DIGITAL LOGIC DESIGN PROJECT

Daniel Colflesh | Armand Ignelzi | Nathaniel Shaffer | Theo Stangebye

The Concentration Companion

Problem Statement:

Grove City College, everyone knows how hard the academics are. In order to do well at this rigorous institution, you must work hard and study to keep your grades up. Finding a place to study at this college can be stressful, but with the Concentration Companion that problematic situation is no more. The Concentration Companion will help the individual locate an open spot to study or do homework. Many people waste their time looking around for an open spot and end up irate and miserable, which may result in the completion of zero work by the end of the day. The Concentration Companion will also be useful specific to the case of signing in and out of classrooms in the Hall of Arts and Letters. There are many great benefits to the addition of the Concentration Companion to Grove City's campus. Most importantly it will help students become even more efficient while providing peace of mind that they have a spot to complete their assignments.

Purpose:

The main purpose of the Concentration Companion is to make it easier for students at Grove City College to find a place to study or do their work. This project idea has other benefits to it as well. For example, many students forget to turn off the lights and projectors when they leave. The Concentration Companion would solve this, optimizing power consumption as well as lowering maintenance costs, and expanding the life of projector/light bulbs. Finally, the Concentration Companion eliminates the need for signing in and out of classrooms during evening hours in HAL. The sign in system is inaccurate and flawed in that many students don't sign in or out. The Concentration Companion automatically determines whether or not a classroom is in use. Clearly, there are many positive attributes to the Concentration Companion that will better the academic lives and careers of the students at Grove City College.

Design Goals:

1. Our first goal is to be able collect data from individual classrooms and store it in a central location.
2. The second goal is to develop a robust communication protocol using flags in our bit streams, a master clock cycle for the whole system, and room IDs and Hi-Z states for inactive rooms.
3. The third goal is to be able to turn off projectors and lights when the room is not in use.
4. The fourth goal is cost efficiency. This is achieved by polling the data onto a bus minimizing the amount of materials needed for the system.

Roles:

Theo Strangebye:

-Project Manager, VHDL coding for Campus Controller and Classroom Controller and design goals.

Daniel Colflesh:

-Campus Controller circuit schematic, documentation, meeting minutes and design goals.

Nathaniel Shaffer:

-Classroom Controller circuit schematic, documentation and design goals.

Armand Ignelzi:

-Building Controller circuit schematic and VHDL code, documentation and design goals.

Documentation:

Classroom Controller:

Each classroom has one Classroom Controller that has 7 inputs and 3 outputs.

Inputs:

- LightsOn
- ProjectorOn
- ClassroomInUse
- RoomID
- Clock_in
- RX

-LightsOn and ProjectorOn are each connected to sensors that output a value of '1' if the lights or projector is on.

-ClassroomInUse is connected to a thermal sensor that outputs a value of '1' if it detects an object with a temperature of at least 90 degrees Fahrenheit.

-RoomID is a 6 bit integer. The campus controller will output the Room ID number of the classroom it wants to communicate with and the classroom controller with that Room ID number will come online and transmit its information.

-Clock_in is the input of the master clock signal.

-RX is not currently used in our designs for the Classroom Controller, but could be used in the future for two way communication.

Outputs:

- Lights_Enable
- Projector_Enable
- Tx_1

-Lights_Enable enables the lights to be turned on from within the classroom, otherwise it disables the lights, turning them off.

-Projector_Enable enables the projector to be turned on from within the classroom, otherwise is disables the projector, turning it off.

- Tx_1 is the serial communication output. Classroom data is communicated to Campus Controller through this output.

Campus Controller:

The Campus Controller has 2 inputs and 3 outputs.

Inputs:

- Rx
- Clock_in

-Rx is the serial communication input. The Campus Controller receives this data from the Classroom Controller.

-Clock_in is the input of the master clock signal. In this case the master clock signal is generated by an Arduino.

Outputs:

- RoomID
- Clock_out
- TX

-RoomID is a 6 bit integer that is used to select a classroom for polling. The Campus Controller outputs the Room ID number of the classroom with which it wishes to communicate.

-Clock_out is the output for the master clock signal to the rest of the system.

-The TX bit is not currently used in the Campus Controller but could be used to establish two way communication with the Classroom Controllers.

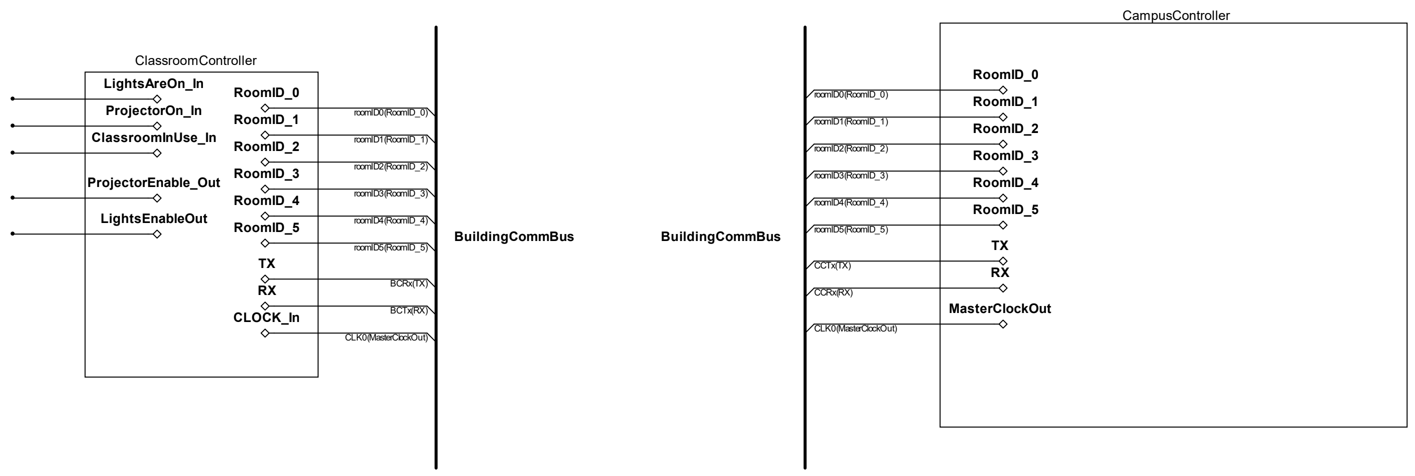
The system itself contains 4 Classroom Controllers and 1 Campus Controller in our altera simulation.

