```
    1: library STD;
    2: library IEEE;
    3: use IEEE.std_logic_1164.all;
    4:
    5: entity Counter is
    6:     port(
    7:         clk         :   in  std_logic;
    8:         hit_miss    :   in  std_logic;  --1 hit 0 miss
    9:         rd_wr       :   in  std_logic;  --1 read 0 write
   10:         start       :   in  std_logic;
   11:         Vdd         :   in  std_logic;
   12:         Gnd         :   in  std_logic;
   13:         reset       :   in  std_logic;
   14:         busy        :   out std_logic;
   15:         rd_wr_o     :   out std_logic;
   16:         cache_write :   out std_logic; --pulses for one clock to write to cach
e
   17:         rm_wr_en    :   out std_logic; --disables decoder output while writing
 from memory
   18:         wr_hit      :   out std_logic;
   19:         cpu_dout_en :   out std_logic; --enables output to data bus
   20:         mem_enable  :   out std_logic; --signals memory to start sending data
   21:         write_0     :   out std_logic;  --write word0 to cache
   22:         write_1     :   out std_logic;  --write word1 to cache
   23:         write_2     :   out std_logic;  --write word2 to cache
   24:         write_3     :   out std_logic  --write word3 to cache
   25:     );
   26: end Counter;
   27:
   28: architecture structural of Counter is
   29:
   30:     component rd_wr_hit_miss_reg is
   31:         port(
   32:             rd_wr       :   in  std_logic;
   33:             rd_wr_set_en:   in  std_logic;
   34:             hit_miss    :   in  std_logic;
   35:             hit_miss_en :   in  std_logic;
   36:             clk         :   in  std_logic;
   37:             reset       :   in  std_logic;
   38:             Gnd         :   in  std_logic;
   39:             rd_wr_o     :   out std_logic;
   40:             rd_hit      :   out std_logic;
   41:             wr_hit      :   out std_logic;
   42:             rd_miss     :   out std_logic;
   43:             wr_miss     :   out std_logic
   44:         );
   45:     end component;
   46:
   47:     component SR18
   48:     port(
   49:         clk     :   in  std_logic;
   50:         reset   :   in  std_logic;
   51:         Vdd     :   in  std_logic;
   52:         Gnd     :   in  std_logic;
   53:         output  :   out std_logic_vector(0 to 17)
   54:     );
   55:     end component;
   56:
   57:     component dff_reset
   58:     port(
   59:         d       :   in  std_logic;
   60:         clk     :   in  std_logic;
   61:         reset   :   in  std_logic;
   62:         Gnd     :   in  std_logic;
   63:         q       :   out std_logic
   64:     );
   65:     end component;
   66:
   67:     component or2
   68:     port(
   69:         in1     :   in  std_logic;
   70:         in2     :   in  std_logic;
   71:         out1    :   out std_logic
   72:     );
   73:     end component;
   74:
   75:     component or3
   76:     port(
   77:         in1     :   in  std_logic;
   78:         in2     :   in  std_logic;
   79:         in3     :   in  std_logic;
   80:         out1    :   out std_logic
   81:     );
   82:     end component;
   83:
   84:     component or5
   85:     port(
   86:         in1     :   in  std_logic;
   87:         in2     :   in  std_logic;
   88:         in3     :   in  std_logic;
   89:         in4     :   in  std_logic;
   90:         in5     :   in  std_logic;
   91:         out1    :   out std_logic
   92:     );
   93:     end component;
   94:
   95:     component and2
   96:     port(
   97:         in1     :   in  std_logic;
   98:         in2     :   in  std_logic;
   99:         out1    :   out std_logic
  100:     );
  101:     end component;
  102:
  103:     component xor2
  104:     port(
  105:         in1     :   in  std_logic;
