Specification RFG V1.1

Tobias Markus

24.02.2015

Contents

1   Introduction

2   RFG

2.1   RFG Language

2.2   RFG API

3   Generator API

3.1   User perspective

3.2   Developer perspective

4   RFG Verilog Generator

4.1   Overview

4.2 Register

Registers are the smallest addressable units in a registerFile. A register consists of one ore more fields with different widths and attributes. Depending on this variants hardware will be generated.

The different attributes and the resulting hardware is given 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.

4.2.1 hardware/software Permissions

The most common and important attributes are the permissions. A register
has a software and a hardware interface. Each Interface can have read and/
or write permissions on a field in a register, 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
        }
    }

    ## Additional registerFile Objects ...

}
```
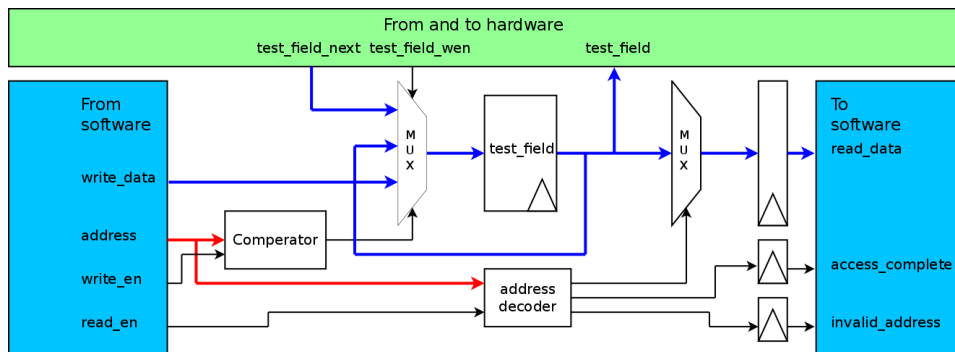
Block Diagramm:

Generated verilog from RFG description:

```verilog
1   module reg_hrw_srw_nhwen
2   (
3       //Software Interface
4       input wire res_n ,
5       input wire clk ,
6       input wire[3:3] address ,
7       output reg[31:0] read_data ,
8       output reg invalid_address ,
9       output reg access_complete ,
10      input wire read_en ,
11      input wire write_en ,
12      input wire[31:0] write_data ,
13      // Hardware Interface
14      input wire[31:0] test_test_field_next ,
15      input wire test_test_field_wen ,
16      output reg[31:0] test_test_field
17
18      // Additional Signals ...
19
20  );
21
22      /* register test */
23      always @(posedge clk)
24      begin
25          if (!res_n)
26          begin
27              test_test_field <= 32'h0;
28          end
29          else
30          begin
31              if((address[3:3]== 0) && write_en)
32              begin
33                  test_test_field <= write_data[31:0];
34              end
35              else if(test_test_field_wen)
36              begin
37                  test_test_field <= test_test_field_next;
38              end
39          end
40      end
41
42      // Additional always registerFile Object blocks...
```

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

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

The always block part from line 30 to line 39, represents the software write and hardware write functionality to one field. If the field has no software write permissions line 31 to line 34 are not generated. If the field has no hardware write permission line 35 to line 38 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 57 is generated or not.

4.2.2 no_wen

With the no_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_wen
            }
        }
    }

    ## Additional registerFile Objects ...

}
```
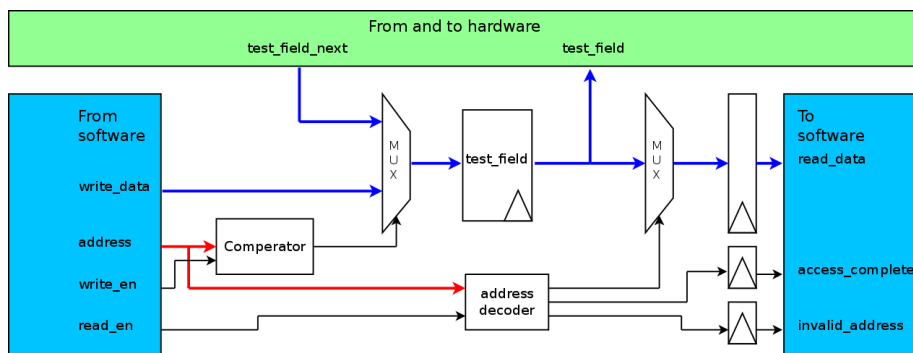
Block Diagramm:

Generated verilog from RFG description:

```verilog
1   module reg_hrw_srw_nhwen
2   (
3       // Software Interface
4       input wire res_n,
5       input wire clk,
6       input wire[3:3] address,
7       output reg[31:0] read_data,
8       output reg invalid_address,
9       output reg access_complete,
10      input wire read_en,
11      input wire write_en,
12      input wire[31:0] write_data,
13      // Hardware Interface
14      input wire[31:0] test_test_field_next,
15      output reg[31:0] test_test_field
16
17      // Additional Signals ...
18
19  );
20
21      /* register test */
22      always @(posedge clk)
23      begin
24          if (!res_n)
25          begin
26              test_test_field <= 32'h0;
27          end
28          else
29          begin
30
31              if((address[3:3]== 0) && write_en)
32              begin
33                  test_test_field <= write_data[31:0];
34              end
35              else
36              begin
37                  test_test_field <= test_test_field_next;
38              end
39          end
40      end
41
42      // Additional always registerFile Object blocks...
```

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

The difference in this verilog output can be observed in line 37. 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 the software interface. The hardware has then one cycle to react to it and to rewrite the field.

