12/04/14 10:06:06 /home/atmelfan/vhdl/C204/registers/registery.vhd

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_unsigned.all;
 4
 5  entity registery is
 6      generic (
 7          WIDTH: integer := 8;
 8          PROG: integer := 16
 9      );
10      port (
11          --Register input/outputs
12          clock, wren: in std_logic;
13          addr_A, addr_B: in integer range 0 to 15;
14          read_A, read_B: out std_logic_vector(WIDTH-1 downto 0);
15          write_B: in std_logic_vector(WIDTH-1 downto 0);
16          --
17          leds: out std_logic_vector(WIDTH-1 downto 0);
18          --Program counter output
19          skip: in std_logic;
20          pcout: out std_logic_vector(PROG-1 downto 0);
21          --External memory output
22          wr: inout std_logic;
23          data: inout std_logic_vector(WIDTH-1 downto 0);
24          addr: inout std_logic_vector(15 downto 0)
25      );
26  end entity ; -- alu
27
28  architecture arch of registery is
29      type mem is array(0 to 14) of std_logic_vector(WIDTH-1 downto 0);
30      signal registers: mem;
31      signal pc: std_logic_vector(PROG-1 downto 0) := (others => '0');
32  begin
33      --Asynchronous read from registers or databuss
34      --A read port
35      process(addr_A, data, pc, registers)
36      begin
37          case addr_A is
38              when 15 => read_A <= data;
39              when 12 => read_A <= pc(15 downto 8);
40              when 11 => read_A <= pc( 7 downto 0);
41              when others => read_A <= registers(addr_A);
42          end case;
43      end process;
44      --B read port
45      process(addr_B, data, pc, registers)
46      begin
47          case addr_B is
48              when 15 => read_B <= data;
49              when 12 => read_B <= pc(15 downto 8);
50              when 11 => read_B <= pc( 7 downto 0);
51              when others => read_B <= registers(addr_B);
52          end case;
53      end process;
54
55      --Synchronous write on falling edge
56      process(clock, wren, registers, skip)
57      begin
58          --On falling edge, write data to registers or databuss
59          if(falling_edge(clock)) then
60              --If writing to PCH register, update PC
61              if(addr_B = 12  and wren = '1') then
62                  pc <= write_B&registers(11);
63                  registers(addr_B) <= write_B;
64              --If not, increment PC as normal and write to registers
65              else
66                  if(skip = '1') then
67                      pc <= pc + 2;
68                  else
69                      pc <= pc + 1;
70                  end if;
71                  --If writing to DAT, write to databus instead
72                  if(addr_B = 15 and wren = '1') then
73                      data <= write_B;
74                  --If writing to PCL DO NOT UPDATE REGISTERS
75                  elsif(addr_B = 11  and wren = '1') then
```

```
76                      registers(addr_B) <= write_B;
77                  --Not working with PCL or PCH, update them (and write to whatever register
    selected)
78                  elsif(wren = '1') then
79                      registers(11) <= pc( 7 downto 0);
80                      registers(12) <= pc(15 downto 8);
81                      registers(addr_B) <= write_B;
82                  end if;
83              end if;
84          end if;
85      end process;
86
87      --If reading/writing to register 15(DAT) set address so to external memory, else let them
    float.
88      addr <= registers(13)&registers(14) when addr_A = 15 or addr_B = 15 else (others => 'Z');
89      wr <= '0' when addr_A = 15 or addr_B = 15 else 'Z';
90      pcout <= pc;
91      leds <= registers(0);
92
93  end architecture ; -- arch
```