```
1   -- $Id: ddl_top.vhd 428 2013-10-29 21:15:53Z jschamba $
2   ------------------------------------------------------------------------------
3   -- Title      : DDL top level design
4   -- Project    : HFT PXL
5   ------------------------------------------------------------------------------
6   -- File       : ddl_top.vhd
7   -- Author     : Joachim Schambach
8   -- Company    : University of Texas at Austin
9   -- Created    : 2012-02-16
10  -- Last update: 2014-03-04
11  -- Platform   : Windows, Xilinx ISE 13.4
12  -- Target     : Virtex-6 (XC6VLX240T-FF1759)
13  -- Standard   : VHDL'93/02
14  ------------------------------------------------------------------------------
15  -- Description: Top Level Component for the DDL interface.
16  ------------------------------------------------------------------------------
17  -- Copyright (c) 2012
18  ------------------------------------------------------------------------------
19  -- Revisions  :
20  -- Date        Version  Author          Description
21  -- 2012-02-16  1.0      jschamba        Created
22  ------------------------------------------------------------------------------
23  LIBRARY ieee;
24  USE ieee.std_logic_1164.ALL;
25  USE ieee.numeric_std.ALL;
26
27  --  Entity Declaration
28  ENTITY ddl IS
29    PORT (
30      RESET          : IN  std_logic;
31      CLK            : IN  std_logic;     -- system clock
32      -- DDL signals
33      FICLK          : IN  std_logic;     -- DDL clock
34      FITEN_n        : IN  std_logic;
35      FIDIR          : IN  std_logic;
36      FIBEN_n        : IN  std_logic;
37      FILF_n         : IN  std_logic;
38      FICTRL_n       : IN  std_logic;
39      FID            : IN  std_logic_vector(31 DOWNTO 0);
40      FOD            : OUT std_logic_vector(31 DOWNTO 0);
41      FOBSY_n        : OUT std_logic;
42      FOCTRL_n       : OUT std_logic;
43      FOTEN_n        : OUT std_logic;
44      SIU_T          : OUT std_logic;     -- BiDir signal direction
45      -- control signals
46      FEE_RESET      : OUT std_logic;
47      EVT_RD_ENABLE  : OUT std_logic;
48      -- From FPGA to PC
49      FIFO_Q         : IN  std_logic_vector(35 DOWNTO 0);  -- interface fifo data
    output port
50      FIFO_EMPTY     : IN  std_logic;     -- interface fifo "emtpy" signal
51      FIFO_RDREQ     : OUT std_logic;     -- interface fifo read request
52      FIFO_RDCLK     : OUT std_logic;     -- interface fifo read clock
53      -- From PC to FPGA
54      CMD_FIFO_Q     : OUT std_logic_vector(35 DOWNTO 0);  -- interface command fifo
    data out port
55      CMD_FIFO_EMPTY : OUT std_logic;  -- interface command fifo "emtpy" signal
```

```
56        CMD_FIFO_RDREQ : IN  std_logic      -- interface command fifo read request
57
58
59        );
60    END ddl;
61
62    -- Architecture body
63    ARCHITECTURE a OF ddl IS
64      CONSTANT CMD_FECTRL : std_logic_vector := "11000100";  -- 0xC4
65      CONSTANT CMD_FESTRD : std_logic_vector := "01000100";  -- 0x44
66      CONSTANT CMD_STBWR  : std_logic_vector := "11010100";  -- 0xD4
67      CONSTANT CMD_STBRD  : std_logic_vector := "01010100";  -- 0x54
68      CONSTANT CMD_RDYRX  : std_logic_vector := "00010100";  -- 0x14
69      CONSTANT FESTW      : std_logic_vector := "01000100";  -- 0x44
70      CONSTANT FESTW_EODB : std_logic_vector := "01100100";  -- 0x64
71
72      COMPONENT fifo36x512_DDL
73        PORT (
74          rst    : IN  std_logic;
75          wr_clk : IN  std_logic;
