated RTL

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>

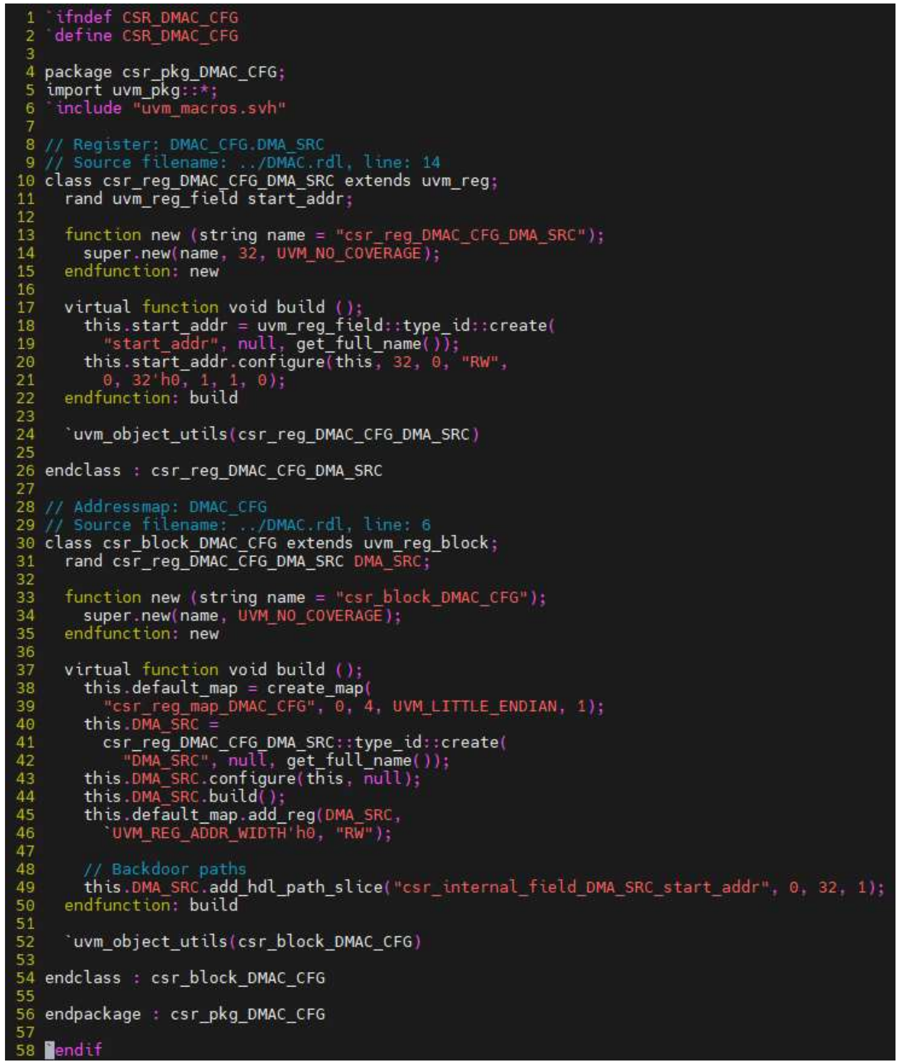


```
module DMAC_CFG (
   output wire [31:0] DMA_SRC_start_addr,
   output wire PREADY,
   output wire [31:0] PRDATA,
   input  wire PSEL,
   input  wire PWRITE,
   input  wire PENABLE,
   input  wire [31:0] PWDATA,
   input  wire RESET,
   input  wire PCLK
   );

   // internal net declarations
   reg    [31:0] csr_internal_field_DMA_SRC_start_addr;

   wire   [31:0] csr_internal_next_field_DMA_SRC_start_addr;
   wire   csr_internal_write_access_DMA_SRC_start_addr;
   wire   [31:0] csr_internal_read_value_DMA_SRC;
   wire   [31:0] csr_internal_read_bus_DMA_SRC;
   wire   csr_internal_bus_select;
   wire   csr_internal_bus_write_command;
   wire   csr_internal_bus_enable;
   wire   csr_internal_bus_ready;
   wire   [31:0] csr_internal_bus_read_data;
   wire   [31:0] csr_internal_read_data;
   wire   csr_internal_read_access;
   wire   [31:0] csr_internal_bus_write_data;
   wire   csr_internal_write_access;

   //   Bus Protocol: AMBA_3_APB
   //   Bus Address Units: bytes

   assign csr_internal_bus_select = PSEL;
   assign csr_internal_bus_write_command = PWRITE;
   assign csr_internal_bus_enable = PENABLE;
   assign csr_internal_bus_write_data = PWDATA;

   assign PREADY = csr_internal_bus_ready;

   assign PRDATA = csr_internal_bus_read_data;

   assign csr_internal_read_access =
      csr_internal_bus_select &
      csr_internal_bus_enable &
      (~csr_internal_bus_write_command);

   assign csr_internal_bus_read_data =
      (csr_internal_read_access) ?
         csr_internal_read_data:
         32'b0;

   assign csr_internal_write_access =
      csr_internal_bus_select &
      csr_internal_bus_enable &
      csr_internal_bus_write_command;

   assign csr_internal_bus_ready =
      csr_internal_bus_select &
      csr_internal_bus_enable;


   //
   // Register: DMA_SRC
   // Source filename: ../DMAC.rdl, line: 14
   // Addressmap Byte Offset: 0x0
   // Access: read-write
   //
   assign csr_internal_read_value_DMA_SRC =
      csr_internal_field_DMA_SRC_start_addr;
   assign csr_internal_read_bus_DMA_SRC =
      csr_internal_read_value_DMA_SRC;

      // Field: DMA_SRC.start_addr
      // Source filename: ../DMAC.rdl, line: 13
      // Position: [31:0]
      // Access: read-write
      // Type: configuration
      assign csr_internal_write_access_DMA_SRC_start_addr =
         csr_internal_write_access;

      assign csr_internal_next_field_DMA_SRC_start_addr =
         (csr_internal_write_access_DMA_SRC_start_addr) ?
            csr_internal_bus_write_data:
            csr_internal_field_DMA_SRC_start_addr;

      always @(posedge PCLK or negedge RESET)
         if (!RESET)
            csr_internal_field_DMA_SRC_start_addr <=
               32'h0;
         else
            csr_internal_field_DMA_SRC_start_addr <=
               csr_internal_next_field_DMA_SRC_start_addr;

      assign DMA_SRC_start_addr =
         csr_internal_field_DMA_SRC_start_addr;


   assign csr_internal_read_data =
      csr_internal_read_bus_DMA_SRC;

endmodule
```

Generated Verification Codes (UVM)

