Appendix B

Appendix Contents

Appendix B.1 - Event System Area Estimation

To compare the new Event System solutions with the existing Event System used in the AVR® XMEGA, some notions of the cost- and process references used by Atmel Corp. must be presented. **All information released in this section related to library- and die sizes are given with courtesy of the Atmel Corporation.**

A typical AVR® XMEGA device occupies an area from 10-25mm^2 depending on complexity and available I/O-resources, when all analog, digital and padding logic is included. On average the Atmel 35k9 in-house 0.35/0.25µm process library**Feil! Fant ikke referansekilden.** can fit 25k of NAND equivalent gates per mm^2, including associated routing and SRAM. The routing ratio is about 0.90-0.95, which means that 5-10% of the occupied area is for routing. FLASH memory is implemented on 2mm^2-28mm^2 for 16kB-256kB, while SRAM cells occupy 0.2mm^2 for each kB on average. 1kB of FLASH will occupy 0.125mm^2.The current Event System occupies approximately 4.5k NAND gates, or 0.18mm^2.

Area estimations for tables B-2 – B-5 are derived using the following equations:

Estimated area for logic: $$A_L = \frac{25000}{NAND_{Log}}\left[mm^2\right]$$

Additional routing area: $$A_R = A_L \cdot 0.10$$

Additional logic area: $$\mathrm{A_{Le}} = \mathrm{A_L} \cdot 0.05 + \mathrm{n} \cdot 0.125$$

Total estimated area: $$A_{Tot} = A_L + A_R + A_{Le}$$

Total NAND equivalents: $$NAND_{Tot} = 25000 \cdot A_{Tot}$$

For additional logic area *n* represents the amount of additional kB of FLASH memory included.
Table B-1 summarizes standard cell element size relative to NAND2 equivalents. This table is the basis of all architectural NAND equivalent estimations presented in section 7, 8, 9 and 12.
By looking at the architectural drawings presented in appendix A 16.1 – A 16.3 the total NAND gate equivalent for all Event System solutions are calculated and additional logic is derived from Atmel's estimated numbers from the previous paragraph. Tables B-2 – B-5 summarizes the total area results for all Event System solutions.

Gate Type	NAND equivalents	Drive Strength
NAND2	1.0	A-B
SRAM	0.67	
AND2	1.33	A-C
Inv	0.67	A-C
Mux2	2	A-B
Mux4	4.67	A-B
OR2	1.33	A-C
XOR2	2.67	A-C
WCHB	10.05 (estimated)	-
Müller C-element	3.0 (estimated)	-

Table B-1: Standard cell NAND equivalents for Atmel 39k5 in-house process*Feil! Fant ikke referansekilden.*

Domain	NAND equivalents logic	Estimated area for logic [mm^2]	Routing area [mm^2] (+10%)	Additional logic [mm^2] (5% + mem)	**Total estimated area [mm^2]**	**Total NAND equivalents**
Simple I/O	2552	0.102	0.0102	0.0139	**0.126**	**3150**
Medium I/O	3852	0.154	0.0154	0.0165	**0.186**	**4650**
Advanced I/O	6352	0.254	0.0254	0.0215	**0.301**	**7525**

TableB-2: Estimated HERN area

# CPLD MCBs	NAND equivalents logic	Estimated area for logic [mm^2]	Routing area [mm^2] (+10%)	Additional logic [mm^2] (5% + mem)	**Total estimated area [mm^2]**	**Total NAND equivalents**
4	3564	0.142	0.0142	0.0561	**0.212**	**5308**
8	6428	0.257	0.0257	0.1379	**0.420**	**10515**

Table B-3: CERN area estimations

Domain	NAND equivalents logic	Estimated area for logic [mm^2]	Routing area [mm^2] (+10%)	Additional logic [mm^2] (25% + mem)	**Total estimated area [mm^2]**	**Total NAND equivalents**
AGERN + ALERN	3730	0.149	0.0149	0.0529	**0.217**	**5425**
AGERN + ALFERN	3837	0.153	0.0153	0.0539	**0.222**	**5555**
AGERN + ALFERN + 4 TCCFBs	7865	0.315	0.0315	0.110	**0.457**	**11413**

Table B-4: AESRN size overview

# TCFBs	NAND equivalents logic	Estimated area for logic [mm^2]	Routing area [mm^2] (+10%)	Additional logic [mm^2] (25% +mem)	**Total estimated area [mm^2]**	**Total NAND equivalents**
0	3837	0.153	0.0153	0.0529	**0.217**	**5555**
2	4173	0.167	0.0167	0.0730	**0.257**	**6418**
4	4509	0.180	0.0180	0.0764	**0.274**	**6860**
8	5181	0.201	0.0210	0.0831	**0.305**	**7627**
12	5853	0.234	0.0234	0.0898	**0.347**	**8680**
16	6525	0.261	0.0261	0.0965	**0.384**	**9590**
24	7865	0.315	0.0315	0.110	**0.457**	**11413**

Table B-5: TCFB impact on the Asynchronous Event System circuit size

Appendix B.2 – Verilog Code

```
1   //------------------------------------------------------------------------
2   //
3   // Title        : c_element
4   // Design       : Asynchronous logic
5   // Author       : Rune A. Bjørnerud
6   // Company      : NTNU / Atmel Norway
7   //
8   //------------------------------------------------------------------------
9   //
10  // File         : c element.v
11  // Generated    : Mon Mar  9 10:06:03 2009
12  // From         : interface description file
13  // By           : Itf2Vhdl ver. 1.21
14  //
15  //------------------------------------------------------------------------
16  //
17  // Description :
18  // This module describes an hazard free rtl implementation of a Müller
19  //C-element. It is a very simple construction, with a feedback wire
20  //from the output to AND gates connected to both inputs. This feedback
21  //wire produce the stateholding quailty of the C-element.
22  //The realization of this C-element is built on the sketches from appendix A
23  //in the main report.
24  //------------------------------------------------------------------------
25
26  `timescale 1 ns / 1 ps
27
28  //   ****Instanciating module ****
29  module c_element (
30
31  //   ****Input wires ****
32   input wire  a_in,
33   input wire  b_in,
34
35  //   **** Output wires and registers ****
36   output reg  s_out
37
38  );
39
40  //   **** Registers and wires ****
41
42  reg w_1,w_2,w_3;
43
44  //   - Signal calculation -
45
46   always @ (a_in , b_in)
47       begin
48           w_1 = a_in & b_in;
49  //Loop-back connection for stateholding
50           w_2 = a_in & s_out;
51           w_3 = b_in & s_out;
52           s_out = w_1 | w_2 | w_3;
53           end
54
55  endmodule
56
```

```
1   //------------------------------------------------------------------------------
2   //
3   // Title        : c_el_testbench
4   // Design       : Asynchronous_logic
5   // Author       : pcdemon
6   // Company      : NTNU
7   //
8   //------------------------------------------------------------------------------
9   //
10  // File         : c_el_testbench.v
11  // Generated    : Mon Mar  9 11:08:56 2009
12  // From         : interface description file
13  // By           : Itf2Vhdl ver. 1.21
14  //
15  //------------------------------------------------------------------------------
16  //
17  // Description :
18  //   Simple testbench for the C-element.
19  //------------------------------------------------------------------------------
20  `timescale 1 ns / 1 ps
21
22  module c_el_testbench ();
23   parameter ClockPeriod = 20;
24
25  // **** Input registers ****
26  reg clk, a_in, b_in; //inputs
27
28  // **** Output wires ****
29  wire s_out; //output
30
31  // - instantiated module -
32  c_element C_EL (
33   .a_in   (a_in),
34   .b_in   (b_in),
35   .s_out  (s_out)
36   );
37
38  initial clk = 1;
39   always
40    #(ClockPeriod / 2) clk = ! clk;
41  // - Testbench stimulus. To check signal transitions a clock is introduced. -
42    initial
43        begin
44            clk = 0; //rst = 1;
45            a_in = 0; b_in = 0; //initial values
46
47
48           # ClockPeriod a_in = 0; b_in = 0;
49           # ClockPeriod a_in = 0; b_in = 1;
50           # ClockPeriod a_in = 1; b_in = 0;
51           # ClockPeriod a_in = 1; b_in = 1;
52
53           # ClockPeriod a_in = 0; b_in = 0;
54           # ClockPeriod a_in = 0; b_in = 1;
55           # ClockPeriod a_in = 1; b_in = 0;
56           # ClockPeriod a_in = 1; b_in = 1;
57
58           # ClockPeriod a_in = 0; b_in = 1;
59           # ClockPeriod a_in = 0; b_in = 0;
60           # ClockPeriod a_in = 1; b_in = 1;
61           # ClockPeriod a_in = 1; b_in = 0;
62
63           # ClockPeriod a_in = 0; b_in = 0;
64  end
65
66  endmodule
67
```

```
1   //-------------------------------------------------------------------------------
2   //
3   // Title       : WCHB
4   // Design      : Asynchronous_logic
5   // Author      : Rune A. Børnerud
6   // Company     : NTNU / Atmel Norway
7   //
8   //-------------------------------------------------------------------------------
9   //
10  // File        : WCHB.v
11  // Generated   : Tue Mar 10 18:42:00 2009
12  // From        : interface description file
13  // By          : Itf2Vhdl ver. 1.21
14  //
15  //-------------------------------------------------------------------------------
16  //
17  // Description :
18  // This module describes the WCHB handshake buffer element used in all switch
19  //points for the AESRN. C-element instances are included to provide
20  //stateholding capabilities. The module uses an enable signals instead of
21  //the usual acknowledge signal as described in the main report. The interface
22  //can communicate with all instances using a 4- Phase Dual Rail protocol.
23  //-------------------------------------------------------------------------------
24  `timescale 1 ns / 1 ps
25
26
27  module WCHB (
28
29  // **** Input wires ****
30   input wire  ev_c_in,        //Represents the ev_c wire
31   input wire  ev_d_in ,       //Represents the ev_d wire
32   input wire  r_e,            //Received enable for the WCHB instance
33   input wire  rst,
34
35  // **** Output registers ****
36   output reg  ev_c_out,       //output ev_c
37   output reg  ev_d_out,       //output ev_d
38   output reg  t_e             //transmitted enable from the WCHB instance
39  );
40
41  // **** Registers and wires ****
42
43   wire c_elc_out;             //C-element for ev_c wire
44   wire c_eld_out;             //C-element for ev_d wire
45
46  // - instanciating two C_elements for the buffer -
47   c_element  C_EL1(ev_c_in,r_e,c_elc_out);
48
49   c_element  C_EL2(ev_d_in,r_e,c_eld_out);
50
51
52   always @ (c_elc_out or c_eld_out or rst ) //asynchronous
53       begin
54           if (rst == 0)   //active low reset
55               begin
56               ev_c_out <= 0;
57               ev_d_out <= 0;
58               t_e <= 0;   //instancated as busy
59               end
60           else
61               begin
62  //T_e uses the inverse of the C-element output
63  //The DR convention prevents "0" on both inputs
64  //to be coded as a valid value.
65  // Added dealy is just for simulation purposes, especially considering
66  //pipeline distribution.
67               t_e <= #5 !c_elc_out & !c_eld_out;
68               ev_c_out <= #5 c_elc_out;
69               ev_d_out <= #5  c_eld_out;
70               end
71       end
```

```
72
73  endmodule
74
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title       : wchb_testb
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File        : wchb_testb.v
11   // Generated   : Tue Mar 10 19:16:21 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   //  Simple testbench to verify the behaviour of the WCHB buffer element
19   //------------------------------------------------------------------------------
20   `timescale 1 ns / 1 ps
21
22   module wchb_testb ();
23
24       parameter ClockPeriod = 20;
25
26   // **** Input registers ****
27   reg clk, ev_c_in, ev_d_in, t_e_test, rst;
28
29   // **** Output wires ****
30   wire ev_c_out, ev_d_out, r_e_test;
31
32   // - instantiated module -
33   WCHB WCHB_t(
34       .ev_c_in    (ev_c_in),
35       .ev_d_in    (ev_d_in),
36       .r_e        (t_e_test),
37       .rst        (rst),
38       .ev_c_out   (ev_c_out),
39       .ev_d_out   (ev_d_out),
40       .t_e        (r_e_test)
41   );
42
43   initial clk = 1;
44   initial rst = 1;
45    always
46       #(ClockPeriod / 2) clk = ! clk;
47
48       initial //initial values
49           begin
50               clk = 1'b 0;
51               rst = 1'b 0;
52               ev_c_in = 1'b 0; ev_d_in = 1'b 0; t_e_test = 1'b 0;
53
54
55              # ClockPeriod rst = 1'b 1;
56       end
57
58   always @ (posedge clk)
59       begin: Test
60
61           integer counter = 0;
62           reg notify_test = 1'b 0;
63           reg test_free = 1'b 1;
64
65       if(rst)
66           begin
67   // Different test scenarios. When r_e_test is "1" the WCHB is ready
68   // to receive test input.
69               if (r_e_test == 1 & counter < 5)
70                   begin
71                       casez (counter )
```

```
72                          0:
73                              begin
74                                  ev_c_in <= 1'b 1;
75                                  ev_d_in <= 1'b 0;
76                              end
77                          1:
78                              begin
79                                  ev_c_in <= 1'b 0;
80                                  ev_d_in <= 1'b 1;
81                              end
82                          2:
83                              begin
84                                  ev_c_in <= 1'b 1;
85                                  ev_d_in <= 1'b 1;
86                              end
87                          3:
88                              begin
89                                  ev_c_in <= 1'b 0;
90                                  ev_d_in <= 1'b 0;
91                              end
92                          4:
93                                  begin
94                                  ev_c_in <= 1'b 1;
95                                  ev_d_in <= 1'b 0;
96                              end
97                      endcase
98
99                      counter = counter +1;
100                     notify_test = 1;
101                 end
102     //The inner if-sentence introduced a hazard situation for the WCHB
103     //by removing the reset token when the counter is 3. This causes
104     //the operation to stall, and halt the WCHB. For error free operation
105     //remove or comment the inner if sentence.
106             if (r_e_test == 0 & notify_test == 1)
107                 begin
108                     if (counter != 3)
109                         begin           // To provoke a handshake error
110                             ev_c_in <= 1'b 0;
111                             ev_d_in <= 1'b 0;
112                             notify_test = 1'b 0;
113                     end
114                 end
115 // Sets the testbench ready to receive new inputs, after it has issued
116 //the required {00} spacer to the WCHB.
117             if (ev_c_out == 0 & ev_d_out == 0)
118                 begin
119                     if (test_free == 0)
120                         begin
121                             t_e_test <= 1'b 1;
122                             test_free = 1'b 1;
123                         end
124                 end
125 // The testbench receives datat from the WCHB instance, and is set to busy.

126             if(ev_c_out == 1 | ev_d_out == 1)
127                 begin
128                     t_e_test <= 1'b 0;
129                     test_free = 1'b 0;
130                 end
131 // REady to receive new WCHB data
132             if (test_free == 1)
133                 begin
134                     t_e_test <= 1'b 1;
135                 end
136         end
137
138
139 end
140 //after receiving the negative event "01", an invalid condition is tested
141 endmodule
```