4.2.3 write_clear

With the 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
                 write_clear
            }
            hardware rw
        }
    }

    ## Additional registerFile Objects ...

}
```
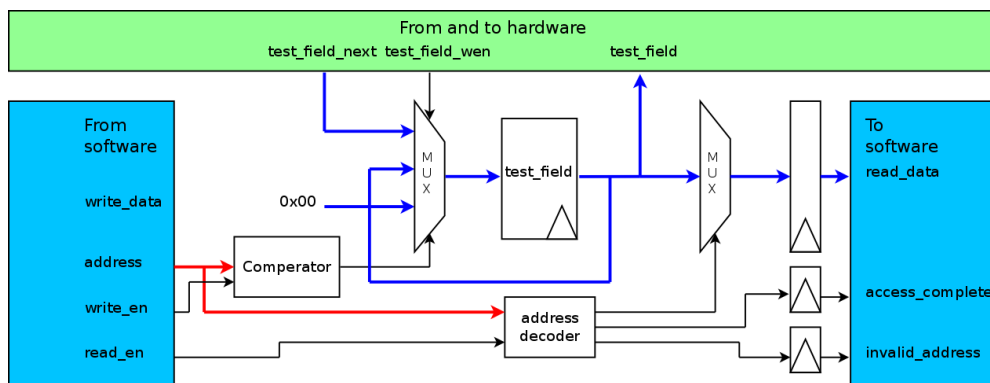
Block Diagramm:

Generated verilog from RFG description:

```verilog
1  module reg_hrw_srw_swrite_clear
2  (
3      input wire res_n,
4      input wire clk,
5      // Software Interface
6      input wire[3:3] address,
7      output reg[31:0] read_data,
8      output reg invalid_address,
9      output reg access_complete,
10     input wire read_en,
11     input wire write_en,
12     input wire[31:0] write_data,
13     // Hardware Interface
14     input wire[31:0] test_test_field_next,
15     input wire test_test_field_wen,
16     output reg[31:0] test_test_field
17
18     // Additional Signals ...
19
20 );
21
22     /* register test */
23     always @(posedge clk)
24     begin
25         if (!res_n)
26         begin
27             test_test_field <= 32'h0;
28         end
29         else
30         begin
31             if((address[3:3]== 0) && write_en)
32             begin
33                 test_test_field <= 32'h0;
34             end
35             else if(test_test_field_wen)
36             begin
37                 test_test_field <= test_test_field_next;
38             end
39         end
40     end
41
42     // Additional always registerFile Object blocks...
```

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

In line 33 we can see that now the register is cleared when the register is written from the software.

4.2.4   software_written

With the software_written signal an additional hardware output is generated which is high when the software writes the register. 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
            }
        }
    }

    ## Additional registerFile Objects ...

}
```
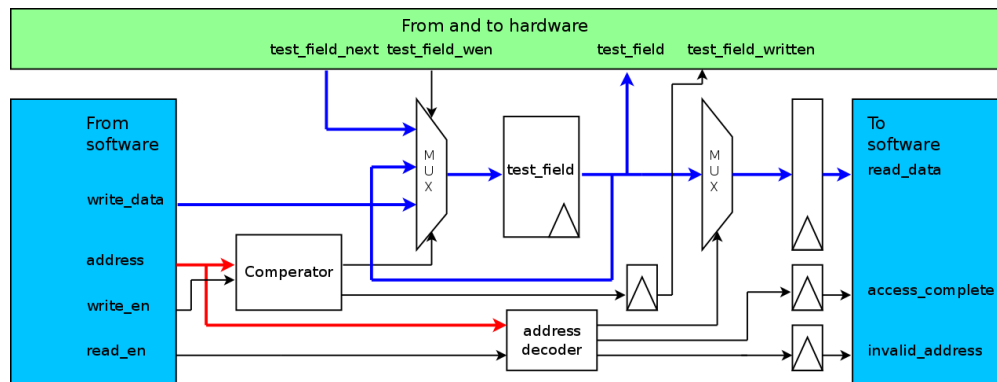
Block Diagramm:

