15. ΜΕΛΕΤΗ ΚΑΤΑΧΩΡΗΤΩΝ

1. Εισαγωγή

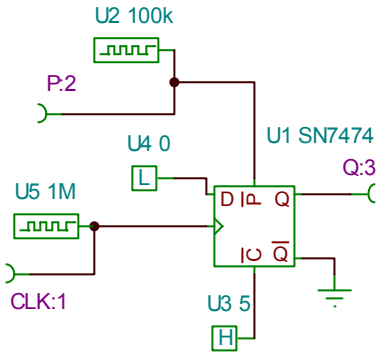
Ένας καταχωρητής (register) είναι μια ομάδα από δυαδικά κύτταρα αποθήκευσης, που είναι κατάλληλα για να κρατάνε δυαδικές πληροφορίες. Μια ομάδα FFs αποτελεί έναν καταχωρητή, αφού κάθε FF μπορεί να αποθηκεύσει ένα bit πληροφορίας. Ένας καταχωρητής των n-bits περιέχει n-FFs και άρα είναι σε θέση να αποθηκεύσει οποιαδήποτε δυαδική πληροφορία περιέχει n-bits. Επιπλέον των FFs, ένας καταχωρητής μπορεί να περιέχει πύλες για να εκτελούν διάφορες λειτουργίες επεξεργασίας δεδομένων. Με την ευρύτερη έννοια ένας καταχωρητής αποτελείται από ένα σύνολο FFs και από πύλες για την επίτευξη της μεταφοράς των πληροφοριών. Τα FFs κρατούν τις δυαδικές πληροφορίες και οι πύλες ελέγχουν το πότε και πως θα μεταφερθούν νέες πληροφορίες μέσα στον καταχωρητή.

Υπάρχουν διάφοροι τύποι καταχωρητών που είναι διαθέσιμοι σαν τσίπς MSI. Ο απλούστερος δυνατός τύπος αποτελείται μονάχα από FFs χωρίς εξωτερικές πύλες.

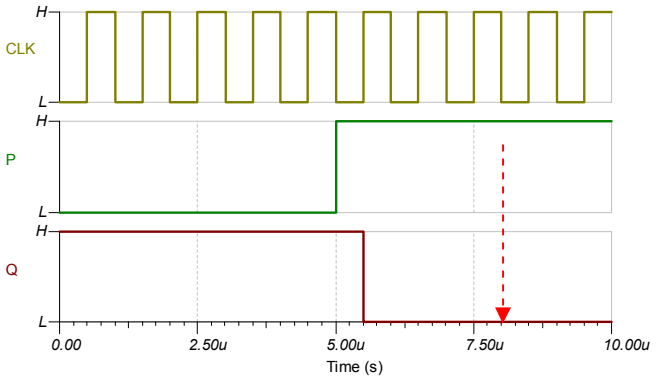
Στη συνέχεια θα δούμε παραδείγματα λειτουργίας καταχωρητών που χρησιμοποιούν D ή JK τύπου FFs.

2. Επανάληψη χαρακτηριστικών λειτουργίας D-FF

Για το λόγο αυτό επαναλλαμβάνουμε την αρχή λειτουργίας του D-FF όπως φαίνεται στο κύκλωμα της **Εικόνας 1** και με το διάγραμμα χρονισμού της **Εικόνας 2**. Όπως βλέπουμε η πυροδότηση του D-FF γίνεται στη θετική ακμή του ρολογιού. Προφανώς η έξοδος Q του D-FF είναι ίση με την είσοδό του, αλλά με καθυστέρηση ίση με το χρόνο που θα έλθει ο θετικός παλμός ρολογιού.

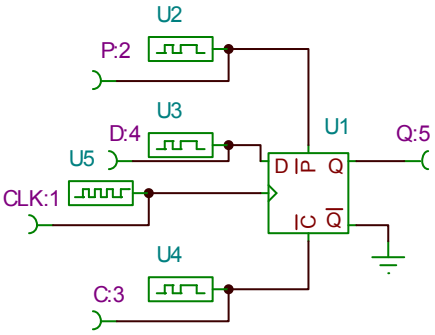


Εικόνα 1. Κύκλωμα D-FF υπό μελέτη.

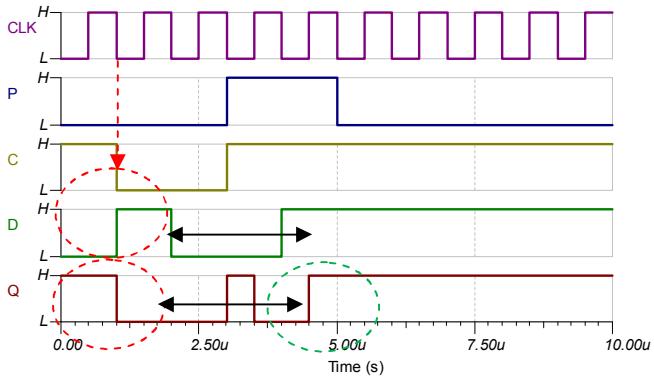


Εικόνα 2. Διάγραμμα χρονισμού κυκλώματος Εικόνας 1.

Στη συνέχεια δοκιμάζουμε αλλαγές στις εισόδους Preset και Clear όπως φαίνεται για το κύκλωμα της **Εικόνας 3** στο διάγραμμα χρονισμού της **Εικόνας 4**.



Εικόνα 3. Κύκλωμα D-FF για τη μελέτη της επίδρασης των Preset (P) και Clear (C).



Εικόνα 4. Διάγραμμα χρονισμού κυκλώματος **Εικόνας 3**.

Όταν η είσοδος Preset = 0, τότε η έξοδος είναι Q = 1. (κόκκινοι κύκλοι)

Όταν είναι η είσοδος Clear = 0, τότε η έξοδος είναι Q = 0. (μαύρα οριζόντια βέλη)

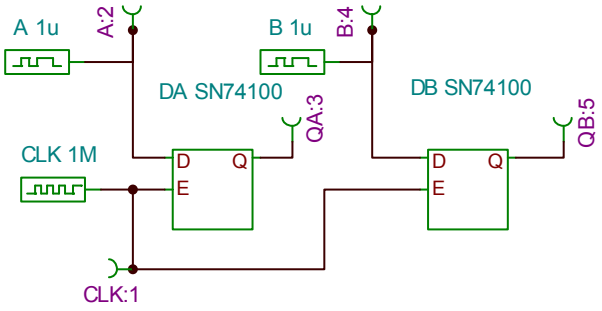
Οι είσοδοι Preset και Clear προφανώς δεν πρέπει να είναι ταυτόχρονα 0. Στο παράδειγμά μας παραπάνω παρότι το Preset = 0, μόλις το Clear = 0 , τότε η Q = 0.

