```
1   ------------------------------------------------------------------------------
2   --  AMBA AHB SLAVE UNIT
3   --  Author: Laurentiu Acasandrei
4   ------------------------------------------------------------------------------
5   --  This program is free software; you can redistribute it and/or modify
6   --  it under the terms of the GNU General Public License as published by
7   --  the Free Software Foundation; either version 2 of the License, or
8   --  (at your option) any later version.
9   --
10  --  This program is distributed in the hope that it will be useful,
11  --  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  --  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  --  GNU General Public License for more details.
14  --
15  --  You should have received a copy of the GNU General Public License
16  --  along with this program; if not, write to the Free Software
17  --  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
18  --  Description: Only Byte, Halfword, Word transfers are supported.
19
20  -- Bug 2. The address phases started witch one clk earlier after IP receiving
21  --      ownership of the bus. The AMBA bus was not ready for data transfer.
22  --      Solution. Delay with one clock the address phase after receiving
23  --        ownership (critical bug)
24  -- Bug 3. Reduced offset range that produced wrapping in address due to limited
25  --      range of address for the offset and no carry out was sent bus address
26  --        Solution. Making the offset and bus address adder on 32 bits so the
27  --        carry can be propagated correct (critical bug)
28  --
29
30  ------------------------------------------------------------------------------
31
32  library ieee;
33  use ieee.std_logic_1164.all;
34  library grlib;
35  use grlib.amba.all;
36  use grlib.stdlib.all; -- has the same print function
37
38
39  entity TFG_AHB_MASTER is
40    generic (
41     -- Compatibility with GRLIB
42     hindex  : integer := 0;
43      venid   : integer := 1;
44      devid   : integer := 0;
45      version : integer := 0
46    );
47    port (
48      rst_n     : in  std_ulogic;
49      clk     : in  std_ulogic;
50      ahbi : in ahb_mst_in_type;
51      ahbo : out ahb_mst_out_type;
52     --debug ports
53     transfer_type  : in std_logic_vector(2 downto 0);
54     size        : in std_logic_vector(2 downto 0);
55     start       : in std_logic;
56     addro       : in std_logic_vector(31 downto 0);
57     dwrite         : in std_logic;
```

```
58      dataw       : in std_logic_vector(31 downto 0);
59
60
61      transfer_end   : out std_logic;
62      transfer_active   : out std_logic;
63      datar       : out std_logic_vector(31 downto 0);
64      data_valid     : out std_logic;
65      beat_counter   : out std_logic_vector(9 downto 0)
66
67      );
68    end;
69
70    architecture behav of TFG_AHB_MASTER is
71
72     constant hconfig : ahb_config_type := (
73      0 => ahb_device_reg ( venid, devid, 0, version, 0),
74      others => zero32);
75
76     type ipici is record
77      transfer_type  : std_logic_vector(2 downto 0);
78      size        : std_logic_vector(2 downto 0);
79      start       : std_logic;
80      addro       : std_logic_vector(31 downto 0);
81      dwrite         : std_logic;
82      dataw       : std_logic_vector(31 downto 0);
83     end record;
84
85     type ipico is record
86      transfer_end   : std_logic;
87      transfer_active   : std_logic;
88      data_valid     : std_logic;
89      datar       : std_logic_vector(31 downto 0);
90      beat_counter   : std_logic_vector(9 downto 0);
91     end record;
92
93
94      type control_type is record
95        bus_req_detect   : std_logic;
96      data_phase  : std_logic;
97      busreq  : std_logic;
98      start_hready : std_logic; -- bug 4
99      beat_limit : std_logic_vector(9 downto 0);
100     beat_counter : std_logic_vector(9 downto 0);
101     off_set     : std_logic_vector(31 downto 0);  --  Solving bug 3.
