dscal
DIGITAL SYSTEMS & COMPUTER ARCHITECTURE LABORATORY

9ο Εργαστήριο Space Data Systems

VHDL

-

Report, assert - Finite State Machines - Timing

Βασιλόπουλος Διονύσης

ΕΔΙΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

dit ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

VHDL – Report

Ακολουθιακή εντολή (εντός Process και εντός simulation)

report <message_string> [severity <severity_level>];

report "this is a serious message" severity warning;

Severity levels: **note**, warning, error, failure

https://insights.sigasi.com/tech/vhdl-assert-and-report/

VHDL – Assert

Ακολουθιακή και σύγχρονη εντολή (συνήθως εντός simulation)

Ελέγχει εάν μια συνθήκη είναι αληθής. Εάν ΔΕΝ είναι τότε τυπώνει ένα μήνυμα.

```
assert <condition>;
assert <condition> severity <severity_level>;
assert <condition> report <message_string>;
assert <condition> report <message_string> severity <severity_level>;

condition_to_check <= true;
assert condition_to_check = true  report "Assertion failed! Condition is false." severity error;
    -- If the assert passes, this line will be executed
report "Assertion passed! Condition is true." severity note;
```

Severity levels: note, warning, **error**, **failure**

Scalar type

```
assert i < 5 report "unexpected value. i = " & integer'image(i);
```

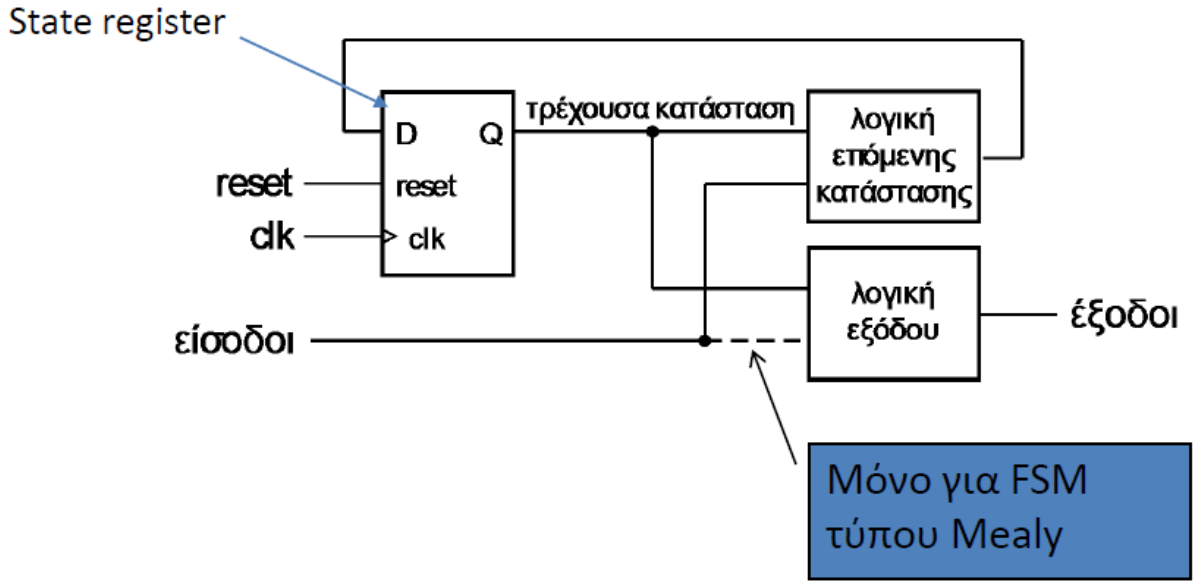
https://peterfab.com/ref/vhdl/vhdl_renerta/mobile/source/vhd00007.htm

VHDL – Finite State Machines

- Χρησιμοποιούνται για να υλοποιήσουν μια ακολουθία ελέγχου
- Μία FSM (Finite-State Machine – Μηχανή Πεπερασμένων καταστάσεων) ορίζεται από
  - σύνολο εισόδων: I
  - σύνολο εξόδων: O
  - σύνολο καταστάσεων: S
  - αρχική κατάσταση: s0 ανήκει στο S
  - συνάρτηση μετάβασης από μία κατάσταση στην επόμενη
  - συνάρτηση εξόδου

VHDL – Finite State Machines

Γενικό διάγραμμα



VHDL – Finite State Machines

Κωδικοποίηση Καταστάσεων

- Κωδικοποιούνται στο δυαδικό
  - Ν καταστάσεις: χρειάζονται τουλάχιστον log2*N* bit
- Η κωδικοποιημένη τιμή χρησιμοποιείται στα κυκλώματα για τις συναρτήσεις μετάβασης και εξόδου
  - η κωδικοποίηση επηρεάζει την πολυπλοκότητα του κυκλώματος
- Είναι δύσκολο να βρεθεί βέλτιστη κωδικοποίηση
  - τα εργαλεία CAD συνήθως κάνουν την κωδικοποίηση
- Η κωδικοποίηση one-hot δουλεύει καλά στα FPGA
- Συχνά χρησιμοποιείται το 000...0 για την κατάσταση αδράνειας
  - μηδενίζει τον καταχωρητή στην κατάσταση αδράνειας

VHDL – Finite State Machines

Κωδικοποίηση Καταστάσεων - Παθητική

- Χρησιμοποιούμε ένα τύπο **απαρίθμησης** (enumerated type) για τις τιμές των καταστάσεων.
- Η μέθοδος είναι **αφηρημένη** ώστε να μην προδιαγράψουμε εμείς την κωδικοποίηση

Η πραγματική κωδικοποίηση (binary, grey,…) καθορίζεται από τον synthesizer.

type state_type is (zero, one, two)
signal current_state, next_state: state_type

VHDL – Finite State Machines

Κωδικοποίηση Καταστάσεων – Ενεργητική -1

- Χρησιμοποιούμε ένα υπο-τύπο **απαρίθμησης** για τις τιμές των καταστάσεων.
- Η μέθοδος καθορίζει τις τιμές της κωδικοποίησης. Έχουμε καθορίσει εμείς ορισμένες προδιαγραφές (π.χ. μήκος κωδικής λέξης)

```
subtype state_type is std_logic_vector(1 downto 0);
constant zero:std_logic_vector(1 downto 0) :="00";
constant one :std_logic_vector(1 downto 0) :="01";
constant two :std_logic_vector(1 downto 0) :="10";
signal current_state, next_state: state_type

current_state<=one;
```