Evaluation:

The final implementation of the system achieved all of the design goals that we originally decided. The system solved the main problem of not knowing which classrooms were available to study in, with the added benefit of limiting unnecessary power consumption and extending the life of projector and ceiling light bulbs. Ultimately the system saves Grove City's students time and effort while simultaneously saving money. While the final implementation of our system is limited in scope, it can be scaled up to accommodate as many rooms necessary. The implementation presented in class is a basic proof of concept.

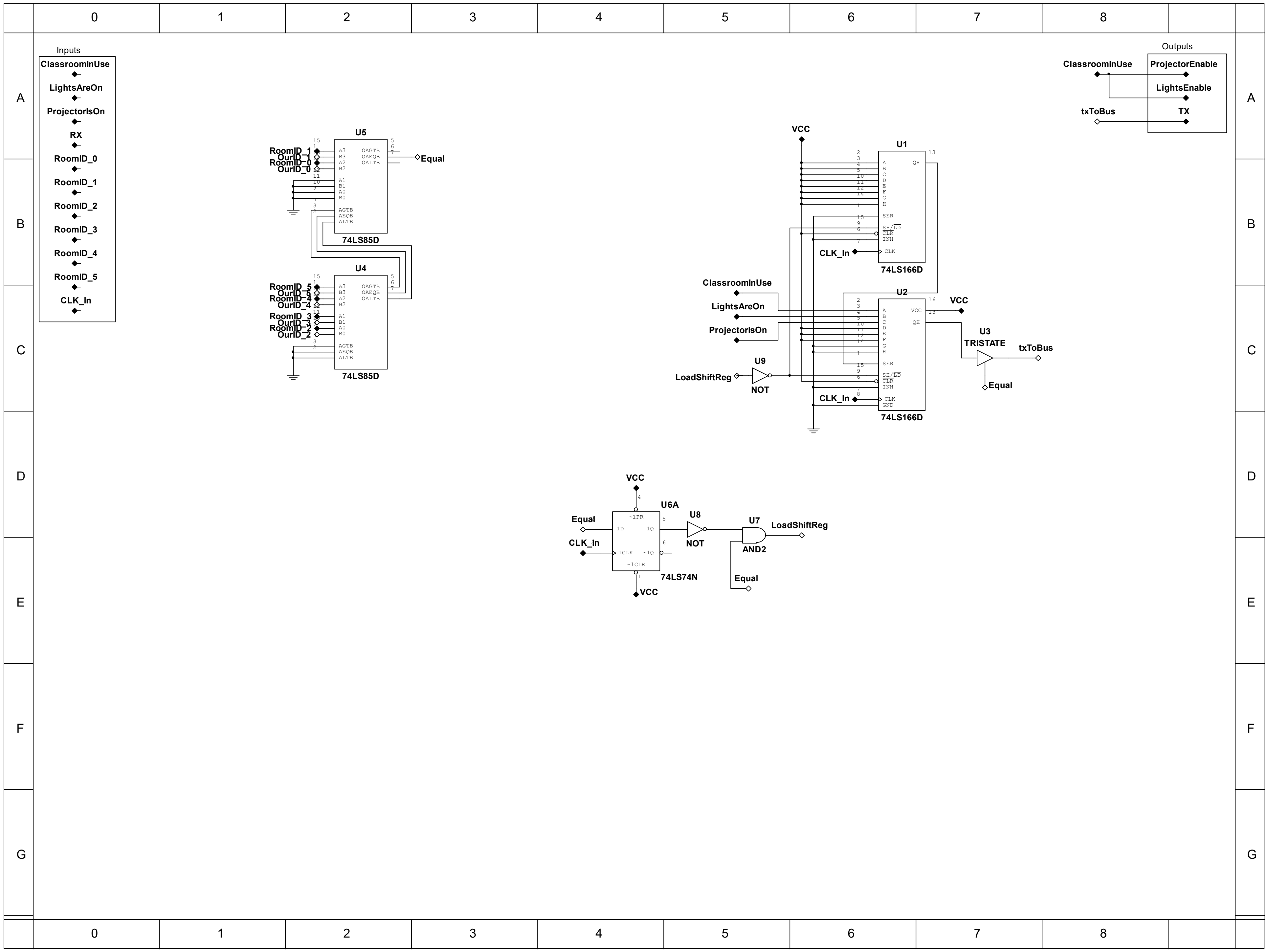
There were several design stages that we went through. Initially we started with a design that had 3 controllers: classroom, building and campus. We also had 64 classrooms per building and 4 buildings. Due to limited availability of hardware for simulation purposes, we decided to make system have 4 classrooms, 2 buildings and 1 campus controller. After the three-controller system failed, and because of time restraints, the decision was made to simplify the system by removing the Building Controller completely and modify the Classroom and Campus controllers to communicate with one another.

Overall the process of designing and implementing our system was challenging but went fairly smoothly. We started with an initial solution to the problem but were forced to simplify and modify the solution to a more realistic design. We all learned a lot about digital logic and the process of engineering.

