Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

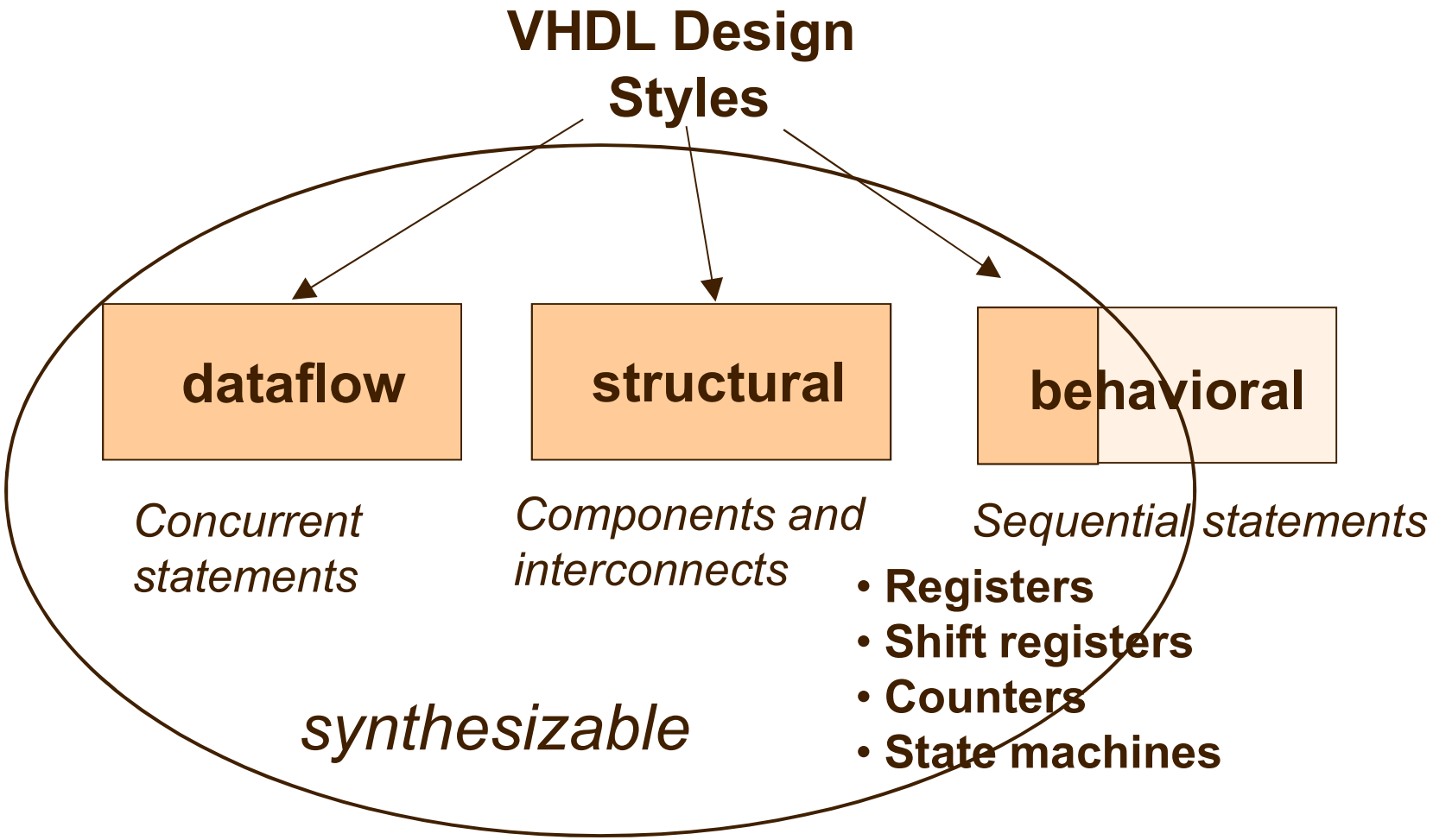
Πανεπιστήμιο Δυτικής Αττικής

Ακαδημαϊκό έτος 2017-18

Ενσωματωμένα Συστήματα

Ακολουθιακά μπλοκ σε VHDL

VHDL Design Styles



Processes στη VHDL

- Περιγράφουν ακολουθιακή συμπεριφορά
- Επιτρέπουν τον ορισμό συμπεριφοράς που αναπαρίσταται δύσκολα από πραγματικό κύκλωμα
- Δεν είναι πάντα συνθέσιμες
- Χρησιμοποιούνται **με προσοχή** σε κώδικα στον οποίο θα γίνει σύνθεση

PROCESS με SENSITIVITY LIST

- Λίστα από signals στα οποία είναι sensitive η process.
- Όταν πραγματοποιείται ένα event σε οποιοδήποτε από αυτά η process ενεργοποιείται.
- Κάθε φορά που ενεργοποιείται, τρέχει ολόκληρη.
- **ΔΕΝ ΕΠΙΤΡΕΠΟΝΤΑΙ WAIT statements σε process με SENSITIVITY LIST.**

label: process (*sensitivity list*)
    declaration part
begin
    statement part
end process;

Παραδείγματα events:

