



# Vga NIOS Code - VHDL Code for the Homework Task VGA NIOS PROCESSOR.

Logic Systems And Processors (České Vysoké Učení Technické v Praze)



Scan to open on Studocu

```
1
2
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;          -- type integer and unsigned
7
8  entity DisplayLogicNios is
9  port(
10     busAddress : in std_LOGIC_vector(7 downto 0); -- Address Bus
11     busEnable : in std_logic;                      -- Enable Bus
12     byteEnable : in std_LOGIC_VECTOR(1 downto 0); -- Enable Byte
13     rw:in std_logic;
14     writeData : in std_logic_vector(15 downto 0);   -- Data
15     --data : in std_logic_vector(17 DOWNT0 0); -- position of parts
16     yrow, xcolumn : in unsigned(9 downto 0); -- row and column number of VGA video
17     VGA_CLK : in std_logic;           -- clock signal for memories
18     CLOCK_50: in std_LOGIC;
19     VGA_VS : in std_logic; -- synchronization of position values
20     ACLRN : in std_logic; -- power up initialization
21     ACK: out std_logic;
22     IRQ: out std_logic;
23     readData: out std_LOGIC_VECTOR(15 downto 0);
24     VGA_R, VGA_G, VGA_B: out std_logic_vector(9 downto 0)-- color information
25   );
26 end;
27
28 architecture behavioral of DisplayLogicNios is
29 -- Intensity of 10bit color in percent
30 constant C100 : std_logic_vector(9 downto 0) := (others=>'1');
31 constant C75 : std_logic_vector(9 downto 0) := (9=>'1', 8=>'0', others=>'1');
32 constant C50 : std_logic_vector(9 downto 0) := (9=>'0', others=>'1');
33 constant C25 : std_logic_vector(9 downto 0) := (9=>'0', 8=>'0', others=>'1');
34 constant C0 : std_logic_vector(9 downto 0) := (others=>'0');
35 constant G588 : std_logic_vector(9 downto 0) := "1001001100";
36 constant B884 : std_logic_vector(9 downto 0) := "1101110100";
37 constant R40 : std_logic_vector(9 downto 0) := "0010100000";
38 constant B22 : std_logic_vector(9 downto 0) := "0001011000";
39 constant G111 : std_logic_vector(9 downto 0) := "0110111100";
40 constant R255 : std_logic_vector(9 downto 0) := "1111111111";
41
42 -- records are VHDL equivalents of structures
43 type RGB_type is
44   record
45     R : std_logic_vector(9 downto 0);
46     G : std_logic_vector(9 downto 0);
47     B : std_logic_vector(9 downto 0);
48   end record;
49
50 constant BLUE : RGB_type := (C0,C0,C50);
51 constant GREEN : RGB_type := (C0,C50,C0);
52 constant RED : RGB_type := (C50,C0,C0);
53 constant YELLOW : RGB_type := (C75,C75,C0);
54 constant BLACK : RGB_type := (C0,C0,C0);
55 constant NAVY_BLUE: RGB_type := (C0,G588,B884);
56 constant DARK_BLUE: RGB_type := (R40,B22,G111);
57 constant WHITE: RGB_type := (R255,R255,R255);
58
59
60 constant YSIZE : integer := 240; -- height of flag
61 constant XSIZE : integer := 320; -- width of flag
62 constant XY_MAXSIZE : integer := XSIZE;
63
64 subtype xyintd is integer range 0 to XY_MAXSIZE;
65
66 constant SORGX : integer := 16; -- position of picture in the flag
```

```
67  constant SORGY : integer := 15;
68  constant SROW : integer := 128; -- memory organization
69  constant SROWCOUNT : integer := 56;
70
71  --definition of array
72  subtype xyinteger is integer range -1024 to 1023;
73  type xyPosition_type is array (0 to 7) of xyinteger;
74  signal xyPosition, xyBus : xyPosition_type;
75
76  type AvalonFSM_type is (Waiting, Reading, SetACK, Hold, Stop);
77  type FSM_type is (Waiting, SettingAddress, ProcessingData, Writing);
78  signal AvalonFSM : AvalonFSM_type := Waiting;
79  signal FSM : FSM_type := Waiting;
80
81  component Shark is
82    PORT
83    (
84      address      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
