12. ΣΥΓΧΡΟΝΟΙ ΜΕΤΡΗΤΕΣ

1. Εισαγωγή

Σκοπός αυτού του κεφαλαίου είναι η γνωριμία και η κατανόηση της διαδικασίας για τη σχεδίαση ενός σύγχρονου μετρητή, με επιθυμητό μέτρο και η πειραματική επαλήθευση της λειτουργίας του.

Μετρητής είναι το ακολουθιακό κύκλωμα που αποτελείται από λογικές πύλες και στοιχεία μνήμης, το οποίο ακολουθεί μια καθορισμένη σειρά καταστάσεων και μεταβάλλει το περιεχόμενό του κατά 1 σε κάθε παλμό ρολογιού συγχρονισμού.

Οι μετρητές αποτελούν πολύ διαδεδομένα κυκλώματα που συναντώνται σε ψηφιακές συσκευές και ηλεκτρονικούς υπολογιστές.

Ο τρόπος σύνδεσης του ρολογιού στα Flip-Flops ορίζει δύο τύπους μετρητών:

- τους **σύγχρονους**, όπου το ρολόι εφαρμόζεται ταυτόχρονα σε όλα τα Flip-Flops και
- στους **ασύγχρονους**, όπου το ρολόι συνδέεται άμεσα μόνο στο Flip – Flop που αντιπροσωπεύει το λιγότερο σημαντικό ψηφίο του αριθμού μέτρησης.

Το πλήθος των τιμών μέτρησης ορίζει το **Μέτρο** του μετρητή (δηλαδή το MOD). Π.χ . μέτρο-8 σημαίνει ότι ο μετρητής μετρά 8 καταστάσεις, από 0 μέχρι το 7 (δηλαδή από το 000 στο 111 στο δυαδικό σύστημα). Η δυαδική σειρά μέτρησης μερικών μετρητών φαίνεται **Πίνακα 1**.

Ανάλογα με το μέτρο του μετρητή ορίζουμε το πλήθος n των Flip-Flops που απαιτούνται για τη σχεδίαση του μετρητή. Το μέγιστο πλήθος διαφορετικών συνδυασμών με n Flip-Flops είναι : 2^n. Πρέπει να ισχύει $\mathbf{2^n \geq M}$, όπου M το Modulo του μετρητή.

Αν το μέτρο του μετρητή είναι πέντε (Μέτρο-5) ο μετρητής μετρά από 0-4 (δυαδικός 000-100), ενώ αν το μέτρο του μετρητή είναι ένδεκα (Μέτρο-11) τότε μετρά από 0-10 (δυαδικός 0000-1010) κ.ο κ.

α/α	**A B C**	**A B C**	**A B C**
0	0 0 0	0 0 0	0 0 0
1	0 0 1	0 0 1	0 0 1
2	0 1 0	0 1 0	0 1 0
3	0 1 1	0 1 1	0 1 1
4	1 0 0	1 0 0	1 0 0
5	MOD-5	1 0 1	1 0 1
6		MOD-6	1 1 0
7			MOD-7
8			
9			

α/α	**A B C**	**A B C**	**A B C**
0	0 0 0	0 0 0 0	0 0 0 0
1	0 0 1	0 0 0 1	0 0 0 1
2	0 1 0	0 0 1 0	0 0 1 0
3	0 1 1	0 0 1 1	0 0 1 1
4	1 0 0	0 1 0 0	0 1 0 0
5	1 0 1	0 1 0 1	0 1 0 1
6	1 1 0	0 1 1 0	0 1 1 0
7	1 1 1	0 1 1 1	0 1 1 1
8	MOD-8	1 0 0 0	1 0 0 0
9		MOD-9	1 0 0 1
			MOD-10

Πίνακας 1. Η δυαδική σειρά μέτρησης μερικών μετρητών.

Η **μελέτη σχεδίασης του σύγχρονου μετρητή** περιλαμβάνει τα παρακάτω βήματα:

1. Είδος Μετρητή και επιθυμητό **Μέτρο μέτρησης** (Μέτρο-**X**) ή **Σειρά Μέτρησης**.
2. Πλήθος **n** (από σχέση $\mathbf{2^n \geq M}$) και είδος FF π. χ. JK ή SR τύπου PETr / NETr (Θετικής ή αρνητικής λογικής) (δίδονται ή επιλέγονται).
3. Πίνακας Καταστάσεων μέτρησης (παρούσα και επόμενη κατάσταση) και Καθορισμός των συναρτήσεων των εισόδων των FF (με τη βοήθεια του πίνακα διέγερσής των).
4. Απλοποιημένες εκφράσεις των λογικών συναρτήσεων των εισόδων των FF, που προκύπτουν από τους πίνακες Καρνώ, λαμβάνοντας υπ' όψιν μας κατά την απλοποίηση **οπωσδήποτε** τους αδιάφορους όρους, **αν** υπάρχουν. (Τους σημειώνουμε στον αρχικό πίνακα, τον λεγόμενο **πίνακα κλειδί** και τους μεταφέρουμε στους υπόλοιπους πίνακες για την κάθε είσοδο των FF).
5. Σχεδίαση του ακολουθιακού κυκλώματος του μετρητή από τις απλοποιημένες **εκφράσεις των συναρτήσεων λογικής** (**ΕΣΛ**) των εισόδων των FF.

2. Σχεδίαση σύγχρονου μετρητή Μέτρου-8 με JK-FF τύπου NETr

Να μελετηθεί η σχεδίαση σύγχρονου μετρητή Μέτρου-8 με JK-FF τύπου NETr (δηλαδή με αρνητική λογική, που σημαίνει ότι η

αλλαγή στην έξοδο του FF θα γίνεται κατά τη μετάβαση από 1 σε 0 στον παλμό του ρολογιού).

Λύση. Σύμφωνα με τα δεδομένα της εκφώνησης έχουμε τα εξής:

1. Σύγχρονος μετρητής με Μέτρο-8 (Μετρά από 0-7).
2. Πρέπει να ισχύει $\mathbf{2^n} \geq M = 8$ οπότε απαιτούνται **n** = 3 FF (τα JK-FF τύπου NETr).
3. Πίνακας καταστάσεων μέτρησης και Καθορισμός των εισόδων των FF.

α/α	**Παρούσα Κατάσταση Μετρητή (Π.Κ.)**			**Επόμενη Κατάσταση Μετρητή (Ε.Κ.)**			**Είσοδοι FF**					
	A	**B**	**C**	**A+**	**B+**	**C+**	**J_A**	**K_A**	**J_B**	**K_B**	**J_C**	**K_C**
0	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	0	0	X	1	X	X	1
2	0	1	0	0	1	1	0	X	X	0	1	X
3	0	1	1	1	0	0	1	X	X	1	X	1
4	1	0	0	1	0	1	X	0	0	X	1	X
5	1	0	1	1	1	0	X	0	1	X	X	1
6	1	1	0	1	1	1	X	0	X	0	1	X
7	1	1	1	0	0	0	X	1	X	1	X	1

Πίνακας 2. Σύγχρονος μετρητής με Μέτρο-8.

Οι στήλες για τις εισόδους των FF συμπληρώνονται με τη βοήθεια του πίνακα διέγερσης του τύπου του FF σύμφωνα με την παρούσα και την επόμενη κατάσταση μέτρησης (δηλαδή από την παρούσα κατάσταση **000** πάμε στην επόμενη κατάσταση **001**).

