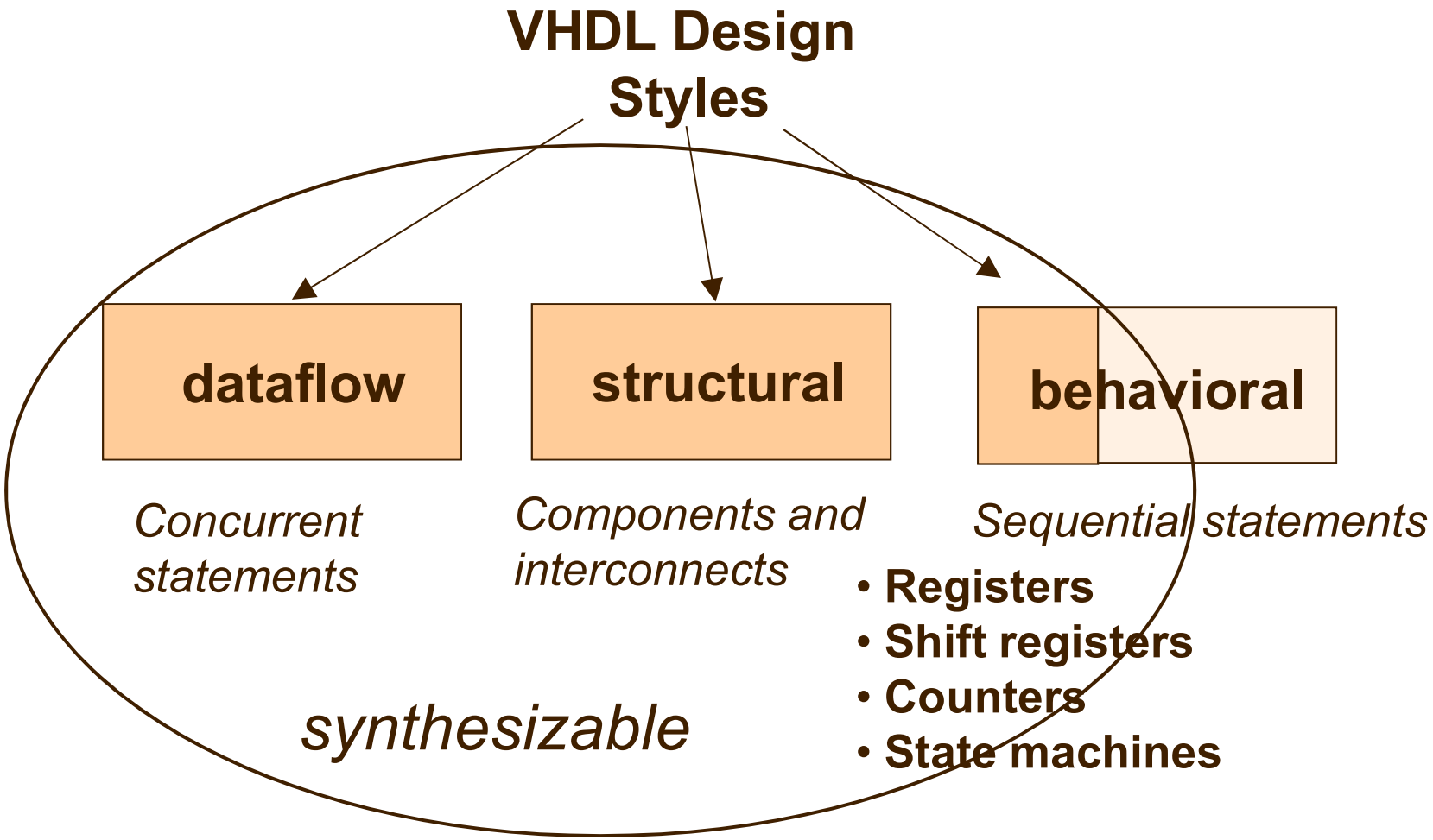
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Ακαδημαϊκό έτος 2018-19

Διάχυτα και 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

Ανατομία μιας Process

OPTIONAL

[label:] PROCESS [(*sensitivity list*)]
    [d*eclaration part*]
BEGIN
    statement part
END PROCESS [label];

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.

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

Παραδείγματα events:

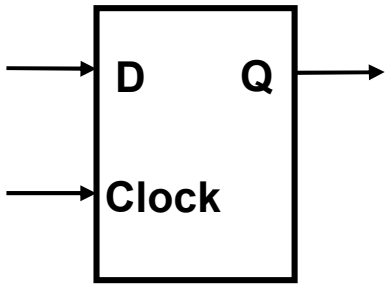
```
IF Clock'EVENT
```

Flip flops & Registers

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

D latch

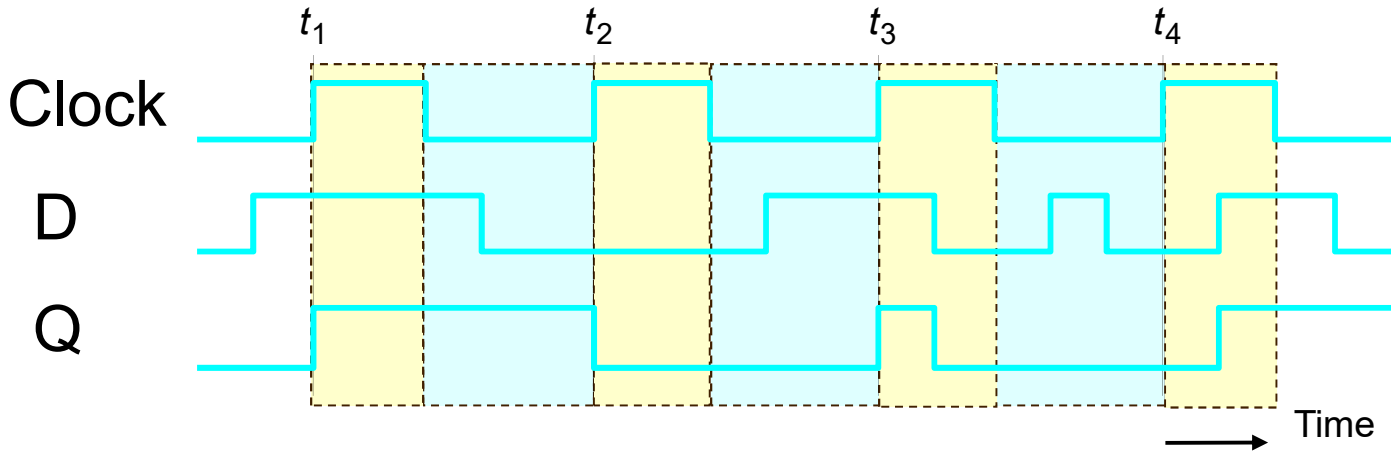
Graphical symbol



Truth table

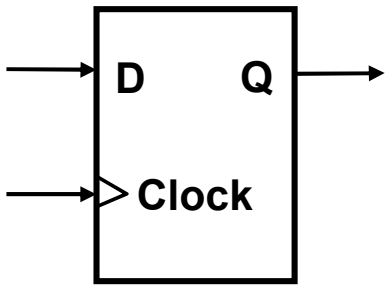
Clock	D	$Q(t+1)$
0	–	$Q(t)$
1	0	0
1	1	1

Timing diagram



D flip-flop

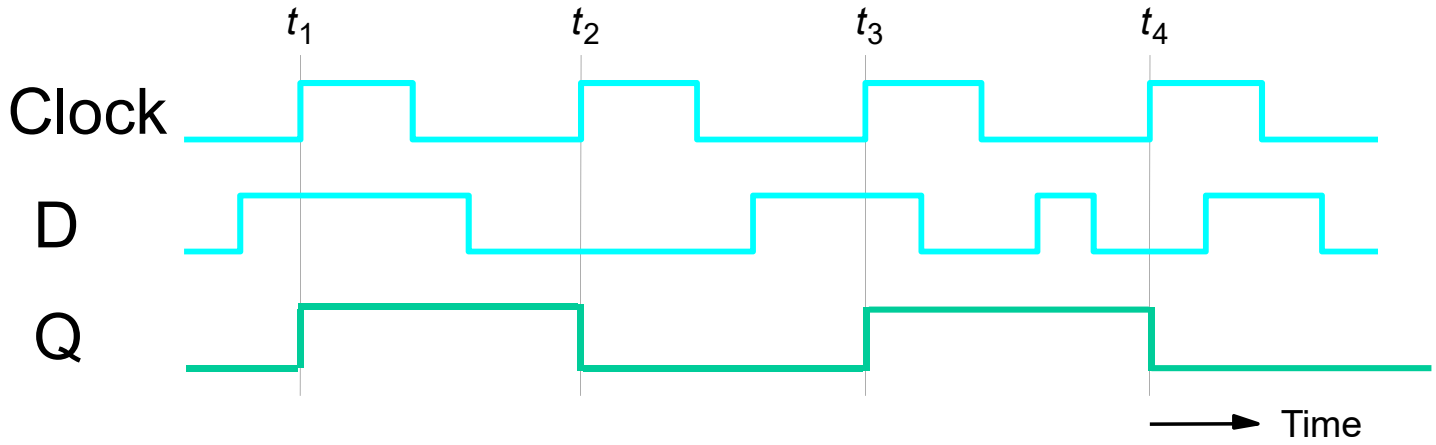
Graphical symbol



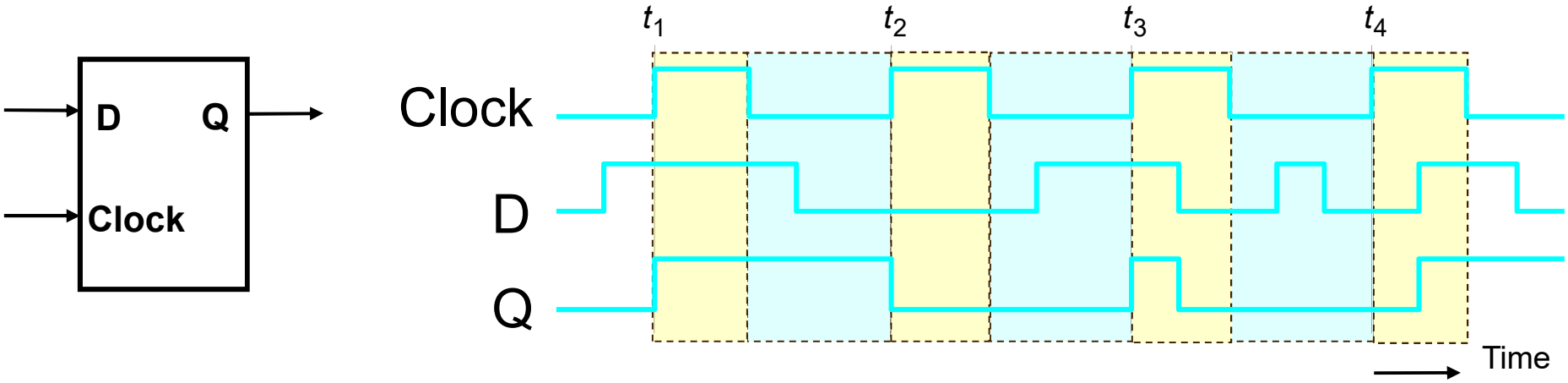
Truth table

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

Timing diagram



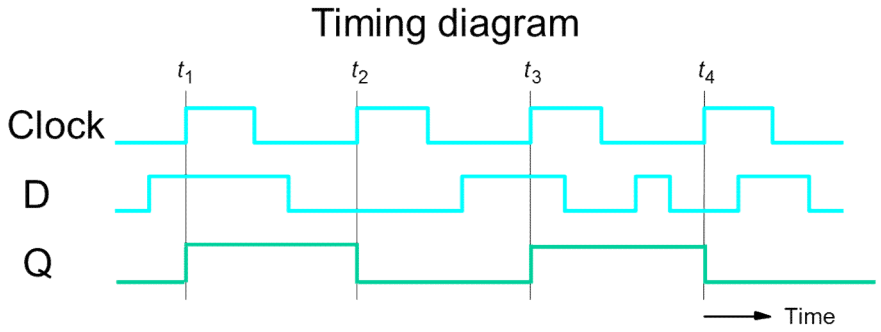
D latch

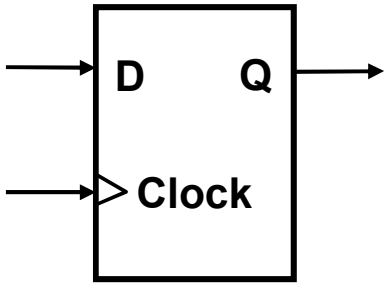



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY latch IS PORT (
 D, Clock : IN  STD_LOGIC;
 Q        : OUT STD_LOGIC);
END latch;
ARCHITECTURE behavioral OF latch IS
BEGIN
    PROCESS (D, Clock)
    BEGIN
        IF Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END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

D flip-flop

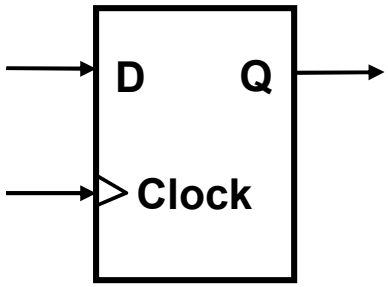
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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

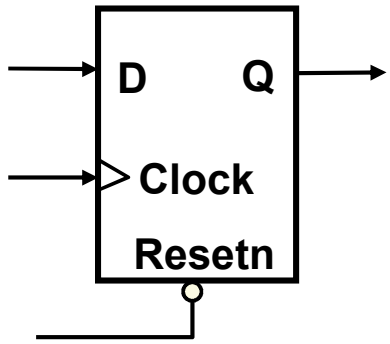
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

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

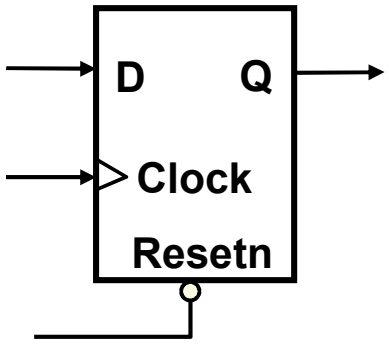
Το σήμα reset μηδενίζει την έξοδο του flip flop
Υπάρχουν δύο είδη reset

- Σύγχρονο
- Ασύγχρονο



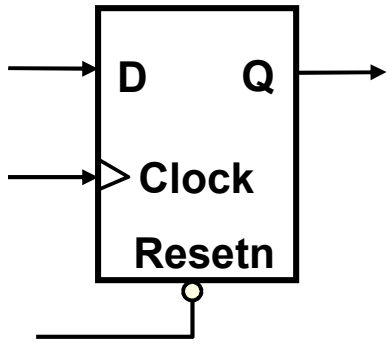
D flip-flop με ασύγχρονο reset

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

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

ARCHITECTURE behavioral OF flipflop_ar IS
BEGIN
    PROCESS (Resetn, Clock)
    BEGIN
        IF Resetn = '0' THEN
            Q <= '0';
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END behavioral;
```



D flip-flop με σύγχρονο reset

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY flipflop_sr IS PORT (
 D, Resetn, Clock  : IN  STD_LOGIC;
 Q                 : OUT   STD_LOGIC);
END flipflop_sr;

ARCHITECTURE behavioral OF flipflop_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;
```



Asychronous vs. Synchronous

Στο IF:

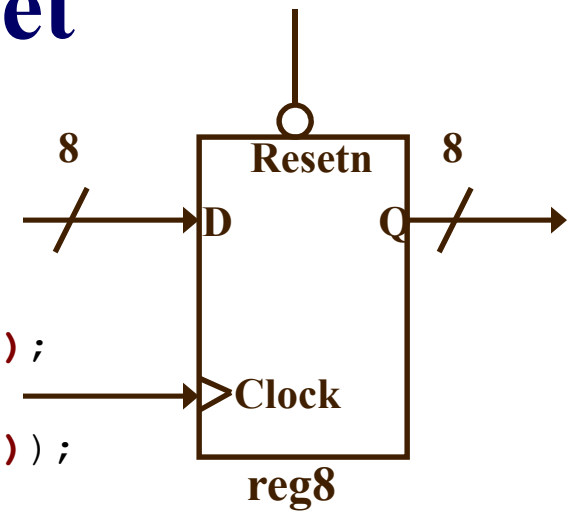
- Asynchronous items:
  - **Πριν το** rising_edge(Clock)
- Synchronous items:
  - **Μετά το** rising_edge(Clock) statement

```
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 με 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;
```



Ρολόι με περίοδο 100 ns
Τη χρονική στιγμή 0 κάνω reset
Στα 0 ns βάζω στην είσοδο 10101010
Στα 100 ns βάζω στην είσοδο 01010101
Στα 150 ns κάνω reset=0.
Ποιες θα είναι οι τιμές της εξόδου;

Generics

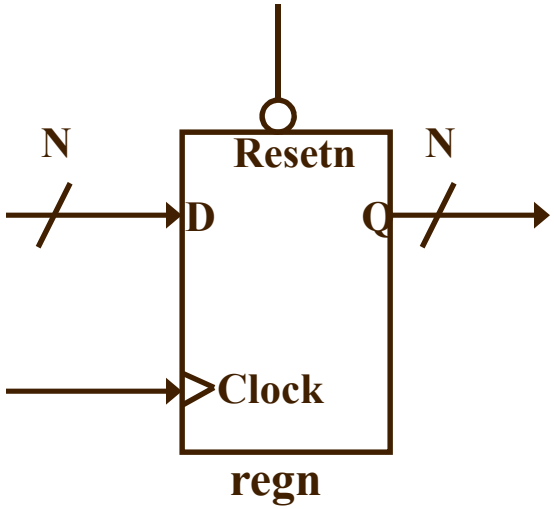
- Είναι **integer** τιμές
- Παίρνουν μια default τιμή

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 ;
```



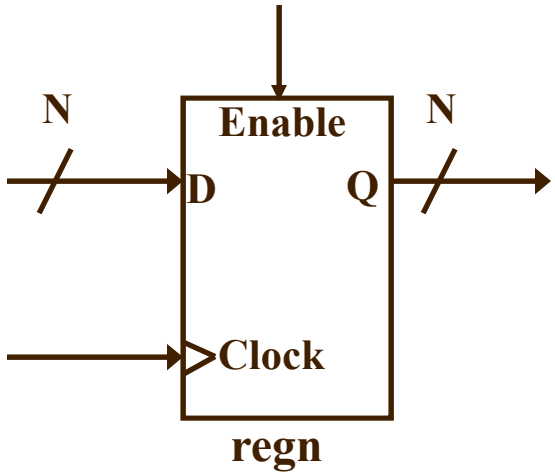
Q: Τι reset έχουμε, σύγχρονο ή ασύγχρονο;

N-bit register with enable

Όταν το enable είναι:

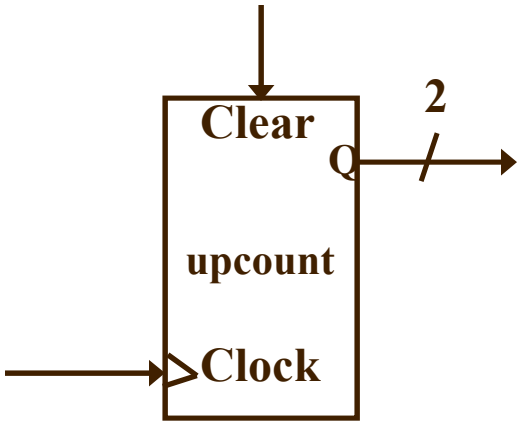
- 0: Η έξοδος μένει σταθερή
- 1: Λειτουργεί κανονικά

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

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

ARCHITECTURE behavioral OF regne IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF rising_edge(Clock) THEN
            IF Enable = '1' THEN
                Q <= D ;
            END IF;
        END IF;
    END PROCESS;
END behavioral;
```



2-bit up-counter με 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;

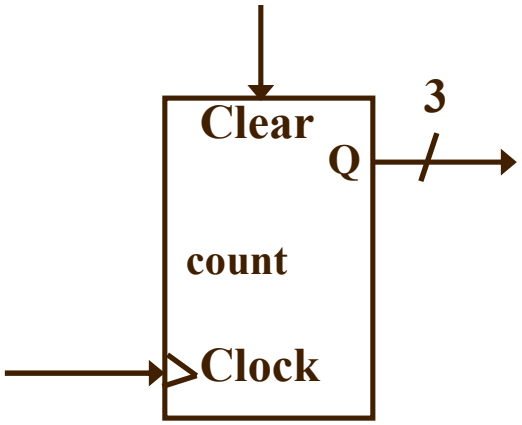
Q4: Τι θα έπρεπε να αλλάξουμε στον κώδικα αν θέλαμε να μετράει ανάποδα;

3-bit up counter με βήμα 3

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

Τι εισόδους θα έχει;

Τι εξόδους;



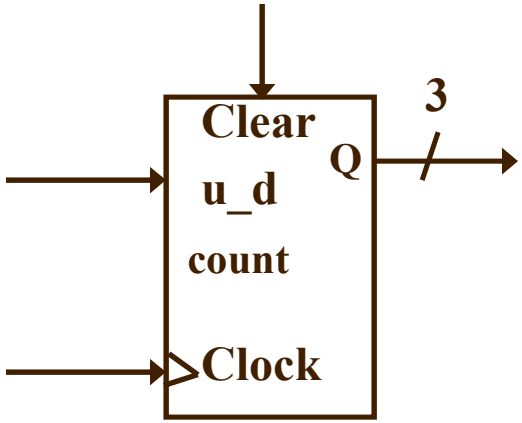

```
ENTITY count IS PORT (
 Clear, Clock: IN  STD_LOGIC;
 Q : OUT STD_LOGIC_VECTOR(2 DOWNTO 0)) ;
END upcount;

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.

Τι εισόδους θα έχει;

Τι εξόδους;