Generated verilog from RFG description:

```verilog
1   module reg_hrw_srw_swritten
2   (
3       input wire res_n,
4       input wire clk,
5       // Software Interface
6       input wire [3:3] address,
7       output reg [31:0] read_data,
8       output reg invalid_address,
9       output reg access_complete,
10      input wire read_en,
11      input wire write_en,
12      input wire [31:0] write_data,
13      // Hardware Interface
14      input wire [31:0] test_test_field_next,
15      input wire test_test_field_wen,
16      output reg [31:0] test_test_field,
17      output reg test_test_field_written
18
19      // Additional Signals ...
20
21  );
22
23      /* register test */
24      always @(posedge clk)
25      begin
26          if (!res_n)
27          begin
28              test_test_field <= 32'h0;
29              test_test_field_written <= 1'b0;
30          end
31          else
32          begin
33
34              if ((address[3:3]== 0) && write_en)
35              begin
36                  test_test_field <= write_data[31:0];
37                  test_test_field_written <= 1'b1;
38              end
39              else if (test_test_field_wen)
40              begin
41                  test_test_field <= test_test_field_next;
42                  test_test_field_written <= 1'b0;
```

```verilog
43              end
44              else
45              begin
46                  test_test_field_written <= 1'b0;
47              end
48
49          end
50      end
51
52      // Additional always registerFile Object blocks...
53
54      always @(posedge clk)
55      begin
56          if (!res_n)
57          begin
58              invalid_address <= 1'b0;
59              access_complete <= 1'b0;
60          end
61          else
62          begin
63
64              casex(address[3:3])
65                  1'h0:
66                  begin
67                      read_data[31:0] <= test_test_field;
68                      invalid_address <= 1'b0;
69                      access_complete <= write_en || read_en;
70                  end
71
72                  // Additional addresses...
73
74                  default:
75                  begin
76                      invalid_address <= read_en || write_en;
77                      access_complete <= read_en || write_en;
78                  end
79              endcase
80          end
81      end
82  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 34 to 38. It is reset on every cycle in which the software interface does not do a write

16

operation.

4.2.5 changed

The changed attribute behaves like the software_written attribute described before. The only difference is that the changed signal is also set to high when the register is changed from a reset.

RFG Description:

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

    ## Additional registerFile Objects ...

}
```

Block Diagramm:
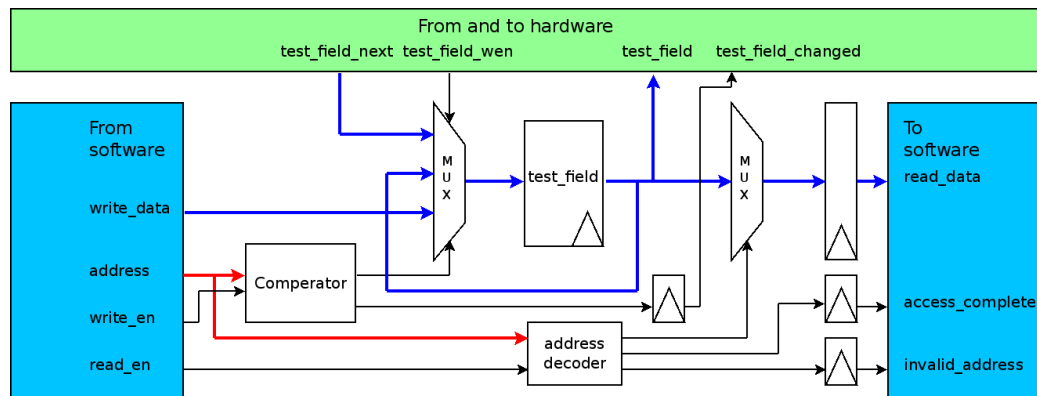
4.2.6 sticky

With the sticky feature a field is generated in which the bits in the field can only be set from the hardware. The field can only be reseted from the software interface or with a reset.

RFG Description:

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

    ## Additional registerFile Objects ...

}
```

Block Diagramm:

19

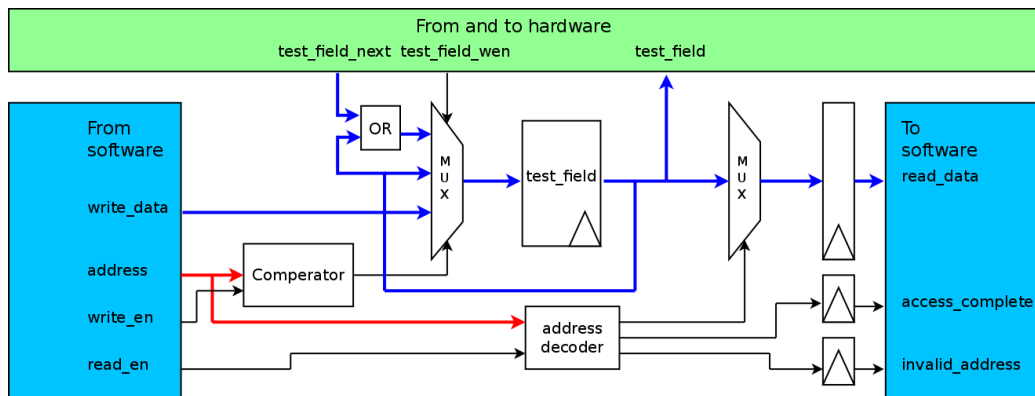Generated verilog from RFG description:

```verilog
1  module reg_hrw_srw_sticky
2  (
3      input wire res_n,
4      input wire clk,
5      // Software Interface
6      input wire[3:3] address,
7      output reg[31:0] read_data,
8      output reg invalid_address,
9      output reg access_complete,
10     input wire read_en,
11     input wire write_en,
12     input wire[31:0] write_data,
13     // Hardware Interface
14     input wire[31:0] test_test_field_next,
15     input wire test_test_field_wen,
16     output reg[31:0] test_test_field
17
18     // Additional Signals ...
19
20 );
21
22     /* register test */
23     always @(posedge clk)
24     begin
25         if (!res_n)
26         begin
27             test_test_field <= 32'h0;
28         end
29         else
30         begin
31
32             if((address[3:3]== 0) && write_en)
33             begin
34                 test_test_field <= write_data[31:0];
35             end
36             else if(test_test_field_wen)
37             begin
38                 test_test_field <= test_test_field_next |
                        test_test_field;
39             end
40         end
41     end
```

