Specification RFG V1.1

Tobias Markus

24.02.2015

Contents

1 RFG Hardware Generation

1.1 Overview

1.2 Register

Registers are the smallest addressable elements in a registerfile. A register can consist of one ore more fields with different widths and attributes. These variants of generated hardware depending on the attributes is described in this subsection.

```
## A register with several fields and different attributes
register test {
    field field_1 {
        width 32
        reset 32'h0
        software ro
        hardware wo
    }
    field field_2 {
        width 16
        reset 16'h0
        software rw
        hardware rw
    }
    field field_3 {
        width 16
        reset 16'h0
        software rw
    }

}
```

The size of a register can be set with an attribute in the registerfile (register_size). The default size of a register is 64 bit.

Each regiter in the following section is generated with the Gnerator script below:

```
package require osys::rfg 1.0.0
package require osys::generator 1.0.0

readRF [lindex $argv 0]

generator verilog {
    destinationPath "verilog/"
    options {
        reset sync
    }
}
```

1.2.1 hardware/software Permissions

The most common and important Attribute are the Permissions. A register has a software and a hardware interface. Each Interface can have read and/or write permissions defined with the attributes shown in the table below:

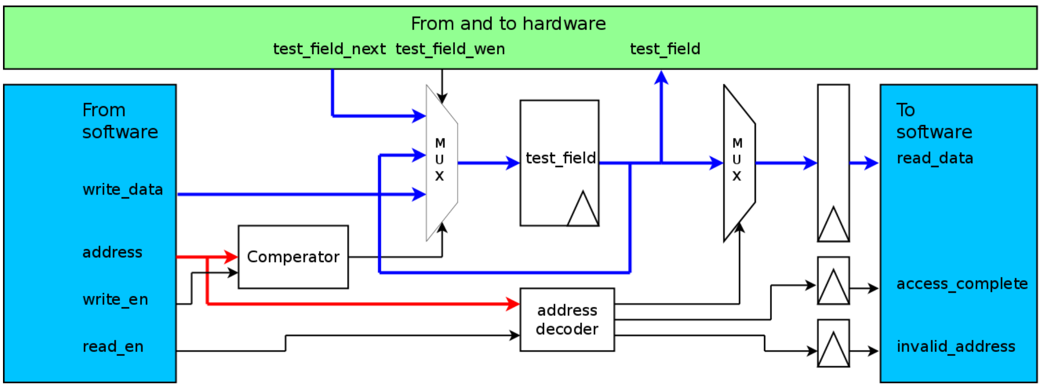
attribute name	description
ro	read only
wo	write only
rw	read and write

In this example we describe a register which has one 32 bit field with a reset value of zero and hardware read and write and software read and write permissions.

RFG Description:

```
registerFile reg_hrw_srw_hwen {
    register test {
        field test_field {
            width 32
            reset 32'h0
            software rw
            hardware rw
        }
    }
}
```

Block Diagramm:



Generated verilog from RFG description:

```
module reg_hrw_srw_nhwen
(
    //Software Interface
    input wire res_n,
    input wire clk,
    input wire[3:3] address,
    output reg[31:0] read_data,
    output reg invalid_address,
    output reg access_complete,
    input wire read_en,
    input wire write_en,
    input wire[31:0] write_data,
    // Hardware Interface
    input wire[31:0] test_test_field_next,
    input wire test_test_field_wen,
    output reg[31:0] test_test_field
);

    /* register test */
    always @(posedge clk)
    begin
        if (!res_n)
        begin
            test_test_field <= 32'h0;
        end
        else
        begin
            if((address[3:3]== 0) && write_en)
            begin
                test_test_field <= write_data[31:0];
            end
            else if(test_test_field_wen)
            begin
                test_test_field <= test_test_field_next;
            end
        end
    end

    always @(posedge clk)
    begin
        if (!res_n)
        begin
```

```
43              invalid_address <= 1'b0;
44              access_complete <= 1'b0;
45          end
46          else
47          begin
48
49              casex(address[3:3])
50                  1'h0:
51                  begin
52                      read_data[31:0] <= test_test_field;
53                      invalid_address <= 1'b0;
54                      access_complete <= write_en || read_en;
55                  end
56                  default:
57                  begin
58                      invalid_address <= read_en || write_en;
59                      access_complete <= read_en || write_en;
60                  end
61              endcase
62          end
63      end
64  endmodule
```

Depending on the permission attributes the verilog output is slightly different.

The always block from line 21 to line 38, represents the software write and hardware write functionallaty to one field. If the field have no software write permissions line 29 to line 32 are not generated. If the field has no hardware write permission line 33 to line 35 are not generated. If the field has neither a software write nor a hardware write only the reset logic is generated. If the field also does not have a reset attribute the always block is not generated. These descriptions without any hardware or software permissions are used to define reserved fields in a register.

The hardware read is generated with the output reg on line 16 if there is no hardware read permission this signal is generated as internal reg.

In the second always block the address decoder for the software read is generated. Depending on the read permission line 51 is generated or not.