  106:         in2     :   in  std_logic;
  107:         out1    :   out std_logic
  108:     );
  109:     end component;
  110:
  111:     component and3
  112:     port(
  113:         in1     :   in  std_logic;
  114:         in2     :   in  std_logic;
  115:         in3     :   in  std_logic;
  116:         out1    :   out std_logic
  117:     );
  118:     end component;
  119:
  120:     component invX1
  121:     port(
  122:         in1     :   in  std_logic;
```

```
123:         out1    :   out std_logic
124:     );
125:     end component;
126:
127:     component srff
128:     port(
129:         s   :   in  std_logic;
130:         r   :   in  std_logic;
131:         clk :   in  std_logic;
132:         q   :   out std_logic;
133:         qbar:   out std_logic
134:     );
135:     end component;
136:
137:     --rd_wr_hit_miss_reg output signals
138:     signal r_rd_hit     :   std_logic;
139:     signal r_wr_hit     :   std_logic;
140:     signal r_rd_miss    :   std_logic;
141:     signal r_wr_miss    :   std_logic;
142:
143:
144:     --hit miss latch signals
145:     signal hm_latch     :   std_logic;
146:     signal ncount1      :   std_logic;
147:
148:     --busy SR latch signal
149:     signal busy_in      :   std_logic;
150:
151:     --reset signals
152:     signal busy_reset   :   std_logic := '0';
153:     signal busy_reg_rst :   std_logic;
154:     signal sr_reset     :   std_logic;
155:
156:     --logic signals to set busy_reset
157:     signal wr_miss_or_rd_hit    :   std_logic;
158:     signal wr_and_clk2          :   std_logic;
159:     signal rm_and_clk18         :   std_logic;
160:     signal rh_and_clk1          :   std_logic;
161:
162:     --sr signals
163:     signal n_busy               :   std_logic;
164:     signal enable_clk           :   std_logic;
165:     signal enabled_clk          :   std_logic;
166:     signal nbusy_reg_reset      :   std_logic;
167:     signal sr_input             :   std_logic;
168:
169:     signal count        :   std_logic_vector(0 to 17);
170:
171:     --write signals
172:     signal n_1          :   std_logic;
173:     signal n_2          :   std_logic;
174:     signal n_10         :   std_logic;
175:     signal n_12         :   std_logic;
176:     signal n_14         :   std_logic;
177:     signal n_16         :   std_logic;
178:     signal n_17         :   std_logic;
179:     signal w0           :   std_logic;
180:     signal w1           :   std_logic;
181:     signal w2           :   std_logic;
182:     signal w3           :   std_logic;
183:
184:     --internal busy
185:     signal busy_internal:   std_logic := '0';
186:
187:     signal done_counting:   std_logic;
188:
189:     signal rd_operation :   std_logic;
190:     signal cdout_en     :   std_logic;
191:     signal s_mem_enable :   std_logic;
192:
193:     signal r_rd_wr      :   std_logic;
194:     signal nrd_wr       :   std_logic;
195:     signal read_miss_wr :   std_logic;
196:
197:     signal cache_wr0    :   std_logic;
198:     signal cache_wr1    :   std_logic;
199:     signal cache_wr2    :   std_logic;
200:     signal cache_wr_hit :   std_logic;
201:
202:     for rm_write0, rm_write1, SR, wrh_or_wrm, rst_busy, rst_sr, rd_op   : or2
use entity work.or2(structural);
203:     for wh_write, mem_en  :   and3 use entity work.and3(structural);
204:     for cache_wr, busy_rst, en  :   or3 use entity work.or3(structural);
205:     for wr_rm, reset_and1, reset_and2, reset_and3, enable, srin, w0out, w1out,
 w2out, w3out, dout_set_en, hml   :   and2 use entity work.and2(structural);
206:     for nc1, n2, nrw, not_busy, nbusy_rst, n10, n12, n14, n16, n17    :   invX
1 use entity work.invX1(structural);
207:     for busy_reg    :   srff use entity work.srff(structural);
208:     for counter     :   SR18 use entity work.SR18(structural);