85      clock        : IN STD_LOGIC := '1';
86      q            : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
87    );
88  end component;
89
90  COMPONENT Shark2Ports IS
91    PORT
92    (
93      address_a      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
94      address_b      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
95      clock        : IN STD_LOGIC := '1';
96      q_a          : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
97      q_b          : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
98    );
99  END COMPONENT;
100
101 COMPONENT RAM IS
102   PORT
103   (
104     address      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
105     clock        : IN STD_LOGIC := '1';
106     data         : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
107     wren         : IN STD_LOGIC ;
108     q            : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
109   );
110 END COMPONENT;
111
112
113 COMPONENT BlueCurve is
114   PORT(
115     xcolumn : in unsigned(9 downto 0);
116     q       : out unsigned(6 downto 0)
117   );
118 END COMPONENT;
119
120 signal shark_address_s : STD_LOGIC_VECTOR (12 DOWNTO 0);
121 signal shark_addressY_s : STD_LOGIC_VECTOR (12 DOWNTO 0);
122 signal shark_q_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
123 signal shark_qY_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
124
125 -- array of BlueCurve
126 signal blueCurve_addressB_s, blueCurve_addressY_s : unsigned(9 downto 0);
127 signal blueCurveB_q_s, blueCurveY_q_s : unsigned(6 downto 0);
128
129 signal ramAddress : STD_LOGIC_VECTOR (12 DOWNTO 0);
130 signal ramAddressIn : STD_LOGIC_VECTOR (12 DOWNTO 0);
131 signal ramAddressOut : STD_LOGIC_VECTOR (12 DOWNTO 0);
132 signal ramDataIn : STD_LOGIC_VECTOR (15 DOWNTO 0);
```

```

133     signal ramWren : STD_LOGIC ;
134     signal ramDataOut : STD_LOGIC_VECTOR (15 DOWNTO 0);
135     signal dataDone : std_LOGIC := '0';
136
137
138 begin
139
140
141
142     ram_inst : RAM
143         port map(
144             address => ramAddress,
145             clock => CLOCK_50,
146             data => ramDataIn,
147             wren => ramWren,
148             q => ramDataOut
149         );
150
151     Shark2Ports_inst : Shark2Ports
152         PORT MAP(
153             address_a => shark_address_s, -- picture
154             address_b => shark_addressY_s, -- yellow - cut out from yellow part
155             clock => VGA_CLK,
156             q_a => shark_q_s,
157             q_b => shark_qY_s);
158
159
160     blueCB_inst : BlueCurve
161         PORT MAP(xcolumn => blueCurve_addressB_s,
162                   q => blueCurveB_q_s);
163
164     blueCY_inst : BlueCurve
165         PORT MAP(xcolumn => blueCurve_addressY_s,
166                   q => blueCurveY_q_s);
167
168
169
170     LSPflag : process(xcolumn, yrow, xyPosition, blueCurveB_q_s, blueCurveY_q_s, shark_q_s)
171     -- output of process depends on xcolumn and yrow
172     variable RGB : RGB_type; -- output colors
173     variable x0d, y0d, x1d, y1d, x2d, y2d, x3d, y3d : xyintd; -- value of xcolumn and
174     yrow inside frame, i.e., they have smaller range
175     variable x0d_outside, y0d_outside, x1d_outside, y1d_outside: boolean;
176     variable x2d_outside, y2d_outside, x3d_outside, y3d_outside: boolean; -- x3d_outside,
177     y3d_outside are not necessary
178
179     variable shark01, shark01Y, shark02, shark02Y, shark03, shark03Y, isNavyBlueInFlag,
180     isWhiteInFlag, isDarkBlueInFlag :boolean;
181
182
183     function IsShark(x,y,mode:xyintd) return boolean is