142

```
1   //-------------------------------------------------------------------------------
2   //
3   // Title       : copy_element
4   // Design      : Asynchronous_logic
5   // Author      : Rune A. Bjørnerud
6   // Company     : NTNU / Atmel Norway
7   //
8   //-------------------------------------------------------------------------------
9   //
10  // File        : vopy_element.v
11  // Generated   : Fri Apr 17 15:08:18 2009
12  // From        : interface description file
13  // By          : Itf2Vhdl ver. 1.21
14  //
15  //-------------------------------------------------------------------------------
16  //
17  // Description :
18  // This module descirbes a copy element used to fork a signal to different
19  //destinations, and make sure that the correct handshake is sent to and
20  //received from all sources. It is constructed of C-elments providing
21  //the stateholding functionalities needed to wait for the response of all
22  //sources. When the correct handshake response has been received, one
23  //signal is sent to the original source of the forked signal ending the
24  //transmission. The copy element is constructed for use in the ALFERN, and
25  //is therefor constructed to copy four inputs to eight destinations.
26  //The one input - two output copy element described in the main report
27  //is the basic element, and the ALFERN copy element is constructed of four
28  //basic copy elements.
29  //-------------------------------------------------------------------------------
30  `timescale 1 ns / 1 ps
31
32
33  module copy_element (
34
35  // **** Input wires ****
36      input wire [3:0] configuration_bits,
37      input wire [3:0] ev_c_in_CE,
38      input wire [3:0] ev_d_in_CE,
39      input wire [7:0] r_e_CE,       //Received from forked destinations
40
41  // **** Output wires and registers ****
42      output reg [7:0] ev_c_out_CE,
43      output reg [7:0] ev_d_out_CE,
44      output reg [3:0] t_e_CE     //Enable signal to peripherals
45      );
46
47  // **** Local wires and registers ****
48
49  wire [3:0] c_element_out;          //From receiving C-elements
50
51  // - Generate and connect the C-elements
52
53  generate
54      genvar g_i;
55      for (g_i = 0; g_i < 4; g_i = g_i +1 )
56          begin: GEN_C
57              if ( g_i < 4 )
58                  begin
59                      c_element
60                          c_element_i
61                          (
62                              .a_in   (r_e_CE[g_i]),
63                              .b_in   (r_e_CE[g_i+4]),
64                              .s_out  (c_element_out[g_i])
65                          );
66                  end
67          end
68  endgenerate
69
70  // This is the MUX control part, where the configuration bit determines
71  //if one or two receivers communicate with the each two-input Copy Element.
```

```
72   // If a configuration bit is "1" the corresponding 2-input Copy element
73   // communicates with two receivers. If the conf. bit is "0" only one
74   // source is connected.
75   always @ (configuration_bits, c_element_out, ev_c_in_CE, ev_d_in_CE )
76       begin: SELECT_SOURCE
77           if ( configuration_bits[0] == 1 )
78               begin
79                   t_e_CE[0] = c_element_out[0];
80               end
81           else
82               begin
83                   t_e_CE[0] = r_e_CE[0];
84               end
85
86           if ( configuration_bits[1] == 1 )
87               begin
88                   t_e_CE[1] = c_element_out[1];
89               end
90           else
91               begin
92                   t_e_CE[1] = r_e_CE[1];
93               end
94
95           if ( configuration_bits[2] == 1 )
96               begin
97                   t_e_CE[2] = c_element_out[2];
98               end
99           else
100              begin
101                  t_e_CE[2] = r_e_CE[2];
102              end
103
104          if ( configuration_bits[3] == 1 )
105              begin
106                  t_e_CE[3] = c_element_out[3];
107              end
108          else
109              begin
110                  t_e_CE[3] = r_e_CE[3];
111              end
112
113  // Assigning the desired input to the forked outputs.
114  // The forking is in accord with the architectural drawing featured
115  //in Appendix A of the main report.
116          ev_c_out_CE[0] = ev_c_in_CE[0];
117          ev_d_out_CE[0] = ev_d_in_CE[0];
118          ev_c_out_CE[1] = ev_c_in_CE[1];
119          ev_d_out_CE[1] = ev_d_in_CE[1];
120          ev_c_out_CE[2] = ev_c_in_CE[2];
121          ev_d_out_CE[2] = ev_d_in_CE[2];
122          ev_c_out_CE[3] = ev_c_in_CE[3];
123          ev_d_out_CE[3] = ev_d_in_CE[3];
124          ev_c_out_CE[4] = ev_c_in_CE[0];
125          ev_d_out_CE[4] = ev_d_in_CE[0];
126          ev_c_out_CE[5] = ev_c_in_CE[1];
127          ev_d_out_CE[5] = ev_d_in_CE[1];
128          ev_c_out_CE[6] = ev_c_in_CE[2];
129          ev_d_out_CE[6] = ev_d_in_CE[2];
130          ev_c_out_CE[7] = ev_c_in_CE[3];
131          ev_d_out_CE[7] = ev_d_in_CE[3];
132
133      end
134
135
136  endmodule
137
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title       : copy_element_test
4    // Design      : Asynchronous_logic
5    // Author      : pcdemon
6    // Company     : NTNU
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File        : copy_element_test.v
11   // Generated   : Mon Apr 20 13:39:02 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   // SImple test for the output copy element. A configuration is read from file
19   //and the copy element is configured accordingly.
20   //------------------------------------------------------------------------------
21   `timescale 1 ns / 1 ps
22
23   module copy_element_test ();
24
25       parameter ClockPeriod = 20;
26
27   // **** Input registers to the test module ****
28       reg [3:0] configuration_bits;
29       reg [3:0]  ev_c_in_CE;
30       reg [3:0]  ev_d_in_CE;
31       reg [7:0]  t_e_test;
32       reg rst;
33       reg clk;
34   // **** Output wires to the test module ****
35       wire [7:0]  ev_c_out_CE;
36       wire [7:0]  ev_d_out_CE;
37       wire [3:0]  r_e_test;
38
39       reg [4:1] Test_vectors[1:1];    //Configuration vector for the Copy Element
40
41   // - Instanciating test module -
42       copy_element
43           copy_element_test (
44               .configuration_bits         (configuration_bits),
45               .ev_c_in_CE                 (ev_c_in_CE),
46               .ev_d_in_CE                 (ev_d_in_CE),
47               .r_e_CE                     (t_e_test),
48               .ev_c_out_CE                (ev_c_out_CE),
49               .ev_d_out_CE                (ev_d_out_CE),
50               .t_e_CE                     (r_e_test)
51           );
52
53   initial clk = 1;
54   initial rst = 1;
55    always
56       #(ClockPeriod / 2) clk = ! clk;
57
58       initial
59           begin
60
61                $readmemb("CopyElementConf.vec",Test_vectors); //read input vectors
62   // - Initial values of input signals to copy element under test -
63                clk = 0;
64                rst = 0;
65                t_e_test = 4'b 0000;
66                ev_c_in_CE = 4'b 0000;
67                ev_d_in_CE = 4'b 0000;
68   // - Reads the first testvector set from the CopyElementConf.vec file. -
69
70                configuration_bits = Test_vectors[1];
71
```

```
72              # ClockPeriod rst = 1;
73
74      end
75
76      always @ (posedge clk)
77          begin: TEST_STIMULUS
78
79          // - Local registers -
80          reg  [3:0] CE_input_free;       //r_e_test
81          reg  [7:0] wchb_notify;         //t_e_test
82          reg  [7:0] module_free = 8'b 11111111;
83
84          integer i,n,m,k,l,o;    //For couting purposes
85
86          if (rst)
87              begin
88
89                  for( i = 0; i < 8; i = i+1)
90                          begin: CHECK_WCHBs
91                              if( r_e_test[i] == 1 )
92                                  CE_input_free[i] = 1'b 1;
93                              else
94                                  CE_input_free[i] = 1'b 0;
95                          end
96      // - Test sequence with signals issued on output 2 from the peripheral device,
97      // forking to the branches 2 and 6 for the individual channels -
98                  if (CE_input_free[2] == 1 )
99                      begin
100                         ev_c_in_CE[2] = 1;
101                         ev_d_in_CE[2] = 1;
102                     end
103
104     // -Test sequence with forking from peripheral line 2 to wires 4 and 5 -
105                 if (ev_c_out_CE[2] == 1 & ev_c_out_CE[6] == 1)
106                     begin
107                         t_e_test[2] = 0;
108                         t_e_test[6] = 0;
109                         module_free[2] = 0;
110                         module_free[6] = 0;
111                     end
112
113     // - When the zeros have propagated to the output of each channel,
114     //the handshake is completed. -
115     if ( ev_c_out_CE[2] == 0 & ev_d_out_CE[6] == 0 )
116         begin
117             if (module_free[2] == 0 & module_free[6] == 0 )
118                 begin
119                     t_e_test[2] = 1;
120                     t_e_test[6] = 1;
121                     module_free[2] = 1;
122                     module_free[6] = 1;
123                 end
124         end
125
126
127     // - Setting data outputs to zero -
128                 if ( r_e_test[2] == 0 )
129                     begin
130                         ev_c_in_CE[2] = 0;
131                         ev_d_in_CE[2] = 0;
132                     end
133
134     // - For each clock cycle the transmitted enable signal from the testbench
135     //is updated. -
136                     for( n = 0; n < 8; n = n+1)
137                         begin: SET_AVAILABILITY
138                             if ( module_free[n] == 1 )
139                                 begin
140                                     t_e_test[n] = 1'b 1;
141                                 end
142                         end
```

```
143
144
145                 end
146         end
147
148   endmodule
149
```

```
1    //-------------------------------------------------------------------------------
2    //
3    // Title       : switch_block
4    // Design      : Asynchronous_logic
5    // Author      : pcdemon
6    // Company     : NTNU
7    //
8    //-------------------------------------------------------------------------------
9    //
10   // File        : switch_block.v
11   // Generated   : Wed Apr 15 10:55:29 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------------
16   //
17   // Description :
18   // This module describes the infrastructure of a programmable cross switch,
19   //with 32 programming bits to control 8 WCHBs, and provide the correct
20   //handshake for each channel. The cross switch has 16 input channels,
21   //two for each of the receiving directions north and south, and 16 output
22   //channels for the output directions east and west. The cross switch can
23   //be viewed in the appendix of the main report. Of the possible 16 input/output
24   //channels only 8 input/outputs can be used simultainiously,
25   //combined by the configuration bit sequence.
26   //-------------------------------------------------------------------------------
27   `timescale 1 ns / 1 ps
28
29   // - Local size definitions -
30
31   `define CONFIG_SIZE         32
32   `define INPUT_SIZE          8
33   `define OUTPUT_SIZE         8
34   `define HALF_BYTE           4
35   `define WCHB_CNT            8
36
37   module switch_block (
38
39   // **** Input wires ****
40       input wire  [`CONFIG_SIZE-1 : 0] configuration_bits ,  //32 configuration
     bits
41       input wire [`INPUT_SIZE-1 : 0] ev_c_in_N,
42       input wire [`INPUT_SIZE-1 : 0] ev_d_in_N,
43       input wire [`INPUT_SIZE-1 : 0] ev_c_in_S,
44       input wire [`INPUT_SIZE-1 : 0] ev_d_in_S,
45       input wire [`INPUT_SIZE-1 : 0] r_e_W,
46       input wire [`INPUT_SIZE-1 : 0] r_e_E,
47       input wire rst,
48
49       // **** Output wires and registers ****
50       output reg [`OUTPUT_SIZE-1 : 0] ev_c_out_W,
51       output reg [`OUTPUT_SIZE-1 : 0] ev_d_out_W,
52       output reg [`OUTPUT_SIZE-1 : 0] ev_c_out_E,
53       output reg [`OUTPUT_SIZE-1 : 0] ev_d_out_E,
54       output reg [`OUTPUT_SIZE-1 : 0] t_e_N,
55       output reg [`OUTPUT_SIZE-1 : 0] t_e_S
56
57   );
58
59   // **** Internal registers and wires ****
60
61   reg [`INPUT_SIZE-1 : 0] ev_c_in_wchb;
62   reg [`INPUT_SIZE-1 : 0] ev_d_in_wchb;
63   reg [`INPUT_SIZE-1 : 0] r_e_wchb;
64   wire [`OUTPUT_SIZE-1 : 0] ev_c_out_wchb;
65   wire [`OUTPUT_SIZE-1 : 0] ev_d_out_wchb;
66   wire [`OUTPUT_SIZE-1 : 0] t_e_wchb;
67
68
69   // The GEN_CONNECTIONS block evaluates the WCHB connecting to it's
70   //correspondig event channel input/output, and sets the connection
```

```
71   //accordingly.
72
73   always @ ( configuration_bits, r_e_W, r_e_E, t_e_wchb, rst, ev_c_in_N, ev_c_in_S,
74           ev_c_out_wchb, ev_d_out_wchb,ev_d_in_S, ev_d_in_N)
75     begin : GEN_CONNECTIONS
76
77     if ( ! rst ) //reset of all output signals and internal registers
78         begin
79             t_e_N = 8'b 00000000;
80             t_e_S = 8'b 00000000;
81             ev_c_out_W = 8'b 00000000;
82             ev_d_out_W = 8'b 00000000;
83             ev_c_out_E = 8'b 00000000;
84             ev_d_out_E = 8'b 00000000;
85             ev_c_in_wchb = 8'b 00000000;
86             ev_d_in_wchb = 8'b 00000000;
87             r_e_wchb = 8'b 00000000;
88         end
89     else
90         begin
91       //Configuration for event channel 0 connections. The input/output
92       //channels are connected to their corresponding WCHB's input/outputs
93       //in accord with the received configuraion bits.
94           casez ( configuration_bits[3:0] )
95               4'b 0001:
96                   begin
97                       ev_c_in_wchb[0] = ev_c_in_N[0];
98                       ev_d_in_wchb[0] = ev_d_in_N[0];
99                       r_e_wchb[0] = r_e_W[0];
100                      ev_c_out_W[0] = ev_c_out_wchb[0];
101                      ev_d_out_W[0] = ev_d_out_wchb[0];
102                      t_e_N[0] = t_e_wchb[0];
103                      t_e_S[0] = 1'b 0;       //Also setting the unused channels
104                      ev_c_out_E[0] = 1'b 0;
105                      ev_d_out_E[0] = 1'b 0;
106                  end
107              4'b 0010:
108                  begin
109                     ev_c_in_wchb[0] = ev_c_in_S[0];
110                      ev_d_in_wchb[0] = ev_d_in_S[0];
111                      r_e_wchb[0] = r_e_W[0];
112                      ev_c_out_W[0] = ev_c_out_wchb[0];
113                      ev_d_out_W[0] = ev_d_out_wchb[0];
114                      t_e_S[0] = t_e_wchb[0];
115                      t_e_N[0] = 1'b 0;       //Also setting the unused channels
116                      ev_c_out_E[0] = 1'b 0;
117                      ev_d_out_E[0] = 1'b 0;
118                  end
119
120              4'b 0100:
121                  begin
122                      ev_c_in_wchb[0] = ev_c_in_S[0];
123                      ev_d_in_wchb[0] = ev_d_in_S[0];
124                      r_e_wchb[0] = r_e_E[0];
125                      ev_c_out_E[0] = ev_c_out_wchb[0];
126                      ev_d_out_E[0] = ev_d_out_wchb[0];
127                      t_e_S[0] = t_e_wchb[0];
128                      t_e_N[0] = 1'b 0;       //Also setting the unused channels
129                      ev_c_out_W[0] = 1'b 0;
130                      ev_d_out_W[0] = 1'b 0;
131                  end
132
133              4'b 1000:
134                  begin
135                      ev_c_in_wchb[0] = ev_c_in_N[0];
136                      ev_d_in_wchb[0] = ev_d_in_N[0];
137                      r_e_wchb[0] = r_e_E[0];
138                      ev_c_out_E[0] = ev_c_out_wchb[0];
139                      ev_d_out_E[0] = ev_d_out_wchb[0];
140                      t_e_N[0] = t_e_wchb[0];
```

```
141                     t_e_S[0] = 1'b 0;       //Also setting the unused channels
142                     ev_c_out_W[0] = 1'b 0;
143                     ev_d_out_W[0] = 1'b 0;
144                 end
145     //all occurences are covered to remove unvanted latches
146         default:
147                 begin
148                     ev_c_in_wchb[0] = 1'b 0;
149                     ev_d_in_wchb[0] = 1'b 0;
150                     r_e_wchb[0]     = 1'b 0;
151                     ev_c_out_W[0] = 1'b 0;
152                     ev_d_out_W[0] = 1'b 0;
153                     ev_c_out_E[0] = 1'b 0;
154                     ev_d_out_E[0] = 1'b 0;
155                     t_e_N[0] = 1'b 0;
156                     t_e_S[0] = 1'b 0;
157                 end
158         endcase
159
160  //Configuration for event channel 1 connections.
161         casez ( configuration_bits[7:4] )
162             4'b 0001:
163                 begin
164                     ev_c_in_wchb[1] = ev_c_in_N[1];
165                     ev_d_in_wchb[1] = ev_d_in_N[1];
166                     r_e_wchb[1] = r_e_W[1];
167                     ev_c_out_W[1] = ev_c_out_wchb[1];
168                     ev_d_out_W[1] = ev_d_out_wchb[1];
169                     t_e_N[1] = t_e_wchb[1];
170                     t_e_S[1] = 1'b 0;       //Also setting the unused channels
171                     ev_c_out_E[1] = 1'b 0;
172                     ev_d_out_E[1] = 1'b 0;
173                 end
174             4'b 0010:
175                 begin
176                    ev_c_in_wchb[1] = ev_c_in_S[1];
177                     ev_d_in_wchb[1] = ev_d_in_S[1];
178                     r_e_wchb[1] = r_e_W[1];
179                     ev_c_out_W[1] = ev_c_out_wchb[1];
180                     ev_d_out_W[1] = ev_d_out_wchb[1];