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>

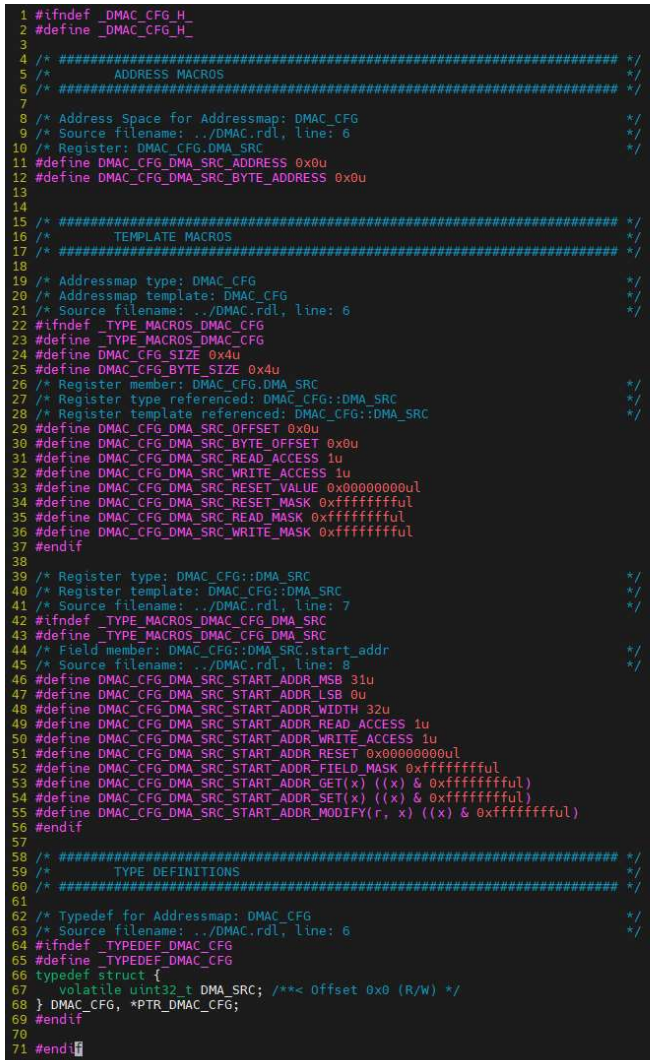


```
`ifndef CSR_DMAC_CFG
`define CSR_DMAC_CFG

package csr_pkg_DMAC_CFG;
import uvm_pkg::*;
`include "uvm_macros.svh"

// Register: DMAC_CFG.DMA_SRC
// Source filename: ../DMAC.rdl, line: 14
class csr_reg_DMAC_CFG_DMA_SRC extends uvm_reg;
  rand uvm_reg_field start_addr;

  function new (string name = "csr_reg_DMAC_CFG_DMA_SRC");
    super.new(name, 32, UVM_NO_COVERAGE);
  endfunction: new

  virtual function void build ();
    this.start_addr = uvm_reg_field::type_id::create(
      "start_addr", null, get_full_name());
    this.start_addr.configure(this, 32, 0, "RW",
      0, 32'h0, 1, 1, 0);
  endfunction: build

  `uvm_object_utils(csr_reg_DMAC_CFG_DMA_SRC)

endclass : csr_reg_DMAC_CFG_DMA_SRC

// Addressmap: DMAC_CFG
// Source filename: ../DMAC.rdl, line: 6
class csr_block_DMAC_CFG extends uvm_reg_block;
  rand csr_reg_DMAC_CFG_DMA_SRC DMA_SRC;

  function new (string name = "csr_block_DMAC_CFG");
    super.new(name, UVM_NO_COVERAGE);
  endfunction: new

  virtual function void build ();
    this.default_map = create_map(
      "csr_reg_map_DMAC_CFG", 0, 4, UVM_LITTLE_ENDIAN, 1);
    this.DMA_SRC =
      csr_reg_DMAC_CFG_DMA_SRC::type_id::create(
        "DMA_SRC", null, get_full_name());
    this.DMA_SRC.configure(this, null);
    this.DMA_SRC.build();
    this.default_map.add_reg(DMA_SRC,
      `UVM_REG_ADDR_WIDTH'h0, "RW");

    // Backdoor paths
    this.DMA_SRC.add_hdl_path_slice("csr_internal_field_DMA_SRC_start_addr", 0, 32, 1);
  endfunction: build

  `uvm_object_utils(csr_block_DMAC_CFG)

endclass : csr_block_DMAC_CFG

endpackage : csr_pkg_DMAC_CFG

`endif
```

Generated C++ Header

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>

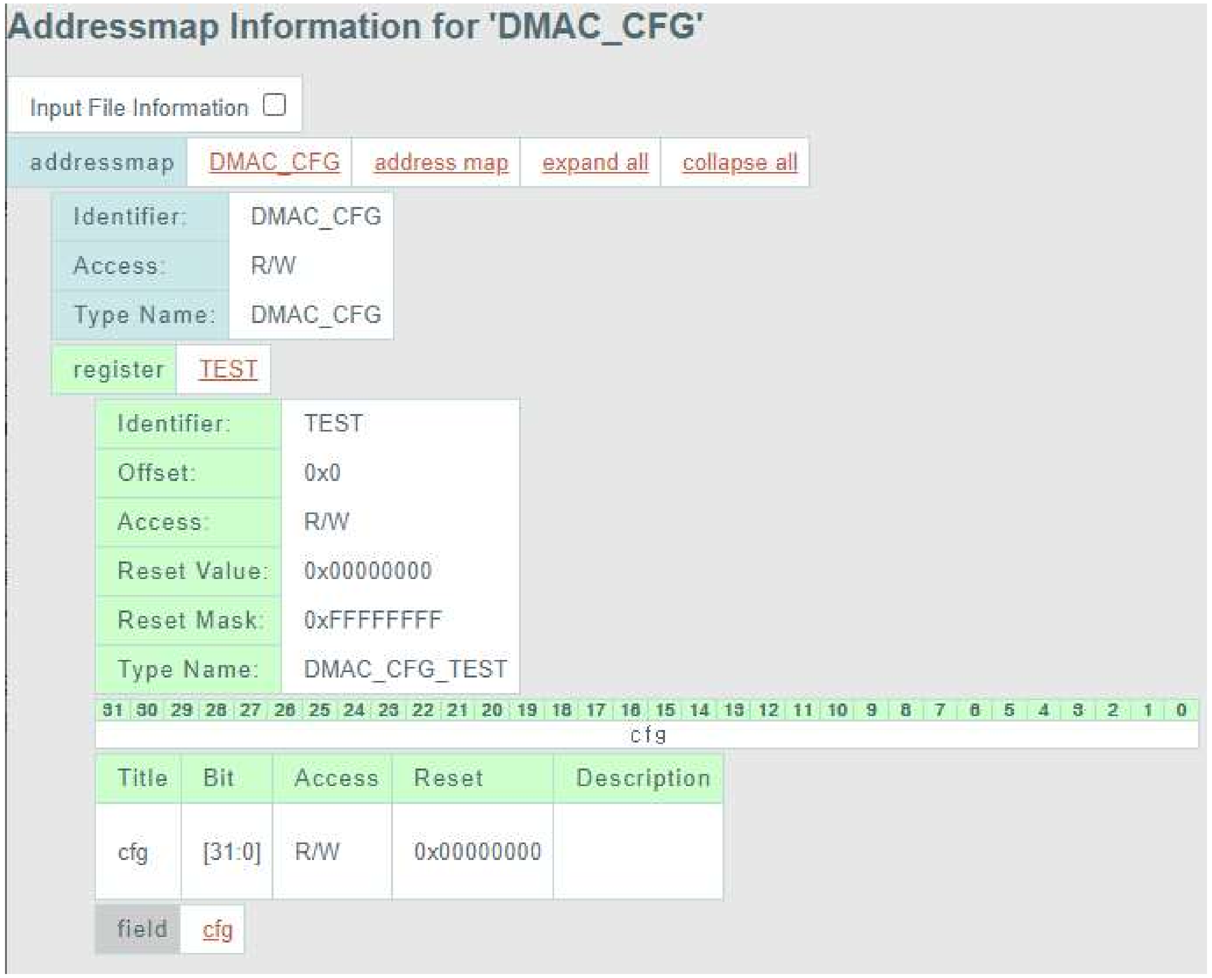


```
#ifndef _DMAC_CFG_H_
#define _DMAC_CFG_H_

/* ####################################################################### */
/*        ADDRESS MACROS                                                    */
/* ####################################################################### */

/* Address Space for Addressmap: DMAC_CFG                                 */
/* Source filename: ../DMAC.rdl, line: 6                                  */
/* Register: DMAC_CFG.DMA_SRC                                             */
#define DMAC_CFG_DMA_SRC_ADDRESS 0x0u
#define DMAC_CFG_DMA_SRC_BYTE_ADDRESS 0x0u


/* ####################################################################### */
/*        TEMPLATE MACROS                                                   */
/* ####################################################################### */

/* Addressmap type: DMAC_CFG                                              */
/* Addressmap template: DMAC_CFG                                          */
/* Source filename: ../DMAC.rdl, line: 6                                  */
#ifndef _TYPE_MACROS_DMAC_CFG
#define _TYPE_MACROS_DMAC_CFG
#define DMAC_CFG_SIZE 0x4u
#define DMAC_CFG_BYTE_SIZE 0x4u
/* Register member: DMAC_CFG.DMA_SRC                                      */
/* Register type referenced: DMAC_CFG::DMA_SRC                            */
/* Register template referenced: DMAC_CFG::DMA_SRC                        */
#define DMAC_CFG_DMA_SRC_OFFSET 0x0u
#define DMAC_CFG_DMA_SRC_BYTE_OFFSET 0x0u
#define DMAC_CFG_DMA_SRC_READ_ACCESS 1u
#define DMAC_CFG_DMA_SRC_WRITE_ACCESS 1u
#define DMAC_CFG_DMA_SRC_RESET_VALUE 0x00000000ul
#define DMAC_CFG_DMA_SRC_RESET_MASK 0xfffffffful
#define DMAC_CFG_DMA_SRC_READ_MASK 0xfffffffful
#define DMAC_CFG_DMA_SRC_WRITE_MASK 0xfffffffful
#endif

/* Register type: DMAC_CFG::DMA_SRC                                       */
/* Register template: DMAC_CFG::DMA_SRC                                   */
/* Source filename: ../DMAC.rdl, line: 7                                  */
#ifndef _TYPE_MACROS_DMAC_CFG_DMA_SRC
#define _TYPE_MACROS_DMAC_CFG_DMA_SRC
/* Field member: DMAC_CFG::DMA_SRC.start_addr                             */
/* Source filename: ../DMAC.rdl, line: 8                                  */
#define DMAC_CFG_DMA_SRC_START_ADDR_MSB 31u
#define DMAC_CFG_DMA_SRC_START_ADDR_LSB 0u
#define DMAC_CFG_DMA_SRC_START_ADDR_WIDTH 32u
#define DMAC_CFG_DMA_SRC_START_ADDR_READ_ACCESS 1u
#define DMAC_CFG_DMA_SRC_START_ADDR_WRITE_ACCESS 1u
#define DMAC_CFG_DMA_SRC_START_ADDR_RESET 0x00000000ul
#define DMAC_CFG_DMA_SRC_START_ADDR_FIELD_MASK 0xfffffffful
#define DMAC_CFG_DMA_SRC_START_ADDR_GET(x) ((x) & 0xfffffffful)
#define DMAC_CFG_DMA_SRC_START_ADDR_SET(x) ((x) & 0xfffffffful)
#define DMAC_CFG_DMA_SRC_START_ADDR_MODIFY(r, x) ((x) & 0xfffffffful)
#endif

/* ####################################################################### */
/*        TYPE DEFINITIONS                                                  */
/* ####################################################################### */

/* Typedef for Addressmap: DMAC_CFG                                       */
/* Source filename: ../DMAC.rdl, line: 6                                  */
#ifndef _TYPEDEF_DMAC_CFG
#define _TYPEDEF_DMAC_CFG
typedef struct {
   volatile uint32_t DMA_SRC; /**< Offset 0x0 (R/W) */
} DMAC_CFG, *PTR_DMAC_CFG;
#endif

#endif
```