Ενώ μπορούν προφανώς να είναι ταυτόχρονα 1, για να λειτουργεί το D-FF.

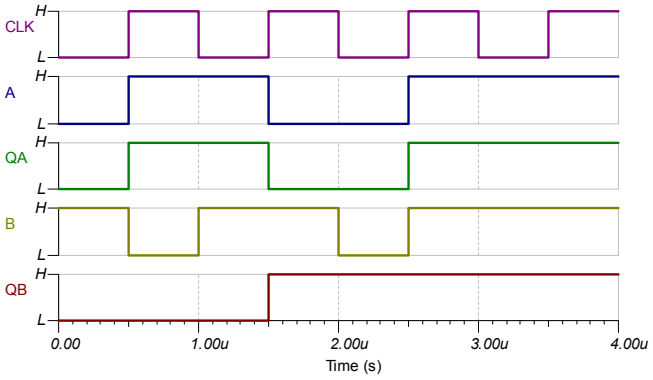
Εξηγήστε πως μπορεί να εμφανίζεται Q=1 μέσα στον πράσινο κύκλο με D=0.

3. Παράλληλος καταχωρητής 2bit με D-Flip-Flop

Αποτελεταί από δύο D-FF. Η φόρτωση γίνεται παράλληλα από τις γεννήτριες Α και Β. Για δεδομένες τιμές στις γεννήτριες Α και Β και για το σήμα ρολογιού όπως φαίνεται στο διάγραμμα χρονισμού, πως δικαιολογείται ότι QA = A ενώ QB ≠ B;



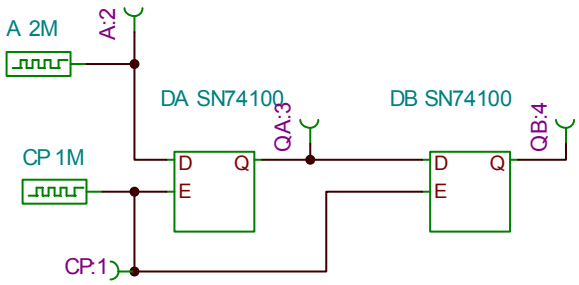
Εικόνα 5. Παράλληλος καταχωρητής 2bit με D-Flip-Flop.



Εικόνα 6. Διάγραμμα χρονισμού **Εικόνας 5**.

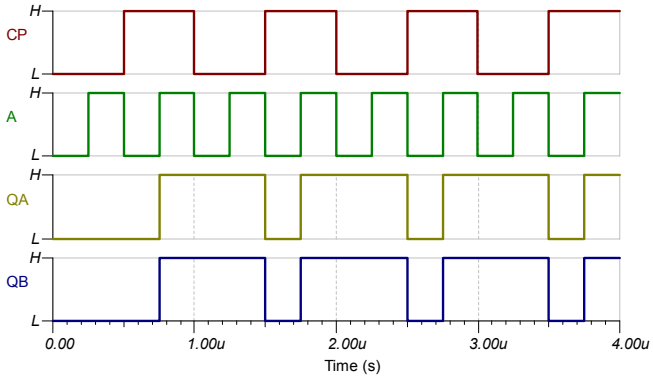
4. Καταχωρητής ολίσθησης 2bit με D-Flip-Flop

Σχεδιάστε τον επόμενο καταχωρητή ολίσθησης 2bit. Να παράγετε το διάγραμμα χρονισμού του για τα πρώτα 4μs λειτουργίας και να εξηγήσετε τη μορφή των QA και QB. Δοκιμάστε διάφορες συχνότητες ρολογιού για το CP και το Α. Σχολιάστε τα αποτελέσματα.

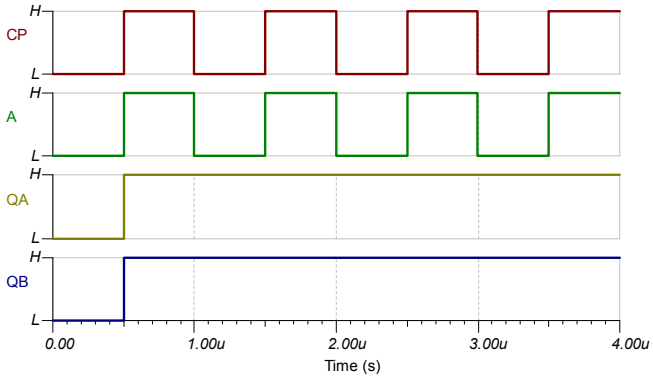


Εικόνα 7. Καταχωρητής ολίσθησης 2bit με D-FF.

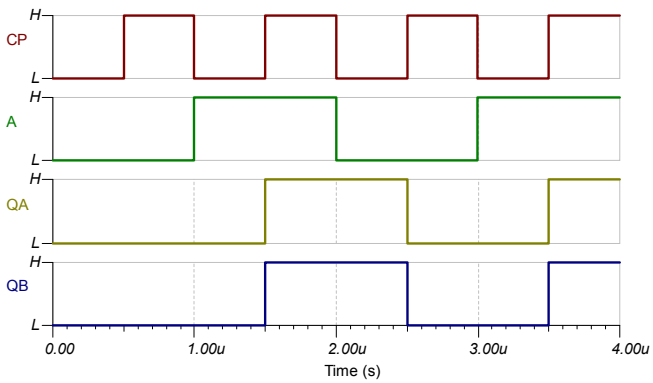
Στα επόμενα παραδείγματα έχουμε χρησιμοποιήσει στο ρόλο του Α γεννήτρια ρολογιού συχνότητας 2, 1 και 0.5 MHz αντίστοιχα.



