

```

1  -----
2  -- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
3  -- Institut REDS, Reconfigurable & Embedded Digital Systems
4  --
5  -- File      : axi4lite_slave.vhd
6  -- Author    : E. Messerli    27.07.2017
7  -- Description : slave interface AXI (without burst)
8  -- used for   : SOCF lab
9  --| Modifications |-----
10 -- Ver  Date      Auteur  Description
11 -- 1.0  26.03.2019  EMI     Adaptation du chablon pour les etudiants
12 -- 1.1  03.04.2020  ISS     Complète le chablon pour le laboratoire 5
13 -----
14
15 library ieee;
16     use ieee.std_logic_1164.all;
17     use ieee.numeric_std.all;
18
19 entity axi4lite_slave is
20     generic (
21         -- Users to add parameters here
22
23         -- User parameters ends
24
25         -- Width of S_AXI data bus
26         AXI_DATA_WIDTH  : integer    := 32; -- 32 or 64 bits
27         -- Width of S_AXI address bus
28         AXI_ADDR_WIDTH  : integer    := 12
29     );
30     port (
31         -- AXI4-Lite
32         axi_clk_i      : in  std_logic;
33         axi_reset_i     : in  std_logic;
34
35         -- Write Address Channel
36         axi_awaddr_i    : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
37         axi_awprot_i     : in  std_logic_vector( 2 downto 0); -- not used
38         axi_awvalid_i    : in  std_logic;
39         axi_awready_o    : out std_logic;
40
41         -- Write Data Channel
42         axi_wdata_i      : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
43         axi_wstrb_i      : in  std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
44         axi_wvalid_i     : in  std_logic;
45         axi_wready_o     : out std_logic;
46
47         -- Write Response Channel
48         axi_bresp_o      : out std_logic_vector(1 downto 0);
49         axi_bvalid_o     : out std_logic;
50         axi_bready_i     : in  std_logic;
51
52         -- Read Address Channel
53         axi_araddr_i     : in  std_logic_vector(AXI_ADDR_WIDTH-1 downto 0);
54         axi_arprot_i     : in  std_logic_vector( 2 downto 0); -- not used
55         axi_arvalid_i    : in  std_logic;
56         axi_arready_o    : out std_logic;
57
58         -- Read Data Channel
59         axi_rdata_o      : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
60         axi_rresp_o      : out std_logic_vector(1 downto 0);
61         axi_rvalid_o     : out std_logic;
62         axi_rready_i     : in  std_logic;
63
64         -- User input-output
65         switch_i         : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
66         key_i            : in  std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
67
68         leds_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
69

```

```

70         hex0_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
71         hex1_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
72         hex2_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
73         hex3_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
74         hex4_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
75         hex5_o           : out std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
76
77
78         -- Interruption
79         irq_o             : out std_logic
80     );
81 end entity axi4lite_slave;
82
83 architecture rtl of axi4lite_slave is
84
85     signal reset_s : std_logic;
86
87     -- local parameter for addressing 32 bit / 64 bits, cst: AXI_DATA_WIDTH
88     -- ADDR_LSB is used for addressing word 32/64 bits registers/memories
89     -- ADDR_LSB = 2 for 32 bits (n-1 downto 2)
90     -- ADDR_LSB = 3 for 64 bits (n-1 downto 3)
91     constant ADDR_LSB      : integer := (AXI_DATA_WIDTH/32)+ 1;
92
93     ----- SIGNAUX AXI 4 LIGHT -----
94
95     --signal for the AXI slave
96     --intern signal for output
97     signal axi_awready_s    : std_logic;
98     signal axi_arready_s    : std_logic;
99
100    signal axi_wready_s      : std_logic;
101    signal axi_rready_s      : std_logic;
102
103    signal axi_rvalid_s      : std_logic;
104    signal axi_rresp_s       : std_logic_vector(1 downto 0);
105    signal axi_rdata_mem_s   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
106
107    -- write enable
108    signal axi_data_wren_s    : std_logic;
109
110    --intern signal for the axi interface
111    signal axi_waddr_mem_s    : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
112    signal axi_araddr_mem_s   : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
113
114    signal axi_wdata_mem_s    : std_logic_vector(AXI_DATA_WIDTH-1 downto 0);
115    signal axi_wstrb_mem_s    : std_logic_vector((AXI_DATA_WIDTH/8)-1 downto 0);
116    -- signal axi_araddr_mem_s : std_logic_vector(AXI_ADDR_WIDTH-1 downto ADDR_LSB);
117
118    signal axi_bresp_s        : std_logic_vector(1 downto 0);
119    signal axi_bvalid_s       : std_logic;
120
121
122    ----- SIGNAUX ENTREES / SORTIES -----
123
124    constant registre_cst_mem : std_logic_vector(AXI_DATA_WIDTH-1 downto 0) :=
x"deedbeef";
125    signal registre_test_mem   : std_logic_vector(AXI_DATA_WIDTH-1 downto 0) :=
x"12345678";
126
127    -- signal for registre input (switch / key)
128    signal registre_switch_mem : std_logic_vector(9 downto 0) := (others => 'X');
129    signal registre_key_mem     : std_logic_vector(3 downto 0) := (others => 'X');
130
131    -- signal for registre leds
132    signal registre_led_mem     : std_logic_vector(9 downto 0) := (others => 'X');
133
134    -- signal for registre 7 seg
135    signal registre_hex0_mem    : std_logic_vector(6 downto 0) := (others => 'X');
136    signal registre_hex1_mem    : std_logic_vector(6 downto 0) := (others => 'X');