Επειδή ο παλμός του ρολογιού (CLK) εφαρμόζεται σε όλα τα FF ταυτόχρονα πρέπει να έχουμε:

- J_A=0, K_A=X στο A-FF ώστε να μην αλλάξει κατάσταση
- J_B=0, K_B=X στο B-FF ώστε να μην αλλάξει κατάσταση
- J_C=1, K_C=X στο C-FF ώστε να αλλάξει κατάσταση από '0' στο '1'.

Παρόμοια συνεχίζουμε και με τις άλλες καταστάσεις.

4. Απλοποιημένες εκφράσεις των λογικών συναρτήσεων των εισόδων των FF.

Προκύπτουν από τους πίνακες Καρνώ, λαμβάνοντας υπ' όψιν μας κατά την απλοποίηση οπωσδήποτε τους αδιάφορους όρους, αν υπάρχουν. (Τους σημειώνουμε στον αρχικό πίνακα, τον λεγόμενο πίνακα κλειδί και τους μεταφέρουμε στους υπόλοιπους πίνακες που αντιστοιχούν σε κάθε μια είσοδο).

Από τους πίνακες Καρνώ θα προκύψουν οι Ε.Λ.Σ. των εισόδων των FF με τη γνωστή διαδικασία.

ΣΗΜΕΙΩΣΗ: Στο παράδειγμα δε σημειώσαμε τον πίνακα κλειδί γιατί δεν υπάρχουν αχρησιμοποίητες καταστάσεις.

AB ➔	00	01	11	10
C 0			X	X
1		1	X	X

J_A = B.C

AB ➔	00	01	11	10
C 0	X	X		
1	X	X	1	

K_A = B.C

AB ➔	00	01	11	10
C 0		X	X	
1	1	X	X	1

J_B = C

AB ➔	00	01	11	10
C 0	X			X
1	X	1	1	X

K_B = C

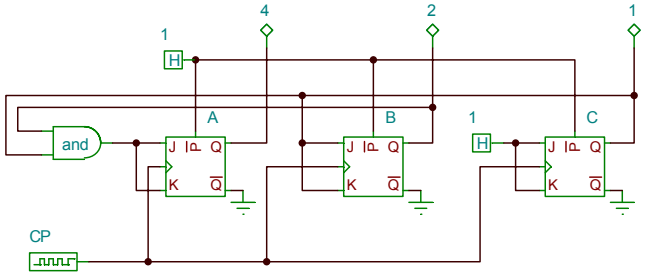
AB ➔	00	01	11	10
C 0	1	1	1	1
1	X	X	X	X

J_C = 1

AB ➔	00	01	11	10
C 0	X	X	X	X
1	1	1	1	1

K_C = 1

5. Σχεδίαση του ακολουθιακού κυκλώματος του μετρητή από τις απλοποιημένες εκφράσεις των λογικών συναρτήσεων (Α.Λ.Σ.) των εισόδων των FF. (Βλέπε **Εικόνα 1**).



Εικόνα 1. Σύγχρονος μετρητής με Μέτρο-8 με JK-FF.

3. Σχεδίαση σύγχρονου μετρητή που μετρά τις καταστάσεις 0,7,4,2,5,3 (με τη σειρά που δίδονται) με JK-FF τύπου NETr

Να μελετηθεί η σχεδίαση σύγχρονου μετρητή που μετρά τις καταστάσεις 0,7,4,2,5,3 (με τη σειρά που δίδονται) με JK-FF τύπου NETr.

Λύση.

α/α	**Π.Κ.**			**Ε.Κ.**			**Είσοδοι FF**					
	A	**B**	**C**	**A+**	**B+**	**C+**	**J_A**	**K_A**	**J_B**	**K_B**	**J_C**	**K_C**
0	0	0	0	1	1	1	1	X	1	X	1	X
7	1	1	1	1	0	0	X	0	X	1	X	1
4	1	0	0	0	1	0	X	1	1	X	0	X
2	0	1	0	1	0	1	1	X	X	1	1	X
5	1	0	1	0	1	1	X	1	1	X	X	0
3	0	1	1	0	0	0	0	X	X	1	X	1
ΠΙΝΑΚΑΣ ΚΛΕΙΔΙ												
1	0	0	1				D	D	D	D	D	D
6	1	1	0				D	D	D	D	D	D

Πίνακας 3. Πίνακας καταστάσεων σύγχρονου μετρητή καταστάσεων 0,7,4,2,5,3.

1. Σύγχρονος μετρητής μη δυαδικής μέτρησης, μετρά τις καταστάσεις 0,7,4,2,5,3.
2. Πρέπει να ισχύει $\mathbf{2^n \geq 7}$ (**το 2^n πρέπει να είναι μεγαλύτερο από το μεγαλύτερο βάρος συνδυασμού που βρίσκεται στη σειρά μέτρησης**) οπότε απαιτούνται **n** = 3 FF (τα JK-FF τύπου NETr).
3. Πίνακας καταστάσεων μέτρησης και Καθορισμός των εισόδων των FF.
4. Απλοποιημένες εκφράσεις των λογικών συναρτήσεων των εισόδων των FF.

Προκύπτουν από τους πίνακες Καρνώ, λαμβάνοντας υπ' όψιν μας κατά την απλοποίηση **οπωσδήποτε** τους αδιάφορους όρους. **Στην περίπτωση μας υπάρχουν και είναι οι καταστάσεις 1,6 που δε χρησιμοποιούνται στη μέτρηση. (Τους σημειώνουμε στον πίνακα κλειδί).**

Από τους πίνακες Καρνώ θα προκύψουν οι Ε.Λ.Σ. των εισόδων των FF με τη γνωστή διαδικασία.

Ο πίνακας κλειδί συμπληρώνεται απαραίτητα στις περιπτώσεις που δεν έχουμε δυαδική μέτρηση στο μέγιστο δυνατό πλήθος των συνδυασμών των χρησιμοποιούμενων FF ή γενικά σε περίπτωση μη δυαδικής μέτρησης.

AB ➔	00	01	11	10	AB ➔	00	01	11	10
C 0	1	1	D	X	C 0	X	X	D	1
1	D		X	D	1	D	X		1
J_A = C'					**K_A = B'**				

AB ➔	00	01	11	10	AB ➔	00	01	11	10
C 0	1	X	D	1	C 0	X	1	D	X
1	D	X	X	1	1	D	1	1	1
J_B = 1					**K_B = 1**				

AB ➔	00	01	11	10	AB ➔	00	01	11	10
C 0	1	1	D		C 0	X	X	D	X
1	D	X	X	X	1	D	1	1	
J_C = A'					**K_C = B**				

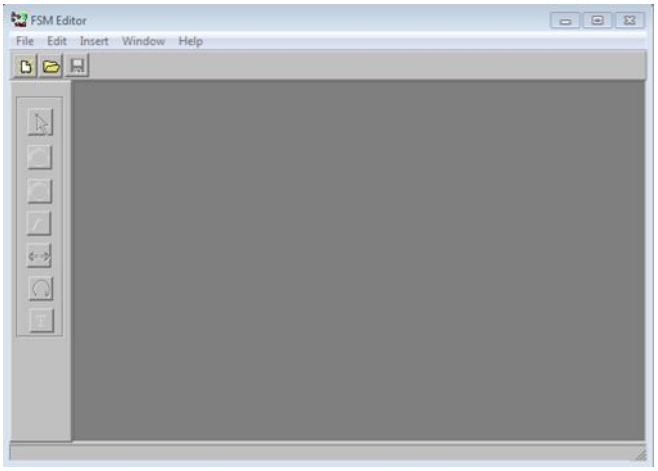
5. Σχεδίαση του ακολουθιακού κυκλώματος του μετρητή από τις απλοποιημένες εκφράσεις των λογικών συναρτήσεων (Α.Λ.Σ.) των εισόδων των FF. (Αφήνεται ως εξάσκηση).