Εικόνα 8. f_A = 2MHz



Εικόνα 9. f_A = 1MHz



Εικόνα 10. $f_A = 0.5MHz$

5. Παράλληλος καταχωρητής 4bit με D-Flip-Flop

Στην **Εικόνα 11** βλέπουμε έναν καταχωρητή 4bit με D-FFs. Όταν η είσοδος του ρολογιού πυροδοτεί όλα τα FFs, οι είσοδοί τους εκείνη τη στιγμή μεταφέρονται μέσα στον καταχωρητή. Από τις 4 εξόδους μπορούμε να πάρουμε τις πληροφορίες όποτε το θελήσουμε που βρίσκονται αποθηκευμένες εκείνη τη στιγμή στον καταχωρητή.

Ένα FF μπορεί να χρησιμοποιηθεί σε ακολουθιακά κυκλώματα με ρολόι, εφόσον ενεργοποιείται στην ακμή του παλμού και όχι στη διάρκειά του. Αυτό σημαίνει ότι τα FFs του καταχωρητή πρέπει να είναι ακμοπυρόδοτα ή τύπου αφέντη-σκλάβου**. Ένα σύνολο από FFs που είναι ευαίσθητα στη διάρκεια του παλμού λέγεται συνήθως «μανδαλωτής» (latch), ενώ αν είναι ευαίσθητα στην ακμή του παλμού λέγεται καταχωρητής (register)**. Στη συνέχεια θα υποθέτουμε ότι όλες οι ομάδες FFs στα σχέδιά μας είναι καταχωρητές και ότι όλα τα FFs είναι ακμοπυρόδοτα ή τύπου αφέντη – σκλάβου.

Η μεταφορά νέων πληροφοριών μέσα σε έναν καταχωρητή λέγεται «**φόρτωση**» (loading) του καταχωρητή. Αν όλα τα bits ενός καταχωρητή φορτώνονται ταυτόχρονα, με ένα μόνο παλμό του ρολογιού, λέμε ότι η φόρτωση γίνεται **παράλληλα**.

Αν εφαρμόσουμε θετικό παλμό στο CP θα φορτωθούν παράλληλα οι είσοδοι από τα DATA στον καταχωρητή της **Εικόνας 11**. Με αυτή την κατασκευή, αν θέλουμε να αφήσουμε το περιεχόμενο του καταχωρητή αναλλοίωτο, πρέπει να απαγορέψουμε στον παλμό του ρολογιού να φτάσει στον ακροδέκτη E του FF.

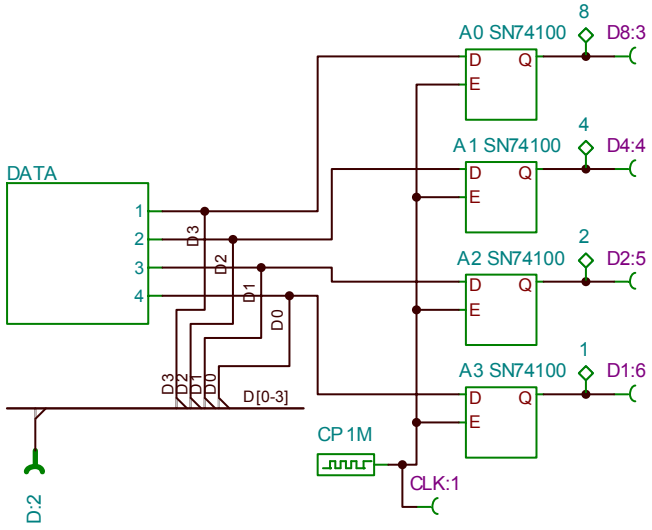
Δηλαδή το E δρά σα σήμα επίτρεψης (enable) που ελέγχει το φόρτωμα των πληροφοριών στον καταχωρητή. Όταν CP=1 οι πληροφορίες εισόδου φορτώνονται. Όταν CP=0, τότε το περιεχόμενο του καταχωρητή δεν αλλάζει.

Τα περισσότερα ψηφιακά συστήματα έχουν μια γεννήτρια για το κύριο ρολόι τους, η οποία τα τροφοδοτεί με μια συνεχή σειρά παλμών ρολογιού. Όλοι αυτοί οι παλμοί εφαρμόζονται σε όλα τα FFs και σε όλους τους καταχωρητές ολόκληρου του συστήματος. Αυτό το γενικό ρολόι είναι σα μια αντλία που δίνει ένα σταθερό ρυθμό σε όλα τα μέρη του συστήματος. Υπ' αυτές τις συνθήκες, μετά, υπάρχει ένα ξεχωριστό σήμα ελέγχου που κανονίζει ποιοι συγκεκριμένοι παλμοί του ρολογιού θα έχουν κάποια επίδραση σε ένα συγκεκριμένο καταχωρητή.

Σε ένα τέτοιο σύστημα πρέπει να περάσουμε πρώτα τους παλμούς του ρολογιού μέσα από μια πύλη AND με το σήμα ελέγχου και μετά την έξοδό της πύλης να την εφαρμόσουμε στον ακροδέκτη CP ή E του FF. Μια τέτοια μεταβλητή ελέγχου ονομάζεται είσοδος ελέγχου φόρτωσης (**load control**).

