

```

1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 use ieee.std_logic_textio.all;
5 use std.textio.all;
6
7 entity TB_ExecUnit is
8     generic (
9         N           : natural := 64;
10        TestVectorFile : string := "TestVectors/Exec64.tvs" -- default
11    );
12 end TB_ExecUnit;
13
14 architecture behavior of TB_ExecUnit is
15
16     -- DUT ports
17     signal TBA, TBB : std_logic_vector(N-1 downto 0) := (others => '0');
18     signal TBFuncClass, TBLogicFN, TBShiftFN : std_logic_vector(1 downto 0) := (others
19        => '0');
20     signal TBAddnSub, TBExtWord : std_logic := '0';
21
22     signal TBY      : std_logic_vector(N-1 downto 0);
23     signal TBZero   : std_logic;
24     signal TBAltB   : std_logic;
25     signal TBAltBu  : std_logic;
26
27     -- Test-vector file
28     constant PreStimTime    : time    := 1 ns;
29     constant PostStimTime   : time    := 200 ns;
30     constant StableTime     : time    := 50 ns; -- Time to consider signal stable
31
32     -- Signal array for monitoring all outputs together
33     signal DUTout : std_logic_vector(N+2 downto 0);
34
35     component TestUnit is
36         generic ( N : natural := 64 );
37         port (
38             A, B : in std_logic_vector(N-1 downto 0);
39             FuncClass, LogicFN, ShiftFN : in std_logic_vector(1 downto 0);
40             AddnSub, ExtWord : in std_logic;
41             Y : out std_logic_vector(N-1 downto 0);
42             Zero, AltB, AltBu : out std_logic
43         );
44     end component;
45
46     -- Internal signals for timing measurement
47     signal start_measurement : boolean := false;
48     signal measurement_done  : boolean := false;
49
50 begin
51     -- Group all outputs for stability checking
52     DUTout <= TBY & TBZero & TBAltB & TBAltBu;
53
54     -- Device Under Test (DUT)
55     DUT: TestUnit
56         generic map ( N => N )
57         port map (
58             A          => TBA,
59             B          => TBB,
60             FuncClass => TBFuncClass,
61             LogicFN   => TBLogicFN,
62             ShiftFN   => TBShiftFN,
63             AddnSub   => TBAddnSub,
64             ExtWord   => TBExtWord,
65             Y          => TBY,
66             Zero       => TBZero,
67             AltB       => TBAltB,
68             AltBu      => TBAltBu
69         );
70
71     -- Stimulus Process with proper propagation delay measurement
72     stimulus : process
73         -- Simulation variables

```

```

73      file      tvf : text;
74      variable L, L_parse : line;
75      constant MAXLEN : natural := 2048;
76      variable s : string(1 to MAXLEN);
77
78      -- expected values from TVS file
79      variable vA, vB, vY : std_logic_vector(N-1 downto 0);
80      variable vFuncClass, vLogicFN, vShiftFN : std_logic_vector(1 downto 0);
81      variable vAddnSub, vExtWord : std_logic;
82      variable vZero, vAltB, vAltBu : std_logic;
83
84      variable skip_line : boolean;
85      variable idx       : natural := 0;
86      variable pass      : boolean;
87      variable OUTL      : line;
88
89      -- Timing measurement variables
90      variable StartTime    : time := 0 ns;
91      variable EndTime     : time := 0 ns;
92      variable PropDelay_Y : time := 0 ns;
93      variable PropDelay_Zero : time := 0 ns;
94      variable PropDelay_AltB : time := 0 ns;
95      variable PropDelay_AltBu : time := 0 ns;
96
97      -- Worst-case tracking
98      variable MaxDelay_Y : time := 0 ns;
99      variable MaxDelay_Zero : time := 0 ns;
100     variable MaxDelay_AltB : time := 0 ns;
101     variable MaxDelay_AltBu : time := 0 ns;