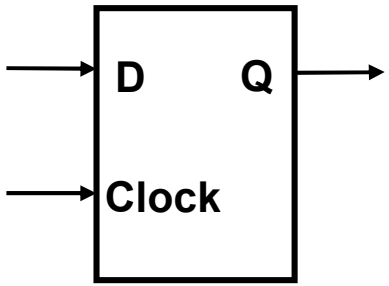
```
IF Clock'EVENT
IF rising_edge(Clock)
```

Flip flops & Registers

D Latch
D flip flop
 Asynchronous reset
 Synchronous reset
Registers
Generic Registers

D latch

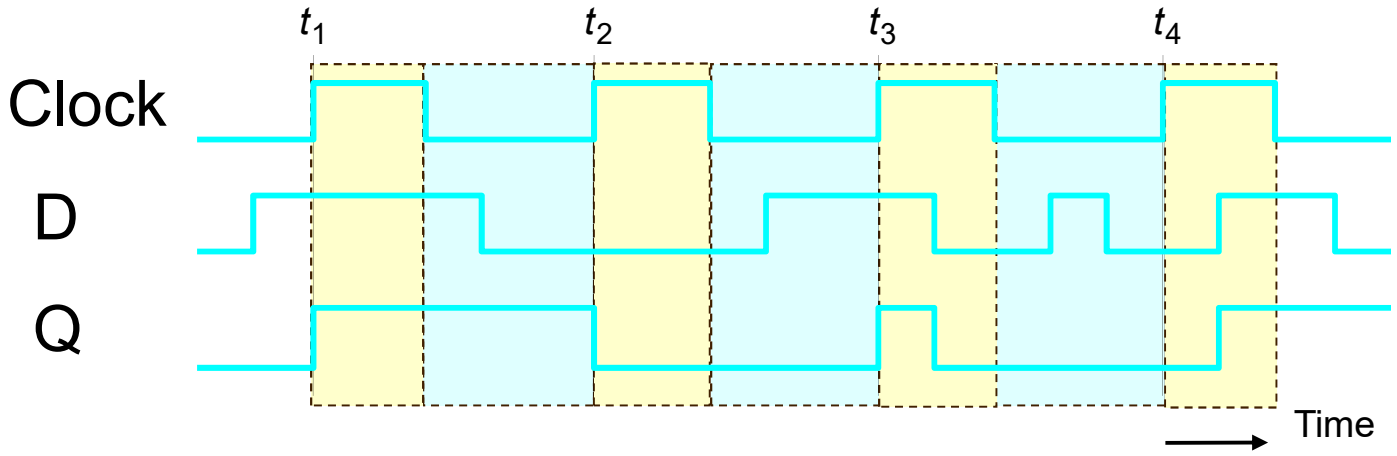
Graphical symbol



Truth table

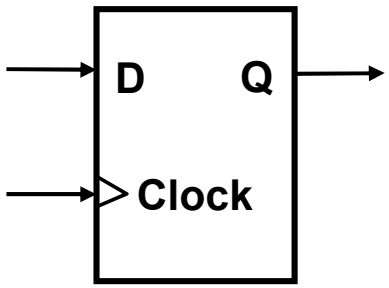
Clock	D	$Q(t+1)$
0	–	$Q(t)$
1	0	0
1	1	1

Timing diagram



D flip-flop

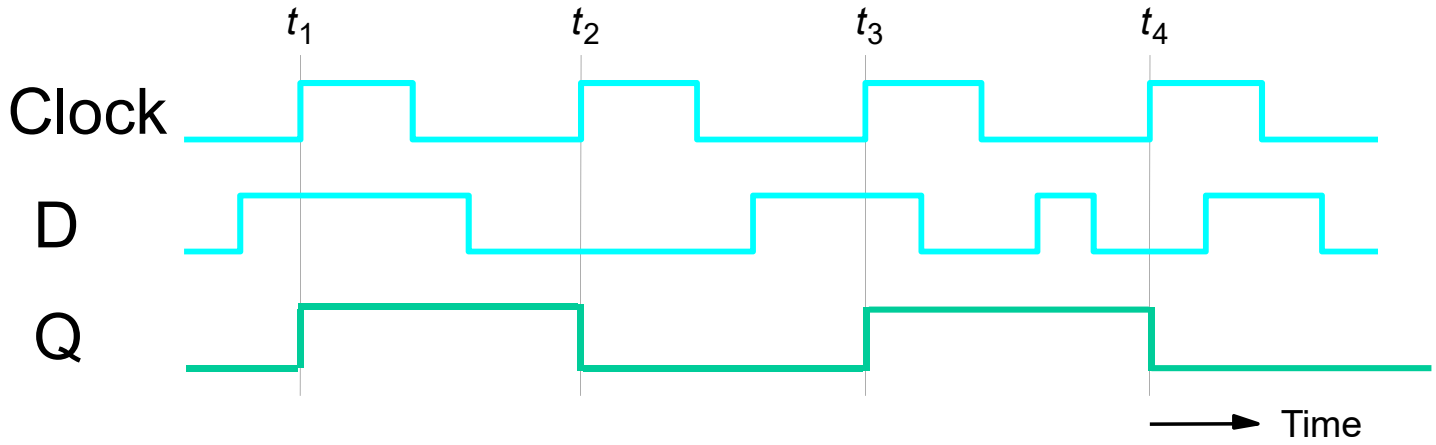
Graphical symbol



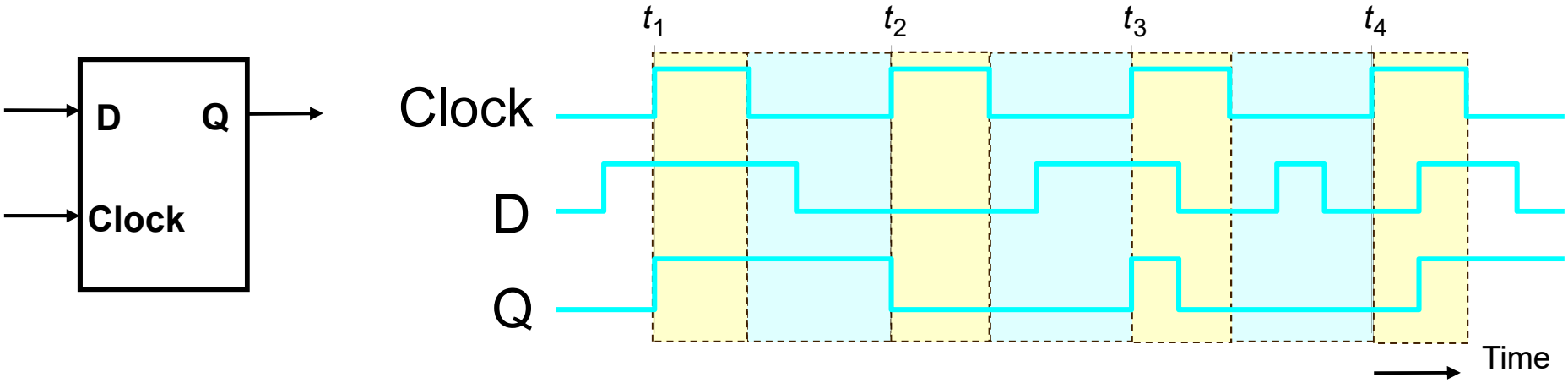
Truth table

Clk	D	Q(*t*+1)
↑	0	0
↑	1	1
0	–	Q(*t*)
1	–	Q(*t*)

Timing diagram



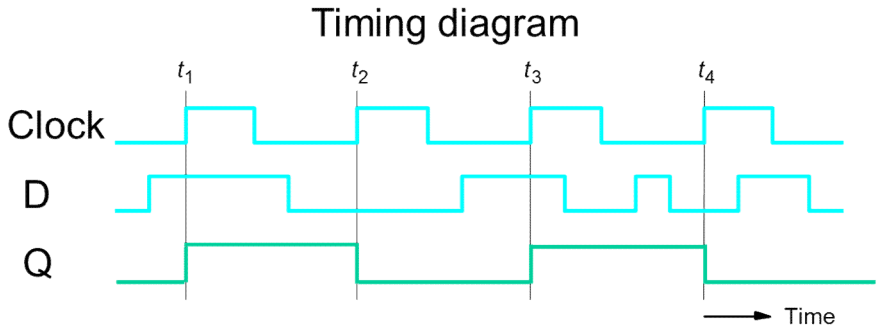
D latch

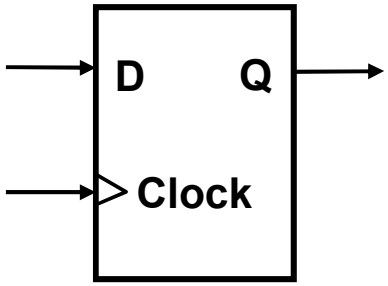



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY latch IS PORT (
 D, Clock : IN  STD_LOGIC;
 Q        : OUT STD_LOGIC);
