

## Εργαστήριο 8: Σχεδίαση στο περιβάλλον ISE με VHDL

- Δημιουργία κυκλώματος ολισθητή (shifter) :
  - ✓ δημιουργήσετε ένα νέο project στο ISE με όνομα **day7** και δημιουργήσετε το παρακάτω κύκλωμα του ολισθητή.
  - ✓ Συνδέστε το κατάλληλα στο board και ελέγξτε το ότι λειτουργεί σωστά σε όλες τις περιπτώσεις.
    - Sin: είσοδοι από τα 8 switches
    - shL: button
    - shR: button
    - Sout: έξοδοι στα Led

```
entity shifter
port(
  Sin : in std_logic_vector(7 downto 0);
  Sout: out std_logic_vector(7 downto 0);
  shL : in std_logic;
  shR : in std_logic
);
end shifter;
```

```
architecture beh of shifter is
begin
```

```
p1: process( shL, shR, Sin )
begin
  sout <= (others=>'0');
  if (shL = '1') then
    sout <= sin(6 downto 0) & '0' ;
  end if ;
  if (shR = '1') then
    sout <= '0' & sin(7 downto 1) ;
  end if;
  if (shL = '1' and shR = '1') then
    sout <= sin ;
  end if ;
end process;
```

```
end beh;
```



## Δημιουργία ενός καταχωρητή 8-bit

Προσθέστε στο project σας ένα νέο αρχείο VHD με όνομα reg8.vhd

```
entity reg8 is
port(
  din : in std_logic_vector(7 downto 0);
  dout: out std_logic_vector(7 downto 0);
  clk : in std_logic;
  rst : in std_logic
);
end reg8;

architecture rtl of reg8 is
begin
  process ( clk, rst )
  begin
    if rs='1' then
      dout <= (others=>'0') ;
    elsif (clk'event and clk='1') then
      dout <= din ;
    end if;
  end process;
end rtl;
```

Για να χρησιμοποιήσετε τον καταχωρητή ανοίξτε το αρχείο του shifter και αρχικά δηλώσετε τον καταχωρητή στο σημείο:

```
....
...
```

architecture beh of shifter is

```
component reg8
port(
  din : in std_logic_vector(7 downto 0);
  dout: out std_logic_vector(7 downto 0);
  clk : in std_logic;
  rst : in std_logic
);
end component;

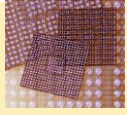
signal soutbus : std_logic_vector(7 downto 0);

begin
....
```

Το νέο σήμα **soutbus** το δηλώσαμε διότι το ζητούμενο είναι ο ολισθητής μας να αποθηκεύεται στον καταχωρητή και η έξοδος του καταχωρητή να φαίνεται στα Led.

Άρα σε όλο το process του shifter αλλάξτε το **sout** με **soutbus**.

Μετά το begin πρέπει να κάνετε ένα instance τον καταχωρητή ως εξής:



## Σχεδίαση Ψηφιακών Κυκλωμάτων σε FPGA

```
Ureg : reg8 port map (  
    Din => soutbus,  
    Dout => sout,  
    Clk => clk,  
    Rst => reset  
);
```

Πρέπει να δηλώσετε νέες εισόδους στον shifter τα νέα σήματα clk και reset.

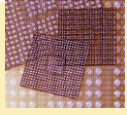
Υλοποιήστε και κάνετε demo το κύκλωμα.

## Φάση 2<sup>η</sup>

Για να αποφύγουμε να πατάμε τα button **shL**, **shR** και να δημιουργούνται πολλοί παλμοί '1' πρέπει να γίνει debounce το κάθε button, δηλαδή να αποφευχθεί ο θόρυβος που προκαλείται.

Προσθέστε στο project σας ένα νέο αρχείο VHD με όνομα debounce.vhd

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity debounce is  
port (  
    clk                : in  std_logic;  
    switch_input       : in  std_logic;  
    debounced_output  : out  std_logic := '0'  
);  
end debounce ;  
  
architecture behavioral of debounce is  
  
    type states is (noPushButton, ButtonPushed);  
    signal state      : states;  
  
    signal stop_counter      : std_logic := '0';  
    signal run_counter       : std_logic;  
    signal counter           : std_logic_vector(23 downto 0);  
  
begin  
  
state_machine: process (clk)  
begin  
    if clk'event and clk='1' then  
        case state is  
            when noPushButton =>  
                if (switch_input = '1' and stop_counter = '0') then  
                    state <= ButtonPushed;  
                    debounced_output <= '0';  
                    run_counter <= '1';  
                else  
                    state <= noPushButton;  
                    debounced_output <= '0';  
                    run_counter <= '0';  
                end if;  
            end case;  
        end if;  
    end process;
```



## Σχεδίαση Ψηφιακών Κυκλωμάτων σε FPGA

```
        end if;

        when ButtonPushed =>
            if stop_counter = '1' then
                state <= noPushButton;
                debounced_output <= '1';
                run_counter <= '0';
            else
                state <= ButtonPushed;
                debounced_output <= '0';
                run_counter <= '1';
            end if;

        when others =>
            debounced_output <= '0';
            run_counter <= '0';
            state <= noPushButton;
        end case;
    end if;

end process state_machine;

-- When the state machine goes 'high', it starts the counter (run_counter = 1).
-- The counter will count off from 0 to 11,500,000. When it is
-- done it will give stop_counter a '1', which will output 1 and change
-- state to 'low'.

    count_machine: process(clk, run_counter)
    begin
        if run_counter = '1' then
            if clk'event and clk = '1' then
                if counter = x"AF79E0" then -- x"AF79E0" then
                    stop_counter <= '1'; -- this hex value is 11,500,000
                else -- each cycle is 20 ns
                    stop_counter <= '0'; -- total time of counter is
                end if; -- 11,500,000 * 20 ns = 0.23 sec
            end if;

            counter <= counter + 1;
        end if;
        else
            counter <= x"000000";
        end if;
    end process count_machine;

end behavioral;
```

Όπως κάνατε και με τον καταχωρητή πρέπει δηλώσετε το :

```
component debounce
port (
    clk          : in    std_logic;
    switch_input : in    std_logic;
    debounced_output: out  std_logic := '0'
);
end component;
```

Χρησιμοποιήστε ένα instance του debounce για καθένα button.

Υλοποιήστε και κάνετε demo το κύκλωμα.