SystemRDL Guideline

Neo

2022/03/07

Contents

1 Introduction

This document provides guideline for coding register module with SystemRDL ([1])

2

FIELD DESCRIPTION

2.1 Naming Convention

Each RDL **field** will be generated to an instance of `field` module. In generated RTL, naming convention of field stem is `<reg_inst_name>__<field_inst_name>`. Other signals belong to the field are named by prefixing/suffixing elements. e.g., Register instance name is `ring_cfg`, Field instance name is `rd_ptr`:

1. `field` instance name is `<stem>` prefixed with `x__`: `x__ring_cfg__rd_ptr`
2. output port name for current Field value is `<stem>` posfixed with `__curr_value`: `ring_cfg__rd_ptr__curr_value`
3. input port for update its value from hardware is `<stem>` posfixed with `__next_value`: `ring_cfg__rd_ptr__next_value`
4. input port for quarlifying update is `<stem>` posfixed with `__pulse`: `ring_cfg__rd_ptr__pulse`

2.2 Description Guideline

SystemRDL defines several properties for describing Field, however, only a subset of them are interpreted by the scripts. Only properties documented in this section are allowed for Field description, others are prohibited to use.

Table 2.1: Field Properties Supported in scripts

Property	Notes	Type	Default	Dynamic
fieldwidth	Width of Field.	*longint unsighed*	1	No
reset	Reset value of Field.	*bit*	0	Yes
resetsignal	Reference to signal used as **Asynchornous reset** of the Field.	*reference*		Yes
syncresetsignal	Reference to signal used as **Synchronous Reset** of the Field.	*reference*		Yes
name	Specifies a more descriptive name (for documentation purposes).	*string*	""	Yes
desc	Describes the component's purpose. MarkDown syntax is allowed	*string*	""	Yes
sw	Software access type, one of `rw`, `r`, `w`, `rw1`, `w1`, or `na`.	*access type*	`rw`	Yes
onread	Software read side effect, one of `rclr`, `rset`, or `na`.	*onreadtype*	`na`	Yes
onwrite	Software write side effect, one of `woset`, `woclr`, `wot`, `wzs`, `wzc`, `wzt`, or `na`.	*onwritetype*	`na`	Yes
swmod	Populate an output signal which is asserted when field is modified by software (written or read with a set or clear side effect).	*boolean*	false	Yes
swacc	Populate an output signal which is asserted when field is read.	*boolean*	false	Yes
singlepulse	Populate an output signal which is asserted for one cycle when field is written 1.	*boolean*	false	Yes
hw	Hardware access type, one of `rw`, or `r`	*access type*	`r`	No
hwclr	Hardware clear. Field is cleared upon assertion on hardware signal in bitwise mode.	*boolean*	false	Yes
hwset	Hardware set. Field is set upon assertion on hardware signal in bitwise mode.	*boolean*	false	Yes
precedence	One of `hw` or `sw`, controls whether precedence is granted to hardware (`hw`) or software (`sw`) when contention occurs.	*precedencetype*	`sw`	Yes

resetsignal specifies signal used as **Asynchronous reset** for the Field. By default, `rst_n` is used as asynchronous reset signal. When set to a reference of signal, an input port is populated for the signal and the field's asynchronous reset will be connected to the signal.

syncresetsignal is a *User-defined* property that specifies signal (or multiple signals) used as **Synchronous Reset** for the Field. By default, a Field doesn't have Synchronous reset. User can set **syncresetsignal** property more than once to specify multiple synchronous reset signals. Each synchronous reset signal **must** be active high and one clock cycle wide. Reset value of synchronous reset is the same as that of asynchronous reset.

Current value of Field (`<stem>__curr_value`) is always output to user logic. If **hw** is `rw`, two more inputs are populated (`<stem>__next_value` and `<stem>__pulse`) for updating field value from user logic. If value from hardware is expected to be continously updated into Field, user should tie `<stem>__pulse` to `1'b1`. If either **hwclr** or **hwset** is `true` (they are mutually exclusive), `field` module use `<stem>__next_value` in bitwide mode and ignores `<stem>__pulse`. Each pulse in `<stem>__next_value` will clear or set corresponding bit on Field.

2.3 Examples

3

Regsiter Description

3.1 Naming Convention

Each Register is a concatenation of Fields. No RTL module is implemented for Register. Instead, an `always_comb` block is used to concatenate Fields `curr_value` as below:

```
// ring_cfg
always_comb begin
   ring_cfg[31:0] = 32'd0;
   ring_cfg[31] = ring_cfg__ring_en__curr_value;
   ring_cfg[7:4] = ring_cfg__ring_size__curr_value[3:0];
end
```

All Fields in a Register share same register `rd_en`, `wr_en`, and `wr_data`. Scripts will connect the correct signal from address decoder to Field instances.

3.2 Description Guideline

internal, **external**, and **alias** are all supported. Refer to [1] section 10.3, 10.4, 10.5 for detailed descriptions.

Table 3.1: Register Properties Supported in scripts

Property	Notes	Type	Default	Dynamic
regwidth	Width of Register.	*longint unsighed*	32	No
errextbus	The associated regsiter has error input	*boolean*	false	No
shared	Defines a register as being shared in different address maps.	*boolean*	false	No

3.3 Examples

4 User-defined Property

4.1 syncresetsignal

```
property syncresetsignal {
  component = field|reg|regfile|addrmap;
  type = ref;
}
```

5 BIBLIOGRAPHY

[1] Accellera. *SystemRDL 2.0 Register Description Language*. January 2018.