184     begin
185         case mode is
186             when 1 => return x>=SORGX and x<SORGX+SROW
187                 and y>=SORGY and y<SORGY+SROWCOUNT;
188             when 2 => return x>=(SORGX + 150) and x<(SORGX + 150)+SROW
189                 and y>=(SORGY) and y<SORGY+SROWCOUNT;
190             when 3 => return x>=(SORGX + 90) and x<(SORGX + 90)+SROW
191                 and y>=(SORGY + 160) and y<(SORGY + 160)+SROWCOUNT;
192             when others => return false;
193         end case;
194     end function;
195
196
197     function InCurve(x,y:xyintd;yoffset:unsigned) return boolean is
198     begin

```

```

195         return ((y > to_integer(yoffset) + 117) and (y < to_integer(yoffset)+ 135))  

196             or ((y > to_integer(yoffset) + 86) and (y < to_integer(yoffset)+ 104));  

197     end function;  

198  

199     function InRectangle (x,y:xyintd) return boolean is  

200     begin  

201         return (y >= 81) and (y <= 161);  

202     end function;  

203  

204     function CalculatePictureAddress (x,y,pic:xyintd;yoffset:STD_LOGIC_VECTOR) return  

205 std_logic_vector is  

206     variable address:std_logic_vector (12 DOWNTO 0);  

207     begin  

208         case pic is  

209             when 1 => address := std_logic_vector(to_unsigned((y-SORGY)*SROW + (SORGX+  
SROW-x),yoffset'LENGTH));  

210             when 2 => address := std_logic_vector(to_unsigned((y-SORGY)*SROW + (SORGX+  
SROW + 150 - x),yoffset'LENGTH));  

211             when others => address := std_logic_vector(to_unsigned((y-(SORGY+160))*SROW +  
(x-(SORGX+90)),yoffset'LENGTH));  

212         end case;  

213         return address;  

214     end function;  

215  

216     function CalculateColour (address:STD_LOGIC_VECTOR (1 DOWNTO 0)) return RGB_type is  

217     variable colour:RGB_type := YELLOW;  

218     begin  

219         if address = "00" then  

220             colour:= NAVY_BLUE;  

221         end if;  

222         return colour;  

223     end function;  

224  

225     procedure diff_limit(  

226         xy: in unsigned(9 downto 0); --input value  

227         offset: in xyinteger; --offset of position  

228         limit: in natural; -- integer>=0  

229         z: out xyintd; -- result  

230         outside:out boolean -- xy is outside of <0,limit>  

231         ) is  

232     variable diff:integer;  

233     begin  

234         diff:=to_integer(xy)-limit/2-offset;  

235  

236         if diff>=limit or diff<0 then z:=limit; outside:=true;  

237         else z:=xyintd(diff); outside:=false;  

238         end if;  

239     end procedure diff_limit;  

240  

241  

242     function "or"(color1, color2 : RGB_type) return RGB_type is  

243     variable result : RGB_type;  

244     begin  

245         if color2/=WHITE then  

246             result.R := color1.R or color2.R;  

247             result.G := color1.G or color2.G;  

248             result.B := color1.B or color2.B;  

249         else  

250             result.R := color1.R xor color2.R;  

251             result.G := color1.G xor color2.G;  

252             result.B := color1.B xor color2.B;  

253         end if;  

254         return result;  

255     end function;  

256 
```

```
257      begin
258
259      diff_limit(xcolumn, xyPosition(0), XSIZE, x0d, x0d_outside);
260      diff_limit(xcolumn, xyPosition(1), XSIZE, x1d, x1d_outside);
261      diff_limit(xcolumn, xyPosition(2), XSIZE, x2d, x2d_outside);
262      diff_limit(xcolumn, xyPosition(3), XSIZE, x3d, x3d_outside); -- out paramaters
263      can be omitted