181                     t_e_S[1] = t_e_wchb[1];
182                     t_e_N[1] = 1'b 0;       //Also setting the unused channels
183                     ev_c_out_E[1] = 1'b 0;
184                     ev_d_out_E[1] = 1'b 0;
185                 end
186
187             4'b 0100:
188                 begin
189                     ev_c_in_wchb[1] = ev_c_in_S[1];
190                     ev_d_in_wchb[1] = ev_d_in_S[1];
191                     r_e_wchb[1] = r_e_E[1];
192                     ev_c_out_E[1] = ev_c_out_wchb[1];
193                     ev_d_out_E[1] = ev_d_out_wchb[1];
194                     t_e_S[1] = t_e_wchb[1];
195                     t_e_N[1] = 1'b 0;       //Also setting the unused channels
196                     ev_c_out_W[1] = 1'b 0;
197                     ev_d_out_W[1] = 1'b 0;
198                 end
199
200             4'b 1000:
201                 begin
202                     ev_c_in_wchb[1] = ev_c_in_N[1];
203                     ev_d_in_wchb[1] = ev_d_in_N[1];
204                     r_e_wchb[1] = r_e_E[1];
205                     ev_c_out_E[1] = ev_c_out_wchb[1];
206                     ev_d_out_E[1] = ev_d_out_wchb[1];
207                     t_e_N[1] = t_e_wchb[1];
208                     t_e_S[1] = 1'b 0;       //Also setting the unused channels
209                     ev_c_out_W[1] = 1'b 0;
210                     ev_d_out_W[1] = 1'b 0;
211                 end
```

```
212
213                 default:
214                 begin
215                     ev_c_in_wchb[1] = 1'b 0;
216                     ev_d_in_wchb[1] = 1'b 0;
217                     r_e_wchb[1]     = 1'b 0;
218                     ev_c_out_W[1] = 1'b 0;
219                     ev_d_out_W[1] = 1'b 0;
220                     ev_c_out_E[1] = 1'b 0;
221                     ev_d_out_E[1] = 1'b 0;
222                     t_e_N[1] = 1'b 0;
223                     t_e_S[1] = 1'b 0;
224                 end
225         endcase
226
227     //Configuration for event channel 2 connections.
228         casez ( configuration_bits[11:8] )
229             4'b 0001:
230                 begin
231                     ev_c_in_wchb[2] = ev_c_in_N[2];
232                     ev_d_in_wchb[2] = ev_d_in_N[2];
233                     r_e_wchb[2] = r_e_W[2];
234                     ev_c_out_W[2] = ev_c_out_wchb[2];
235                     ev_d_out_W[2] = ev_d_out_wchb[2];
236                     t_e_N[2] = t_e_wchb[2];
237                     t_e_S[2] = 1'b 0;       //Also setting the unused channels
238                     ev_c_out_E[2] = 1'b 0;
239                     ev_d_out_E[2] = 1'b 0;
240                 end
241             4'b 0010:
242                 begin
243                    ev_c_in_wchb[2] = ev_c_in_S[2];
244                     ev_d_in_wchb[2] = ev_d_in_S[2];
245                     r_e_wchb[2] = r_e_W[2];
246                     ev_c_out_W[2] = ev_c_out_wchb[2];
247                     ev_d_out_W[2] = ev_d_out_wchb[2];
248                     t_e_S[2] = t_e_wchb[2];
249                     t_e_N[2] = 1'b 0;       //Also setting the unused channels
250                     ev_c_out_E[2] = 1'b 0;
251                     ev_d_out_E[2] = 1'b 0;
252                 end
253
254             4'b 0100:
255                 begin
256                     ev_c_in_wchb[2] = ev_c_in_S[2];
257                     ev_d_in_wchb[2] = ev_d_in_S[2];
258                     r_e_wchb[2] = r_e_E[2];
259                     ev_c_out_E[2] = ev_c_out_wchb[2];
260                     ev_d_out_E[2] = ev_d_out_wchb[2];
261                     t_e_S[2] = t_e_wchb[2];
262                     t_e_N[2] = 1'b 0;       //Also setting the unused channels
263                     ev_c_out_W[2] = 1'b 0;
264                     ev_d_out_W[2] = 1'b 0;
265                 end
266
267             4'b 1000:
268                 begin
269                     ev_c_in_wchb[2] = ev_c_in_N[2];
270                     ev_d_in_wchb[2] = ev_d_in_N[2];
271                     r_e_wchb[2] = r_e_E[2];
272                     ev_c_out_E[2] = ev_c_out_wchb[2];
273                     ev_d_out_E[2] = ev_d_out_wchb[2];
274                     t_e_N[2] = t_e_wchb[2];
275                     t_e_S[2] = 1'b 0;       //Also setting the unused channels
276                     ev_c_out_W[2] = 1'b 0;
277                     ev_d_out_W[2] = 1'b 0;
278                 end
279
280             default:
281                 begin
282                     ev_c_in_wchb[2] = 1'b 0;
```

```
283                     ev_d_in_wchb[2] = 1'b 0;
284                     r_e_wchb[2]     = 1'b 0;
285                     ev_c_out_W[2] = 1'b 0;
286                     ev_d_out_W[2] = 1'b 0;
287                     ev_c_out_E[2] = 1'b 0;
288                     ev_d_out_E[2] = 1'b 0;
289                     t_e_N [2] = 1'b 0;
290                     t_e_S[2] = 1'b 0;
291                 end
292         endcase
293     //Configuration for event channel 3 connections.
294         casez ( configuration_bits[15:12] )
295             4'b 0001:
296                 begin
297                     ev_c_in_wchb[3] = ev_c_in_N[3];
298                     ev_d_in_wchb[3] = ev_d_in_N[3];
299                     r_e_wchb[3] = r_e_W[3];
300                     ev_c_out_W[3] = ev_c_out_wchb[3];
301                     ev_d_out_W[3] = ev_d_out_wchb[3];
302                     t_e_N[3] = t_e_wchb[3];
303                     t_e_S[3] = 1'b 0;       //Also setting the unused channels
304                     ev_c_out_E[3] = 1'b 0;
305                     ev_d_out_E[3] = 1'b 0;
306                 end
307             4'b 0010:
308                 begin
309                    ev_c_in_wchb[3] = ev_c_in_S[3];
310                     ev_d_in_wchb[3] = ev_d_in_S[3];
311                     r_e_wchb[3] = r_e_W[3];
312                     ev_c_out_W[3] = ev_c_out_wchb[3];
313                     ev_d_out_W[3] = ev_d_out_wchb[3];
314                     t_e_S[3] = t_e_wchb[3];
315                     t_e_N[3] = 1'b 0;       //Also setting the unused channels
316                     ev_c_out_E[3] = 1'b 0;
317                     ev_d_out_E[3] = 1'b 0;
318                 end
319 
320             4'b 0100:
321                 begin
322                     ev_c_in_wchb[3] = ev_c_in_S[3];
323                     ev_d_in_wchb[3] = ev_d_in_S[3];
324                     r_e_wchb[3] = r_e_E[3];
325                     ev_c_out_E[3] = ev_c_out_wchb[3];
326                     ev_d_out_E[3] = ev_d_out_wchb[3];
327                     t_e_S[3] = t_e_wchb[3];
328                     t_e_N[3] = 1'b 0;       //Also setting the unused channels
329                     ev_c_out_W[3] = 1'b 0;
330                     ev_d_out_W[3] = 1'b 0;
331                 end
332 
333             4'b 1000:
334                 begin
335                     ev_c_in_wchb[3] = ev_c_in_N[3];
336                     ev_d_in_wchb[3] = ev_d_in_N[3];
337                     r_e_wchb[3] = r_e_E[3];
338                     ev_c_out_E[3] = ev_c_out_wchb[3];
339                     ev_d_out_E[3] = ev_d_out_wchb[3];
340                     t_e_N[3] = t_e_wchb[3];
341                     t_e_S[3] = 1'b 0;       //Also setting the unused channels
342                     ev_c_out_W[3] = 1'b 0;
343                     ev_d_out_W[3] = 1'b 0;
344                 end
345 
346             default:
347                 begin
348                     ev_c_in_wchb[3] = 1'b 0;
349                     ev_d_in_wchb[3] = 1'b 0;
350                     r_e_wchb[3]     = 1'b 0;
351                     ev_c_out_W[3] = 1'b 0;
352                     ev_d_out_W[3] = 1'b 0;
353                     ev_c_out_E[3] = 1'b 0;
```

```
354                     ev_d_out_E[3] = 1'b 0;
355                     t_e_N[3] = 1'b 0;
356                     t_e_S[3] = 1'b 0;
357                 end
358         endcase
359  //Configuration for event channel 4 connections.
360         casez ( configuration_bits[19:16] )
361             4'b 0001:
362                 begin
363                     ev_c_in_wchb[4] = ev_c_in_N[4];
364                     ev_d_in_wchb[4] = ev_d_in_N[4];
365                     r_e_wchb[4] = r_e_W[4];
366                     ev_c_out_W[4] = ev_c_out_wchb[4];
367                     ev_d_out_W[4] = ev_d_out_wchb[4];
368                     t_e_N[4] = t_e_wchb[4];
369                     t_e_S[4] = 1'b 0;       //Also setting the unused channels
370                     ev_c_out_E[4] = 1'b 0;
371                     ev_d_out_E[4] = 1'b 0;
372                 end
373             4'b 0010:
374                 begin
375                    ev_c_in_wchb[4] = ev_c_in_S[4];
376                     ev_d_in_wchb[4] = ev_d_in_S[4];
377                     r_e_wchb[4] = r_e_W[4];
378                     ev_c_out_W[4] = ev_c_out_wchb[4];
379                     ev_d_out_W[4] = ev_d_out_wchb[4];
380                     t_e_S[4] = t_e_wchb[4];
381                     t_e_N[4] = 1'b 0;       //Also setting the unused channels
382                     ev_c_out_E[4] = 1'b 0;
383                     ev_d_out_E[4] = 1'b 0;
384                 end
385
386             4'b 0100:
387                 begin
388                     ev_c_in_wchb[4] = ev_c_in_S[4];
389                     ev_d_in_wchb[4] = ev_d_in_S[4];
390                     r_e_wchb[4] = r_e_E[4];
391                     ev_c_out_E[4] = ev_c_out_wchb[4];
392                     ev_d_out_E[4] = ev_d_out_wchb[4];
393                     t_e_S[4] = t_e_wchb[4];
394                     t_e_N[4] = 1'b 0;       //Also setting the unused channels
395                     ev_c_out_W[4] = 1'b 0;
396                     ev_d_out_W[4] = 1'b 0;
397                 end
398
399             4'b 1000:
400                 begin
401                     ev_c_in_wchb[4] = ev_c_in_N[4];
402                     ev_d_in_wchb[4] = ev_d_in_N[4];
403                     r_e_wchb[4] = r_e_E[4];
404                     ev_c_out_E[4] = ev_c_out_wchb[4];
405                     ev_d_out_E[4] = ev_d_out_wchb[4];
406                     t_e_N[4] = t_e_wchb[4];
407                     t_e_S[4] = 1'b 0;       //Also setting the unused channels
408                     ev_c_out_W[4] = 1'b 0;
409                     ev_d_out_W[4] = 1'b 0;
410                 end
411
412             default:
413                 begin
414                     ev_c_in_wchb[4] = 1'b 0;
415                     ev_d_in_wchb[4] = 1'b 0;
416                     r_e_wchb[4]     = 1'b 0;
417                     ev_c_out_W[4] = 1'b 0;
418                     ev_d_out_W[4] = 1'b 0;
419                     ev_c_out_E[4] = 1'b 0;
420                     ev_d_out_E[4] = 1'b 0;
421                     t_e_N[4] = 1'b 0;
422                     t_e_S[4] = 1'b 0;
423                 end
424         endcase
```

```
425  //Configuration for event channel 5 connections.
426
427         casez ( configuration_bits[23:20] )
428             4'b 0001:
429                 begin
430                     ev_c_in_wchb[5] = ev_c_in_N[5];
431                     ev_d_in_wchb[5] = ev_d_in_N[5];
432                     r_e_wchb[5] = r_e_W[5];
433                     ev_c_out_W[5] = ev_c_out_wchb[5];
434                     ev_d_out_W[5] = ev_d_out_wchb[5];
435                     t_e_N[5] = t_e_wchb[5];
436                     t_e_S[5] = 1'b 0;       //Also setting the unused channels
437                     ev_c_out_E[5] = 1'b 0;
438                     ev_d_out_E[5] = 1'b 0;
439                 end
440             4'b 0010:
441                 begin
442                    ev_c_in_wchb[5] = ev_c_in_S[5];
443                     ev_d_in_wchb[5] = ev_d_in_S[5];
444                     r_e_wchb[5] = r_e_W[5];
445                     ev_c_out_W[5] = ev_c_out_wchb[5];
446                     ev_d_out_W[5] = ev_d_out_wchb[5];
447                     t_e_S[5] = t_e_wchb[5];
448                     t_e_N[5] = 1'b 0;       //Also setting the unused channels
449                     ev_c_out_E[5] = 1'b 0;
450                     ev_d_out_E[5] = 1'b 0;
451                 end
452
453             4'b 0100:
454                 begin
455                     ev_c_in_wchb[5] = ev_c_in_S[5];
456                     ev_d_in_wchb[5] = ev_d_in_S[5];
457                     r_e_wchb[5] = r_e_E[5];
458                     ev_c_out_E[5] = ev_c_out_wchb[5];
459                     ev_d_out_E[5] = ev_d_out_wchb[5];
460                     t_e_S[5] = t_e_wchb[5];
461                     t_e_N[5] = 1'b 0;       //Also setting the unused channels
462                     ev_c_out_W[5] = 1'b 0;
463                     ev_d_out_W[5] = 1'b 0;
464                 end
465
466             4'b 1000:
467                 begin
468                     ev_c_in_wchb[5] = ev_c_in_N[5];
469                     ev_d_in_wchb[5] = ev_d_in_N[5];
470                     r_e_wchb[5] = r_e_E[5];
471                     ev_c_out_E[5] = ev_c_out_wchb[5];
472                     ev_d_out_E[5] = ev_d_out_wchb[5];
473                     t_e_N[5] = t_e_wchb[5];
474                     t_e_S[5] = 1'b 0;       //Also setting the unused channels
475                     ev_c_out_W[5] = 1'b 0;
476                     ev_d_out_W[5] = 1'b 0;
477                 end
478
479             default:
480                 begin
481                     ev_c_in_wchb[5] = 1'b 0;
482                     ev_d_in_wchb[5] = 1'b 0;
483                     r_e_wchb[5]     = 1'b 0;
484                     ev_c_out_W[5] = 1'b 0;
485                     ev_d_out_W[5] = 1'b 0;
486                     ev_c_out_E[5] = 1'b 0;
487                     ev_d_out_E[5] = 1'b 0;
488                     t_e_N[5] = 1'b 0;
489                     t_e_S[5] = 1'b 0;
490                 end
491         endcase
492
493     //Configuration for event channel 6 connections.
494         casez ( configuration_bits[27:24] )
495             4'b 0001:
```

```
496                 begin
497                     ev_c_in_wchb[6] = ev_c_in_N[6];
498                     ev_d_in_wchb[6] = ev_d_in_N[6];
499                     r_e_wchb[6] = r_e_W[6];
500                     ev_c_out_W[6] = ev_c_out_wchb[6];
501                     ev_d_out_W[6] = ev_d_out_wchb[6];
502                     t_e_N[6] = t_e_wchb[6];
503                     t_e_S[6] = 1'b 0;       //Also setting the unused channels
504                     ev_c_out_E[6] = 1'b 0;
505                     ev_d_out_E[6] = 1'b 0;
506                 end
507             4'b 0010:
508                 begin
509                    ev_c_in_wchb[6] = ev_c_in_S[6];
510                     ev_d_in_wchb[6] = ev_d_in_S[6];
511                     r_e_wchb[6] = r_e_W[6];
512                     ev_c_out_W[6] = ev_c_out_wchb[6];
513                     ev_d_out_W[6] = ev_d_out_wchb[6];
514                     t_e_S[6] = t_e_wchb[6];
515                     t_e_N[6] = 1'b 0;       //Also setting the unused channels
516                     ev_c_out_E[6] = 1'b 0;
517                     ev_d_out_E[6] = 1'b 0;
518                 end
519
520             4'b 0100:
521                 begin
522                     ev_c_in_wchb[6] = ev_c_in_S[6];
523                     ev_d_in_wchb[6] = ev_d_in_S[6];
524                     r_e_wchb[6] = r_e_E[6];
525                     ev_c_out_E[6] = ev_c_out_wchb[6];
526                     ev_d_out_E[6] = ev_d_out_wchb[6];
527                     t_e_S[6] = t_e_wchb[6];
528                     t_e_N[6] = 1'b 0;       //Also setting the unused channels
529                     ev_c_out_W[6] = 1'b 0;
530                     ev_d_out_W[6] = 1'b 0;
531                 end
532
533             4'b 1000:
534                 begin
535                     ev_c_in_wchb[6] = ev_c_in_N[6];
536                     ev_d_in_wchb[6] = ev_d_in_N[6];
537                     r_e_wchb[6] = r_e_E[6];
538                     ev_c_out_E[6] = ev_c_out_wchb[6];
539                     ev_d_out_E[6] = ev_d_out_wchb[6];
540                     t_e_N[6] = t_e_wchb[6];
541                     t_e_S[6] = 1'b 0;       //Also setting the unused channels
542                     ev_c_out_W[6] = 1'b 0;
543                     ev_d_out_W[6] = 1'b 0;
544                 end
545
546             default:
547                 begin
548                     ev_c_in_wchb[6] = 1'b 0;
549                     ev_d_in_wchb[6] = 1'b 0;
550                     r_e_wchb[6]     = 1'b 0;
551                     ev_c_out_W[6] = 1'b 0;
552                     ev_d_out_W[6] = 1'b 0;
553                     ev_c_out_E[6] = 1'b 0;
554                     ev_d_out_E[6] = 1'b 0;
555                     t_e_N[6] = 1'b 0;
556                     t_e_S[6] = 1'b 0;
557                 end
558         endcase
559 //Configuration for event channel 7 connections.
560         casez ( configuration_bits[31:28] )
561             4'b 0001:
562                 begin
563                     ev_c_in_wchb [7] = ev_c_in_N [7];
564                     ev_d_in_wchb [7] = ev_d_in_N [7];
565                     r_e_wchb [7] = r_e_W [7];
566                     ev_c_out_W [7] = ev_c_out_wchb [7];
```

```
567                     ev_d_out_W [7] = ev_d_out_wchb [7];
568                     t_e_N [7] = t_e_wchb [7];
569                     t_e_S[7] = 1'b 0;       //Also setting the unused channels
570                     ev_c_out_E[7] = 1'b 0;
571                     ev_d_out_E[7] = 1'b 0;
572                 end
573             4'b 0010:
574                 begin
575                    ev_c_in_wchb [7] = ev_c_in_S [7];
576                     ev_d_in_wchb [7] = ev_d_in_S [7];
577                     r_e_wchb [7] = r_e_W [7];
578                     ev_c_out_W [7] = ev_c_out_wchb [7];
579                     ev_d_out_W [7] = ev_d_out_wchb [7];
580                     t_e_S [7] = t_e_wchb [7];
581                     t_e_N[7] = 1'b 0;       //Also setting the unused channels
582                     ev_c_out_E[7] = 1'b 0;