END latch;
ARCHITECTURE behavioral OF latch IS
BEGIN
    PROCESS (D, Clock)
    BEGIN
        IF Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END behavioral;
```

D flip-flop

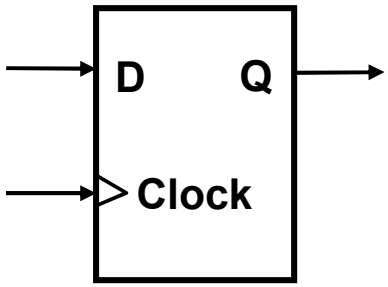
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
 PORT (D, Clock    : IN  STD_LOGIC;
       Q           : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE behavioral OF flipflop IS
BEGIN
    PROCESS (Clock)
    BEGIN
       IF Clock'EVENT AND Clock = '1' THEN
    -- IF rising_edge(Clock) THEN
            Q <= D ;
       END IF ;
    END PROCESS ;
END behavioral ;
```





D flip-flop

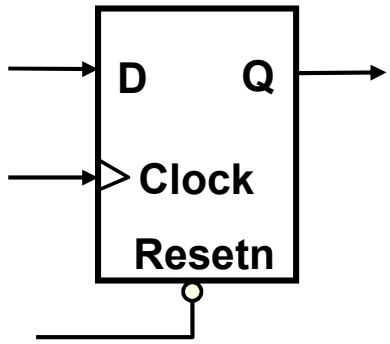
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
PORT (D, Clock : IN STD_LOGIC ;
      Q        : OUT STD_LOGIC) ;
END flipflop ;

