```
1   Library ieee;
2
3   use ieee.std_logic_1164.all;
4
5   -- This program is written for the 2017 Spring DLD class at Grove City College. The goal is
    to create a system which polls classrooms around campus to clooect information from them.
6   -- Information collected is ClassroomInUse, LightsAreOn, and ProjectorIsOn.
7   -- ClassroomInUse being a 1 represents that the classroom is being used, all of these
    signals come from sensors in the classroom that are explained in the writup attached with
    this code.
8
9   -- This VHDL program is written by Theo Stangebye, stangebyeTO1@gcc.edu, April 2017.
10
11  entity CampusController is
12  port( gpio : inout std_logic_vector(7 downto 0);   -- We clear all of the io on the baord
    so that the LEDs are not "Ghosted on".
13        ledr : out std_logic_vector(17 downto 0);
14        ledg : out std_logic_vector(8 downto 0);
15        sw : in std_logic_vector(17 downto 0);
16        key : in std_logic_vector (3 downto 0)
17     );
18  end CampusController;
19
20  -- The campus Controller will poll classrooms (which are simulated on another VHDL board).
    It will output a 2 bit integer which represents the ID of 1 of 4 classrooms.
21  -- When a classroom's id is broadcasted on the RoomID bits, that classroom will connect to
    the communication bus (1 bit wide) and send it's information over a serial connection to
    this campus controller.
22  architecture a of CampusController is
23
24     -- COMPONENT DECLARATIONS
25     component ls74 is -- This is a standard DFF.
26        port( d, clr, pre, clk : in std_logic;
27              q : out std_logic
28        );
29     end component;
30
31     -- Delcare Asynchronous clear 4 bit counter
32     component vhdl_binary_counter is
33        port (    C, CLR : in std_logic;
34                  Q : out std_logic_vector(3 downto 0)
35              );
36     end component;
37
38     -- Synchronous 4 bit counter
39     component ls163 is
40        port( C, CLR : in std_logic;
41              Q : out std_logic_vector(3 downto 0)
42        );
43     end component;
44
45     -- SIPO Shift Regerister.
46     component sipo is
47        port (    clk, clear : in std_logic;
48                  Input_Data: in std_logic;
49                  Q: out std_logic_vector(15 downto 0)
50              );
51     end component;
52
53     -- Much of the register file code is derived from online lecture slides, see entity
    declaration for more.
54     component register_file is
55        Port (    src_s0 : in std_logic; -- src_s0 and src_s0 are the selection bits which
    decide which register the selectedData out gets its bits from.
56              src_s1 : in std_logic;
57              des_A0 : in std_logic; -- address of register file for writing to.
58              des_A1 : in std_logic;
59              writeToReg : in std_logic; -- when we want to write to the Register.
60              Clk : in std_logic; -- clock signal.
61              data_src : in std_logic; -- this is an artifact from the walkthrough slides.
62              data : in std_logic_vector(3 downto 0); -- Data input that we want to write.
63              reg0 : out std_logic_vector(3 downto 0); -- register 0 contents
64              reg1 : out std_logic_vector(3 downto 0); -- register 1 contents, etc.
```

```
65              reg2 : out std_logic_vector(3 downto 0);
66              reg3 : out std_logic_vector(3 downto 0);
67              selectedData : out std_logic_vector(3 downto 0) -- the data selected by src_s1 and src_s0
68        );
69     end component;
70
71     -- SIGNALS
72     Signal rxin : std_logic_vector(15 downto 0); -- this signal represents the last 16 bits recieved on the RX input.
73     Signal RoomID : std_logic_vector(3 downto 0); -- RoomID will be the binary number of the classroom that we are talking to,
74     Signal tx, Master_Clock, rx : std_logic; -- tx can be used to communicate in serial on, Master_Clock is the main clock signal, and RX is our recieving bit.
75     Signal StartFlag, EndFlag, BitStringAlligned : std_logic; -- the startFlag is thrown after a predetermined serial stream is recieved which represeents a ClassroomController Coming Online.
76     -- the END is thrown after a predetermined serial stream is recieved which represeents a ClassroomController Coming Online.
77     -- BitStringAlligned is thrown when we are ready to load the bitstream from RX into our DB.