583                     ev_d_out_E[7] = 1'b 0;
584                 end
585
586             4'b 0100:
587                 begin
588                     ev_c_in_wchb [7] = ev_c_in_S [7];
589                     ev_d_in_wchb [7] = ev_d_in_S [7];
590                     r_e_wchb [7] = r_e_E [7];
591                     ev_c_out_E [7] = ev_c_out_wchb [7];
592                     ev_d_out_E [7] = ev_d_out_wchb [7];
593                     t_e_S [7] = t_e_wchb [7];
594                     t_e_N[7] = 1'b 0;       //Also setting the unused channels
595                     ev_c_out_W[7] = 1'b 0;
596                     ev_d_out_W[7] = 1'b 0;
597                 end
598
599             4'b 1000:
600                 begin
601                     ev_c_in_wchb [7] = ev_c_in_N [7];
602                     ev_d_in_wchb [7] = ev_d_in_N [7];
603                     r_e_wchb [7] = r_e_E [7];
604                     ev_c_out_E [7] = ev_c_out_wchb [7];
605                     ev_d_out_E [7] = ev_d_out_wchb [7];
606                     t_e_N [7] = t_e_wchb [7];
607                     t_e_S[7] = 1'b 0;       //Also setting the unused channels
608                     ev_c_out_W[7] = 1'b 0;
609                     ev_d_out_W[7] = 1'b 0;
610                 end
611
612             default:
613                 begin
614                     ev_c_in_wchb[7] = 1'b 0;
615                     ev_d_in_wchb[7] = 1'b 0;
616                     r_e_wchb[7]     = 1'b 0;
617                     ev_c_out_W[7] = 1'b 0;
618                     ev_d_out_W[7] = 1'b 0;
619                     ev_c_out_E[7] = 1'b 0;
620                     ev_d_out_E[7] = 1'b 0;
621                     t_e_N [7] = 1'b 0;
622                     t_e_S[7] = 1'b 0;
623
624                 end
625         endcase
626       end
627   end
628
629 // - Mapping WCHB instances to connected signals -
630
631 generate
632     genvar g_i;
633     for (g_i = 0; g_i <= 7; g_i = g_i +1 )
634         begin: GEN_WCHBs
635             if ( g_i < `WCHB_CNT )
636                 begin
637                     WCHB
```

```
638                         WCHB_i
639                         (
640                             .ev_c_in    (ev_c_in_wchb[g_i]),
641                             .ev_d_in    (ev_d_in_wchb[g_i]),
642                             .r_e        (r_e_wchb[g_i]),
643                             .rst        (rst),
644                             .ev_c_out   (ev_c_out_wchb[g_i]),
645                             .ev_d_out   (ev_d_out_wchb[g_i]),
646                             .t_e        (t_e_wchb[g_i])
647                         );
648                 end
649         end
650 endgenerate
651
652 endmodule
653
654 `undef CONFIG_SIZE
655 `undef INPUT_SIZE
656 `undef OUTPUT_SIZE
657 `undef HALF_BYTE
658 `undef WCHB_CNT
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title       : switch_block_test
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File        : switch_block_test.v
11   // Generated   : Wed Apr 15 13:55:25 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   // This testbench reads a switch block configuration from file, and programmes
19   //the switch block accordingly. The testbench interacts with the switch block
20   //according to the 4-PDR handshake protocol, and uses a clock signal
21   //to synchronize the data sent into the switch. The clock is for input
22   //synchronization only, and will not interfere with the asynchronous transfer
23   //scheme in any way.
24   //------------------------------------------------------------------------------
25   `timescale 1 ns / 1 ps
26
27
28   module switch_block_test ();
29
30       parameter ClockPeriod = 20;
31
32   // **** Input registers to the test module ****
33       reg [31:0] configuration_bits;
34       reg [7:0]  ev_c_in_N;
35       reg [7:0]  ev_d_in_N;
36       reg [7:0]  ev_c_in_S;
37       reg [7:0]  ev_d_in_S;
38       reg [7:0]  t_e_test_W;
39       reg [7:0]  t_e_test_E;
40       reg rst;
41       reg clk;
42
43   // **** Output wire from the test module ****
44       wire [7:0]  ev_c_out_W;
45       wire [7:0]  ev_d_out_W;
46       wire [7:0]  ev_c_out_E;
47       wire [7:0]  ev_d_out_E;
48       wire [7:0]  r_e_test_N;
49       wire [7:0]  r_e_test_S;
50
51   // - Register memory to hold input data from file -
52
53   reg [32:1] Test_vectors[8:1];   //One testvector for Each WCHB (configuration)
54   integer counter;    //To read different inputs from file
55   reg update_stimulus;
56   reg shift_conf;
57   reg  [7:0] wchb_free_N;     //r_e_test
58   reg  [7:0] wchb_free_S;
59   reg  [7:0] wchb_notify_S;
60   reg  [7:0] wchb_notify_N;
61   reg  [7:0] wchb_free_W;
62   reg  [7:0] wchb_free_E;
63   reg first_pass;
64   integer i,n,m,k,l,o;
65   //- Instanciation of switch block model -
66
67   switch_block
68       switch_block_tested (
69           .configuration_bits (configuration_bits),
70           .ev_c_in_N          (ev_c_in_N),
71           .ev_d_in_N          (ev_d_in_N),
```

```
72          .ev_c_in_S          (ev_c_in_S),
73          .ev_d_in_S          (ev_d_in_S),
74          .r_e_W              (t_e_test_W),
75          .r_e_E              (t_e_test_E),
76          .rst                (rst),
77          .ev_c_out_W         (ev_c_out_W),
78          .ev_d_out_W         (ev_d_out_W),
79          .ev_c_out_E         (ev_c_out_E),
80          .ev_d_out_E         (ev_d_out_E),
81          .t_e_N              (r_e_test_N),
82          .t_e_S              (r_e_test_S)
83      );
84
85  initial clk = 1;
86  initial rst = 1;
87   always
88       #(ClockPeriod / 2) clk = ! clk;
89
90       initial
91           begin
92
93               $readmemb("SwitchBlockConf.vec",Test_vectors); //read input vectors
94  // - Initial values of input signals to switch block module -
95               clk = 0;
96               rst = 0;
97               t_e_test_W = 8'b 00000000;
98               t_e_test_E = 8'b 00000000;
99               ev_c_in_N = 8'b 00000000;
100              ev_d_in_N = 8'b 00000000;
101              ev_c_in_S = 8'b 00000000;
102              ev_d_in_S = 8'b 00000000;
103              counter = 0;
104              shift_conf = 1'b 0;
105              wchb_free_W = 8'b 00000000;
106              wchb_free_E = 8'b 00000000;
107              wchb_free_N = 8'b 00000000;
108              wchb_free_S = 8'b 00000000;
109              wchb_notify_N = 8'b 00000000;
110              wchb_notify_S = 8'b 00000000;
111              first_pass = 1'b 0;
112
113 // - Reads the first testvector set from the SwitchBlockConf.vec file. -
114 //- The testvectors are used to program the switch block- -
115              configuration_bits = Test_vectors[1];
116
117             # ClockPeriod rst = 1;
118
119 // - Using the testvectors to construct the configuration sequence -
120
121         end
122
123
124 always @ (shift_conf, first_pass,r_e_test_N,r_e_test_S)
125     begin: CHECK_AVAILABILITY
126     // - The first loop iterates through all t_e signals from all WCHBs (r_e to
127     // the testbench),, and marks the ones that are free for data transfer in
128     // each of the switch block input directions -
129     if( shift_conf == 1 | first_pass == 0 )
130         begin
131                     for( i = 0; i < 8; i = i+1)
132                         begin: CHECK_WCHBs
133                             if( r_e_test_N[i] == 1 )
134                                 begin
135                                     wchb_free_N[i] <= 1'b 1;
136                                 end
137                             else
138                                 begin
139                                     wchb_free_N[i] <= 1'b 0;
140                                 end
141                             if( r_e_test_S[i] == 1 )
142                                 begin
```

```
143                                     wchb_free_S[i] <= 1'b 1;
144                                 end
145                             else
146                                 begin
147                                 wchb_free_S[i] <= 1'b 0;
148                                 end
149
150                             end
151                         shift_conf = 1'b 0;
152         end
153         end
154     // - This loop iterates through all of the wchb_free variables in each of
155     //the switchs input directions, and sends a positive event bit on all
156     // free transmission paths -
157     always @ (posedge clk)
158         begin
159
160             if (rst)
161                 begin
162
163                     for ( o = 0; o < 8; o = o+1 )
164                         begin: SEND_DATA
165                             if ( wchb_free_S[o] == 1 )
166                                 begin
167                                     ev_c_in_S[o] <= 1'b 1;
168                                     ev_d_in_S[o] <= 1'b 0;
169                                     wchb_notify_S[o] <= 1'b 1;
170                                 end
171                              if ( wchb_free_N[o] == 1 )
172                                 begin
173                                     ev_c_in_N[o] <= 1'b 1;
174                                     ev_d_in_N[o] <= 1'b 0;
175                                     wchb_notify_N[o] <= 1'b 1;
176                                 end
177                          end
178
179     // - An event bit is send through the switch, and received by the testbench.
180     //To make a correct handshake, the TB sets its transmitted enable signal
181     //to "0", letting the sending WCHB know it is busy. -
182                     for( m = 0; m < 8; m = m+1 )
183                         begin: START_HS
184                             if  (ev_c_out_W[m] == 1 | ev_d_out_W[m] == 1 )
185                                 begin
186                                     t_e_test_W[m] <= 1'b 0;
187                                     wchb_free_W[m] <= 1'b 0;
188                                 end
189                             if  (ev_c_out_E[m] == 1 | ev_d_out_E[m] == 1 )
190                                 begin
191                                     t_e_test_E[m] <= 1'b 0;
192                                     wchb_free_E[m] <= 1'b 0;
193                                 end
194                         end
195     // -The WCHB which has issued an event bit to the TB is sent the appropriate
196     // spacer in order to complete the handshake. This spacer means resetting
197     // for another reception.
198                     for (k = 0; k < 8; k = k+1 )
199                         begin: SET_SPACER
200                             if ( r_e_test_N[k] == 0 & wchb_notify_N[k] == 1)
201                                 begin
202                                     ev_c_in_N[k] <= 1'b 0;
203                                     ev_d_in_N[k] <= 1'b 0;
204                                     wchb_notify_N[k] <= 0;
205                                 end
206                             if ( r_e_test_S[k] == 0 & wchb_notify_S[k] == 1)
207                                 begin
208                                     ev_c_in_S[k] <= 1'b 0;
209                                     ev_d_in_S[k] <= 1'b 0;
210                                     wchb_notify_S[k] <= 0;
211                                 end
212                         end
213     // - The final handshake procedure is issued in this loop, where the
    testbench
```

```
214     //sets its transmitted enable signal to "1" if a spacer is received from
215     //the correct WCHB element. The handshake is complete. This is done for both
216     //eastern and western outputs.
217                     for (l = 0; l < 8; l = l+1 )
218                         begin: RESET_HS
219                             if ( ev_c_out_W[l] == 0 & ev_d_out_W[l] == 0 )
220                                 begin
221                                     if( wchb_free_W[l] == 0)
222                                         begin
223                                             t_e_test_W[l] <= 1'b 1;
224                                             wchb_free_W[l] <= 1'b 1;
225
226                                         end
227                                 end
228                             if ( ev_c_out_E[l] == 0 & ev_d_out_E[l] == 0 )
229                                 begin

230                                     if( wchb_free_E[l] == 0)
231                                         begin
232                                             t_e_test_E[l] <= 1'b 1;
233                                             wchb_free_E[l] <= 1'b 1;
234
235                                         end
236                                 end
237
238
239                         end
240
241     // - For each clock cycle the transmitted enable signal from the testbench
242     //is updated. -
243                     for( n = 0; n < 8; n = n+1)
244                         begin: SET_AVAILABILITY
245                             if ( wchb_free_E[n] == 1 )
246                                 begin
247                                     t_e_test_E[n] <= 1'b 1;
248                                 end
249                              if (wchb_free_W[n] == 1)
250                                  begin
251                                     t_e_test_W[n] <= 1'b 1;
252                                 end
253                             first_pass = 1;
254                         end
255
256             end
257     end
258
259
260 endmodule
261
```

```
1    //-------------------------------------------------------------------------------
2    //
3    // Title       : TCCFB_LUT
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //-------------------------------------------------------------------------------
9    //
10   // File        : TCCFB_LUT.v
11   // Generated   : Tue Apr 21 09:57:33 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------------
16   //
17   // Description :
18   // This module describes the asynchronous LUT with associated PCHB handshake
19   //mechanisms described in the main report. The LUT is configued to make
20   //a logical computation on a given input sequence of events, and produce
21   // the result transmitted according to the PCHB handshake interface.
22   //Tis LUT structure is designed to fit into both the TCCFB and original TC
23   //context. For simplicity this module is a 3LUT designed to fit into an 8bit
24   //register, making it possible to contain two such LUTs for the CCxBUF register.
25   //An extension to a 4LUT is trivial.  Inputs to the LUT is the ev_c wires.
26   //The output is not forked, and communicates with one event channel.
27   //Internal elements include the INPUT_VALID
28   //block, OUTPUT_VALID and INPUT_ENABLE.
29   //-------------------------------------------------------------------------------
30   `timescale 1 ns / 1 ps
31
32
33   module TCCFB_LUT (
34   // *** Input wires ****
35       input wire [7:0] configuration_bits,
36       input wire [2:0] ev_c_in_LUT,
37       input wire [2:0] ev_d_in_LUT,
38       input wire       r_e_LUT,
39       input wire       rst,
40
41   // *** Output registers ***
42       output reg       ev_c_out_LUT,
43       output reg       ev_d_out_LUT,
44       output reg [2:0] t_e_LUT
45       );
46
47   // **** Local registers and wires ****
48   reg XOR_out_1;
49   reg XOR_out_2;
50   reg XOR_out_3;
51   reg [2:0] LUT_input;
52   reg LUT_output;
53   reg output_valid;
54   reg once ;
55   reg res_o;
56   reg latch_enable;
57   reg Latch_output_1;
58   reg Latch_output_2;
59
60   wire C_out;
61   wire input_valid;
62   wire input_enable;
63
64   //Instanciating the C-elements used in the PCHB environment
65
66   //C-element validating the first two input channels
67   c_element
68       c_element_in1 (
69           .a_in               (XOR_out_1),
70           .b_in               (XOR_out_2),
71           .s_out              (C_out)
```

```
72      );
73
74  //Validates the result from the first input channels, and the
75  //third event channel
76  c_element
77      c_element_in2 (
78          .a_in               (XOR_out_3),
79          .b_in               (C_out),
80          .s_out              (input_valid)
81      );
82  //This is the input enable control. If the input and the output is valid, the
83  //input enable is valid. It is however improtant to notice that the enable
84  //signal from the LUT-module is the inverse of input_enable. This is
85  //consistent with the applied transfer protocol.
86  c_element
87      c_element_en (
88          .a_in               (input_valid),
89          .b_in               (output_valid),
90          .s_out              (input_enable)
91      );
92
93  //This block evaluates the LUT inputs according to the present configuration.
94
95  always @ (ev_c_in_LUT)
96      begin: LOOK_UP
97          LUT_input[0] = ev_c_in_LUT[0];
98          LUT_input[1] = ev_c_in_LUT[1];
99          LUT_input[2] = ev_c_in_LUT[2];
100
101         casez(LUT_input)
102             3'b 000:
103                 begin
104                    LUT_output = configuration_bits[0];
105                 end
106             3'b 001:
107                 begin
108                    LUT_output = configuration_bits[1];
109                 end
110             3'b 010:
111                 begin
112                    LUT_output = configuration_bits[2];
113                 end
114
115             3'b 011:
116                 begin
117                    LUT_output = configuration_bits[3];
118                 end
119             3'b 100:
120                 begin
121                    LUT_output = configuration_bits[4];
122                 end
123
124             3'b 101:
125                 begin
126                    LUT_output = configuration_bits[5];
127                 end
128             3'b 110:
129                 begin
130                    LUT_output = configuration_bits[6];
131                 end