ARCHITECTURE behavioral3 OF flipflop IS
BEGIN
    PROCESS
    BEGIN
       WAIT UNTIL rising_edge(Clock) ;
       Q <= D ;
    END PROCESS ;
END behavioral3 ;
```



D flip-flop & reset

Το σήμα reset μηδενίζει την έξοδο του flip flop
Υπάρχουν δύο είδη reset

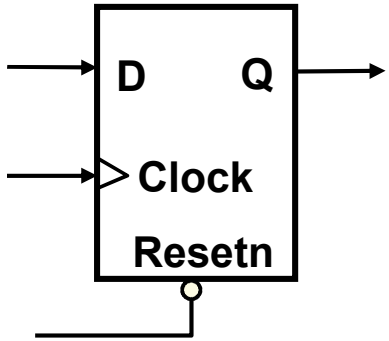
- Σύγχρονο
- Ασύγχρονο



Asychronous vs. Synchronous

Στο IF:

- Asynchronous items:
  - **Before** the rising_edge(Clock) statement
- Synchronous items:
  - **After** the rising_edge(Clock) statement

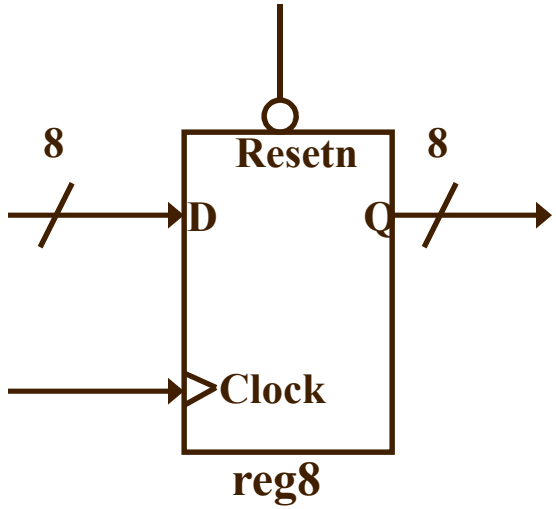



```
ARCHITECTURE behav OF ff_ar IS
BEGIN
 PROCESS (Resetn, Clock)
  BEGIN
   IF Resetn = '0' THEN
    Q <= '0';
   ELSIF rising_edge(Clock) THEN
    Q <= D;
   END IF;
  END PROCESS;
END behavioral;
```

```
ARCHITECTURE behav OF ff_sr IS
BEGIN
 PROCESS(Clock)
 BEGIN
  IF rising_edge(Clock) THEN
   IF Resetn = '0' THEN
    Q <= '0';
   ELSE
    Q <= D;
   END IF;
  END IF;
 END PROCESS;
END behavioral;
```

8-bit register with asynchronous reset

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

ENTITY reg8 IS PORT (
 D              : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
 Resetn, Clock  : IN  STD_LOGIC;
 Q              : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END reg8;

ARCHITECTURE behavioral OF reg8 IS
BEGIN
PROCESS (Resetn, Clock)
        BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000";
        ELSIF rising_edge(Clock) THEN
            Q <= D;
        END IF;
    END PROCESS;
END behavioral;
```



Generics

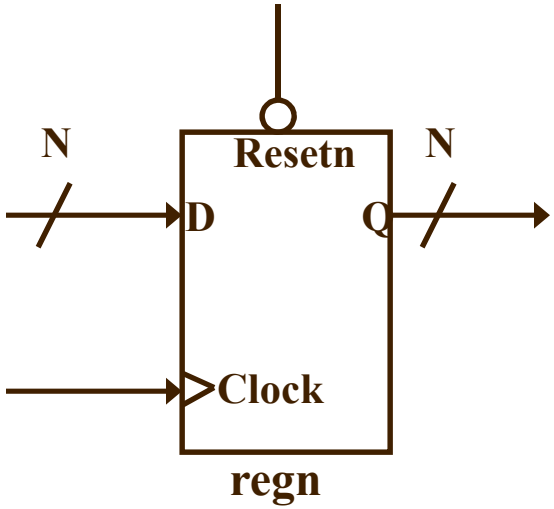
- Είναι **integer** τιμές
- Παίρνουν μια default τιμή

GENERIC (N : INTEGER := 16) ;

- Η τιμή αυτή μπορεί να γίνει overwritten όταν η entity χρησιμοποιείται σαν component
- Είναι χρήσιμα σε συχνά χρησιμοποιούμενα component
  - Ένας 32-bit register σε ένα σημείο και ένας 16-bit register σε ένα άλλο
  - Μπορούμε να χρησιμοποιήσουμε τον ίδιο κώδικα, με διαφορετική διαμόρφωση

N-bit register with reset

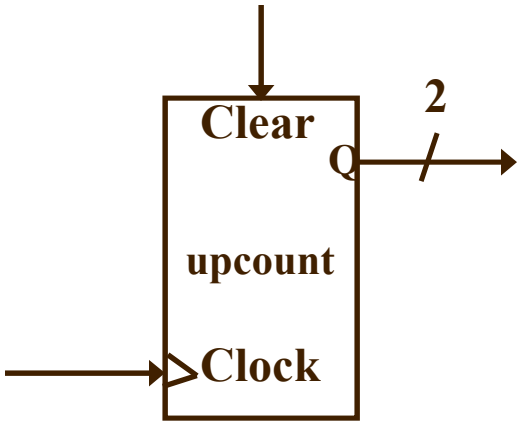
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
 GENERIC (N: INTEGER := 16);
 PORT(D         : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
 Resetn, Clock : IN STD_LOGIC ;
 Q             : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
END regn;