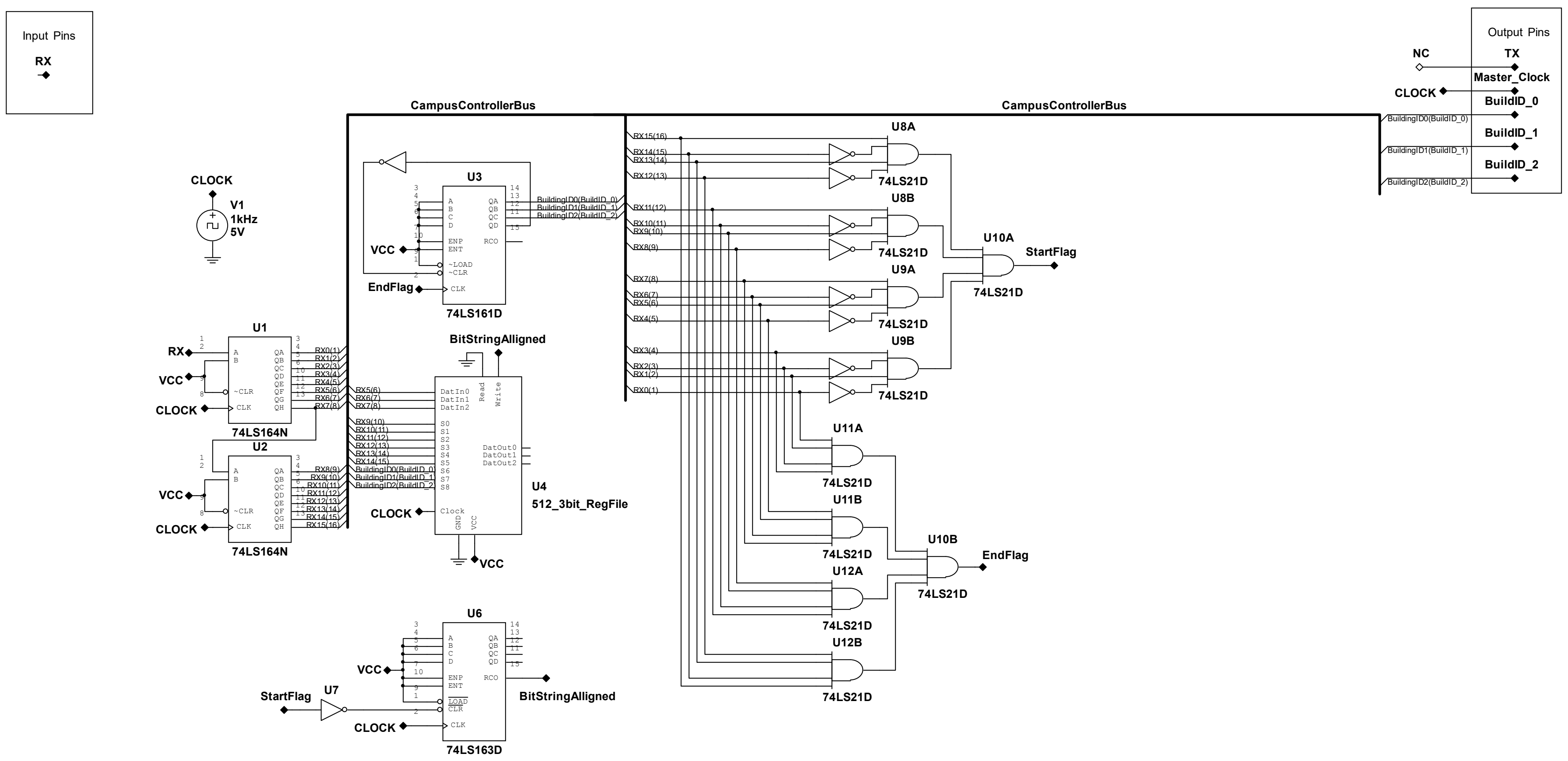


Title: GroupProjectInitial

Designed By:
Theo Stangebye



Inputs
ClassroomInUse
LightsAreOn
ProjectorIsOn
RX
RoomID_0
RoomID_1
RoomID_2
RoomID_3
RoomID_4
RoomID_5
CLK_In
U5
RoomID_1
OurID_1
RoomID_0
OurID_0
A3
B3
A2
B2
A1
B1
A0
B0
AGTB
AEQB
ALTB
OAGTB
OAEQB
OALTB
Equal
74LS85D
U4
RoomID_5
OurID_5
RoomID_4
OurID_4
RoomID_3
OurID_3
RoomID_2
OurID_2
74LS85D
VCC
U1
A
B
C
D
E
F
G
H
SER
SH/LD
CLR
INH
CLK
QH
CLK_In
74LS166D
ClassroomInUse
LightsAreOn
ProjectorIsOn
U2
VCC
GND
74LS166D
U9
LoadShiftReg
NOT
U3
TRISTATE
txToBus
Equal
CLK_In
Outputs
ClassroomInUse
ProjectorEnable
LightsEnable
txToBus
TX
VCC
U6A
~1PR
1D
1Q
1CLK
~1Q
~1CLR
Equal
CLK_In
74LS74N
VCC
U8
NOT
U7
AND2
LoadShiftReg
Equal
0
1
2
3
4
5
6
7
8
A
B
C
D
E
F
G



Title: CampusController

Size: A

Document N:

Revision: 1

Date: 4/22/2018

Sheet1 of 1

```
Library ieee;

use ieee.std_logic_1164.all;

-- 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.
-- Information collected is ClassroomInUse, LightsAreOn, and ProjectorIsOn.
-- 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.

-- This VHDL program is written by Theo Stangebye, stangebyeTO1@gcc.edu, April 2017.

entity CampusController is
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".
      ledr : out std_logic_vector(17 downto 0);
      ledg : out std_logic_vector(8 downto 0);
      sw : in std_logic_vector(17 downto 0);
      key : in std_logic_vector (3 downto 0)
   );
end CampusController;

-- 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.
-- 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.
architecture a of CampusController is

   -- COMPONENT DECLARATIONS
   component ls74 is -- This is a standard DFF.
      port( d, clr, pre, clk : in std_logic;
            q : out std_logic
      );
   end component;

   -- Delcare Asynchronous clear 4 bit counter
   component vhdl_binary_counter is
      port (   C, CLR : in std_logic;
               Q : out std_logic_vector(3 downto 0)
            );
   end component;

   -- Synchronous 4 bit counter
   component ls163 is
      port( C, CLR : in std_logic;
            Q : out std_logic_vector(3 downto 0)
      );
   end component;

   -- SIPO Shift Regerister.
   component sipo is
      port (   clk, clear : in std_logic;
               Input_Data: in std_logic;
               Q: out std_logic_vector(15 downto 0)
            );
   end component;

   -- Much of the register file code is derived from online lecture slides, see entity declaration for more.
   component register_file is
      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.
            src_s1 : in std_logic;
            des_A0 : in std_logic; -- address of register file for writing to.
            des_A1 : in std_logic;
            writeToReg : in std_logic; -- when we want to write to the Register.
            Clk : in std_logic; -- clock signal.
            data_src : in std_logic; -- this is an artifact from the walkthrough slides.
            data : in std_logic_vector(3 downto 0); -- Data input that we want to write.
            reg0 : out std_logic_vector(3 downto 0); -- register 0 contents
            reg1 : out std_logic_vector(3 downto 0); -- register 1 contents, etc.
```

```
            reg2 : out std_logic_vector(3 downto 0);
            reg3 : out std_logic_vector(3 downto 0);
            selectedData : out std_logic_vector(3 downto 0) -- the data selected by src_s1 and src_s0
      );
   end component;

   -- SIGNALS
   Signal rxin : std_logic_vector(15 downto 0); -- this signal represents the last 16 bits recieved on the RX input.
   Signal RoomID : std_logic_vector(3 downto 0); -- RoomID will be the binary number of the classroom that we are talking to,
   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.
   Signal StartFlag, EndFlag, BitStringAlligned : std_logic; -- the startFlag is thrown after a predetermined serial stream is recieved which represeents a ClassroomController Coming Online.
   -- the End is thrown after a predetermined serial stream is recieved which represeents a ClassroomController Coming Online.
   -- BitStringAlligned is thrown when we are ready to load the bitstream from RX into our DB.

begin

   -- GPIO INPUTS AND OUTPUTS
      rx <= gpio(4);    -- input from classroomController
      gpio(3) <= tx;    -- we could talk to classroomControllers on this.
      gpio(1 downto 0) <= RoomID(1 downto 0); -- Broadcast RoomID to ClassroomControllers
      Master_Clock <= gpio(7); -- Read clock from arduino.
      ledg(8) <= Master_Clock; -- flash LEDG(8) with clock signal.
      -- share clock signal with children.
      gpio(5) <= Master_clock; -- Tranmit clock signal to connected classroomContorllers.

      -- 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.
      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.
      -- rxin(0) should be the newest bit recieved, rxin(15) the oldest.

      -- This is some combinational logic that sets StartFlag to '1' when rxin is "1010101010101010"
      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));
      -- End flag when rxin is "1111111111111111"
      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));

      -- Implement our model for a 74x161 with asynchronous clear.  This counter drives our RoomID Count.
      RoomIDCounter : vhdl_binary_counter port map (
            C => EndFlag,
            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
            Q => RoomID
      );

      -- 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.
      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));

      -- Implement Memory Component:
      -- 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.
      -- This specific register file has 4 bit wide registers, we will wire '0' to the MSB of the register.
      Database : register_file port map(
         src_s0 => sw(16), -- since we don't need to read from the register file, these can
```

```
be Zero.
         src_s1 => sw(17),
         des_A0 => RoomID(0), -- we index our data based on what classroom we are reading from
         des_A1 => RoomID(1), -- we index our data based on what classroom we are reading from
         writeToReg => bitStringAlligned, -- just like the circuit diagram, this is wired to execute when the bit string is alligned.
         Clk => Master_Clock,
         data_src => '0', -- we always want our data to flow from the input bus.
         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.
         reg0 => ledr(3 downto 0), -- here we put the contents of our registers onto LEDs.
         reg1 => ledr(7 downto 4),
         reg2 => ledr(11 downto 8),
         reg3 => ledr(15 downto 12),
         selectedData => ledg( 3 downto 0)
      );

      -- update leds with information as to what our circuit is doing.
      Ledg(6) <= BitStringAlligned; -- this implies we're writing to the register file.
      Ledg(7) <= StartFlag; -- this means we got the start flag
      Ledg(5) <= EndFlag; -- we got the end flag.

end a;


-- Create a 74x74 chip a DFF
-- This component intends to simulate the behaviors of a 74x74 chipset.
Library ieee;
use ieee.std_logic_1164.all;
Entity ls74 is
   port( d, clr, pre, clk : IN std_logic;
      -- d is the data input
      -- clr: ACTIVE LOW: clears the output, q, asynchrnously.
      -- Pre: ACTIVE LOW: sets the output q to 1 asynchronously,
      -- clk is a clock signal (q is typically representitive of what d was 1 clock cycle ago)
         q : out std_logic -- single bit output which is d delayed by 1 clock cycle.
   );
end ls74;
Architecture a of ls74 is
begin
   Process(clk, clr, pre) -- the DFF should update its output when any of these change.
   begin
      if clr = '0' then -- preset q to zero, reguardless of d.
         q <= '0'; -- note that clr and pre are active low.
      elsif pre = '0' then  -- preset q to zero, reguardless of d.
         q <= '1';
      elsif clk'EVENT and clk = '1' then -- mimic d 1 clock cycle ago on q.
         if d = '1' then
            q <= '1';
         else
            q <= '0';
         end if;
      end if;
   end process;
End a;

-- Create 4 bit counters.
-- BASIC 4 bit counter:
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

-- this 4 bit counter has an asynchronous clear.
entity vhdl_binary_counter is