78
79  begin
80
81     -- GPIO INPUTS AND OUTPUTS
82        rx <= gpio(4);    -- input from classroomController
83        gpio(3) <= tx;    -- we could talk to classroomControllers on this.
84        gpio(1 downto 0) <= RoomID(1 downto 0); -- Broadcast RoomID to ClassroomControllers
85        Master_Clock <= gpio(7); -- Read clock from arduino.
86        ledg(8) <= Master_Clock; -- flash LEDG(8) with clock signal.
87        -- share clock signal with children.
88        gpio(5) <= Master_clock; -- Tranmit clock signal to connected classroomContorllers.
89
90        -- Hook up RX to shift Regerister - this takes a serial stream in and produces a parallel output representing the last 16 bits that came through on the shift register.
91        inputReg : sipo port map (clk => Master_Clock, Clear => '0', Input_Data => rx, q => rxin); -- rxin is the 16bit history of what came in on RX.
92        -- rxin(0) should be the newest bit recieved, rxin(15) the oldest.
93
94        -- This is some combinational logic that sets StartFlag to '1' when rxin is "1010101010101010"
95        StartFlag <= (rxin(15) and rxin(13) and rxin(11) and rxin(9) and rxin(7) and rxin(5) and rxin(3) and rxin(1)) and Not(rxin(14) OR rxin(12) OR rxin(10) Or rxin(8) or rxin(6) or rxin(4) or rxin(2) or rxin(0));
96        -- End flag when rxin is "1111111111111111"
97        EndFlag <= (rxin(15) and rxin(14) and rxin(12) and rxin(11) and rxin(10) and rxin(9) and rxin(8) and rxin(7) and rxin(6) and rxin(5) and rxin(4) and rxin(3) and rxin(2) and rxin(1) and rxin(0));
98
99        -- Implement our model for a 74x161 with asynchronous clear.  This counter drives our RoomID Count.
100       RoomIDCounter : vhdl_binary_counter port map (
101             C => EndFlag,
102             CLR => RoomID(2),  -- since we are only talking two 4 classrooms, we will reset the counter as soon as the binar number changes from 0011 to 0100
103             Q => RoomID
104       );
105
106       -- Because this chip does not have an RCO, implement some combinational logic to simulate the RCO, when this simulation triggers RCO, we know that the we are ready to read from our input shift regerister into our memory circuitry.
107       BitStringAlligned <= Not(rxin(15) or rxin(14) or rxin(13) or rxin(12) or rxin(11) or rxin(10) or rxin(9) or rxin(8) or rxin(0) or rxin(1) or rxin(2) or rxin(3) or rxin(4));
108
109       -- Implement Memory Component:
110       -- A note on the memory component, in a full system where we have as many as 8 classrooms with 64 classrooms a piece, we would need 512 register, since we are only demonstrating 4 classrooms across two buildings, we are going to use a 4 register register file.
111       -- This specific register file has 4 bit wide registers, we will wire '0' to the MSB of the register.
112       Database : register_file port map(
113          src_s0 => sw(16), -- since we don't need to read from the register file, these can
```

```
    be Zero.
114             src_s1 => sw(17),
115             des_A0 => RoomID(0), -- we index our data based on what classroom we are reading
    from
116             des_A1 => RoomID(1), -- we index our data based on what classroom we are reading
    from
117             writeToReg => bitStringAlligned, -- just like the circuit diagram, this is wired
    to execute when the bit string is alligned.
118             Clk => Master_Clock,
119             data_src => '0', -- we always want our data to flow from the input bus.
120             data => '0' & Rxin(7 downto 5),  -- Here we assign our 4 bits of Data, zero padded
    on the MSB to arrange the 3 bits of data in a 4 bit register.
121             reg0 => ledr(3 downto 0), -- here we put the contents of our registers onto LEDs.
122             reg1 => ledr(7 downto 4),
123             reg2 => ledr(11 downto 8),
124             reg3 => ledr(15 downto 12),
125             selectedData => ledg( 3 downto 0)
126         );
127
128         -- update leds with information as to what our circuit is doing.
129         Ledg(6) <= BitStringAlligned; -- this implies we're writing to the register file.
130         Ledg(7) <= StartFlag; -- this means we got the start flag
131         Ledg(5) <= EndFlag; -- we got the end flag.
132
133   end a;
134
135
136   -- Create a 74x74 chip a DFF
137   -- This component intends to simulate the behaviors of a 74x74 chipset.