```
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;
```

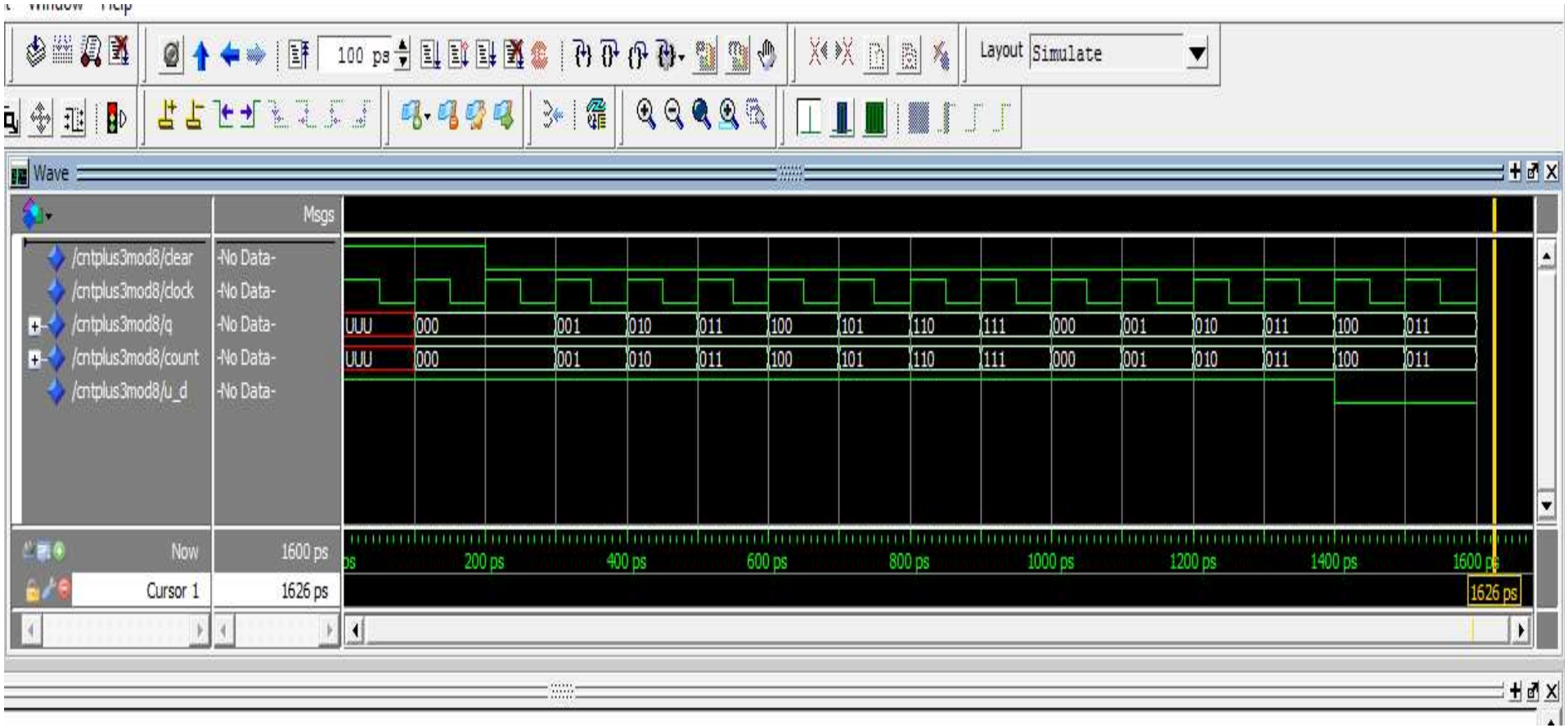
3-bit up/down counter with clear

```
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;

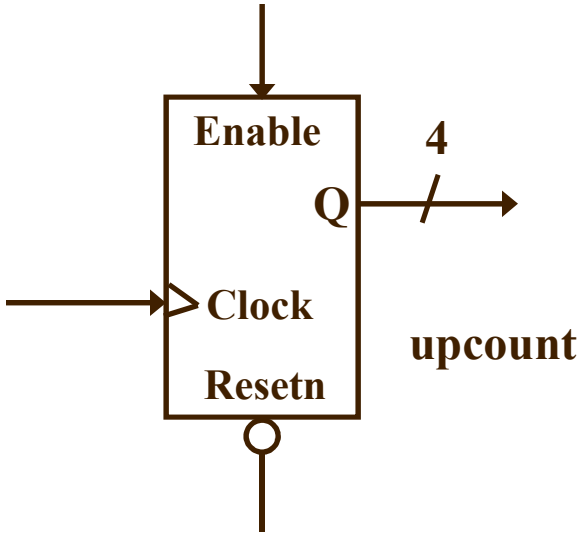
Q2: Γιατί η έξοδος μένει για δύο κύκλους στο 0;

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



4-bit up-counter με ασύγχρονο reset

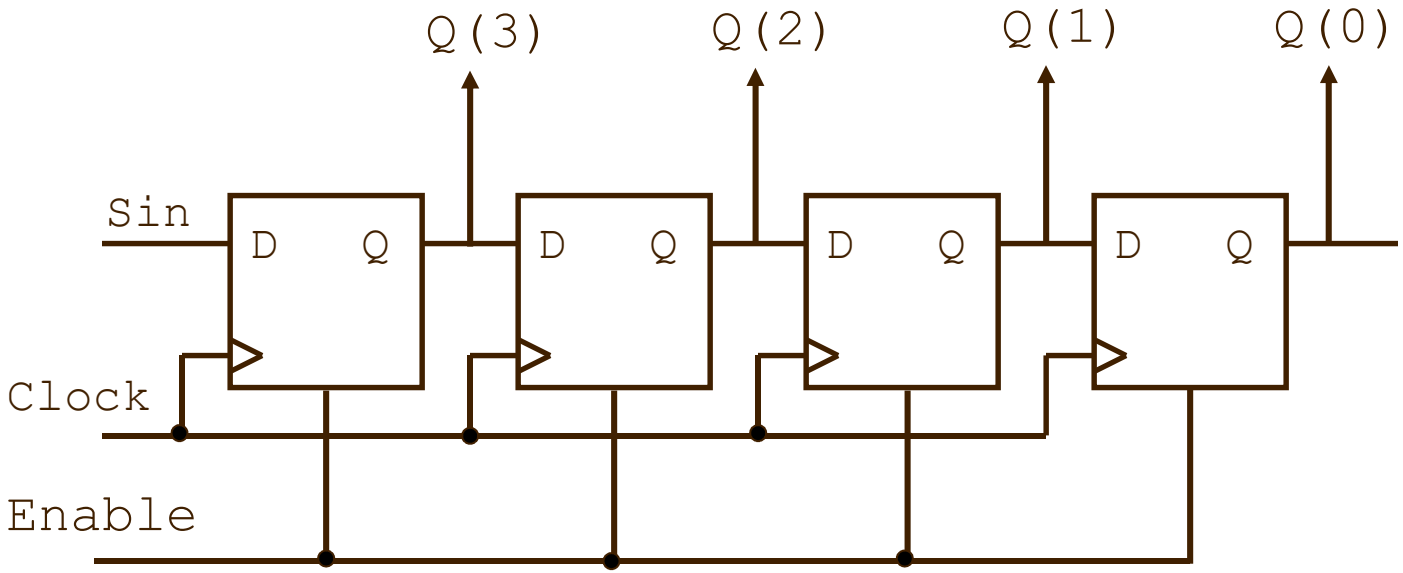
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY upcount_ar IS
    PORT (Clock, Resetn, Enable : IN STD_LOGIC ;
          Q                     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount_ar;
ARCHITECTURE behavioral OF upcount_ar IS
    SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
    PROCESS (Clock, Resetn)
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000";
        ELSIF rising_edge(Clock) THEN
            IF Enable = '1' THEN
                Count <= Count + 1;
            END IF;
        END IF;
    END PROCESS;
    Q <= Count;
END behavioral;
```



Τι κύκλωμα είναι αυτό;

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

 entity up_down_counter is
   port (
     cout    :out std_logic_vector (7 downto 0);
     up_down :in  std_logic;
     clk     :in  std_logic;
     reset   :in  std_logic
   );
 end entity;

 architecture rtl of up_down_counter is
     signal count :std_logic_vector (7 downto 0);
 begin
     process (clk, reset) begin
         if (reset = '1') then
             count <= (others=>'0');
         elsif (rising_edge(clk)) then
             if (up_down = '1') then
                 count <= count + 1;
             else
                 count <= count - 1;
             end if;
         end if;
     end process;
     cout <= count;
 end architecture;
```