```

```

137     signal registre_hex2_mem      : std_logic_vector(6 downto 0) := (others => 'X');
138     signal registre_hex3_mem      : std_logic_vector(6 downto 0) := (others => 'X');
139     signal registre_hex4_mem      : std_logic_vector(6 downto 0) := (others => 'X');
140     signal registre_hex5_mem      : std_logic_vector(6 downto 0) := (others => 'X');
141
142     ----- SIGNAUX GESTION IRQ -----
143     signal irq_s                   : std_logic;
144     signal irq_source              : std_logic_vector(3 downto 0) := (others => '0');
145     signal key_val_save            : std_logic_vector(3 downto 0) := (others => '1');
146     -- par défaut, toutes les irq actives
147     signal key_irq_mask            : std_logic_vector(3 downto 0) := (others => '0');
148
149 begin
150
151     -- mise à jour des entrées
152     reset_s <= axi_reset_i;
153
154     registre_switch_mem <= switch_i(9 downto 0);
155     registre_key_mem    <= key_i(3 downto 0);
156
157
158
159     -----
160     -- Write address channel
161
162     process (reset_s, axi_clk_i)
163     begin
164         -- En cas de reset
165         if reset_s = '1' then
166             -- Valeur par défaut
167             axi_awready_s <= '0';
168             axi_waddr_mem_s <= (others => '0');
169         elsif rising_edge(axi_clk_i) then
170             -- Si une adresse d'écriture est valide
171             if (axi_awready_s = '0' and axi_awvalid_i = '1') then --and axi_wvalid_i =
172                 '1') then modif EMI 10juil2018
173                 -- slave is ready to accept write address when
174                 -- there is a valid write address
175                 axi_awready_s <= '1';
176                 -- Write Address memorizing
177                 axi_waddr_mem_s <= axi_awaddr_i (AXI_ADDR_WIDTH-1 downto ADDR_LSB);
178             else
179                 axi_awready_s <= '0';
180                 axi_waddr_mem_s <= (others => '0');
181             end if;
182         end if;
183     end process;
184     axi_awready_o <= axi_awready_s;
185
186     -----
187     -- Write data channel
188
189     -- Implement axi_wready generation
190     process (reset_s, axi_clk_i)
191     begin
192         -- En cas de reset
193         if reset_s = '1' then
194             -- Valeur par défaut
195             axi_wready_s <= '0';
196             axi_wdata_mem_s <= (others => '0');
197             axi_wstrb_mem_s <= (others => '0');
198         elsif rising_edge(axi_clk_i) then
199             -- Si les données d'écriture est valide
200             if (axi_wready_s = '0' and axi_wvalid_i = '1') then
201                 -- slave is ready to accept write data when
202                 -- there is a valid write data
203                 axi_wready_s <= '1';
204

```

```

205         -- Read axi_wstrb_i
206         axi_wstrb_mem_s <= axi_wstrb_i((AXI_DATA_WIDTH/8)-1 downto 0);
207
208
209         -- Mémorisation des données à écrire en fonction du paramètre strobe
210         axi_wdata_mem_s <= (others => '0');
211
212         if (axi_wstrb_i(0) = '1') then
213             axi_wdata_mem_s(7 downto 0) <= axi_wdata_i(7 downto 0);
214         end if;
215         if (axi_wstrb_i(1) = '1') then
216             axi_wdata_mem_s(15 downto 8) <= axi_wdata_i(15 downto 8);
217         end if;
218         if (axi_wstrb_i(2) = '1') then
219             axi_wdata_mem_s(23 downto 16) <= axi_wdata_i(23 downto 16);
220         end if;
221         if (axi_wstrb_i(3) = '1') then
222             axi_wdata_mem_s(31 downto 24) <= axi_wdata_i(31 downto 24);
223         end if;
224
225         -- Test sans la fonctionnalité strobe
226         -- axi_wdata_mem_s <= axi_wdata_i;
227
228     else
229         axi_wready_s <= '0';
230         axi_wdata_mem_s <= (others => '0');
231         axi_wstrb_mem_s <= (others => '0');
232
233     end if;
234 end if;
235 end process;
236
237 -- Met à jour la sortie
238 axi_wready_o <= axi_wready_s;
239
240
241 -- condition to write data : si on est prêt à écrire
242 axi_data_wren_s <= '1' when axi_wready_s = '1' else
243     '0';
244
245
246 process (reset_s, axi_clk_i)
247     --number address to access 32 or 64 bits data
248     variable int_waddr_v : natural;
249 begin
250     if reset_s = '1' then
251         -- Valeur par défaut : RESET
252         registre_test_mem <= x"12345678";
253         registre_led_mem <= "0101010101";
254         registre_hex0_mem <= "1000000" ;
255         registre_hex1_mem <= "1111001";
256         registre_hex2_mem <= "0100100";
257         registre_hex3_mem <= "0110000";
258         registre_hex4_mem <= "0011001";
259         registre_hex5_mem <= "0010010";
260
261         key_irq_mask <= "0000";
262
263     elsif rising_edge(axi_clk_i) then
264         -- Si une écriture est active
265         if axi_data_wren_s = '1' then
266             -- convertie l'adresse d'écriture en integer
267             int_waddr_v := to_integer(unsigned(axi_waddr_mem_s));
268             case int_waddr_v is
269                 -- offset 0 : constante
270                 when 0 =>
271                     -- offset 4 : registre de test
272                 when 1 =>
273                     registre_test_mem <= axi_wdata_mem_s;

```

```

274
275         -- offset 64 : leds
276         when 64 =>
277             registre_led_mem <= axi_wdata_mem_s(9 downto 0);
278
279         -- offset 130 : mask irq key
280         when 130 =>
281             key_irq_mask <= axi_wdata_mem_s(3 downto 0);
282
283         -- offset 256 - 276 : afficheur 7 seg
284         when 256 =>
285             registre_hex0_mem <= axi_wdata_mem_s(6 downto 0);
286         when 260 =>
287             registre_hex1_mem <= axi_wdata_mem_s(6 downto 0);
288         when 264 =>
289             registre_hex2_mem <= axi_wdata_mem_s(6 downto 0);
290         when 268 =>
291             registre_hex3_mem <= axi_wdata_mem_s(6 downto 0);
292         when 272 =>
293             registre_hex4_mem <= axi_wdata_mem_s(6 downto 0);
294         when 276 =>
295             registre_hex5_mem <= axi_wdata_mem_s(6 downto 0);
296
297
298         when others => null;
299     end case;
300 end if;
301 end if;
302 end process;
303
304

```

```

305 -----
306 -- Write response channel
307

```

```

308     process (reset_s, axi_clk_i)
309     begin
310         -- En cas de reset
311         if reset_s = '1' then
312             -- Valeur par défaut
313             axi_bresp_s <= "00";
314             axi_bvalid_s <= '0';
315         elsif rising_edge(axi_clk_i) then
316             -- Si le master est prêt à lire la réponse
317             if (axi_bvalid_s = '0' and axi_bready_i = '1') then
318                 -- slave is ready to accept write data when
319                 -- there is a valid write data
320                 axi_bvalid_s <= '1';
321                 -- Write response
322                 axi_bresp_s <= "00";
323             else
324                 axi_bvalid_s <= '0';
325                 axi_bresp_s <= "--";
326             end if;
327         end if;
328     end if;
329 end process;
330 -- Met à jours les sorties
331 axi_bresp_o <= axi_bresp_s;
332 axi_bvalid_o <= axi_bvalid_s;
333
334