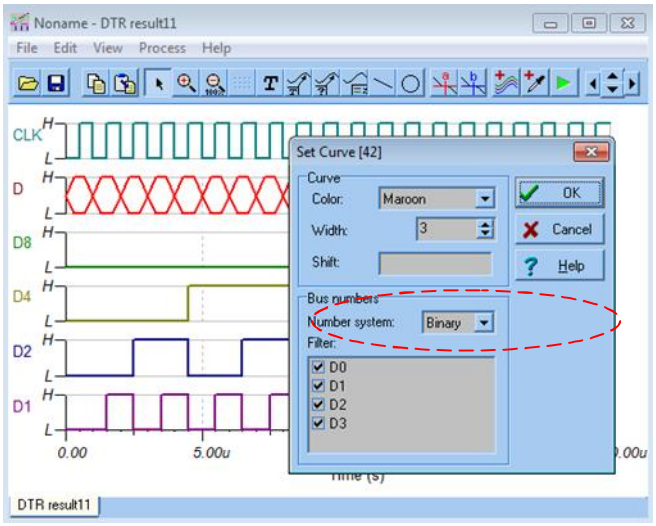
Το να βάλουμε μια πύλη AND στην πορεία του ρολογιού σημαίνει ότι αρχίζουμε να κάνουμε λογικές πράξεις με τους παλμούς του ρολογιού. Η παρεμβολή όμως λογικών πυλών δημιουργεί καθυστερήσεις ανάμεσα στο κύριο ρολόι και στις εισόδους ρολογιού των διαφόρων FFs. Για να συγχρονίσουμε πλήρως το σύστημα πρέπει να εξασφαλίσουμε ότι όλοι οι παλμοί του ρολογιού φτάνουν την ίδια στιγμή στις εισόδους όλων των FFs, έτσι ώστε όλα τους να αλλάζουν ταυτόχρονα. Οι λογικές πράξεις πάνω στους παλμούς του ρολογιού εισάγουν διάφορες καθυστερήσεις, οι οποίες μπορεί να ρίξουν το σύστημα εκτός συγχρονισμού.

Σχεδιάστε το επόμενο κύκλωμα και κάντε τις κατάλληλες ενέργειες ώστε να παραχθεί το αντίστοιχο διάγραμμα χρονισμού. Έχουμε κάνει χρήση του Bus (Από το μενού Insert) και του έχουμε δόσει το όνομα D[0-3]. Στη συνέχεια με απλό wire ενώνουμε τις εξόδους της γεννήτριας DATA πάνω στο Bus και ονομάζουμε κάθε wire ανάλογα με το απο που ξεκινά D0, D1, D2 και D3. Από την παλέτα Meters επιλέγουμε το Bus Display σε ρόλο Voltage Pin για το Bus (**Εικόνα 12**). Στο διάγραμμα χρονισμού μπορούμε να κάνουμε διπλό κλικ πάνω στη γραμμή του D και να επιλέγουμε στο Number System την επιλογή Binary (**Εικόνα 13**), ώστα να δούμε τα δεδομένα DATA του D σε δυαδική μορφή (**Εικόνα 14**).

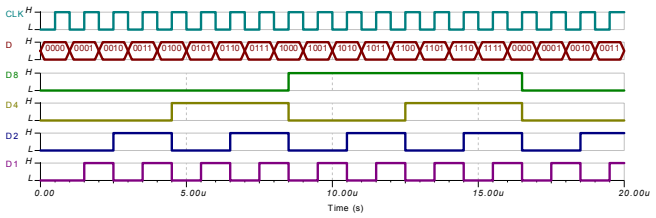


Εικόνα 11. Παράλληλος καταχωρητής 4bit με D-Flip-Flop.

Εικόνα 12. Bus Display.



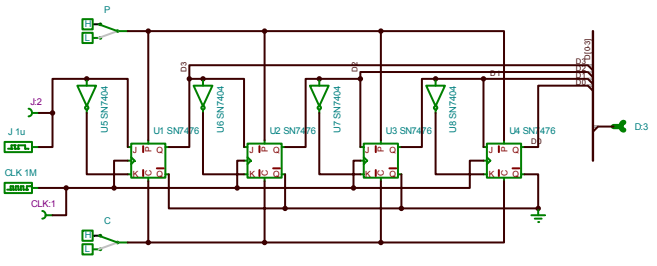
Εικόνα 13. Επιλογή Binary συστήματος παρουσίασης αποτελεσμάτων.



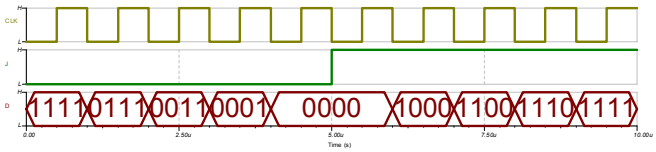
Εικόνα 14. Διάγραμμα χρονισμού με τα δεδομένα D σε binary μορφή.

6. Καταχωρητής ολίσθησης 4 – bit με JK Flip-Flop σε ρόλο D-Flip-Flop

Να σχεδιάσετε ένα κύκλωμα καταχωρητή 4 – bit με JK-Flip-Flop σε συνδεσμολογία D-τύπου Flip-Flop. Μελετήστε και προσομοιώστε τη λειτουργία του. (Βλέπε **Εικόνα 15** και **Εικόνα 16**)



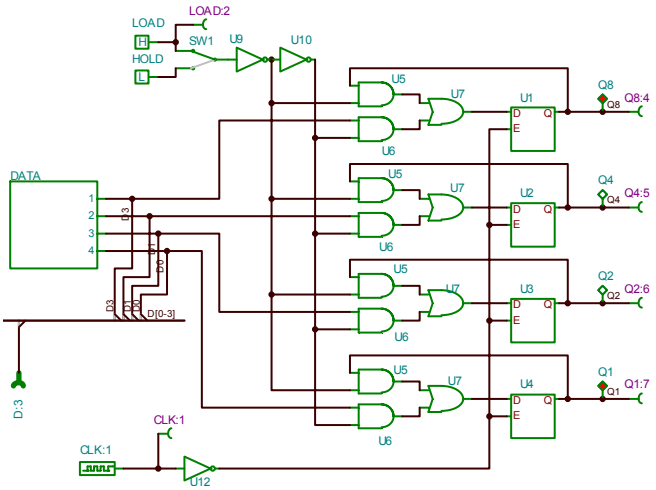
Εικόνα 15. Καταχωρητής ολίσθησης 4 – bit με JK FF σε ρόλο D-FF.



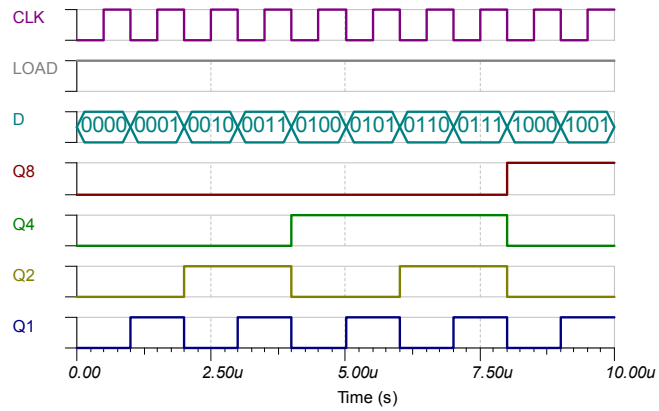
Εικόνα 16. Διάγραμμα χρονισμού καταχωρητή ολίσθησης 4bit.

