Drexel UNIVERSITY
Department of Electrical and Computer Engineering

Register Tutorials and Exercises

By Prawat Nagvajara

II. Cascading Register with Single Step

1.1 Create a project e.g., cascading_registers
1.2 Under Project Manager, Add Sources, Add Files, navigate to register.srcs/sources1/new/reg.vhd select reg.vhd and OK
1.3 Check Copy Sources into Project box and Finish
1.4 Create cascade_reg, edit structural description style code for 4-bit data (register 1 and register 2) (Fig. 1).

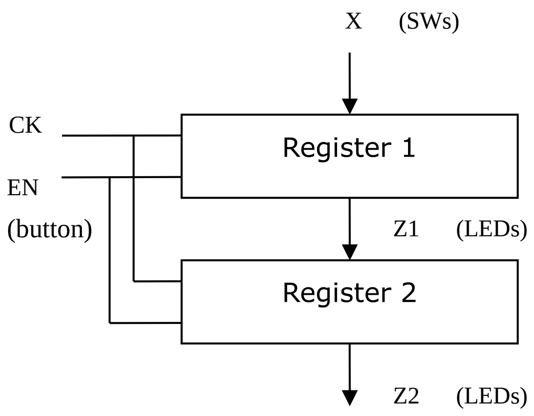

Fig. 1 Cascading Register with Load Enable

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity cascade_reg4 is
port (   X :  in  std_logic_vector(3 downto 0);
        Z1 : out  std_logic_vector(3 downto 0);
        Z2 : out  std_logic_vector(3 downto 0);
en, ck, btn0, btn1 :  in  std_logic);
end cascade_reg4;
```

```
architecture structural of cascade_reg4 is
component reg
generic (n: natural);
Port ( ck, ld_en: in std_logic;
 x: in  std_logic_vector(n-1 downto 0);
 z: out std_logic_vector(n-1 downto 0));
end component;
signal w : std_logic_vector(3 downto 0);
begin
R1: reg generic map (4)
        port map(x=>x,z=>w,ck=>ck,ld_en=>en);
R2: reg generic map (4)
        port map(x=>w,z=>Z2,ck=>ck,ld_en=>en);
Z1 <= w; -- wire w to output port
end structural;
```

2. Create design constraints file
3. Synthesize, implement, generate bitstream, program FPGA and verify correctness.
4. Questions
   a) temp1 (z1 LEDs) should get new input vector x and temp2 (z2 LEDs) gets old temp1 when en button pressed, but it doesn't do that, does it?
   b) Do both new Z1 and Z2 equal to X when en button pressed? Why?

5. Cascading Registers with Single Step

To allow a single stepping of the clock signal a new design uses a state machine which generates a single pulse on button presses. Figure 3 shows its block diagram.

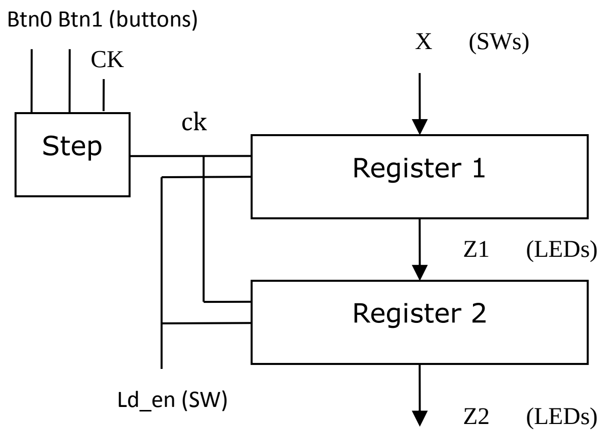

Fig. 3 Cascading Registers with Single-Step Clock Signal