```
42
43        // Additional always registerFile Object blocks...
44
45        always @(posedge clk)
46        begin
47            if (!res_n)
48            begin
49                invalid_address <= 1'b0;
50                access_complete <= 1'b0;
51            end
52            else
53            begin
54                casex(address[3:3])
55                    1'h0:
56                    begin
57                        read_data[31:0] <= test_test_field;
58                        invalid_address <= 1'b0;
59                        access_complete <= write_en || read_en;
60                    end
61
62                    // Additional addresses...
63
64                    default:
65                    begin
66                        invalid_address <= read_en || write_en;
67                        access_complete <= read_en || write_en;
68                    end
69                endcase
70            end
71        end
72 endmodule
```

The difference in the verilog output with the sticky attribute is that the new hardware value is ored on write with the register value itself line 38.

4.2.7   software_write_xor

The software_write_xor attributes writes on a software write the new value xored with the register value.

RFG Description:

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

    ## Additional registerFile Objects ...

}
```

Block Diagramm:
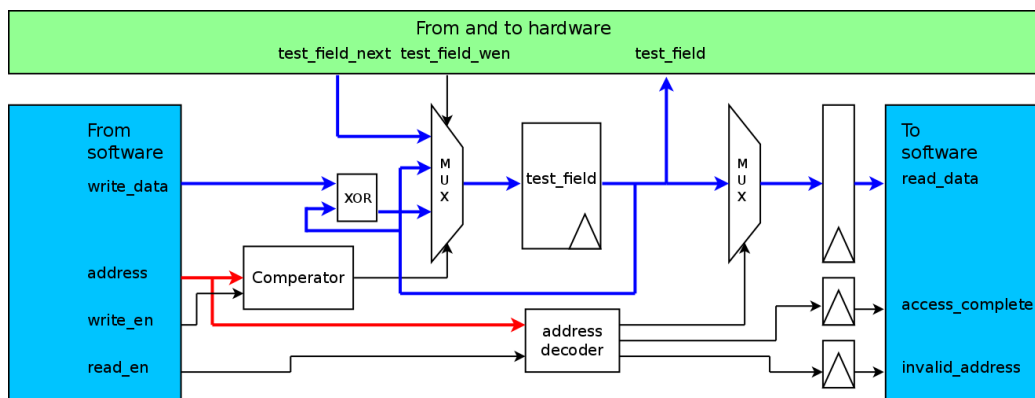
Generated verilog from RFG description:

```verilog
1  module reg_hrw_srw_swrite_xor
2  (
3      input wire res_n,
4      input wire clk,
5      // Software Interface
6      input wire[3:3] address,
7      output reg[31:0] read_data,
8      output reg invalid_address,
9      output reg access_complete,
10     input wire read_en,
11     input wire write_en,
12     input wire[31:0] write_data,
13     // Hardware Interface
14     input wire[31:0] test_test_field_next,
15     input wire test_test_field_wen,
16     output reg[31:0] test_test_field
17
18     // Additional Signals ...
19
20 );
21
22     /* register test */
23     always @(posedge clk) `endif
24     begin
25         if (!res_n)
26         begin
27             test_test_field <= 32'h0;
28         end
29         else
30         begin
31
32             if((address[3:3]== 0) && write_en)
33             begin
34                 test_test_field <= (write_data[31:0] ^
                        test_test_field);
35             end
36             else if(test_test_field_wen)
37             begin
38                 test_test_field <= test_test_field_next;
39             end
40         end
41     end
```

```verilog
42
43        // Additional always registerFile Object blocks...
44
45        always @(posedge clk)
46        begin
47            if (!res_n)
48            begin
49                invalid_address <= 1'b0;
50                access_complete <= 1'b0;
51            end
52            else
53            begin
54
55                casex(address[3:3])
56                    1'h0:
57                    begin
58                        read_data[31:0] <= test_test_field;
59                        invalid_address <= 1'b0;
60                        access_complete <= write_en || read_en;
61                    end
62
63                    // Additional addresses...
64
65                    default:
66                    begin
67                        invalid_address <= read_en || write_en;
68                        access_complete <= read_en || write_en;
69                    end
70                endcase
71            end
72        end
73    endmodule
```

The software_write_xor attribute does also just do small change. When the register is written form the software interface it gets xored with itself line 32.

4.2.8 clear

The hardware_clear attribute adds an additional signal to the hardware interface. This signal clears the register when the hardware_clear signal is asserted.

RFG Description:

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

    ## Additional registerFile Objects ...

}
```

Block Diagramm:
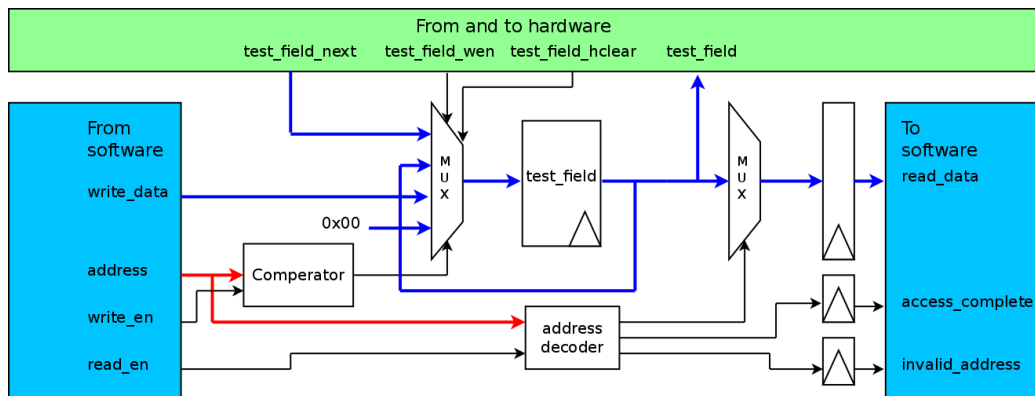
Generated verilog from RFG description:

```verilog
 1  module reg_hrw_srw_hclear
 2  (
 3      input wire res_n,
 4      input wire clk,
 5      // Software Interface
 6      input wire[3:3] address,
 7      output reg[31:0] read_data,
 8      output reg invalid_address,
 9      output reg access_complete,
10      input wire read_en,
11      input wire write_en,
12      input wire[31:0] write_data,
13      // Hardware Interface
14      input wire[31:0] test_test_field_next,
15      input wire test_test_field_wen,
16      output reg[31:0] test_test_field,
17      input wire test_test_field_clear
18
19      // Additional Signals ...
20
21  );
22
23      /* register test */
24      always @(posedge clk)
25      begin
26          if (!res_n)
27          begin
28              test_test_field <= 32'h0;
29          end
30          else
31          begin
32              if((address[3:3]== 0) && write_en)
33              begin
34                  test_test_field <= write_data[31:0];
35              end
36              else if(test_test_field_clear)
37              begin
38                  test_test_field <= 32'h0;
39              end
40              else if(test_test_wen)
41              begin
42                  test_test_field <= test_test_field_next;
```

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

4.2.9 trigger

4.2.10 counter

The counter attribute transforms the internal register into a counter with count up signal for the hardware interface.

RFG Description:

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

    ## Additional registerFile Objects ...

}
```

Block Diagramm:
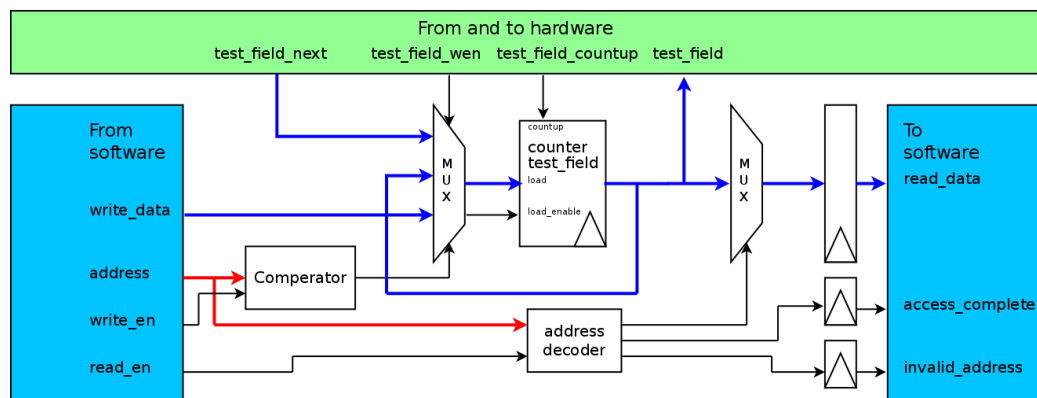
Generated verilog from RFG description:

```verilog
1   module reg_hrw_srw_counter
2   (
3       input wire clk,
4       input wire res_n,
5       // Software Interface
6       input wire[3:3] address,
7       output reg[31:0] read_data,
8       output reg invalid_address,
9       output reg access_complete,
10      input wire read_en,
11      input wire write_en,
12      input wire[31:0] write_data,
13      // Hardware Interface
14      input wire[31:0] test_test_field_next,
15      output wire[31:0] test_test_field,
16      input wire test_test_field_wen,
17      input wire test_test_field_countup
18  );
19
20      reg test_test_field_load_enable;
21      reg[31:0] test_test_field_load_value;
22
23      counter48 #(
24          .DATASIZE(32)
25      ) test_test_field_I (
26          .clk(clk),
27          .res_n(res_n),
28          .increment(test_test_field_countup),
29          .load(test_test_field_load_value),
30          .load_enable(test_test_field_load_enable),
31          .value(test_test_field)
32      );
33
34      /* register test */
35      always @(posedge clk)
36      begin
37          if (!res_n)
38          begin
39              test_test_field_load_enable <= 1'b0;
40          end
41          else
42          begin
```

```verilog
43
44                 if((address[3:3]== 0) && write_en)
45                 begin
46                     test_test_field_load_enable <= 1'b1;
47                     test_test_field_load_value <= write_data
                            [31:0];
48                 end
49                 else if(test_test_field_wen)
50                 begin
51                     test_test_field_load_value <=
                            test_test_field_next;
52                     test_test_field_load_enable <= 1'b1;
53                 end
54                 else
55                 begin
56                     test_test_field_load_enable <= 1'b0;
57                     test_test_field_load_value <= 32'b0;
58                 end
59            end
60        end
61
62        always @(posedge clk)
63        begin
64            if (!res_n)
65            begin
66                invalid_address <= 1'b0;
67                access_complete <= 1'b0;
68            end
69            else
70            begin
71
72                casex(address[3:3])
73                    1'h0:
74                    begin
75                        read_data[31:0] <= test_test_field;
76                        invalid_address <= 1'b0;
77                        access_complete <= write_en || read_en;
78                    end
79                    default:
80                    begin
81                        invalid_address <= read_en || write_en;
82                        access_complete <= read_en || write_en;
83                    end
```

```verilog
84                  endcase
85             end
86         end
87 endmodule
```

4.2.11 rreinit_source, rreinit

With the rreinit_source and rreinit combination a register with the rreinit_source attribute can be generated which resets a counter field marked with the rreinit attribute.