7. Καταχωρητής παράλληλης φόρτωσης με 4bit, D Flip-Flops και σήμα LOAD και HOLD

Η σύνδεση αναδρασης σε κάθε D-FF είναι απαραίτητη γιατί τα FF τύπου D δεν έχουν συνθήκη εισόδου «μη αλλαγής» - με τον κάθε παλμό ρολογιού, η είσοδος D καθορίζει την επόμενη έξοδο. Για να μείνει η έξοδος αμετάβλητη πρέπει να κάνουμε την είσοδο D ίση με την παρούσα έξοδο Q για κάθε FF. (Βλέπε **Εικόνα 17** και **Εικόνα 18**).



Εικόνα 17. Καταχωρητής παράλληλης φόρτωσης με 4bit, D Flip-Flops και σήμα LOAD και HOLD.

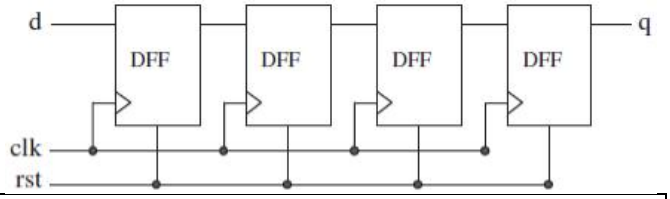


Εικόνα 18. Διάγραμμα χρονισμού καταχωρητή παράλληλης φόρτωσης 4bit.

Υλοποιήσεις με VHDL

Καταχωρητής ολίσθησης

Θα υλοποιήσουμε τον καταχωρητή ολίσθησης με το επόμενο μπλοκ διάγραμμα:

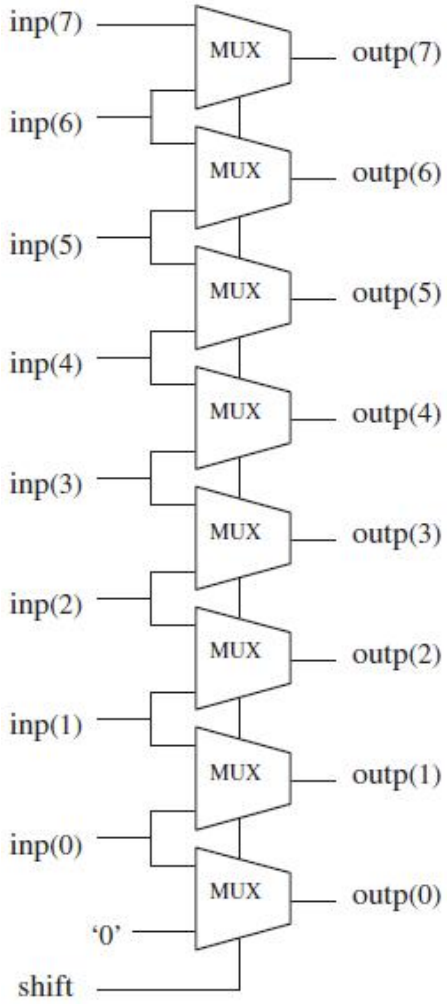



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftreg IS
GENERIC (n: INTEGER := 4);
-- # of stages
PORT (d, clk, rst: IN STD_LOGIC;
q: OUT STD_LOGIC);
END shiftreg;

ARCHITECTURE behavior OF shiftreg IS
SIGNAL internal:
STD_LOGIC_VECTOR (n-1 DOWNTO 0);
BEGIN
PROCESS (clk, rst)
BEGIN
IF (rst='1') THEN
internal <= (OTHERS => '0');
ELSIF (clk'EVENT AND clk='1') THEN
internal <=
d &internal(internal'LEFT DOWNTO 1);
END IF;
END PROCESS;
q <= internal(0);
END behavior;
```

Υλοποίηση απλού barrel shifter

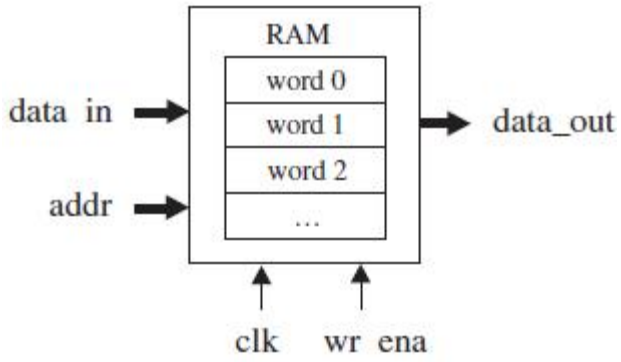
Στην επόμενη εικόνα φαίνεται το κύκλωμα του απλού ολισθητή τύπου βαρελιού. Το κύκλωμα θα πρέπει να ολισθήσει το διάνυσμα εισόδου (μήκους 8bit) είτε κατά 0 είτε κατά 1 θέση προς τα αριστερά. Όταν γίνει η ολίσθηση (shift=1), το LSB θα πρέπει να γεμίσει με 0 (όπως φαίνεται στην κάτω αριστερή γωνία του διαγράμματος). Αν shift=0, τότε outp=inp, αν shift=1, τότε outp(0)=0 και outp(i)=inp(i-1) για $1 \le i \le 7$.

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

ENTITY barrel IS
GENERIC (n: INTEGER := 8);
PORT ( inp: IN
STD_LOGIC_VECTOR (n-1 DOWNTO 0);
shift: IN INTEGER RANGE 0 TO 1;
outp: OUT
STD_LOGIC_VECTOR (n-1 DOWNTO 0));
END barrel;

ARCHITECTURE RTL OF barrel IS
BEGIN
PROCESS (inp, shift)
BEGIN
IF (shift=0) THEN
outp <= inp;
ELSE
outp(0) <= '0';
FOR i IN 1 TO inp'HIGH LOOP
outp(i) <= inp(i-1);
END LOOP;
END IF;
END PROCESS;
END RTL;
```