102
103 begin
104     -- open test vector file
105     file_open(tvf, TestVectorFile, read_mode);
106
107     -- report file name
108     report "Using test vectors from file: " & TestVectorFile;
109
110     -- loop through every test vector
111     while not endfile(tvf) loop
112         readline(tvf, L);
113
114         -- Skip blank lines
115         if L'length = 0 then
116             next;
117         end if;
118
119         skip_line := false;
120
121         -- check if line is too long
122         if L'length > MAXLEN then
123             report "Input line exceeds MAXLEN=" & integer'image(MAXLEN) severity
124             failure;
125         end if;
126
127         s := (others => ' ');
128         s(1 to L'length) := L.all;
129
130         -- Check for comments (lines starting with "--" after optional whitespace)
131         for i in s'range loop
132             if s(i) > ' ' then
133                 if i < s'high and s(i) = '-' and s(i + 1) = '-' then
134                     skip_line := true;
135                 end if;
136                 exit;
137             end if;
138         end loop;
139
140         if skip_line then
141             next;
142         end if;
143
144         -- Rebuild the line to parse values
145         L_parse := null;

```

```

145      write(L_parse, s(1 to L'length));
146
147
148      -- Parse: A, B, FuncClass, LogicFN, ShiftFN, AddnSub, ExtWord,
149      --          Y, Zero, AltB, AltBu
150      -- Assume A, B, Y are hex; control bits/flags are binary or '0'/'1'.
151
152      HREAD(L_parse, vA);
153      HREAD(L_parse, vB);
154      read (L_parse, vFuncClass);    -- 2-bit std_logic_vector
155      read (L_parse, vLogicFN);    -- 2-bit std_logic_vector
156      read (L_parse, vShiftFN);    -- 2-bit std_logic_vector
157      read (L_parse, vAddnSub);    -- std_logic
158      read (L_parse, vExtWord);    -- std_logic
159      HREAD(L_parse, vY);
160      read (L_parse, vZero);
161      read (L_parse, vAltB);
162      read (L_parse, vAltBu);
163
164      -- 1) Drive 'X' for PreStimTime
165      TBA      <= (others => 'X');
166      TBB      <= (others => 'X');
167      TBFuncClass<= (others => 'X');
168      TBLogicFN <= (others => 'X');
169      TBShiftFN <= (others => 'X');
170      TBAddnSub <= 'X';
171      TBExtWord <= 'X';
172      wait for PreStimTime;
173
174      -- 2) Apply inputs and record start time
175      TBA      <= vA;
176      TBB      <= vB;
177      TBFuncClass<= vFuncClass;
178      TBLogicFN <= vLogicFN;
179      TBShiftFN <= vShiftFN;
180      TBAddnSub <= vAddnSub;
181      TBExtWord <= vExtWord;
182
183      StartTime := now;
184
185      -- 3) Wait for outputs to become stable
186      -- Wait for any change, then wait for stability
187      wait until DUTout'event for PostStimTime;
188
189      if DUTout'event then
190          -- Signal changed, now wait for it to be stable
191          wait until DUTout'stable(StartTime) for PostStimTime;
192      end if;
193
194      EndTime := now;
195
196      -- 4) Calculate individual propagation delays
197      -- Reset delays to 0 for this measurement
198      PropDelay_Y := 0 ns;
199      PropDelay_Zero := 0 ns;
200      PropDelay_AltB := 0 ns;
201      PropDelay_AltBu := 0 ns;
202
203      -- For each output, calculate delay from last event to start time
204      -- But only if the signal changed during this measurement
205      if TBY'last_event < (EndTime - StartTime) then
206          PropDelay_Y := TBY'last_event;
207      end if;
208
209      if TBZero'last_event < (EndTime - StartTime) then
210          PropDelay_Zero := TBZero'last_event;