4. Ασκήσεις

1. **Σχεδιάστε και προσομοιώστε το κύκλωμα του σύγχρονου μετρητή 0,7,4,2,5,3 με JK – FLIP-FLOP**
2. **Να σχεδιάσετε ένα σύγχρονο μετρητή mod-11 με JK Flip-Flops.**
3. **Να σχεδιάσετε ένα σύγχρονο μετρητή των καταστάσεων 0, 6, 3, 7, 1, 2, 5 με JK Flip-Flops.**
4. **Να σχεδιάσετε ένα σύγχρονο μετρητή MOD-8 ως μηχανή-πεπερασμένω-καταστάσεων, χρησιμοποιώντας τον State Editor του TINA.**

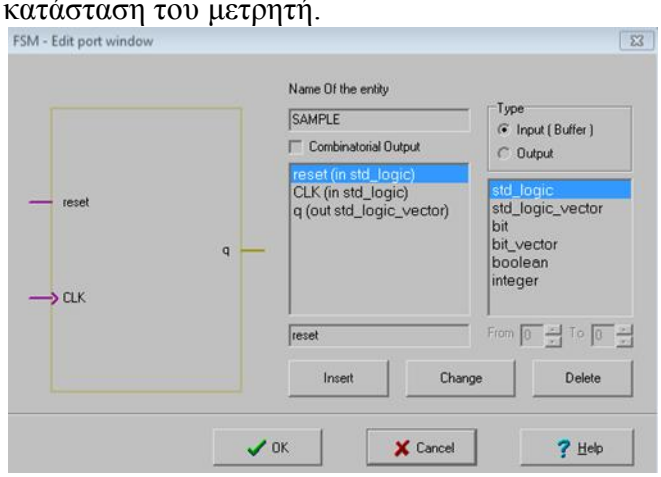
Λύση.

Ο Finite State Machine (FSM) Editor του TINA ξεκινά ανεξάρτητα από το TINA. Στην **Εικόνα 1** φαίνεται η αρχική διεπιφάνεια του FSM.

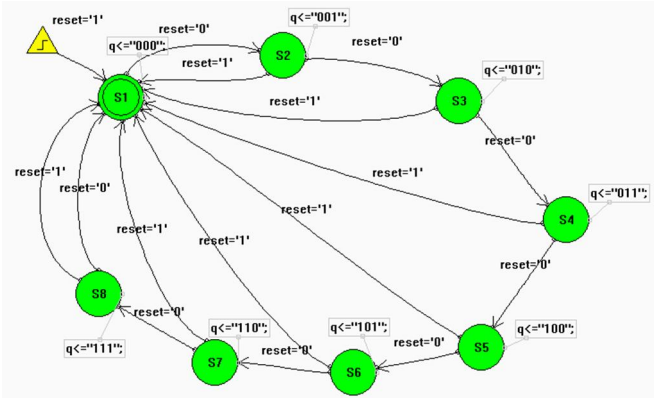


Εικόνα 1.

Με File➔New ξεκινάμε νέα εργασία, οπότε εκκινείται το πλαίσιο διαλόγου της **Εικόνας 2**.Ορίζουμε τη δομή του μετρητή ως μπλόκ διάγραμμα. Στη συγκεκριμένη περίπτωση θα έχει εισόδους το CLK για το ρολόι και το reset για ασύγχρονο μηδενισμό. Η έξοδος q θα είναι πίνακας 3bits που θα αποθηκεύει κάθε φορά την κατάσταση του μετρητή.



Εικόνα 2.



Εικόνα 3.

Πατώντας OK στον προηγούμενο διάλογο επιστρέφουμε στην κύρια επιφάνεια του FSM Editor, όπου σχεδιάζουμε τις καταστάσεις (S1,…,S8) και τη τις μεταβάσεις μεταξύ τους, όπως φαίνεται στην **Εικόνα 3**.

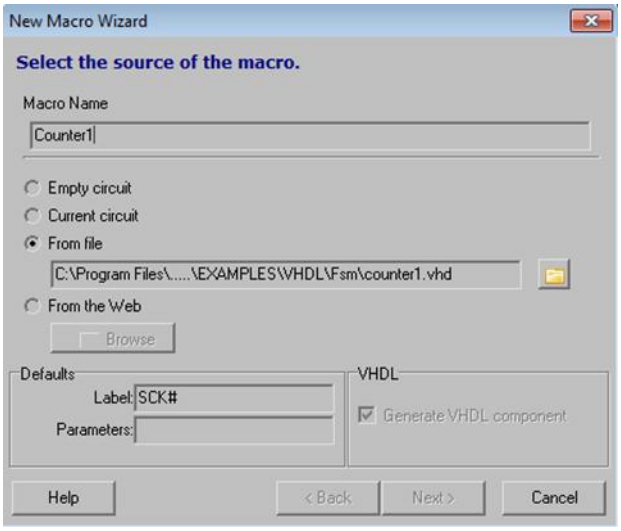
Το πλεονέκτημα του FSM Editor, είναι ότι το διάγραμμα καταστάσεων που υλοποιήσαμε μετατρέπεται αυτόματα σε VHDL, όπως φαίνεται στη **Λίστα 1**. Τον κώδικα αυτό τον εξάγουμε σε vhd αρχείο, μέσω της εντολής **File ➔ Export (VHDL code)**.

```
-- Generated   : 11/16/2013 5:51:49 PM
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity SAMPLE is
        port (
                reset: in std_logic;
                CLK: in std_logic;
                q: out std_logic_vector ( 2 downto
0 ));
end;

architecture SAMPLE_arch of SAMPLE is
-- SYMBOLIC ENCODED state machine: SMachine0
type SMachine0_type is (S1,S2,S3,S4,S5,S6,S7,S8);
signal SMachine0: SMachine0_type := S1;
begin
-- concurrent signals assignments
-------------------------------------------------
-----
-- Machine: SMachine0
-------------------------------------------------
-----
SMachine0_machine: process (CLK)
begin
if reset='1' then
        SMachine0 <= S1;
elsif CLK'event and CLK = '1' then
        --  Set  default  values  for  registered
outputs/signals and for variables
        case SMachine0 is
                when S1 =>
                q<="000";
                if reset='0' then SMachine0 <= S2;
                end if;
                when S2 =>
                q<="001";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S3;
                        end if;
                when S3 =>
                q<="010";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S4;
                end if;
                when S4 =>
                q<="011";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S5;
                end if;
                when S5 =>
                q<="100";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S6;
                end if;
                when S6 =>
                q<="101";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
```