   port(C, CLR : in std_logic; -- C is the clock signal.
   Q : out std_logic_vector(3 downto 0)); -- 4 bit integer output.
end vhdl_binary_counter;

architecture bhv of vhdl_binary_counter is
   signal tmp: std_logic_vector(3 downto 0);
```

```
begin
   process (C, CLR)
   begin
   if (CLR='1') then
      tmp <= "0000";
   elsif (C'event and C='1') then
      tmp <= tmp + 1;
   end if;
   end process;
   Q <= tmp;
end bhv;

-- 4 bit counter with Synchronous clear:
-- Note that a 163 would normally include a load function.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity ls163 is
   port(C, CLR : in std_logic;
   Q : out std_logic_vector(3 downto 0));
end ls163;
architecture bhv of ls163 is
   signal tmp: std_logic_vector(3 downto 0);
begin
   process (C, CLR)
   begin
   if (C'event and C='1' and CLR='1') then
      tmp <= "0000";
   elsif (C'event and C='1') then
      tmp <= tmp + 1;
   end if;
   end process;
   Q <= tmp;
end bhv;

-- Begin SIPO Shift Register - adapted from https://allaboutfpga.com/vhdl-code-for-4-bit-shift-register/
library ieee;
use ieee.std_logic_1164.all;
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.
 port(
 clk, clear : in std_logic;
 Input_Data: in std_logic;
 Q: out std_logic_vector(15 downto 0) ); -- this one remembers 16 bits.
end sipo;

architecture arch of sipo is
 Signal temp : std_logic_vector(15 downto 0);
begin

 process (clk)
 begin
 if clear = '1' then
 Q <= "0000000000000000";
 temp <= "0000000000000000";
 elsif (CLK'event and CLK='1') then
 temp(15 downto 1) <= temp(14 downto 0); -- 15 is going to be the oldest data, 0 will be the newest.
 temp(0) <= Input_Data;
 Q <= temp;
 end if;
 end process;
end arch;

-- Create RegisterFile Components:
-- This code comes from the guide at https://www.scss.tcd.ie/Michael.Manzke/CS2022/CS2022_vhdl_eighth.pdf
-- 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.
-- 2 to 4 decoder
-- NOTE THIS IS ONLY ONE HALF of a ls139 chip.
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity decoder_2to4 is
   Port (
      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.
      A0 : in std_logic;
      A1 : in std_logic;
      Q0 : out std_logic;
      Q1 : out std_logic;
      Q2 : out std_logic;
      Q3 : out std_logic);
end decoder_2to4;
architecture Behavioral of decoder_2to4 is
   begin
      Q0<= ((not A0) and (not A1)) and Enable; -- and enable to implement 0 out of m when the enable bit is '0'
      Q1<= (A0 and (not A1)) and Enable;
      Q2<= ((not A0) and A1) and Enable;
      Q3<= (A0 and A1) and Enable;
end Behavioral;

-- 4 bit wide 2 to 1 mux
-- this is 1 half of a 74x157 mux
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mux2_4bit is
   port (   In0 : in std_logic_vector(3 downto 0);
            In1 : in std_logic_vector(3 downto 0);
s : in std_logic;
            Z : out std_logic_vector(3 downto 0)
         );
end mux2_4bit;
architecture Behavioral of mux2_4bit is
begin
   Z <= In0 when S='0' else
   In1 when S='1'else
   "0000";
end Behavioral;

-- 4 bit wide 4 to 1 MUX
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity mux4_4bit is
Port (   In0, In1, In2, In3 : in std_logic_vector(3 downto 0);
         S0, S1 : in std_logic;
         Z : out std_logic_vector(3 downto 0)
      );
end mux4_4bit;
architecture Behavioral of mux4_4bit is
begin
   Z <= In0 when S0='0' and S1='0' else
   In1 when S0='1' and S1='0' else
   In2 when S0='0' and S1='1' else
   In3 when S0='1' and S1='1' else
   "0000";
end Behavioral;

-- Finally, here is the register component -- this register is 4 bits wide and simulates 4 DFFs wired in parallel.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reg4 is
   port ( D : in std_logic_vector(3 downto 0); -- input data to register.
      load, Clk : in std_logic;
```

```
      Q : out std_logic_vector(3 downto 0) -- output of the register (4 bits in parallel)
   );
end reg4;
architecture Behavioral of reg4 is begin
   process(Clk)
   begin
      if (rising_edge(Clk)) then
         if load='1' then
            Q<=D;
         end if;
      end if;
   end process;
end Behavioral;

-- Use the register component to create a register file component:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity register_file is
   Port (   src_s0 : in std_logic;
            src_s1 : in std_logic;
            des_A0 : in std_logic;
            des_A1 : in std_logic;
            writeToReg : in std_logic; -- I added an enable bit to the 2 to 4 decoder so that i can have a write signal here.
            Clk : in std_logic;
            data_src : in std_logic;
            data : in std_logic_vector(3 downto 0);
            reg0 : out std_logic_vector(3 downto 0);
            reg1 : out std_logic_vector(3 downto 0);
            reg2 : out std_logic_vector(3 downto 0);
            reg3 : out std_logic_vector(3 downto 0);
            selectedData : out std_logic_vector(3 downto 0)
         );
end register_file;
architecture Behavioral of register_file is
-- components
-- 4 bit Register for register file
   COMPONENT reg4 PORT(
      D : IN std_logic_vector(3 downto 0);
      load : IN std_logic;
      Clk : IN std_logic;
      Q : OUT std_logic_vector(3 downto 0)
   );
   END COMPONENT;

   -- 2 to 4 Decoder
   COMPONENT decoder_2to4 PORT(
      Enable : IN std_logic; -- I'm adding an enable signal that we can turn high to write to the regerister file.
      A0 : IN std_logic;
      A1 : IN std_logic;
      Q0 : OUT std_logic;
      Q1 : OUT std_logic;
      Q2 : OUT std_logic;
      Q3 : OUT std_logic
   );
   END COMPONENT;
   -- 2 to 1 line multiplexer
   COMPONENT mux2_4bit PORT(
      In0 : IN std_logic_vector(3 downto 0);
      In1 : IN std_logic_vector(3 downto 0);
      s : IN std_logic;
      Z : OUT std_logic_vector(3 downto 0)
   );
   END COMPONENT;

   -- 4 to 1 line multiplexer
   COMPONENT mux4_4bit PORT(
      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;
      S1 : IN std_logic;
      Z : OUT std_logic_vector(3 downto 0)
   );
   END COMPONENT;

   -- signals
   signal load_reg0, load_reg1, load_reg2, load_reg3 : std_logic;
   signal reg0_q, reg1_q, reg2_q, reg3_q, data_src_mux_out, src_reg : std_logic_vector(3 downto 0);

begin

   -- port maps ;-)
   -- register 0
   reg00: reg4 PORT MAP(
      D => data_src_mux_out,
      load => load_reg0, Clk => Clk,
      Q => reg0_q
   );
   -- register 1
   reg01: reg4 PORT MAP(
      D => data_src_mux_out,
      load => load_reg1,
      Clk => Clk,
      Q => reg1_q
   );
   -- register 2
   reg02: reg4 PORT MAP(
      D => data_src_mux_out,
      load => load_reg2, Clk => Clk,
      Q => reg2_q
   );
   -- register 3
   reg03: reg4 PORT MAP(
      D => data_src_mux_out,
      load => load_reg3, Clk => Clk,
      Q => reg3_q
   );
   -- Destination register decoder
      des_decoder_2to4: decoder_2to4 PORT MAP( Enable => writeToReg,  A0 => des_A0,
      A1 => des_A1, Q0 => load_reg0, Q1 => load_reg1, Q2 => load_reg2, Q3 => load_reg3
   );
   -- 2 to 1 Data source multiplexer
   data_src_mux2_4bit: mux2_4bit PORT MAP( In0 => data,
      In1 => src_reg,
      s => data_src,
      Z => data_src_mux_out
   );
   -- 4 to 1 source register multiplexer
      Inst_mux4_4bit: mux4_4bit PORT MAP( In0 => reg0_q,
      In1 => reg1_q, In2 => reg2_q, In3 => reg3_q, S0 => src_s0, S1 => src_s1, Z => src_reg
   );

   selectedData <= src_reg ;-- Send Selected data to selectedData output

   reg0 <= reg0_q; reg1 <= reg1_q; reg2 <= reg2_q; reg3 <= reg3_q;
end Behavioral;

-- END REGISTER FILE