Shift register

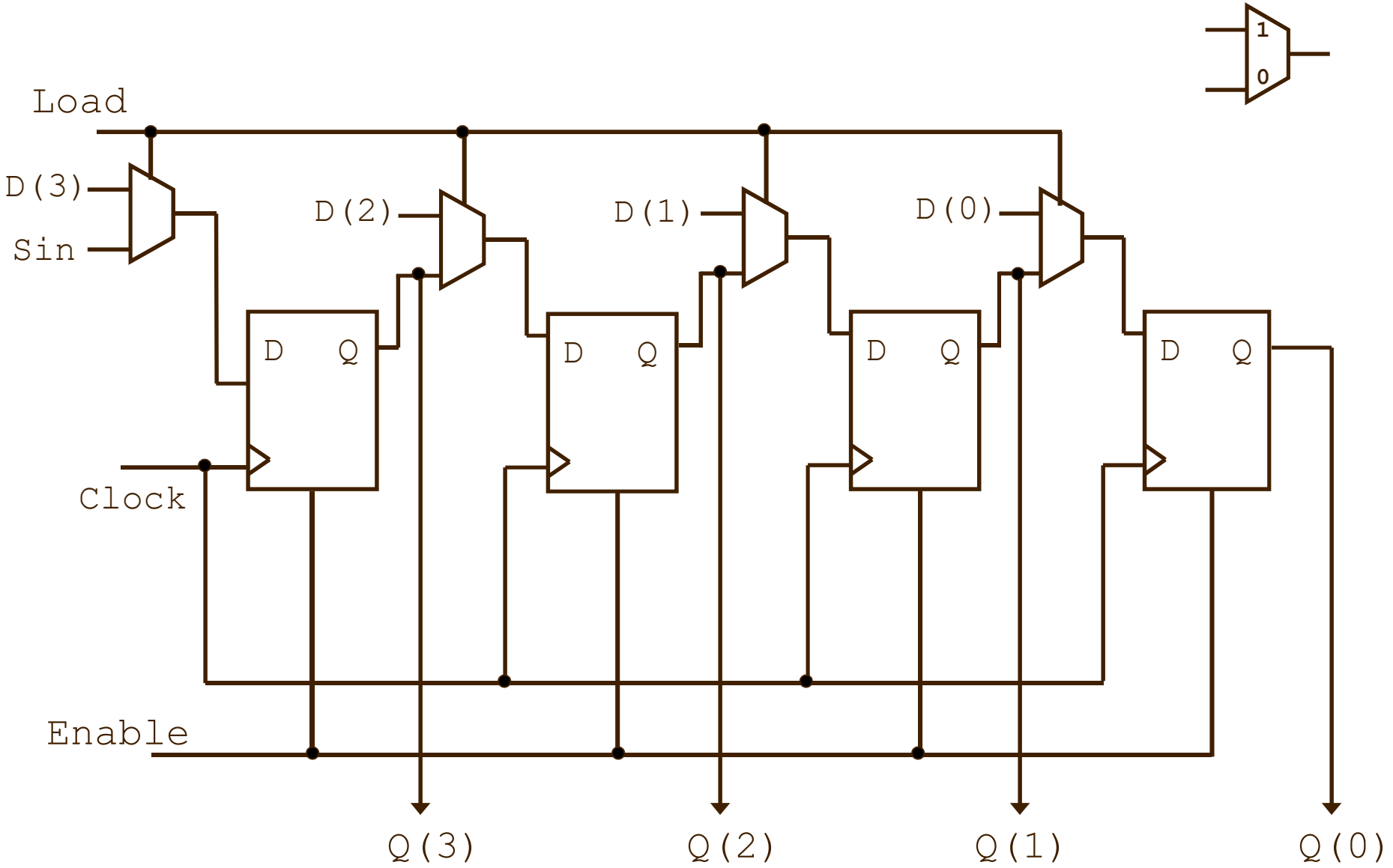


Q1: Πόσοι κύκλοι χρειάζονται για να φορτωθούν τα flip flop;
Q2: Τι τιμή πρέπει να έχει το enable;
Q3: Θεωρήστε τις ακόλουθες τιμές για τις εισόδους Sin, Enable.

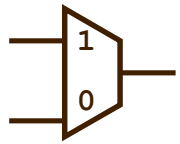
```
Sin: 11010101010
En : 11101011010
```

