

Εργαστήριο Space Data Systems

VHDL

-

FSM - Timing

Βασιλόπουλος Διονύσης

ΕΔΙΠ Τμήματος Πληροφορικής & Τηλεπικοινωνιών - ΕΚΠΑ

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**

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

Scalar
type

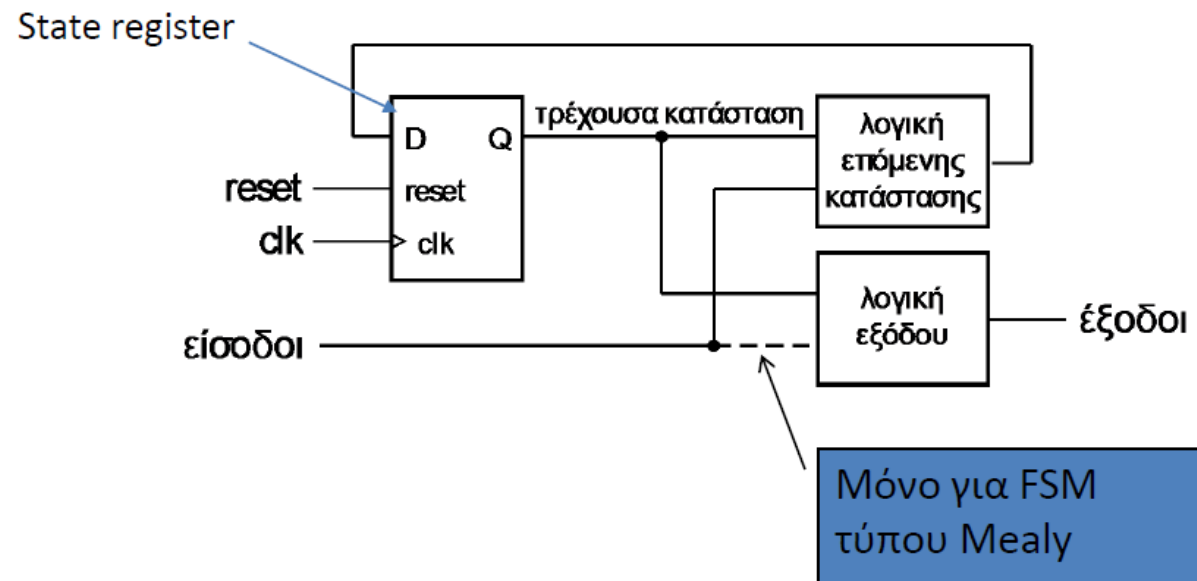
https://peterfab.com/ref/vhdl/vhdl_renerta/mobile/source/vhd00007.htm

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: s_0 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
 - N καταστάσεις: χρειάζονται τουλάχιστον $\log_2 N$ 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