```

```

336 -----
337 -- Read address channel
338

```

```

339     process (reset_s, axi_clk_i)
340     begin
341         -- en cas de reset
342         if reset_s = '1' then

```

```

343         -- valeur par défaut
344         axi_arready_s    <= '0';
345         axi_araddr_mem_s <= (others => '1');
346     elsif rising_edge(axi_clk_i) then
347         -- Si une adresse de lecture est valide
348         if axi_arready_s = '0' and axi_arvalid_i = '1' then
349             -- indicates that the slave has accepted the valid read address
350             axi_arready_s    <= '1';
351             -- Read Address memorizing
352             axi_araddr_mem_s <= axi_araddr_i (AXI_ADDR_WIDTH-1 downto ADDR_LSB);
353         else
354             axi_arready_s    <= '0';
355         end if;
356     end if;
357 end process;
358 -- Met à jour la sortie
359 axi_arready_o <= axi_arready_s;
360
361 -----
362 -- Read data channel
363
364 -- Implement axi_wready generation
365 process (reset_s, axi_clk_i)
366     --number address to access 32 or 64 bits data
367     variable int_raddr_v : natural;
368 begin
369
370     -- En cas de reset
371     if reset_s = '1' then
372         -- valeur par défaut
373         axi_rvalid_s    <= '0';
374         axi_rdata_mem_s <= (others => '0');
375         axi_rresp_s     <= "00";
376
377         irq_source <= "0000";
378         irq_s <= '0';
379
380     elsif rising_edge(axi_clk_i) then
381         -- Gestion des interruptions
382         if (key_val_save(0) /= registre_key_mem(0) and registre_key_mem(0) = '0'
383             and key_irq_mask(0) = '0') then
384             irq_source(0) <= '1';
385             irq_s <= '1';
386         elsif (key_val_save(1) /= registre_key_mem(1) and registre_key_mem(1) = '0'
387             and key_irq_mask(1) = '0') then
388             irq_source(1) <= '1';
389             irq_s <= '1';
390         elsif (key_val_save(2) /= registre_key_mem(2) and registre_key_mem(2) = '0'
391             and key_irq_mask(2) = '0') then
392             irq_source(2) <= '1';
393             irq_s <= '1';
394         elsif (key_val_save(3) /= registre_key_mem(3) and registre_key_mem(3) = '0'
395             and key_irq_mask(3) = '0') then
396             irq_source(3) <= '1';
397             irq_s <= '1';
398         end if;
399         -- Met à jour l'ancienne valeur des keys
400         key_val_save <= registre_key_mem;
401
402         -- Si une lecture est faite
403         if (axi_arready_s = '1' and axi_rvalid_s = '0') then
404             -- Pré-charge une lecture à 0
405             axi_rdata_mem_s <= (others => '0');
406
407             -- slave is ready to accept write data when
408             -- there is a valid write data
409             axi_rvalid_s <= '1';

```

```

408         -- read Data go
409         int_raddr_v      := to_integer(unsigned(axi_araddr_mem_s));
410         axi_rresp_s      <= "00";
411
412         -- En fonction de l'adresse qu'on souhaite lire
413         case int_raddr_v is
414             -- Lecture de la constante
415             when 0      =>
416                 axi_rdata_mem_s <= registre_cst_mem;
417             -- Lecture du registre de test
418             when 1      =>
419                 axi_rdata_mem_s <= registre_test_mem;
420             -- Lecture des leds
421             when 64      =>
422                 axi_rdata_mem_s(9 downto 0) <= registre_led_mem;
423             -- Lecture des keys
424             when 128     =>
425                 axi_rdata_mem_s(3 downto 0) <= registre_key_mem;
426             -- lecture de la source d'interruption et acquitement
427             when 129     =>
428                 axi_rdata_mem_s(3 downto 0) <= irq_source;
429                 irq_s <= '0';
430                 irq_source <= "0000";
431
432             -- lecture des masque des irq
433             when 130     =>
434                 axi_rdata_mem_s(3 downto 0) <= key_irq_mask;
435             -- Lecture des switches
436             when 192     =>
437                 axi_rdata_mem_s(9 downto 0) <= registre_switch_mem;
438
439             -- Lecture d'un afficheur 7 seg (256 - 276)
440             when 256     =>
441                 axi_rdata_mem_s(6 downto 0) <= registre_hex0_mem;
442             when 260     =>
443                 axi_rdata_mem_s(6 downto 0) <= registre_hex1_mem;
444             when 264     =>
445                 axi_rdata_mem_s(6 downto 0) <= registre_hex2_mem;
446             when 268     =>
447                 axi_rdata_mem_s(6 downto 0) <= registre_hex3_mem;
448             when 272     =>
449                 axi_rdata_mem_s(6 downto 0) <= registre_hex4_mem;
450             when 276     =>
451                 axi_rdata_mem_s(6 downto 0) <= registre_hex5_mem;
452
453
454             when others =>
455                 axi_rresp_s      <= "00";
456         end case;
457
458         else
459             axi_rvalid_s <= '0';
460             axi_rresp_s  <= "--";
461
462         end if;
463     end if;
464 end process;
465
466 -- Mise à jour de la ligne l'interruption
467 irq_o <= irq_s;
468
469 -- Mise à jour de la validité de lecture
470 axi_rvalid_o <= axi_rvalid_s;
471
472 -- Mise à jour des données lues
473 axi_rdata_o <= axi_rdata_mem_s;
474
475 -- Mise à jour de la réponse de lecture
476 axi_rresp_o <= axi_rresp_s;