Generated Document (HTML)

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>



Addressmap Information for 'DMAC_CFG'

Identifier:	DMAC_CFG
Access:	R/W
Type Name:	DMAC_CFG

Identifier:	TEST
Offset:	0x0
Access:	R/W
Reset Value:	0x00000000
Reset Mask:	0xFFFFFFFF
Type Name:	DMAC_CFG_TEST

Title	Bit	Access	Reset	Description
cfg	[31:0]	R/W	0x00000000	

Generated Document (docx)

Scalable Archi Lab.

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>

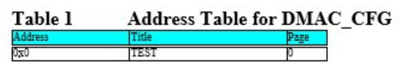
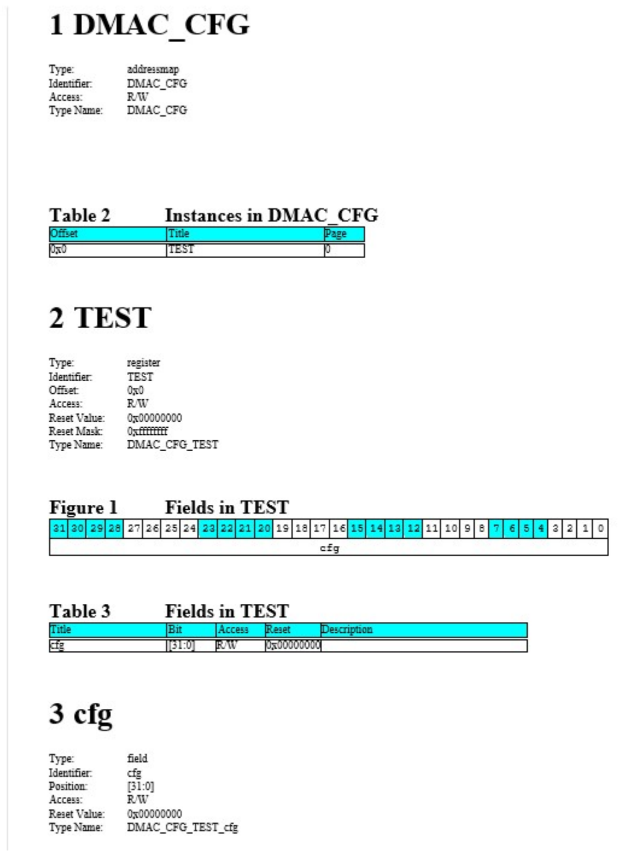


Table 1 Address Table for DMAC_CFG

Address	Title	Page
0x0	TEST	0



1 DMAC_CFG

Type:	addressmap
Identifier:	DMAC_CFG
Access:	R/W
Type Name:	DMAC_CFG

Table 2 Instances in DMAC_CFG

Offset	Title	Page
0x0	TEST	0

2 TEST

Type:	register
Identifier:	TEST
Offset:	0x0
Access:	R/W
Reset Value:	0x00000000
Reset Mask:	0xffffffff
Type Name:	DMAC_CFG_TEST

Figure 1 Fields in TEST

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

Table 3 Fields in TEST

Title	Bit	Access	Reset	Description
cfg	[31:0]	R/W	0x00000000	

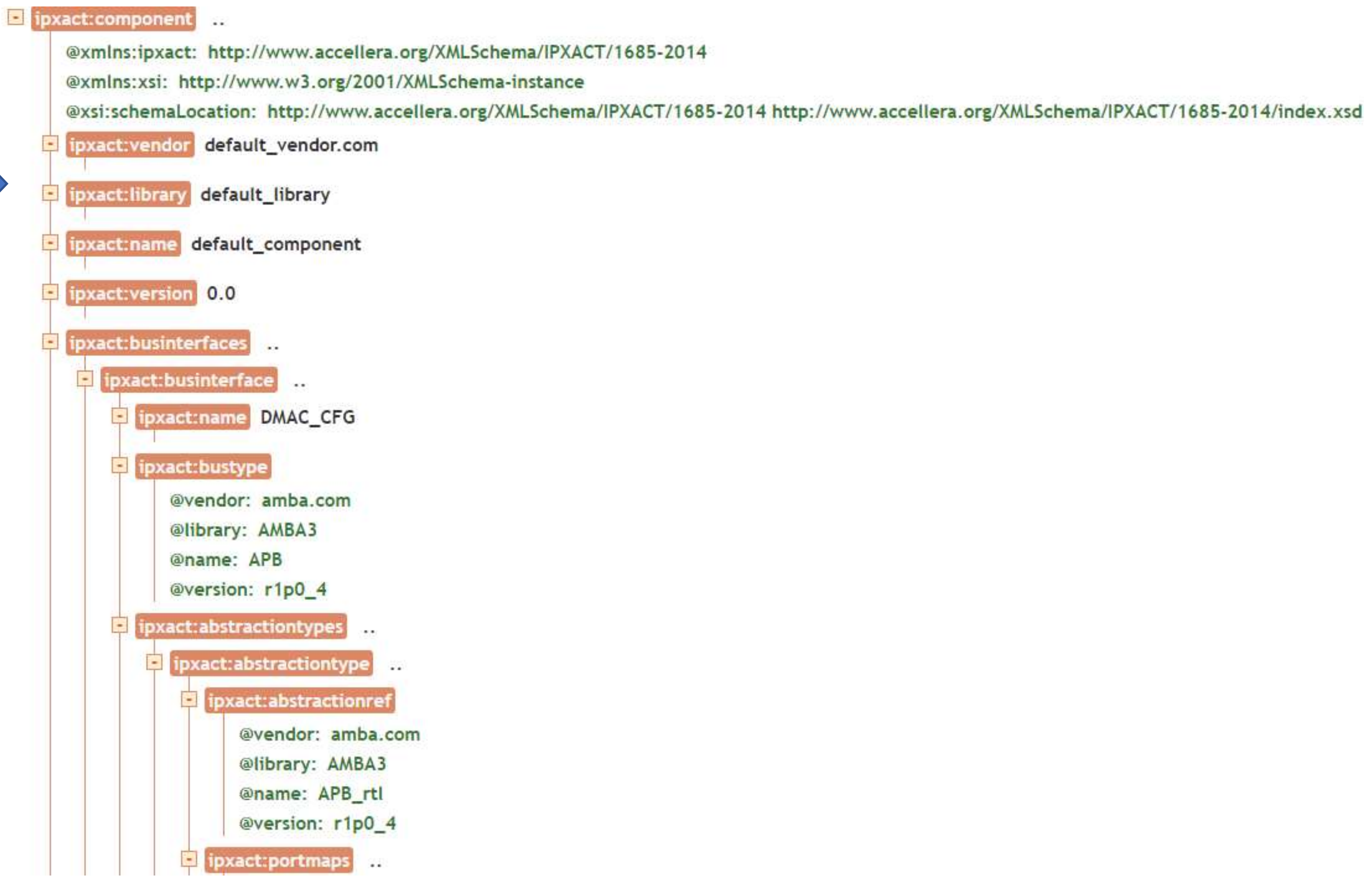
3 cfg

Type:	field
Identifier:	cfg
Position:	[31:0]
Access:	R/W
Reset Value:	0x00000000
Type Name:	DMAC_CFG_TEST_cfg