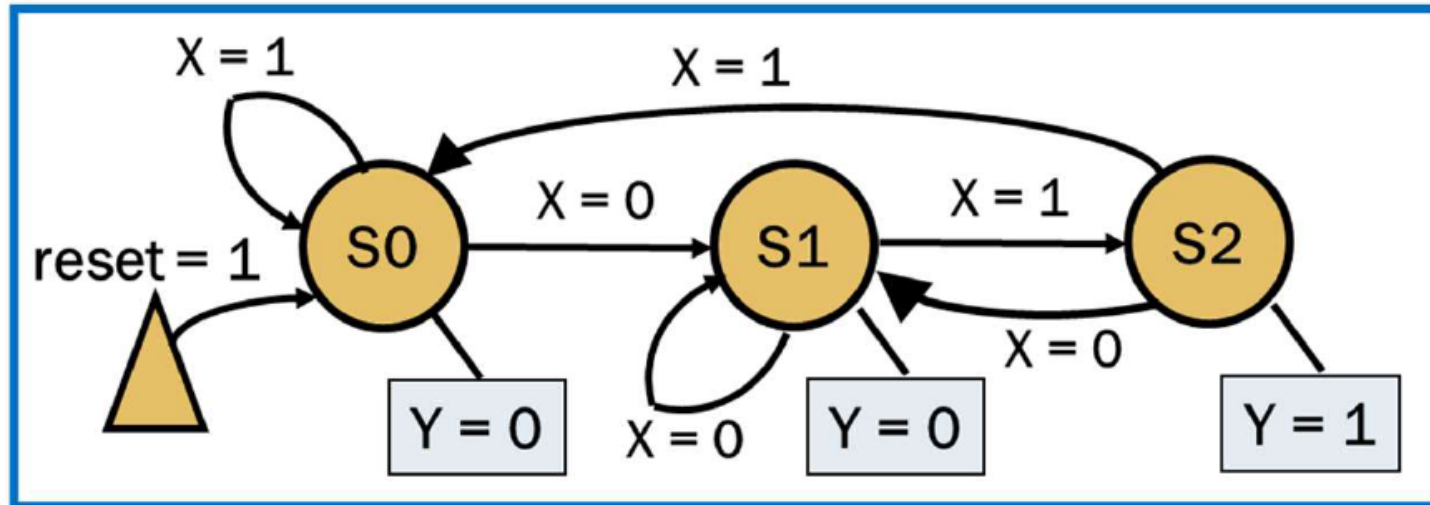
Κωδικοποίηση Καταστάσεων - Type

- Χρησιμοποιούμε έναν τύπο απαρίθμησης για τις τιμές των καταστάσεων
 - αφηρημένο, έτσι αποφεύγουμε να προδιαγράψουμε την κωδικοποίηση

```
type state_type is (zero, edge, one);  
signal state_reg, state_next: state_type;  
...
```

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
```

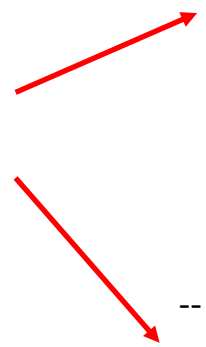
```
-- state definition
```

```
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;