```

```
477
478
479     -- Mise à jour des sorties
480     leds_o(9 downto 0)      <= registre_led_mem;
481
482     hex0_o(6 downto 0)      <= registre_hex0_mem;
483     hex1_o(6 downto 0)      <= registre_hex1_mem;
484     hex2_o(6 downto 0)      <= registre_hex2_mem;
485     hex3_o(6 downto 0)      <= registre_hex3_mem;
486     hex4_o(6 downto 0)      <= registre_hex4_mem;
487     hex5_o(6 downto 0)      <= registre_hex5_mem;
488
489
490 end rtl;
491
```



```

1  -----
2  --
3  -- HEIG-VD
4  -- Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  -- School of Business and Engineering in Canton de Vaud
6  -----
7  --
8  -- REDS Institute
9  -- Reconfigurable Embedded Digital Systems
10 -----
11 --
12 -- File           : DE1_SoC_top.vhd
13 -- Author          : Sébastien Masle
14 -- Date            : 17.01.2018
15 --
16 -- Context         : HPA
17 -----
18 --
19 -- Description : top design for DE1-SoC board
20 -----
21 --
22 -- Dependencies :
23 -----
24 --
25 -- Modifications :
26 -- Ver    Date      Engineer    Comments
27 -- 0.0    17.01.2018 SMS         Initial version.
28 -----
29
30
31 library ieee;
32 use ieee.std_logic_1164.all;
33
34 entity DE1_SoC_top is
35     port (
36         -- clock pins
37         CLOCK_50_i : in std_logic;
38         CLOCK2_50_i : in std_logic;
39         CLOCK3_50_i : in std_logic;
40         CLOCK4_50_i : in std_logic;
41
42         -- ADC
43         ADC_CS_N_o : out std_logic;
44         ADC_DIN_o  : out std_logic;
45         ADC_DOUT_i : in std_logic;
46         ADC_SCLK_o : out std_logic;
47
48         -- Audio
49         AUD_ADCLRCK_io : inout std_logic;
50         AUD_ADCDATA_i  : in std_logic;
51         AUD_DACLCK_io  : inout std_logic;
52         AUD_DACDATA_o  : out std_logic;
53         AUD_XCK_o      : out std_logic;
54         AUD_BCLK_io    : inout std_logic;
55
56         -- SDRAM
57         DRAM_ADDR_o : out std_logic_vector(12 downto 0);
58         DRAM_BA_o   : out std_logic_vector(1 downto 0);
59         DRAM_CAS_N_o : out std_logic;
60         DRAM_CKE_o   : out std_logic;
61         DRAM_CLK_o   : out std_logic;
62         DRAM_CS_N_o  : out std_logic;
63         DRAM_DQ_io   : inout std_logic_vector(15 downto 0);
64         DRAM_LDQM_o  : out std_logic;
65         DRAM_RAS_N_o : out std_logic;
66         DRAM_UDQM_o  : out std_logic;

```

```

63     DRAM_WE_N_o      : out std_logic;
64
65     --I2C Bus for Configuration of the Audio and Video-In Chips
66     FPGA_I2C_SCLK_o   : out std_logic;
67     FPGA_I2C_SDAT_io  : inout std_logic;
68
69     -- 40-pin headers
70     GPIO_0_io         : inout std_logic_vector(35 downto 0);
71     GPIO_1_io         : inout std_logic_vector(35 downto 0);
72
73     -- Seven Segment Displays
74     HEX0_o            : out std_logic_vector(6 downto 0);
75     HEX1_o            : out std_logic_vector(6 downto 0);
76     HEX2_o            : out std_logic_vector(6 downto 0);
77     HEX3_o            : out std_logic_vector(6 downto 0);
78     HEX4_o            : out std_logic_vector(6 downto 0);
79     HEX5_o            : out std_logic_vector(6 downto 0);
80
81     -- IR
82     IRDA_RXD_i        : in std_logic;
83     IRDA_TXD_o        : out std_logic;
84
85     -- Pushbuttons
86     KEY_i             : in std_logic_vector(3 downto 0);
87
88     -- LEDs
89     LEDR_o            : out std_logic_vector(9 downto 0);
90
91     -- PS2 Ports
92     PS2_CLK_io        : inout std_logic;
93     PS2_DAT_io        : inout std_logic;
94     PS2_CLK2_io       : inout std_logic;
95     PS2_DAT2_io       : inout std_logic;
96
97     -- Slider Switches
98     SW_i              : in std_logic_vector(9 downto 0);
99
100    -- Video-In
101    TD_CLK27_i         : in std_logic;
102    TD_DATA_i          : in std_logic_vector(7 downto 0);
103    TD_HS_i            : in std_logic;
104    TD_RESET_N_o       : out std_logic;
105    TD_VS_i            : in std_logic;
106
107    -- VGA
108    VGA_R_o            : out std_logic_vector(7 downto 0);
109    VGA_G_o            : out std_logic_vector(7 downto 0);
110    VGA_B_o            : out std_logic_vector(7 downto 0);
111    VGA_CLK_o          : out std_logic;
112    VGA_SYNC_N_o       : out std_logic;
113    VGA_BLANK_N_o      : out std_logic;
114    VGA_HS_o           : out std_logic;
115    VGA_VS_o           : out std_logic;
116
117    -- DDR3 SDRAM
118    HPS_DDR3_ADDR_o     : out std_logic_vector(14 downto 0);
119    HPS_DDR3_BA_o       : out std_logic_vector(2 downto 0);
120    HPS_DDR3_CAS_N_o    : out std_logic;
121    HPS_DDR3_CKE_o      : out std_logic;
122    HPS_DDR3_CK_N_o     : out std_logic;
123    HPS_DDR3_CK_P_o     : out std_logic;
124    HPS_DDR3_CS_N_o     : out std_logic;
125    HPS_DDR3_DM_o       : out std_logic_vector(3 downto 0);
126    HPS_DDR3_DQ_io      : inout std_logic_vector(31 downto 0);
127    HPS_DDR3_DQS_N_io   : inout std_logic_vector(3 downto 0);
128    HPS_DDR3_DQS_P_io   : inout std_logic_vector(3 downto 0);
129    HPS_DDR3_ODT_o      : out std_logic;
130    HPS_DDR3_RAS_N_o    : out std_logic;
131    HPS_DDR3_RESET_N_o  : out std_logic;