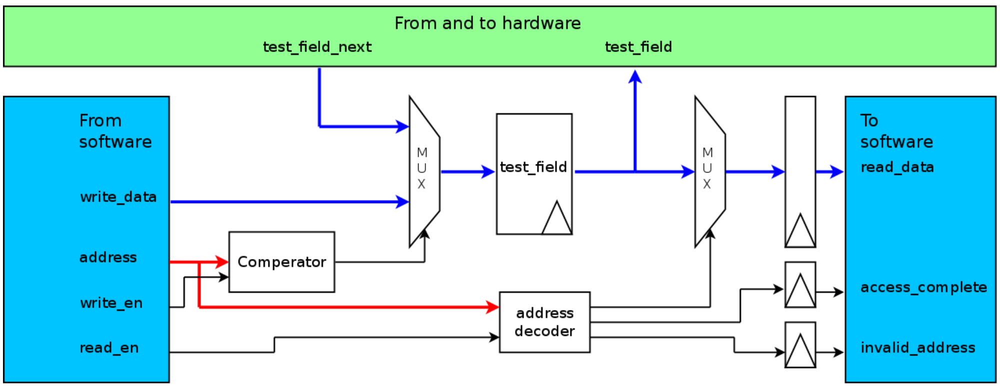
1.2.2 no_hardware_wen

With the no_hardware_wen attribute the hardware generator will not generate the hardware write enable signal on the register hardware interface. Attention when you write something with the software the hardware will rewrite the register in the next clock cycle.

RFG Description:

```
registerFile reg_hrw_srw_nhwen {
    register test {
        field test_field {
            width 32
            reset 32'h0
            software rw
            hardware {
                rw
                no_hardware_wen
            }
        }
    }
}
```

Block Diagramm:



Generated verilog from RFG description:

```
module reg_hrw_srw_nhwen
(
    // Software Interface
    input wire res_n,
    input wire clk,
    input wire[3:3] address,
    output reg[31:0] read_data,
    output reg invalid_address,
    output reg access_complete,
    input wire read_en,
    input wire write_en,
    input wire[31:0] write_data,
    // Hardware Interface
    input wire[31:0] test_test_field_next,
    output reg[31:0] test_test_field

);

    /* register test */
    always @(posedge clk)
    begin
        if (!res_n)
        begin
            test_test_field <= 32'h0;
        end
        else
        begin

            if((address[3:3]== 0) && write_en)
            begin
                test_test_field <= write_data[31:0];
            end
            else
            begin
                test_test_field <= test_test_field_next;
            end
        end
    end

    always @(posedge clk)
    begin
        if (!res_n)
```

```
43            begin
44                invalid_address <= 1'b0;
45                access_complete <= 1'b0;
46            end
47            else
48            begin
49                casex(address[3:3])
50                    1'h0:
51                    begin
52                        read_data[31:0] <= test_test_field;
53                        invalid_address <= 1'b0;
54                        access_complete <= write_en || read_en;
55                    end
56                    default:
57                    begin
58                        invalid_address <= read_en || write_en;
59                        access_complete <= read_en || write_en;
60                    end
61                endcase
62            end
63        end
64    endmodule
```

The difference in this Verilog output can be observed in line 33. Now there is no hardware write enable signal the register is written on each clock cycle. Keep this in mind if you write it via Software. The Hardware has then one cycle to react to it and then to rewrite the field.

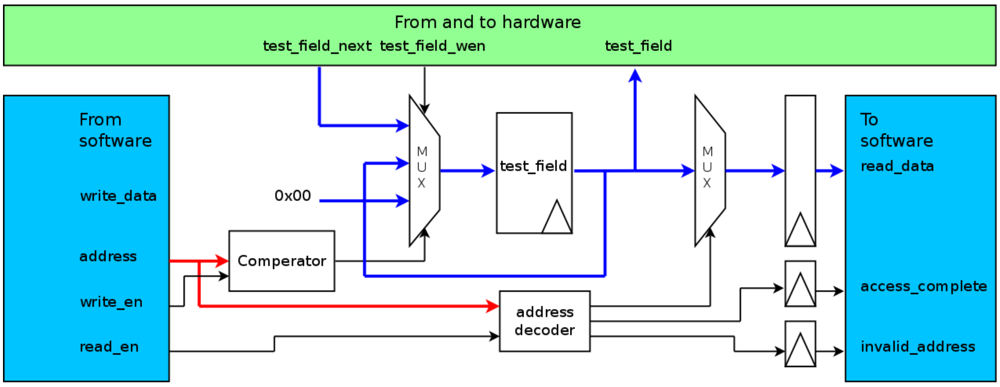
1.2.3 software_write_clear

With the software_write_clear attribute the field is cleared on a software write operation.