138   Library ieee;
139   use ieee.std_logic_1164.all;
140   Entity ls74 is
141      port( d, clr, pre, clk : IN std_logic;
142         -- d is the data input
143         -- clr: ACTIVE LOW: clears the output, q, asynchrnously.
144         -- Pre: ACTIVE LOW: sets the output q to 1 asynchronously,
145         -- clk is a clock signal (q is typically representitive of what d was 1 clock cycle
    ago)
146            q : out std_logic -- single bit output which is d delayed by 1 clock cycle.
147      );
148   end ls74;
149   Architecture a of ls74 is
150   begin
151      Process(clk, clr, pre) -- the DFF should update its output when any of these change.
152      begin
153         if clr = '0' then -- preset q to zero, reguardless of d.
154            q <= '0'; -- note that clr and pre are active low.
155         elsif pre = '0' then  -- preset q to zero, reguardless of d.
156            q <= '1';
157         elsif clk'EVENT and clk = '1' then -- mimic d 1 clock cycle ago on q.
158            if d = '1' then
159               q <= '1';
160            else
161               q <= '0';
162            end if;
163         end if;
164      end process;
165   End a;
166
167   -- Create 4 bit counters.
168   -- BASIC 4 bit counter:
169   library ieee;
170   use ieee.std_logic_1164.all;
171   use ieee.std_logic_unsigned.all;
172
173   -- this 4 bit counter has an asynchronous clear.
174   entity vhdl_binary_counter is
175      port(C, CLR : in std_logic; -- C is the clock signal.
176      Q : out std_logic_vector(3 downto 0)); -- 4 bit integer output.
177   end vhdl_binary_counter;
178
179   architecture bhv of vhdl_binary_counter is
180      signal tmp: std_logic_vector(3 downto 0);
```

```
181   begin
182      process (C, CLR)
183      begin
184      if (CLR='1') then
185         tmp <= "0000";
186      elsif (C'event and C='1') then
187         tmp <= tmp + 1;
188      end if;
189      end process;
190      Q <= tmp;
191   end bhv;
192
193   -- 4 bit counter with Synchronous clear:
194   -- Note that a 163 would normally include a load function.
195   library ieee;
196   use ieee.std_logic_1164.all;
197   use ieee.std_logic_unsigned.all;
198   entity ls163 is
199      port(C, CLR : in std_logic;
200      Q : out std_logic_vector(3 downto 0));
201   end ls163;
202   architecture bhv of ls163 is
203      signal tmp: std_logic_vector(3 downto 0);
204   begin
205      process (C, CLR)
206      begin
207      if (C'event and C='1' and CLR='1') then
208         tmp <= "0000";
209      elsif (C'event and C='1') then
210         tmp <= tmp + 1;
211      end if;
212      end process;
213      Q <= tmp;
214   end bhv;
215
216   -- Begin SIPO Shift Register - adapted from
      https://allaboutfpga.com/vhdl-code-for-4-bit-shift-register/
217   library ieee;
218   use ieee.std_logic_1164.all;
219   entity sipo is -- the SIPO register takes data in in serial and produces a parallel string
      representing the last so many values that appeared in its bitstream input.
220    port(
221    clk, clear : in std_logic;
222    Input_Data: in std_logic;
223    Q: out std_logic_vector(15 downto 0) ); -- this one remembers 16 bits.
224   end sipo;
225
226   architecture arch of sipo is
227    Signal temp : std_logic_vector(15 downto 0);
228   begin
229
230    process (clk)
231    begin
232    if clear = '1' then
233    Q <= "0000000000000000";
234    temp <= "0000000000000000";
235    elsif (CLK'event and CLK='1') then
236    temp(15 downto 1) <= temp(14 downto 0); -- 15 is going to be the oldest data, 0 will be
      the newest.
237    temp(0) <= Input_Data;
238    Q <= temp;
239    end if;
240    end process;
241   end arch;
242
243   -- Create RegisterFile Components:
244   -- This code comes from the guide at
      https://www.scss.tcd.ie/Michael.Manzke/CS2022/CS2022_vhdl_eighth.pdf
245   -- from here to the end of this file, the contents are created from the lecture slides at
      the URL above. As such, this code is implemented with sparse comments as the author of the
      lecture materials above neglected to comment his code.