VHDL – Finite State Machines

Κωδικοποίηση Καταστάσεων – Ενεργητική -2

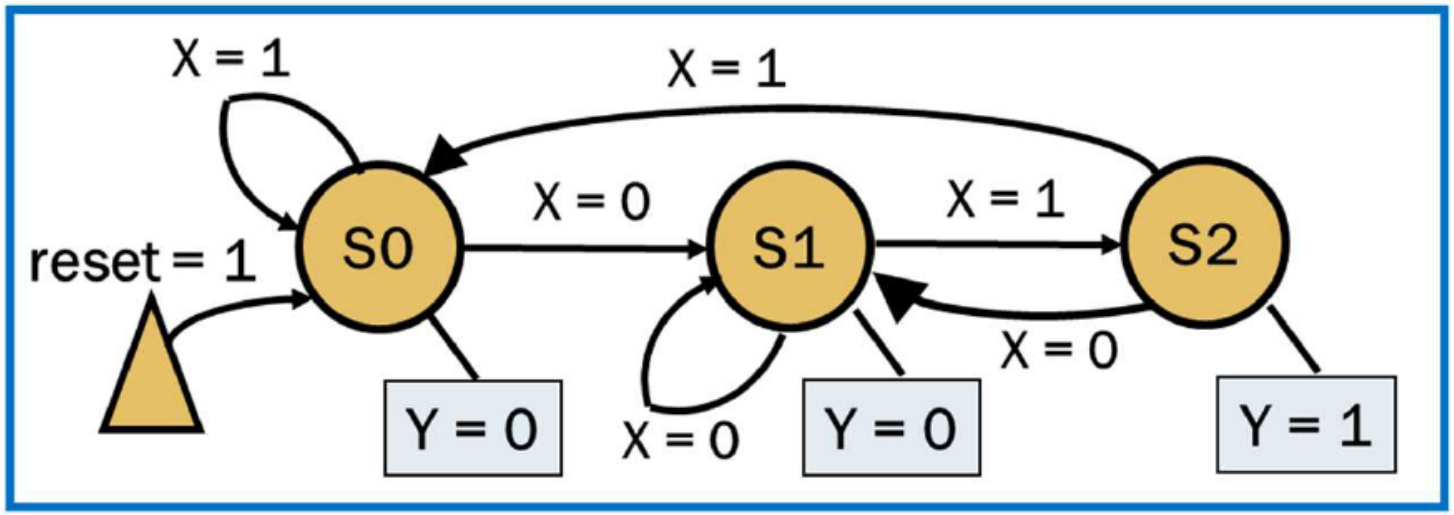
- Χρήση ενός, ορισμένου από τον χρήστη, χαρακτηριστικού (attribute).
- Απόδοση του χαρακτηριστικού σε κάποιο τύπο.

Είναι string που γίνεται parse, και οι τιμές αποδίδονται σε FSM_states οριζόμενες από τα κενά του string.

```
type FSM_states is (zero, one ,two);

attribute enum_encoding: string;

attribute enum_encoding of FSM_states: type is "001 010 100";

signal current_state, next_state: FSM_states;
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PATTERN_FSM is
   Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          X : in STD_LOGIC;
          Y : out STD_LOGIC);
end PATTERN_FSM;

architecture Behavioral of PATTERN_FSM is

-- state definition
   type FSM_states is (S0, S1, S2);

-- internal signals
   signal current_state, next_state: FSM_states;
   signal X_in : STD_LOGIC; -- Only when there is an INREG
```

ΛΟΓΙΚΗ για RESET

```
begin
-- Optional for synchronization. RESET case
   INREG:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then X_in <= '1'; -- to trap state S0 during reset
         else X_in <= X;
         end if;
      end if;
   end process;

-- Common process for all FSMs to create state register
   SYNC:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then current_state <= S0;
         else current_state <= next_state;
         end if;
      end if;
   end process;
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Moore

```
-- Process to create next state logic and output logic
   ASYNC: process (current_state, X_in)  -- Moore
   begin
   -- FSM next state and output initialization
      next_state <= S0;
      Y <= '0';
      case current_state is
         when S0 =>
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         when S1 =>
            if (X_in = '1') then next_state <= S2;
            else next_state <= S1;
            end if;
         when S2 => Y <= '1';
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         -- fail-safe behavior
         when others => next_state <= S0;
      end case;
   end process;

end Behavioral;
```

Αρχική τιμή οι τιμές Reset

ΛΟΓΙΚΗ για ΕΠΟΜΕΝΗ ΚΑΤΑΣΤΑΣΗ

ΛΟΓΙΚΗ για ΕΞΟΔΟ

VHDL – Finite State Machines

Παράδειγμα –- Moore – The 3-process solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PATTERN_FSM is
   Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          X : in STD_LOGIC;
          Y : out STD_LOGIC);
end PATTERN_FSM;

architecture Behavioral of PATTERN_FSM is

-- state definition
   type FSM_states is (S0, S1, S2);

-- internal signals
   signal current_state, next_state: FSM_states;
   signal X_in : STD_LOGIC; -- Only when there is an INREG
```

```
begin

-- Common process for all FSMs to create state register
-- STATE MEMORY PROCESS plus input synchronization

   SYNC:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then
                current_state <= S0; -- to trap state S0 during reset
                X_in<='1';
         else
                current_state <= next_state;
                X_in<=X;
         end if;
      end if;
   end process;
```

VHDL – Finite State Machines

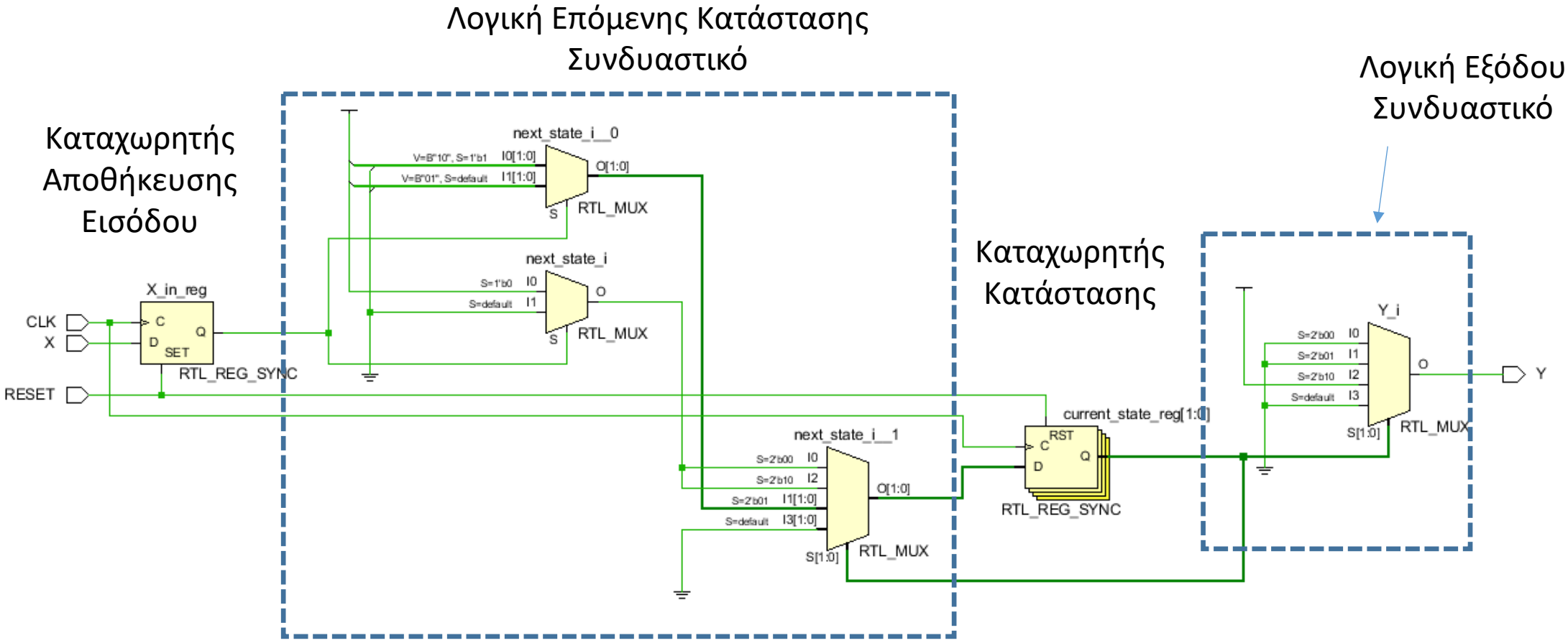
Παράδειγμα –- Moore – The 3-process solution

```
-- NEXT STATE LOGIC PROCESS
next_state: process (current_state, X_in)  -- Moore
begin
next_state <= S0;
case current_state is
        when S0 =>
           if (X_in = '0') then next_state <= S1;
           else next_state <= S0;
           end if;
        when S1 =>
           if (X_in = '1') then next_state <= S2;
           else next_state <= S1;
           end if;
        when S2 => if (X_in = '0') then next_state <= S1;
           else next_state <= S0;
           end if;
        when others => next_state <= S0; -- fail-safe behavior
end case;
end next_state process;
end Behavioral;
```

```
-- OUTPUT LOGIC PROCESS
output_logic: process (current_state)  -- Moore
begin

   -- FSM output initialization

case current_state is
      when S2 =>
                Y <= '1';
       when others=>
                Y<='0';
end case;
end process output_logic;

end Behavioral;
```