```

```
Library ieee;

use ieee.std_logic_1164.all;

-- 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.
-- Information collected is ClassroomInUse, LightsAreOn, and ProjectorIsOn.
-- 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.

-- This VHDL program is written by Theo Stangebye, stangebyeTO1@gcc.edu, April 2017.

-- The classroomController Entity is the component which is being polled. Each classroom will have a classroomController which communicates with at campusController when it's id is selected by the campus controller for polling.  The classroomController will then connect to the communication line and transmit its data in serial to the campus controller.
entity ClassroomController is -- we'll ue classroomControllerHardware as our actual classroomController Simulator
port( gpio : inout std_logic_vector(39 downto 0);
      ledr : out std_logic_vector(17 downto 0);
      ledg : out std_logic_vector(8 downto 0);
      sw : in std_logic_vector(17 downto 0);
      key : in std_logic_vector (3 downto 0)
   );
end ClassroomController;

-- Our architecture instantiates 4 classroomController components, simulating 4 classrooms which will be polled by a campusController
architecture a of ClassroomController is

   -- Declare the components that we created below.
   Component ClassroomControllerHardware is -- ClassroomControllerHardware is the model which simulates the hardware to be placed in each classroom.
      port (   ClassroomInUse, LightsAreOn, ProjectorIsOn, RX : in std_logic; -- ClassroomInUse, LightsAreOn, and ProjectorIsOn are boolean signals from sensors in the room which give infromation about that classroom's current condition.
               RoomID, OurID : in std_logic_vector(1 downto 0); -- here we are simulating 4 classrooms, so the classroom IDs, only need to be 2 bits wides.  -- RoomID is broadcasted by the CampusController, and OurID is set to the the unique ID of a classroomController in a specific classroom.
               Clk_In : in std_logic; -- the clock signal.
               projectorEnable, LightsEnable, TX, transmitting : out std_logic -- each classroomController has outputs which can disable the lights or projector in a classroom.
               -- The TX bit is used to communicate with the campusController, it is directly connected to the connection line, and is internally set to highZ when it is not a classroomController's turn to communicate with the campusController, (a classroomController communicates when it is polled by the campusController or when it's OURID = RoomId.)
            );
   end Component;

   signal master_clock : std_logic; -- the master signal of the alera board - this is read from another altera board in our case through gpio pin. (see GPIO below)
   signal c0len, c0pen, c1len, c1pen, c2len, c2pen, c3len, c3pen : std_logic; -- c0len is Classroom0, lights, enable. c2pen is Classroom2, Projector, Enable.  these signals are used internally to interact with the altera board's physical hardware for IO purposes.
   signal sen0u, sen0l, sen0p, sen1u, sen1l, sen1p, sen2u, sen2l, sen2p, sen3u, sen3l, sen3p : std_logic; -- sensor associated with classroom #, sensing use, lights, projector - these repsresent the input sensors to a classroom.
   signal net1RoomID : std_logic_vector(1 downto 0); -- the RoomId on network 1, (in this case, this is the default network since there is only one network.)
   signal net1tx : std_logic; -- the communication line for network 1.
   signal trans0, trans1, trans2, trans3 : std_logic; -- we'll use to display LEDs on the baord when a specific classroomControllerHardware is transmitting, (when it is being polled by the campusController)

begin
   -- GPIO
   -- Read clock from external source.
   master_clock <= gpio(8);
   -- Read roomIds from other altera board using GPIO ports.
   net1RoomID <= gpio(1 downto 0);
   -- TX bit for communicating with CampusController
```

```
48     gpio(6) <= net1tx;
49
50     -- flash ledg8 with clock signal.
51     ledg(8) <= master_clock;
52
53     -- input switches - used to simulate classrom sensor values.
54     -- Class 0:
55     sen0u <= sw(0); -- classroom0 in Use
56     sen0l <= sw(1); -- classroom0 lightsAreOn
57     sen0p <= sw(2); -- classroom0 projectorIsOn
58     -- Class 1:
59     sen1u <= sw(4);
60     sen1l <= sw(5);
61     sen1p <= sw(6);
62     -- Class 2:
63     sen2u <= sw(8);
64     sen2l <= sw(9);
65     sen2p <= sw(10);
66     -- Class 3:
67     sen3u <= sw(12);
68     sen3l <= sw(13);
69     sen3p <= sw(14);
70
71     -- Outpus LEDs:
72     -- Class 0:
73     ledr(0) <= c0len; -- classroom0 lightsEnabled
74     ledr(1) <= c0pen; -- classroom0 projectorEnabled
75     ledr(2) <= trans0; -- classroom0 is being polled.
76     -- Class 1:
77     ledr(4) <= c1len;
78     ledr(5) <= c1pen;
79     ledr(6) <= trans1;
80     -- Class 2:
81     ledr(8) <= c2len;
82     ledr(9) <= c2pen;
83     ledr(10) <= trans2;
84     -- Class 3:
85     ledr(12) <= c3len;
86     ledr(13) <= c3pen;
87     ledr(14) <= trans3;
88
89     -- we will plot network 1 room id and network 1 tx on ledg
90     ledg(1 downto 0) <= net1RoomID;
91     ledg(7) <= net1tx;
92
93     -- Instantiate our classrooms.
94     Classroom0 : classroomControllerHardware port map(
95        ClassroomInUse => sen0u, -- sensor representing whether or not someone is in the room.