```

```

132 HPS_DDR3_RZQ_i      : in std_logic;
133 HPS_DDR3_WE_N_o     : out std_logic;
134
135 -- Ethernet
136 --HPS_ENET_GTX_CLK_o : out std_logic;
137 --HPS_ENET_INT_N_io  : inout std_logic;
138 --HPS_ENET_MDC_o     : out std_logic;
139 --HPS_ENET_MDIO_io   : inout std_logic;
140 --HPS_ENET_RX_CLK_i  : in std_logic;
141 --HPS_ENET_RX_DATA_i : in std_logic_vector(3 downto 0);
142 --HPS_ENET_RX_DV_i   : in std_logic;
143 --HPS_ENET_TX_DATA_o : out std_logic_vector(3 downto 0);
144 --HPS_ENET_TX_EN_o   : out std_logic;
145
146 -- Flash
147 --HPS_FLASH_DATA_io  : inout std_logic_vector(3 downto 0);
148 --HPS_FLASH_DCLK_o   : out std_logic;
149 --HPS_FLASH_NCSO_o   : out std_logic;
150
151 -- Accelerometer
152 --HPS_GSENSOR_INT_io : inout std_logic;
153
154 -- General Purpose I/O
155 --HPS_GPIO_io        : inout std_logic_vector(1 downto 0);
156
157 -- I2C
158 --HPS_I2C_CONTROL_io : inout std_logic;
159 --HPS_I2C1_SCLK_io   : inout std_logic;
160 --HPS_I2C1_SDAT_io   : inout std_logic;
161 --HPS_I2C2_SCLK_io   : inout std_logic;
162 --HPS_I2C2_SDAT_io   : inout std_logic;
163
164 -- Pushbutton
165 HPS_KEY_io           : inout std_logic;
166
167 -- LED
168 HPS_LED_io           : inout std_logic;
169
170 -- SD Card
171 --HPS_SD_CLK_o        : out std_logic;
172 --HPS_SD_CMD_io       : inout std_logic;
173 --HPS_SD_DATA_io      : inout std_logic_vector(3 downto 0);
174
175 -- SPI
176 --HPS_SPIM_CLK_o      : out std_logic;
177 --HPS_SPIM_MISO_i     : in std_logic;
178 --HPS_SPIM_MOSI_o     : out std_logic;
179 --HPS_SPIM_SS_io      : inout std_logic;
180
181 -- UART
182 --HPS_UART_RX_i       : in std_logic;
183 --HPS_UART_TX_o       : out std_logic;
184
185 -- USB
186 --HPS_CONV_USB_N_io   : inout std_logic;
187 --HPS_USB_CLKOUT_i    : in std_logic;
188 --HPS_USB_DATA_io     : inout std_logic_vector(7 downto 0);
189 --HPS_USB_DIR_i       : in std_logic;
190 --HPS_USB_NXT_i       : in std_logic;
191 --HPS_USB_STP_o       : out std_logic;
192
193 -- LTC connector
194 --HPS_LTC_GPIO_io     : inout std_logic;
195
196 -- FAN
197 FAN_CTRL_o           : out std_logic
198 );
199 end DE1_SoC_top;
200

```

```

201 architecture top of DE1_SoC_top is
202
203     component qsys_system is
204     port (
205         -----
206         -- FPGA Side
207         -----
208
209         -- Global signals
210         clk_clk          : in    std_logic          :=
                'X';          -- clk
211
212         -----
213         -- HPS Side
214         -----
215         -- DDR3 SDRAM
216         memory_mem_a      : out    std_logic_vector(14 downto
                0);          -- mem_a
217         memory_mem_ba     : out    std_logic_vector(2  downto
                0);          -- mem_ba
218         memory_mem_ck     : out    std_logic          -- mem_ck
219         memory_mem_ck_n   : out    std_logic          -- mem_ck_n
220         memory_mem_cke    : out    std_logic          -- mem_cke
221         memory_mem_cs_n   : out    std_logic          -- mem_cs_n
222         memory_mem_ras_n  : out    std_logic          -- mem_ras_n
223         memory_mem_cas_n  : out    std_logic          -- mem_cas_n
224         memory_mem_we_n   : out    std_logic          -- mem_we_n
225         memory_mem_reset_n : out    std_logic          -- mem_reset_n
226         memory_mem_dq      : inout  std_logic_vector(31 downto 0) :=
                (others => 'X'); -- mem_dq
227         memory_mem_dqs     : inout  std_logic_vector(3  downto 0) :=
                (others => 'X'); -- mem_dqs
228         memory_mem_dqs_n   : inout  std_logic_vector(3  downto 0) :=
                (others => 'X'); -- mem_dqs_n
229         memory_mem_odt     : out    std_logic          -- mem_odt
230         memory_mem_dm      : out    std_logic_vector(3  downto
                0);          -- mem_dm
231         memory_oct_rzqin   : in     std_logic          :=
                'X';          -- oct_rzqin
232
233         conduit_export_switch_i : in    std_logic_vector(31 downto
                0) := (others => 'X'); -- switch_i
234         conduit_export_key_i    : in    std_logic_vector(31 downto
                0) := (others => 'X'); -- key_i
235
236         conduit_export_leds_o   : out    std_logic_vector(31 downto
                0);          -- leds_o
237
238         conduit_export_hex0_o   : out    std_logic_vector(31 downto
                0);          -- hex0_o
239         conduit_export_hex1_o   : out    std_logic_vector(31 downto
                0);          -- hex1_o
240         conduit_export_hex2_o   : out    std_logic_vector(31 downto
                0);          -- hex2_o
241         conduit_export_hex3_o   : out    std_logic_vector(31 downto
                0);          -- hex3_o
242         conduit_export_hex4_o   : out    std_logic_vector(31 downto
                0);          -- hex4_o
243         conduit_export_hex5_o   : out    std_logic_vector(31 downto
                0);          -- hex5_o