246   -- 2 to 4 decoder
247   -- NOTE THIS IS ONLY ONE HALF of a ls139 chip.
```

```
248   library IEEE;
249   use IEEE.STD_LOGIC_1164.ALL;
250   use IEEE.STD_LOGIC_ARITH.ALL;
251   use IEEE.STD_LOGIC_UNSIGNED.ALL;
252   entity decoder_2to4 is
253      Port (
254         Enable : IN std_logic; -- I'm adding an enable signal that we can turn high to write
      to the regerister file.  Disabling the decoder writes 0 to all of the outputs.
255         A0 : in std_logic;
256         A1 : in std_logic;
257         Q0 : out std_logic;
258         Q1 : out std_logic;
259         Q2 : out std_logic;
260         Q3 : out std_logic);
261   end decoder_2to4;
262   architecture Behavioral of decoder_2to4 is
263      begin
264         Q0<= ((not A0) and (not A1)) and Enable; -- and enable to implement 0 out of m when
      the enable bit is '0'
265         Q1<= (A0 and (not A1)) and Enable;
266         Q2<= ((not A0) and A1) and Enable;
267         Q3<= (A0 and A1) and Enable;
268   end Behavioral;
269
270   -- 4 bit wide 2 to 1 mux
271   -- this is 1 half of a 74x157 mux
272   library IEEE;
273   use IEEE.STD_LOGIC_1164.ALL;
274   use IEEE.STD_LOGIC_ARITH.ALL;
275   use IEEE.STD_LOGIC_UNSIGNED.ALL;
276   entity mux2_4bit is
277      port (   In0 : in std_logic_vector(3 downto 0);
278               In1 : in std_logic_vector(3 downto 0);
279   s : in std_logic;
280               Z : out std_logic_vector(3 downto 0)
281            );
282   end mux2_4bit;
283   architecture Behavioral of mux2_4bit is
284   begin
285      Z <= In0 when S='0' else
286      In1 when S='1'else
287      "0000";
288   end Behavioral;
289
290   -- 4 bit wide 4 to 1 MUX
291   library IEEE;
292   use IEEE.STD_LOGIC_1164.ALL;
293   use IEEE.STD_LOGIC_ARITH.ALL;
294   use IEEE.STD_LOGIC_UNSIGNED.ALL;
295   entity mux4_4bit is
296   Port (   In0, In1, In2, In3 : in std_logic_vector(3 downto 0);
297            S0, S1 : in std_logic;
298            Z : out std_logic_vector(3 downto 0)
299         );
300   end mux4_4bit;
301   architecture Behavioral of mux4_4bit is
302   begin
303      Z <= In0 when S0='0' and S1='0' else
304      In1 when S0='1' and S1='0' else
305      In2 when S0='0' and S1='1' else
306      In3 when S0='1' and S1='1' else
307      "0000";
308   end Behavioral;
309
310   -- Finally, here is the register component -- this register is 4 bits wide and simulates 4
      DFFs wired in parallel.
311   library IEEE;
312   use IEEE.STD_LOGIC_1164.ALL;
313   use IEEE.STD_LOGIC_ARITH.ALL;
314   use IEEE.STD_LOGIC_UNSIGNED.ALL;
315   entity reg4 is
316      port ( D : in std_logic_vector(3 downto 0); -- input data to register.
317         load, Clk : in std_logic;
```

```
318         Q : out std_logic_vector(3 downto 0) -- output of the register (4 bits in parallel)
319      );
320   end reg4;
321   architecture Behavioral of reg4 is begin
322      process(Clk)
323      begin
324         if (rising_edge(Clk)) then
325            if load='1' then
326               Q<=D;
327            end if;
328         end if;
329      end process;
330   end Behavioral;
331
332   -- Use the register component to create a register file component:
333   library IEEE;
334   use IEEE.STD_LOGIC_1164.ALL;
335   use IEEE.STD_LOGIC_ARITH.ALL;
336   use IEEE.STD_LOGIC_UNSIGNED.ALL;
337   entity register_file is
338      Port (   src_s0 : in std_logic;
339               src_s1 : in std_logic;
340               des_A0 : in std_logic;
341               des_A1 : in std_logic;
342               writeToReg : in std_logic; -- I added an enable bit to the 2 to 4 decoder so
    that i can have a write signal here.
