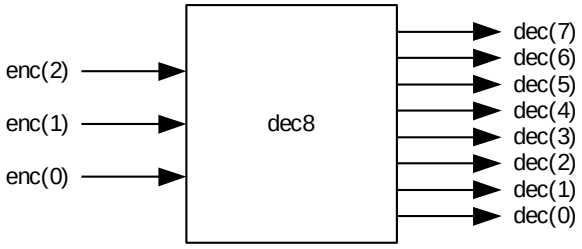
```
architecture binary_encoder of binary_encoder is
begin
        encode(1) <= a or b when exists='1' else 'Z';
        encode(0) <= a or ((not b) and c) when exists='1'
                else 'Z';
        exists <= a or b or c or d;
end binary_encoder;
```

Ζητούμενο: Να περιγραφεί σε behavioral αρχιτεκτονική και να προσομοιωθεί το ίδιο κύκλωμα.

Υπόδειξη: Χρησιμοποιήστε δομή if σε μια κατάλληλη σειρά ελέγχου των εισόδων για να εξασφαλίσετε την ύπαρξη προτεραιότητας.

Θέμα Α.2: Δυαδικός αποκωδικοποιητής 3 σε 8



Σχήμα 1.15 Δυαδικός αποκωδικοποιητής 3 σε *8*

Ένας δυαδικός αποκωδικοποιητής 3 σε 8 είναι ένα συνδυαστικό κύκλωμα που ενεργοποιεί διαφορετική έξοδο (μία από τις 8) για κάθε διαφορετικό συνδυασμό των 3 εισόδων.

Ζητούμενο: Να δοθεί η περιγραφή της οντότητας του δυαδικού αποκωδικοποιητή 3 σε 8 και της αρχιτεκτονικής του σε dataflow και behavioral VHDL. Κατόπιν να γίνει προσομοίωση και στις δύο αρχιτεκτονικές

Υπόδειξη: Χρησιμοποιήστε τον τύπο std_logic_vector τόσο για τις εισόδους όσο και για τις εξόδους. Για την behavioral αρχιτεκτονική χρησιμοποιήστε τη δομή case.

```
            -- Το clear έχει μεγαλύτερη προτεραιότητα
                        q <= '0';
                elsif preset = '1' then
                        q <= '1';
                elsif clk'event and clk='1' then
                        q <= d;
                end if;
        end process;
end bhv_dff;
```

Πρέπει να σημειωθεί εδώ ότι το κύκλωμα που προκύπτει από αυτή την περιγραφή δεν είναι υλοποιήσιμο σε όλες τις τεχνολογίες. Μερικές επιτρέπουν και τις δύο ασύγχρονες εισόδους, ενώ άλλες επιτρέπουν την χρήση μόνο της μιας.

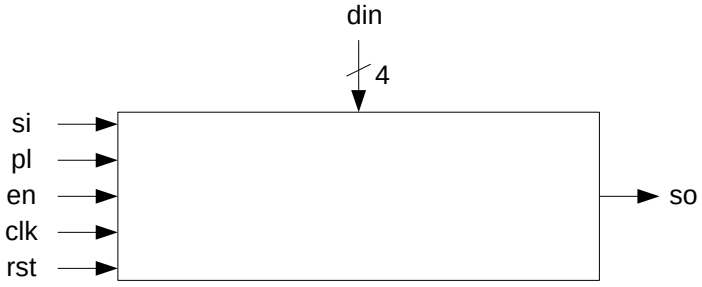
Ζητούμενα:

1. Με βάση την περιγραφή του D Flip-Flop να ορίσετε την behavioral περιγραφή ενός T Flip-Flop με ασύγχρονες εισόδους preset και clear.

   Υπόδειξη: Το Flip-Flop τύπου T είναι αυτό που σε κάθε παλμό του ρολογιού αλλάζει (toggle) η έξοδός του.
2. Με οδηγό την περιγραφή ενός D Flip-Flop, να περιγράψετε σε behavioral VHDL έναν καταχωρητή 4 bits με είσοδο παράλληλης φόρτωσης.