132
133             3'b 111:
134                 begin
135                    LUT_output = configuration_bits[7];
136                 end
137         endcase
138
139     end
140
141 //Input validation block. It is constructed according to the schematics
142 //in appendix A
```

```
143  always  @ (ev_c_in_LUT, ev_d_in_LUT,res_o)
144      begin: INPUT_VALID
145          if (res_o == 1'b 1)
146              begin
147                 once = 1'b 0;
148              end
149          else
150              begin
151                  XOR_out_1 = ev_c_in_LUT[0] ^ ev_d_in_LUT[0];
152                  XOR_out_2 = ev_c_in_LUT[1] ^ ev_d_in_LUT[1];
153                  XOR_out_3 = ev_c_in_LUT[2] ^ ev_d_in_LUT[2];
154                  once = 1'b 1;
155              end
156      end
157
158
159  //output valid block. Validiates the dual-rail latch output.
160  always @ (Latch_output_1, Latch_output_2)
161      begin: OUTPUT_VALID
162
163      output_valid = Latch_output_1 | Latch_output_2;
164      end
165
166  always @ (input_enable,input_valid)
167      begin: LATCH_enable
168          //latch_enable = !(input_enable & output_valid);
169          latch_enable <= #5 input_valid & !(input_enable);
170      end
171
172  //Main control block for the PCHB handshake-process
173  always @ (rst, latch_enable, r_e_LUT, Latch_output_1, Latch_output_2, once,
174            LUT_output,input_enable )
175      begin: LUT_HANDSHAKE
176
177          if (!rst)   //resetting of transmitted values and internal regs.
178              begin
179                  ev_c_out_LUT = 1'b 0;
180                  ev_d_out_LUT = 1'b 0;
181                  t_e_LUT = 1'b 0;
182                  Latch_output_1 = 1'b 0;
183                  Latch_output_2 = 1'b 0;
184              end
185          else
186              begin //Legal to transmit LUT value to module output
187                  if ( latch_enable == 1 & r_e_LUT == 1 & once == 1 )
188                      begin
189                          Latch_output_1 = LUT_output;
190                          Latch_output_2 = !(LUT_output);
191                          ev_c_out_LUT =  Latch_output_1;
192                          ev_d_out_LUT =  Latch_output_2;
193                          res_o = 1'b 1;  // notify that output is sent
194                      end
195                  //This state ends the handshake
196                  else  if(latch_enable == 0 & r_e_LUT == 0)
197                      begin
198                          Latch_output_1 = 1'b 0;
199                          Latch_output_2 = 1'b 0;
200                          ev_c_out_LUT =  Latch_output_1;
201                          ev_d_out_LUT =  Latch_output_2;
202                          res_o = 1'b0;  //reset notify
203                      end
204  //The transmitted enable signals are evaluated for each pass.
205                  t_e_LUT[0] = !(input_enable);
206                  t_e_LUT[1] = !(input_enable);
207                  t_e_LUT[2] = !(input_enable);
208              end
209
210
211
212      end
213
```

```
214
215
216  endmodule
217
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title        : TCCFB_LUT_test
4    // Design       : Asynchronous_logic
5    // Author       : Rune A. Bjørnerud
6    // Company      : NTNU / Atmel Norway
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File         : TCCFB_LUT_test.v
11   // Generated    : Tue Apr 21 11:16:02 2009
12   // From         : interface description file
13   // By           : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   // SImple testbench for LUT functionality
19   //------------------------------------------------------------------------------
20   `timescale 1 ns / 1 ps
21
22
23   module TCCFB_LUT_test ();
24
25       parameter ClockPeriod = 20;
26
27       reg [7:0] configuration_bits;
28       reg [2:0] ev_c_in_LUT;
29       reg [2:0] ev_d_in_LUT;
30       reg       t_e_test;
31       reg       rst;
32       reg       clk;
33
34       wire       ev_c_out_LUT;
35       wire       ev_d_out_LUT;
36       wire [2:0] r_e_test;
37
38
39       reg [8:1] Configuration_vector[1:1];    //Configuration vector for the Copy
     Element
40       reg asynch_test;
41       // - Local registers for test purposes -
42       reg   LUT_free;
43       reg   test_free;
44       integer  counter;
45       reg not_zero;
46       reg second_hs;
47
48
49   // - Instanciating test module -
50       TCCFB_LUT
51           LUT_test (
52               .configuration_bits         (configuration_bits),
53               .ev_c_in_LUT                (ev_c_in_LUT),
54               .ev_d_in_LUT                (ev_d_in_LUT),
55               .r_e_LUT                    (t_e_test),
56               .rst                        (rst),
57               .ev_c_out_LUT               (ev_c_out_LUT),
58               .ev_d_out_LUT               (ev_d_out_LUT),
59               .t_e_LUT                    (r_e_test)
60           );
61
62   initial clk = 1;
63   initial rst = 1;
64    always
65       #(ClockPeriod / 2) clk = ! clk;
66
67       initial
68           begin
69               //read input vectors
70               $readmemb("LUTConf.vec",Configuration_vector);
```

```
71   // - Initial values of input signals to copy element under test -
72                clk = 0;
73                rst = 0;
74                asynch_test = 1'b 1;
75                t_e_test = 1'b 0;
76                ev_c_in_LUT = 3'b 000;
77                ev_d_in_LUT = 3'b 000;
78                LUT_free = 1'b 0;
79                   test_free = 1'b 1;
80                   counter = 0;
81                   not_zero = 1'b 0;
82                   asynch_test = 1'b 1;
83                   second_hs = 1'b 0;
84   // - Reads LUT configuration from the LUTConf.vec file. -
85
86                configuration_bits = Configuration_vector[1];
87
88               # ClockPeriod rst = 1;
89   end
90   // This process handles the asynchronous environment interfacing the LUT.
91   //No clock is used for this part of the testbench.
92   always @ (asynch_test, r_e_test,ev_d_out_LUT,ev_c_out_LUT)
93       begin: ASYNCH_TEST
94
95
96           if ( asynch_test == 1'b 0)
97               begin
98
99                   if (ev_c_out_LUT == 0 & ev_d_out_LUT == 0 & test_free == 0)
100                      begin
101                      if (test_free == 0)
102                          begin
103                              t_e_test = 1'b 1;
104                              test_free = 1'b 1;
105                              asynch_test = 1'b 1;  //Handshake complete
106                          end
107                      end
108
109                  if ((ev_c_out_LUT == 1'b 1 | ev_d_out_LUT == 1'b 1)
110                      & r_e_test == 3'b 000 )
111                      begin
112                          t_e_test =  1'b 0;
113                          test_free = 1'b 0;
114                          second_hs = 1'b 1;
115                      end
116
117                  if (r_e_test == 3'b 000 & LUT_free == 0 & second_hs)
118                      begin
119                          LUT_free = 1;
120                          ev_d_in_LUT = 3'b 000;
121                          ev_c_in_LUT = 3'b 000;
122                          second_hs = 0;
123                      end
124
125              end
126
127      end
128
129  //This is the synchronous testpart used for issuing inputs to the LUT.
130      //Different input scenarios are given by incrementing a counter.
131  always @ (posedge clk)
132      begin: TEST_STIMULUS
133
134      if (rst & asynch_test)
135          begin
136
137              if ( r_e_test == 3'b 111 )
138                  begin
139                      LUT_free = 1'b 1;
140                  end
141
```

```
142             if ( test_free == 1'b 1 )
143                 begin
144                     t_e_test = 1'b 1;
145                     not_zero = 1'b 1;
146                 end
147
148
149             if ( LUT_free == 1'b 1 & counter < 8 & test_free == 1'b 1
150                 & asynch_test == 1 )
151                 begin
152
153                 casez (counter)
154                     0:
155                         begin
156                             ev_c_in_LUT[0] <= 0;
157                             ev_d_in_LUT[0] <= 1;
158                             ev_c_in_LUT[1] <= 1;
159                             ev_d_in_LUT[1] <= 0;
160                             ev_c_in_LUT[2] <= 0;
161                             ev_d_in_LUT[2] <= 1;
162                             asynch_test = 0;
163                         end
164                      1:
165                         begin
166                             ev_c_in_LUT[0] <= 1;
167                             ev_d_in_LUT[0] <= 0;
168                             ev_c_in_LUT[1] <= 0;
169                             ev_d_in_LUT[1] <= 1;
170                             ev_c_in_LUT[2] <= 0;
171                             ev_d_in_LUT[2] <= 1;
172                             asynch_test <= 0;
173                         end
174                     2:
175                         begin
176                             ev_c_in_LUT[0] <= 1;
177                             ev_d_in_LUT[0] <= 0;
178                             ev_c_in_LUT[1] <= 1;
179                             ev_d_in_LUT[1] <= 0;
180                             ev_c_in_LUT[2] <= 0;
181                             ev_d_in_LUT[2] <= 1;
182                             asynch_test <= 0;
183                         end
184                     3:
185                         begin
186                             ev_c_in_LUT[0] <= 1;
187                             ev_d_in_LUT[0] <= 0;
188                             ev_c_in_LUT[1] <= 1;
189                             ev_d_in_LUT[1] <= 0;
190                             ev_c_in_LUT[2] <= 0;
191                             ev_d_in_LUT[2] <= 1;
192                             asynch_test = 0;
193                         end
194                     4:
195                         begin
196                             ev_c_in_LUT[0] <= 1;
197                             ev_d_in_LUT[0] <= 0;
198                             ev_c_in_LUT[1] <= 1;
199                             ev_d_in_LUT[1] <= 0;
200                             ev_c_in_LUT[2] <= 1;
201                             ev_d_in_LUT[2] <= 0;
202                             asynch_test = 0;
203                         end
204                     5:
205                         begin
206                             ev_c_in_LUT[0] <= 1;
207                             ev_d_in_LUT[0] <= 0;
208                             ev_c_in_LUT[1] <= 0;
209                             ev_d_in_LUT[1] <= 1;
210                             ev_c_in_LUT[2] <= 1;
211                             ev_d_in_LUT[2] <= 0;
212                             asynch_test = 0;
```

```
213                         end
214                     6:
215                         begin
216                             ev_c_in_LUT[0] <= 1;
217                             ev_d_in_LUT[0] <= 0;
218                             ev_c_in_LUT[1] <= 1;
219                             ev_d_in_LUT[1] <= 0;
220                             ev_c_in_LUT[2] <= 1;
221                             ev_d_in_LUT[2] <= 0;
222                             asynch_test = 0;
223                         end
224                     7:
225                         begin
226                             ev_c_in_LUT[0] <= 1;
227                             ev_d_in_LUT[0] <= 0;
228                             ev_c_in_LUT[1] <= 1;
229                             ev_d_in_LUT[1] <= 0;
230                             ev_c_in_LUT[2] <= 1;
231                             ev_d_in_LUT[2] <= 0;
232                             asynch_test = 0;
233                         end
234
235             endcase
236                 LUT_free = 1'b 0;
237                 counter = counter +1;
238             end
239
240
241     end
242     end
243
244
245 endmodule
246
```

```
1    //-------------------------------------------------------------------------
2    //
3    // Title       : TCCFB
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //-------------------------------------------------------------------------
9    //
10   // File        : TCCFB.v
11   // Generated   : Tue Apr 21 18:05:30 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------
16   //
17   // Description :
18   // This is an example module of a TCCFB using CCbuf registers as LUTs.
19   // Signals can be received both from the ALFERN and the AGERN, and sent
20   //on the same routing networks. Only ONE 3-LUT is included internally in
21   //this example. Each test-scenario for the TCCFB and LUT block must be
22   //specified manually, and two configuration examples are included. The TCCFB
23   //Interacts as the interface between the impolemented 3-LUT and surrounding
24   //AGERN / ALFERN distribution units. The 4-PDR handshake protocol is
25   //implemented as a part of the LUT, not the TCCFB for this version.
26   //-------------------------------------------------------------------------
27   `timescale 1 ns / 1 ps
28
29   module TCCFB (
30   // **** Input wires ****
31
32   //- The amount of configuration bits may vary depending on how many registers
33   //the TCCFB uses for LUT functionality -
34
35       input wire [7:0]  TCCFB_configuration,
36       input wire [15:0] ev_c_in_TCCFB,    //[15:8]  alfern, [7:0] AGERN
37       input wire [15:0] ev_d_in_TCCFB,
38       input wire [9:0]  r_e_TCCFB,        //[9:8] ALFERN, [7:0] AGERN
39       input wire        rst,
40
41   // **** Output registers ****
42       output reg [9:0] ev_c_out_TCCFB,    //[9:8] ALFERN, [7:0] AGERN
43       output reg [9:0] ev_d_out_TCCFB,
44       output reg [15:0] t_e_TCCFB     //[15:8]  ALFERN, [7:0] AGERN
45       );
46
47   //Handling the reset conditions for the TCCFB. The reset function will only
48   //reset the outputs NOT connected to LUTs inside the TCCFB. These outputs
49   //are reset accordingly by the LUT.
50
51   reg [2:0] inputs_c_LUT;
52   reg [2:0] inputs_d_LUT;
53   wire ev_c_o_LUT;
54   wire ev_d_o_LUT;
55   wire [2:0] t_e_LUT;
56
57   //The values in this section must be set manually for each test scenario,
58   //and must be set according to the adress table defined for the TCCFB.
59   always @ *
60       begin: SET_VALUES
61           inputs_c_LUT[0] = ev_c_in_TCCFB[14];     //0 replaced by 14 for scen 2
62           inputs_c_LUT[1] = ev_c_in_TCCFB[10];     //1 replaced by 10 for scen 2
63           inputs_c_LUT[2] = ev_c_in_TCCFB[8];
64           inputs_d_LUT[0] = ev_d_in_TCCFB[14];
65           inputs_d_LUT[1] = ev_d_in_TCCFB[1];
66           inputs_d_LUT[2] = ev_d_in_TCCFB[10];
67
68
69           t_e_TCCFB[14] = t_e_LUT[0];
70           t_e_TCCFB[10] = t_e_LUT[1];
71           t_e_TCCFB[8] =  t_e_LUT[2];        //   Local ALFERN connection
```

```
72
73          ev_c_out_TCCFB[8] = ev_c_o_LUT;    //Output [7:0] AGERN, [9:8] ALFERN CE
74          ev_d_out_TCCFB[8] = ev_d_o_LUT;
75
76      end
77
78  always @ (rst)
79      begin: RESET
80          if (!rst)
81              begin
82
83  //************Test scenario 2  ************
84                  ev_c_out_TCCFB[7:0] = 8'b 00000000;
85                  ev_c_out_TCCFB[9] = 1'b 0;
86
87                  ev_d_out_TCCFB[7:0] = 8'b 00000000;
88                  ev_d_out_TCCFB[9] = 1'b 0;
89
90
91
92  //- Because the LUT gives handshake signals to its own inputs, these must be
93  //omitted in the TCCFB handshake signaling.
94
95  //************Test scenario 1 ************
96                  //t_e_TCCFB[5:2] = 4'b 0000; //Channel 0 and 1 reset by LUT.
97  //              t_e_TCCFB[7:6] = 2'b 00;          //Channel 8 set by LUT
98  //              t_e_TCCFB[15:9] = 1'b 0;
99
100 //************Test scenario 2 ************
101                 t_e_TCCFB[7:0] = 8'b 00000000;
102                 t_e_TCCFB[9] = 1'b 0;
103                 t_e_TCCFB[13:11] = 3'b 000;
104                 t_e_TCCFB[15] = 1'b 0;
105
106                 //t_e_TCCFB[7:1] = 7'b 0000000;
107                 //t_e_TCCFB[11:9] = 3'b 000;
108                 //t_e_TCCFB[15:13] = 3'b 000;
109
110
111             end
112         else //Must be set to "1" for proper reset
113             begin
114 //************Test scenario 1 ************
115                 //t_e_TCCFB[5:2] = 4'b 1111; //Channel 0 and 1 reset by LUT.
116 //              t_e_TCCFB[7:6] = 2'b 11;          //Channel 8 set by LUT
117 //              t_e_TCCFB[15:9] = 1'b 1;
118
119
120 //************Test scenario 2 ************
121                 t_e_TCCFB[7:0] = 8'b 11111111;
122                 t_e_TCCFB[9] = 1'b 1;
123                 t_e_TCCFB[13:11] = 3'b 111;
124                 t_e_TCCFB[15] = 1'b 1;
125
126                 //t_e_TCCFB[7:1] = 7'b 1111111;
127                 //t_e_TCCFB[11:9] = 3'b 111;
128                 //t_e_TCCFB[15:13] = 3'b 111;
129             end
130     end
131
132 //-Instantiating connections to the LUT module(s) -
133     TCCFB_LUT
134         TCCFB_LUT_1 (
135             .configuration_bits         (TCCFB_configuration),
136             .ev_c_in_LUT                (inputs_c_LUT),
137             .ev_d_in_LUT                (inputs_d_LUT),
138             .r_e_LUT                    (r_e_TCCFB[8]),  //AGERN channel 2 for
    output
139             .rst                        (rst),

140             .ev_c_out_LUT               (ev_c_o_LUT),
```

```
141             .ev_d_out_LUT               (ev_d_o_LUT),
142             .t_e_LUT                    (t_e_LUT)
143         );
144
145
146
147 endmodule
148
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title       : AGERN_port
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File        : AGERN_port.v
11   // Generated   : Thu Apr 16 10:09:08 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   // This module connects eight switch blocks to form the Port / T/C side of