ARCHITECTURE behavioral OF regn IS
BEGIN
    PROCESS (Resetn, Clock)
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF rising_edge(Clock) THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END behavioral ;
```



Q: Τι reset έχουμε, σύγχρονο ή ασύγχρονο;

2-bit up-counter with clear

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount IS PORT (
 Clear, Clock: IN  STD_LOGIC;
 Q           : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)) ;
END upcount;

ARCHITECTURE behavioral OF upcount IS
 SIGNAL Count: std_logic_vector(1 DOWNTO 0);
 BEGIN
  upcount: PROCESS (Clock)
   BEGIN
    IF rising_edge(Clock) THEN
     IF Clear = '1' THEN
      Count <= "00" ;
     ELSE
      Count <= Count + 1 ;
     END IF ;
    END IF;
   END PROCESS;
  Q <= Count;
 END behavioral;
```



Q1: Τι clear έχουμε, σύγχρονο ή ασύγχρονο;

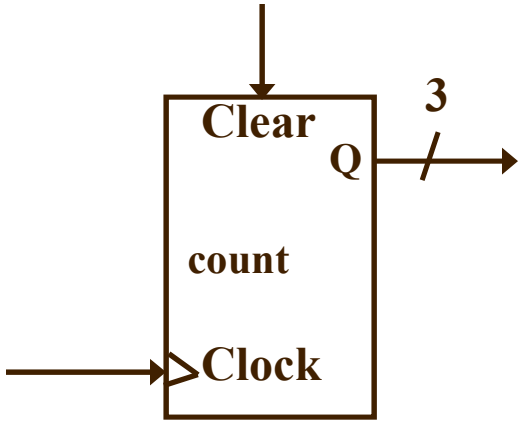
Q2: Ποια ακολουθία εμφανίζεται στην έξοδο;

Q3: Τι πρέπει να κάνουμε για να ξεκινήσει να λειτουργεί;

Q4: Τι θα έπρεπε να αλλάξουμε στον κώδικα αν θέλαμε να μετράει ανάποδα;

3-bit up counter with step = 3

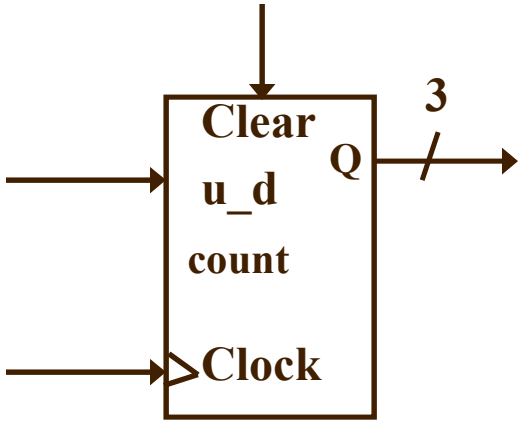
Θέλουμε να υλοποιήσουμε κύκλωμα που να μετράει ως εξής:
000, 011, 110, 001, 100, 111, 010, 101, 000…




```
ENTITY count IS PORT (
 Clear, Clock: IN  STD_LOGIC;
 Q : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)) ;
END upcount;

ARCHITECTURE behavioral OF upcount IS
 SIGNAL Count: std_logic_vector(2 DOWNTO 0);
 BEGIN
  PROCESS (Clock)
   BEGIN
    IF rising_edge(Clock) THEN
     IF Clear = '1' THEN
      Count <= "000" ;
     ELSE
      Count <= Count + 3 ;
     END IF ;
    END IF;
   END PROCESS;
  Q <= Count;
 END behavioral;
```

3-bit up/down counter with clear

Θέλουμε να υλοποιήσουμε κύκλωμα που μπορεί να μετράει και προς τα πάνω και προς τα κάτω. Θα έχουμε ένα επιπλέον σήμα up_down




```
ENTITY count IS PORT (
 Clear, Clock, U_D: IN STD_LOGIC;
 Q : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END count;

ARCHITECTURE behavioral OF count IS
 SIGNAL Count: std_logic_vector(2 DOWNTO 0);
 BEGIN
  PROCESS (Clock)
   BEGIN
    IF rising_edge(Clock) THEN
     IF Clear = '1' THEN
      Count <= "000";
     ELSIF U_D = '1' THEN
      Count <= Count + 1;
     ELSE
      Count <= Count - 1;
     END IF ;
    END IF;
   END PROCESS;
  Q <= Count;
 END behavioral;
```