   Υπόδειξη: Να χρησιμοποιήσετε τον τύπο std_logic_vector για να περιγράψετε την είσοδο και την έξοδο του καταχωρητή.

Θέμα Β.2: Καταχωρητής ολίσθησης των 4 bits με παράλληλη φόρτωση



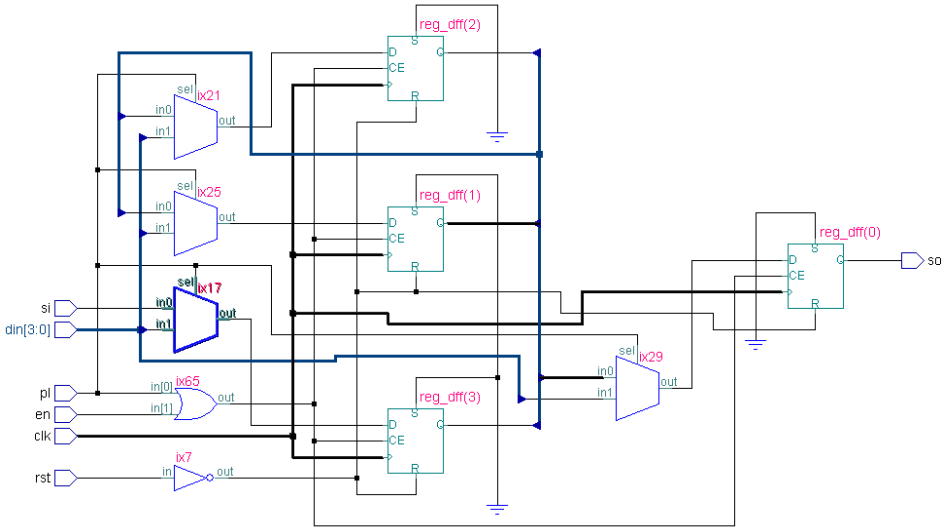
Σχ. 2.7: *Καταχωρητής δεξιάς ολίσθησης των 4 bits*

Ο καταχωρητής ολίσθησης (σχήμα 2.7) είναι ένα κύκλωμα το οποίο δέχεται μια παράλληλη είσοδο din (Data in) η οποία φορτώνεται μέσω του σύγχρονου

σήματος ενεργοποίησης παράλληλης φόρτωσης pl (Parallel Load). Η ενεργοποίηση της ολίσθησης γίνεται μέσω του σήματος en (Enable). Ο καταχωρητής έχει και μια σειριακή είσοδο si (Serial Input), καθώς και μια σειριακή έξοδο so (Serial Output). Κατά την δεξιά ολίσθηση το περιεχόμενο του καταχωρητή ολισθαίνει κατά μια θέση προς το LSB (bit 0). Το LSB βγαίνει στην έξοδο so και η είσοδος si περνά στο MSB του καταχωρητή. Αυτό γίνεται σε κάθε θετικό παλμό του ρολογιού clk. Η ασύγχρονη είσοδος rst (reset) μηδενίζει τα flip-flops του καταχωρητή ολίσθησης. Η περιγραφή του καταχωρητή ολίσθησης σε behavioral VHDL είναι η εξής:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity rshift_reg3 is
        port (
                clk,rst,si,en,pl: in std_logic;
                din: in std_logic_vector(3 downto 0);
                so: out std_logic);
end rshift_reg3;