Generated Integration File (IP-XACT)

```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>

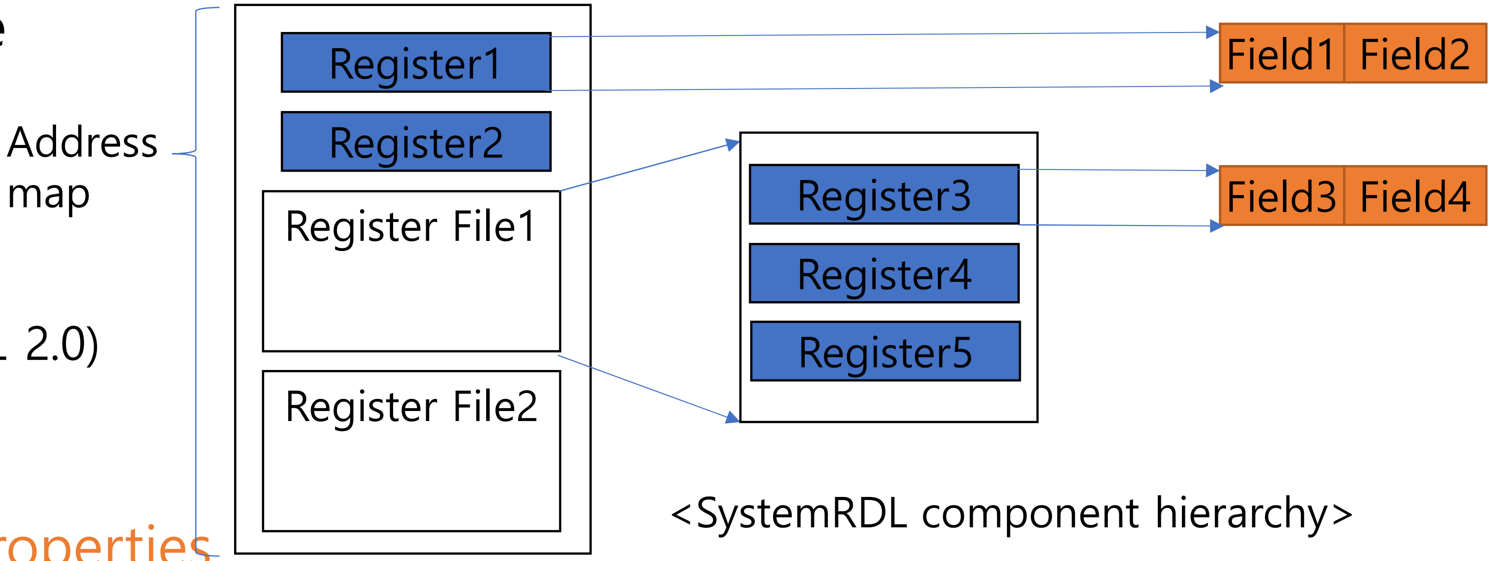


```
ipxact:component ..
    @xmlns:ipxact: http://www.accellera.org/XMLSchema/IPXACT/1685-2014
    @xmlns:xsi: http://www.w3.org/2001/XMLSchema-instance
    @xsi:schemaLocation: http://www.accellera.org/XMLSchema/IPXACT/1685-2014 http://www.accellera.org/XMLSchema/IPXACT/1685-2014/index.xsd
    ipxact:vendor default_vendor.com
    ipxact:library default_library
    ipxact:name default_component
    ipxact:version 0.0
    ipxact:businterfaces ..
        ipxact:businterface ..
            ipxact:name DMAC_CFG
            ipxact:bustype
                @vendor: amba.com
                @library: AMBA3
                @name: APB
                @version: r1p0_4
            ipxact:abstractiontypes ..
                ipxact:abstractiontype ..
                    ipxact:abstractionref
                        @vendor: amba.com
                        @library: AMBA3
                        @name: APB_rtl
                        @version: r1p0_4
                    ipxact:portmaps ..
```

SystemRDL

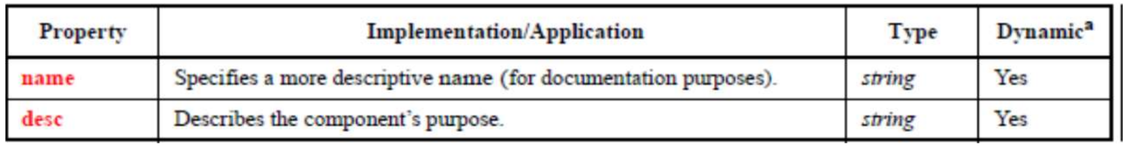
SystemRDL

- SystemRDL describes CSRs in a hierarchy of components
- Components include
  - Address map
  - Register file
  - Register
  - Field
  - Memory (SystemRDL 2.0)



<SystemRDL component hierarchy>

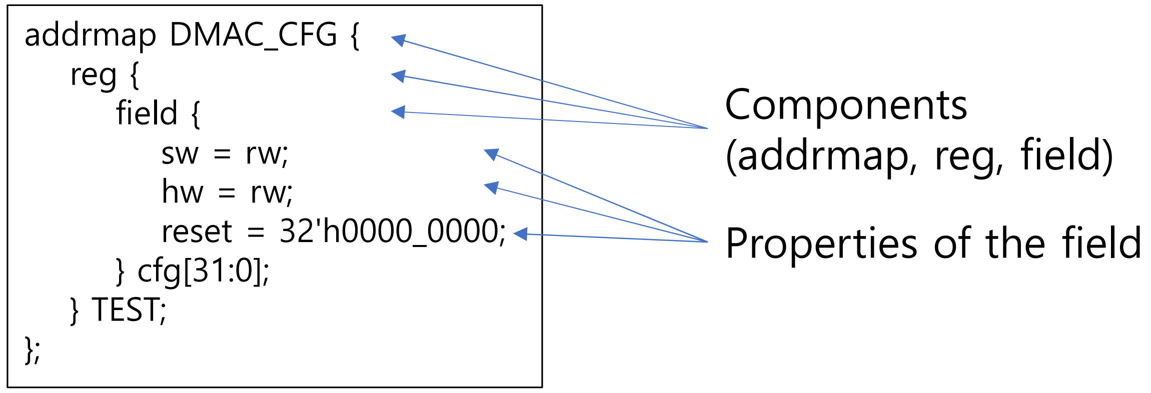
- A component has properties



Property	Implementation/Application	Type	Dynamic[a]
name	Specifies a more descriptive name (for documentation purposes).	*string*	Yes
desc	Describes the component's purpose.	*string*	Yes

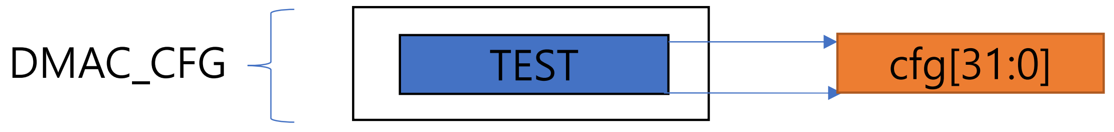
<SystemRDL universal component properties>

SystemRDL Example