76          rd_clk : IN  std_logic;
77          din    : IN  std_logic_vector(35 DOWNTO 0);
78          wr_en  : IN  std_logic;
79          rd_en  : IN  std_logic;
80          dout   : OUT std_logic_vector(35 DOWNTO 0);
81          full   : OUT std_logic;
82          empty  : OUT std_logic
83          );
84      END COMPONENT;
85
86      TYPE inputState_type IS (
87        IS_IDLE,
88        IS_EVAL,
89        IS_STBWR,
90        IS_EVDATA
91        );
92
93      TYPE outputState_type IS (
94        OS_IDLE,
95        OS_WAITOUT,
96        OS_TXDATA,
97        OS_TXEMPTY,
98        OS_TXGETDATA,
99        OS_TXLF,
100       OS_TXEODB,
101       OS_TXINGAP,
102       OS_WAITIN
103       );
104
105     TYPE feebusState_type IS (
106       FB_INPUT,
107       FB_FLOAT,
108       FB_OUTPUT,
109       FB_RESET
110       );
111
112     SIGNAL outputState   : outputState_type;
```

```
113     SIGNAL inputState    : inputState_type;
114     SIGNAL feebusState   : feebusState_type;
115     SIGNAL s_areset      : std_logic;
116     SIGNAL sCmdFifoD     : std_logic_vector (35 DOWNTO 0);
117     SIGNAL sCmdFifoWrreq : std_logic;
118     SIGNAL sCmdFifoFull  : std_logic;
119     SIGNAL dout_val      : std_logic;
120     SIGNAL event_read    : boolean;
121     SIGNAL sDataEnd      : std_logic;
122     SIGNAL sFifoRdreq    : std_logic;
123     SIGNAL gap_timer     : integer RANGE 0 TO 15;
124     SIGNAL fidir_d1      : std_logic := '0';
125     SIGNAL fidir_d2      : std_logic := '0';
126     SIGNAL sFiBen_n      : std_logic := '0';
127
128     -- signals to delay inputs and outputs
129     SIGNAL sDataIn    : std_logic_vector(31 DOWNTO 0);
130     SIGNAL sDataReq   : std_logic;
131     SIGNAL sFifoEmpty : std_logic;
132     SIGNAL sFiD1      : std_logic_vector(31 DOWNTO 0);
133     SIGNAL sFiD2      : std_logic_vector(31 DOWNTO 0);
134     SIGNAL sFiTEN_n   : std_logic;
135
136   BEGIN
137     -- direction of DDL signals
138     SIU_T <= FIBEN_n OR (NOT FIDIR);
139
140     FIFO_RDREQ <= sFifoRdreq AND FILF_n;
141     FIFO_RDCLK <= FICLK;
142
143     FEE_RESET <= s_areset;
144
145     -- cmd FIFO
146     FOBSY_n   <= NOT sCmdFifoFull;
147     sCmdFifoD <= x"0" & sFiD2;
148     cmd_fifo : fifo36x512_DDL
149       PORT MAP (
150         rst    => RESET,
151         wr_clk => FICLK,
152         rd_clk => CLK,
153         din    => sCmdFifoD,
154         wr_en  => sCmdFifoWrreq,
155         rd_en  => CMD_FIFO_RDREQ,
156         dout   => CMD_FIFO_Q,
157         full   => sCmdFifoFull,
158         empty  => CMD_FIFO_EMPTY
159         );
160
161     -- Generate FEE reset:
162     -- The SIU can initiate a FEE reset by activating fiBEN_n and THEN
163     -- switching FIDIR from '1' to '0'
164     PROCESS (FICLK)
165     BEGIN
166       IF rising_edge(FICLK) THEN
167         fidir_d2 <= fidir_d1;
168         fidir_d1 <= FIDIR;
169         sFiBen_n <= FIBEN_n;
```

```
170         IF ((fidir_d2 = '1') AND (fidir_d1 = '0') AND (sFiBen_n = '0')) THEN
171           s_areset <= '1';
172         ELSE
173           s_areset <= RESET;
174         END IF;
175       END IF;
176     END PROCESS;
177
178     -- Main Process
179     main : PROCESS (FICLK, s_areset) IS
180       VARIABLE b_rxdat       : boolean;
181       VARIABLE b_rxcmd       : boolean;
182       VARIABLE command_code  : std_logic_vector (7 DOWNTO 0);
183       VARIABLE command_tid   : std_logic_vector (3 DOWNTO 0);
184       VARIABLE command_param : std_logic_vector (18 DOWNTO 0);