```
------------------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Description: Two 4-bit registers R1, R2
-- cascading using copies of reg component.
-- Output of R1 and R2 wired to LEDs
-- en (load enable) wired to switch
-- btn0 and btn1 wired to btnL and btnR
-- Internal signal ck_step drives the registers
------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity step_cascade_reg4 is
port (    X :   in   std_logic_vector(3 downto 0);
         Z1 :  out   std_logic_vector(3 downto 0);
         Z2 :  out   std_logic_vector(3 downto 0);
en, ck, btn0, btn1 :  in  std_logic);
end step_cascade_reg4;

architecture structural of step_cascade_reg4 is

component reg
generic (n: natural);
Port ( ck, ld_en: in std_logic;
 x: in  std_logic_vector(n-1 downto 0);
 z: out std_logic_vector(n-1 downto 0));
end component;

signal w : std_logic_vector(3 downto 0);
signal ck_step: std_logic;
begin
R1: reg generic map (4)
        port map(x=>x,z=>w,ck=>ck_step,ld_en=>en);
R2: reg generic map (4)
        port map(x=>w,z=>Z2,ck=>ck_step,ld_en=>en);
Z1 <= w; -- wire w to output port
```

```
-- single step clock pulse (debounce)
-- btn0 to enter and btn1 to reset
-- ck_step signal drives components
process(ck)
type state is (not_rdy, rdy, pulse);
variable ns: state;
begin
if ck='1' and ck'event then
  case ns is
   when not_rdy => ck_step <= '0';
   if btn1 = '1' then ns := rdy; end if;
   when rdy => ck_step <= '0';
   if btn0 = '1' then ns := pulse; end if;
   when pulse => ck_step <= '1';
   ns := not_rdy;
   when others => null;
  end case;
end if;
end process;

end structural;
```

5.1 Single-step Enable Signal

The design uses a state machine to generate a single-step signal that replaces the clock signal. The design is based on a two-button reset-activate scheme that generates a single pulse upon a button press. In particular, with the above entity, btn0 activates the single step and btn1 resets the mechanism. User cannot press the two buttons at the same time this is outside the design specification.

The module Step (see Fig. 3) is a state machine with three states; not_ready, ready and pulse.

5.2 Behavioral Code

Below is the new_cascading_reg behavioral architecture. It loads new input x when user presses btn1. To load again user first presses btn0 then presses btn1. The new deisgn omits the load enable input port as in the cascading_reg4. Note that the code does not use the generic register code. It uses a process with two signals temp1 and temps as the placeholders.

```
entity new_cascading_reg is
port(x : in std_logic_vector(3 downto 0);
     z1, z2 : out std_logic_vector(3 downto 0);
btn0,btn1,ck: in std_logic);
end new_cascading_reg;

architecture Behavioral of new_cascading_reg is
signal en : std_logic;
signal temp1, temp2 : std_logic_vector(3 downto 0);
begin
-- single step, debounce (db)
-- btn0 to enter and btn1 to reset
process(ck)
type db_state is (not_rdy, rdy, pulse);
variable db_ns: db_state;
begin
if ck='1' and ck'event then
  case db_ns is
   when not_rdy => en <= '0';
   if btn1 = '1' then db_ns := rdy; end if;
   when rdy => en <= '0';
   if btn0 = '1' then db_ns := pulse; end if;
   when pulse => en <= '1';
   db_ns := not_rdy;
   when others => null;
  end case;
end if;
end process;
-- Single pulse en signal trigger cascading registers
process(en)
begin
if en='1' and en'event then
temp1 <= x; temp2 <= temp1;
end if;
end process;
-- wire temp1 to z1 and temp2 to z2
z1 <= temp1; z2 <= temp2;
end Behavioral;
```

5.6 Implement and Verify Correctness for both architectures.

6. Exercise

Design and implement 4-stage 2-bit wide cascading register

```
entity cascading4_reg2 is
port (   X :   in   std_logic_vector(1 downto 0);
        Z1 : out   std_logic_vector(1 downto 0);
```

```
          Z2 : out    std_logic_vector(1 downto 0);
          Z3 : out    std_logic_vector(1 downto 0);
          Z4 : out    std_logic_vector(1 downto 0);
          Btn0, btn1, ck :    in    std_logic);
end cascading4_reg2;
```