RFG Description:

```
registerFile reg_hrw_srw_rreinit_source {

    register counter_rreinit {
        hardware {
            rreinit_source
        }
    }

    register example {
        field test_field {
            width 32
            reset 32'h0
            software rw
            hardware {
                rw
                counter
                rreinit
            }
        }
    }
}
```

Generated verilog from RFG description:

```verilog
1  module reg_hrw_srw_rreinit_source
2  (
3      input wire res_n ,
4      input wire clk ,
5      // Software Interface
6      input wire [3:3] address ,
7      output reg [31:0] read_data ,
8      output reg invalid_address ,
9      output reg access_complete ,
10     input wire read_en ,
11     input wire write_en ,
12     input wire [31:0] write_data ,
13     // Hardware Interface
14     input wire [31:0] example_test_field_next ,
15     output wire [31:0] example_test_field ,
16     input wire example_test_field_wen ,
17     input wire example_test_field_countup
18 );
19
20     reg rreinit ;
21     reg example_test_field_load_enable ;
22     reg [31:0] example_test_field_load_value ;
23
24     counter48 #(
25         .DATASIZE(32)
26     ) example_test_field_I (
27         .clk(clk) ,
28         .res_n(res_n) ,
29         .increment(example_test_field_countup) ,
30         .load(example_test_field_load_value) ,
31         .load_enable(rreinit ||
32             example_test_field_load_enable) ,
32         .value(example_test_field)
33     );
34
35     /* register counter_rreinit */
36     always @(posedge clk)
37     begin
38         if (!res_n)
39         begin
40             rreinit <= 1'b0;
41         end
```

```verilog
42            else
43            begin
44
45                if((address[3:3]==  0) && write_en)
46                begin
47                    rreinit <= 1'b1;
48                end
49                else
50                begin
51                    rreinit <= 1'b0;
52                end
53            end
54        end
55
56        /* register example */
57        always @(posedge clk)
58        begin
59            if (!res_n)
60            begin
61                example_test_field_load_enable <= 1'b0;
62            end
63            else
64            begin
65
66                if((address[3:3]==  1) && write_en)
67                begin
68                    example_test_field_load_enable <= 1'b1;
69                    example_test_field_load_value <= write_data
                        [31:0];
70                end
71                else if(example_test_field_wen)
72                begin
73                    example_test_field_load_value <=
                        example_test_field_next;
74                    example_test_field_load_enable <= 1'b1;
75                end
76                else
77                begin
78                    example_test_field_load_enable <= 1'b0;
79                    example_test_field_load_value <= 32'b0;
80                end
81            end
82        end
```

```verilog
83
84        always @(posedge clk)
85        begin
86            if (!res_n)
87            begin
88                invalid_address <= 1'b0;
89                access_complete <= 1'b0;
90            end
91            else
92            begin
93
94                casex(address[3:3])
95                    1'h1:
96                    begin
97                        read_data[31:0] <= example_test_field;
98                        invalid_address <= 1'b0;
99                        access_complete <= write_en || read_en;
100                   end
101                   default:
102                   begin
103                       invalid_address <= read_en || write_en;
104                       access_complete <= read_en || write_en;
105                   end
106               endcase
107           end
108       end
109   endmodule
```

4.2.12 edge_trigger

4.3 RamBlock

A RamBlock is a construct which implements an addressspace as RAM inside the hardware. In different to registers, the hardware interface has now also address, data, and control lines. Depending on the read/write Permissions different RAMs are used. See the table below. 1w_1r_1c means a Ram with one write, one read interface. and 2rw_1c means a dual port Ram.

permissions	none	ro	wo	rw	hardware
none	none	none	none	1w_1r_1c	
ro	none	none	1w_1r_1c	2rw_1c	
wo	none	1w_1r_1c	none	2rw_1c	
rw	1w_1r_1c	2rw_1c	2rw_1c	2rw_1c	
software					

RFG Description:

```
registerFile RamBlock {
    register test {
        field test_field {
            width 32
            hardware rw
            software rw
        }
    }
    ramBlock test_ram {
        depth 128
        width 32
        hardware rw
        software rw
    }
}
```