185
186     BEGIN
187       IF s_areset = '1' THEN              -- asynchronous reset (active high)
188         inputState  <= IS_IDLE;
189         feebusState <= FB_INPUT;
190         outputState <= OS_IDLE;
191
192         b_rxdat       := false;
193         b_rxcmd       := false;
194         command_param := (OTHERS => '0');
195         command_code  := (OTHERS => '0');
196         command_tid   := (OTHERS => '0');
197         sCmdFifoWrreq <= '0';
198         event_read    <= false;
199         EVT_RD_ENABLE <= '0';
200
201         gap_timer <= 0;
202
203         FOD        <= (OTHERS => '0');
204         FOTEN_n    <= '1';
205         FOCTRL_n   <= '1';
206         sFifoRdreq <= '0';
207         sDataIn    <= (OTHERS => '0');
208
209         sFiD1    <= (OTHERS => '0');
210         sFiD2    <= (OTHERS => '0');
211         sFiTEN_n <= '1';
212
213       ELSIF rising_edge(FICLK) THEN       -- rising clock edge
214
215         sDataIn <= FIFO_Q(31 DOWNTO 0);
216
217         sFifoEmpty <= FIFO_EMPTY;
218
219         -- delay inputs for Block Write
220         sFiD1    <= FID;
221         sFiD2    <= sFiD1;
222         sFiTEN_n <= FITEN_n OR NOT FICTRL_n;
223
224         ---------------------------------------------------------------------
225         -- INPUT State Machine
226         ---------------------------------------------------------------------
```

```
227         CASE inputState IS
228           WHEN IS_IDLE =>
229             IF (b_rxcmd) THEN
230               inputState <= IS_EVAL;
231             ELSE
232               inputState <= IS_IDLE;
233             END IF;
234
235           WHEN IS_EVAL =>
236             IF (command_code = CMD_STBWR) THEN
237               inputState <= IS_STBWR;
238             ELSIF (command_code = CMD_RDYRX) THEN
239               inputState <= IS_EVDATA;
240             ELSE
241               inputState <= IS_IDLE;
242             END IF;
243
244           WHEN IS_STBWR =>
245             IF (b_rxcmd) THEN
246               inputState <= IS_IDLE;
247             ELSE
248               inputState <= IS_STBWR;
249             END IF;
250
251           WHEN IS_EVDATA =>
252             IF (b_rxcmd OR feebusState = FB_RESET) THEN
253               inputState <= IS_IDLE;
254             ELSE
255               inputState <= IS_EVDATA;
256             END IF;
257
258         END CASE;
259
260         -- decode input
261         b_rxdat := (FIDIR = '0') AND (FITEN_n = '0') AND (FICTRL_n = '1');
262         b_rxcmd := (FIDIR = '0') AND (FITEN_n = '0') AND (FICTRL_n = '0');
263         IF (b_rxcmd) THEN
264           command_code  := FID(7 DOWNTO 0);
265           command_tid   := FID(11 DOWNTO 8);
266           command_param := FID(30 DOWNTO 12);
267         END IF;
268
269         -- Block Write
270         IF (inputState = IS_STBWR) THEN
271           sCmdFifoWrreq <= NOT sFiTEN_n;
272         ELSE
273           sCmdFifoWrreq <= '0';
274         END IF;
275
276         -- event read
277         IF (inputState = IS_EVDATA) THEN
278           EVT_RD_ENABLE <= '1';
279           event_read    <= true;
280         ELSE
281           EVT_RD_ENABLE <= '0';
282           event_read    <= false;
283         END IF;
```

```
284
285
286         ---------------------------------------------------------------------------
287         -- OUTPUT State Machine
288         ---------------------------------------------------------------------------