264
265      diff_limit(yrow, xyPosition(4), YSIZE, y0d, y0d_outside);
266      diff_limit(yrow, xyPosition(5), YSIZE, y1d, y1d_outside);
267      diff_limit(yrow, xyPosition(6), YSIZE, y2d, y2d_outside);
268      diff_limit(yrow, xyPosition(7), YSIZE, y3d, y3d_outside); -- out paramaters can be
269      omitted
270
271      --green yellow blue
272      isNavyBlueInFlag := not x1d_outside and not y1d_outside;
273      isWhiteInFlag := not x0d_outside and not y0d_outside;
274      isDarkBlueInFlag := not x2d_outside and not y2d_outside;
275
276      --Drawing & Cutting Sharks
277      shark01:= IsShark(x3d,y3d,1); -- drawing picture
278      shark01Y:= IsShark(x1d,y1d,1); -- cut out of picture
279      shark02:= isShark(x3d,y3d,2); -- drawing picture
280      shark02Y:= isShark(x1d,y1d,2); -- cut out of picture
281      shark03:= isShark(x3d,y3d,3); -- drawing picture
282      shark03Y:= isShark(x1d,y1d,3); -- cut out of picture
283
284      RGB := BLACK; -- we initilize RGB color to have always a value
285
286      if isNavyBlueInFlag and (shark_qY_s = "01")
287          and (not InRectangle(x1d,y1d) and not InCurve(x1d,y1d,blueCurveB_q_s)
288      )
289          then RGB := RGB or NAVY_BLUE;
290      end if;
291
292
293      if isDarkBlueInFlag and InCurve(x2d,y2d,blueCurveB_q_s)
294          then RGB := RGB or RED;
295      end if;
296
297      if isWhiteInFlag and not InCurve(x0d,y0d,blueCurveY_q_s) and InRectangle(x0d,y0d)
298          then RGB := RGB or WHITE;
299      end if;
300
301      if (shark01 and shark_q_s /= "01") or (shark02 and shark_q_s /= "01")
302          or (shark03 and shark_q_s /= "01")
303      then
304          RGB := RGB or CalculateColour(shark_q_s);
305      end if;
306
307      if isDarkBlueInFlag then
308          blueCurve_addressB_s <= to_unsigned(x2d, blueCurve_addressB_s'LENGTH);
309
310      else
311          blueCurve_addressB_s <= (others=>'0');
312
313      end if;
314
315      if isWhiteInFlag then
316          blueCurve_addressY_s <= to_unsigned(x0d, blueCurve_addressY_s'LENGTH);
317
318      else
319          blueCurve_addressY_s <= (others=>'0')
```

```

320      end if;
321
322
323
324      if shark01 then shark_address_s <= CalculatePictureAddress (x3d,y3d,1,
325 shark_address_s);
325      elsif shark02 then shark_address_s <= CalculatePictureAddress (x3d,y3d,2,
326 shark_address_s);
326      elsif shark03 then shark_address_s <= CalculatePictureAddress (x3d,y3d,3,
327 shark_address_s);
327      else shark_address_s <=(others=>'0');
328      end if;
329
330      if shark01Y then shark_addressY_s <= CalculatePictureAddress (x1d,y1d,1,
331 shark_addressY_s);
331      elsif shark02Y then shark_addressY_s <= CalculatePictureAddress (x1d,y1d,2,
332 shark_addressY_s);
332      elsif shark03Y then shark_addressY_s <= CalculatePictureAddress (x1d,y1d,3,
333 shark_addressY_s);
333      else shark_addressY_s <=(others=>'0');
334      end if;
335
336
337      -- Copy results in RGB to outputs of entity
338      VGA_R<=RGB.R; VGA_G<=RGB.G; VGA_B<=RGB.B;
339 -----
340  end process;
341
342
343
344  avalon_write : process (CLOCK_50, ACLRN, writeData, busEnable, dataDone)
345  constant MAX : integer := 8192; -- number of words
346  variable ctr : integer range 0 to MAX := 0;
347  begin
348      if rising_edge (CLOCK_50) then
349
350          --case AvalonFSM is
351              --when Waiting =>
352                  -- ACK <= '0';
353                  --ramWren <= '0';
354                  --if (busEnable = '1') and (dataDone = '0') then -- waiting for data,
355 prepared to read