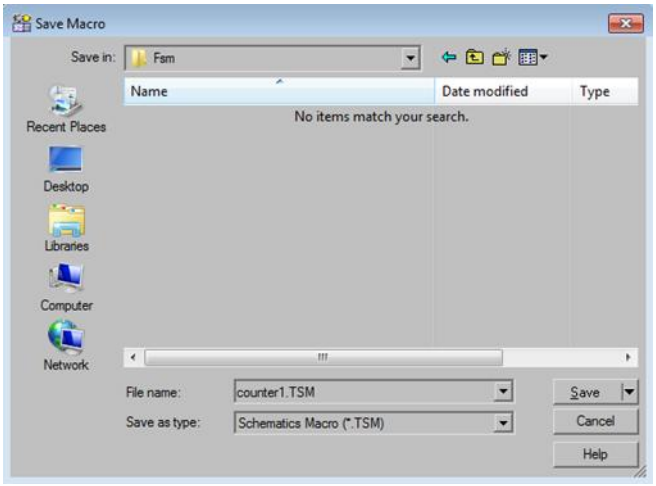
```
                        SMachine0 <= S7;
                end if;
                when S7 =>
                q<="110";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S8;
                end if;
                when S8 =>
                q<="111";
                if reset='1' then SMachine0 <= S1;
                elsif reset='0'then
                        SMachine0 <= S1;
                end if;
                when others =>
                null;
        end case;
end if;
end process;
end SAMPLE arch;
```

Λίστα 1.

Μέσα από το περιβάλλον του TINA επιλέγουμε Tools ➔ New Macro Wizard. Εκκινείται το πλαίσιο διαλόγου της **Εικόνας 4**. Στο Macro Name δίνουμε το όνομα που θέλουμε να έχει το μπλόκ σύμβολο του μετρητή, εδώ το Counter 1. Τσεκάρουμε την επιλογή From File και ορίζουμε τη διαδρομή προς το αρχείο counter1.vhd, όπου προηγούμενα αποθηκεύσαμε τον κώδικα VHDL περιργραφής της λειτουργίας του μετρητή.

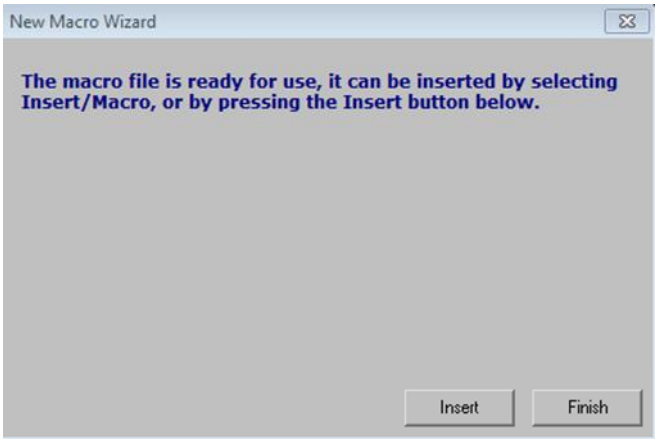


Εικόνα 4.

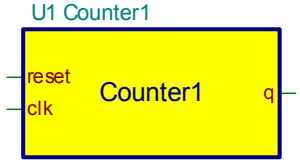


Εικόνα 5.

Πατάμε στο Next, οπότε εκκινείται το πλαίσιο διαλόγου της **Εικόνας 5** που μας ζητά τη διαδρομή που θα αποθηκευτεί ο μεταγλωττισμένος κώδικας TSM λειτουργίας του μετρητή. Δίνουμε όνομα και θέση αποθήκευσης και πατάμε στο Save. Τελικά εμφανίζεται το μήνυμα της **Εικόνας 6** που μας λέει πως μπορούμε στο εξής να εισάγουμε το μπλοκ σύμβολο του μετρητή σε κάθε σχηματικό του TINA. Πατώντας Insert μπορούμε άμεσα να εισάγουμε το σύμβολο του μετρητή μας, όπως φαίνεται στην **Εικόνα 7**.

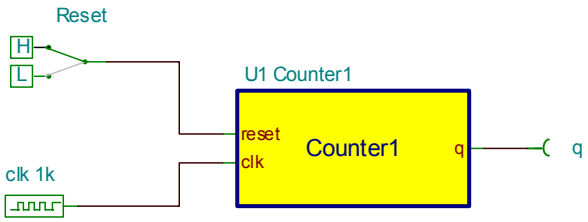


Εικόνα 6.



Εικόνα 7.

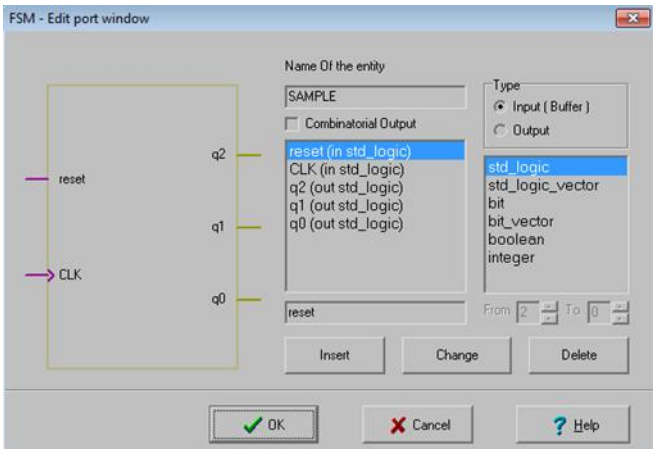
Κάνουμε τις επιπλέον συνδέσεις που φαίνονται στην **Εικόνα 8**. Το Pin που έχουμε επιλέξει είναι το **Bus Pin** από την παλέτα Meters και όχι το στάνταρ Voltage Pin, γιατί το q είναι πίνακας 3bits.



Εικόνα 8.

Το κύκλωμα είναι έτοιμο για διαδραστική VHDL προσομοίωση.

Αν θέλουμε να βλέπουμε καθένα bit εξόδου ξεχωριστά μπορούμε να κάνουμε μια μικρή αλλαγή στη δομή του μπλόκ του μετρητή. Επιστρέφουμε στον FSM Editor και στην περιγραφή του Counter1 και επιλέγουμε File ➔ Edit Ports. Επανεκκινείται το πλαίσιο διαλόγου της **Εικόνας 9**, όπου κάνουμε τις αλλαγές ώστε να φαίνονται τα bits εξόδου q2, q1, q0. Στη συνέχεια με διπλό κλικ πάνω σε κάθε κατάσταση αντικαθιστούμε τον κώδικα λειτουργίας της με τις αντίστοιχες εντολές του **Πίνακα 1**.

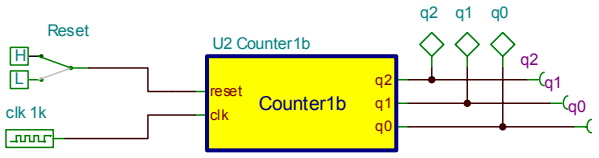


Εικόνα 9.

