```
1   ----------------------------------------------------------------------------
2   -- Hardware AMBA bus IP for image scaling
3   ----------------------------------------------------------------------------
4   -- Entity:      TFG_ROW_INTERPOLATION
5   -- File:        TFG_ROW_INTERPOLATION.vhd
6   -- Author:      Luis Miguel Gonzalez Berrocal
7   ----------------------------------------------------------------------------
8   -- VHDL Standard: VHDL'93
9   ----------------------------------------------------------------------------
10  -- Naming Conventions
11  -- active low signals:                  "*_n"
12  -- clock signals:                       "clk", "clk_div#", "clk_#x"
13  -- reset signals:                       "rst", "rst_n"
14  -- generics:                            "C_*"
15  -- user defined types:                  "*_TYPE"
16  -- state machine next state:            "*_ns"
17  -- state machine current state:         "*_cs"
18  -- combinatorial signals:               "*_com"
19  -- pipelined or register delay signals: "*_d#"
20  -- counter signals:                     "*_cnt*"
21  -- clock enable signals:                "*_ce"
22  -- internal version of output port:     "*_i"
23  -- device pins:                         "*_pin"
24  -- ports:                               "Names begin with Uppercase"
25  -- processes:                           "*_PROCESS"
26  -- component instantiations:            "<ENTITY_>I_<#|FUNC>"
27  -- registers:                           "*_REG"
28  ----------------------------------------------------------------------------
29
30  library ieee;
31  use ieee.std_logic_1164.all;
32  use ieee.std_logic_unsigned.all;
33  use ieee.numeric_std.all;
34
35  entity TFG_ROW_INTERPOLATION is
36        generic (ADDRESS_WIDTH            : integer := 11;
37                 ADDRESS_HEIGHT           : integer := 11;
38                 IMSE_EXTRA_PRECISION_BITS : integer := 4;
39                 INTER_RESIZE_COEF_BITS   : integer := 11;
40                 SHIFT                    : integer := 22);
41        port (Clk                    : in std_logic;
42              Rst_n                  : in std_logic;
43              Pixel_even             : in std_logic_vector (7 downto 0);
44              Pixel_odd              : in std_logic_vector (7 downto 0);
45              Dst_width              : in std_logic_vector (31 downto 0);
46              Shifted_scale_alpha    : in std_logic_vector (31 downto 0);
47              Shifted_scale_beta     : in std_logic_vector (31 downto 0);
48              Row_cnt                : in std_logic_vector (ADDRESS_HEIGHT - 1 downto 0
    );
49              Image_scaling_state    : in std_logic_vector (3 downto 0);
50              Row_interpolation_state : out std_logic_vector (2 downto 0);
51              Embedd_xofs            : out std_logic_vector (ADDRESS_WIDTH - 1 downto 0
    );
52              Embedd_yofs            : out std_logic_vector (ADDRESS_HEIGHT - 1 downto
    0);
53              Internal_address       : out std_logic_vector (ADDRESS_WIDTH - 1 downto 0
    );
```

```
54              Pixel_out               : out std_logic_vector (7 downto 0));
55  end TFG_ROW_INTERPOLATION;
56
57  architecture Behavioral of TFG_ROW_INTERPOLATION is
58
59        -- constants
60        constant UPPER : std_logic_vector (31 - INTER_RESIZE_COEF_BITS -
    IMSE_EXTRA_PRECISION_BITS - 1 downto 0) := (others => '0');
61        constant LOWER : std_logic_vector (INTER_RESIZE_COEF_BITS +
    IMSE_EXTRA_PRECISION_BITS - 1 downto 0) := (others => '0');
62        constant IMSE_RESIZE_COEF_SCALE : integer := to_integer (unsigned (UPPER & '1' &
     LOWER));
63
64        -- TFG_RESIZE_EMBB_1
65        signal alpha_even, alpha_odd : std_logic_vector (15 downto 0);
66
67        -- TFG_RESIZE_EMBB_2
68        signal beta_even, beta_odd : std_logic_vector (15 downto 0);
69
70        -- TFG_HRESIZE_KERNEL_1
71        signal pixel_hinter : std_logic_vector (31 downto 0);
72
73        -- TFG_ROW_BUFFER_1
74        signal ram_internal_enable, write_internal_enable : std_logic;
75        signal internal_address_com : std_logic_vector (ADDRESS_WIDTH - 1 downto 0);
76        signal pixel_hinter_even : std_logic_vector (31 downto 0);
77
78        -- counter
79        signal dx : std_logic_vector (ADDRESS_WIDTH - 1 downto 0);
80        signal inh_x, rst_dx_n : std_logic;
81
82        -- delay dx
83        signal dx_d1, dx_d2, dx_d3, dx_d4, dx_d5, dx_d6 : std_logic_vector (
    ADDRESS_WIDTH - 1 downto 0);
84
85        -- delay alpha
86        signal alpha_even_d1, alpha_odd_d1, alpha_even_d2, alpha_odd_d2 :