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

Shift Register με παράλληλη φόρτιση



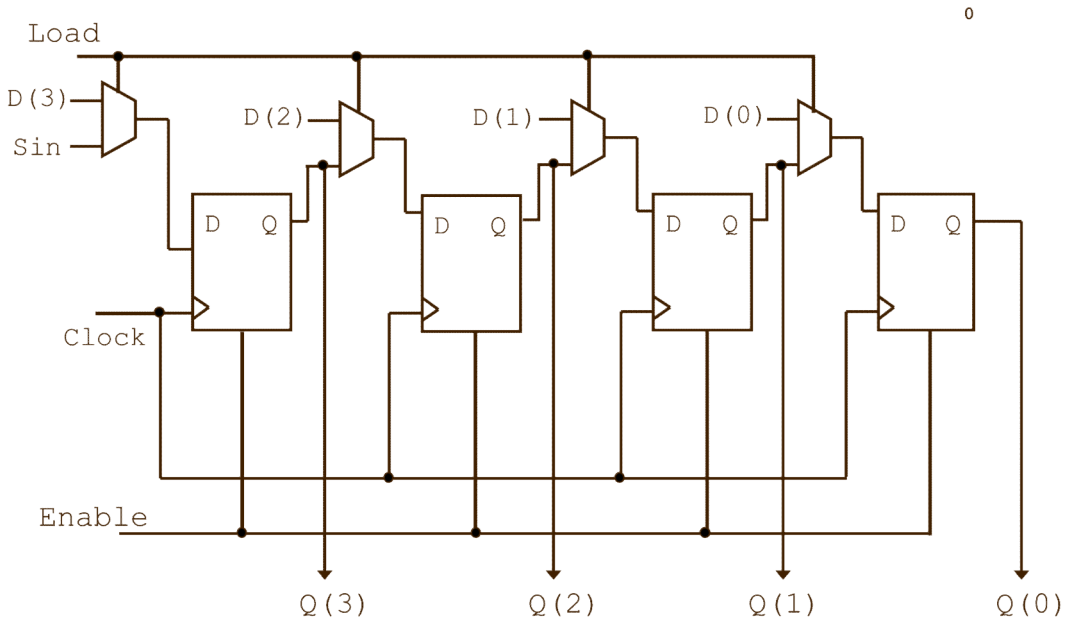
Shift Register με παράλληλη φόρτιση



Q: Θεωρήστε ότι

- αρχικά οι τιμές των flip flop είναι 0
- 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
```

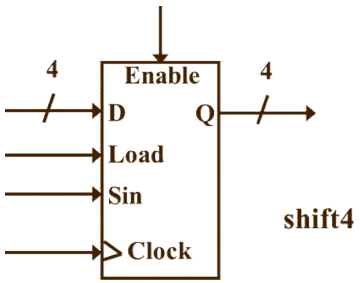


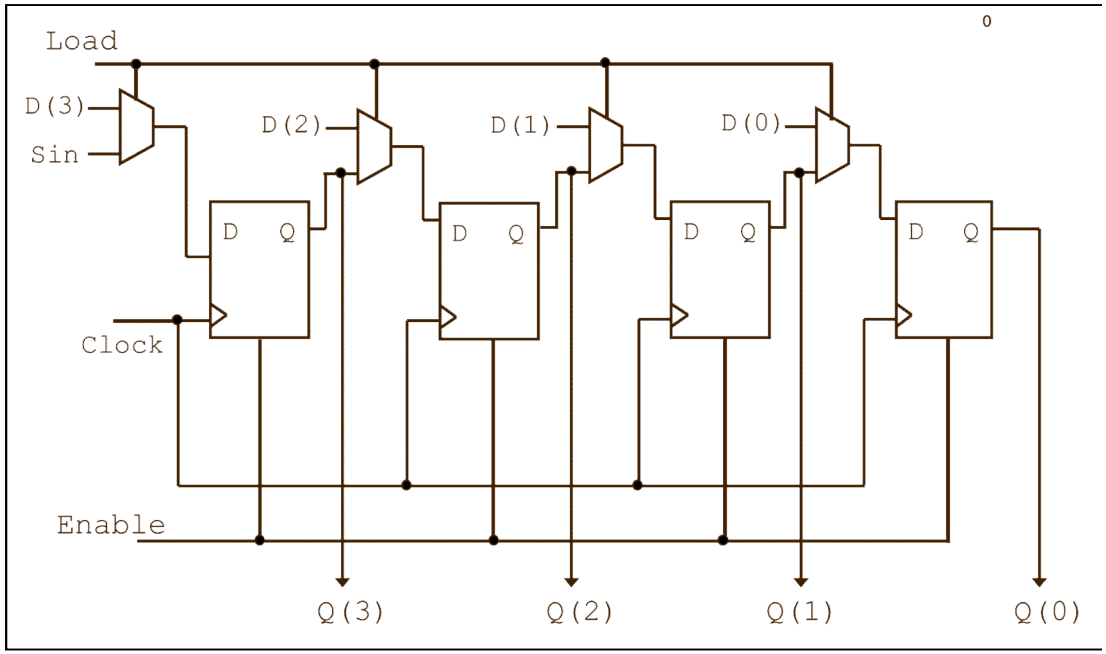
Ποιες οι έξοδοι των flip flop για αυτούς τους 11 κύκλους

Shift Register με παράλληλη φόρτιση

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;
```

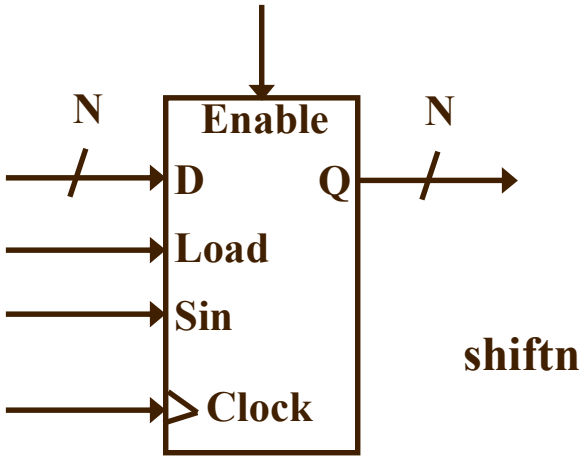





```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shift4 IS PORT (
 D:      IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
 Enable: IN  STD_LOGIC;
 Load:   IN  STD_LOGIC;
 Sin:    IN  STD_LOGIC;
 Clock:  IN  STD_LOGIC;
 Q:      OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END shift4;
```

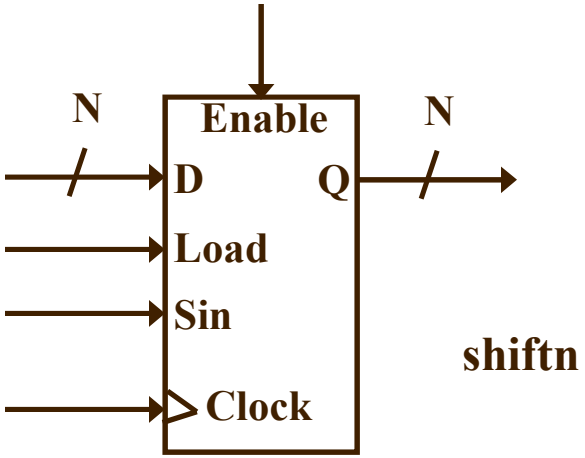
Shift Register με παράλληλη φόρτιση

```
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 ;
```

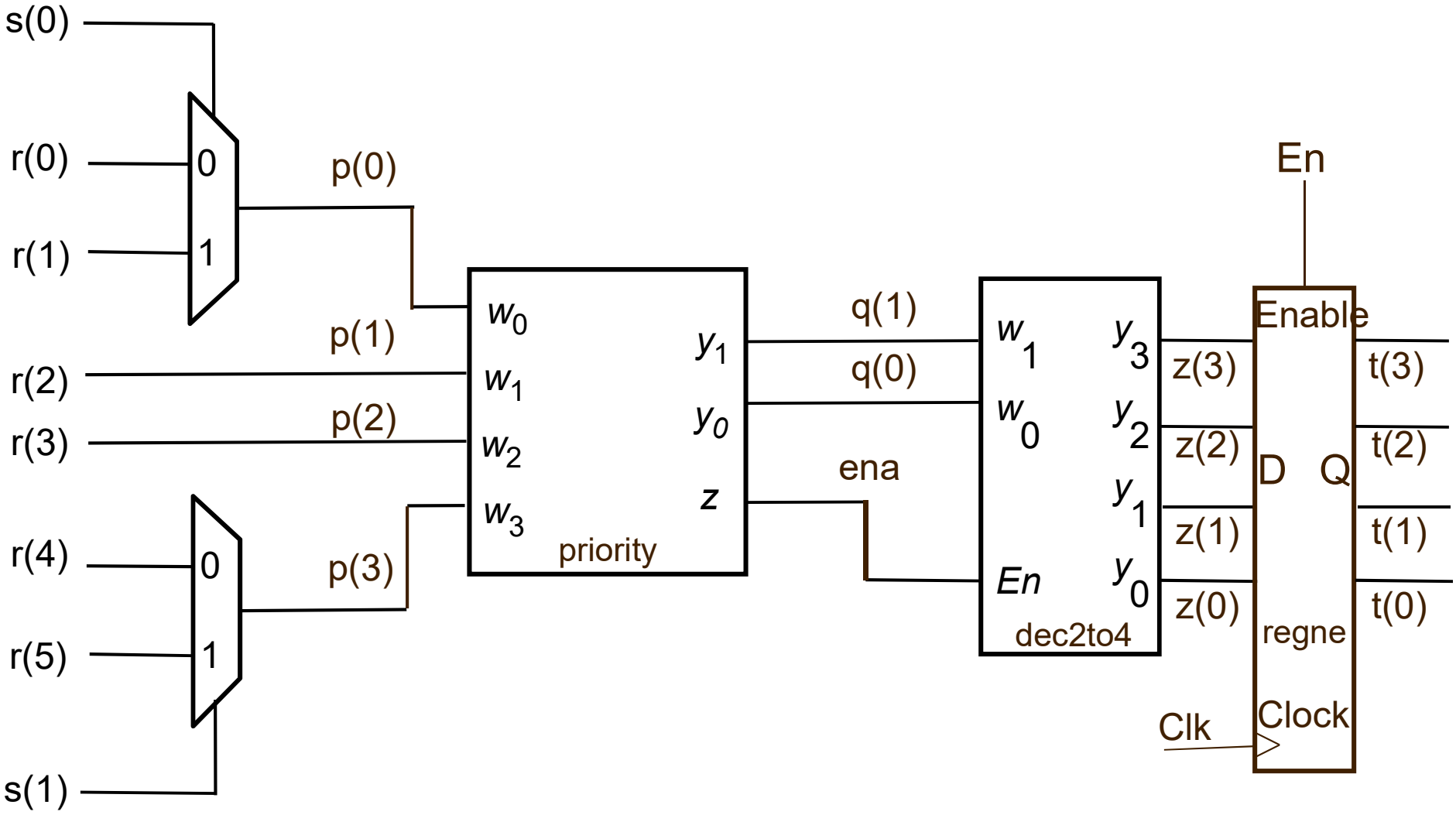


Shift Register με παράλληλη φόρτιση

```
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;
```