209:     for dout_en, mem_en_reg :   dff_reset use entity work.dff_reset(structural
);
210:     for rwrhm_reg   :   rd_wr_hit_miss_reg use entity work.rd_wr_hit_miss_reg(
structural);
211:
212:
213: begin
214:
215:     busy <= busy_internal;
216:
217:     --cache write logic
218:     rm_write0   :   or2     port map(w0, w1, cache_wr0);
219:     rm_write1   :   or2     port map(w2, w3, cache_wr1);
220:     wh_write    :   and3    port map(r_wr_hit, count(1), n_2, cache_wr_hit);
221:     wr_hit <= cache_wr_hit;
222:     nrw         :   invX1   port map(r_rd_wr, nrd_wr);
223:     cache_wr    :   or3     port map(cache_wr0, cache_wr1, cache_wr_hit, cache
_write);
224:     wr_rm       :   and2    port map(count(7), n_17, rm_wr_en);
225:
226:     SR          :   or2     port map(busy_internal, start, busy_in);
227:
228:     rwrhm_reg   :   rd_wr_hit_miss_reg  port map(rd_wr, start, hit_miss, hm_la
tch, clk, reset, Gnd, r_rd_wr, r_rd_hit, r_wr_hit, r_rd_miss, r_wr_miss);
229:     rd_wr_o <= r_rd_wr;
230:     not_busy    :   invX1   port map(busy_internal, n_busy);
231:
232:     --logic for resetting busy
233:     wrh_or_wrm  :   or2     port map(r_wr_hit, r_wr_miss, wr_miss_or_rd_hit);
234:     reset_and1  :   and2    port map(wr_miss_or_rd_hit, count(1), wr_and_clk2)
;
235:     reset_and2  :   and2    port map(count(17), r_rd_miss, rm_and_clk18);
236:     reset_and3  :   and2    port map(count(0), r_rd_hit, rh_and_clk1);
237:     busy_rst    :   or3     port map(wr_and_clk2, rm_and_clk18, rh_and_clk1, b
usy_reset);
```

```
  238:
  239:     --enables counting
  240:     en          :   or3     port map(busy_internal, reset, busy_reset, enable_
clk);
  241:     enable      :   and2    port map(clk, enable_clk, enabled_clk);
  242:
  243:     --holds busy state
  244:     busy_reg    :   srff    port map(sr_input, busy_reg_rst, clk, busy_interna
l, open);
  245:
  246:     --counts clocks
  247:     counter     :   SR18    port map(enabled_clk, n_busy, Vdd, Gnd, count(0 to
 17));
  248:
  249:     --reset_signals
  250:     rst_busy    :   or2     port map(busy_reset, reset, busy_reg_rst);
  251:     rst_sr      :   or2     port map(n_busy, reset, sr_reset);
  252:
  253:     nbusy_rst   :   invX1   port map(busy_reg_rst, nbusy_reg_reset);
  254:     srin        :   and2    port map(nbusy_reg_reset, busy_in, sr_input);
  255:
  256:     --write signals
  257:     n2          :   invX1   port map(count(2), n_2);
  258:     n10         :   invX1   port map(count(10), n_10);
  259:     n12         :   invX1   port map(count(12), n_12);
  260:     n14         :   invX1   port map(count(14), n_14);
  261:     n16         :   invX1   port map(count(16), n_16);
  262:     n17         :   invX1   port map(count(17), n_17);
  263:
  264:     w0out       :   and2    port map(count(9), n_10, w0);
  265:     w1out       :   and2    port map(count(11), n_12, w1);
  266:     w2out       :   and2    port map(count(13), n_14, w2);
  267:     w3out       :   and2    port map(count(15), n_16, w3);
  268:
  269:     write_0 <=  w0;
  270:     write_1 <=  w1;
  271:     write_2 <=  w2;
  272:     write_3 <=  w3;
  273:
  274:     rd_op       :   or2     port map(r_rd_miss, r_rd_hit, rd_operation);
  275:     dout_set_en :   and2    port map(busy_reset, rd_operation, cdout_en);
  276:     dout_en     :   dff_reset   port map(cdout_en, clk, reset, Gnd, cpu_dout_e
n);
  277:
  278:     mem_en      :   and3    port map(count(0), r_rd_miss, n_1, s_mem_enable);
  279:     mem_en_reg  :   dff_reset   port map(s_mem_enable, clk, reset, Gnd, mem_en
able);
  280:
  281:     --hit_miss_latch signals
  282:     hml         :   and2    port map(count(0), n_1, hm_latch);
  283:     nc1         :   invX1   port map(count(1), n_1);
  284: end structural;
```