    std_logic_vector (15 downto 0);
87        signal alpha_even_d3, alpha_odd_d3 : std_logic_vector (15 downto 0);
88
89        -- state machine
90        type state_TYPE is (s0, s1, s2, s3, s4, s5, s6, s7);
91        signal current_state : state_TYPE := s0;
92
93        -- multiplexor
94        signal sel : std_logic;
95
96        component TFG_ROW_BUFFER
97        generic (SIZE : integer := 11;
98                 WORD : integer := 32);
99        port (Clk          : in  std_logic;
100             Rst_n        : in  std_logic;
101             Ram_enable   : in std_logic;
102             Write_enable : in std_logic;
103             Address      : in std_logic_vector (SIZE - 1 downto 0);
104             Data_in      : in std_logic_vector (WORD - 1 downto 0);
105             Data_out     : out std_logic_vector (WORD - 1 downto 0));
```

```
106         end component;
107
108         component TFG_RESIZE_EMBB
109         generic (SIZE                      : integer := 11;
110                  IMSE_EXTRA_PRECISION_BITS : integer := 4;
111                  INTER_RESIZE_COEF_BITS    : integer := 11);
112         port (Clk           : in std_logic;
113               Rst_n         : in std_logic;
114               D             : in std_logic_vector (SIZE - 1 downto 0);
115               Shifted_scale : in std_logic_vector (31 downto 0);
116               Embedd_ofs    : out std_logic_vector (SIZE - 1 downto 0);
117               Coef_even     : out std_logic_vector (15 downto 0);
118               Coef_odd      : out std_logic_vector (15 downto 0));
119         end component;
120
121         component TFG_HRESIZE_KERNEL
122         port (Clk              : in std_logic;
123               Rst_n            : in std_logic;
124               Pixel_even       : in std_logic_vector (7 downto 0);
125               Pixel_odd        : in std_logic_vector (7 downto 0);
126               Alpha_even       : in std_logic_vector (15 downto 0);
127               Alpha_odd        : in std_logic_vector (15 downto 0);
128               Pixel_hinter_out : out std_logic_vector (31 downto 0));
129         end component;
130
131         component TFG_VRESIZE_KERNEL
132         generic (SHIFT : integer := 22);
133         port (Clk              : in std_logic;
134               Rst_n            : in std_logic;
135               Pixel_hinter_even : in std_logic_vector (31 downto 0);
136               Pixel_hinter_odd  : in std_logic_vector (31 downto 0);
137               Beta_even        : in std_logic_vector (15 downto 0);
138               Beta_odd         : in std_logic_vector (15 downto 0);
139               Pixel_out        : out std_logic_vector (7 downto 0));
140         end component;
141
142   begin
143
144         TFG_RESIZE_EMBB_1 : TFG_RESIZE_EMBB
145         generic map (SIZE                      => ADDRESS_WIDTH,
146                      IMSE_EXTRA_PRECISION_BITS => IMSE_EXTRA_PRECISION_BITS,
147                      INTER_RESIZE_COEF_BITS    => INTER_RESIZE_COEF_BITS)
148         port map (Clk           => Clk,
149                   Rst_n         => Rst_n,
150                   D             => dx,
151                   Shifted_scale => Shifted_scale_alpha,
152                   Embedd_ofs    => Embedd_xofs,
153                   Coef_even     => alpha_even,
154                   Coef_odd      => alpha_odd);
155
156         TFG_RESIZE_EMBB_2 : TFG_RESIZE_EMBB
157         generic map (SIZE                      => ADDRESS_HEIGHT,
158                      IMSE_EXTRA_PRECISION_BITS => IMSE_EXTRA_PRECISION_BITS,
159                      INTER_RESIZE_COEF_BITS    => INTER_RESIZE_COEF_BITS)
160         port map (Clk           => Clk,
161                   Rst_n         => Rst_n,
162                   D             => Row_cnt,
```

```
163                   Shifted_scale => Shifted_scale_beta,
164                   Embedd_ofs    => Embedd_yofs,
165                   Coef_even     => beta_even,
166                   Coef_odd      => beta_odd);
167
168         TFG_HRESIZE_KERNEL_1 : TFG_HRESIZE_KERNEL
169         port map (Clk               => Clk,
170                   Rst_n             => Rst_n,
171                   Pixel_even        => Pixel_even,
172                   Pixel_odd         => Pixel_odd,
173                   Alpha_even        => alpha_even_d3,
174                   Alpha_odd         => alpha_odd_d3,
175                   Pixel_hinter_out  => pixel_hinter);
176
177         TFG_ROW_BUFFER_1 : TFG_ROW_BUFFER
178         generic map (SIZE => ADDRESS_WIDTH,
179                      WORD => 32)