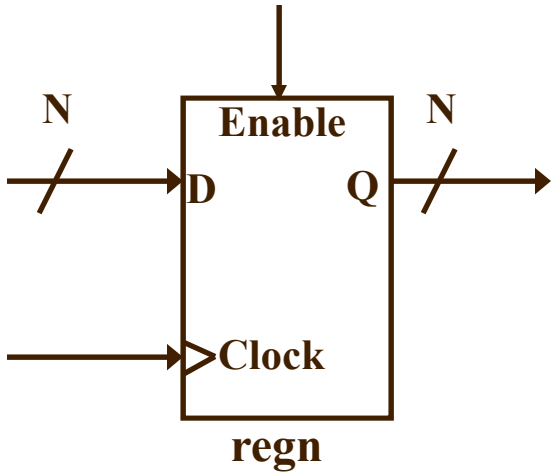


Παράδειγμα "σύνθετης" μονάδας



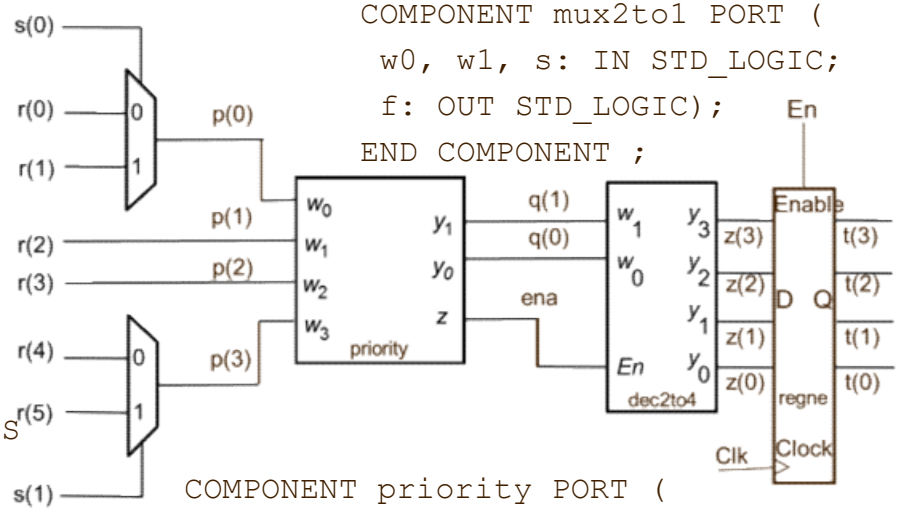
N-bit register with enable

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

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

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



Structural description – example (1)

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

ENTITY priority_resolver IS PORT (
 r: IN    STD_LOGIC_VECTOR(5 DOWNTO 0);
 s: IN    STD_LOGIC_VECTOR(1 DOWNTO 0);
 clk: IN STD_LOGIC;
 en: IN STD_LOGIC;
 t: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END priority_resolver;
ARCHITECTURE structural OF priority_resolver IS
SIGNAL  p : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  q : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
SIGNAL  z : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL  ena : STD_LOGIC ;
BEGIN
 p(1) <= r(2);
 p(2) <= r(3);
 u1: mux2to1 PORT MAP
  (w0 => r(0), w1 => r(1), s => s(0), f => p(0));
 u2: mux2to1 PORT MAP
  (w0 => r(4), w1 => r(5), s => s(1), f => p(3));
 u3: priority PORT MAP
  (w => p, y => q, z => ena);
 u4: dec2to4 PORT MAP
  (w => q, En => ena, y => z);
 u5: regn GENERIC MAP (N => 4) PORT MAP
  (D => z, Enable => En, Clock => Clk, Q => t);
END structural;
```




```
COMPONENT mux2to1 PORT (
 w0, w1, s: IN STD_LOGIC;
 f: OUT STD_LOGIC);
END COMPONENT ;
```

```
COMPONENT priority PORT (
 w: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
 y: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
 z: OUT STD_LOGIC);
END COMPONENT;
```

```
COMPONENT dec2to4 PORT (
 w: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
 En: IN STD_LOGIC;
 y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END COMPONENT ;
```

```
COMPONENT regn GENERIC (N: INTEGER := 8);
 PORT (D: IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
 Enable, Clock: IN STD_LOGIC;
 Q: OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0));
END COMPONENT ;
```

Παράδειγμα ROM

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

ENTITY instruction_rom IS
 GENERIC (w: INTEGER := 16; n: INTEGER := 8; m: INTEGER := 3);
 PORT (
 Instr_addr: IN STD_LOGIC_VECTOR(m-1 DOWNTO 0);
 Instr: out STD_LOGIC_VECTOR(w-1 DOWNTO 0));
END instruction_rom;

ARCHITECTURE ins_rom OF instruction_rom IS
SIGNAL temp: INTEGER RANGE 0 TO n-1;
TYPE vector_array IS ARRAY (0 to n-1) OF STD_LOGIC_VECTOR(w-1 DOWNTO 0);
CONSTANT memory : vector_array :=
( X"0000", X"D459", X"A870", X"7853", X"650D", X"642F", X"F742", X"F548");

BEGIN
 temp <= to_integer(unsigned(Instr_addr));
 Instr <= memory(temp);
END instruction_rom;
```

Πόσες θέσεις έχει;

Πόσα bit κάθε θέση;

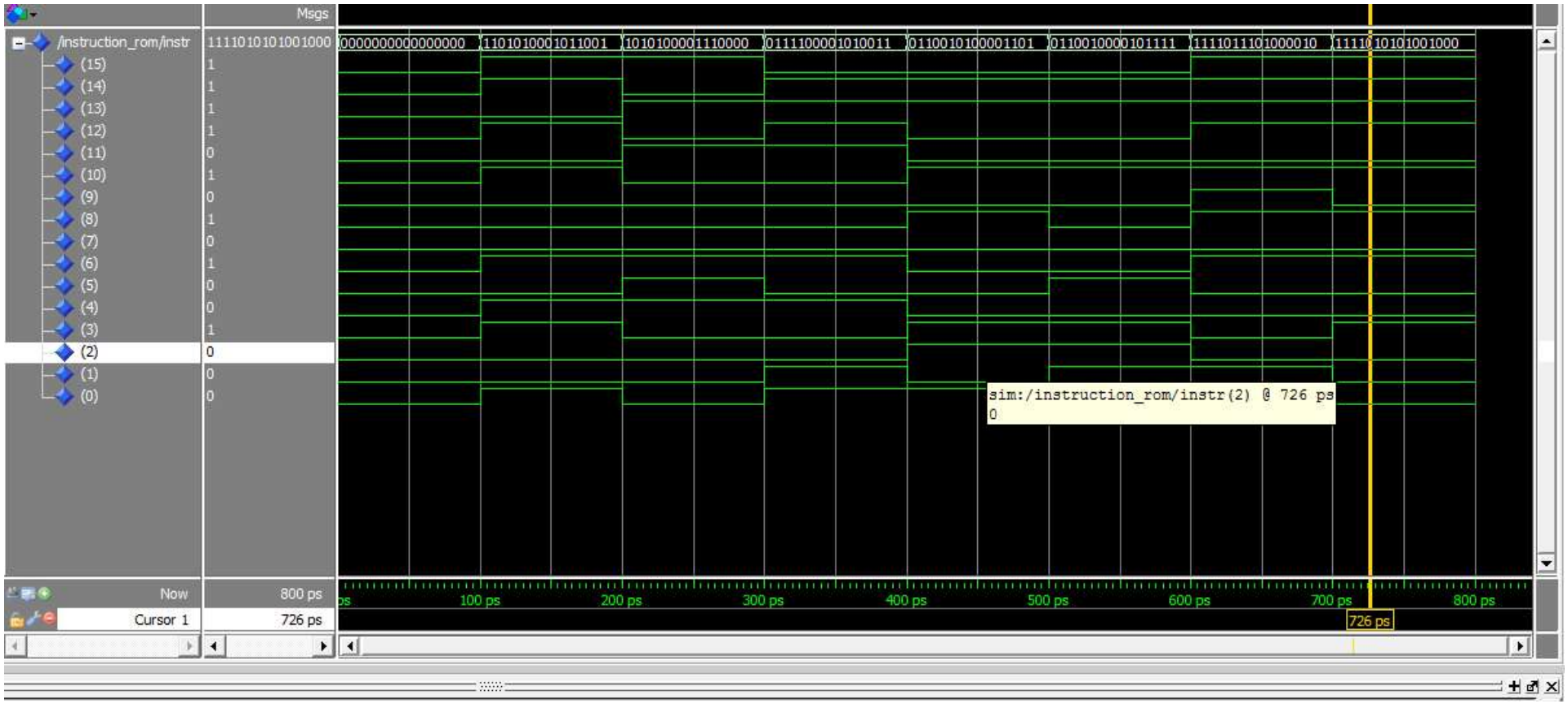
Πώς υλοποιείται στην πράξη;