356                  --AvalonFSM <= Reading;
357                  --ramAddressIn <= std_logic_vector(to_unsigned(ctr,
358 ramAddressIn'length)); -- calculate an address for writing
359                  --ramDataIn <= writeData;
360                  --end if;
361              --when Reading =>
362                  --ACK <= '1';
363                  --ramWren <= '1';
364                  --AvalonFSM = Hold;
365                  --when Hold=> -- waiting for writing completion
366                  --AvalonFSM = Waiting;
367                  --when others =>
368                      --AvalonFSM = Waiting;
369          --end case;
370
371          if (AvalonFSM = Waiting) and (busEnable = '1') and (dataDone = '0') then --
372 waiting for data, prepared to read
373              AvalonFSM <= Reading;
374          elsif (avalonFSM = Reading) then
375              AvalonFSM <= SetACK; -- setting an acknowledge to '1'
376          elsif (avalonFSM = SetACK) then
377              AvalonFSM <= Hold;

```

```
377         elsif (avalonFSM = Hold) then
378             if (busEnable = '0') then
379                 AvalonFSM <= Stop;
380             end if;
381         elsif (avalonFSM = Stop) then
382             ACK <= '0';
383             AvalonFSM <= Waiting;
384
385             if (ctr < MAX) then
386                 ctr := ctr + 1;
387             else
388                 ctr := 0;
389                 dataDone <= '1';
390             end if;
391         end if;
392     end if;
393 end process;
394
395 fms_ram_to_bus : process (VGA_VS, CLOCK_50, ACLRN, FSM, ramDataOut)
396 variable diff : xyPosition_type;
397 variable address: std_logic_vector(12 downto 0);
398 variable coord: integer range 0 to 7 := 0;
399 variable wordsCtr : integer range 0 to 1024 := 0;
400 variable wasZero_flag : boolean := false;
401 begin
402     if ACLRN='0' then xyBus<=(others=>0);
403     elsif rising_edge(CLOCK_50) then
404         if (FSM = Waiting) then
405             if (dataDone = '1') then
406                 FSM <= SettingAddress;
407             end if;
408         elsif (FSM = SettingAddress) then
409             ramAddressOut <= std_logic_vector(to_unsigned(8 * wordsCtr + coord, address'length));
410             FSM <= ProccessingData;
411         elsif (FSM = ProccessingData) then
412             diff(coord) := to_integer(signed(ramDataOut(10 downto 0)));
413             if (coord < 7) then
414                 coord := coord + 1;
415                 FSM <= Writing;
416             else
417                 coord := 0;
418                 FSM <= SettingAddress;
419             end if;
420         elsif (FSM = Writing) and (VGA_VS = '0') then
421             wasZero_flag := false;
422         elsif (FSM = Writing) and (VGA_VS = '1') and (wasZero_flag = false) then
423             xyBus <= diff;
424             wasZero_flag := true;
425             if wordsCtr = 1024-1 then
426                 wordsCtr := 0;
427             else
428                 wordsCtr := wordsCtr + 1;
429             end if;
430             FSM <= SettingAddress;
431         end if;
432     end if;
433 end process;
434
435
436 ram_address_mux:process (ramAddressIn, ramAddressOut, dataDone)
437 begin
438     if dataDone = '0' then
439         ramAddress <= ramAddressIn;
440     else
441         ramAddress <= ramAddressOut;
```

```
442         end if;
443     end process;
444
445     vs_copy : process (VGA_VS)
446     begin
447         if rising_edge (VGA_VS) then
448             xyPosition<=xyBus;
449         end if;
450     end process;
451 end architecture behavioral;
452
```