```
addrmap DMAC_CFG {
    reg {
        field {
            sw = rw;
            hw = rw;
            reset = 32'h0000_0000;
        } cfg[31:0];
    } TEST;
};
```

<DMAC.rdl>



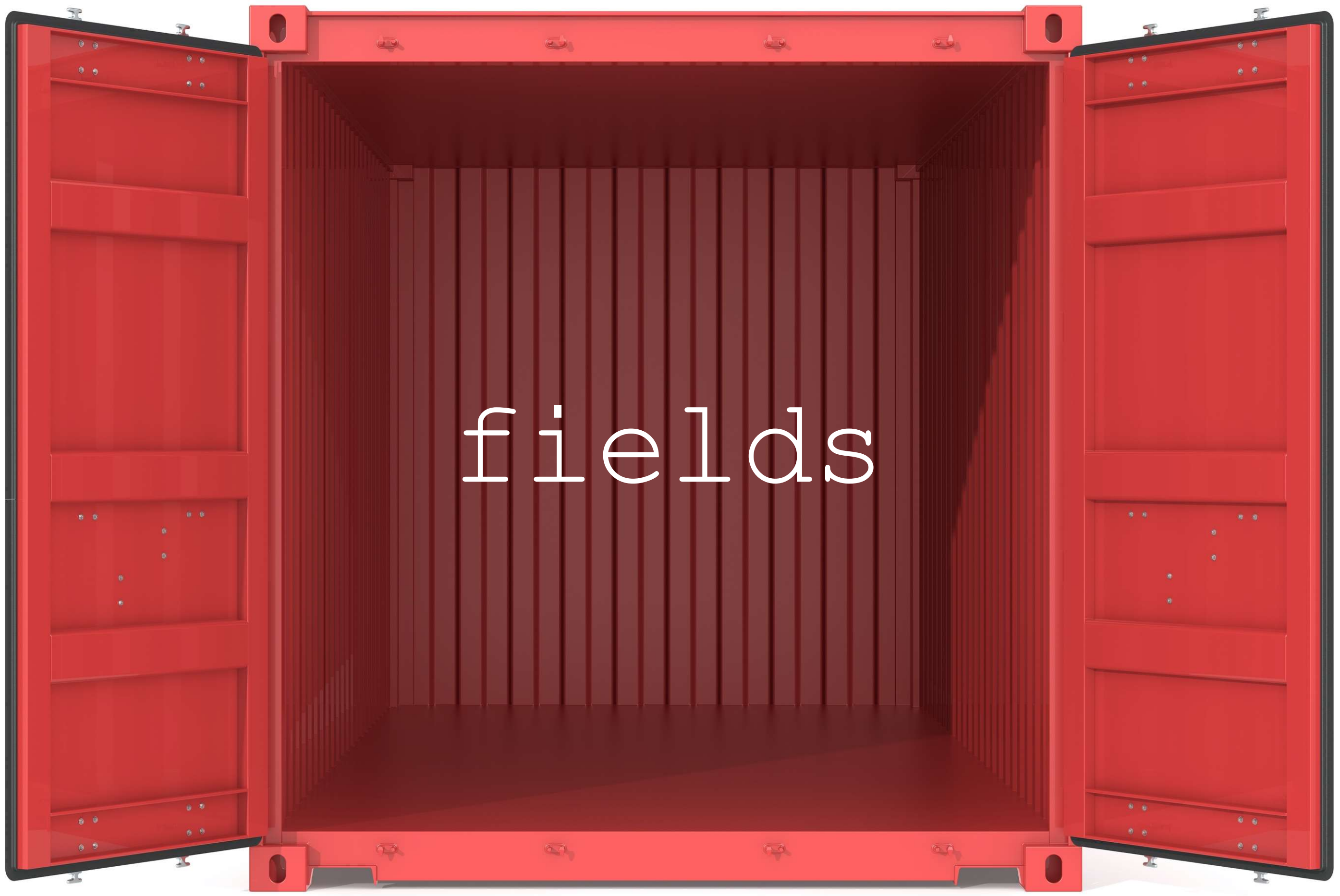
<Address map> <register> <field>

<SystemRDL component hierarchy>

`addrmap` is a Container

`regfile` is a Container

`register` is a Container



`field` is NOT a Container

- The lowest level structural component
- Stores the bit information of a register

Scalable
Archi
Lab.

SystemRDL - Field

There are many types of `field`

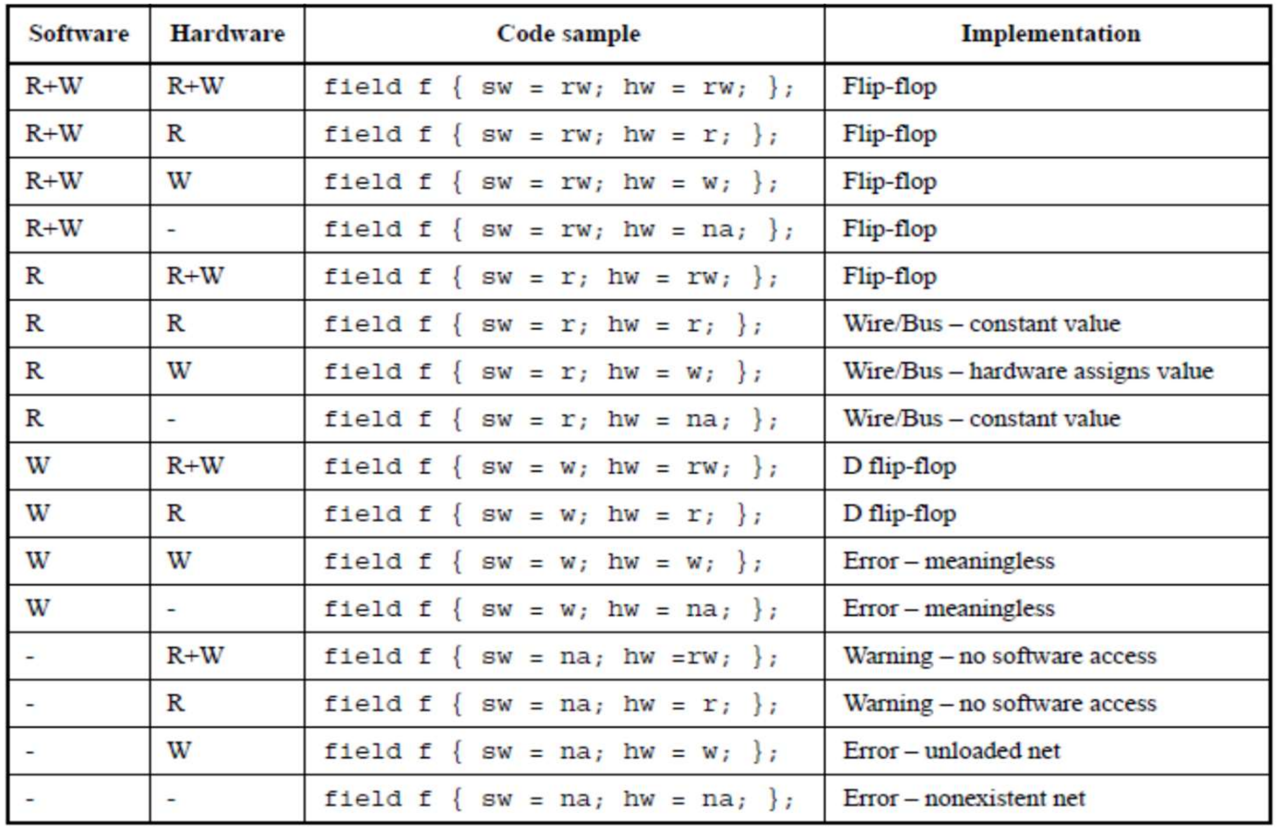
- Common types of CSR fields
  - Configuration
  - Status
  - Constant
  - Counter
  - Command
  - Interrupt
  - …

- You can instantiate a CSR type of your choice
  - By using **field access properties**

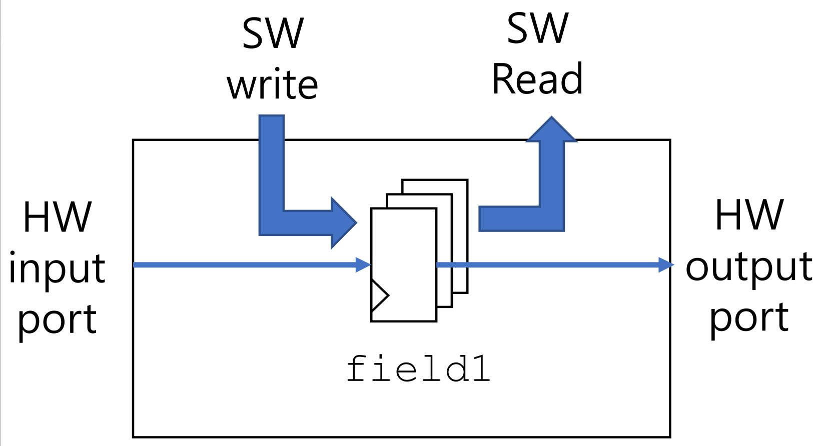
Field Access Properties

Scalable Archi Lab.

Property	Behavior/Application
hw={rw|wr|r|w|na}	Design's ability to sample/update a **field**.
sw={rw|wr|r|w|na}	Programmer's ability to read/write a **field**.



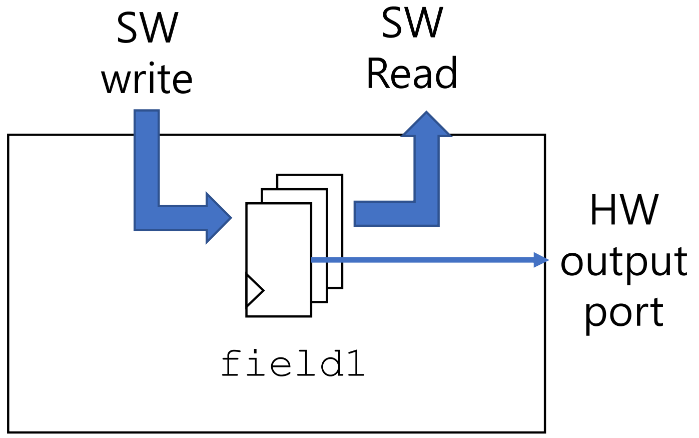
Software	Hardware	Code sample	Implementation
R+W	R+W	`field f { sw = rw; hw = rw; };`	Flip-flop
R+W	R	`field f { sw = rw; hw = r; };`	Flip-flop
R+W	W	`field f { sw = rw; hw = w; };`	Flip-flop
R+W	-	`field f { sw = rw; hw = na; };`	Flip-flop
R	R+W	`field f { sw = r; hw = rw; };`	Flip-flop
R	R	`field f { sw = r; hw = r; };`	Wire/Bus – constant value
R	W	`field f { sw = r; hw = w; };`	Wire/Bus – hardware assigns value
R	-	`field f { sw = r; hw = na; };`	Wire/Bus – constant value
W	R+W	`field f { sw = w; hw = rw; };`	D flip-flop
W	R	`field f { sw = w; hw = r; };`	D flip-flop
W	W	`field f { sw = w; hw = w; };`	Error – meaningless
W	-	`field f { sw = w; hw = na; };`	Error – meaningless
-	R+W	`field f { sw = na; hw =rw; };`	Warning – no software access
-	R	`field f { sw = na; hw = r; };`	Warning – no software access
-	W	`field f { sw = na; hw = w; };`	Error – unloaded net
-	-	`field f { sw = na; hw = na; };`	Error – nonexistent net



<Generic flip-flop implementation>

Configuration `field`

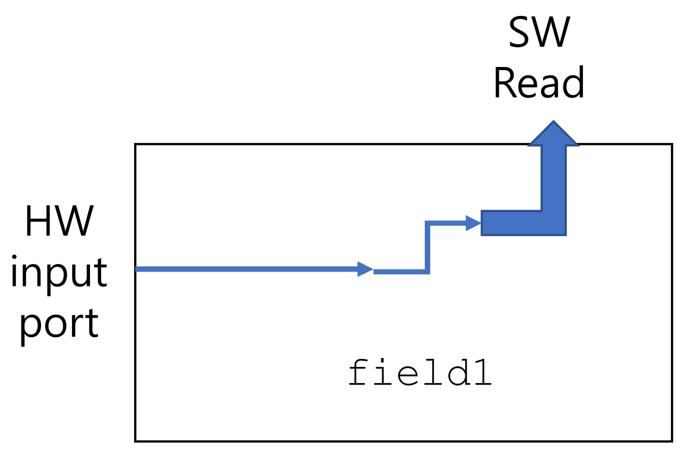
```
field {
    sw=rw;
    hw=r;
} field1;
```



Scalable Archi Lab.

Status `field`

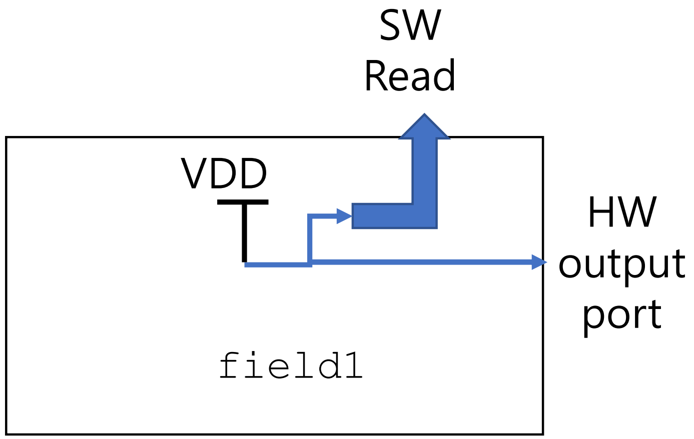
```
field {
    sw=r;
    hw=w;
} field1;
```



Constant `field`

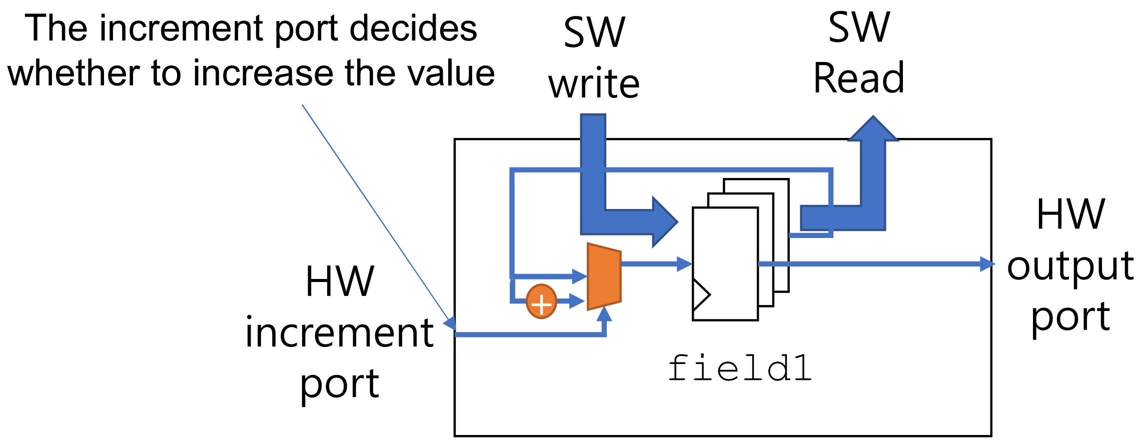
```
field {
    sw=r;
    hw=r;
    reset=1’b1;
} field1;
```

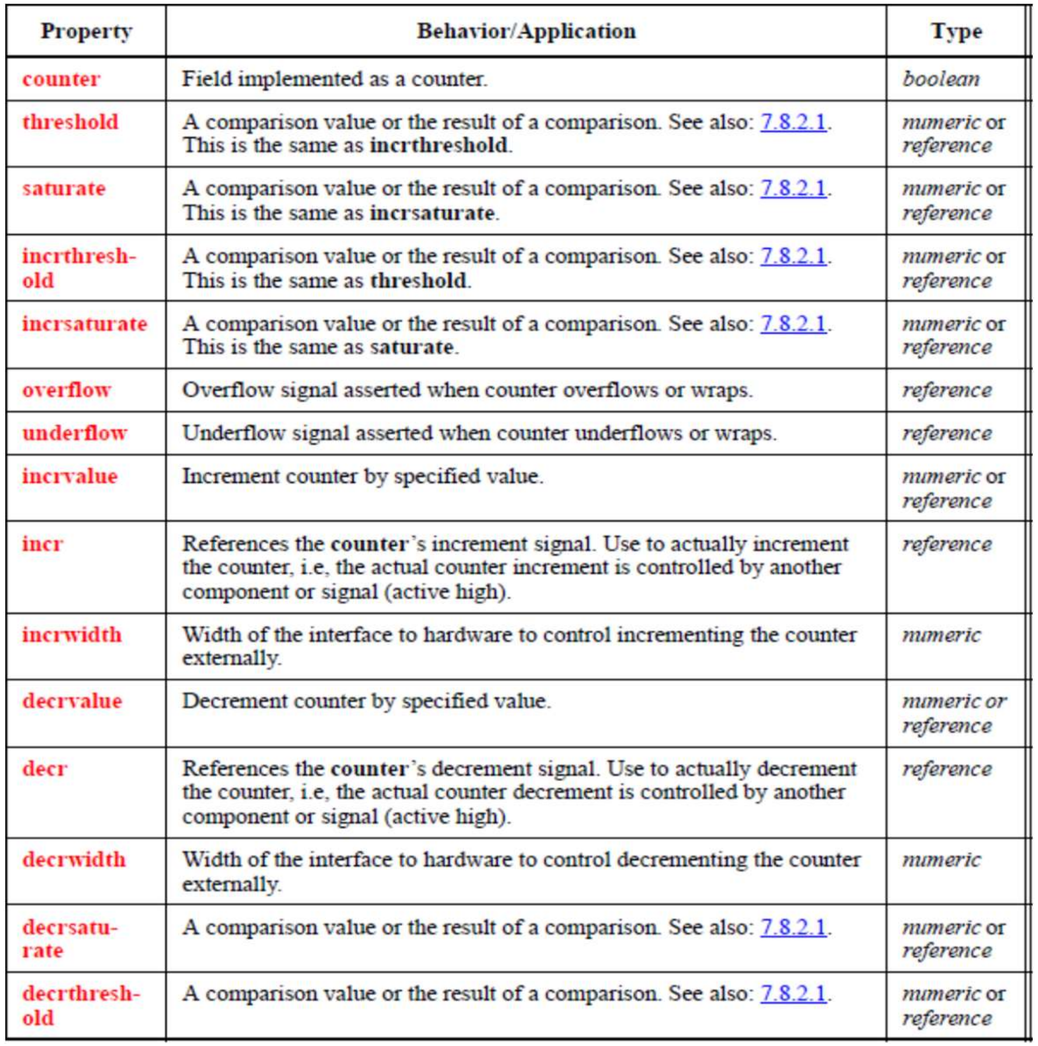
The `reset` specifies the constant value.



Counter `field`