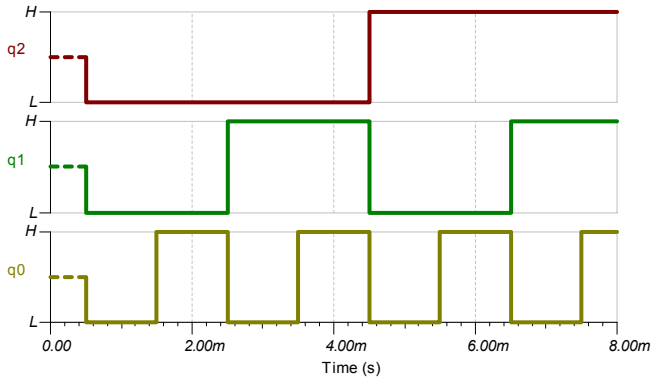
S1	S2	S3	S4
q2<='0'; q1<='0'; q0<='0';	q2<='0'; q1<='0'; q0<='1';	q2<='0'; q1<='1'; q0<='0';	q2<='0'; q1<='1'; q0<='1';
S5	**S6**	**S7**	**S8**
q2<='1'; q1<='0'; q0<='0';	q2<='1'; q1<='0'; q0<='1';	q2<='1'; q1<='1'; q0<='0';	q2<='1'; q1<='1'; q0<='1';

Πίνακας 1.

Κάνουμε πάλι εξαγωγή του VHDL κώδικα στο αρχείο Counter1b.vhd και επιστρέφουμε στο TINA, και ξανά με Tools ➔ New Macro εισάγουμε ένα νέο μπλοκ και το συνδέουμε με τον καινούργιο κώδικα VHDL. Κάνουμε τις συνδέσεις όπως φαίνεται στην **Εικόνα 10** και με Analysis ➔ Digital VHDL Simulation, έχουμε τα χρονικά αποτελέσματα της **Εικόνας 11**.



Εικόνα 10.

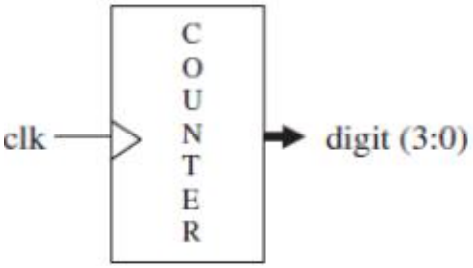


Εικόνα 11.

Υλοποιήσεις με VHDL

Παράδειγμα. Μετρητής ενός ψηφίου με χρήση IF

Το επόμενο κύκλωμα είναι το μπλοκ διάγραμμα ενός μετρητή 1-ψηφίου (0-->9-->0). Περιέχει ένα bit εισόδου (clk) και μια 4-bit έξοδο (digit).



Η υλοποίησή του γίνεται με χρήση της IF. Η μεταβλητή temp χρησιμοποιείται για να δημιουργήσει τα 4 Flip-Flops που είναι απαραίτητα για να αποθηκεύσουμε την 4-bit έξοδο.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

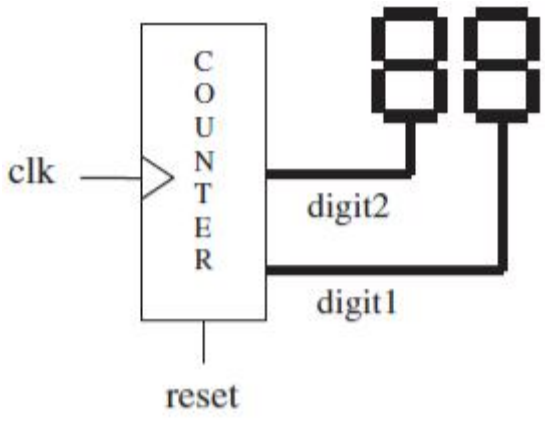
```
ENTITY counter IS
PORT (clk : IN STD_LOGIC;
digit : OUT INTEGER RANGE 0 TO 9);
END counter;

ARCHITECTURE counter OF counter IS
BEGIN
count: PROCESS(clk)
VARIABLE temp :
INTEGER RANGE 0 TO 10;
BEGIN
IF (clk'EVENT AND clk='1') THEN
temp := temp + 1;
IF (temp=10) THEN temp := 0;
END IF;
END IF;
digit <= temp;
END PROCESS count;
END counter;
```

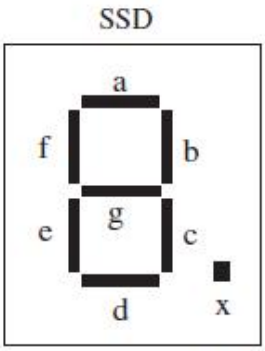
Ο προηγούμενος κώδικας δεν έχει είσοδο reset ή άλλη ενδογενή αρχικοποίηση του temp (οπότε και toy digit). Οπότε η αρχική τιμή του temp στο πραγματικό κύκλωμα μπορεί να είναι οποιαδήποτε 4-bit τιμή.

Παράδειγμα. Μετρητής ενός ψηφίου με χρήση WAIT UNTIL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY counter IS
PORT (clk : IN STD_LOGIC;
digit : OUT INTEGER RANGE 0 TO 9);
END counter;

ARCHITECTURE counter OF counter IS
BEGIN
PROCESS -- no sensitivity list
VARIABLE temp : INTEGER RANGE 0 TO
10;
BEGIN
WAIT UNTIL (clk'EVENT AND clk='1');
temp := temp + 1;
IF (temp=10) THEN temp := 0;
END IF;
digit <= temp;
END PROCESS;
END counter;
```

Παράδειγμα, Διψήφιος μετρητής με έξοδο σε διάταξη 7 segment display (SSD)





Input: "xabcdefg"

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY counter IS
PORT (clk, reset : IN
STD_LOGIC;
digit1, digit2 : OUT
STD_LOGIC_VECTOR (6 DOWNTO 0));
END counter;

ARCHITECTURE counter OF counter IS
BEGIN
PROCESS(clk, reset)
VARIABLE temp1:
INTEGER RANGE 0 TO 10;
VARIABLE temp2:
INTEGER RANGE 0 TO 10;
BEGIN
IF (reset='1') THEN
temp1 := 0;
temp2 := 0;
ELSIF (clk'EVENT AND clk='1') THEN
temp1 := temp1 + 1;
IF (temp1=10) THEN
temp1 := 0;
temp2 := temp2 + 1;
IF (temp2=10) THEN
temp2 := 0;
END IF;
END IF;
END IF;
---- BCD to SSD conversion: -----
```

```
CASE temp1 IS
WHEN 0 => digit1 <= "1111110"; --7E
WHEN 1 => digit1 <= "0110000"; --30
WHEN 2 => digit1 <= "1101101"; --6D
WHEN 3 => digit1 <= "1111001"; --79
WHEN 4 => digit1 <= "0110011"; --33
WHEN 5 => digit1 <= "1011011"; --5B
WHEN 6 => digit1 <= "1011111"; --5F
WHEN 7 => digit1 <= "1110000"; --70
WHEN 8 => digit1 <= "1111111"; --7F
WHEN 9 => digit1 <= "1111011"; --7B
WHEN OTHERS => NULL;
END CASE;
CASE temp2 IS
WHEN 0 => digit2 <= "1111110"; --7E
WHEN 1 => digit2 <= "0110000"; --30
WHEN 2 => digit2 <= "1101101"; --6D
WHEN 3 => digit2 <= "1111001"; --79
WHEN 4 => digit2 <= "0110011"; --33
WHEN 5 => digit2 <= "1011011"; --5B
WHEN 6 => digit2 <= "1011111"; --5F
WHEN 7 => digit2 <= "1110000"; --70
WHEN 8 => digit2 <= "1111111"; --7F
WHEN 9 => digit2 <= "1111011"; --7B
WHEN OTHERS => NULL;
END CASE;
END PROCESS;
END counter;
```

Παράδειγμα της FOR/LOOP:

```
FOR i IN 0 TO 5 LOOP
x(i) <= enable AND w(i+2);
y(0, i) <= w(i);
END LOOP;
```

Μια σημαντική παρατήρηση σε σχέση με την FOR/LOOP (αντίστοιχη με την GENERATE), είναι ότι κατά δύο όριά της θα πρέπει να είναι σταθερά. Π.χ. η δήλωση FOR I IN 0 TO CHOICE LOOP, όπου CHOICE είναι είσοδος (μη στατική), γενικά δεν είναι συνθέσιμη.