architecture rtl of rshift_reg3 is
        signal dff: std_logic_vector(3 downto 0);
begin
        edge: process (clk,rst)
        begin
                if rst='0' then
                        dff<=(others=>'0');
                elsif clk'event and clk='1' then
                        if pl='1' then
                                dff<=din;
                        elsif en='1' then
                                dff<=si&dff(3 downto 1);
                        end if;
                end if;
        end process;
        so <= dff(0);
end rtl;
```

Με τη βοήθεια του Active-HDL εξομοιώστε την αρχιτεκτονική του παραδείγματος. Το κύκλωμα που προκύπτει από τον synthesizer για την περιγραφή αυτή είναι αυτό που φαίνεται στο σχήμα 2.8. Φαίνονται τα Flip-Flops που αποτελούν τον καταχωρητή, μαζί με το συνδυαστικό κύκλωμα που καθορίζει τη λειτουργία του shift register. Ελέγξτε την ορθότητα λειτουργίας του κυκλώματος αυτού και αν θα μπορούσε να σχεδιαστεί "με το χέρι" σε απλούστερη μορφή.



Σχ. 2.8: Καταχωρητής ολίσθησης των 4 bits – Κύκλωμα που προκύπτει από τον synthesizer

Ζητούμενο: Να περιγραφεί η οντότητα του καταχωρητή ολίσθησης με μια επιπλέον είσοδο (std_logic) η οποία θα επιλέγει ανάμεσα σε αριστερή και δεξιά ολίσθηση. Υπενθυμίζεται ότι στην αριστερή ολίσθηση η έξοδος είναι το MSB του καταχωρητή και η σειριακή είσοδος γίνεται από το LSB. Για την περιγραφή την οποία θα φτιάξετε να ελέγξετε και το κύκλωμα που προκύπτει από τον synthesizer, παρατηρώντας τις διαφορές που έχει από το κύκλωμα που δίνεται στο σχήμα 2.8

Υπόδειξη: Χρησιμοποιήστε την περιγραφή του καταχωρητή δεξιάς ολίσθησης, με έναν επιπλέον έλεγχο (μέσω εντολής case) για την επιλογή αριστερής ή δεξιάς ολίσθηση. Εναλλακτικά, δώστε μια δεύτερη λύση με χρήση της εντολής if.

Θέμα Β.3: Μετρητής 3 bit με είσοδο ενεργοποίησης και κρατούμενο εξόδου

Ο μετρητής που θα παρουσιάσουμε στα επόμενα είναι μονής κατεύθυνσης (δηλαδή μετράει μόνο προς τα πάνω) και διαθέτει ασύγχρονη είσοδο μηδενισμού

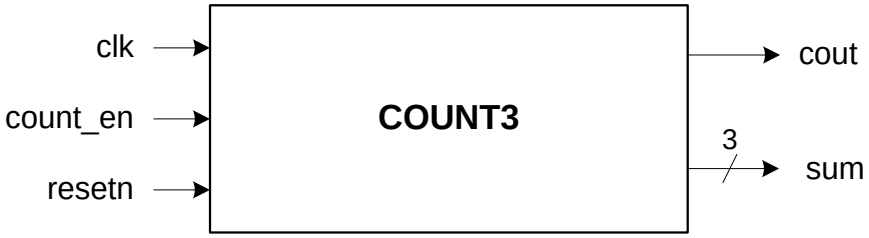
και σύγχρονη είσοδο ενεργοποίησης. Η είσοδος μηδενισμού είναι ενεργή σε λογικό ‘0’, ενώ η είσοδος ενεργοποίησης είναι ενεργή σε λογικό ‘1’.

Εάν χρησιμοποιηθεί το package ieee.std_logic_unsigned, αρκεί να χρησιμοποιηθούν οι συναρτήσεις “+” ή “-” όταν πρέπει να αυξηθεί ή να μειωθεί η τιμή του μετρητή (στο παράδειγμα που ακολουθεί η τιμή του μετρητή μόνο αυξάνει). Εάν ο μετρητής είναι δηλωμένος σαν std_logic_vector η τιμή του θα αλλάξει αυτόματα όταν όλα τα bits έχουν την τιμή ‘1’ (στην περίπτωση του “+”) και η τιμή του μετρητή θα μηδενιστεί. Εάν ο μετρητής πρέπει να σταματήσει σε μια τιμή, π.χ. “101” η τιμή αυτή πρέπει να ελεγχθεί πριν την πρόσθεση του +1. Ακολουθούν δύο αρχιτεκτονικές, μια με έλεγχο ορίου και μια χωρίς. Δοκιμάστε με τη βοήθεια του Active HDL τις δυο αυτές αρχιτεκτονικές.

Στην περίπτωση που δεν χρησιμοποιηθεί το package ieee.std_logic_unsigned για να γίνει η πρόσθεση ή η αφαίρεση ενός αριθμού σε ένα std_logic_vector θα πρέπει να γραφούν οι ακόλουθες γραμμές σε VHDL:

```
signal count: std_logic_vector (2 downto 0);
signal count_int: integer range (0 to 7);
…
count_int <= conv_integer (count);
count_int <= count_int+1;
count <= conv_std_logic_vector(count,3);
…
```

Οι γραμμές αυτές μετατρέπουν το std_logic_vector σε ακέραιο αριθμό, ώστε να είναι δυνατή η πρόσθεση του 1. Αφού γίνει η πρόσθεση, μετατρέπεται πάλι ο ακέραιος (με τη νέα τιμή) σε std_logic_vector.

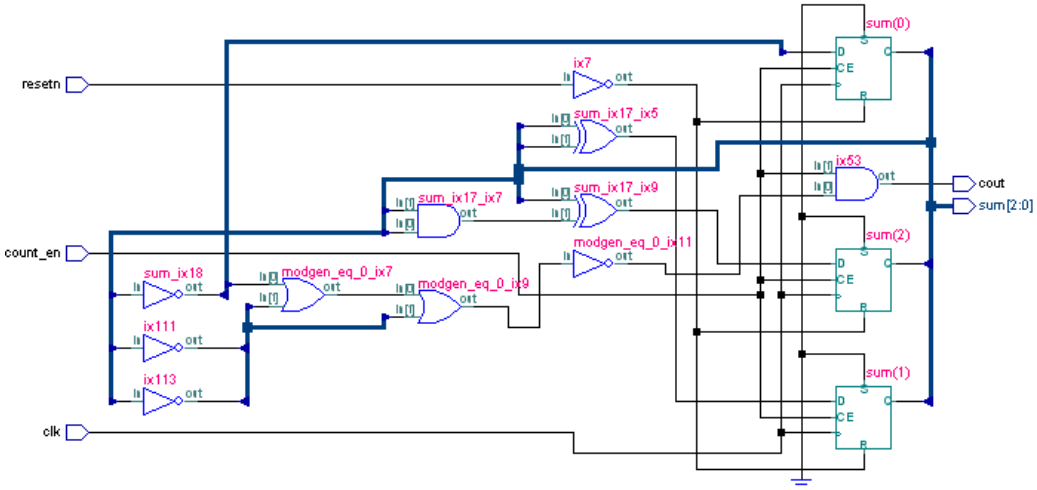


Σχ. 2.9: Μετρητής 3 bit

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity count3 is
        port(   clk,
                resetn,
                count_en        : in std_logic;
                sum             : out std_logic_vector(2 downto 0);
                cout            : out std_logic);
end;


architecture rtl_nolimit of count3 is
signal count: std_logic_vector(2 downto 0);
begin
        process(clk, resetn)
        begin
                if resetn='0' then
                        -- Κώδικας για την περίπτωση του reset (ενεργό χαμηλά)
                        count <= (others=>'0');
                elsif clk'event and clk='1' then
                        if count_en='1' then
                        -- Μέτρηση μόνο αν count_en = 1
                                count<=count+1;
                        end if;
                end if;
        end process;
        -- Ανάθεση τιμών στα σήματα εξόδου
```

```
        sum <= count;
        cout <= '1' when count=7 and count_en='1' else '0';
end rtl_nolimit;



architecture rtl_limit of count3 is
signal count : std_logic_vector(2 downto 0);
begin
        process(clk, resetn)
        begin
                if resetn='0' then
                -- Ασύγχρονος μηδενισμός
                        count <= (others=>'0');
                elsif clk'event and clk='1' then
                        if count_en = '1' then
                        -- Μέτρηση μόνο αν count_en=’1’
                                if count/=7 then
                                -- Αυξάνουμε το μετρητή μόνο αν
                                -- δεν είναι 7
                                        count <= count+1;
                                else
                                -- Αλλιώς τον μηδενίζουμε
                                        count<=(others=>'0');
                                end if;
                        end if;
                end if;
        end process;
        sum<= count;
        cout <= '1' when count=7 and count_en='1' else  '0';
end;
```