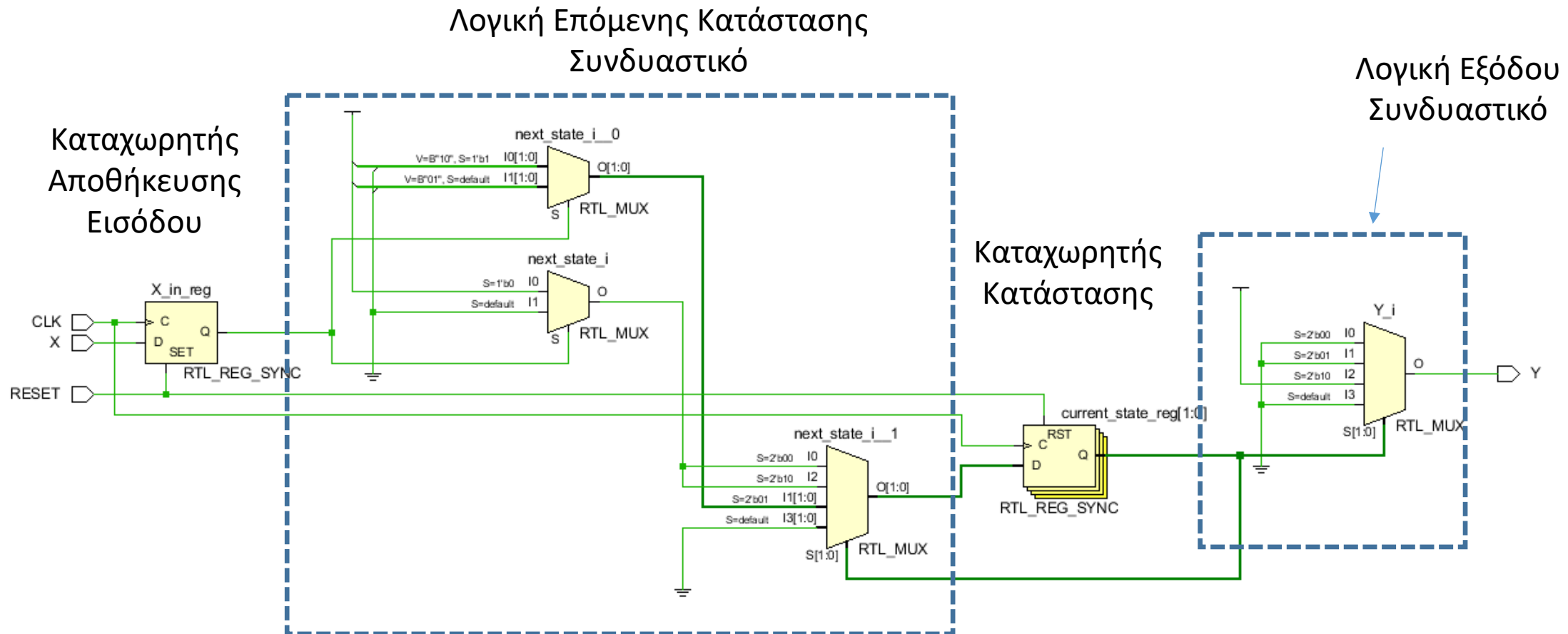
ΛΟΓΙΚΗ για
ΕΞΟΔΟ

ΛΟΓΙΚΗ για
ΕΠΟΜΕΝΗ ΚΑΤΑΣΤΑΣΗ

Αρχική τιμή
οι τιμές
Reset

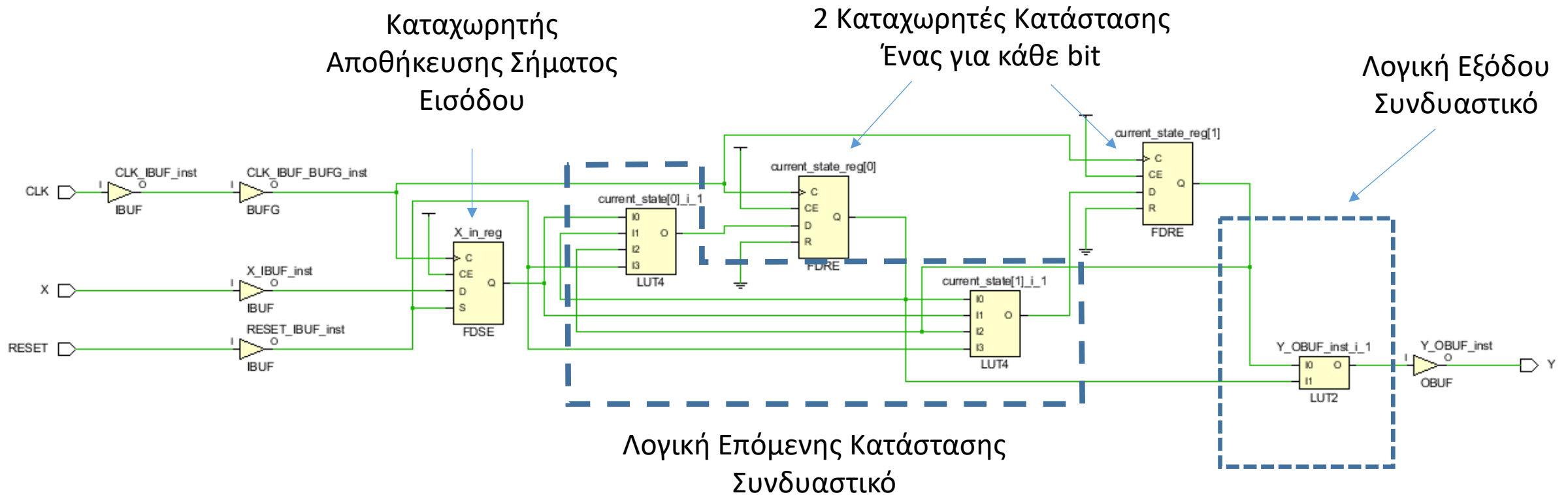
VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore - RTL