Παράδειγμα της WHILE/LOOP:

```
WHILE (i < 10) LOOP
WAIT UNTIL clk'EVENT AND clk='1';
(other statements)
END LOOP;
```

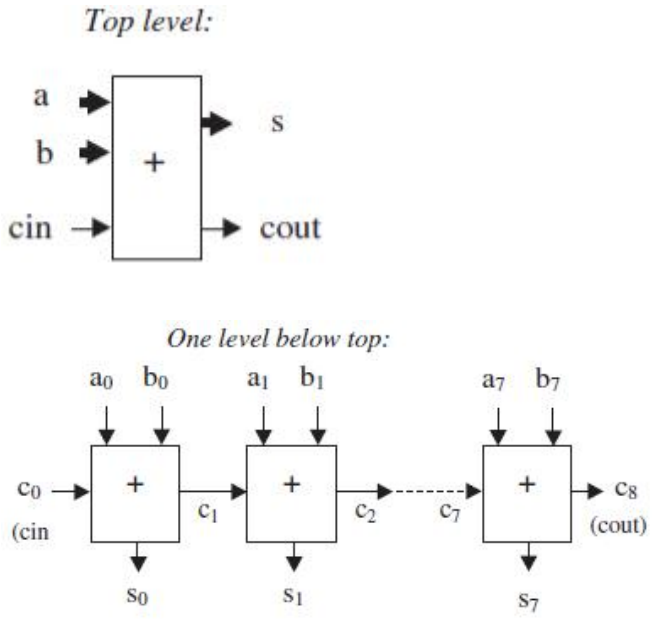
Παράδειγμα με την EXIT: Στον παρακάτω κώδικα, το EXIT δεν υπονοεί έξοδο από την τρέχουσα επανάληψη του βρόγχου, αλλά μια προσδιορισμένη έξοδο (δηλαδή ακόμα και αν είναι εντός της εμβέλειας δεδομένων, η δήλωση LOOP θα θεωρηθεί ολοκληρωμένη). Στην περίπτωση αυτή ο βρόγχος θα τερματιστεί επειδή μια τιμή διαφορετική από «0» βρίσκεται στο διάνυσμα δεδομένων:

```
FOR i IN data'RANGE LOOP
CASE data(i) IS
WHEN '0' => count:=count+1;
WHEN OTHERS => EXIT;
END CASE;
END LOOP;
```

Παράδειγμα με την NEXT: Στο επόμενο παράδειγμα η NEXT κάνει το LOOP να χάσει μια επανάληψη όταν i=skip:

```
FOR i IN 0 TO 15 LOOP
NEXT WHEN i=skip;
-- jumps to next iteration
(...)
END LOOP;
```

Υλοποίηση Carry Ripple Adder



Η πρώτη υλοποίηση είναι τύπου generic, δηλαδή ανεξάρτητη από το πλήθος των bits. Η δεύτερη είναι ειδικά για 8bit αριθμούς.

Στην πρώτη περίπτωση θα χρησιμοποιήσουμε διανύσματα και την FOR/LOOP, ενώ στη δεύτερη ακεραίους και την IF.

```
--Solution 1: Generic, with VECTORS

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder IS
GENERIC (length : INTEGER := 8);
PORT ( a, b: IN
STD_LOGIC_VECTOR   (length-1   DOWNTO
0);
cin: IN
STD_LOGIC;
s: OUT
STD_LOGIC_VECTOR   (length-1   DOWNTO
0);
cout: OUT
STD_LOGIC);
END adder;

ARCHITECTURE adder OF adder IS
BEGIN
PROCESS (a, b, cin)
```

```
VARIABLE carry :
STD_LOGIC_VECTOR (length DOWNTO 0);
BEGIN
carry(0) := cin;
FOR i IN 0 TO length-1 LOOP
s(i) <= a(i) XOR b(i) XOR carry(i);
carry(i+1) := (a(i) AND b(i)) OR
(a(i) AND
carry(i)) OR (b(i) AND carry(i));
END LOOP;
cout <= carry(length);
END PROCESS;
END adder;
```

```
--Solution 2: non-generic,
--with INTEGERS

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder IS
PORT ( a, b: IN
INTEGER RANGE 0 TO 255;
c0: IN STD_LOGIC;
s: OUT INTEGER RANGE 0 TO 255;
c8: OUT STD_LOGIC);
END adder;

ARCHITECTURE adder OF adder IS
BEGIN
PROCESS (a, b, c0)
VARIABLE temp :
INTEGER RANGE 0 TO 511;
BEGIN
IF (c0='1') THEN temp:=1;
ELSE temp:=0;
END IF;
temp := a + b + temp;
IF (temp > 255) THEN
c8 <= '1';
temp := temp---256;
ELSE c8 <= '0';
END IF;
s <= temp;
END PROCESS;
END adder;
```

Προβλήματα ρολογιού (Bad clocking)

Ο μεταγλωττιστής γενικά δε θα μπορεί να συνθέσει κώδικες που να περιέχουν αναθέσεις στο ίδιο σήμα και στις δύο μεταβάσεις του σήματος αναφοράς (ρολογιού) (δηλαδή και στην ακμή ανόδου και στην ακμή καθόδου). Αυτό συμβαίνει συνήθως όταν η τεχνολογία που χρησιμοποιούμε (π.χ. CPLDs) αποτελείται μόνο από flip-flop μοναδικής ακμής. Σε τέτοιες περιπτώσεις ο μεταγλωττιστής μπορεί να δείξει μήνυμα «signal does not hold value after clock edge» ή αντίστοιχο.

Ως παράδειγμα θα θεωρήσουμε την περίπτωση ενός μετρητή που αυξάνεται σε κάθε ακμή ρολογιού (δηλαδή και στην ακμή ανόδου και στην ακμή καθόδου). Μια υλοποίηση είναι η επόμενη:

```
PROCESS (clk)
BEGIN
IF(clk'EVENT AND clk='1') THEN
counter <= counter + 1;
ELSIF(clk'EVENT AND clk='0') THEN
counter <= counter + 1;
END IF;
...
END PROCESS;
```

Στην περίπτωση αυτή, εκτός από το παραπάνω μήνυμα που μπορεί να εμφανίσει ο μεταγλωττιστής, μπορεί να δηλώσει ότι το σήμα counter οδηγείται πολλαπλά. Όπως και να έχει, η μεταγλώττιση θα σταματήσει.