180         port map (Clk          => Clk,
181                   Rst_n        => Rst_n,
182                   Ram_enable   => ram_internal_enable,
183                   Write_enable => write_internal_enable,
184                   Address      => internal_address_com,
185                   Data_in      => pixel_hinter,
186                   Data_out     => pixel_hinter_even);
187
188         TFG_VRESIZE_KERNEL_1 : TFG_VRESIZE_KERNEL
189         generic map (SHIFT => SHIFT)
190         port map (Clk               => Clk,
191                   Rst_n             => Rst_n,
192                   Pixel_hinter_even => pixel_hinter_even,
193                   Pixel_hinter_odd  => pixel_hinter,
194                   Beta_even         => beta_even,
195                   Beta_odd          => beta_odd,
196                   Pixel_out         => Pixel_out);
197
198         delay_PROCESS : process (Clk, Rst_n)
199         begin
200               if (Rst_n = '0') then
201                     dx_d1 <= (others => '0');
202                     dx_d2 <= (others => '0');
203                     dx_d3 <= (others => '0');
204                     dx_d4 <= (others => '0');
205                     dx_d5 <= (others => '0');
206                     dx_d6 <= (others => '0');
207                     alpha_even_d1 <= (others => '0');
208                     alpha_odd_d1 <= (others => '0');
209                     alpha_even_d2 <= (others => '0');
210                     alpha_odd_d2 <= (others => '0');
211                     alpha_even_d3 <= (others => '0');
212                     alpha_odd_d3 <= (others => '0');
213               elsif (Clk = '1' and Clk'event) then
214                     dx_d1 <= dx;
215                     dx_d2 <= dx_d1;
216                     dx_d3 <= dx_d2;
217                     dx_d4 <= dx_d3;
218                     dx_d5 <= dx_d4;
219                     dx_d6 <= dx_d5;
```

```
220                     alpha_even_d1 <= alpha_even;
221                     alpha_odd_d1 <= alpha_odd;
222                     alpha_even_d2 <= alpha_even_d1;
223                     alpha_odd_d2 <= alpha_odd_d1;
224                     alpha_even_d3 <= alpha_even_d2;
225                     alpha_odd_d3 <= alpha_odd_d2;
226               end if;
227         end process;
228
229         counter_PROCESS : process (Clk, rst_dx_n)
230         begin
231               if (rst_dx_n = '0') then
232                     dx <= (others => '0');
233               elsif (Clk = '1' and Clk'event) then
234                     if (inh_x = '0') then
235                           dx <= dx + 1;
236                     end if;
237               end if;
238         end process;
239
240         signals_PROCESS : process (current_state)
241         begin
242               case current_state is
243                     when s0 =>
244                           ram_internal_enable <= '0';       -- enable de ram
245                           write_internal_enable <= '0';     -- enable de escritura en ram
246                           inh_x <= '1';                     -- inhibicion de cuenta de dx
247                           sel <= '0';                       -- multiplexor de direccion
      de lectura/escritura de ram
248                           Row_interpolation_state <= "000"; -- numeracion de los
      estados de TFG_ROW_INTERPOLATION
249                           rst_dx_n <= '0';                  -- reset contador dx
250                     when s1 =>
251                           ram_internal_enable <= '0';
252                           write_internal_enable <= '0';
253                           inh_x <= '1';
254                           sel <= '1';
255                           Row_interpolation_state <= "001";
256                           rst_dx_n <= '1';
257                     when s2 =>
258                           ram_internal_enable <= '1';
259                           write_internal_enable <= '1';
260                           inh_x <= '0';
261                           sel <= '0';
262                           Row_interpolation_state <= "010";
263                           rst_dx_n <= '1';
264                     when s3 =>
265                           ram_internal_enable <= '0';
266                           write_internal_enable <= '0';
267                           inh_x <= '1';
268                           sel <= '0';
269                           Row_interpolation_state <= "011";
270                           rst_dx_n <= '0';
271                     when s4 =>
272                           ram_internal_enable <= '0';