```

```

244
245         -- Pushbutton
246         hps_io_hps_io_gpio_inst_GPIO54 : inout std_logic           :=
        'X';           -- hps_io_gpio_inst_GPIO54

247
248         -- LED
249         hps_io_hps_io_gpio_inst_GPIO53 : inout std_logic           :=
        'X';           -- hps_io_gpio_inst_GPIO53

250     );
251     end component qsys_system;
252
253 begin
254
255     -----
256     -- HPS mapping
257     -----
258
259     System : component qsys_system
260     port map (
261         -----
262         -- FPGA Side
263         -----
264
265         -- Global signals
266         clk_clk          => CLOCK_50_i,
267
268         -----
269         -- HPS Side
270         -----
271         -- DDR3 SDRAM
272         memory_mem_a      => HPS_DDR3_ADDR_o,
273         memory_mem_ba     => HPS_DDR3_BA_o,
274         memory_mem_ck     => HPS_DDR3_CK_P_o,
275         memory_mem_ck_n   => HPS_DDR3_CK_N_o,
276         memory_mem_cke    => HPS_DDR3_CKE_o,
277         memory_mem_cs_n   => HPS_DDR3_CS_N_o,
278         memory_mem_ras_n  => HPS_DDR3_RAS_N_o,
279         memory_mem_cas_n  => HPS_DDR3_CAS_N_o,
280         memory_mem_we_n   => HPS_DDR3_WE_N_o,
281         memory_mem_reset_n => HPS_DDR3_RESET_N_o,
282         memory_mem_dq     => HPS_DDR3_DQ_io,
283         memory_mem_dqs    => HPS_DDR3_DQS_P_io,
284         memory_mem_dqs_n  => HPS_DDR3_DQS_N_io,
285         memory_mem_odt    => HPS_DDR3_ODT_o,
286         memory_mem_dm     => HPS_DDR3_DM_o,
287         memory_oct_rzqin  => HPS_DDR3_RZQ_i,
288
289         conduit_export_switch_i (9 downto 0)           => SW_i ,           -- switch_i
290         conduit_export_switch_i (31 downto 10)         => (others => '0') ,
291         conduit_export_key_i   (3 downto 0)            => KEY_i ,           --
        key_i
292         conduit_export_key_i   (31 downto 4)           => (others => '0') ,
293
294         conduit_export_leds_o  (9 downto 0)            => LEDR_o ,           --
        leds_o
295
296         conduit_export_hex0_o  (6 downto 0)            => HEX0_o ,           --
        hex0_o
297         conduit_export_hex1_o  (6 downto 0)            => HEX1_o ,           --
        hex1_o
298         conduit_export_hex2_o  (6 downto 0)            => HEX2_o ,           --
        hex2_o
299         conduit_export_hex3_o  (6 downto 0)            => HEX3_o ,           --
        hex3_o
300         conduit_export_hex4_o  (6 downto 0)            => HEX4_o ,           --
        hex4_o
301         conduit_export_hex5_o  (6 downto 0)            => HEX5_o ,           -- hex5_o
302
303

```

```
304         -- Pushbutton
305         hps_io_hps_io_gpio_inst_GPIO54 => HPS_KEY_io,
306
307         -- LED
308         hps_io_hps_io_gpio_inst_GPIO53 => HPS_LED_io
309     );
310
311 end top;
```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 * File           : labo5.c
11 * Author          : Spinelli Isaia
12 * Date            : 01.05.2020
13 *
14 * Context         : SOCF tutorial lab
15 *
16 *****/
17 * Brief: Programme for labo 5 of SOCF, for DE1-SoC board
18 *
19 *
20 *****/
21 * Modifications :
22 * Ver    Date      Student      Comments
23 * 0.1    01.05.20   Isaia Spinelli : Modif pour la partie 1
24 * 1.1    03.05.20   Isaia Spinelli : Ajout de la partie 2
25 *****/
26 /
27 #include "defines.h"
28
29
30 /* Variable globales */
31
32 int irqKey2 = 0;
33 int irqKey3 = 0;
34
35
36
37
38 int main(void){
39
40     // tableau de conversion
41     8      9      a      b      c      d      e      f
42     char tab_dec_to_hex_7seg[16] = {0x40, 0xF9, 0x24, 0x30, 0x19, 0x12, 0x02, 0xF8,
43     0x00, 0x10, 0x08, 0x03, 0x27, 0x21, 0x06, 0x0e };
44     int led_tmp, Seg_tmp;
45
46     /*----- INTI -----*/
47     AXI_HEX5 = 0x40;
48     AXI_HEX4 = 0xF9;
49     AXI_HEX3 = 0x24;
50     AXI_HEX2 = 0x30;
51     AXI_HEX1 = 0x19;
52     AXI_HEX0 = 0x02;
53
54     AXI_LEDS = AXI_SWITCHES;
55
56     unsigned int cst = AXI_REG_CONST;
57     AXI_REG_TEST = cst;

```

```

58 // Masque le bouton key3 (pour tester le masquage des interruptions)
59 // AXI_INT_MASK = KEY3;
60
61 disable_A9_interrupts(); // disable interrupts in the A9 processor
62 set_A9_IRQ_stack();      // initialize the stack pointer for IRQ mode
63 config_GIC();            // configure the general interrupt controller
64 config_KEYS();           // configure KEYS to generate interrupts
65 enable_A9_interrupts();  // enable interrupts in the A9 processor
66
67
68
69
70 while(1){
71     /* Appuie sur KEY 0 */
72     if ((AXI_KEYS & KEY0) == 0) {
73         // l'états des switches est copiés sur les LEDs.
74         AXI_LEDS = AXI_SWITCHES;
75         // Les afficheurs HEX5 à HEX0 affichent en hexadécimal les bits 23 à 0 de
76         // la constante définie dans l'IP.
77         AXI_HEX0 = tab_dec_to_hex_7seg[cst & 0xF];
78         AXI_HEX1 = tab_dec_to_hex_7seg[(cst>>4) & 0xF];
79         AXI_HEX2 = tab_dec_to_hex_7seg[(cst>>8) & 0xF];
80         AXI_HEX3 = tab_dec_to_hex_7seg[(cst>>12) & 0xF];
81         AXI_HEX4 = tab_dec_to_hex_7seg[(cst>>16) & 0xF];
82         AXI_HEX5 = tab_dec_to_hex_7seg[(cst>>20) & 0xF];
83
84         /* Appuie sur KEY 1 */
85     } else if ((AXI_KEYS & KEY1) == 0) {
86         // l'états inverses des switches est copiés sur les LEDs.
87         AXI_LEDS = ~AXI_SWITCHES;
88
89         // Les afficheurs HEX5 à HEX0 affichent en hexadécimal l'inverse des bits
90         // 23 à 0 de la
91         // constante définie dans l'IP.
92         AXI_HEX0 = ~tab_dec_to_hex_7seg[cst & 0xF];
93         AXI_HEX1 = ~tab_dec_to_hex_7seg[(cst>>4) & 0xF];
94         AXI_HEX2 = ~tab_dec_to_hex_7seg[(cst>>8) & 0xF];
95         AXI_HEX3 = ~tab_dec_to_hex_7seg[(cst>>12) & 0xF];
96         AXI_HEX4 = ~tab_dec_to_hex_7seg[(cst>>16) & 0xF];
97         AXI_HEX5 = ~tab_dec_to_hex_7seg[(cst>>20) & 0xF];
98
99         // Si le bouton 2 est pressé (via une interruption)
100     } else if (irqKey2) {
101         irqKey2 = 0;
102
103         // l'affichage des LEDs et des afficheurs 7 segments subit unerotation à
104         // droite */
105         led_tmp = AXI_LEDS & 0x1;
106         AXI_LEDS = ((AXI_LEDS & 0x3ff) >> 1) | (led_tmp << 9);
107
108         Seg_tmp = AXI_HEX0;
109         AXI_HEX0 = AXI_HEX1;
110         AXI_HEX1 = AXI_HEX2;
111         AXI_HEX2 = AXI_HEX3;
112         AXI_HEX3 = AXI_HEX4;
113         AXI_HEX4 = AXI_HEX5;
114         AXI_HEX5 = Seg_tmp;
115
116         // Si le bouton 3 est pressé (via une interruption)
117     } else if (irqKey3) {
118         irqKey3 = 0;
119
120         // l'affichage des LEDs et des afficheurs 7 segments subit une rotation à
121         // gauche */
122         led_tmp = AXI_LEDS & 0x200;
123         AXI_LEDS = (AXI_LEDS << 1) | (led_tmp >> 9);

```



