Electrical and Computer Engineering

Synopsis

This note covers three design problems on register buffers: Parallel to Serial Buffer, Serial to Parallel Buffer and Double Buffer. Problem 3 shows an implementation on FPGA.

Problem 1 Parallel-in Serial-out Register

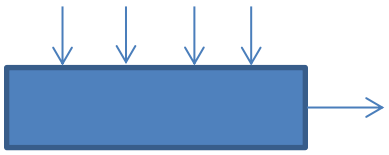

Fig. 1.1 Parallel-in Serial-out Shift Register

```
----------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- parallel-in serial-out register
-- After reset the register loads new data,
-- then shifts the n bits content out.
-- Once the leftmost bit at output,
-- the register loads the new data.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity p2s is
generic (n : natural := 2);
port (ck, reset : in std_logic;
      x: in  std_logic_vector(n-1 downto 0);
      z: out std_logic);
end p2s;
```

```
architecture Behavioral of p2s is
signal temp : std_logic_vector(n-1 downto 0);
begin
z <= temp(0);-- wire output
process(ck)
subtype my_int is integer range 0 to n-1;
variable count : my_int;
begin
if ck='1' and ck'event then
  if reset = '1' then temp <= x; count := 0; else
    if count < n-1 then
    temp <= '0'&temp(n-1 downto 1);
    count := count + 1;
    else
    temp <= x; count := 0;
    end if;
   end if;
 end if;
 end process;
end Behavioral;
```

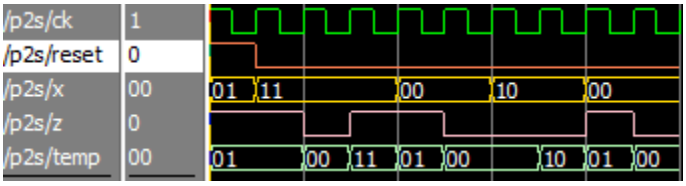

Fig. 1.2 Simulation Wave

Problem 2 Serial-in parallel-out register

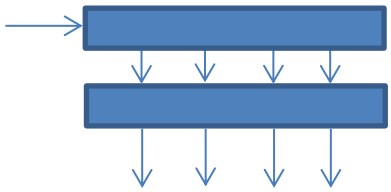

Fig. 2.1 Serial-in Parallel-out Register

```
----------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- serial-in parallel-out register
-- After reset the register loads a new bit
-- by right-shifting the content of temp1.
-- After n loads and temp2 <= temp1;
-- In this fashion, the circuit chunks bit
-- serial into a string of n-bit vectors.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity s2p is
generic (n : natural := 2);
port (x, ck, reset : in std_logic;
      z: out std_logic_vector(n-1 downto 0)
);
end s2p;
```

```
architecture Behavioral of s2p is
signal temp1, temp2 :
       std_logic_vector(n-1 downto 0);
begin
z <= temp2;-- wire output
process(ck)
subtype my_int is integer range 0 to n;
variable count : my_int;
begin
if ck='1' and ck'event then
  if reset = '1' then
  temp1 <= (others => '0');
  temp2 <= (others => '0');
  count := 0; else
    if count < n then
    temp1 <= x&temp1(n-1 downto 1);
    count := count + 1;
    else
    -- shift and chunk to temp2
    temp1 <= x&temp1(n-1 downto 1);
    temp2 <= temp1;
    -- one already one shift
    count := 1;
    end if;
   end if;
 end if;
 end process;
end Behavioral;
```

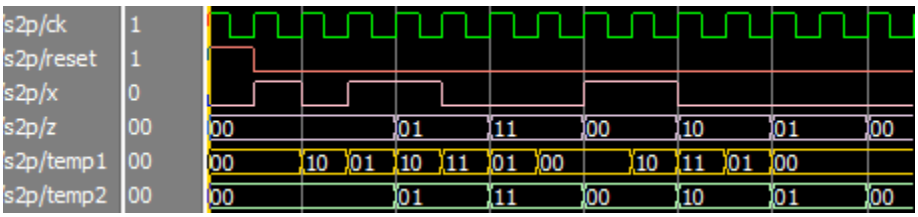

Fig. 2.2 Simulation Wave

Problem 3 Double buffer

Fig. 3 Double Buffer

```
----------------------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Double buffers: shift registers temp1, temp2
-- Two-state state machine. The states are
-- in_temp1_out_temp2, in_temp2_out_temp1
-- the state toggles as buffered register filled
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity double_buffer is
generic(n : natural);
port (ck, reset : in std_logic;
      x: in  std_logic;
      z: out std_logic;
      z1: out std_logic_vector(n-1 downto 0);
      z2: out std_logic_vector(n-1 downto 0);
      t1, t2: out std_logic);
end double_buffer;

architecture Behavioral of double_buffer is
signal temp1, temp2 : std_logic_vector(n-1 downto 0);
type my_state is (in_temp1_out_temp2, in_temp2_out_temp1);
signal n_s: my_state;
signal sel: std_logic; -- output mux select

begin
---------------------------------
-- debug signals to LEDs
---------------------------------
z1 <= temp1; z2 <= temp2; -- buffer contents
t1 <= sel; t2 <= not sel; -- present state
---------------------------------
```

```
process(ck)
variable count : integer;
begin
 if ck='1' and ck'event then
 if reset = '1' then
 n_s <= in_temp1_out_temp2;
 temp1 <= (others => '0');
 temp2 <= (others => '0');
 sel <= '0'; count := 0; else
  case n_s is
    when in_temp1_out_temp2 =>
     temp1 <= x&temp1(n-1 downto 1);
     temp2 <= '0'&temp2(n-1 downto 1);
     if count = n-1 then
       n_s <= in_temp2_out_temp1;
       sel <= '1'; -- mux temp1(0) to output
       count := 0;
     else
       sel <= '0'; -- mux temp2(0) to output
       count := count + 1;
     end if;
    when in_temp2_out_temp1 =>
     temp2 <= x&temp2(n-1 downto 1);
     temp1 <= '0'&temp1(n-1 downto 1);
     if count = n-1 then
       n_s <= in_temp1_out_temp2;
       sel <= '0'; -- mux temp2(0) to output
       count := 0;
     else
       sel <= '1'; -- mux temp1(0) to output
       count := count + 1;
     end if;
    when others => null;
  end case;
 end if;
end if;
end process;
```

```
-- output multiplexor
process(sel, temp1(0), temp2(0))
begin
if sel = '0' then z <= temp2(0); else
z <= temp1(0);
end if;
end process;

end Behavioral;

----------------------------------------
-- test bench n = 4
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity double_buffer_tb is
port (ck, btn0, btn1, reset : in std_logic;
      x: in  std_logic;
      z: out std_logic;
      z1: out std_logic_vector(3 downto 0);
      z2: out std_logic_vector(3 downto 0);
      t1, t2: out std_logic);
end double buffer tb;
architecture struc of double_buffer_tb is
signal en: std_logic;
component double_buffer
generic(n : natural);
port (ck, reset : in std_logic;
      x: in  std_logic;
      z: out std_logic;
      z1: out std_logic_vector(n-1 downto 0);
      z2: out std_logic_vector(n-1 downto 0);
       t1, t2: out std_logic);
end component;
begin
dut: double_buffer generic map (4)
port map(en, reset, x, z, z1, z2, t1, t2);
```

```
-- 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;
end struc;
```