102     mode     : std_logic;
103     grant    : std_logic; --v2: solving bug
104      end record;
105
106
107      signal    r,rin       : control_type;
108      signal ctrli    :ipici;
109      signal ctrlo    :ipico;
110
111   BEGIN
112
113    --- Gaisler  â€™two-processâ€™ design method
114    --- http://www.gaisler.com/doc/vhdl2proc.pdf
```

```
115
116     --debug in
117     ctrli.transfer_type <= transfer_type;
118     ctrli.size          <= size;
119     ctrli.start         <= start;
120     ctrli.addro         <= addro;
121     ctrli.dwrite        <= dwrite;
122     ctrli.dataw         <= dataw;
123     --debug out
124     transfer_end        <= ctrlo.transfer_end;
125     transfer_active     <= ctrlo.transfer_active;
126     datar               <= ctrlo.datar;
127     data_valid          <= ctrlo.data_valid;
128     beat_counter        <= ctrlo.beat_counter;
129
130
131
132      control_proc : process(r,rst_n,ahbi,ctrli, transfer_type)
133        variable v        : control_type;
134       --variable grant    : std_logic;  --v2: solving bug
135       variable hwdata     : std_logic_vector(31 downto 0);
136        variable hbusreq  : std_logic;
137       variable hlock   : std_logic;
138       variable hwrite     : std_logic;
139       variable haddr   : std_logic_vector(31 downto 0);
140       variable hburst     : std_logic_vector(2 downto 0);
141       --variable burst_mask : std_logic_vector(9 downto 0);
142       variable hsize   : std_logic_vector(2 downto 0);
143       variable htrans     : std_logic_vector(1 downto 0);
144
145       variable transfer_active_v : std_logic;
146       variable transfer_end_v, data_valid_v : std_logic;
147       variable beat_limit : std_logic_vector(9 downto 0);
148       variable beat_counter_v : std_logic_vector(9 downto 0);
149       variable off_set, inc_val: std_logic_vector(31 downto 0);  --  Solving bug 3.
150       variable data_phase : std_logic;
151      begin
152
153        ----------------------------------------------------------------------
154        -- Default or latched cases
155        ----------------------------------------------------------------------
156       v := r;
157       --grant  := ahbi.hgrant(hindex); --original
158       v.grant  := ahbi.hgrant(hindex); --v2: solving bug
159
160       hwdata   := (others => '0');
161       hwrite   := '0';
162       hbusreq := ctrli.start;
163       hlock    := ctrli.start;
164       haddr := (others=>'0');
165       hburst   := (others=>'0');
166       hsize := (others=>'0');
167       htrans   := (others=>'0');
168       transfer_end_v       := '0';
169
170       off_set           := ctrli.addro;  --ctrli.addro(31 downto 0); Luismi
171       inc_val           := (others=>'0');
```

```
172      beat_counter_v    := (others=>'0');
173      transfer_active_v    := r.data_phase;
174      data_valid_v      := r.data_phase and ahbi.hready;
175      data_phase        := '0';
176
177      -- if grant = '0' then  --original
178      if v.grant = '0' then  --v2: solving bug
179         v.mode   := '0';
180      end if;
181
182      case ctrli.transfer_type is
183         when "000" =>     -- Single transfer
184            --burst_mask := (others => '1');
185            beat_limit := (others => '0');
186
187         when "001" =>     -- Incrementing burst of unspecified length
188            --burst_mask := (others => '1');
189            beat_limit := (others => '1');
190
191         when "010" =>     -- 4-beat wrapping burst
192            -- if ctrli.size = "000" then       --byte transfer
193               -- burst_mask := "00"&X"03";
194            -- elsif ctrli.size = "001" then    -- half-word transfer
195               -- burst_mask := "00"&X"07";
196            -- else                         -- word transfer
197               -- burst_mask := "00"&X"0F";
198            -- end if;
199
200            beat_limit := "00"&X"03";
201
202         when "011" =>     -- 4-beat incrementing burst
203            --burst_mask := (others => '1');
204            beat_limit := "00"&X"03";
205
206         when "100" =>     -- 8-beat wrapping burst
207            -- if ctrli.size = "000" then       --byte transfer
208               -- burst_mask := "00"&X"07";
209            -- elsif ctrli.size = "001" then    -- half-word transfer
210               -- burst_mask := "00"&X"0F";
211            -- else                         -- word transfer
212               -- burst_mask := "00"&X"1F";
213            -- end if;
214
215            beat_limit := "00"&X"07";
216
217         when "101" =>     -- 8-beat incrementing burst
218            --burst_mask := (others => '1');
219            beat_limit := "00"&X"07";
220         when "110" =>     -- 16-beat wrapping burst
221            -- if ctrli.size = "000" then       --byte transfer
222               -- burst_mask := "00"&X"0F";
223            -- elsif ctrli.size = "001" then    -- half-word transfer
224               -- burst_mask := "00"&X"1F";
225            -- else                         -- word transfer
226               -- burst_mask := "00"&X"2F";
227            -- end if;
228            beat_limit := "00"&X"0F";
```

```
229
230         when "111" =>     --  16-beat incrementing burst
231            --burst_mask := (others => '1');
232            beat_limit := "00"&X"0F";
233
234         when others => -- when XX, ZZ, ZU, etc..
