

```

1
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_arith.all;
5 use ieee.std_logic_unsigned.all;
6
7 package hamming_pack is
8     constant C_INIT : std_logic := '0';
9     function get_codeword_size (constant data_width : integer) return integer;
10    function swop (d : std_logic_vector) return std_logic_vector;
11 end package hamming_pack;
12
13 package body hamming_pack is
14
15     function get_codeword_size (constant data_width : integer) return integer is
16     begin
17         for i in 1 to data_width loop
18             if ((2**i) >= (data_width+i+1)) then
19                 return (data_width+i);
20             end if;
21         end loop;
22         return 0;
23     end function get_codeword_size;
24
25     function swop (d : std_logic_vector) return std_logic_vector is
26     variable t : std_logic_vector(d'range);
27     begin
28         for i in d'low to d'high loop
29             t(i):=d(d'high-(i-d'low));
30         end loop;
31         return t;
32     end function swop;
33
34 end package body hamming_pack;
35
36
37 library ieee;
38 use ieee.std_logic_1164.all;
39 use ieee.std_logic_arith.all;
40 use ieee.std_logic_unsigned.all;
41 library work;
42 use work.hamming_pack.all;
43
44 entity hamming_enc is
45     generic (data_width : integer := 12);
46     port(
47         clk      : in  std_logic;
48         rst      : in  std_logic;
49         en       : in  std_logic;
50         data     : in  std_logic_vector((data_width-1) downto 0);
51         val      : out std_logic;
52         codeword : out std_logic_vector((get_codeword_size(data_width)-1) downto 0)
53     );
54 end entity hamming_enc;
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

```

72 architecture arc_hamming_enc of hamming_enc is
73 begin
74     process (clk, rst) is
75         variable p : std_logic_vector(0 to ((codeword'length-data'length)-1));
76         variable codeword_ascending_range : std_logic_vector(1 to codeword'length);
77         variable check_bits_position      : std_logic_vector((p'length-1) downto 0);
78         variable k                        : integer;
79         alias data_ascending_range : std_logic_vector(1 to data'length) is data;
80     begin
81         if rst = C_INIT then
82             codeword <= (others => '0');
83             val      <= '0';
84         elsif rising_edge(clk) then
85             codeword_ascending_range := (others => '0');
86             p                        := (others => '0');
87             k                        := 1;
88             for n in 1 to p'high loop
89                 for m in ((2**n)+1) to ((2**(n+1))-1) loop
90                     if k <= data'length then
91                         codeword_ascending_range(m) := data_ascending_range(k);
92                         k                            := k+1;
93                     end if;
94                 end loop;
95             end loop;
96             for i in p'range loop
97                 for j in codeword_ascending_range'range loop
98                     check_bits_position :=
99                         conv_std_logic_vector(j, check_bits_position'length);
100                    if (check_bits_position(i) = '1') then
101                        p(i) := p(i) xor codeword_ascending_range(j);
102                        codeword_ascending_range(2**i) := p(i);
103                    end if;
104                end loop;
105            end loop;
106            val <= en;
107            if (en = '1') then
108                codeword <= codeword_ascending_range;
109            end if;
110        end if;
111    end process;
112 end architecture arc_hamming_enc;
113
114
115 library ieee;
116 use ieee.std_logic_1164.all;
117 use ieee.std_logic_arith.all;
118 use ieee.std_logic_unsigned.all;
119 library work;
120 use work.hamming_pack.all;
121
122 entity hamming_dec is
123     generic (data_width : integer := 12);
124     port(
125         clk      : in  std_logic;
126         rst      : in  std_logic;
127         en       : in  std_logic;
128         codeword : in  std_logic_vector((get_codeword_size(data_width)-1) downto 0);
129         val      : out std_logic;
130         data     : out std_logic_vector((data_width-1) downto 0)
131     );
132 end entity hamming_dec;
133
134
135
136
137
138
139
140
141
142

```