RFG Descpription:

```
registerFile reg_hrw_srw_swrite_clear {
    register test {
        field test_field {
            width 32
            reset 32'h0
            software {
                rw
                software_write_clear
            }
            hardware rw
        }
    }
}
```

Block Diagramm:



Generated verilog from RFG description:

```
module reg_hrw_srw_swrite_clear
(
    input wire res_n,
    input wire clk,
    // Software Interface
    input wire[3:3] address,
    output reg[31:0] read_data,
    output reg invalid_address,
    output reg access_complete,
    input wire read_en,
    input wire write_en,
    input wire[31:0] write_data,
    // Hardware Interface
    input wire[31:0] test_test_field_next,
    input wire test_test_field_wen,
    output reg[31:0] test_test_field
);

    /* register test */
    always @(posedge clk)
    begin
        if (!res_n)
        begin
            test_test_field <= 32'h0;
        end
        else
        begin
            if((address[3:3]== 0) && write_en)
            begin
                test_test_field <= 32'h0;
            end
            else if(test_test_field_wen)
            begin
                test_test_field <= test_test_field_next;
            end
        end
    end

    always @(posedge clk)
    begin
        if (!res_n)
        begin
```

```
43            invalid_address <= 1'b0;
44            access_complete <= 1'b0;
45        end
46        else
47        begin
48            casex(address[3:3])
49                1'h0:
50                begin
51                    read_data[31:0] <= test_test_field;
52                    invalid_address <= 1'b0;
53                    access_complete <= write_en || read_en;
54                end
55                default:
56                begin
57                    invalid_address <= read_en || write_en;
58                    access_complete <= read_en || write_en;
59                end
60            endcase
61        end
62    end
63 endmodule
```

In line 30 we can see that now the register is cleared when the register is written from the software.

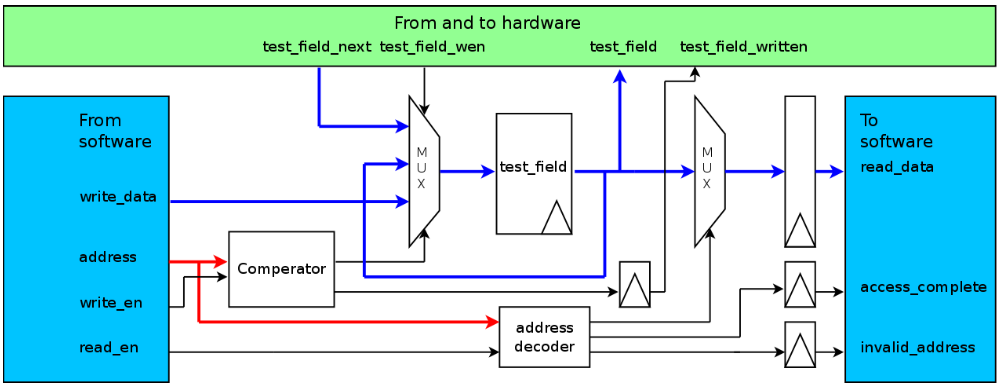
1.2.4 software_written

With the software_written signal an additional hardware output is generated which is high when the software writes the register and depending on its value also when the register is resetted. Otherwise the software_written signal is low.

RFG Description:

```
registerFile reg_hrw_srw_swritten {
    register test {
        field test_field {
            width 32
            reset 32'h0
            software rw
            hardware {
                rw
                software_written
            }
        }
    }
}
```

Block Diagramm:



Generated verilog from RFG description:

```
module reg_hrw_srw_swritten
(
    input wire res_n,
    input wire clk,
    // Software Interface
    input wire[3:3] address,
    output reg[31:0] read_data,
    output reg invalid_address,
    output reg access_complete,
    input wire read_en,
    input wire write_en,
    input wire[31:0] write_data,
    // Hardware Interface
    input wire[31:0] test_test_field_next,
    input wire test_test_field_wen,
    output reg[31:0] test_test_field,
    output reg test_test_field_written

);

    /* register test */
    always @(posedge clk)
    begin
        if (!res_n)
        begin
            test_test_field <= 32'h0;
            test_test_field_written <= 1'b0;
        end
        else
        begin

            if((address[3:3]== 0) && write_en)
            begin
                test_test_field <= write_data[31:0];
                test_test_field_written <= 1'b1;
            end
            else if(test_test_field_wen)
            begin
                test_test_field <= test_test_field_next;
                test_test_field_written <= 1'b0;
            end
            else
```

```
43                begin
44                    test_test_field_written <= 1'b0;
45                end
46
47            end
48        end
49
50        always @(posedge clk)
51        begin
52            if (!res_n)
53            begin
54                invalid_address <= 1'b0;
55                access_complete <= 1'b0;
56            end
57            else
58            begin
59
60                casex(address[3:3])
61                    1'h0:
62                    begin
63                        read_data[31:0] <= test_test_field;
64                        invalid_address <= 1'b0;
65                        access_complete <= write_en || read_en;
66                    end
67                    default:
68                    begin
69                        invalid_address <= read_en || write_en;
70                        access_complete <= read_en || write_en;
71                    end
72                endcase
73            end
74        end
75    endmodule
```

In this verilog output an additional hardware output signal is added (line 17). It is set when the software interface writes the field, line 32 to 36. It is reset on every cycle in which the software interface does not do a write operation. In this example the software_written attribute is configured to output a zero, when the register is resetted. It can also be configured to output a one.

1.2.5 sticky

1.2.6 software_write_xor

1.2.7 hardware_clear

1.2.8 counter

1.2.9 rreinit_source, rreinit

1.3 RamBlock

1.4 external/internal RegisterFiles