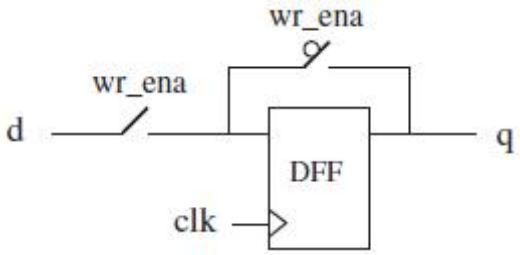
Υλοποίηση RAM με χρήση IF

Στη συνέχεια θα χρησιμοποιήσουμε την IF για να υλοποιήσουμε μια μνήμη τυχαίας προσπέλασης (RAM). Το κύκλωμα έχεις έναν δρομέα-δεδομένων (bus: data_in), έναν δρομέα-εξόδους (bus: data_out), έναν δρομέα-διευθύνσεων (bus: addr), σήμα ρολογιού (clk) και ενεργοποίηση-εγγραφής (wr_ana), όπως στο μπλοκ διάγραμμα του σχήματος:



Όταν το wr_ena είναι 1, στην επόμενη ακμή ανόδου του clk, το διάνυσμα που βρίσκεται στο data_in θα πρέπει να αποθηκευτεί στη θέση που προσδιορίζει το addr. Το διάνυσμα data_out, θα πρέπει πάντα να παρουσιάζει τα δεδομένα που επιλέγονται από τον addr.

Από τη σκοπιά των καταχωρητών, το κύκλωμα υλοποιείται ως εξής:



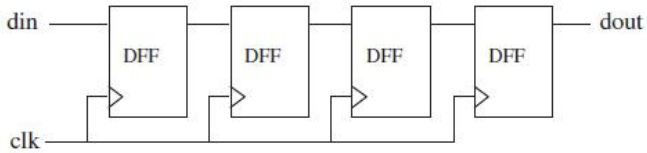
Όταν wr_ena είναι 0, το q συνδέεται στην είσοδο του flip-flop και το d είναι ανοικτό, οπότε δε θα γραφούν νέα δεδομένα στη μνήμη. Όταν το wr_ena είναι 1, το δ συνδέεται με την είσοδο του καταχωρητή, οπότε στην επόμενη ακμή ανόδου του clk, το d θα γράψει πάνω στην προηγούμενη τιμή.

Μια πιθανή υλοποίηση της λειτουργίας της RAM:

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

ENTITY ram IS
GENERIC ( bits: INTEGER := 8;
-- # of bits per word
words: INTEGER := 16);
-- # of words in the memory
PORT ( wr_ena, clk: IN STD_LOGIC;
addr: IN INTEGER RANGE 0 TO words-1;
data_in: IN STD_LOGIC_VECTOR (bits-1
DOWNTO 0);
data_out: OUT
STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
END ram;

ARCHITECTURE ram OF ram IS
TYPE vector_array IS
ARRAY (0 TO words-1) OF
STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
SIGNAL memory: vector_array;
BEGIN
PROCESS (clk, wr_ena)
BEGIN
IF (wr_ena='1') THEN
IF (clk'EVENT AND clk='1') THEN
memory(addr) <= data_in;
END IF;
END IF;
END PROCESS;
data_out <= memory(addr);
END ram;
```

Υλοποίηση καταχωρητή ολίσθησης για τη μελέτη των VARIABLE και SIGNAL

Θα εξετάσουμε τη συμπεριφορά λειτουργίας ενός καταχωρητή ολίσθησης:



όταν χρησιμοποιήσουμε δηλώσεις VARIABLE και SIGNAL. Η σωστή λύση θα πρέπει να δίνει ως σήμα εξόδου (dout) το σήμα εισόδου (din) καθυστερημένο κατά 4 θετικές ακμές ρολογιού.

Στην επόμενη υλοποίηση χρησιμοποιούμε τρεις VARIABLE (a, b, c). Οι μεταβλητές αυτές χρησιμοποιούνται πριν γίνει σε αυτές απόδοση τιμής (με την αντίστροφη σειρά, δηλαδή c, b, a). Συνεπώς ο μεταγλωττιστής θα θεωρήσει την ύπαρξη flip-flops που αποθηκεύουν τις τιμές από την προηγούμενη εκτέλεση της PROCESS.

```
-------- Solution 1:
ENTITY shift IS
PORT ( din, clk: IN BIT;
dout: OUT BIT);
END shift;

ARCHITECTURE shift OF shift IS
BEGIN
PROCESS (clk)
VARIABLE a, b, c: BIT;
BEGIN
IF (clk'EVENT AND clk='1') THEN
dout <= c;
c := b;
b := a;
a := din;
END IF;
END PROCESS;
END shift;
```

Στην επόμενη υλοποίηση, οι μεταβλητές αντικαθίστανται από SIGNAL και οι αποδόσεις τιμών γίνονται με κανονική σειρά. Επειδή οι αποδόσεις σημάτων στη μετάβαση ενός άλλου σήματος (clk), παράγουν καταχωρητές, και εδώ, θα προκύψει το σωστό κύκλωμα.

```
-------- Solution 2:
ENTITY shift IS
```

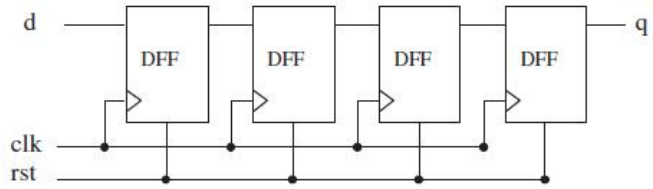
```
PORT ( din, clk: IN BIT;
dout: OUT BIT);
END shift;

ARCHITECTURE shift OF shift IS
SIGNAL a, b, c: BIT;
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
a <= din;
b <= a;
c <= b;
dout <= c;
END IF;
END PROCESS;
END shift;
```

Στην επόμενη υλοποίηση χρησιμοποιούμε την πρώτη λύση, αλλά οι αποδόσεις στις μεταβλητές γίνονται με κανονική σειρά. Ωστόσο, η απόδοση σε μεταβλητή είναι άμεση και επειδή οι μεταβλητές χρησιμοποιούνται σε κανονική σειρά (δηλαδή αφότου τους έχουν αποδοθεί τιμές), οι αποδόσεις τους τελικά αντιστοιχούν όλες στην c := din. Η τιμή του c φεύγει από τη διαδικασία όταν στην επόμενη γραμμή γίνεται απόδοση σήματος (dout <= c) στην μετάβαση του clk. Συνεπώς μόνο ένας καταχωρητής θα θεωρηθεί από το μεταγλωττιστή, οπότε δε θα προκύψει το σωστό κύκλωμα.