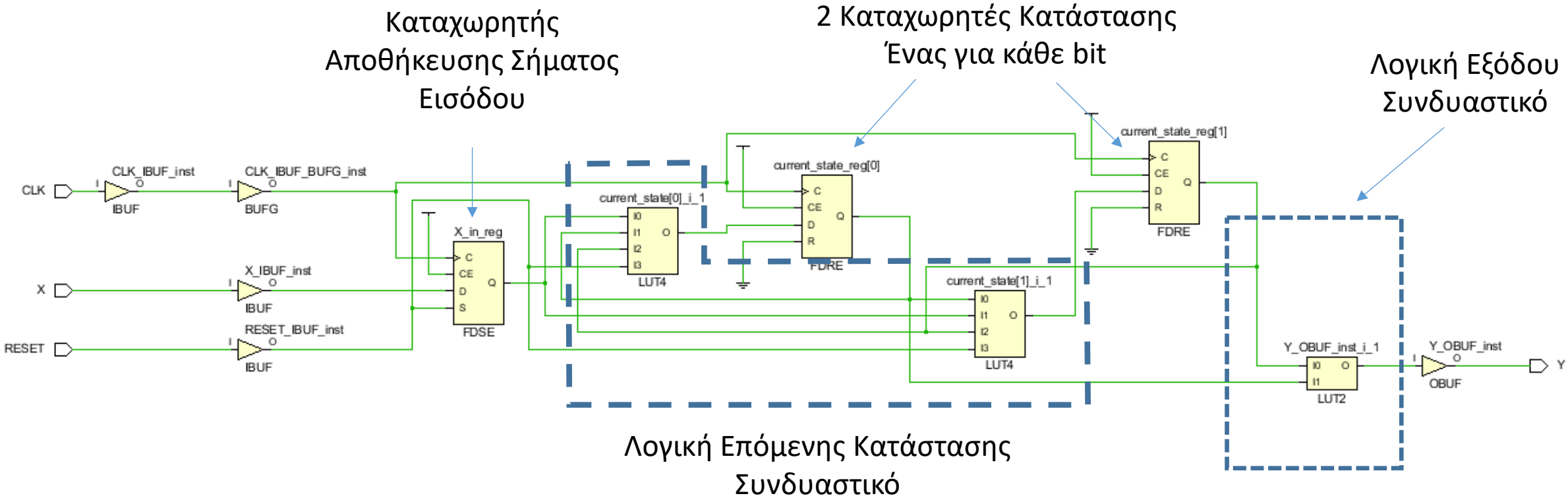
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore - RTL



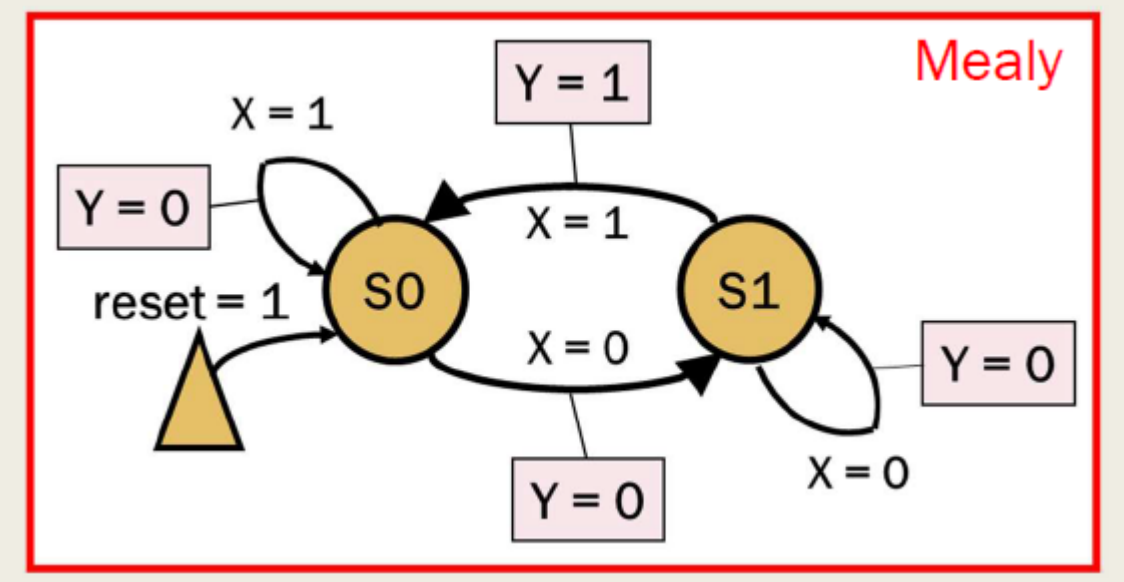
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore - Synthesis



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Mealy



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Mealy

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PATTERN_FSM is
   Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          X : in STD_LOGIC;
          Y : out STD_LOGIC);
end PATTERN_FSM;

architecture Behavioral of PATTERN_FSM is

-- state definition
   type FSM_states is (S0, S1);

-- internal signals
   signal current_state, next_state: FSM_states;
   signal X_in : STD_LOGIC; -- Only when there is an INREG
```

```
begin
-- Optional for sychronization
   INREG:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then X_in <= '1'; -- to trap state S0 during reset
         else X_in <= X;
         end if;
      end if;
   end process;

-- Common process for all FSMs to create state register
   SYNC:  process (CLK)
   begin
      if (CLK = '1' and CLK'event) then
         if (RESET = '1') then current_state <= S0;
         else current_state <= next_state;
         end if;
      end if;
   end process;
```

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 - Mealy