```

123         Seg_tmp = AXI_HEX5;
124         AXI_HEX5 = AXI_HEX4;
125         AXI_HEX4 = AXI_HEX3;
126         AXI_HEX3 = AXI_HEX2;
127         AXI_HEX2 = AXI_HEX1;
128         AXI_HEX1 = AXI_HEX0;
129         AXI_HEX0 = Seg_tmp;
130
131
132     }
133
134     AXI_HEX5 = test1;
135 }
136 AXI_HEX5 = test1;
137
138 }
139
140 /* Routine d'interruption */
141 void pushbutton_ISR(void){
142     // Permet de tester le masquage
143     // static int cpt_int = 0;
144
145     /* Lecture et acquittement des interruptions */
146     int src_irq = AXI_INT_SRC;
147
148     // Key2 pressé
149     if (src_irq & KEY2) {
150         irqKey2 = 1;
151     }
152
153     // Key3 pressé
154     if (src_irq & KEY3) {
155         irqKey3 = 1;
156     }
157
158
159     // Tous les 3 interruptions de KEY0 et KEY1, change le masque de key 2 et 3
160     /*
161     if (src_irq & KEY0 || src_irq & KEY1) {
162         cpt_int++;
163
164         if (cpt_int % 3 == 0)
165             AXI_INT_MASK = AXI_INT_MASK ^ (KEY3 | KEY2);
166     }
167     */
168 }
169
170

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6
7  *****/
8  *
9  * REDS Institute
10 * Reconfigurable Embedded Digital Systems
11
12 *****/
13 *
14 * File           : defines.h
15 * Author        : Sébastien Masle
16 * Date          : 16.02.2018
17 *
18 * Context       : SOCF class
19 *
20 *****/
21 * Brief: some definitions
22 *
23 *****/
24 * Modifications :
25 * Ver    Date      Engineer    Comments
26 * 0.0    16.02.2018 SMS          Initial version.
27 * 1.1    06.05.20  Isaia Spinelli : Refactor
28 *****/
29 /
30
31 #include "exceptions.h"
32
33 // Déclaration de fonction
34 void pushbutton_ISR(void);
35
36 // Defines
37
38 #define EDGE_TRIGGERED      0x1
39 #define LEVEL_SENSITIVE    0x0
40 #define CPU0                0x01 // bit-mask; bit 0 represents cpu0
41 #define ENABLE              0x1
42
43 #define USER_MODE          0b10000
44 #define FIQ_MODE            0b10001
45 #define IRQ_MODE            0b10010
46 #define SVC_MODE            0b10011
47 #define ABORT_MODE          0b10111
48 #define UNDEF_MODE          0b11011
49 #define SYS_MODE            0b11111
50
51 #define INT_ENABLE          0b01000000
52 #define INT_DISABLE        0b11000000
53
54 // Valeur des keys
55 #define KEY0 0x01
56 #define KEY1 0x02
57 #define KEY2 0x04
58 #define KEY3 0x08
59
60 // Typedef
61 typedef volatile unsigned char vcint;
62 typedef volatile unsigned short vsint;
63 typedef volatile unsigned int vuint;

```

```
60 // Adresses
61 #define FPGA_BASE_ADDR_IO 0xFF200000
62 #define AXI_LIGHT_BASE_ADDR FPGA_BASE_ADDR_IO
63
64
65 #define AXI_REG_CONST_CHAR *(vcint *) (AXI_LIGHT_BASE_ADDR + 0x0)
66 #define AXI_REG_CONST_SHORT *(vsint *) (AXI_LIGHT_BASE_ADDR + 0x0)
67 #define AXI_REG_CONST *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x0)
68
69 #define AXI_REG_TEST *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x4)
70
71 #define AXI_LEDS *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x100)
72
73 #define AXI_KEYS *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x200)
74 // Lecture de la source d'int. + acquitement
75 #define AXI_INT_SRC *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x204)
76 // 1 = interruption masquée
77 #define AXI_INT_MASK *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x208)
78
79 #define AXI_SWITCHES *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x300)
80
81 #define AXI_HEX0 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x400)
82 #define AXI_HEX1 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x410)
83 #define AXI_HEX2 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x420)
84 #define AXI_HEX3 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x430)
85 #define AXI_HEX4 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x440)
86 #define AXI_HEX5 *(vuint *) (AXI_LIGHT_BASE_ADDR + 0x450)
87
```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 * File           : exceptions.h
11 * Author          : Isaia Spinelli
12 * Date            : 06.05.2020
13 *
14 * Context         : SOCF class
15 *
16 *****/
17 * Modifications :
18 * Ver    Date      Engineer    Comments
19 * 1.1    06.05.20   Isaia Spinelli : Refactor
20 *
21 *****/
22 /
23 void disable_A9_interrupts (void);
24 void set_A9_IRQ_stack (void);
25 void config_GIC (void);
26 void config_KEYS (void);
27 void enable_A9_interrupts (void);
28 void config_interrupt (int, int);

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6  *
7  * REDS Institute
8  * Reconfigurable Embedded Digital Systems
9  *
10 *
11 * File           : exceptions.c
12 * Author        : Sébastien Masle
13 * Date          : 16.02.2018
14 * Context       : SOCF class
15 *
16 *
17 * Brief: defines exception vectors for the A9 processor
18 *        provides code that sets the IRQ mode stack, and that dis/enables interrupts
19 *        provides code that initializes the generic interrupt controller
20 *
21 *
22 * Modifications :
23 * Ver    Date      Engineer    Comments
24 * 0.0    16.02.2018 SMS          Initial version.
25 * 1.0    13.03.2020 Spinelli Isaia
26 *
27 *****/
28 #include <stdint.h>
29
30 #include "address_map_arm.h"
31 #include "defines.h"
32
33
34
35
36 // Référence : Exemple dans Using The ARM Generic
37
38 // Define the IRQ exception handler
39 void __attribute__((interrupt)) __cs3_isr_irq(void)
40 {
41     /*****
42     * Attention dans Qsys mettre sur flanc et non level !
43     *****/
44
45     // Read CPU Interface registers to determine which peripheral has caused an
46     // interrupt
47     int interrupt_ID = *((int*) 0xFFFFEC10C);
48
49     // Handle the interrupt if it comes from the KEYS
50     if (interrupt_ID == 72) {
51         pushbutton_ISR();
52     } else {
53         while (1); // if unexpected, then stay here
54     }
55
56     // Clear interrupt from the CPU Interface
57     *((int*) 0xFFFFEC110) = interrupt_ID;
58
59     return;