Μια άλλη παρατήρηση αφορά την EVENT και τη συνθήκη ελέγχου της. Η δήλωση IF(clk'EVENT AND clk='1') είναι σωστή, αλλά χρησιμοποιώντας την απλά ως IF(clk'EVENT), θα οδηγήσει το μεταγλωττιστή να υποθέσει μιαν εξορισμού τιμή ελέγχου (π.χ. την AND clk='1') ή να τυπώσει μήνυμα: clock not locally stable. Για παράδειγμα ας θεωρήσουμε και πάλι, την περίπτωση του μετρητή που πρέπει να αυξηθεί και στις δύο ακμές του ρολογιού. Μια υλοποίηση είναι η επόμενη:

```
PROCESS (clk)
BEGIN
IF(clk'EVENT) THEN
counter := counter + 1;
END IF;
...
END PROCESS;
```

Υποτίθεται ότι η PROCESS θα τρέξει κάθε φορά που το clk αλλάζει, οπότε θα περιμέναμε ο μετρητή να αυξάνεται δύο φορές σε κάθε κύκλο ρολογιού. Ωστόσο, για το λόγου που αναφέραμε παραπάνω, αυτό δε θα συμβεί. Αν ο μεταγλωττιστής υποθέσει μιαν εξορισμού τιμή, θα παραχθεί το λάθος κύκλωμα, γιατί μόνον μια ακμή ρολογιού θα ληφθεί υπόψη. Αν δεν υποτεθεί εξορισμού τιμή, τότε θα τυπωθεί μήνυμα σφάλματος και δε θα συνεχιστεί η μεταγλώττιση.

Αν ένα σήμα βρίσκεται σε λίστα ευαισθησίας, αλλά δεν εμφανίζεται σε καμία από τις αναθέσεις που αποτελούν την PROCESS, τότε είναι πιθανό ο μεταγλωττιστής να το αγνοήσει. Π.χ. έστω:

```
PROCESS (clk)
BEGIN
counter := counter + 1;
...
END PROCESS;
```

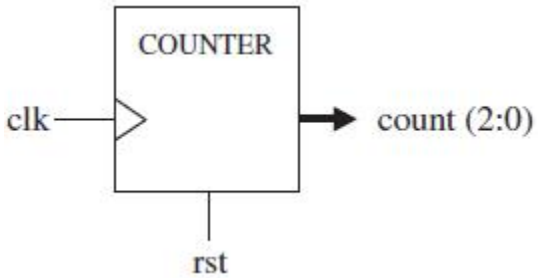
Ενώ με τον κώδικα αυτό θα περιμέναμε το σήμα counter να αυξάνεται κάθε φορά που θα συμβαίνει κάποιο γεγονός στο clk (είτε κατά τη θετική είτε κατά την αρνητική ακμή), αντίθετα ο μεταγλωττιστής θα τυπώσει «ignored unnecessary pin clk».

Σε αντίθεση με τις περιπτώσεις που περιγράφηκαν παραπάνω, ο κώδικας με τις δύο διαδικασίες που περιγράφεται στη συνέχεια, θα συντεθεί σωστά από κάθε μεταγλωττιστή. Ωστόσο, σημειώνουμε ότι έχουμε χρησιμοποιήσει διαφορετικό σήμα σε κάθε διαδικασία.

```
PROCESS (clk)
BEGIN
IF(clk'EVENT AND clk='1') THEN
x <= d;
END IF;
END PROCESS;
----------------------
PROCESS (clk)
BEGIN
IF(clk'EVENT AND clk='0') THEN
y <= d;
END IF;
END PROCESS;
```

Υλοποίηση μετρητή για μελέτη των VARIABLE και SIGNAL

Θα υλοποιήσουμε το μετρητή 0-7 του επόμενου σχήματος. Δύο λύσεις θα παρουσιαστούν Η πρώτη θα χρησιμοποιήσει σύγχρονη VARIABLE ανάθεση. Η δεύτερη σύγχρονη SIGNAL ανάθεση.



Για την πρώτη υλοποίηση ο κώδικας είναι:

```
--Solution 1: With a VARIABLE

ENTITY counter IS
PORT ( clk, rst: IN BIT;
count: OUT INTEGER RANGE 0 TO 7);
END counter;

ARCHITECTURE counter OF counter IS
BEGIN
PROCESS (clk, rst)
VARIABLE temp: INTEGER RANGE 0 TO 7;
BEGIN
IF (rst='1') THEN
temp:=0;
ELSIF (clk'EVENT AND clk='1') THEN
temp := temp+1;
END IF;
count <= temp;
END PROCESS;
END counter;
```

Η λύση 1 είναι ένα παράδειγμα του ότι η VARIABLE μπορεί να παράγει καταχωρητές. Αυτό οφείλεται στο ότι η ανάθεση της γίνεται σε μετάβαση άλλου σήματος (clk) και η τιμή της φεύγει από τη διαδικασία.

Για τη δεύτερη υλοποίηση ο κώδικας είναι:

```
-- Solution 2: With SIGNALS only

ENTITY counter IS
PORT ( clk, rst: IN BIT;
count: BUFFER INTEGER RANGE 0 TO 7);
END counter;

ARCHITECTURE counter OF counter IS
BEGIN
PROCESS (clk, rst)
BEGIN
IF (rst='1') THEN
count <= 0;
ELSIF (clk'EVENT AND clk='1') THEN
count <= count + 1;
END IF;
END PROCESS;
END counter;
```

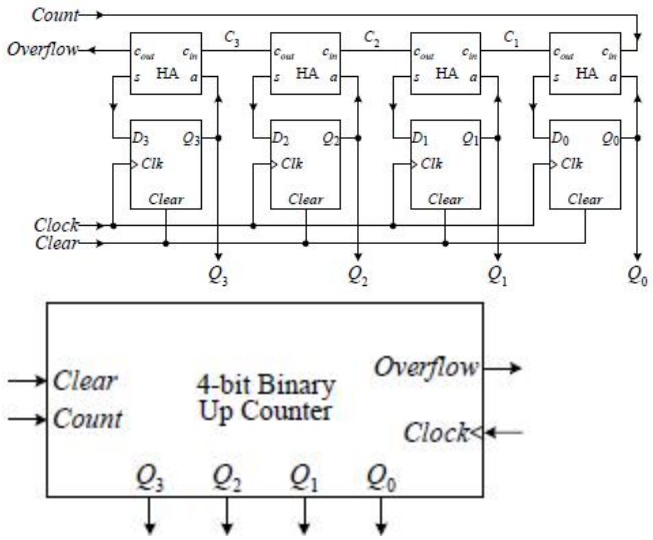
Η λύση 2, χρησιμοποιεί μόνο SIGNALS. Σημειώνουμε, ότι επειδή δε χρησιμοποιούμε βοηθητικό σήμα, το count πρέπει να δηλωθεί ως BUFFER, γιατί και του τίθεται μια τιμή αλλά και χρησιμοποιείται εσωτερικά (για ανάγνωση). Βλέπουμε επίσης ότι το SIGNAL όπως και η VARIABLE μπορεί να χρησιμοποιηθεί σε ακολουθιακό κώδικα.

Και στις δύο λύσεις δε χρησιμοποιήσαμε το std_logic_1164 γιατί δεν κάναμε χρήση STD_LOGIC δομών δεδομένων.