```
-- Process to create next state logic and output logic
   ASYNC: process (current_state, X_in)  -- Moore
   begin
   -- FSM next state and output initialization
      next_state <= S0;
      Y <= '0';
      case current_state is
         when S0 =>
            if (X_in = '0') then next_state <= S1;
            else next_state <= S0;
            end if;
         when S1 =>
            if (X_in = '1') then
                  next_state <= S0;
                  Y<='1';
            else next_state <= S1;
            end if;
-- fail-safe behavior
            when others => next_state <= S0;
      end case;
   end process;
end Behavioral;
```

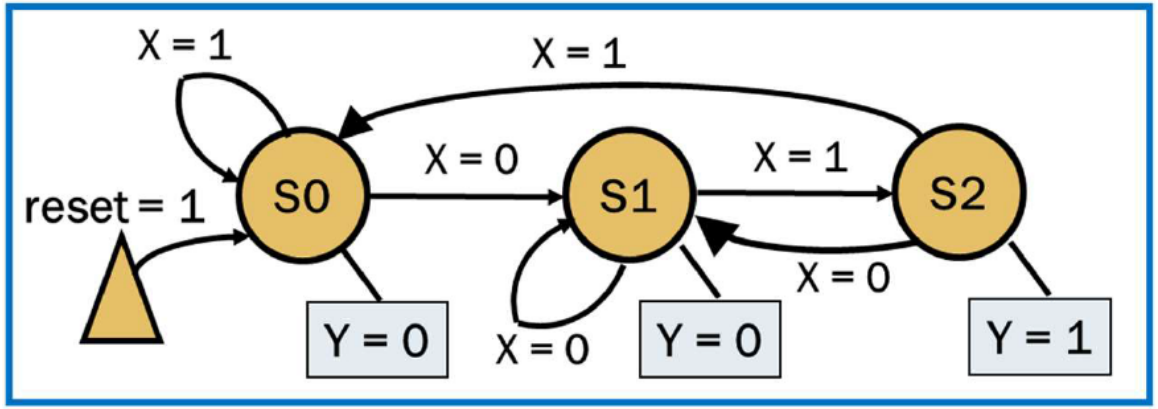
ΛΟΓΙΚΗ για ΕΞΟΔΟ

VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

S_{old}	$S_{old}(1)$	$S_{old}(0)$	Xin	$S_{new}(1)$	$S_{new}(0)$	S_{new}
S_0	0	0	0	0	1	S_1
S_1	0	1	0	0	1	S_1
S_2	1	0	0	0	1	S_1
S_3	1	1	X	0	0	S_0
S_0	0	0	1	0	0	S_0