Generated verilog from RFG description:

```verilog
 1  module RamBlock
 2  (
 3      input wire clk,
 4      input wire res_n,
 5      // Software Interface
 6      input wire[10:3] address,
 7      output reg[31:0] read_data,
 8      output reg invalid_address,
 9      output reg access_complete,
10      input wire read_en,
11      input wire write_en,
12      input wire[31:0] write_data,
13      // Hardware Interface
14      input wire[31:0] test_test_field_next,
15      output reg[31:0] test_test_field,
16      input wire[6:0] test_ram_addr,
17      input wire test_ram_ren,
18      output wire[31:0] test_ram_rdata,
19      input wire test_ram_wen,
20      input wire[31:0] test_ram_wdata
21  );
22
23      reg[6:0] test_ram_rf_addr;
24      reg test_ram_rf_ren;
25      wire[31:0] test_ram_rf_rdata;
26      reg test_ram_rf_wen;
27      reg[31:0] test_ram_rf_wdata;
28      reg read_en_dly0;
29      reg read_en_dly1;
30      reg read_en_dly2;
31
32      ram_2rw_1c #(
33          .DATASIZE(32),
34          .ADDRSIZE(7),
35          .PIPELINED(0)
36      ) test_ram (
37          .clk(clk),
38          .wen_a(test_ram_rf_wen),
39          .ren_a(test_ram_rf_ren),
40          .addr_a(test_ram_rf_addr),
41          .wdata_a(test_ram_rf_wdata),
42          .rdata_a(test_ram_rf_rdata),
```

```verilog
43              .wen_b(test_ram_wen),
44              .ren_b(test_ram_ren),
45              .addr_b(test_ram_addr),
46              .wdata_b(test_ram_wdata),
47              .rdata_b(test_ram_rdata)
48          );
49
50
51          /* register test */
52          always @(posedge clk)
53          begin
54              if (!res_n)
55              begin
56                  test_test_field <= 0;
57              end
58              else
59              begin
60
61                  if((address[10:3]== 0) && write_en)
62                  begin
63                      test_test_field <= write_data[31:0];
64                  end
65                  else
66                  begin
67                      test_test_field <= test_test_field_next;
68                  end
69              end
70          end
71
72          /* RamBlock test_ram */
73          always @(posedge clk)
74          begin
75              if (!res_n)
76              begin
77                  `ifdef ASIC
78                  test_ram_rf_addr <= 7'b0;
79                  test_ram_rf_wdata  <= 32'b0;
80                  `endif
81                  test_ram_rf_wen <= 1'b0;
82                  test_ram_rf_ren <= 1'b0;
83              end
84              else
85              begin
```

```verilog
86                    if (address[10:10] == 1)
87                    begin
88                        test_ram_rf_addr <= address[9:3];
89                        test_ram_rf_wdata <= write_data[31:0];
90                        test_ram_rf_wen <= write_en;
91                        test_ram_rf_ren <= read_en;
92                    end
93            end
94        end
95
96        always @(posedge clk)
97        begin
98            if (!res_n)
99            begin
100                invalid_address <= 1'b0;
101                access_complete <= 1'b0;
102
103                read_en_dly0 <= 1'b0;
104                read_en_dly1 <= 1'b0;
105                read_en_dly2 <= 1'b0;
106            end
107            else
108            begin
109                read_en_dly0 <= read_en;
110                read_en_dly1 <= read_en_dly0;
111                read_en_dly2 <= read_en_dly1;
112
113                casex(address[10:3])
114                    8'h0:
115                    begin
116                        read_data[31:0] <= test_test_field;
117                        invalid_address <= 1'b0;
118                        access_complete <= write_en || read_en;
119                    end
120                    {1'h1,7'bxxxxxxx}:
121                    begin
122                        read_data[31:0] <= test_ram_rf_rdata;
123                        invalid_address <= 1'b0;
124                        access_complete <= write_en ||
125                            read_en_dly2;
125                    end
126                    default:
127                    begin
```

```verilog
128                     invalid_address <= read_en || write_en;
129                     access_complete <= read_en || write_en;
130                 end
131             endcase
132         end
133     end
134 endmodule
```

4.3.1   attributes RamBlock

external attribute:

With the external attribute there is just a RamBlock Interface generated
to communicate with a RamBlock outside the registerfile.

address_shift attribute:

The address_shift attributes allows to address the Ram Block with a shift
inside the registerfile address space. In example each ramblock entry each
4kB with (address_shift 12)

4.4 external/internal RegisterFiles

A registerfile can be included in another one. There are two different constructs to include the registerfile internal or external.

RFG Description:

```
registerFile RF {
    external RamBlock.rf RamBlockRF_external
    internal RamBlock.rf RamBlockRF_internal
}
```