289         CASE outputState IS
290           WHEN OS_IDLE =>
291             FOD        <= (OTHERS => '0');
292             FOTEN_n    <= '1';
293             FOCTRL_n   <= '1';
294             sFifoRdreq <= '0';
295
296             IF (feebusState = FB_FLOAT) THEN
297               outputState <= OS_WAITOUT;
298             ELSE
299               outputState <= OS_IDLE;
300             END IF;
301
302           WHEN OS_WAITOUT =>
303             FOD        <= (OTHERS => '0');
304             FOTEN_n    <= '1';
305             FOCTRL_n   <= '1';
306             sFifoRdreq <= '0';
307
308             IF (feebusState = FB_OUTPUT) THEN
309               IF event_read THEN
310                 gap_timer   <= 0;
311                 outputState <= OS_TXEMPTY;
312               ELSE
313                 outputState <= OS_WAITIN;
314               END IF;
315             ELSIF (feebusState = FB_INPUT) THEN
316               outputState <= OS_IDLE;
317             ELSE
318               outputState <= OS_WAITOUT;
319             END IF;
320
321           WHEN OS_TXEMPTY =>
322             FOD      <= (OTHERS => '0');
323             FOTEN_n  <= '1';
324             FOCTRL_n <= '1';
325
326             IF (feebusState = FB_FLOAT OR feebusState = FB_RESET) THEN
327               sFifoRdreq  <= '0';
328               outputState <= OS_WAITIN;
329             ELSIF (FIFO_EMPTY = '1') THEN
330               sFifoRdreq  <= '0';
331               outputState <= OS_TXEMPTY;
332             ELSE
333               sFifoRdreq  <= '1';
334               outputState <= OS_TXGETDATA;
335             END IF;
336
337           WHEN OS_TXGETDATA =>
338             sFifoRdreq  <= '1';
339             outputState <= OS_TXDATA;
340
```

```
341             WHEN OS_TXDATA =>
342               FOD      <= FIFO_Q(31 DOWNTO 0);
343               FOTEN_n  <= sFifoEmpty;
344               foCTRL_N <= '1';
345
346               IF (feebusState = FB_FLOAT OR feebusState = FB_RESET) THEN
347                 sFifoRdreq  <= '0';
348                 outputState <= OS_WAITIN;
349               ELSIF FILF_n = '0' THEN
350                 FOTEN_n     <= '1';
351                 outputState <= OS_TXLF;
352               ELSIF FIFO_Q(32) = '1' THEN
353                 sFifoRdreq  <= '0';
354                 gap_timer   <= 0;
355                 outputState <= OS_TXEODB;
356               ELSIF FIFO_EMPTY = '1' THEN
357                 sFifoRdreq  <= '0';
358                 outputState <= OS_TXEMPTY;
359               ELSE
360                 sFifoRdreq  <= '1';
361                 outputState <= OS_TXDATA;
362               END IF;
363
364             WHEN OS_TXLF =>
365               FOD        <= (OTHERS => '0');
366               FOTEN_n    <= '1';
367               FOCTRL_n   <= '1';
368               sFifoRdreq <= (NOT FIFO_EMPTY) AND FILF_n;
369
370               IF (feebusState = FB_FLOAT OR feebusState = FB_RESET) THEN
371                 sFifoRdreq  <= '0';
372                 outputState <= OS_WAITIN;
373               ELSIF (FILF_n = '0') THEN
374                 sFifoRdreq  <= '0';
375                 outputState <= OS_TXLF;
376               ELSE
377                 -- re-transmit previous data word
378                 FOD         <= sDataIn;
379                 -- make sure FOTEN_n is asserted at least once
380                 -- if FIFO is empty at this point
381                 -- otherwise, FOTEN_n is asserted in OS_TXDATA