Τρέξιμο της ROM

```
CONSTANT memory : vector_array := (
X"0000",X"D459",X"A870",X"7853",X"650D",X"642F",X"F742",X"F548");
```

Q1: Πόσους κύκλους έτρεξε;

Q2: Τι διεύθυνση δώσαμε σε κάθε κύκλο;



Τύποι Signed and Unsigned

Behave exactly **like**

STD_LOGIC_VECTOR

plus, they determine whether a given vector should be treated as a signed or unsigned number.

Require

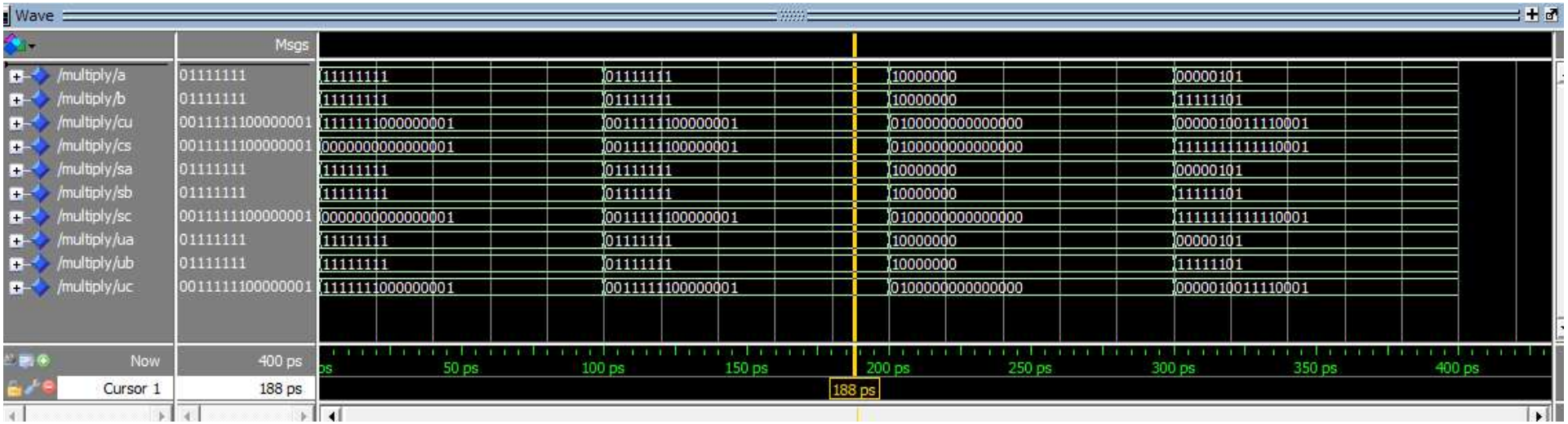
USE ieee.numeric_std.all;

Multiplication of signed and unsigned numbers (1)

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

entity multiply is port(
 a  : in  STD_LOGIC_VECTOR( 7 downto 0);
 b  : in  STD_LOGIC_VECTOR( 7 downto 0);
 cu : out STD_LOGIC_VECTOR(15 downto 0);
 cs : out STD_LOGIC_VECTOR(15 downto 0));
end multiply;

architecture dataflow of multiply is
 SIGNAL sa: SIGNED(7 downto 0);
 SIGNAL sb: SIGNED(7 downto 0);
 SIGNAL sc: SIGNED(15 downto 0);

 SIGNAL ua: UNSIGNED(7 downto 0);
 SIGNAL ub: UNSIGNED(7 downto 0);
 SIGNAL uc: UNSIGNED(15 downto 0);
```

```
begin
-- signed multiplication
 sa <= SIGNED(a);
 sb <= SIGNED(b);
 sc <= sa * sb;
 cs <= STD_LOGIC_VECTOR(sc);

-- unsigned multiplication
 ua <= UNSIGNED(a);
 ub <= UNSIGNED(b);
 uc <= ua * ub;
 cu <= STD_LOGIC_VECTOR(uc);
end dataflow;
```

Εξηγήστε τα αποτελέσματα



Q1: Πόσους κύκλους έτρεξε;
Q2: Ποιους αριθμούς πολλαπλασιάσαμε σε κάθε κύκλο;
Q3: Ποια έξοδος έδωσε ορθό αποτέλεσμα σε κάθε περίπτωση;

```
begin
-- signed multiplication
 sa <= SIGNED(a);
 sb <= SIGNED(b);
 sc <= sa * sb;
 cs <= STD_LOGIC_VECTOR(sc);

-- unsigned multiplication
 ua <= UNSIGNED(a);
 ub <= UNSIGNED(b);
 uc <= ua * ub;
 cu <= STD_LOGIC_VECTOR(uc);
end dataflow;
```

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

entity multiply is port(
 a  : in  STD_LOGIC_VECTOR( 7 downto 0);
 b  : in  STD_LOGIC_VECTOR( 7 downto 0);
 cu : out STD_LOGIC_VECTOR(15 downto 0);
 cs : out STD_LOGIC_VECTOR(15 downto 0));
end multiply;

architecture dataflow of multiply is

 SIGNAL sa: SIGNED(7 downto 0);
 SIGNAL sb: SIGNED(7 downto 0);
 SIGNAL sc: SIGNED(15 downto 0);

 SIGNAL ua: UNSIGNED(7 downto 0);
 SIGNAL ub: UNSIGNED(7 downto 0);
 SIGNAL uc: UNSIGNED(15 downto 0);
```

Αριθμητικές πράξεις

Συνθέσιμες:

- Addition, +
- Subtraction, -
- Comparisons, >, >=, <, <=
- Multiplication, *
- Division by a power of 2, (equivalent to right shift)
- Shifts by a constant, SHL, SHR

Mixed Style Modeling

architecture ARCHITECTURE_NAME **of** ENTITY_NAME **is**

- Declare signals, constants, functions, procedures…
- Component declarations

begin

Concurrent statements:

- Concurrent simple signal assignment
- Conditional signal assignment
- Selected signal assignment
- Generate statement

- Component instantiation statement

- Process statement
- **inside process we can use only sequential statements**

end ARCHITECTURE_NAME;

Τέλος παρουσίασης

Sequential Logic Synthesis (1)

Use processes with very simple structure only
to describe

- registers
- shift registers
- counters
- state machines.

Create **generic** entities for registers, shift registers, and counters, and instantiate the corresponding components in a higher level circuit using GENERIC MAP PORT MAP.

Supplement sequential components with combinational logic described using concurrent statements.

Sequential Logic Synthesis (2)

1. Use Processes with IF and CASE statements only. Do not use LOOPS or VARIABLES.
2. Sensitivity list of the PROCESS should include **only** signals that can by themsleves change the outputs of the sequential circuit (typically, clock and asynchronous set or reset)
3. Do not use PROCESSes without sensitivity list

   (they can be synthesizable, but make simulation inefficient)