96        LightsAreOn => sen0l, -- sensor representing whether or not the lights are on the room
97        ProjectorIsOn => sen0p, -- sensor representing whether or not the projector is on in the room.
98        RX => '0', -- for now, we are not recieving serial from the campusController.
99        RoomID => net1RoomID, -- set the roomID in this classroomControllerHardware to be the ID recieved on the GPIO pins.
100       OurID => "00", -- set the unique ID of this specific classroomControllerHardware
101       Clk_In => master_clock, -- pass our clock signal
102       ProjectorEnable => c0pen, -- output enabling the projector
103       LightsEnable => c0len, -- ouput enabling the lights.
104       TX => net1tx, -- classroom0's communication line.
105       transmitting => trans0 -- this signal is true if classroom0 is online on the communication line and is transmitting it's data to the campusController.
106    );
107
108       Classroom1 : classroomControllerHardware port map(
109       ClassroomInUse => sen1u,
110       LightsAreOn => sen1l,
111       ProjectorIsOn => sen1p,
112       RX => '0',
113       RoomID => net1RoomID,
114       OurID => "01",
115       Clk_In => master_clock,
116       ProjectorEnable => c1pen,
117       LightsEnable => c1len,
```

```
118       TX => net1tx,
119       transmitting => trans1
120    );
121
122       Classroom2 : classroomControllerHardware port map(
123       ClassroomInUse => sen2u,
124       LightsAreOn => sen2l,
125       ProjectorIsOn => sen2p,
126       RX => '0',
127       RoomID => net1RoomID,
128       OurID => "10",
129       Clk_In => master_clock,
130       ProjectorEnable => c2pen,
131       LightsEnable => c2len,
132       TX => net1tx,
133       transmitting => trans2
134    );
135
136       Classroom3 : classroomControllerHardware port map(
137       ClassroomInUse => sen3u,
138       LightsAreOn => sen3l,
139       ProjectorIsOn => sen3p,
140       RX => '0',
141       RoomID => net1RoomID,
142       OurID => "11",
143       Clk_In => master_clock,
144       ProjectorEnable => c3pen,
145       LightsEnable => c3len,
146       TX => net1tx,
147       transmitting => trans3
148    );
149
150 end a;
151
152 -- Begin Component Declarations
153
154 -- here we declare the classroomControllerHardware which represents the hardware placed in each classroom.
155 Library ieee;
156 use ieee.std_logic_1164.all;
157 Entity ClassroomControllerHardware is -- these signals are explained at line 25.
158    port (   ClassroomInUse, LightsAreOn, ProjectorIsOn, RX : in std_logic;
159             RoomID, OurID : in std_logic_vector(1 downto 0);
160             Clk_In : in std_logic;
161             projectorEnable, LightsEnable, TX, transmitting : out std_logic
162          );
163 end ClassroomControllerHardware;
164
165 Architecture a of ClassroomControllerHardware is
166
167    -- declare signals and hardware components.
168    component ls74 is
169    port( d, clr, pre, clk : IN std_logic;
170       -- d is the data input
171       -- clr: ACTIVE LOW: clears the output, q, asynchrnously.
172       -- Pre: ACTIVE LOW: sets the output q to 1 asynchronosly.
173       -- clk is a clock signal (q is typically representitive of what d was 1 clock cycle ago)
174          q : out std_logic -- single bit output which is d delayed by 1 clock cycle.
175    );
176    end component;
177
178    component piso48b is -- this is a parallel input serial output shift register which is 48b wide.
179       port (   parallel_In : in std_logic_vector(47 downto 0); -- the 16 bits of input for parallel loading
180                SorL : in std_logic; -- the Shift/Load signal. 1 = shift, 0 = load
181                clk : in std_logic; -- the clock signal for the DFFs contained in the shift reg.
182                q : out std_logic -- we shift out through this bit.
183             );
184    end component;
185
186    component comparator6b is -- a six bit comparator which only determines whether or not
```

```
two six bit integers are equal.
   port (   op1, op2 : in std_logic_vector(5 downto 0); -- our two 6b inputs.
            equal : out std_logic -- our 1 bit equal signal. 1 if op1 = op2, else 0.
         );
   end component;

   component tri_state_buffer_top is
   Port (   A  : in  STD_LOGIC;    -- single buffer input
            EN : in  STD_LOGIC;    -- single buffer enable
            Y  : out STD_LOGIC    -- single buffer output
         );
   end component;

   signal Equal, LoadShiftReg, txToBus, lastEqual, projectorIsEnabledAndOn, lightsEnabledAndOn: std_logic;
   Signal toLoad : std_logic_vector(47 downto 0);
   -- Equal is an internal signal which is true if the RoomID input matches OURID.
   -- LoadShiftReg tells the ClassroomControllerHardware when to load its PISO register.
   -- TX to bus stores our tx value for the bus before sending it through a tri_state_buffer
   -- lastEqual is the equal signal 1 clock cycle ago.
   -- projectorIsEnabledAndOn and lightsEnabledAndOn are signals which are used to simulate the closed loop system created by this hardware's ability to disable the lights and projector.

begin

   -- for indication purposes, show when this classroom is transmitting
   transmitting <= Equal;

   -- Circuitry to determine when we are selected by the BuildingController to Transmit
   classroomComparator : comparator6b port map(
      op1 => "0000" & OurID,