```
-------- Solution 3:
ENTITY shift IS
PORT ( din, clk: IN BIT;
dout: OUT BIT);
END shift;

ARCHITECTURE shift OF shift IS
BEGIN
PROCESS (clk)
VARIABLE a, b, c: BIT;
BEGIN
IF (clk'EVENT AND clk='1') THEN
a := din;
b := a;
c := b;
dout <= c;
END IF;
END PROCESS;
END shift;
```

Υλοποίηση καταχωρητή ολίσθησης με ασύγχρονης εισόδου reset

Θα υλοποιήσουμε τον επόμενο καταχωρητή ολίσθησης:



Το bit εξόδου (q) θα πρέπει να είναι καθυστερημένο κατά 4 θετικές ακμές ρολογιού σε σχέση με το σήμα εισόδου (d). Το reset θα είναι ασύγχρονο και θα θέτει όλες τις εξόδους των flip-flops στο 0.

Στην πρώτη υλοποίηση, χρησιμοποιούμε SIGNAL για να παράγουμε τα flip-flops, ενώ στη δεύτερη υλοποίηση χρησιμοποιούμε VARIABLE. Τα κυκλώματα που θα συντεθούν και στις δύο περιπτώσεις είναι τα ίδια (δηλαδή σε κάθε υλοποίηση ο μεταγλωττιστής θα υποθέσει 4 flip-flops).

Στη λύση 1, οι καταχωρητές προκύπτουν γιατί γίνεται απόδοση σε σήμα κατά τη μετάβαση άλλου σήματος.

```
-- Solution 1:
-- With an internal SIGNAL
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftreg IS
PORT ( d, clk, rst: IN STD_LOGIC;
q: OUT STD_LOGIC);
END shiftreg;

ARCHITECTURE behavior OF shiftreg IS
SIGNAL internal:
STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
PROCESS (clk, rst)
BEGIN
IF (rst='1') THEN
internal <= (OTHERS => '0');
ELSIF (clk'EVENT AND clk='1') THEN
internal <= d & internal(3 DOWNTO
1);
END IF;
END PROCESS;
q <= internal(0);
END behavior;
```

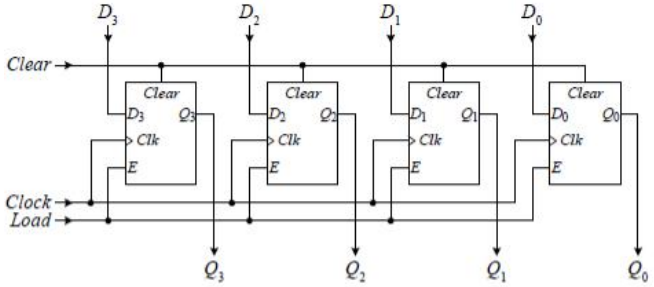
Στη λύση 2, γίνεται απόδοση τιμής σε μεταβλητή κατά τη μετάβαση ενός άλλου σήματος, αλλά επειδή η τιμή της μεταβλητής φεύγει από τη διαδικασία (η τιμή περνάει σε μια θύρα) και πάλι γίνεται η υπόθεση ύπαρξης καταχωρητών.

```
-- Solution 2:
-- With an internal VARIABLE
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shiftreg IS
PORT ( d, clk, rst: IN STD_LOGIC;
```

```
q: OUT STD_LOGIC);
END shiftreg;

ARCHITECTURE behavior OF shiftreg IS
BEGIN
PROCESS (clk, rst)
VARIABLE internal:
STD_LOGIC_VECTOR (3 DOWNTO 0);
BEGIN
IF (rst='1') THEN
internal := (OTHERS => '0');
ELSIF (clk'EVENT AND clk='1') THEN
internal := d & internal(3 DOWNTO
1);
END IF;
q <= internal(0);
END PROCESS;
END behavior;
```

Υλοποίηση 4bit καταχωρητή με παράλληλο φορτίο και ασύγχρονη είσοδο εκκαθάρισης

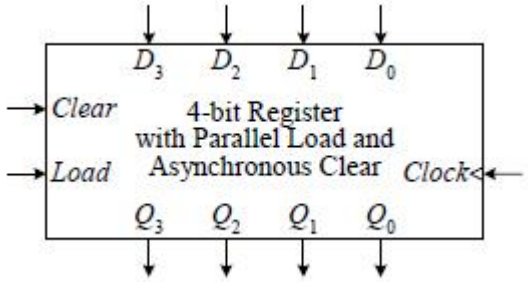
Η δομή του καταχωρητή που θα υλοποιήσουμε φαίνεται στην επόμενη εικόνα:



Η λειτουργία του συνοψίζεται στον επόμενο πίνακα:

Clear	Load	Λειτουργία
1	X	Θέτει του καταχωρητές σε τιμή μηδέν (ασύγχρονα)
0	0	Καμία αλλαγή
0	1	Φορτώνεται η τιμή στην ακμή ανόδου του ρολογιού

Σε μορφή μπλοκ διαγράμματος, ο παραπάνω καταχωρητής φαίνεται στη συνέχεια:

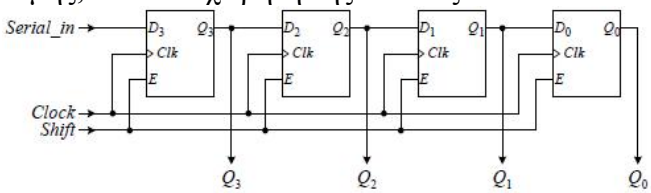


Στη συνέχεια παρουσιάζουμε τον VHDL κώδικα για τον προηγούμενο καταχωρητή:

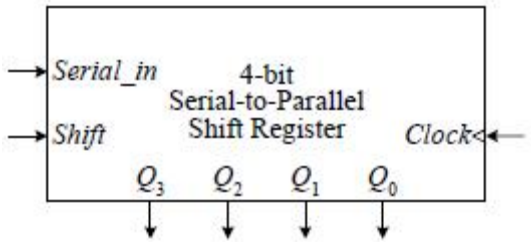
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY reg IS
GENERIC (size: INTEGER := 3);
-- size of the register
PORT (
Clock, Clear, Load: IN STD_LOGIC;
D: IN
STD_LOGIC_VECTOR(size DOWNTO 0);
Q: OUT
STD_LOGIC_VECTOR(size DOWNTO 0));
END reg;

ARCHITECTURE Behavior OF reg IS
BEGIN
PROCESS(Clock, Clear)
BEGIN
IF Clear = '1' THEN
Q <= (OTHERS => '0');
ELSIF (Clock'EVENT AND Clock = '1')
THEN
IF Load = '1' THEN
Q <= D;
END IF;
END IF;
END PROCESS;
END Behavior;
```