```

```

59     }
60
61     // Define the remaining exception handlers
62     void __attribute__((interrupt)) __cs3_reset (void)
63     {
64         while(1);
65     }
66
67     void __attribute__((interrupt)) __cs3_isr_undef (void)
68     {
69         while(1);
70     }
71
72     void __attribute__((interrupt)) __cs3_isr_swi (void)
73     {
74         while(1);
75     }
76
77     void __attribute__((interrupt)) __cs3_isr_pabort (void)
78     {
79         while(1);
80     }
81
82     void __attribute__((interrupt)) __cs3_isr_dabort (void)
83     {
84         while(1);
85     }
86
87     void __attribute__((interrupt)) __cs3_isr_fiq (void)
88     {
89         while(1);
90     }
91
92     /*
93      * Initialize the banked stack pointer register for IRQ mode
94      */
95     void set_A9_IRQ_stack(void)
96     {
97         uint32_t stack, mode;
98         stack = A9_ONCHIP_END - 7;          // top of A9 onchip memory, aligned to 8 bytes
99         /* change processor to IRQ mode with interrupts disabled */
100        mode = INT_DISABLE | IRQ_MODE;
101        asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
102        /* set banked stack pointer */
103        asm("mov sp, %[ps]" : : [ps] "r" (stack));
104
105        /* go back to SVC mode before executing subroutine return! */
106        mode = INT_DISABLE | SVC_MODE;
107        asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
108    }
109
110    /*
111     * Turn on interrupts in the ARM processor
112     */
113    void enable_A9_interrupts(void)
114    {
115        uint32_t status = SVC_MODE | INT_ENABLE;
116        asm("msr cpsr, %[ps]" : : [ps] "r" (status));
117    }
118
119    /** Turn off interrupts in the ARM processor*/
120    void disable_A9_interrupts(void) {
121        int status = 0b11010011;
122        asm("msr cpsr, %[ps]" : : [ps] "r" (status));
123    }
124
125    void config_GIC (void) {
126        // configure the FPGA KEYs interrupt (72)
127        config_interrupt (72, 1);

```

```

128
129 // Set Interrupt Priority Mask Register (ICCPMR). Enable all priorities
130 *((int*) 0xFFFFEC104) = 0xFFFF;
131
132 // Set the enable in the CPU Interface Control Register (ICCICR)
133 *((int*) 0xFFFFEC100) = 1;
134
135 // Set the enable in the Distributor Control Register (ICDDCR)
136 *((int*) 0xFFFFED000) = 1;
137 }
138
139 void config_KEYS (void) {
140     volatile int*KEY_ptr = (int*) 0xFF200050; // KEY base address
141
142     *(KEY_ptr + 2) = 0xF; // enable interrupts for all four KEYS
143
144 }
145
146 void config_interrupt (int N, int CPU_target) {
147     int reg_offset, index, value, address;
148
149     /*Configure the Interrupt Set-Enable Registers (ICDISERn).
150     *reg_offset = (integer_div(N / 32)*4; value = 1 << (N mod 32)*/
151
152     reg_offset = (N >> 3) & 0xFFFFFFFFFC;
153     index = N & 0x1F;
154     value = 0x1 << index;
155     address = 0xFFFFED100 + reg_offset;
156
157     /*Using the address and value, set the appropriate bit*/
158     *(int*)address |= value;
159
160     /*Configure the Interrupt Processor Targets Register (ICDIPTRn)
161     * reg_offset = integer_div(N / 4)*4; index = N mod 4*/
162     reg_offset = (N & 0xFFFFFFFFFC);
163     index = N & 0x3;
164     address = 0xFFFFED800 + reg_offset + index;
165
166     /*Using the address and value, write to (only) the appropriate byte*/
167     *(char*)address = (char) CPU_target;
168 }
169

```

```

1  /*****
2  *
3  * HEIG-VD
4  * Haute Ecole d'Ingenierie et de Gestion du Canton de Vaud
5  * School of Business and Engineering in Canton de Vaud
6
7  *****/
8  *
9  * REDS Institute
10 * Reconfigurable Embedded Digital Systems
11
12 *****/
13 *
14 * File           : address_map_arm.h
15 * Author        : Sébastien Masle
16 * Date          : 16.02.2018
17 *
18 * Context       : SOCF class
19 *
20 *****/
21 * Brief: provides address values that exist in the system
22 *
23 *****/
24 * Modifications :
25 * Ver    Date      Engineer    Comments
26 * 0.0    16.02.2018 SMS         Initial version.
27 *
28 *****/
29 /
30
31 #define BOARD                "DE1-SoC"
32
33 /* Memory */
34 #define DDR_BASE              0x00000000
35 #define DDR_END                0x3FFFFFFF
36 #define A9_ONCHIP_BASE        0xFFFF0000
37 #define A9_ONCHIP_END          0xFFFFFFFF
38 #define SDRAM_BASE             0xC0000000
39 #define SDRAM_END               0xC3FFFFFF
40 #define FPGA_ONCHIP_BASE       0xC8000000
41 #define FPGA_ONCHIP_END         0xC803FFFF
42 #define FPGA_CHAR_BASE         0xC9000000
43 #define FPGA_CHAR_END           0xC9001FFF

```