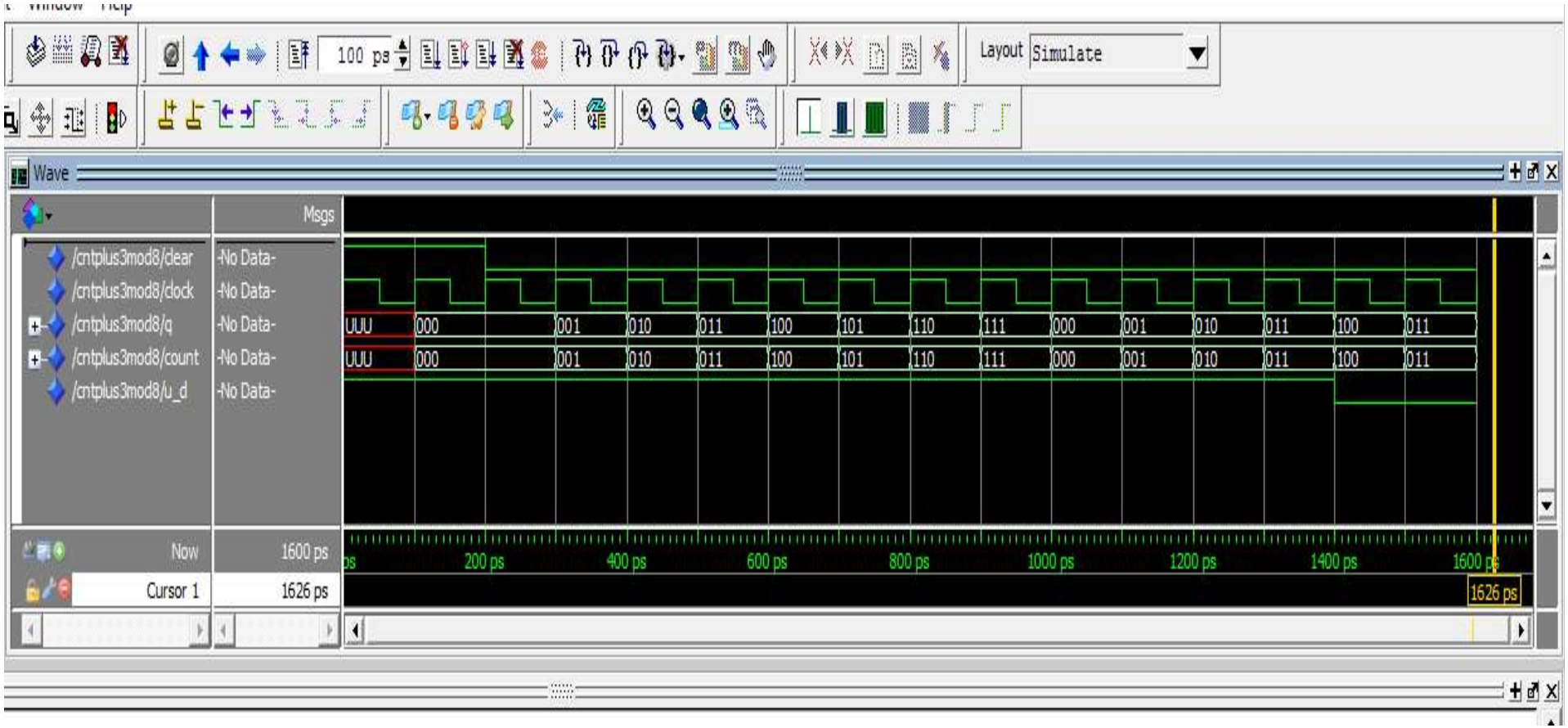
3-bit up/down counter with clear

```
PROCESS (Clock)
   BEGIN
    IF rising_edge(Clock) THEN
     IF Clear = '1' THEN Count <= "000";
     ELSIF U_D = '1' THEN Count <= Count + 1;
     ELSE Count <= Count - 1;
     END IF;
    END IF;
   END PROCESS;
  Q <= Count;
```

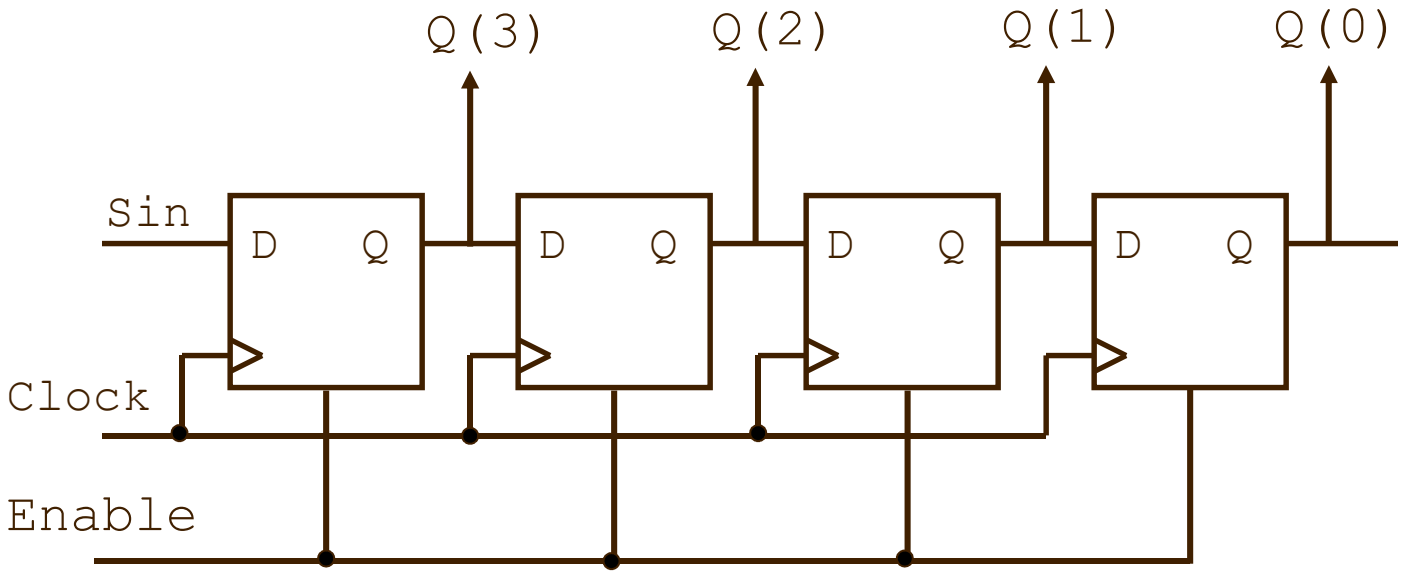
Q1: Γιατί στα πρώτα 100 ps το σήμα εξόδου είναι κόκκινο;