S_1	0	1	1	1	0	S_2
S_2	1	0	1	0	0	S_0
S_X	1	1	X	0	0	S_0

S	S(1)	S(0)	Y
S_0	0	0	0
S_1	0	1	0
S_2	1	0	1
S_x	X	X	0



VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

Πίνακες KARNAUGH

$S_{new}(1)$

S_{cur}\Xin	0	1
00	0	0
01	0	1
11	0	0
10	0	0

$S_{new}(0)$

S_{cur}\Xin	0	1
00	1	0
01	1	0
11	0	0
10	1	0

Y

S_{cur}	Y
00	0
01	0
11	0
10	(1)

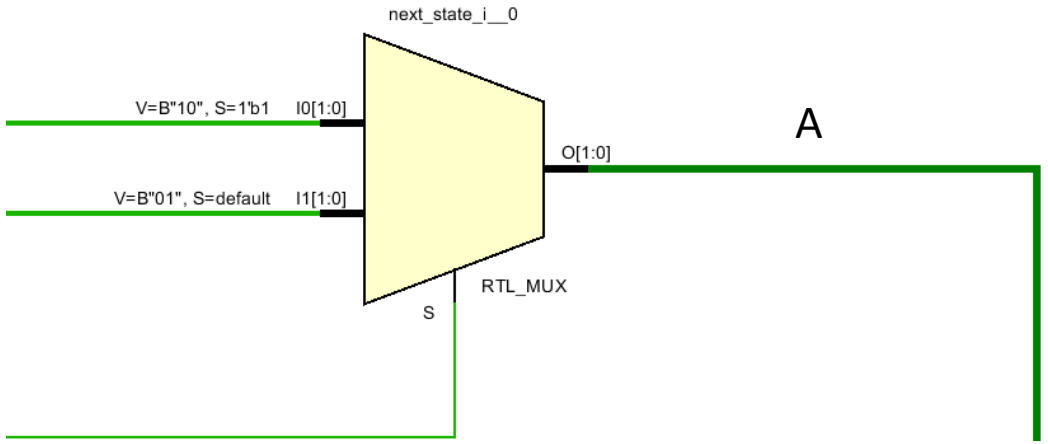
$S_{new}(1)= S_{cur}(1)' S_{cur}(0)Xin$
$S_{new}(0)=S_{cur}(1)'Xin'+ S_{cur}(1) S_{cur}(0)'Xin'$
$Y = S_{cur}(1)S_{cur}(0)'$

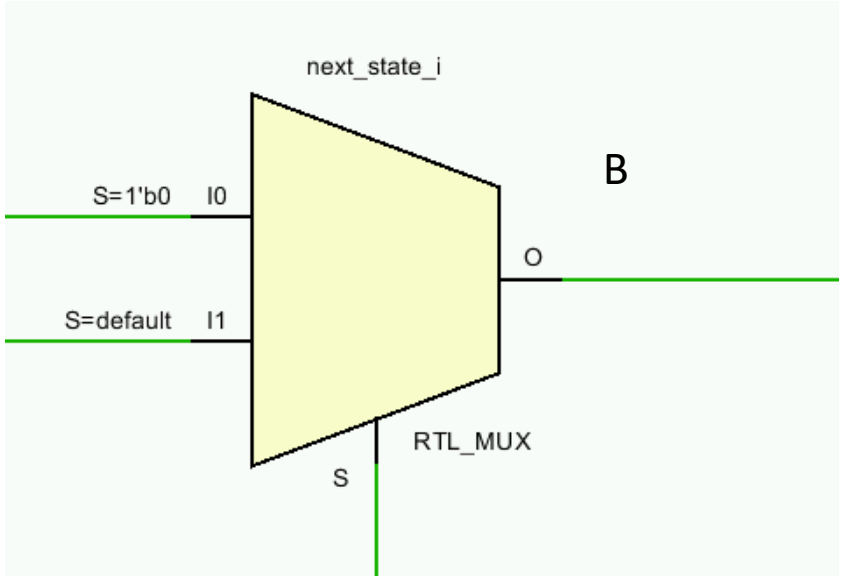
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

X_{in}=S	Out	$S1_{new}$
1	10	S_2
0	01	S_1

X_{in}=S	Out	$S1_{new}$
0	1	S_1
1	0	S_0





VHDL – Finite State Machines

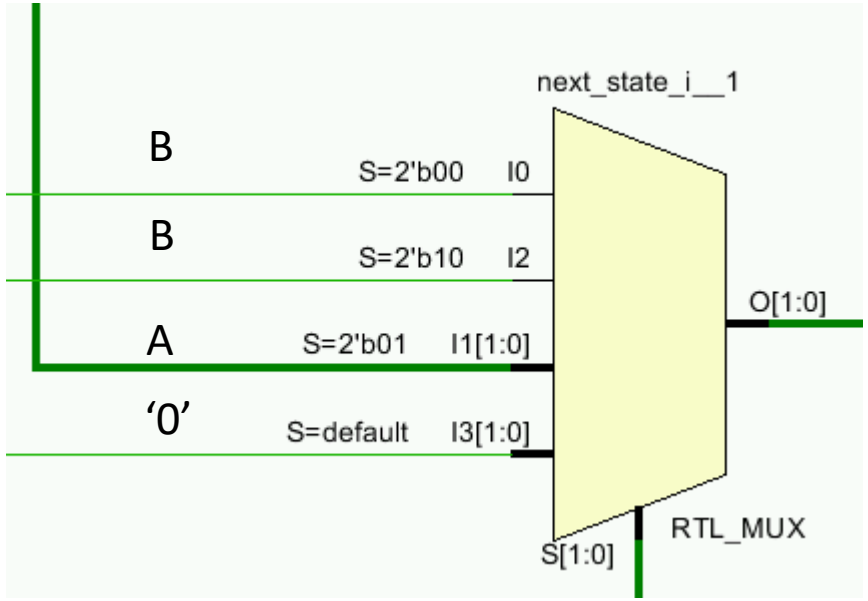
Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

X_{in}=S	Out	S_{new}
1	10	S_2
0	01	S_1

A

X_{in}=S	Out	S_{new}
0	01	S_1
1	00	S_0

B

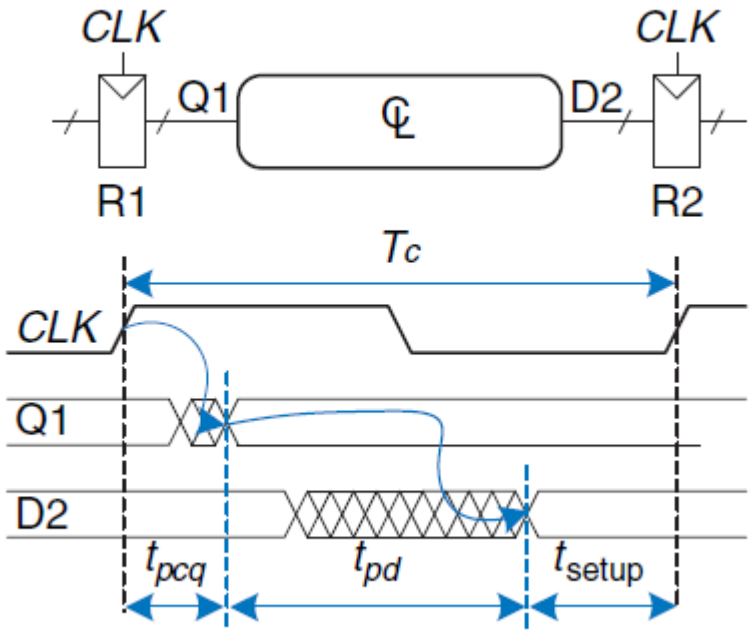