235            --burst_mask := (others => '1');
236            beat_limit := (others => '1');
237      end case;
238
239
240       ------------------------------------------------------------------------
241       -- Logic Control
242       ------------------------------------------------------------------------
243      if ctrli.size(1 downto 0) = "00" then           --byte   increment
244         inc_val := X"00000001";
245      elsif ctrli.size(1 downto 0) = "01"  then       --half-word increment
246         inc_val := X"00000002";
247      else                                  --word increment
248         inc_val := X"00000004";
249      end if;
250
251
252      --if grant='1' then  --v2: solving bug
253      if r.grant='1' then
254         if r.bus_req_detect = '1' then -- address phase
255            if r.start_hready = '1' and ahbi.hready = '1' then --bug 4 solved, bug 5
    solved
256               haddr := ctrli.addro;
257               hwrite := ctrli.dwrite;
258               hburst := ctrli.transfer_type;
259               hsize  := ctrli.size;
260               htrans := "10";
261               hwdata := ctrli.dataw;
262               data_phase := '1';
263               v.mode   := ctrli.dwrite;
264            end if;
265         elsif r.data_phase='1'  then     -- data phase
266
267            data_phase := '1';
268            hwdata := ctrli.dataw;
269
270            if ahbi.hready = '1' then
271               beat_counter_v := r.beat_counter + "0000000001";
272
273               if  r.beat_counter = r.beat_limit then
274                  data_phase  := '0';
275                  transfer_end_v := '1';
276               else
277                  data_phase  := ctrli.start; -- transfer ended by the master
278                  transfer_end_v := not ( ctrli.start);
279               end if;
280
281               if data_phase = '1' then      -- check if we are in data phase
282                  off_set:= r.off_set + inc_val; -- increment low part of address address
283               end if;
284
```

```
285
286            else
287               off_set:= r.off_set;
288               beat_counter_v := r.beat_counter;
289            end if;
290
291
292            if r.busreq ='1' and data_phase='1'  then
293               hburst := ctrli.transfer_type;
294            end if;
295
296            if transfer_type /="000" and r.busreq ='1' and data_phase='1'  then -- if we
    have a burst transfer
297                  hwrite := ctrli.dwrite;
298                  hsize  := ctrli.size;
299                  htrans := "11";
300                  --haddr(9 downto 0) := off_set and burst_mask;        --  Solving bug
    3.
301                  --haddr(31 downto 10) := ctrli.addro(31 downto 10);      --  Solving
    bug 3.
302                  haddr := off_set; -- Luismi --haddr(31 downto 0) :=
    off_set;                    --  Solving bug 3.
303            end if;
304
305         end if;
306      end if;
307
308
309
310      v.busreq  := hbusreq;
311      v.beat_limit := beat_limit;
312      v.beat_counter := beat_counter_v;
313      v.off_set      := off_set;
314      v.data_phase   := data_phase;
315      --v.bus_req_detect := (ctrli.start and (not r.busreq)) or (not grant); -- signals
    the address phase --original
316      v.bus_req_detect := (ctrli.start and (not data_phase)) or (not r.grant); --
    signals the address phase --v2: solving bug
317      v.start_hready  := ahbi.hready; -- bug 4 solved
318
319       ----------------------------------------------------------------------
320       -- Drive process outputs
321       ----------------------------------------------------------------------
322
323      -- two-process update
324      rin           <= v;
325
326
327
328      ahbo.hirq    <= (others => '0');
329      ahbo.hconfig <= hconfig;
330      ahbo.hindex  <= hindex;
331      ahbo.hprot   <= "0011";
332      ahbo.hbusreq <= hbusreq;
333      ahbo.hlock   <= hlock;
334      ahbo.hwdata  <= hwdata & hwdata; -- X"00000000"&hwdata;
335      ahbo.hwrite  <= hwrite;
```

```
336      ahbo.haddr   <= haddr;
337      ahbo.hburst  <= hburst;
338      ahbo.hsize   <= hsize;
339      ahbo.htrans  <= htrans;
340
341      -- debug port output
342      ctrlo.transfer_active   <= transfer_active_v;
343      ctrlo.transfer_end      <= transfer_end_v;
344      ctrlo.data_valid     <= data_valid_v;
345
346      if data_valid_v = '1'  and  r.mode ='0' then
347         ctrlo.datar          <= ahbi.hrdata (31 downto 0);
348      else
349         ctrlo.datar          <= (others=>'0');
350      end if;
351
352      ctrlo.beat_counter      <= beat_counter_v;
353
354    end process;
355
356
357
358
359    ----------------------------------------------------------------------------
360    -- Registers in system clock domain
361    ----------------------------------------------------------------------------
362    proc_clk : process(CLK)
363    begin
364      if rising_edge(CLK) then
365        r <= rin;              -- Control
366      end if;
367    end process;
368  ------------------------------------------------------------------------------
369  ------------------------------------------------------------------------------
370
371        -- pragma translate_off
372        bootmsg : report_version
373        generic map ("TFG_AHB_MASTER" &
374        ": Trabajo de Fin de Grado : Hardware AMBA bus IP for image scaling");
375        -- pragma translate_on
376
377  END;
```