Από καθεμία από τις δύο λύσεις δύο flip-flops υποτίθενται (για να κρατήσουν την 3bit έξοδο count).

Υλοποίηση δυαδικού μετρητή 4bit μέτρησης προς τα πάνω

Θα υλοποιήσουμε τον 4bit μετρητή με τη δομή και το μπλοκ διάγραμμα που φαίνεται στη συνέχεια. Η λειτουργία του συνοψίζεται στην επόμενο πίνακα.



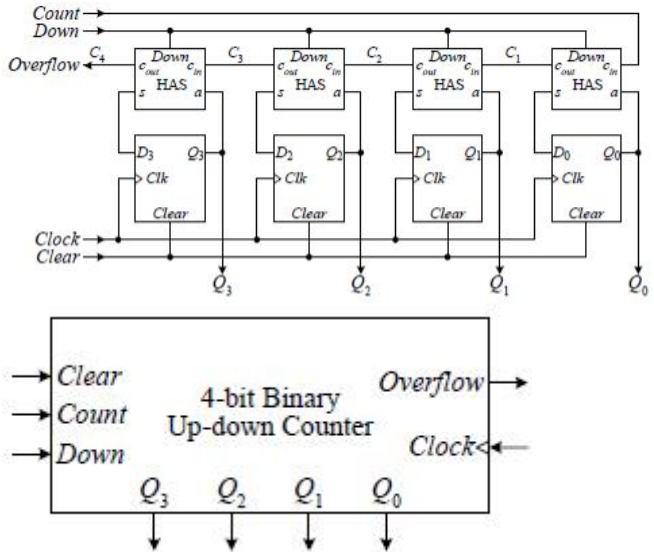
Clear	Count	Λειτουργία
1	X	Θέτει το μετρητή στο μηδέν
0	0	Καμία αλλαγή
0	1	Μετρά προς τα πάνω

Στη συνέχεια κάνουμε υλοποίηση με VHDL με χρήση αρχιτεκτονικής περιγραφής συμπεριφοράς. Η δήλωση USE IEEE.STD_LOGIC_UNSIGNED.ALL χρειάζεται για να πραγματοποιήσουμε πράξεις με STD_LOGIC_VECTOR.Το εσωτερικό σήμα value χρησιμοποιείται για να αποθηκεύσουμε την τρέχουσα τιμή μέτρησης. Όταν το clear γίνει 1, το value θα γίνει 0000 με χρήση της OTHERS => ‘0’. Διαφορετικά, αν το count γίνει 1, το value θα αυξηθεί κατά 1 στην επόμενη ακμή ανόδου του ρολογιού. Επιπλέον η μέτρηση που αποθηκεύεται στο count αποδίδεται στην έξοδο Q του μετρητή με τη χρήση της σύγχρονης δήλωσης Q<=value, γιατί είναι έξω από το μπλοκ της PROCESS.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
-- need this to
-- add STD_LOGIC_VECTORs

ENTITY counter IS PORT (
Clock: IN STD_LOGIC;
Clear: IN STD_LOGIC;
Count: IN STD_LOGIC;
Q : OUT
STD_LOGIC_VECTOR(3 DOWNTO 0));
END counter;

ARCHITECTURE Behavioral OF counter
IS
SIGNAL value:
STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
PROCESS (Clock, Clear)
BEGIN
IF Clear = '1' THEN
value <= (OTHERS => '0');
-- 4-bit vector of 0, same as "0000"
ELSIF (Clock'EVENT AND Clock='1')
THEN
IF Count = '1' THEN
value <= value + 1;
END IF;
END IF;
END PROCESS;
Q <= value;
END Behavioral;
```

Υλοποίηση δυαδικού μετρητή μέτρησης πάνω-κάτω

Θα υλοποιήσουμε τον δυαδικό μετρητή μέτρησης πάνω-κάτω με κύκλωμα και μπλοκ διάγραμμα που εικονίζονται στη συνέχεια.



Ο επόμενος πίνακας συνοψίζει τη συμπεριφορά λειτουργίας του μετρητή.

Clear	Count	Down	Λειτουργία
1	X	X	Θέτει το μετρητή στο μηδέν
0	0	X	Καμία αλλαγή
0	1	0	Μέτρηση προς τα πάνω
0	1	1	Μέτρηση προς τα κάτω

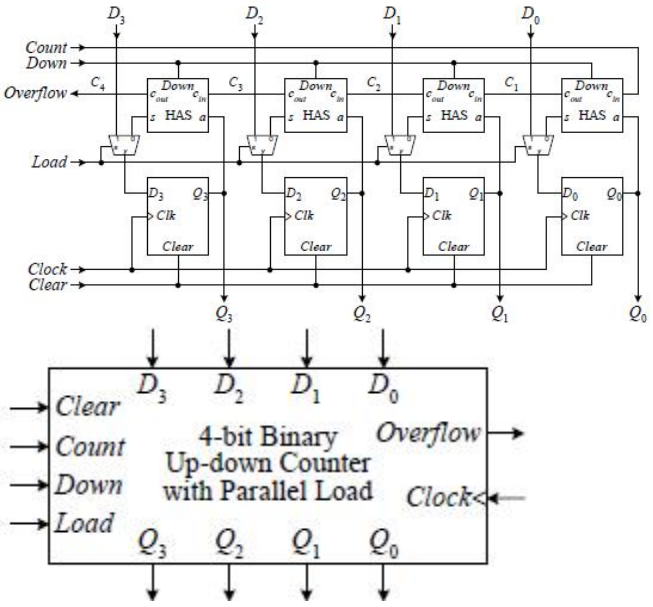
Η υλοποίηση με VHDL και χρήση αρχιτεκτονικής συμπεριφοράς φαίνεται στη συνέχεια.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY udcounter IS PORT (
Clock: IN STD_LOGIC;
```

```
Clear: IN STD_LOGIC;
Count: IN STD_LOGIC;
Down: IN STD_LOGIC;
Q: OUT INTEGER RANGE 0 TO 15);
END udcounter;

ARCHITECTURE Behavioral OF udcounter
IS
BEGIN
PROCESS (Clock, Clear)
VARIABLE value:
INTEGER RANGE 0 TO 15;
BEGIN
IF (Clear = '1') THEN
value := 0;
ELSIF (Clock'EVENT AND Clock='1')
THEN
IF (Count = '1') THEN
IF (Down = '0') THEN
value := value + 1;
ELSE
value := value - 1;
END IF;
END IF;
END IF;
Q <= value;
END PROCESS;
END Behavioral;
```

Υλοποίηση πάνω-κάτω δυαδικού μετρητή με παράλληλο φόρτο

Στη συνέχεια θα υλοποιήσουμε το δυαδικό μετρητή μέτρησης πάνω-κάτω. Η δομή του και το μπλοκ διάγραμμά του φαίνονται στην επόμενη εικόνα.



Ο επόμενος πίνακας συνοψίζει τη συμπεριφορά λειτουργίας του μετρητή.

Clear	Load	Count	Down	Λειτουργία
1	X	X	X	Αρχικοποιεί το μετρητή στο μηδέν
0	0	0	X	Καμία αλλαγή
0	0	1	0	Μέτρηση προς τα πάνω
0	0	1	1	Μέτρηση προς τα κάτω
0	1	X	X	Φόρτωση τιμής

Η υλοποίηση με VHDL αφήνεται ως άσκηση.