273                           write_internal_enable <= '0';
274                           inh_x <= '1';
```

```
275                         sel <= '0';
276                         Row_interpolation_state <= "100";
277                         rst_dx_n <= '1';
278                   when s5 =>
279                         ram_internal_enable <= '1';
280                         write_internal_enable <= '0';
281                         inh_x <= '0';
282                         sel <= '1';
283                         Row_interpolation_state <= "101";
284                         rst_dx_n <= '1';
285                   when s6 =>
286                         ram_internal_enable <= '0';
287                         write_internal_enable <= '0';
288                         inh_x <= '1';
289                         sel <= '1';
290                         Row_interpolation_state <= "110";
291                         rst_dx_n <= '1';
292                   when s7 =>
293                         ram_internal_enable <= '0';
294                         write_internal_enable <= '0';
295                         inh_x <= '1';
296                         sel <= '1';
297                         Row_interpolation_state <= "111";
298                         rst_dx_n <= '0';
299             end case;
300       end process;
301
302       state_PROCESS : process (Clk, Rst_n)
303       begin
304             if (Rst_n = '0') then
305                   current_state <= s0;
306             elsif (Clk = '1' and Clk'event) then
307                   case current_state is
308                         when s0 =>                                    -- reset
309                               if (Image_scaling_state = "0001") then
310                                     current_state <= s1;
311                               else
312                                     current_state <= s0;
313                               end if;
314                         when s1 =>                                    -- espera
    para fila even
315                               if (Image_scaling_state = "0000") then
316                                     current_state <= s0;
317                               elsif (Image_scaling_state = "0100") then
318                                     current_state <= s2;
319                               else
320                                     current_state <= s1;
321                               end if;
322                         when s2 =>                                    -- obtencion
    de pseudo fila even
323                               if (dx_d5 >= Dst_width - 1) then
324                                     current_state <= s3;
325                               else
326                                     current_state <= s2;
327                               end if;
328                         when s3 =>                                    -- reset del
    contador
```

```
329                                     current_state <= s4;
330                               when s4 =>                                    -- espera
    para fila odd
331                                     if (Image_scaling_state = "0111") then
332                                           current_state <= s5;
333                                     else
334                                           current_state <= s4;
335                                     end if;
336                               when s5 =>                                    -- obtencion
    de pseudo fila odd y fila destino
337                                     if (dx_d6 >= Dst_width - 1 + 2) then
338                                           current_state <= s6;
339                                     else
340                                           current_state <= s5;
341                                     end if;
342                               when s6 =>                                    --
    sincronizacion
343                                     current_state <= s7;
344                               when s7 =>                                    -- reset del
    contador
345                                     current_state <= s1;
346                         end case;
347                   end if;
348             end process;
349
350             with sel select
351             internal_address_com <= dx_d5 when '0',
352                                     dx_d4 when '1',
353                                     (others => '0') when others;
354
355             Internal_address <= internal_address_com;
356
357       end Behavioral;
358
359
```