S_{old}	X_{in}	S_{new}
00(0)	0	S_1
01(1)	0	S_1
10(2)	0	S_1
11(X)	0	S_0
00(0)	1	S_0
01(1)	1	S_2
10(2)	1	S_0
11(X)	1	S_0

Sequential - Timing

Tc ≥ tpcq+ tpd + tsetup

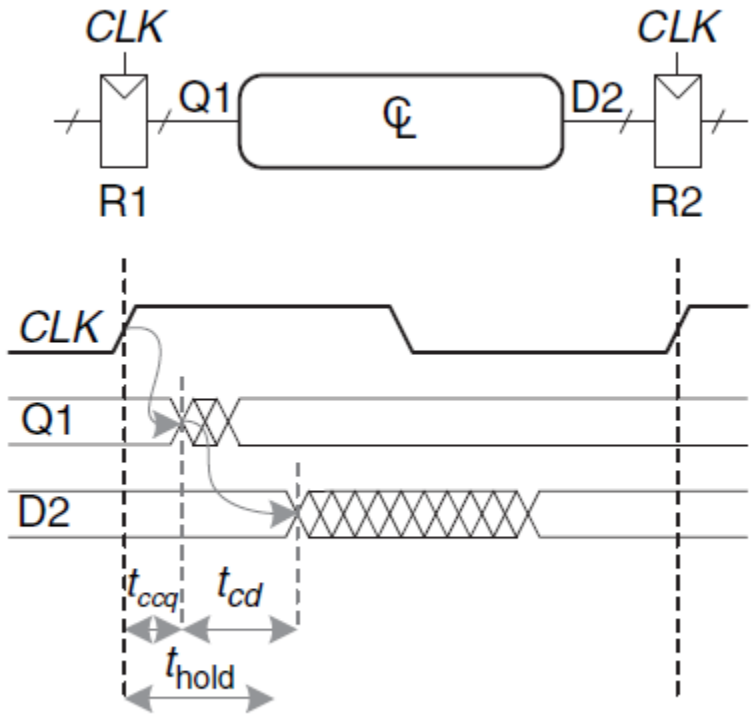
Περίοδος ρολογιού> χρόνος για πάρει τη σωστή τιμή η έξοδος του Reg
+χρόνος διάδοσης
+χρόνος setup



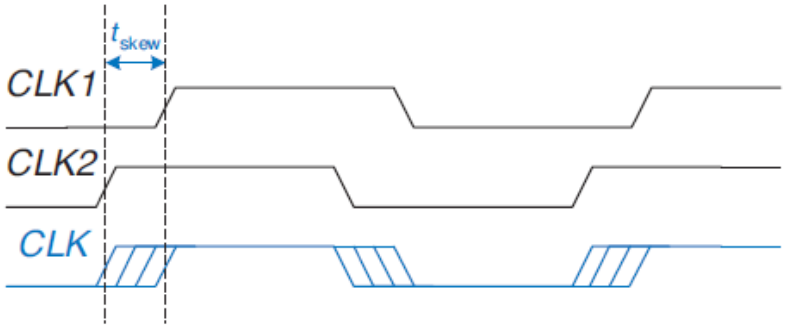
Sequential - Timing

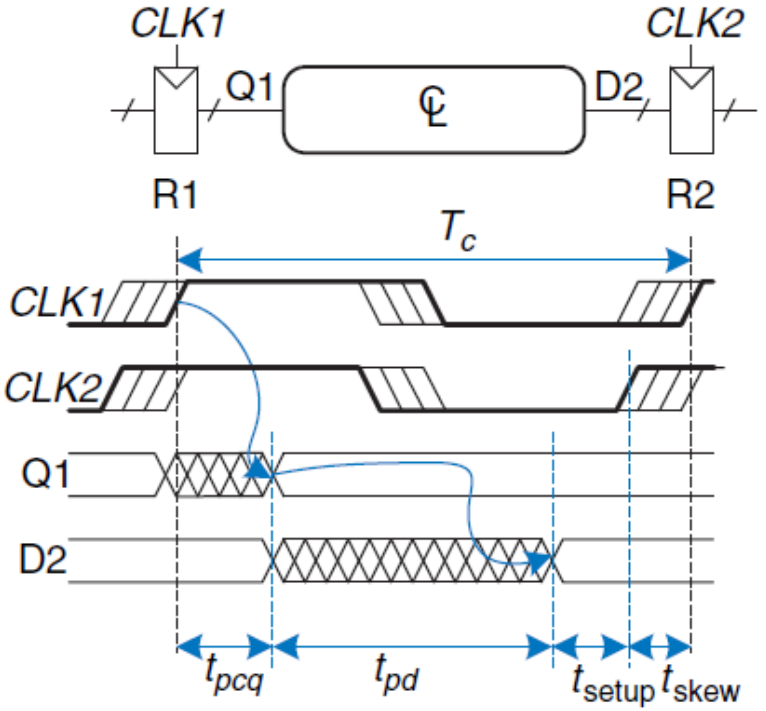
thold < tccq + tcd

Χρόνος hold< Χρόνος μόλυνσης καταχωρητή+
+χρόνος μόλυνσης συνδυαστικού



Sequential - Timing



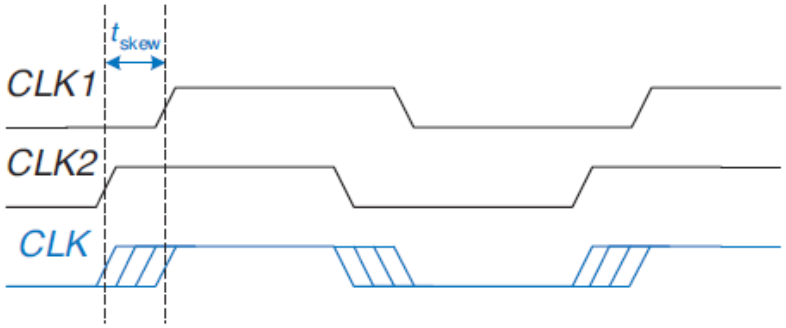


Tc ≥ tpcq + tpd + tsetup + tskew

Περίοδος ρολογιού> χρόνος για πάρει τη σωστή τιμή η έξοδος του Reg
+χρόνος διάδοσης συνδυαστικού
+χρόνος setup
+χρόνος skew

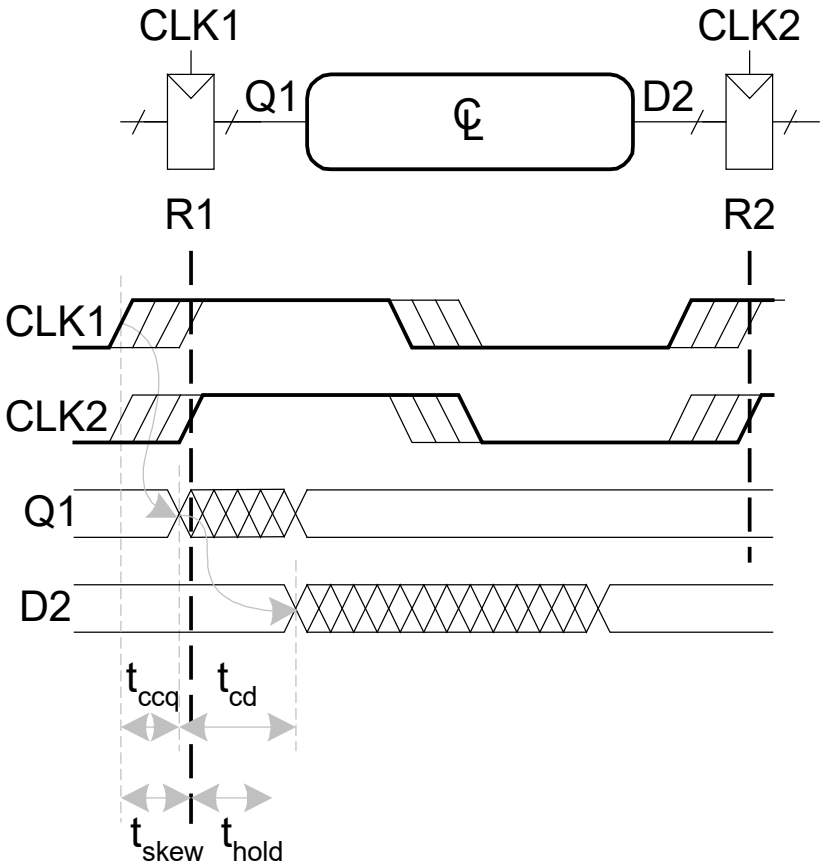
Κονσόλα (tcl): report_timing_summary -datasheet

Sequential - Timing



thold + tskew < tccq + tcd

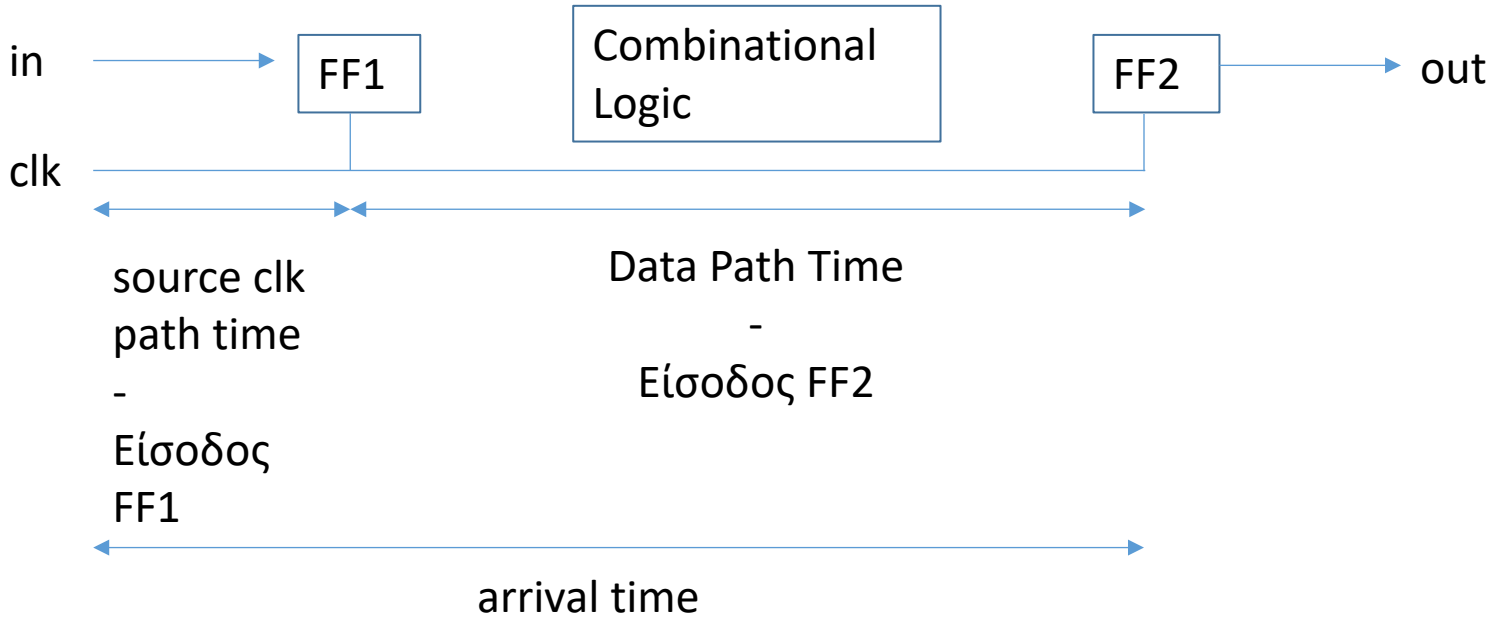