```
field {
    sw=rw;
    hw=r;
    counter;
    incrvalue=2;
    incrsaturate=15;
    incrthreshold=10;
} field1[3:0];
```





Property	Behavior/Application	Type
counter	Field implemented as a counter.	*boolean*
threshold	A comparison value or the result of a comparison. See also: 7.8.2.1. This is the same as **incrthreshold**.	*numeric* or *reference*
saturate	A comparison value or the result of a comparison. See also: 7.8.2.1. This is the same as **incrsaturate**.	*numeric* or *reference*
incrthresh-old	A comparison value or the result of a comparison. See also: 7.8.2.1. This is the same as **threshold**.	*numeric* or *reference*
incrsaturate	A comparison value or the result of a comparison. See also: 7.8.2.1. This is the same as **saturate**.	*numeric* or *reference*
overflow	Overflow signal asserted when counter overflows or wraps.	*reference*
underflow	Underflow signal asserted when counter underflows or wraps.	*reference*
incrvalue	Increment counter by specified value.	*numeric* or *reference*
incr	References the **counter**'s increment signal. Use to actually increment the counter, i.e, the actual counter increment is controlled by another component or signal (active high).	*reference*
incrwidth	Width of the interface to hardware to control incrementing the counter externally.	*numeric*
decrvalue	Decrement counter by specified value.	*numeric* or *reference*
decr	References the **counter**'s decrement signal. Use to actually decrement the counter, i.e, the actual counter decrement is controlled by another component or signal (active high).	*reference*
decrwidth	Width of the interface to hardware to control decrementing the counter externally.	*numeric*
decrsatu-rate	A comparison value or the result of a comparison. See also: 7.8.2.1.	*numeric* or *reference*
decrthresh-old	A comparison value or the result of a comparison. See also: 7.8.2.1.	*numeric* or *reference*

Defining Field Size