      op2 => "0000" & RoomID,  -- the comparator expects 6b of input but roomId and ourID are only 2 bits now.
      equal => Equal
   );

   -- Circuitry to determine when we should load our shift registers with new data to shift out over serial.
   -- we want to load the first clock cycle after being selected by the Building Controller (first equal cycle)
   RisingEqualDFF : ls74 port map(
      d => equal,
      clr => '1',
      pre => '1',
      clk => Clk_In,
      q => lastEqual
   );

   LoadShiftReg <= Not(lastEqual) and equal;

   -- Finally implement shift out register for parallel in and serial out - used to communicate with campusControllers in serial.
   serialOutReg : piso48b port map(
      parallel_In => toLoad,
      SorL => Not(LoadShiftReg), -- Load is low state.
      clk => Clk_In,
      q => txToBus
   );

   -- specify toLoad
   toLoad <= "1010101010101010" & "00000000" & projectorIsEnabledAndOn & lightsEnabledAndOn & ClassroomInUse & "00000" & "1111111111111111";
   -- these bitstrings come from predetermined serial communication flags.

   -- wire txto bus to tx bus with a tristate buffer
   busBuffer : tri_state_buffer_top Port map (
      A => txToBus,
      En => Equal,
      Y => TX
   );

   -- disable lights and projector when classroom is not in use.
```

```
   ProjectorEnable <= ClassroomInUse;
   LightsEnable <= ClassroomInUse;

      -- We simulate that the projector is on or off by doing the following:
   projectorIsEnabledAndOn <= ClassroomInUse and projectorIsOn;
   lightsEnabledAndOn <= ClassroomInUse and lightsAreOn;

end a;

-- Create a 74x74 chip a DFF
-- This component intends to simulate the behaviors of a 74x74 chipset.
Library ieee;
use ieee.std_logic_1164.all;
Entity ls74 is
   port( d, clr, pre, clk : IN std_logic;
      -- d is the data input
      -- clr: ACTIVE LOW: clears the output, q, asynchrnously.
      -- Pre: ACTIVE LOW: sets the output q to 1 asynchronously,
      -- clk is a clock signal (q is typically representitive of what d was 1 clock cycle ago)
         q : out std_logic -- single bit output which is d delayed by 1 clock cycle.
   );
end ls74;
Architecture a of ls74 is
begin
   Process(clk, clr, pre) -- the DFF should update its output when any of these change.
   begin
      if clr = '0' then -- preset q to zero, reguardless of d.
         q <= '0'; -- note that clr and pre are active low.
      elsif pre = '0' then  -- preset q to zero, reguardless of d.
         q <= '1';
      elsif clk'EVENT and clk = '1' then -- mimic d 1 clock cycle ago on q.
         if d = '1' then
            q <= '1';
         else
            q <= '0';
         end if;
      end if;
   end process;
End a;

-- We also need a 6 bit comparator.  In actual hardware, this would likely be two 74x85s in series but here we will make our own 6b comparator.
-- We only need to know if the two operands are equal, so we'll leave out greater than/less than capabilities.
Library ieee;
Use ieee.std_logic_1164.all;
Entity comparator6b is
   port (   op1, op2 : in std_logic_vector(5 downto 0); -- our two 6b inputs.
            equal : out std_logic -- our 1 bit equal signal. 1 if op1 = op2, else 0.
         );
end comparator6b;
Architecture a of comparator6b is
begin
   Process (op1, op2)
   begin
      if op1 = op2 then -- if they are equal, represent that on equal.
         equal <= '1';
      else
         equal <= '0';
      end if;
   end process;
end a;

-- Finally, we will need a 16b PISO shift regerister.
--    in our circuit schematic, we wired two 8 bit shift regeristers together, but here, we can just create a 48b shift register.
Library ieee;
Use ieee.std_logic_1164.all;
Entity piso48b is
   port (   parallel_In : in std_logic_vector(47 downto 0); -- the 16 bits of input for parallel loading
            SorL : in std_logic; -- the Shift/Load signal. 1 = shift, 0 = load
            clk : in std_logic; -- the clock signal for the DFFs contained in the shift reg.
            q : out std_logic -- we shift out through this bit.
         );
end piso48b;
Architecture a of piso48b is
   signal temp : std_logic_vector(47 downto 0);
begin
   -- Note: in this shift regerister, elements are shifted "up" meaning that an item which
enters at temp(0) is consecutively shifted down the register to temp(47) at which point it
shows up on the output q.
   process(clk) -- Our register updates every clock cycle, nothing is asynchronous.
   begin
      if clk'EVENT and clk = '1' then
         if SorL = '0' then -- we should load from our parallel input
            temp <= parallel_In;
         else -- otherwise we should shift down the register.
            temp(47 downto 1) <= temp(46 downto 0);
            temp(0) <= '1';  -- we never need to shift in serial for this project
component, so we can simply simulate shifting in a zero.
         end if;
      end if;
   end process;

   q <= temp(47); -- connect our temp vector (the zero element) to our output.
end a;

-- Tristate Buffer
Library ieee;
use ieee.std_logic_1164.all;
entity tri_state_buffer_top is
   Port( A : in std_logic;    -- single buffer input
         EN : in std_logic;    -- single buffer enable
         Y : out std_logic    -- single buffer output
   );
end tri_state_buffer_top;
architecture Behavioral of tri_state_buffer_top is
   begin
    -- single active low enabled tri-state buffer
   Y <= A when (EN = '1') else 'Z';
end Behavioral;
```