Q2: Γιατί η έξοδος μένει για δύο κύκλους στο 0;

Q3: Ρίχνω το σήμα u_d στο 1400. Γιατί η έξοδος αυξάνει για ένα κύκλο και μετά αρχίζει να μειώνεται;



Shift register

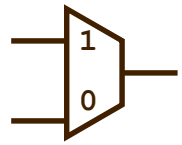


Q1: Πόσοι κύκλοι χρειάζονται για να φορτωθούν τα flip flop;
Q2: Τι τιμή πρέπει να έχει το enable;
Q3: Θεωρήστε τις ακόλουθες τιμές για τις εισόδους Sin, Enable.

```
Sin: 11010101010
En : 11101011010
```

Ποιες οι έξοδοι των flip flop για αυτούς τους 11 κύκλους
Θεωρήστε ότι τα flip flop έχουν αρχικά την τιμή 0.

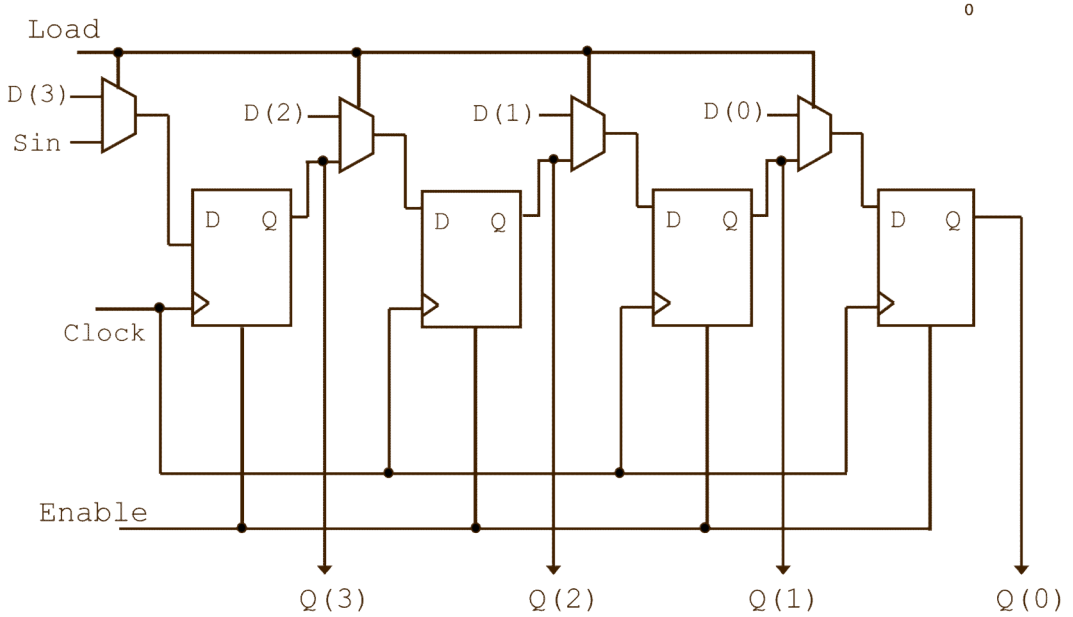
Shift Register With Parallel Load



Q: Θεωρήστε ότι

- αρχικά οι τιμές των flip flop είναι 0 και ότι
- οι είσοδοι D(3:0) έχουν όλες σταθερά την τιμή 1
- Θεωρήστε ακόμη τις ακόλουθες τιμές για τα Sin, Enable, Load.

```
Sin: 11010101010
En : 11101011010
Ld : 10110101011
```