```
…
field {
   fieldwidth = 4;
} exampleField;
…
```

Explicitly
By Property

Implicitly
By Position

```
…
field {
} exampleField[3:0];
…
```

```
…
field {
} exampleField;
…
```

Default, One Bit

Defining Field Location

By
Position

```
…
field {
} exampleField[4:4];
…
```

```
…
field {
} exampleField;
…
```

Let the tool define

Command `field`

```
field {
    sw=w;
    hw=r;
    reset=1'b0;
    singlepulse;
} field1;
```

The field asserts for one cycle when written 1 and then clears back to 0 on the next cycle.
This creates a sigle-cycle pulse on the hardware interface

Interrupt `field`

```
field {
    sw=rw;
    hw=w;
    intr;
    woclr;
} field1;
```

Software Access Properties

Property	Behavior/Application	Type
rclr	Clear on read (field=0)	Boolean
rset	Set on read (field=all 1s)	Boolean
woset	Write one to set (field=field|wdata)	Boolean
woclr	Write one to clear (field=field&~wdata)	Boolean
singlepulse	The field asserts for one cycle when written 1 and then clears back to 0 on the next cycle. This creates a sigle-cycle purlse on the hardware interface	Boolean

```
reg register1{
  field {} fld1,fld2;
  field {
    hw=rw;
    sw=rw;
    rclr;
    woset;
  } fld3;
  field {
    hw=r;
    sw=w;
    reset=1'b0;
    singlepulse;
  } fld4;
};
```

Hardware Access Properties

Property	Description
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. Bits that are set to 1 will be updated
hwmask	Determines which bits may be updated after any write enables. Bits that are set to 1 will not be updated

```
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;
};
```

SystemRDL - Register

Register

- Represents a single address
  - A set of one or more field instances that are atomically accessible by software

Register Definition

Definitive definition

- Separate statements for register definition and register instantiation
- Suitable for reuse.
- Synax: **reg** *reg_name* **{**[*reg_body*]**};**
  reg_name reg_instance;

```
reg r1 {
  regwidth=32;
  field f1{
    hw=rw;
    sw=rw;
  };
  f1 field1[31:0] = 31’b0;
};
r1 reg1 @0x100;
```

Anonymous definition

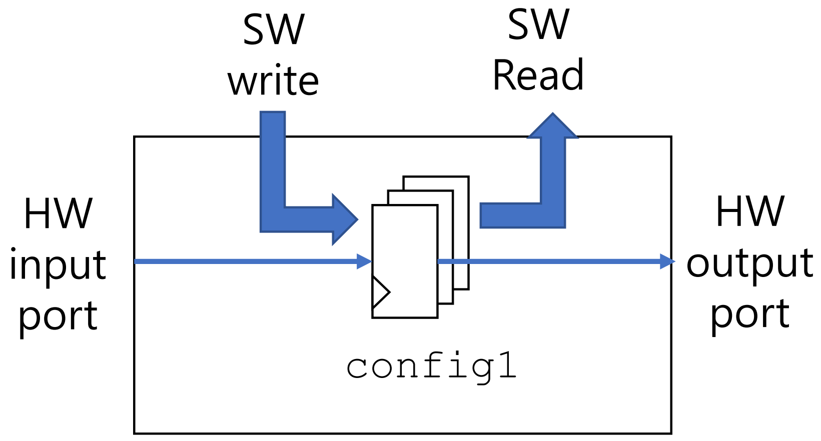
- Instantiate the component in the same statement
- Suitable for components that are used once
- Synax: **reg {**[*reg_body*]**}** *reg_instance;*

```
reg {
  field {
    hw=rw;
    sw=rw;
  } field1[31:0];
} reg1 @0x100;
```

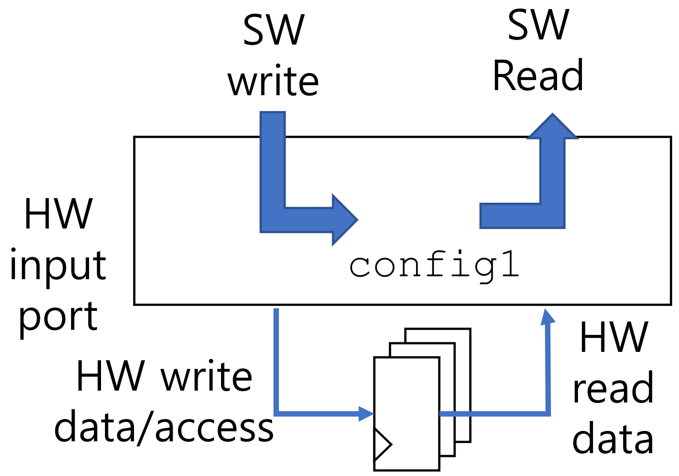
Register Instantiation

Instantiation forms	Behavior/Application
internal	All register logic is created by the SystemRDL compiler for the instantiatin (default)
external	The register is implemented by the designer and the SystemRDL compiler provides an interface from the instantiation
alias	Alias registers are used where designers want to allow alternative software access to register. SystemRDL allows designers to specify alias register for internal or external registers