211      end if;
212
213      if TBAltB'last_event < (EndTime - StartTime) then
214          PropDelay_AltB := TBAltB'last_event;
215      end if;
216
217      if TBAltBu'last_event < (EndTime - StartTime) then

```

```

218 PropDelay_AltBu := TBAltBu'last_event;
219 end if;
220
221 -- Track worst-case (max) delays
222 if PropDelay_Y > MaxDelay_Y then
223     MaxDelay_Y := PropDelay_Y;
224 end if;
225 if PropDelay_Zero > MaxDelay_Zero then
226     MaxDelay_Zero := PropDelay_Zero;
227 end if;
228 if PropDelay_AltB > MaxDelay_AltB then
229     MaxDelay_AltB := PropDelay_AltB;
230 end if;
231 if PropDelay_AltBu > MaxDelay_AltBu then
232     MaxDelay_AltBu := PropDelay_AltBu;
233 end if;
234
235 -- define pass condition
236 pass := (TBY = vY) and
237     (TBZero = vZero) and
238     (TBAltB = vAltB) and
239     (TBAltBu = vAltBu);
240
241 -- 5) Compute pass/fail and assert
242 assert pass
243     report "Mismatch: i=" & integer'image(idx) &
244         " A=" & to_hstring(TBA) &
245         " B=" & to_hstring(TBB) &
246         " FuncClass=" & to_hstring(TBFuncClass) &
247         " LogicFN=" & to_hstring(TBLogicFN) &
248         " ShiftFN=" & to_hstring(TBShiftFN) &
249         " AddnSub=" & std_logic'image(TBAddnSub) &
250         " ExtWord=" & std_logic'image(TBExtWord) &
251         " got Y=" & to_hstring(TBY) &
252         " Zero=" & std_logic'image(TBZero) &
253         " AltB=" & std_logic'image(TBAltB) &
254         " AltBu=" & std_logic'image(TBAltBu) &
255         " exp Y=" & to_hstring(vY) &
256         " Zero=" & std_logic'image(vZero) &
257         " AltB=" & std_logic'image(vAltB) &
258         " AltBu=" & std_logic'image(vAltBu)
259 severity error;
260
261 -- 6) Print one concise summary line with timing information
262 OUTL := null;
263 write(OUTL, idx);
264 write(OUTL, string'(" A="));           write(OUTL, to_hstring(TBA));
265 write(OUTL, string'(" B="));           write(OUTL, to_hstring(TBB));
266 write(OUTL, string'(" FuncClass="));   write(OUTL, to_hstring(TBFuncClass));
267 );;
267 write(OUTL, string'(" LogicFN="));    write(OUTL, to_hstring(TBLogicFN));
268 write(OUTL, string'(" ShiftFN="));    write(OUTL, to_hstring(TBShiftFN));
269 write(OUTL, string'(" AddnSub="));    write(OUTL, TBAddnSub);
270 write(OUTL, string'(" ExtWord="));    write(OUTL, TBExtWord);
271 write(OUTL, string'(" | Y="));        write(OUTL, to_hstring(TBY));
272 write(OUTL, string'(" Zero="));       write(OUTL, TBZero);
273 write(OUTL, string'(" AltB="));       write(OUTL, TBAltB);
274 write(OUTL, string'(" AltBu="));      write(OUTL, TBAltBu);
275 write(OUTL, string'(" Delays(Y/Zero/AltB/AltBu)="));
276 write(OUTL, PropDelay_Y);           write(OUTL, string'("/"));
277 write(OUTL, PropDelay_Zero);        write(OUTL, string'("/"));
278 write(OUTL, PropDelay_AltB);        write(OUTL, string'("/"));
279 write(OUTL, PropDelay_AltBu);       write(OUTL, string'("/"));
280 write(OUTL, string'(" status="));
281 if pass then
282     write(OUTL, string'("PASS"));
283 else
284     write(OUTL, string'("FAIL"));
285 end if;
286 writeln(output, OUTL);
287
288     idx := idx + 1;
289 end loop;

```

```
290
291    -- Report worst-case delays at the end
292    report "Simulation completed: reached end of " & TestVectorFile;
293    report "Worst-case propagation delays - Y: "      & time'image(MaxDelay_Y) &
294        ", Zero: "   & time'image(MaxDelay_Zero) &
295        ", AltB: "   & time'image(MaxDelay_AltB) &
296        ", AltBu: "  & time'image(MaxDelay_AltBu);
297
298    file_close(tvf);
299    wait;
300 end process;
301
302 end architecture;
```