19   // the AGERN. The routing pattern is programmable by a total of 256
20   // configuration bits. When connecting switches, input channel 0 from North or
21   //South direction can arbiarily connect to output channel 0 in West or East
22   //direction. It is not possible to connect input channel 1
23   // to output channel 5. See the adress table given in the main report for a
24   //description of how the input/output and enable signals are adressed
25   //to the surrounding environment.
26   //------------------------------------------------------------------------------
27   `timescale 1 ns / 1 ps
28
29   // **** Local defines ****
30   `define CONFIG_SIZE         256
31   `define INPUT_SIZE          48
32   `define OUTPUT_SIZE         48
33
34
35   module AGERN_port (
36   // **** INput wires ****
37       input wire [`CONFIG_SIZE-1:0] configuration_bits,
38       input wire [`INPUT_SIZE-1:0]  ev_c_in,
39       input wire [`INPUT_SIZE-1:0]  ev_d_in,
40       input wire [`INPUT_SIZE-1:0]  r_e,
41       input wire rst,
42
43   // **** Output wires ****
44       output wire [`OUTPUT_SIZE-1:0] ev_c_out,
45       output wire [`OUTPUT_SIZE-1:0] ev_d_out,
46       output wire [`INPUT_SIZE-1:0]  t_e
47
48       );
49
50   // **** Internal wires and registers ****
51
52   // - Intenal signals for switch 1 -
53   wire [7:0] ev_c_out_sw1_E;
54   wire [7:0] ev_d_out_sw1_E;
55   wire [7:0] ev_c_out_sw1_W;
56   wire [7:0] ev_d_out_sw1_W;
57
58   // - Internal signals for switch 2 -
59   wire [7:0] t_e_sw2_S;
60   wire [7:0] t_e_sw2_N;
61
62   // -Internal signals for switch 3 -
63   wire [7:0] ev_c_out_sw3_E;
64   wire [7:0] ev_d_out_sw3_E;
65   wire [7:0] ev_c_out_sw3_W;
66   wire [7:0] ev_d_out_sw3_W;
67
68   // -Internal signals for switch 4 -
69   wire [7:0] t_e_sw4_S;
70   wire [7:0] t_e_sw4_N;
71
```

```
72   //Internal signals for switch 5 -
73   wire [7:0] t_e_sw5_S;
74   wire [7:0] t_e_sw5_N;
75   wire [7:0] ev_c_out_sw5_E;
76   wire [7:0] ev_d_out_sw5_E;
77   wire [7:0] ev_c_out_sw5_W;
78   wire [7:0] ev_d_out_sw5_W;
79
80   // - Internal signals for switch 6 -
81   wire [7:0] t_e_sw6_S;
82   wire [7:0] t_e_sw6_N;
83   wire [7:0] ev_c_out_sw6_E;
84   wire [7:0] ev_d_out_sw6_E;
85   wire [7:0] ev_c_out_sw6_W;
86   wire [7:0] ev_d_out_sw6_W;
87
88   //Internal signals for switch 7 -
89   wire [7:0] t_e_sw7_S;
90   wire [7:0] t_e_sw7_N;
91   wire [7:0] ev_c_out_sw7_E;
92   wire [7:0] ev_d_out_sw7_E;
93
94   // - Internal signals for switch 8 -
95   wire [7:0] t_e_sw8_N;
96   wire [7:0] ev_c_out_sw8_E;
97   wire [7:0] ev_d_out_sw8_E;
98
99   // **** Connecting the switches to form the AGERN ****
100
101  switch_block
102      switch_block_1 (
103      .configuration_bits     (configuration_bits[31:0]),
104      .ev_c_in_N              (ev_c_in[7:0]),     // North side from Port/ I/O 1
105      .ev_d_in_N              (ev_d_in[7:0]),
106      .ev_c_in_S              (ev_c_in[15:8]),    //South side from    Port / I/O 2
107      .ev_d_in_S              (ev_d_in[15:8]),
108      .r_e_W                  (t_e_sw5_N),        //West received from switch 5
109      .r_e_E                  (t_e_sw2_N),        //East received from switch 2
110      .rst                    (rst),
111      .ev_c_out_W             (ev_c_out_sw1_W),   //West output to switch 5
112      .ev_d_out_W             (ev_d_out_sw1_W),
113      .ev_c_out_E             (ev_c_out_sw1_E),   //East output to switch 2
114      .ev_d_out_E             (ev_d_out_sw1_E),
115      .t_e_N                  (t_e[7:0]),         // North output to Port / I/O 1
116      .t_e_S                  (t_e[15:8])         // South output to Port / I/O 2
117      );
118
119  switch_block
120      switch_block_2 (
121      .configuration_bits     (configuration_bits[63:32]),
122      .ev_c_in_N              (ev_c_out_sw1_E),   // North side from Switch 1
123      .ev_d_in_N              (ev_d_out_sw1_E),
124      .ev_c_in_S              (ev_c_out_sw6_E),   //South side from switch 6
125      .ev_d_in_S              (ev_d_out_sw6_E),
126      .r_e_W                  (r_e[7:0]),         //West received from Port / I/O 1
127      .r_e_E                  (r_e[15:8]),        //East received from Port / I/O 2
128      .rst                    (rst),
129      .ev_c_out_W             (ev_c_out[7:0]),    //West output to Port / I/O 1
130      .ev_d_out_W             (ev_d_out[7:0]),
131      .ev_c_out_E             (ev_c_out[15:8]),   //East output to Port / I/O 2
132      .ev_d_out_E             (ev_d_out[15:8]),
133      .t_e_N                  (t_e_sw2_N),        // North output to Switch 1
134      .t_e_S                  (t_e_sw2_S)         // South output to Switch 6
135      );
136
137  switch_block
138      switch_block_3 (
139      .configuration_bits     (configuration_bits[95:64]),
140      .ev_c_in_N              (ev_c_in[23:16]),   // North side from Port/ I/O 3
141      .ev_d_in_N              (ev_d_in[23:16]),
142      .ev_c_in_S              (ev_c_in[31:24]),   //South side from Port / I/O 4
```

```
143      .ev_d_in_S              (ev_d_in[31:24]),
144      .r_e_W                  (t_e_sw5_S),        //West received from switch 5
145      .r_e_E                  (t_e_sw4_N),        //East received from switch 4
146      .rst                    (rst),
147      .ev_c_out_W             (ev_c_out_sw3_W),   //West output to switch 5
148      .ev_d_out_W             (ev_d_out_sw3_W),
149      .ev_c_out_E             (ev_c_out_sw3_E),   //East output to switch 4
150      .ev_d_out_E             (ev_d_out_sw3_E),
151      .t_e_N                  (t_e[23:16]),       // North output to Port / I/O 3
152      .t_e_S                  (t_e[31:24])        // South output to Port / I/O 4
153      );
154
155    switch_block
156       switch_block_4 (
157      .configuration_bits     (configuration_bits[127:96]),
158      .ev_c_in_N              (ev_c_out_sw3_E),   // North side from Switch 3
159      .ev_d_in_N              (ev_d_out_sw3_E),
160      .ev_c_in_S              (ev_c_out_sw6_W),   //South side from switch 6
161      .ev_d_in_S              (ev_d_out_sw6_W),
162      .r_e_W                  (r_e[23:16]),       //West received from Port / I/O 3
163      .r_e_E                  (r_e[31:24]),       //East received from Port / I/O 4
164      .rst                    (rst),
165      .ev_c_out_W             (ev_c_out[23:16]),  //West output to Port / I/O 3
166      .ev_d_out_W             (ev_d_out[23:16]),
167      .ev_c_out_E             (ev_c_out[31:24]),  //East output to Port / I/O 4
168      .ev_d_out_E             (ev_d_out[31:24]),
169      .t_e_N                  (t_e_sw4_N),        // North output to Switch 3
170      .t_e_S                  (t_e_sw4_S)         // South output to Switch 6
171      );
172
173    switch_block
174       switch_block_5 (
175      .configuration_bits     (configuration_bits[159:128]),
176      .ev_c_in_N              (ev_c_out_sw1_W),   // North side from Switch 1
177      .ev_d_in_N              (ev_d_out_sw1_W),
178      .ev_c_in_S              (ev_c_out_sw3_W),   //South side from switch 3
179      .ev_d_in_S              (ev_d_out_sw3_W),
180      .r_e_W                  (t_e_sw7_N),        //West received from Switch 7
181      .r_e_E                  (t_e_sw6_N),        //East received from Switch 6
182      .rst                    (rst),
183      .ev_c_out_W             (ev_c_out_sw5_W),  //West output to Switch 7
184      .ev_d_out_W             (ev_d_out_sw5_W),
185      .ev_c_out_E             (ev_c_out_sw5_E),  //East output to Switch 6
186      .ev_d_out_E             (ev_d_out_sw5_E),
187      .t_e_N                  (t_e_sw5_N),        // North output to Switch 1
188      .t_e_S                  (t_e_sw5_S)         // South output to Switch 3
189      );
190
191      switch_block
192      switch_block_6 (
193      .configuration_bits     (configuration_bits[191:160]),
194      .ev_c_in_N              (ev_c_out_sw5_E),   // North side from Switch 5
195      .ev_d_in_N              (ev_d_out_sw5_E),
196      .ev_c_in_S              (ev_c_out_sw8_E),   //South side from switch 8
197      .ev_d_in_S              (ev_d_out_sw8_E),
198      .r_e_W                  (t_e_sw4_S),        //West received from Switch 4
199      .r_e_E                  (t_e_sw2_S),        //East received from Switch 2
200      .rst                    (rst),
201      .ev_c_out_W             (ev_c_out_sw6_W),  //West output to Switch 4
202      .ev_d_out_W             (ev_d_out_sw6_W),
203      .ev_c_out_E             (ev_c_out_sw6_E),  //East output to Switch 2
204      .ev_d_out_E             (ev_d_out_sw6_E),
205      .t_e_N                  (t_e_sw6_N),        // North output to Switch 5
206      .t_e_S                  (t_e_sw6_S)         // South output to Switch 8
207      );
208
209    switch_block
210       switch_block_7 (
211      .configuration_bits     (configuration_bits[223:192]),
212      .ev_c_in_N              (ev_c_out_sw5_W),   // North side from Switch 5
213      .ev_d_in_N              (ev_d_out_sw5_W),
```

```
214      .ev_c_in_S              (ev_c_in[47:40]),   //South side from Port A/B
215      .ev_d_in_S              (ev_d_in[47:40]),
216      .r_e_W                  (r_e[39:32]),       //West received from GLOBAL
217      .r_e_E                  (t_e_sw8_N),        //East received from switch 8
218      .rst                    (rst),
219      .ev_c_out_W             (ev_c_out[39:32]),  //West output to GLOBAL
220      .ev_d_out_W             (ev_d_out[39:32]),
221      .ev_c_out_E             (ev_c_out_sw7_E),   //East output to switch 8
222      .ev_d_out_E             (ev_d_out_sw7_E),
223      .t_e_N                  (t_e_sw7_N),        // North output to Switch 5
224      .t_e_S                  (t_e[47:40])        // South output to Port A/B
225      );
226
227  switch_block
228      switch_block_8 (
229      .configuration_bits     (configuration_bits[255:224]),
230      .ev_c_in_N              (ev_c_out_sw7_E),   // North side from Switch 7
231      .ev_d_in_N              (ev_d_out_sw7_E),
232      .ev_c_in_S              (ev_c_in[39:32]),   //South side from GLOBAL
233      .ev_d_in_S              (ev_d_in[39:32]),
234      .r_e_W                  (r_e[47:40]),       //West received from PORT A/B
235      .r_e_E                  (t_e_sw6_S),        //East received from Switch 6
236      .rst                    (rst),
237      .ev_c_out_W             (ev_c_out[47:40]),  //West output to PORT A/B
238      .ev_d_out_W             (ev_d_out[47:40]),
239      .ev_c_out_E             (ev_c_out_sw8_E),   //East output to switch 6
240      .ev_d_out_E             (ev_d_out_sw8_E),
241      .t_e_N                  (t_e_sw8_N),        // North output to Switch 7
242      .t_e_S                  (t_e[39:32])        // South output to GLOBAL
243      );
244
245  `undef CONFIG_SIZE
246  `undef INPUT_SIZE
247  `undef OUTPUT_SIZE
248
249  endmodule
250
```

```
1   //------------------------------------------------------------------------------
2   //
3   // Title        : AGERN_port_test
4   // Design       : Asynchronous_logic
5   // Author       : Rune A. Bjørnerud
6   // Company      : NTNU / Atmel Norway
7   //
8   //------------------------------------------------------------------------------
9   //
10  // File         : AGERN_port_test.v
11  // Generated    : Thu Apr 16 11:29:11 2009
12  // From         : interface description file
13  // By           : Itf2Vhdl ver. 1.21
14  //
15  //------------------------------------------------------------------------------
16  //
17  // Description :
18  // This module reads configuration vectors from file, configures the AGERN
19  // routing module, and distributes events according to the configuration file.
20  //------------------------------------------------------------------------------
21  `timescale 1 ns / 1 ps
22
23  module AGERN_port_test ();
24
25
26      parameter ClockPeriod = 20;
27
28  // **** Input registers to the test module ****
29      reg [255:0] configuration_bits;
30      reg [47:0]  ev_c_in;
31      reg [47:0]  ev_d_in;
32      reg [47:0]  t_e_test;
33      reg rst;
34      reg clk;
35
36  // **** Output wire from the test module ****
37      wire [47:0]  ev_c_out;
38      wire [47:0]  ev_d_out;
39      wire [47:0]  r_e_test;
40
41
42  // - Register memory to hold input data from file -
43  // - Each configuration vector is 32 bits, one for each switch block -
44
45  reg [32:1] Conf_vectors[8:1];
46
47  // - Connecting test bench signals to the AGERN module
48      AGERN_port
49          AGERN_port_instance (
50          .configuration_bits         (configuration_bits),
51          .ev_c_in                    (ev_c_in),
52          .ev_d_in                    (ev_d_in),
53          .r_e                        (t_e_test), //transmitted to AGERN_port
    module
54          .rst                        (rst),
55          .ev_c_out                   (ev_c_out),
56          .ev_d_out                   (ev_d_out),
57          .t_e                        (r_e_test) //received in testbench
58      );
59
60  initial clk = 1;
61  initial rst = 1;
62   always
63       #(ClockPeriod / 2) clk = ! clk;
64
65       initial
66           begin
67
68               $readmemb("AGERNConf.vec",Conf_vectors); //read input vectors
69
70  // - Initial values of input signals to switch block module -
```

```
71                clk = 0;
72                rst = 0;
73                t_e_test = 48'b 000000000000000000000000000000000000000000000000;
74                ev_c_in = 48'b 000000000000000000000000000000000000000000000000;
75                ev_d_in = 48'b 000000000000000000000000000000000000000000000000;
76
77    // - Reads the initial configuration from the file AGERNConf.vec -
78    // One line in the configuration file configures one switch block
79                configuration_bits[31:0] = Conf_vectors[1];
80                configuration_bits[63:32] = Conf_vectors[2];
81                configuration_bits[95:64] = Conf_vectors[3];
82                configuration_bits[127:96] = Conf_vectors[4];
83                configuration_bits[159:128] = Conf_vectors[5];
84                configuration_bits[191:160] = Conf_vectors[6];
85                configuration_bits[223:192] = Conf_vectors[7];
86                configuration_bits[255:224] = Conf_vectors[8];
87
88                # ClockPeriod rst = 1;
89
90            end
91
92    always @ (posedge clk)
93        begin: TEST_STIMULUS
94
95        // - Local registers -
96        reg  [47:0] wchb_free;          //r_e_test
97        reg  [47:0] wchb_notify;        //t_e_test
98        reg  [47:0] module_free = 48'b
      111111111111111111111111111111111111111111111111;
99        integer i,n,m,k,l,o;    //For couting purposes
100
101           if (rst)
102               begin
103   // - The first loop iterates through all t_e signals from all WCHBs (r_e to
104   // the testbench),, and marks the ones that are free for data transfer in
105   // each of the switch block input directions -
106                   for( i = 0; i < 48; i = i+1)
107                       begin: CHECK_WCHBs
108                           if( r_e_test[i] == 1 )
109                               wchb_free[i] = 1'b 1;
110                           else
111                               wchb_free[i] = 1'b 0;
112                       end
113
114   // - This loop iterates through all of the wchb_free variables in each of
115   //the switchs input directions, and sends a positive event bit on all
116   // free transmission paths -
117                   for ( o = 0; o < 48; o = o+1 )
118                       begin: SEND_DATA
119                           if ( wchb_free[o] == 1 )
120                               begin
121                                   ev_c_in[o] = 1'b 1;
122                                   ev_d_in[o] = 1'b 0;
123                                   wchb_notify[o] = 1'b 1;
124                               end
125                        end
126
127   // - An event bit is send through the switch, and received by the testbench.
128   //To make a correct handshake, the TB sets its transmitted enable signal
129   //to "0", letting the sending WCHB know it is busy. -
130                   for( m = 0; m < 48; m = m+1 )
131                       begin: START_HS
132                           if  (ev_c_out[m] == 1 | ev_d_out[m] == 1 )
133                               begin
134                                   t_e_test[m] = 1'b 0;
135                                   module_free[m] = 1'b 0;
136                               end
137                       end
138
139   // -The WCHB which has issued an event bit to the TB is sent the appropriate
140   // spacer in order to complete the handshake. This spacer means resetting
```

```
141  // for another reception.
142                  for (k = 0; k < 48; k = k+1 )
143                      begin: SET_SPACER
144                          if ( r_e_test[k] == 0 & wchb_notify[k] == 1)
145                              begin
146                                  ev_c_in[k] = 1'b 0;
147                                  ev_d_in[k] = 1'b 0;
148                                  wchb_notify[k] = 1'b 0;
149                              end
150
151                      end
152
153  // - The final handshake procedure is issued in this loop, where the testbench
154  //sets its transmitted enable signal to "1" if a spacer is received from
155  //the correct WCHB element. The handshake is complete.
156                  for (l = 0; l < 48; l = l+1 )
157                      begin: RESET_HS
158                          if ( ev_c_out[l] == 0 & ev_d_out[l] == 0 )
159                              begin
160                                  if( module_free[l] == 0 )
161                                      begin