Generated verilog from RFG description:

```verilog
1   module RF (
2       input  wire clk ,
3       input  wire res_n ,
4       input  wire[11:3] address ,
5       output reg invalid_address ,
6       output reg access_complete ,
7       input  wire read_en ,
8       output reg[31:0] read_data ,
9       input  wire write_en ,
10      input  wire[31:0] write_data ,
11      output reg[10:3] RamBlockRF_external_address ,
12      input  wire RamBlockRF_external_invalid_address ,
13      input  wire RamBlockRF_external_access_complete ,
14      output reg RamBlockRF_external_read_en ,
15      input  wire[31:0] RamBlockRF_external_read_data ,
16      output reg RamBlockRF_external_write_en ,
17      output reg[31:0] RamBlockRF_external_write_data ,
18      output wire[31:0] RamBlockRF_internal_test_test_field ,
19      input  wire[31:0]
            RamBlockRF_internal_test_test_field_next ,
20      input  wire RamBlockRF_internal_test_test_field_wen ,
21      input  wire[6:0] RamBlockRF_internal_test_ram_addr ,
22      input  wire RamBlockRF_internal_test_ram_ren ,
23      output wire[31:0] RamBlockRF_internal_test_ram_rdata ,
24      input  wire RamBlockRF_internal_test_ram_wen ,
25      input  wire[31:0] RamBlockRF_internal_test_ram_wdata
26  ) ;
27      reg[10:3] RamBlockRF_internal_address ;
28      wire RamBlockRF_internal_invalid_address ;
29      wire RamBlockRF_internal_access_complete ;
30      reg RamBlockRF_internal_read_en ;
31      wire[31:0] RamBlockRF_internal_read_data ;
32      reg RamBlockRF_internal_write_en ;
33      reg[31:0] RamBlockRF_internal_write_data ;
34
35      RamBlockRF_internal RamBlockRF_internal_I (
36          .res_n(res_n) ,
37          .clk(clk) ,
38          .address(RamBlockRF_internal_address) ,
39          .read_data(RamBlockRF_internal_read_data) ,
40          .invalid_address(
                RamBlockRF_internal_invalid_address) ,
```

```verilog
41             .access_complete(
                   RamBlockRF_internal_access_complete),
42             .read_en(RamBlockRF_internal_read_en),
43             .write_en(RamBlockRF_internal_write_en),
44             .write_data(RamBlockRF_internal_write_data),
45             .RamBlockRF_internal_test_test_field_next(
                   RamBlockRF_internal_test_test_field_next),
46             .RamBlockRF_internal_test_test_field(
                   RamBlockRF_internal_test_test_field),
47             .RamBlockRF_internal_test_test_field_wen(
                   RamBlockRF_internal_test_test_field_wen),
48             .RamBlockRF_internal_test_ram_addr(
                   RamBlockRF_internal_test_ram_addr),
49             .RamBlockRF_internal_test_ram_ren(
                   RamBlockRF_internal_test_ram_ren),
50             .RamBlockRF_internal_test_ram_rdata(
                   RamBlockRF_internal_test_ram_rdata),
51             .RamBlockRF_internal_test_ram_wen(
                   RamBlockRF_internal_test_ram_wen),
52             .RamBlockRF_internal_test_ram_wdata(
                   RamBlockRF_internal_test_ram_wdata)
53         );
54
55
56         //RegisterFile: RamBlockRF_external
57         always @(posedge clk)
58         begin
59
60             if(!res_n)
61             begin
62                 RamBlockRF_external_write_en <= 1'b0;
63                 RamBlockRF_external_read_en <= 1'b0;
64                 RamBlockRF_external_write_data <= 32'b0;
65                 RamBlockRF_external_address <= 8'b0;
66             end
67             else
68             begin
69
70                 if(address[11:11] == 1'h0)
71                 begin
72                     RamBlockRF_external_address <= address
                           [10:3];
73                 end
```

```verilog
74
75                 if((address[11:11] == 1'h0) && write_en)
76                 begin
77                     RamBlockRF_external_write_data <=
                            write_data[31:0];
78                     RamBlockRF_external_write_en <= 1'b1;
79                 end
80                 else
81                 begin
82                     RamBlockRF_external_write_en <= 1'b0;
83                 end
84
85                 if((address[11:11] == 1'h0) && read_en)
86                 begin
87                     RamBlockRF_external_read_en <= 1'b1;
88                 end
89                 else
90                 begin
91                     RamBlockRF_external_read_en <= 1'b0;
92                 end
93         end
94
95     end
96
97     //RegisterFile: RamBlockRF_internal
98     always @(posedge clk)
99     begin
100
101         if(!res_n)
102         begin
103             RamBlockRF_internal_write_en <= 1'b0;
104             RamBlockRF_internal_read_en <= 1'b0;
105             RamBlockRF_internal_write_data <= 32'b0;
106             RamBlockRF_internal_address <= 8'b0;
107         end
108         else
109         begin
110
111             if(address[11:11] == 1'h1)
112             begin
113                 RamBlockRF_internal_address <= address
                        [10:3];
114             end
```

45

```verilog
115
116                 if((address[11:11] == 1'h1) && write_en)
117                 begin
118                     RamBlockRF_internal_write_data <=
                            write_data[31:0];
119                     RamBlockRF_internal_write_en <= 1'b1;
120                 end
121                 else
122                 begin
123                     RamBlockRF_internal_write_en <= 1'b0;
124                 end
125
126                 if((address[11:11] == 1'h1) && read_en)
127                 begin
128                     RamBlockRF_internal_read_en <= 1'b1;
129                 end
130                 else
131                 begin
132                     RamBlockRF_internal_read_en <= 1'b0;
133                 end
134         end
135
136     end
137
138     //Address Decoder Software Read:
139     always @(posedge clk)
140     begin
141
142         if(!res_n)
143         begin
144             invalid_address <= 1'b0;
145             access_complete <= 1'b0;
146             read_data <= 32'b0;
147         end
148         else
149         begin
150
151             casex(address[11:3])
152                 {1'h0,8'bxxxxxxxx}:
153                 begin
154                     read_data[31:0] <=
                            RamBlockRF_external_read_data;
```

```verilog
155                    invalid_address <=
                           RamBlockRF_external_invalid_address;
156                    access_complete <=
                           RamBlockRF_external_access_complete;
157              end
158              {1'h1,8'bxxxxxxxx}:
159              begin
160                  read_data[31:0] <=
                         RamBlockRF_internal_read_data;
161                  invalid_address <=
                         RamBlockRF_internal_invalid_address;
162                  access_complete <=
                         RamBlockRF_internal_access_complete;
163              end
164              default:
165              begin
166                  invalid_address <= read_en || write_en;
167                  access_complete <= read_en || write_en;
168              end
169
170          endcase
171      end
172
173  end
174
175 endmodule
```