343               Clk : in std_logic;
344               data_src : in std_logic;
345               data : in std_logic_vector(3 downto 0);
346               reg0 : out std_logic_vector(3 downto 0);
347               reg1 : out std_logic_vector(3 downto 0);
348               reg2 : out std_logic_vector(3 downto 0);
349               reg3 : out std_logic_vector(3 downto 0);
350               selectedData : out std_logic_vector(3 downto 0)
351            );
352   end register_file;
353   architecture Behavioral of register_file is
354   -- components
355   -- 4 bit Register for register file
356      COMPONENT reg4 PORT(
357         D : IN std_logic_vector(3 downto 0);
358         load : IN std_logic;
359         Clk : IN std_logic;
360         Q : OUT std_logic_vector(3 downto 0)
361      );
362      END COMPONENT;
363
364      -- 2 to 4 Decoder
365      COMPONENT decoder_2to4 PORT(
366         Enable : IN std_logic; -- I'm adding an enable signal that we can turn high to write
    to the regerister file.
367         A0 : IN std_logic;
368         A1 : IN std_logic;
369         Q0 : OUT std_logic;
370         Q1 : OUT std_logic;
371         Q2 : OUT std_logic;
372         Q3 : OUT std_logic
373      );
374      END COMPONENT;
375      -- 2 to 1 line multiplexer
376      COMPONENT mux2_4bit PORT(
377         In0 : IN std_logic_vector(3 downto 0);
378         In1 : IN std_logic_vector(3 downto 0);
379         s : IN std_logic;
380         Z : OUT std_logic_vector(3 downto 0)
381      );
382      END COMPONENT;
383
384      -- 4 to 1 line multiplexer
385      COMPONENT mux4_4bit PORT(
386         In0 : IN std_logic_vector(3 downto 0); In1 : IN std_logic_vector(3 downto 0); In2 : IN
     std_logic_vector(3 downto 0); In3 : IN std_logic_vector(3 downto 0); S0 : IN std_logic;
387         S1 : IN std_logic;
```

```
388         Z : OUT std_logic_vector(3 downto 0)
389      );
390      END COMPONENT;
391
392      -- signals
393      signal load_reg0, load_reg1, load_reg2, load_reg3 : std_logic;
394      signal reg0_q, reg1_q, reg2_q, reg3_q, data_src_mux_out, src_reg : std_logic_vector(3
     downto 0);
395
396   begin
397
398      -- port maps ;-)
399      -- register 0
400      reg00: reg4 PORT MAP(
401         D => data_src_mux_out,
402         load => load_reg0, Clk => Clk,
403         Q => reg0_q
404      );
405      -- register 1
406      reg01: reg4 PORT MAP(
407         D => data_src_mux_out,
408         load => load_reg1,
409         Clk => Clk,
410         Q => reg1_q
411      );
412      -- register 2
413      reg02: reg4 PORT MAP(
414         D => data_src_mux_out,
415         load => load_reg2, Clk => Clk,
416         Q => reg2_q
417      );
418      -- register 3
419      reg03: reg4 PORT MAP(
420         D => data_src_mux_out,
421         load => load_reg3, Clk => Clk,
422         Q => reg3_q
423      );
424      -- Destination register decoder
425         des_decoder_2to4: decoder_2to4 PORT MAP( Enable => writeToReg,  A0 => des_A0,
426         A1 => des_A1, Q0 => load_reg0, Q1 => load_reg1, Q2 => load_reg2, Q3 => load_reg3
427      );
428      -- 2 to 1 Data source multiplexer
429      data_src_mux2_4bit: mux2_4bit PORT MAP( In0 => data,
430         In1 => src_reg,
431         s => data_src,
432         Z => data_src_mux_out
433      );
434      -- 4 to 1 source register multiplexer
435         Inst_mux4_4bit: mux4_4bit PORT MAP( In0 => reg0_q,
436         In1 => reg1_q, In2 => reg2_q, In3 => reg3_q, S0 => src_s0, S1 => src_s1, Z => src_reg
437      );
438
439      selectedData <= src_reg ;-- Send Selected data to selectedData output
440
441      reg0 <= reg0_q; reg1 <= reg1_q; reg2 <= reg2_q; reg3 <= reg3_q;
442   end Behavioral;
443
444   -- END REGISTER FILE
445
```