162                                          t_e_test[l] = 1'b 1;
163                                          module_free[l] = 1'b 1;
164                                      end
165                              end
166                      end
167
168  // - For each clock cycle the transmitted enable signal from the testbench
169  //is updated. -
170                  for( n = 0; n < 48; n = n+1)
171                      begin: SET_AVAILABILITY
172                          if ( module_free[n] == 1 )
173                              begin
174                                  t_e_test[n] = 1'b 1;
175                              end
176                      end
177              end
178      end
179
180  endmodule
181
```

```
1    //-------------------------------------------------------------------------------
2    //
3    // Title       : ALFERN_port
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //-------------------------------------------------------------------------------
9    //
10   // File        : ALFERN_port.v
11   // Generated   : Wed Apr 22 08:48:30 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------------
16   //
17   // Description :
18   // This module connectes the ports with the ALFERN, and also distributes the
19   ///global signals to the AGERN. This description is at a Port / TCCFB level
     meaning
20   //that the described connections are between one Port I/O instance and ta
21   //Timer Counter Combined Functional Block. Described are both local and
22   //global connections. In this case the timer counter is represented as a
23   //LUT instance, only providing LUT functionality. The Port / I/O is described
24   ///thorugh the test bench, only issuing events for test pruposes initially.
25   //The only inputs are connected to a Port / I/O instance.
26   //The global switches originally depicted on the system sketches is replaced
27   //by a cross switch for simplicity of connection.
28   //-------------------------------------------------------------------------------
29   `timescale 1 ns / 1 ps
30
31   module ALFERN_port (
32       input wire [83:0] configuration_bits,   //Dependent on number of LUTs in
     TCCFB
33       input wire [27:0] ev_c_in_module,       // See adress table for details
34       input wire [27:0] ev_d_in_module,       // See adress table for details
35       input wire [31:0] r_e_module,
36       input wire        rst,
37
38       output reg [31:0] ev_c_out_module,      //See adress table for details
39       output reg [31:0] ev_d_out_module,      //See adress table for details
40       output reg [27:0] t_e_module
41
42       );
43
44   // **** Registers and wires for internal connection ****
45
46   // - Registers and wires for the Output copy elements -
47   reg [3:0] r_e_CEin1;
48   reg [3:0] r_e_CEin2;
49   reg [7:0] r_e_CE_ALFin1;
50   reg [7:0] r_e_CE_ALFin2;
51
52
53   reg [3:0] c_in_ALFTC;
54   reg [3:0] d_in_ALFTC;
55   wire [7:0] c_out_ALFTC;
56   wire [7:0] d_out_ALFTC;
57   wire [3:0] t_e_ALFTC;
58
59   wire [7:0] c_out_ALFin1;
60   wire [7:0] d_out_ALFin1;
61   wire [7:0] c_out_ALFin2;
62   wire [7:0] d_out_ALFin2;
63   wire [3:0] t_e_ALFin1;
64   wire [3:0] t_e_ALFin2;
65   wire [3:0] t_e_CEin1;
66   wire [3:0] t_e_CEin2;
67
68   // - Registers and wires for AGERN input/output switch blocks -
69
```

```
70   wire [7:0] ev_c_out_TC;
71   wire [7:0] ev_d_out_TC;
72   wire [7:0] t_e_swi_S;
73   wire [7:0] t_e_swo_S;
74   wire  [7:0] swo_ev_c_out;
75   wire  [7:0] swo_ev_d_out;
76   wire  [7:0] swo_t_e;
77   wire  [7:0] swi_ev_c_out;
78   wire  [7:0] swi_ev_d_out;
79   wire  [7:0] swi_t_e;
80
81
82   // - Registers and wires for the TCCFB connections -
83   reg [1:0] t_e_ALF_TC;
84   reg [15:0] c_in_TCCFB;
85   reg [15:0] d_in_TCCFB;
86   reg [9:0]  TCCFB_r_e;
87   reg [7:0] ev_c_out_TCCFB;              //To the AGERN out switch
88   reg [7:0] ev_d_out_TCCFB;
89   reg [7:0] t_e_AG_TCCFB;                //TCCFB AGERN enable
90
91   wire [9:0] c_out_TCCFB;
92   wire [9:0] d_out_TCCFB;
93   wire [15:0] TCCFB_t_e;
94   wire [1:0] TCCFB_c_out_ALF;
95   wire [1:0] TCCFB_d_out_ALF;
96   wire [3:0] t_e_ALF_inH;
97   wire [3:0] t_e_ALF_inL;
98
99   // - Registers and wires for Dummy connections to unused switch in/outs
100  wire [7:0] t_e_switch;
101  wire [7:0] ev_d_out_switch;
102  wire [7:0] ev_c_out_switch;
103
104
105  always @ *
106      begin: SET_VALUES
107  // - AGERN switches -
108          ev_c_out_module[15:8] = swo_ev_c_out;          //To AGERN switches
109          ev_d_out_module[15:8] = swo_ev_d_out;
110          t_e_module[7:0] = swo_t_e;                     //TO I/O port
111          ev_c_out_module[7:0] = swi_ev_c_out;           //TO I/O port
112          ev_d_out_module[7:0] = swi_ev_d_out;
113              t_e_module[15:8] = swi_t_e;                //To AGERN Switches
114
115  // - ALFERN copy element 1 -
116          r_e_CE_ALFin1[3:0] = r_e_module[27:24];        //From Port I/O
117          r_e_CE_ALFin1[7:4] = TCCFB_t_e[11:8];          //From TCCFB
118          ev_c_out_module[27:24] = c_out_ALFin1[3:0];    //To Port / I/O
119          ev_d_out_module[27:24] = d_out_ALFin1[3:0];
120          t_e_module[19:16] = t_e_ALFin1;                //To module enable
121
122  // -ALFERN copy element 2 -
123          r_e_CE_ALFin2[3:0] = r_e_module[31:28];        //From Port I/O
124          r_e_CE_ALFin2[7:4] = TCCFB_t_e[15:12];         //From TCCFB
125          ev_c_out_module[31:28] = c_out_ALFin2[3:0];
126          ev_d_out_module[31:28] = d_out_ALFin2[3:0];
127          t_e_module[23:20] = t_e_ALFin2;                //To Module enable
128
129  // -TCCFB copy element for output -
130          c_in_ALFTC[1:0] = ev_c_in_module[25:24];    // From Port I/O
131          c_in_ALFTC[3:2] = c_out_TCCFB[9:8];         //Output from TCCFB
132          d_in_ALFTC[1:0] = ev_d_in_module[25:24];
133          d_in_ALFTC[3:2] = d_out_TCCFB[9:8];         //From TCCFB
134          ev_c_out_module[23:16] = c_out_ALFTC;       //From input copy element
135          ev_d_out_module[23:16] = d_out_ALFTC;
136          t_e_module[25:24] = t_e_ALFTC[1:0];         //From input copy element
137          t_e_ALF_TC = t_e_ALFTC[3:2];
138
139  // - For TCCFB module -
140      c_in_TCCFB[7:0] = ev_c_out_TC;           //From the AGERN in switch
```

```
141     c_in_TCCFB[11:8] = c_out_ALFin1[7:4];   //From the forked output of CElow
142     c_in_TCCFB[15:12] = c_out_ALFin2[7:4];  //From the forked output of CEhigh
143     d_in_TCCFB[7:0] = ev_d_out_TC;
144     d_in_TCCFB[11:8] = d_out_ALFin1[7:4];
145     d_in_TCCFB[15:12] = d_out_ALFin2[7:4];
146     TCCFB_r_e[7:0] = t_e_swo_S;                 //From global out switch
147     TCCFB_r_e[9:8] = t_e_ALFTC[3:2];            //From copy element ALFout
148     ev_c_out_TCCFB = c_out_TCCFB[7:0];          //To AGERN out switch
149     ev_d_out_TCCFB = d_out_TCCFB[7:0];
150     t_e_AG_TCCFB = TCCFB_t_e[7:0];              //To AGERN in switch
151
152
153     end
154
155 switch_block
156     switch_block_out (
157         .configuration_bits (configuration_bits[31:0]),
158         .ev_c_in_N          (ev_c_in_module[7:0]),  //Input from Port I/O pins
159         .ev_d_in_N          (ev_d_in_module[7:0]),
160         .ev_c_in_S          (ev_c_out_TCCFB),       //From TCCFB output
161         .ev_d_in_S          (ev_d_out_TCCFB),
162         .r_e_W              (r_e_module[15:8]),     //From AGERN switches
163         .r_e_E              (t_e_switch),
164         .rst                (rst),
165         .ev_c_out_W         (swo_ev_c_out),          // TO AGERN switches
166         .ev_d_out_W         (swo_ev_d_out),
167         .ev_c_out_E         (ev_c_out_switch),
168         .ev_d_out_E         (ev_d_out_switch),
169         .t_e_N              (swo_t_e),              //TO Port / I/O
170         .t_e_S              (t_e_swo_S)             // TO TCCFB r_e[7:0]
171     );
172
173 switch_block
174     switch_block_in (
175         .configuration_bits (configuration_bits[63:32]),
176         .ev_c_in_N          (ev_c_in_module[15:8]),//INputs from AGERN switches
177         .ev_d_in_N          (ev_d_in_module[15:8]),
178         .ev_c_in_S          (ev_c_out_switch),    //Dummy inputs with zero value
179         .ev_d_in_S          (ev_d_out_switch),
180         .r_e_W              (r_e_module[7:0]),  //From Port / I/O peripheral
181         .r_e_E              (t_e_AG_TCCFB),     //From TCCFB
182         .rst                (rst),
183         .ev_c_out_W         (swi_ev_c_out),
184         .ev_d_out_W         (swi_ev_d_out),
185         .ev_c_out_E         (ev_c_out_TC),          // TO TCCFB in [7:0]
186         .ev_d_out_E         (ev_d_out_TC),
187         .t_e_N              (swi_t_e),              //To R_e [15:8]
188         .t_e_S              (t_e_swi_S)
189     );
190
191 copy_element
192     copy_element_ALFERNinL (
193             .configuration_bits         (configuration_bits[67:64]),
194             .ev_c_in_CE                 (ev_c_in_module[19:16]),
195             .ev_d_in_CE                 (ev_d_in_module[19:16]),
196             .r_e_CE                     (r_e_CE_ALFin1),
197             .ev_c_out_CE                (c_out_ALFin1),
198             .ev_d_out_CE                (d_out_ALFin1),
199             .t_e_CE                     (t_e_ALFin1)
200         );
201
202 copy_element
203     copy_element_ALFERNinH (
204             .configuration_bits         (configuration_bits[71:68]),
205             .ev_c_in_CE                 (ev_c_in_module[23:20]),
206             .ev_d_in_CE                 (ev_d_in_module[23:20]),
207             .r_e_CE                     (r_e_CE_ALFin2),
208             .ev_c_out_CE                (c_out_ALFin2),
209             .ev_d_out_CE                (d_out_ALFin2),
210             .t_e_CE                     (t_e_ALFin2)
211         );
```

```
212
213  copy_element
214      copy_element_TCCFBin (
215              .configuration_bits         (configuration_bits[75:72]),
216              .ev_c_in_CE                 (c_in_ALFTC),
217              .ev_d_in_CE                 (d_in_ALFTC),
218              .r_e_CE                     (r_e_module[23:16]),
219              .ev_c_out_CE                (c_out_ALFTC),
220              .ev_d_out_CE                (d_out_ALFTC),
221              .t_e_CE                     (t_e_ALFTC)
222          );
223
224  TCCFB
225      TCCFB_inst (
226          .TCCFB_configuration                (configuration_bits[83:76]),
227          .ev_c_in_TCCFB                      (c_in_TCCFB),
228          .ev_d_in_TCCFB                      (d_in_TCCFB),
229          .r_e_TCCFB                          (TCCFB_r_e),
230          .rst                                (rst),
231          .ev_c_out_TCCFB                     (c_out_TCCFB),
232          .ev_d_out_TCCFB                     (d_out_TCCFB),
233          .t_e_TCCFB                          (TCCFB_t_e)
234
235      );
236
237
238
239  endmodule
240
```

```
1    //-------------------------------------------------------------------------------
2    //
3    // Title       : ALFERN_module_test
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU
7    //
8    //-------------------------------------------------------------------------------
9    //
10   // File        : ALFERN_module_test.v
11   // Generated   : Wed Apr 22 10:39:50 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------------
16   //
17   // Description :
18   // Test module for the ALFERN distribution system, also testing LUT
19   //functionalities in the TCCFB module. All instances are programmed from file.
20   //-------------------------------------------------------------------------------
21   `timescale 1 ns / 1 ps
22
23   module ALFERN_module_test ();
24
25       parameter ClockPeriod = 20;
26
27       reg [83:0] configuration_bits;
28       reg [27:0] ev_c_in_module;
29       reg [27:0] ev_d_in_module;
30       reg [31:0] t_e_test;
31       reg        rst;
32       reg        clk;
33
34       wire [31:0] ev_c_out_module;
35       wire [31:0] ev_d_out_module;
36       wire [27:0] r_e_test;
37
38       reg [83:0] conf_temp;
39       reg [84:1] Conf_vector[1:1];
40
41       initial clk = 1;
42       initial rst = 1;
43
44   ALFERN_port
45       ALFERN_module_inst (
46           .configuration_bits         (configuration_bits),
47           .ev_c_in_module             (ev_c_in_module),
48           .ev_d_in_module             (ev_d_in_module),
49           .r_e_module                 (t_e_test),
50           .rst                        (rst),
51           .ev_c_out_module            (ev_c_out_module),
52           .ev_d_out_module            (ev_d_out_module),
53           .t_e_module                 (r_e_test)
54       );
55
56    always
57        #(ClockPeriod / 2) clk = ! clk;
58
59        initial
60            begin
61               clk = 0;
62                rst = 0;
63                t_e_test = 32'b 00000000000000000000000000000000;
64                ev_c_in_module = 28'b 0000000000000000000000000000;
65                ev_d_in_module = 28'b 0000000000000000000000000000;
66
67
68                $readmemb("ALFERNConf.vec",Conf_vector); //read input vectors
69                //COnfiguration for ALFERN read from file
70                conf_temp = Conf_vector[1];
71
```

```
72                configuration_bits[31:0] = conf_temp[31:0];
73                configuration_bits[63:32] = conf_temp[63:32];
74                configuration_bits[67:64] = conf_temp[67:64];
75                configuration_bits[71:68] = conf_temp[71:68];
76                configuration_bits[75:72] = conf_temp[75:72];
77                configuration_bits[83:76] = conf_temp[83:76];
78
79                # ClockPeriod rst = 1;
80
81              end
82    always @ (posedge clk)
83        begin: TEST_STIMULUS
84
85        // - Local registers -
86        reg  [27:0] module_free = 28'b 0000000000000000000000000000;    //r_e_test
87        reg  [27:0] notify_test = 28'b 0000000000000000000000000000;    //t_e_test

88        reg  [27:0] test_free = 28'b 1111111111111111111111111111;
89        integer i,n,m,k,l,o;    //For couting purposes
90
91            if (rst)
92                begin
93
94                     for (o = 0; o <  28; o = o+1 )
95                       begin: CHECK_STATUS
96                           if( test_free[o] == 1)
97                               begin
98                                   if (o > 15 & o < 24 )
99                                       begin
100                                          t_e_test[o+8] = 1'b 1;
101                                          t_e_test[o] = 1'b 1;
102                                      end
103                                  else
104                                      t_e_test[o] = 1'b 1;
105
106                                  end
107
108                       end
109
110
111   // - For each iteration all modules that are transmitting the enable
112   //signals making them available for data sending will be marked as free.
113                   for( i = 0; i < 28; i = i+1)
114                       begin: CHECK_INST
115                           if( r_e_test[i] == 1 )
116                               begin
117                               module_free[i] = 1'b 1;
118                               end
119                           else
120                               begin
121                               module_free[i] = 1'b 0;
122                               end
123                       end
124   //The test setup is manually programmed.
125                   if (module_free[14] == 1 & module_free[21] == 1 & module_free[23]
      == 1 )
126                       begin
127                       ev_c_in_module[14] = 1'b 1;    //Selected AGERN input
128                       ev_d_in_module[14] = 1'b 0;
129                       notify_test[14] = 1'b 1;
130                       ev_c_in_module[21] = 1'b 0;     //ALFERN Inputs
131                       ev_d_in_module[21] = 1'b 1;
132                       ev_c_in_module[23] = 1'b 1;
133                       ev_d_in_module[23] = 1'b 0;
134                       notify_test[21] = 1'b 1;
135                       notify_test[23] = 1'b 1;
136                       end
137
138
139   // -  Setting the correct handshake while providing a compensation for the fact
140   //that there are more inputs than outputs to the module. This is due to the
```

```
141  //copy elements.
142                  for (n = 0; n <  28; n = n+1 )
143                      begin: HANDSHAKE
144                          if( r_e_test[n] == 0 & notify_test[n] == 1)
145                              begin
146                                          ev_c_in_module[n] = 1'b 0;
147                                          ev_d_in_module[n] = 1'b 0;
148                                          notify_test[n] = 1'b 0;
149                              end
150                      end
151      //Handshake is provided with the correct offset to account for Copy element
152      //forking. This means that the outputs between 15 and 23 are controlled
153      //by the same handshake signals as the ones from [31:24]. This is in accord
154      //with the copy element forking protocol.
155                  for (m = 0; m <  32; m = m+1 )
156                      begin: END_HS
157                          if ( ev_c_out_module[m] == 0 & ev_d_out_module[m] == 0)
158                              begin
159                                  if (m > 23)
160                                      begin
161                                          if (test_free[m-8] == 0 )
162                                              begin
163                                                  t_e_test[m] = 1'b 1;
164                                                  test_free[m-8] = 1'b 1;
165                                              end
166                                      end