Το όριο του μετρητή στη δεύτερη αρχιτεκτονική είναι το 7, πράγμα που σημαίνει ότι το κύκλωμα που θα προκύψει από τον synthesizer θα είναι το ίδιο και στις δύο περιπτώσεις. Το σχήμα 2.10 δείχνει το κύκλωμα αυτό. Ελέγξτε την ορθότητα λειτουργίας του κυκλώματος αυτού και αν θα μπορούσε να σχεδιαστεί “με το χέρι” σε απλούστερη μορφή.



Σχ. 2.10: Κύκλωμα μετρητή 3 bits από τον synthesizer.

Στην περίπτωση που δεν χρησιμοποιηθεί το package ieee.std_logic_unsigned για να γίνει η πρόσθεση ή η αφαίρεση ενός αριθμού σε ένα std_logic_vector θα πρέπει να γραφούν οι ακόλουθες γραμμές σε VHDL, ώστε να γίνουν οι πράξεις με ακεραίους αριθμούς:

```
use ieee.std_logic_arith.all;
…
signal count: std_logic_vector (2 downto 0);
signal count_int: integer range (0 to 7);
…
count_int <= conv_integer (count);
count_int <= count_int+1;
count <= conv_std_logic_vector(count,3);
…
```

Οι γραμμές αυτές μετατρέπουν το std_logic_vector σε ακέραιο αριθμό, ώστε να είναι δυνατή η πρόσθεση του 1. Αφού γίνει η πρόσθεση, μετατρέπεται πάλι ο ακέραιος (με τη νέα τιμή) σε std_logic_vector. Η πρακτική αυτή δεν χρησιμοποιείται συνήθως για την πρόσθεση και την αφαίρεση, διότι οδηγεί σε

μακροσκελείς περιγραφές επιρρεπείς στα λάθη και δεν προσφέρει κάτι παραπάνω από την περιγραφή που προκύπτει με τη χρήση του std_logic_unsigned.

Ζητούμενα:

1. Βασιζόμενοι στην περιγραφή του μετρητή των 3 bits, να περιγράψετε έναν μετρητή up/down των 3 bits.
   Υπόδειξη: Χρησιμοποιήστε μια είσοδο επιλογής κατεύθυνσης: 1 για μέτρηση προς τα πάνω, 0 για μέτρηση προς τα κάτω και την εντολή case.
2. Βασιζόμενοι στην περιγραφή του μετρητή των 3 bits να περιγράψετε έναν up counter των 3 bits με παράλληλη είσοδο modulo (όριο μέτρησης) των 3 bits.
   Υπόδειξη: Χρησιμοποιήστε μια είσοδο std_logic_vector την οποία θα συγκρίνετε με την τιμή του μετρητή.
3. Ελέγξτε την ορθότητα λειτουργίας του κυκλώματος που δίνει ο synthesizer για τα ζητούμενα 1 και 2 και αν θα μπορούσε να σχεδιαστεί “με το χέρι” σε απλούστερη μορφή.