Υλοποίηση 4bit σειριακού-σε-παράλληλο καταχωρητή ολίσθησης με αρχιτεκτονική δομής

Θα υλοποιήσουμε με αρχιτεκτονική δομής, τον καταχωρητή της εικόνας:



Το ενιαίο μπλοκ διάγραμμά του έχεις ως εξής:



Η λειτουργία του συνοψίζεται στον επόμενο πίνακα:

Shift	Λειτουργία
0	Καμία αλλαγή
1	Ένα bit από το Serial_in ολισθαίνει προς τα μέσα

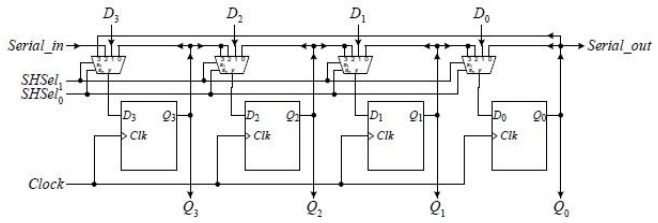
Στη συνέχεια παρουσιάζουμε τον κώδικα υλοποίησης τους καταχωρητή χρησιμοποιώντας αρχιτεκτονική δομικής περιγραφής:

```
-- D flip-flop with enable
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY D_flipflop IS
PORT(D, Clock, E : IN STD_LOGIC;
```

```
Q : OUT STD_LOGIC);
END D_flipflop;

ARCHITECTURE Behavior OF D_flipflop
IS
BEGIN
PROCESS(Clock)
BEGIN
IF (Clock'EVENT AND Clock = '1')
THEN
IF (E = '1') THEN
Q <= D;
END IF;
END IF;
END PROCESS;
END Behavior;

-- 4-bit shift register
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY ShiftReg IS
PORT(Serial_in, Clock, Shift : IN
STD_LOGIC;
Q : OUT
STD_LOGIC_VECTOR(3 downto 0));
END ShiftReg;

ARCHITECTURE Structural OF ShiftReg
IS
SIGNAL N0, N1, N2, N3 : STD_LOGIC;
COMPONENT D_flipflop
PORT (D, Clock, E : IN STD_LOGIC;
Q : OUT STD_LOGIC);
END COMPONENT;
BEGIN
U1: D_flipflop PORT MAP
(Serial_in, Clock, Shift, N3);
U2: D_flipflop PORT MAP
(N3, Clock, Shift, N2);
U3: D_flipflop PORT MAP
(N2, Clock, Shift, N1);
U4: D_flipflop PORT MAP
(N1, Clock, Shift, N0);
Q(3) <= N3;
Q(2) <= N2;
Q(1) <= N1;
Q(0) <= N0;
END Structural;
```

Υλοποίηση σειριακού-σε-παράλληλο και παράλληλο-σε-σειριακό καταχωρητή ολίσθησης

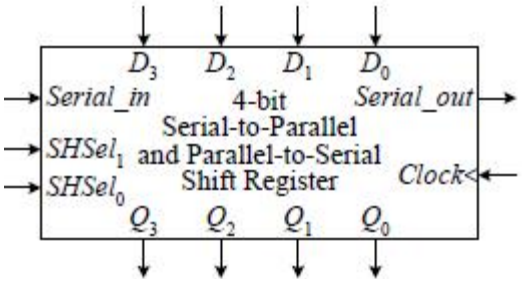
Θα υλοποιήσουμε το επόμενο κύκλωμα:



Η λειτουργία του συνοψίζεται στον επόμενο πίνακα:

SHSel1	SHSel0	Λειτουργία
0	0	Καμία λειτουργία (διατήρηση της τρέχουσας τιμής)
0	1	Παράλληλη φόρτωση της νέας τιμής
1	0	Ολίσθηση δεξιά
1	1	Ολίσθηση αριστερά

Το μπλοκ διάγραμμα όλου του κυκλώματος έχει ως εξής:



Ο κώδικας VHDL για το κύκλωμα φαίνεται στη συνέχεια:

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

ENTITY shiftreg IS PORT (
Clock: IN STD_LOGIC;
SHSel: IN
STD_LOGIC_VECTOR(1 DOWNTO 0);
Serial_in: IN STD_LOGIC;
D: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
Serial_out: OUT STD_LOGIC;
Q: OUT STD_LOGIC_VECTOR(3 DOWNTO
0));
END shiftreg;

ARCHITECTURE Behavioral OF shiftreg
IS
SIGNAL content:
STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
PROCESS(Clock)
BEGIN
IF (Clock'EVENT AND Clock='1') THEN
CASE SHSel IS
WHEN "01" =>
-- load
content <= D;
WHEN "10" =>
-- shift right, pad with bit from
Serial_in
content <= Serial_in & content(3
```

```
DOWNTO 1);
WHEN OTHERS =>
NULL;
END CASE;
END IF;
END PROCESS;
Q <= content;
Serial_out <= content(0);
END Behavioral;
```