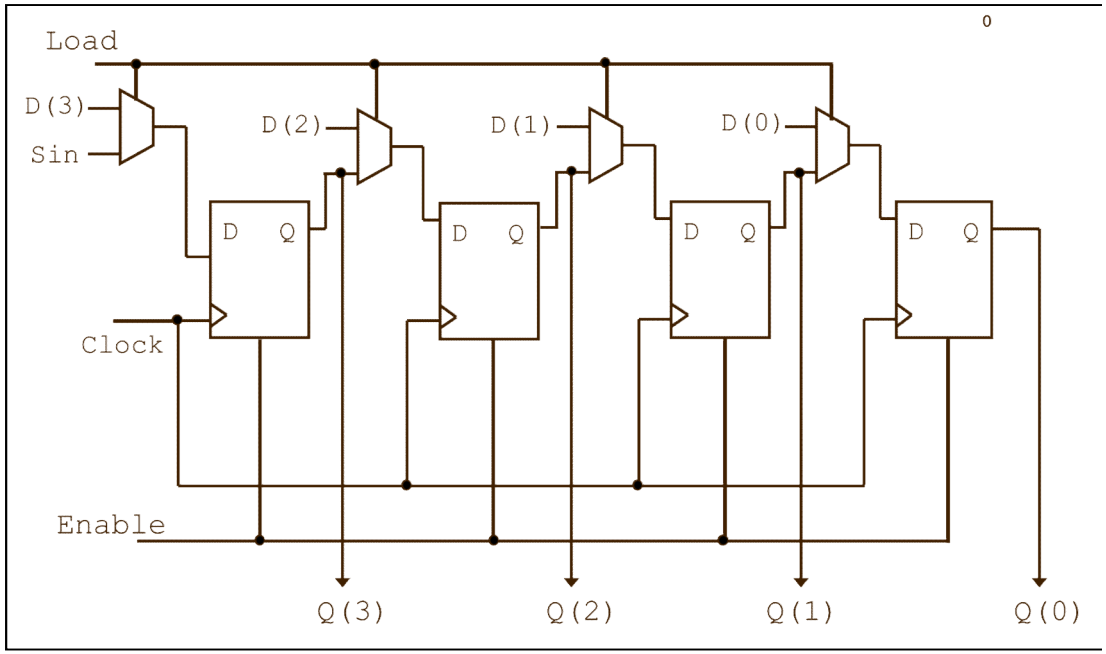


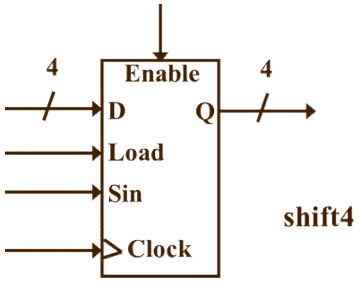
Ποιες οι έξοδοι των flip flop για αυτούς τους 11 κύκλους

4-bit shift register with parallel load (1)

Είναι όλα καλά;

```
ARCHITECTURE behavioral OF shift4 IS
 SIGNAL Qt : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
PROCESS (Clock)
 BEGIN
  IF rising_edge(Clock) THEN
   IF Load = '1' THEN
    Qt <= D ;
   ELSIF Enable = '1' THEN
    Qt(0) <= Qt(1);
    Qt(1) <= Qt(2);
    Qt(2) <= Qt(3);
    Qt(3) <= Sin;
   END IF ;
  END IF ;
 END PROCESS;
 Q <= Qt;
END behavioral;
```

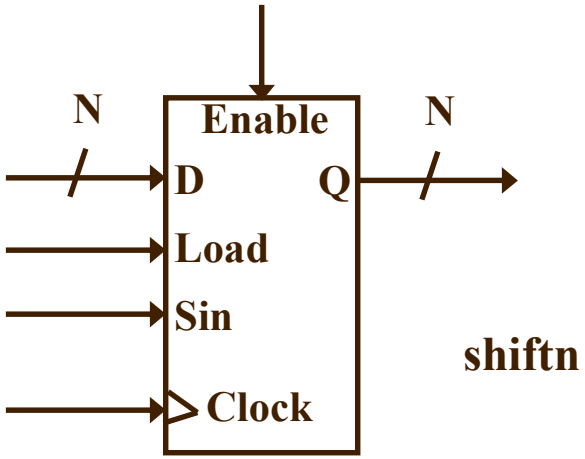





```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS PORT (
 D:      IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
 Enable: IN  STD_LOGIC;
 Load:   IN  STD_LOGIC;
 Sin:    IN  STD_LOGIC;
 Clock:  IN  STD_LOGIC;
 Q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END shift4;
```

N-bit shift register with parallel load (1)

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

ENTITY shiftn IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (  D : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Enable  : IN    STD_LOGIC ;
                Load    : IN    STD_LOGIC ;
                Sin     : IN    STD_LOGIC ;
                Clock   : IN    STD_LOGIC ;
                Q       : OUT   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftn ;
```



N-bit shift register with parallel load (2)

```
ARCHITECTURE behavioral OF shiftn IS
        SIGNAL Qt: STD_LOGIC_VECTOR(N-1 DOWNTO 0);
BEGIN
PROCESS (Clock)
 BEGIN
  IF rising_edge(Clock) THEN
   IF Load = '1' THEN
    Qt <= D ;
   ELSIF Enable = '1' THEN
    Genbits: FOR i IN 0 TO N-2 LOOP
     Qt(i) <= Qt(i+1) ;
    END LOOP ;
    Qt(N-1) <= Sin ;
   END IF;
  END IF ;
 END PROCESS ;
 Q <= Qt;
END behavioral;
```