```
reg myReg {
  field {
    sw=rw;
    hw=rw;
  } config1;
};
myReg reg1;
external myReg reg2;
```



<reg1/Internal instantiation>

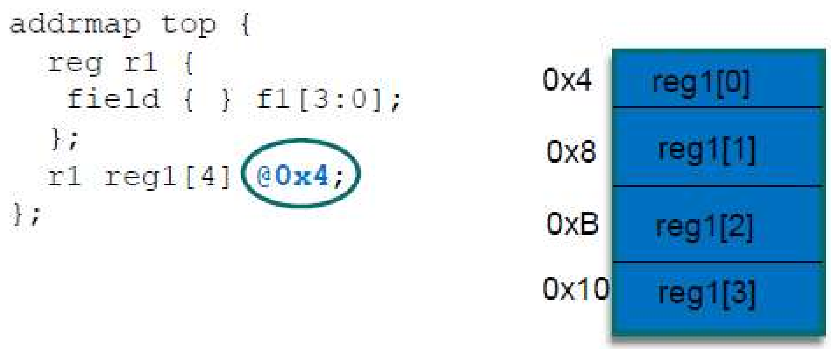


<reg2/External instantiation>

Register Offsets

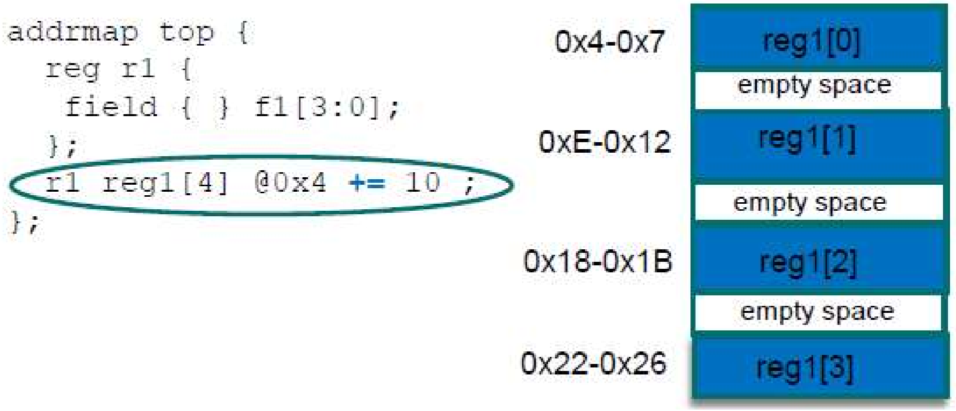
- You can specify the register address offset
  - @**expression**: Specifies the address offset (within the parent component)
  - +=**expression**: Specifies the address stride when instantiating an array of components (controls the spacing of the components).
  - %=**expression**: Specifies the alignment of the next address when instantiating a component (controls the alignment of the components).
- Otherwise, SystemRDL compiler will decide

Offset (@)



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

Stride (+=)

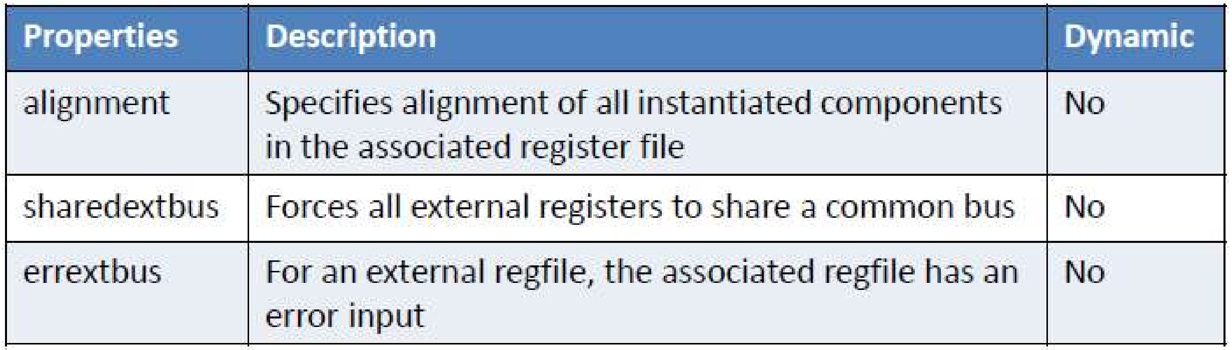


```
addrmap top {
  reg r1 {
   field { } f1[3:0];
  };
  r1 reg1[4] @0x4 += 10 ;
};
```

RegisterFile

- Logical grouping of one or more register and register file instances.
  - The only difference between regfile and addrmap is
    - An addrmap defines an RTL implementation boundary, where the regfile does not.

```
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;
};
```



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

Properties of regfile

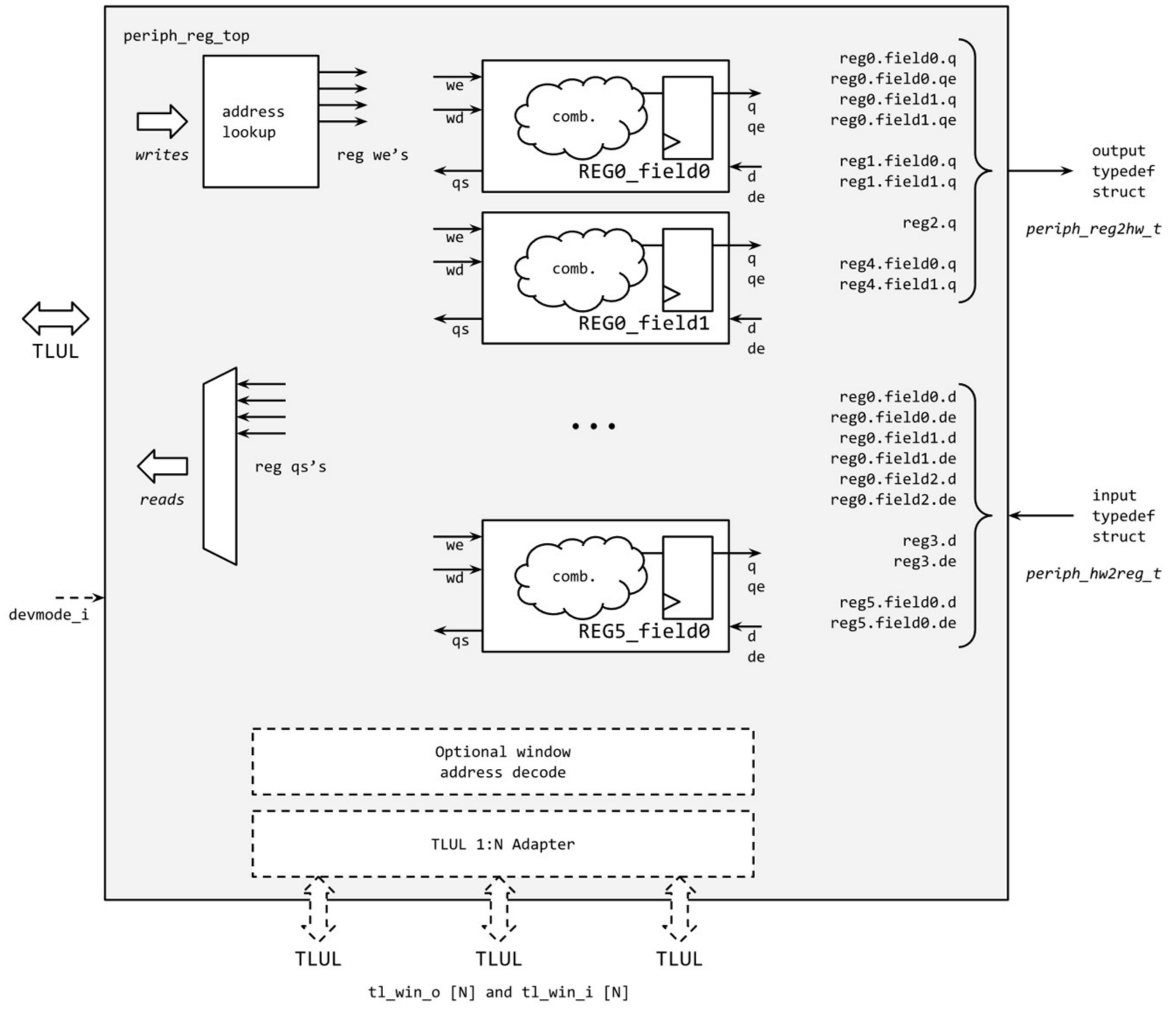
Addressmap

- Contains registers, register files, and/or other address maps
- Assigns a virtual address or final addresses.
- Specifies RTL module boundary

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

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

Generated RTL Example



Summary

- Why Register Automation?
- Register Description Languages
    - SystemRDL
- RDL Tools
    - PeakRDL