167                                  else if (m <= 15)
168                                      begin
169                                          if(test_free[m] == 0)
170                                              begin
171                                                  t_e_test[m] = 1'b 1;
172                                                  test_free[m] = 1'b 1;
173                                              end
174                                      end
175                              end
176                      end
177      //Outputs are received in the testbench, and apropriate enable signals must
178      //be set to "0".
179
180                  for (k = 0; k <  32; k = k+1 )
181                      begin: INITIATE_HS
182                          if (ev_c_out_module[k] == 1 | ev_d_out_module[k] == 1 )
183                              begin
184                                  if ( k > 23 )
185                                      begin
186                                          t_e_test[k] = 1'b 0;
187                                          t_e_test[k-8] = 1'b 0;
188                                          test_free[k-8] = 1'b 0;
189                                      end
190
191                                  else if (k <= 15)
192                                      begin
193                                          t_e_test[k] = 1'b 0;
194                                          test_free[k] = 1'b 0;
195                                      end
196                              end
197                      end
198
199
200              end
201          end
202
203  endmodule
204
```

```
1    //-------------------------------------------------------------------------------
2    //
3    // Title       : AESRN
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //-------------------------------------------------------------------------------
9    //
10   // File        : AESRN.v
11   // Generated   : Tue May 12 10:50:34 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //-------------------------------------------------------------------------------
16   //
17   // Description :
18   // This module represents the complete connection between AGERN, ALFERN and
19   // TCCFBs with inplemented LUTs. See comments for connection guidance.
20   //-------------------------------------------------------------------------------
21   `timescale 1 ns / 1 ps
22
23   module AESRN (
24       //*** Input wires ***
25
26       input wire [339:0] configuration_bits,
27       input wire [59:0] ev_c_in,
28       input wire [59:0] ev_d_in,
29       input wire [63:0] r_e,
30       input wire        rst,
31
32       //***Output regs ****
33       output reg [63:0] ev_c_out,
34       output reg [63:0] ev_d_out,
35       output reg [59:0] t_e
36       );
37
38       // *** AGERN regs and wires ****
39       reg [47:0] AGERN_c_in;
40       reg [47:0] AGERN_d_in;
41       reg [47:0] AGERN_r_e;
42       wire [47:0] AGERN_c_out;
43       wire [47:0] AGERN_d_out;
44       wire [47:0] AGERN_t_e;
45
46       //** ALFERN regs and wires
47       reg [27:0] ALF_c_in;
48       reg [27:0] ALF_d_in;
49       reg [31:0] ALF_r_e;
50       wire [31:0] ALF_c_out;
51       wire [31:0] ALF_d_out;
52       wire [27:0] ALF_t_e;
53
54
55   always @ *
56       begin: SET_VALUES
57
58           //*** AESRN OUtputs ****
59       ev_c_out[39:0] = AGERN_c_out[47:8];   //Excluding sw 1 connected to ALFERN
60       ev_c_out[47:40] = ALF_c_out[7:0];
61       ev_c_out[63:48] = ALF_c_out[31:16];   //excluding sw 2 connected to AGERN
62       ev_d_out[39:0] = AGERN_d_out[47:8];   //Excluding sw 1 connected to ALFERN
63       ev_d_out[47:40] = ALF_d_out[7:0];
64       ev_d_out[63:48] = ALF_d_out[31:16];   //excluding sw 2 connected to AGERN
65
66       t_e[39:0] = AGERN_t_e[47:8];            //Ecxluding t_e from switch 1
67       t_e[47:40] = ALF_t_e[7:0];
68       t_e[59:48] = ALF_t_e[27:16];            //Excluding t_e to sw 1
69
70
71       // *** AGERN connections ****
```

```
72        AGERN_c_in[7:0]  = ALF_c_out[15:8]; //From ALFERN global out switch
73        AGERN_c_in[47:8] = ev_c_in[39:0];   //From testbench environment
74        AGERN_d_in[7:0]  = ALF_d_out[15:8];
75        AGERN_d_in[47:8] = ev_d_in[39:0];
76
77        AGERN_r_e[7:0] = ALF_t_e[15:8];         //From ALFERN global in switch
78        AGERN_r_e[47:8] = r_e[39:0];            //Testbench signals
79
80        // *** ALFERN connections ***
81        ALF_c_in[15:8] = AGERN_c_out[7:0];   //To ALFERN in switch from AGERN switch
    2
82        ALF_c_in[7:0]  = ev_c_in[47:40];     //ALFERN I/O inputs
83        ALF_c_in[27:16] = ev_c_in[59:48];
84        ALF_d_in[15:8] = AGERN_d_out[7:0];
85        ALF_d_in[7:0]  = ev_d_in[47:40];     //ALFERN I/O inputs
86        ALF_d_in[27:16] = ev_d_in[59:48];
87
88        ALF_r_e[7:0] = r_e[47:40];              //To in switch from Port / I/O
89        ALF_r_e[15:8] = AGERN_t_e[7:0];         //To out switch from AGERN
90        ALF_r_e[31:16] = r_e[63:48];            //From testbench environment
91
92        end
93
94
95        AGERN_port
96            AGERN_module (
97                .configuration_bits             (configuration_bits[255:0]),
98                .ev_c_in                        (AGERN_c_in),
99                .ev_d_in                        (AGERN_d_in),
100               .r_e                            (AGERN_r_e),
101               .rst                            (rst),
102               .ev_c_out                       (AGERN_c_out),
103               .ev_d_out                       (AGERN_d_out),
104               .t_e                            (AGERN_t_e)
105               );
106
107        ALFERN_port
108           ALFERN_module (
109             .configuration_bits               (configuration_bits[339:256]),
110             .ev_c_in_module                   (ALF_c_in),
111             .ev_d_in_module                   (ALF_d_in),
112             .r_e_module                       (ALF_r_e),
113             .rst                              (rst),
114             .ev_c_out_module                  (ALF_c_out),
115             .ev_d_out_module                  (ALF_d_out),
116             .t_e_module                       (ALF_t_e)
117             );
118
119   endmodule
120
```

```
1    //------------------------------------------------------------------------------
2    //
3    // Title       : AESRN_test
4    // Design      : Asynchronous_logic
5    // Author      : Rune A. Bjørnerud
6    // Company     : NTNU / Atmel Norway
7    //
8    //------------------------------------------------------------------------------
9    //
10   // File        : AESRN_test.v
11   // Generated   : Tue May 12 11:51:55 2009
12   // From        : interface description file
13   // By          : Itf2Vhdl ver. 1.21
14   //
15   //------------------------------------------------------------------------------
16   //
17   // Description :
18   // Testbench for the AESRN . COnfigurations for two different test scenarios,
19   // the ones described in the main report, is included.
20   //------------------------------------------------------------------------------
21   `timescale 1 ns / 1 ps
22
23   module AESRN_test ();
24
25       parameter ClockPeriod = 20;
26
27       reg [339:0] configuration_bits;
28       reg [59:0] ev_c_in;
29       reg [59:0] ev_d_in;
30       reg [63:0] t_e_test;
31       reg        rst;
32       reg        clk;
33
34       wire [63:0] ev_c_out;
35       wire [63:0] ev_d_out;
36       wire [59:0] r_e_test;
37
38
39       reg [32:1] Conf_vectors[10:1];  //For switch configurations
40       reg [4:1] CE_vectors[3:1];      //Copy element configurations
41       reg [8:1] TCCFB_vector [1:1];   // TCCFB configuration
42
43       initial clk = 1;
44       initial rst = 1;
45
46
47       AESRN
48           AESRN_module_inst (
49               .configuration_bits     (configuration_bits),
50               .ev_c_in                (ev_c_in),
51               .ev_d_in                (ev_d_in),
52               .r_e                    (t_e_test),
53               .rst                    (rst),
54               .ev_c_out               (ev_c_out),
55               .ev_d_out               (ev_d_out),
56               .t_e                    (r_e_test)
57           );
58
59       always
60        #(ClockPeriod / 2) clk = ! clk;
61
62        initial
63            begin
64                clk = 0;
65                rst = 0;
66                t_e_test = 64'b
     0000000000000000000000000000000000000000000000000000000000000000;
67                ev_c_in = 60'b
     000000000000000000000000000000000000000000000000000000000000;
68                ev_d_in = 60'b
     000000000000000000000000000000000000000000000000000000000000;
```

```
69
70
71                $readmemb("AESRNConf.vec",Conf_vectors);   //read  switch vectors
72                $readmemb("CEConf.vec",CE_vectors);        //read CE vectors
73                $readmemb("TCCFBConf.vec",TCCFB_vector);   //read TCCFB vectors
74
75
76               //AGERN configuration
77               configuration_bits[31:0] = Conf_vectors[1];     //AGERN switch 1
78               configuration_bits[63:32] = Conf_vectors[2];    //AGERN switch 2
79               configuration_bits[95:64] = Conf_vectors[3];    //AGERN switch 3
80               configuration_bits[127:96] = Conf_vectors[4];   //AGERN switch 4
81               configuration_bits[159:128] = Conf_vectors[5];  //AGERN switch 5
82               configuration_bits[191:160] = Conf_vectors[6];  //AGERN switch 6
83               configuration_bits[223:192] = Conf_vectors[7];  //AGERN switch 7
84               configuration_bits[255:224] = Conf_vectors[8];    //AGERN switch 8
85
86                configuration_bits[287:256] = Conf_vectors[9];  //ALFERN in switch
87                configuration_bits[319:288] = Conf_vectors[10]; //ALFERN out switch
88                configuration_bits[323:320] = CE_vectors[1];    //Copy element
89                configuration_bits[327:324] = CE_vectors[2];    //Copy element
90                configuration_bits[331:328] = CE_vectors[3];    //Copy element
91                configuration_bits[339:332] = TCCFB_vector[1];  // TCCFB with LUT
92
93                # ClockPeriod rst = 1;
94
95               end
96
97   always @ (posedge clk)
98       begin: TEST_STIMULUS
99
100      // - Local registers -
101      reg  [59:0] module_free = 59'b
     00000000000000000000000000000000000000000000000000000000000;    //r_e_test
102      reg  [59:0] notify_test = 59'b
     00000000000000000000000000000000000000000000000000000000000;    //t_e_test
103      reg  [59:0] test_free =   59'b
     11111111111111111111111111111111111111111111111111111111111;
104      integer i,n,m,k,l,o;    //For couting purposes
105
106          if (rst)
107              begin
108  //Marking all event channels ready to receive inputs
109                  for (o = 0; o <  64; o = o+1 )
110                    begin: CHECK_STATUS
111                        if( test_free[o] == 1)
112                            begin
113                                if (o > 48 & o <= 55 )   //For duplicate copy oCE
     outputs
114                                    begin
115                                        t_e_test[o+8] = 1'b 1;
116                                        t_e_test[o] = 1'b 1;
117                                    end
118                                else
119                                    t_e_test[o] = 1'b 1;
120
121                                end
122
123                    end
124
125
126  // - For each iteration all modules that are transmitting the enable
127  //signals making them available for data reception will be marked as free.
128                 for( i = 0; i < 60; i = i+1)
129                     begin: CHECK_INST
130                         if( r_e_test[i] == 1 )
131                             begin
132                             module_free[i] = 1'b 1;
133                             end
134                         else
135                             begin
```

```
136                             module_free[i] = 1'b 0;
137                             end
138                     end
139     //This if sets up a specific distributions scenario. 40 Issues a signal
140     //on channel 0 from PORT AB. 17 issues a signal on ch 1 from PORT E
141     //and
142
143     //**********Test scenario 1 ***** Signals from Port E and
144     //Port A/B and ALFERN low CE
145             //if (module_free[32] == 1 & module_free[9] == 1 & module_free[48] == 1 )

146     //                  begin
147     //                  ev_c_in[32] = 1'b 1;        //Selected AGERN input
148     //                  ev_d_in[32] = 1'b 0;
149     //                  notify_test[32] = 1'b 1;
150     //                  ev_c_in[9] = 1'b 1;
151     //                  ev_d_in[9] = 1'b 0;
152     //                  ev_c_in[48] = 1'b 1;         //ALFERN Input
153     //                  ev_d_in[48] = 1'b 0;
154     //                  notify_test[9] = 1'b 1;
155     //                  notify_test[48] = 1'b 1;
156     //                  end
157
158     //**********Test scenario 2 *****
159     //This is test scenario 2, only using ALFERN inputs. Two from CElow and 1 from CE
        high.
160
161                     if (module_free[54] == 1 & module_free[50] == 1 & module_free[48]
        == 1 )
162                         begin
163                         ev_c_in[54] = 1'b 1;
164                         ev_d_in[54] = 1'b 0;
165                         notify_test[54] = 1'b 1;
166                         ev_c_in[50] = 1'b 1;
167                         ev_d_in[50] = 1'b 0;
168                         ev_c_in[48] = 1'b 1;         //ALFERN Input
169                         ev_d_in[48] = 1'b 0;
170                         notify_test[50] = 1'b 1;
171                         notify_test[48] = 1'b 1;
172                         end
173
174                     //if (module_free[21] == 1 & module_free[23] == 1 ) ALFERN inputs
175     //                  begin
176     //                      ev_c_in_module[21] = 1'b 1;
177     //                      ev_d_in_module[21] = 1'b 0;
178     //                      ev_c_in_module[23] = 1'b 1;
179     //                      ev_d_in_module[23] = 1'b 0;
180     //                      notify_test[21] = 1'b 1;
181     //                      notify_test[23] = 1'b 1;
182     //                  end
183
184     // -  Setting the correct handshake while providing a compensation for the fact
185     //that there are more inputs than outputs to the module. This is due to the
186     //copy elements.
187                     for (n = 0; n < 60; n = n+1 )
188                         begin: HANDSHAKE
189                             if( r_e_test[n] == 0 & notify_test[n] == 1)
190                                 begin
191                                             ev_c_in[n] = 1'b 0;
192                                             ev_d_in[n] = 1'b 0;
193                                             notify_test[n] = 1'b 0;
194                                 end
195                         end
196     //Correct handshake taking forking of signals from copy elements under
197     //consideration.
198                     for (m = 0; m <  64; m = m+1 )
199                         begin: END_HS
200                             if ( ev_c_out[m] == 0 & ev_d_out[m] == 0)
201                                 begin
202                                     if (m > 55)
203                                         begin
```

```
204                                         if (test_free[m-8] == 0 )
205                                             begin
206                                                 t_e_test[m] = 1'b 1;
207                                                 test_free[m-8] = 1'b 1;
208                                             end
209                                     end
210                                 else if (m < 48)
211                                     begin
212                                         if(test_free[m] == 0)
213                                             begin
214                                                 t_e_test[m] = 1'b 1;
215                                                 test_free[m] = 1'b 1;
216                                             end
217                                     end
218                             end
219                     end
220     //Initiating the handshake by acknowleding data reception
221                 for (k = 0; k <  64; k = k+1 )
222                     begin: INITIATE_HS
223                         if (ev_c_out[k] == 1 | ev_d_out[k] == 1 )
224                             begin
225                                 if ( k > 55 )
226                                     begin
227                                         t_e_test[k] = 1'b 0;
228                                         t_e_test[k-8] = 1'b 0;
229                                         test_free[k-8] = 1'b 0;
230                                     end
231 
232                                  if (k < 48)
233                                     begin
234                                         t_e_test[k] = 1'b 0;
235                                         test_free[k] = 1'b 0;
236                                     end
237                             end
238                     end
239 
240 
241             end
242         end
243 
244 
245 
246 endmodule
247 
```

Appendix B.3 - CPLD Internal Macrocell Logic Estimation

Logic Element	NAND-eq.	# elements	Total NAND-eq.
5 input AND	7 (est)	5	35
inverter	0.67	14	10
6 input OR	8 (est)	1	8
2 input XOR	2.67 NAND	1	2.67
2:1 MUX	2	4	8
4:1 MUX	4.67	1	4.67
2 input OR	1.33	1	1.33
3:1 MUX	3.33	1	3.33
Misc.			7
PTMUX	50 (est)	1	50
TOTAL			130

Table B-6: CPLD macrocell logic gate count