```

143 architecture arc_hamming_dec of hamming_dec is
144 begin
145     process (clk, rst) is
146         variable c : std_logic_vector(0 to ((codeword'length-data'length)-1));
147         variable check_bits_position : std_logic_vector((c'length-1) downto 0);
148         variable k : integer;
149         variable data_ascending_range : std_logic_vector(1 to data'length);
150         alias codeword_ascending_range : std_logic_vector(1 to codeword'length)
151             is codeword;
152         variable codeword_fixed : std_logic_vector(1 to codeword'length);
153         variable error_position : std_logic_vector(0 to ((2**c'length)-1));
154         variable c_reversed : std_logic_vector(c'range);
155     begin
156         if rst = C_INIT then
157             data <= (others => '0');
158             val <= '0';
159         elsif rising_edge(clk) then
160             c := (others => '0');
161             for i in c'range loop
162                 for j in codeword_ascending_range'range loop
163                     check_bits_position :=
164                         conv_std_logic_vector(j, check_bits_position'length);
165                     if (check_bits_position(i) = '1') then
166                         c(i) := c(i) xor codeword_ascending_range(j);
167                     end if;
168                 end loop;
169             end loop;
170             c_reversed:=swop(c);
171             error_position := (others => '0');
172             error_position(conv_integer(c_reversed)) := '1';
173             codeword_fixed :=
174                 codeword_ascending_range xor error_position(codeword_fixed'range);
175             k:= 1;
176             for n in 1 to c'high loop
177                 for m in ((2**n)+1) to ((2**(n+1))-1) loop
178                     if k <= data'length then
179                         data_ascending_range(k) := codeword_fixed(m);
180                         k:= k+1;
181                     end if;
182                 end loop;
183             end loop;
184             val <= en;
185             if (en = '1') then
186                 data <= data_ascending_range;
187             end if;
188         end if;
189     end process;
190 end architecture arc_hamming_dec;
191
192 library ieee;
193 use ieee.std_logic_1164.all;
194 use ieee.std_logic_arith.all;
195 use ieee.std_logic_unsigned.all;
196 use ieee.math_real.all;
197 library work;
198 use work.hamming_pack.all;
199 entity hamming_enc_tb is
200 end entity hamming_enc_tb;
201 architecture arc_hamming_enc_tb of hamming_enc_tb is
202     constant data_width : integer:= 8;
203     signal clk : std_logic:= '0';
204     signal rst : std_logic:= C_INIT;
205     signal en : std_logic:= '1';
206     signal val : std_logic;
207     signal val_out : std_logic;
208     signal data : std_logic_vector((data_width-1) downto 0):=(others => '0');
209     signal dout : std_logic_vector((data_width-1) downto 0);
210     signal codeword: std_logic_vector((get_codeword_size(data_width)-1) downto 0);
211     signal data_to_recv:
212     std_logic_vector((get_codeword_size(data_width)-1) downto 0) :=
213     (others => '0');

```

```

214
215 begin
216     DUT1 : entity work.hamming_enc
217         generic map (
218             data_width => data_width)           -- [integer]
219         port map (
220             clk         => clk,                 -- [in std_logic]
221             rst         => rst,                 -- [in std_logic]
222             en          => en,                 -- [in std logic]
223             data        => data, -- [in std logic vector((data width-1) downto 0)]
224             val         => val,                 -- [out std logic]
225             codeword    => codeword);
226             -- [out std_logic_vector((get_codeword_size(data_width)-1) downto 0)]
227
228     DUT2 : entity work.hamming_dec
229         generic map (data_width => data_width)
230         port map(
231             clk         => clk,                 -- [in std_logic]
232             rst         => rst,                 -- [in std_logic]
233             en          => val,                 -- [in std logic]
234             codeword    => data_to_recv,
235             data        => dout, -- [out std_logic_vector((data_width-1) downto 0)]
236             val         => val_out);           -- [out std_logic]
237
238     -- clock generation
239     clk <= not clk      after 10 ns;
240     rst <= not C_INIT after 5 ns;
241     -- input data to transmitter generation output from tansmitter connects
242     -- to input of receiver via random error generator
243     process is
244         variable seed1      : integer:= 100;
245         variable seed2      : integer:= 105;
246         variable rand_v     : real;
247         variable bit_position : integer range data_to_recv'high downto 0 := 0;
248     begin
249         wait until rising_edge(rst);
250         data_to_recv <= codeword;
251         wait for 10 ns;
252         data         <= x"ac";
253         wait for 1 ns;
254         data_to_recv <= codeword;
255         wait until rising_edge(clk);
256         data         <= x"31";
257         wait for 1 ns;
258         data_to_recv <= codeword;
259         wait until rising_edge(clk);
260         data         <= x"12";
261         wait for 1 ns;
262         data_to_recv <= codeword;
263         wait until rising_edge(clk);
264         -- 10 packets generation with error in each one
265         for i in 0 to 9 loop
266             data         <= data + x"7";
267             wait for 1 ns;           -- wait needs to data update
268             uniform(seed1, seed2, rand_v); -- from math real library
269             -- returns value of real type with range 0.0 to 1.0
270             -- needs updating of seed1 and seed2 values for each call
271             bit_position := integer(real(codeword'high) * rand_v);
272             data_to_recv <= codeword;
273             data_to_recv(bit_position) <= not codeword(bit_position);
274             wait until rising_edge(clk);
275             -- seed to next random number
276             seed1         := seed1 + 1;
277             seed2         := seed2 + 1;
278         end loop;
279         wait;
280     end process;
281 end architecture arc_hamming_enc_tb;

```