report_timing_summary -datasheet



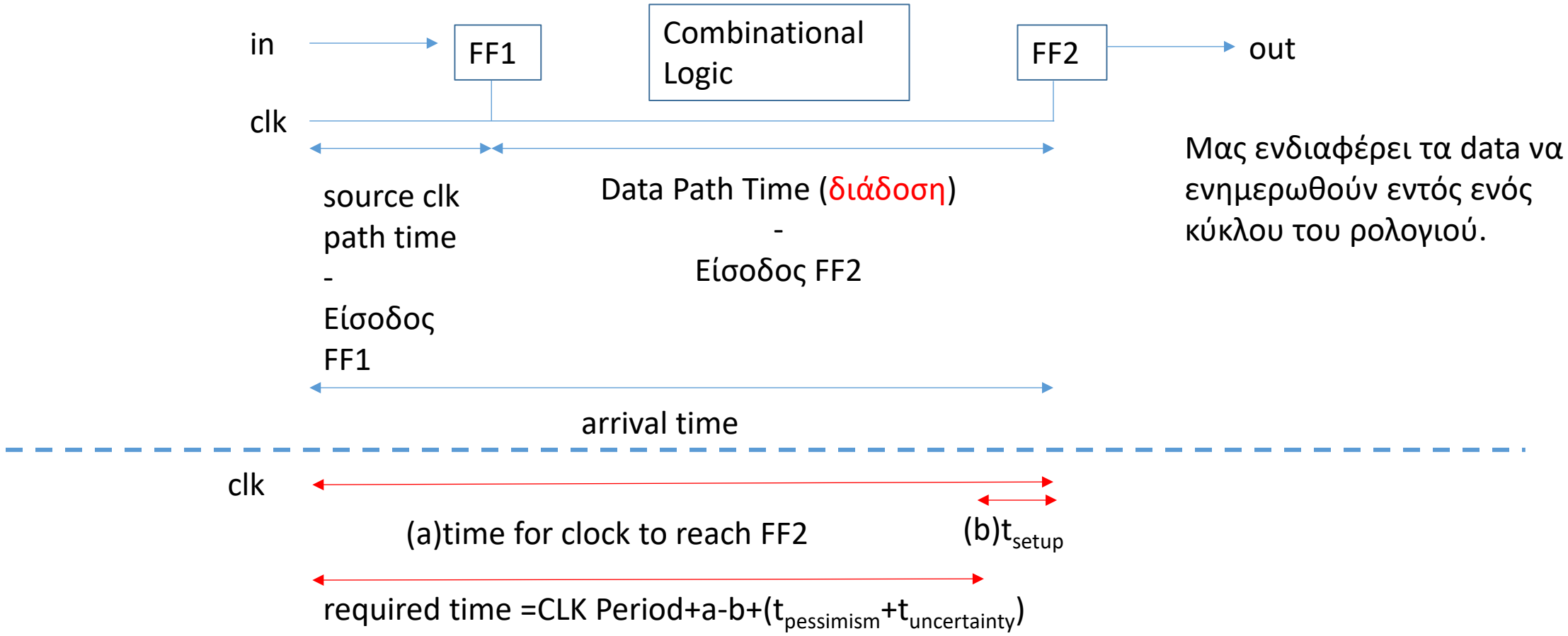
Sequential – Timing – Vivado analysis

Συνδυαστικό: Μόνο data path delay

Ακολουθιακό: Clock delay, path delay

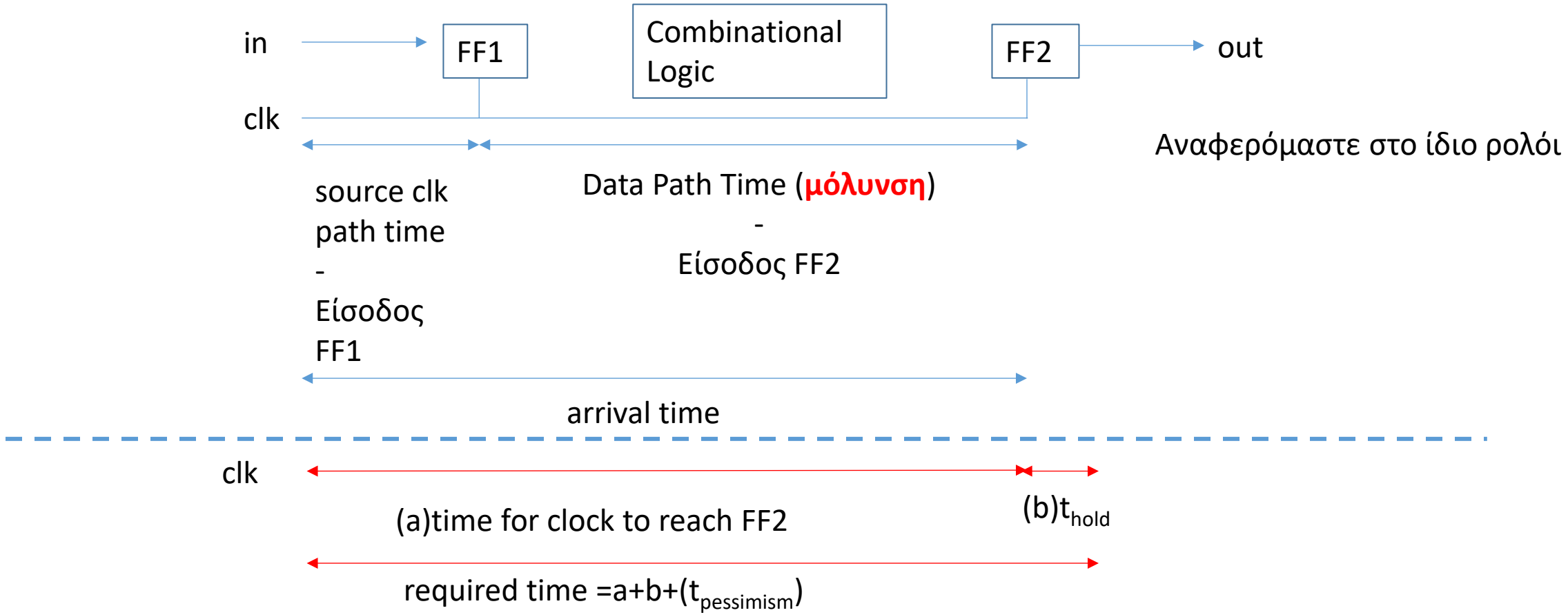


Sequential – Timing – Vivado Analysis



Μας ενδιαφέρει τα data να ενημερωθούν εντός ενός κύκλου του ρολογιού.

Sequential – Timing – Vivado Analysis



Διακόπτες και αποκλυδωνισμός

Οι διακόπτες (switches) και οι πιεστικοί διακόπτες (push-buttons) υποφέρουν από κλυδωνισμό επαφής
– Χρειάζονται περίπου 10ms να σταθεροποιηθούν
– Χρειάζεται να αποκλυδωνίσουμε (debounce) για να αποφύγουμε τις ψεύτικες ενεργοποιήσεις
• Χρησιμοποιούμε ένα διακόπτη μίας θέσης
• Κάνουμε δειγματοληψία στην είσοδο σε διαστήματα μεγαλύτερα από το χρόνο κλυδωνισμού (>10ms)
Στην κάρτα του εργαστηρίου σημαίνει 100000 χτύπους του ρολογιού.
• *Αναζητούμε δύο διαδοχικά δείγματα με την ίδια τιμή*

VHDL - Αποκλυδωνισμός

Παράδειγμα (1/2)