382                 FOTEN_n     <= NOT FIFO_EMPTY;
383                 outputState <= OS_TXDATA;
384               END IF;
385
386             WHEN OS_TXEODB =>
387               IF (feebusState = FB_FLOAT OR feebusState = FB_RESET) THEN
388                 FOD        <= (OTHERS => '0');
389                 FOTEN_n    <= '1';
390                 FOCTRL_n   <= '1';
391                 sFifoRdreq <= '0';
392
393                 outputState <= OS_WAITIN;
394               ELSIF (gap_timer = 1) THEN
395                 gap_timer  <= 0;
396                 FOD        <= x"000000" & FESTW_EODB;
397                 FOTEN_n    <= '0';
```

```
398                 foCTRL_N   <= '0';
399                 sFifoRdreq <= '0';
400
401                 outputState <= OS_TXINGAP;
402               ELSE
403                 FOD        <= (OTHERS => '0');
404                 FOTEN_n    <= '1';
405                 FOCTRL_n   <= '1';
406                 sFifoRdreq <= '0';
407                 gap_timer  <= gap_timer + 1;
408
409                 outputState <= OS_TXEODB;
410               END IF;
411
412             WHEN OS_TXINGAP =>
413               FOD        <= (OTHERS => '0');
414               FOTEN_n    <= '1';
415               FOCTRL_n   <= '1';
416               sFifoRdreq <= '0';
417
418               IF (feebusState = FB_FLOAT OR feebusState = FB_RESET) THEN
419                 outputState <= OS_WAITIN;
420               ELSIF (gap_timer = 15) AND ((FIFO_EMPTY = '1') OR (FIFO_Q(32) = '1')) THEN
421                 outputState <= OS_TXEMPTY;
422               ELSIF gap_timer = 15 THEN
423                 outputState <= OS_TXGETDATA;
424               ELSE
425                 gap_timer   <= gap_timer + 1;
426                 outputState <= OS_TXINGAP;
427               END IF;
428
429             WHEN OS_WAITIN =>
430               FOD        <= (OTHERS => '0');
431               FOTEN_n    <= '1';
432               FOCTRL_n   <= '1';
433               sFifoRdreq <= '0';
434
435               IF (feebusState = FB_INPUT) THEN
436                 outputState <= OS_IDLE;
437               ELSE
438                 outputState <= OS_WAITIN;
439               END IF;
440           END CASE;
441
442
443           ---------------------------------------------------------------------
444           -- FEEBUS State Machine
445           ---------------------------------------------------------------------
446           CASE feebusState IS
447             WHEN FB_INPUT =>
448               IF (fiBEN_N = '1') THEN
449                 feebusState <= FB_FLOAT;
450               ELSE
451                 feebusState <= FB_INPUT;
452               END IF;
453
454             WHEN FB_FLOAT =>
```

```
455             IF (fiBEN_N = '0') THEN
456               IF (fiDIR = '0') THEN
457                 feebusState <= FB_INPUT;
458               ELSE
459                 feebusState <= FB_OUTPUT;
460               END IF;
461             ELSE
462               feebusState <= FB_FLOAT;
463             END IF;
464
465           WHEN FB_OUTPUT =>
466             IF (fiBEN_N = '1') THEN
467               feebusState <= FB_FLOAT;
468             ELSIF (fiDIR = '0') THEN
469               feebusState <= FB_RESET;
470             ELSE
471               feebusState <= FB_OUTPUT;
472             END IF;
473
474           WHEN FB_RESET =>
475             feebusState <= FB_INPUT;
476
477         END CASE;
478
479       END IF;
480     END PROCESS main;
481
482   END a;
483
```