```
library ieee; use ieee.std_logic_1164.all;
entity debouncer is
port ( clk, reset : in std_logic; -- clk frequency = 100MHz
       pb : in std_logic;
       pb_debounced : out std_logic );
end entity debouncer;

architecture rtl of debouncer is

signal count1000000 : integer range 0 to 999999;
signal clk_100Hz : std_logic;    --10ms
signal pb_sampled : std_logic:='0';

begin
```

VHDL - Αποκλυδωνισμός

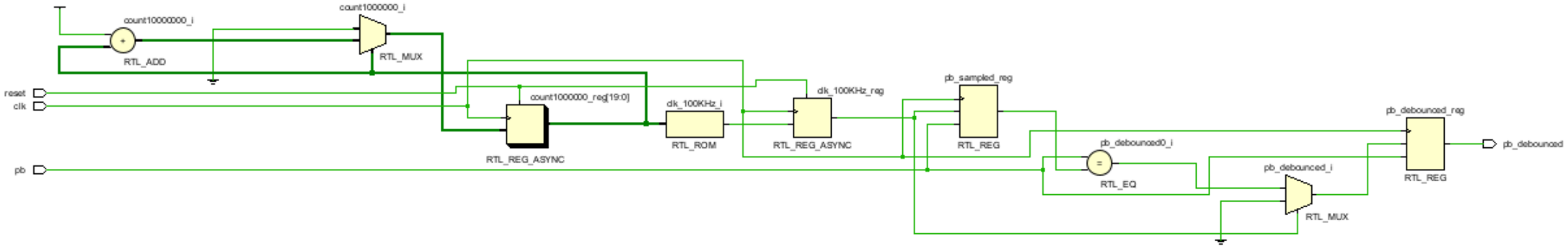
Παράδειγμα (2/2)

```
div_100Hz : process (clk, reset) is
begin
if reset = '1' then
        clk_100Hz <= '0';
        count1000000 <= 0;
elsif rising_edge(clk) then
        if count1000000 = 999999 then
             count1000000 <= 0;
             clk_100Hz <= '1';
        else
             count1000000 <= count1000000 + 1;
             clk_100Hz <= '0';
        end if;
end if;
end process div_100Hz;
```

```
debounce_pb : process (clk) is
begin
if rising_edge(clk) then
        if clk_100Hz = '1' then
                if pb = pb_sampled then
                        pb_debounced <= pb;
                end if;
                pb_sampled <= pb;
        end if;
end if;

end process debounce_pb;

end architecture rtl;
```

VHDL - Αποκλυδωνισμός

Παράδειγμα (3/3)



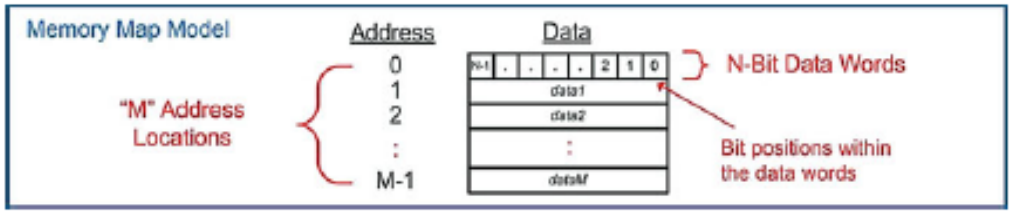
Μνήμες

Γενικό Μοντέλο μνήμης

Η μνήμη έχει μέγεθος, όσο το πλήθος των θέσεων μνήμης
Κάθε θέση μνήμη έχει μια διεύθυνση.
Μια μνήμη μεγέθους Μ, έχει θέσεις διευθύνσεων από 0 έως και Μ-1
Κάθε θέση μνήμης είναι μια λέξη (word) μεγέθους N-bit. Κάθε λέξη μπορεί να έχει μέγεθος πολλαπλάσιο του byte (άρα να είναι 1,2,3 … N byte).

Άρα αν έχουμε λέξεις των 2 byte τότε μια μνήμη 8 θέσεων έχει μέγεθος, 8 λέξεις ή 16 byte ή 128 bit.

Όποτε αλλάζουμε την τιμή μιας θέσης μνήμης κάνουμε εγγραφή (write) και όποτε ανακτούμε την τιμή μιας θέσης μνήμης έχουμε διάβασμα (read)



ROM: Read only memory
RAM: Random access memory

Μνήμες

ROM (1/2)

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL; use IEEE.NUMERIC_STD.ALL;
entity rom_16x16 is
   Port (
      addr  : in  std_logic_vector(3 downto 0);    -- 4-bit address for 16 words
      data  : out std_logic_vector(15 downto 0)  -- 16-bit output
   );
end entity rom_16x16;
architecture rtl of rom_16x16 is
type rom_array is array (0 to 15) of std_logic_vector(15 downto 0);
constant ROM : rom_array := ( 0  => x"1234", 1  => x"5678", …, 15 => x“123C"    );
begin
   data <= ROM(to_integer(unsigned(addr)));
end architecture rtl;
```

Μνήμες

ROM (2/2)

Αν η μνήμη έχει **latency** 5ns και θέλουμε να μοντελοποιήσουμε αυτή την καθυστέρηση, τότε:

data_out <=ROM(to_integer(unsigned(address))) **after 5 ns**;

Επηρεάζει μόνο το design (RTL) και όχι την σύνθεση/υλοποίηση.

Σύγχρονη ROM

```
process(clk)
  begin
    if rising_edge(clk) then
       data_reg <= ROM(to_integer(unsigned(addr)));
    end if;
  end process;
  data_out <= data_reg;
```

Το σήμα clk θα είναι στα port με mode in

Μνήμες

RAM

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ram_16x16_sync is
    Port (
        clk   : in  std_logic;
        we    : in  std_logic;
        addr  : in  std_logic_vector(3 downto 0);
        din   : in  std_logic_vector(15 downto 0);
        dout  : out std_logic_vector(15 downto 0)
    );
end entity ram_16x16_sync;
architecture rtl of ram_16x16_sync is
    type ram_array is array (0 to 15) of std_logic_vector(15 downto 0);
    signal RAM : ram_array := (others => (others => '0'));
    signal data_reg : std_logic_vector(15 downto 0);
begin

    process(clk)
    begin
        if rising_edge(clk) then
            if we = '1' then
                RAM(to_integer(unsigned(addr))) <= din;
            end if;
            data_reg <= RAM(to_integer(unsigned(addr)));
        end if;
    end process;

    dout <= data_reg;

end architecture rtl;
```

Σύγχρονη RAM με WE

Περίληψη

- Διαβάζετε τις παραγράφους 4.3, 4.4, 5.2.1, 5.2.2, 5.2.5 (θεωρία και VHDL) από Ashenden
- Διαβάζετε τις παραγράφους 3.4, 4.6, 5.5.1-5.5.3, 5.5.6-5.5.8 (ΟΧΙ το κομμάτι της VERILOG) από το βιβλίο των Harris.
- Διαβάζετε τις παραγράφους 6.1-6.3, 7.8.4, 12.10.10-12.10.11 από το βιβλίο των Brown-Vranesic.

Επίσης οι σύνδεσμοι:
https://stackoverflow.com/questions/21351273/is-it-possible-to-synthesize-vhdl-code-with-variable-in-it
Assert και Report: https://insights.sigasi.com/tech/vhdl-assert-and-report/