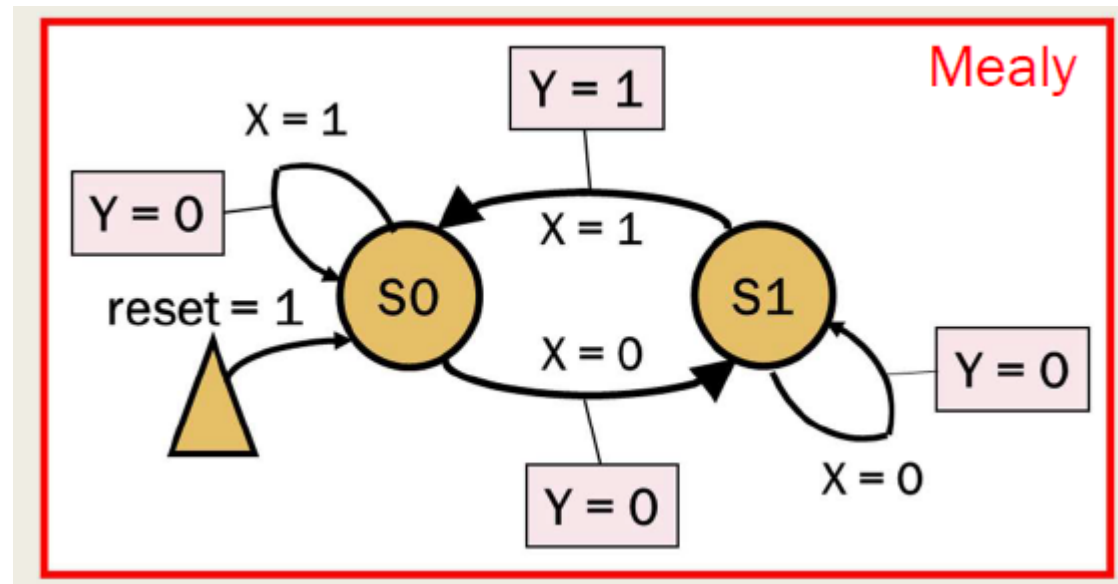
VHDL – Finite State Machines

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 synchronization  
  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;
  end case;
  -- fail-safe behavior
  when others => next_state <= S0;
end case;
end process;
end Behavioral;
```

**ΛΟΓΙΚΗ για
ΕΞΟΔΟ**

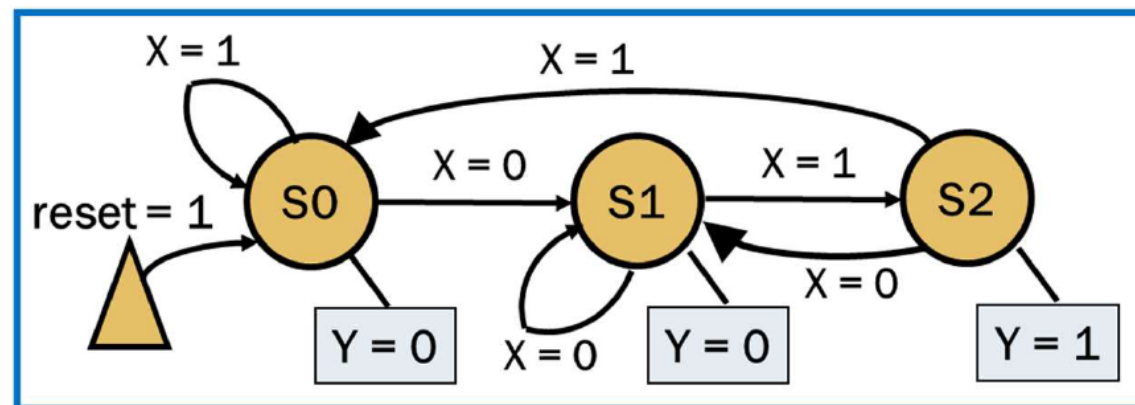


VHDL – Finite State Machines

Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

S_{old}	$S_{old}(1)$	$S_{old}(0)$	X_{in}	$S_{new}(1)$	$S_{new}(0)$	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

$S_{new}(1)$		
$S_{old} \backslash X_{in}$	0	1
00	0	0
01	0	1
11	0	1
10	0	0

$S_{new}(0)$		
$S_{old} \backslash X_{in}$	0	1
00	1	0
01	1	0
11	0	0
10	1	0

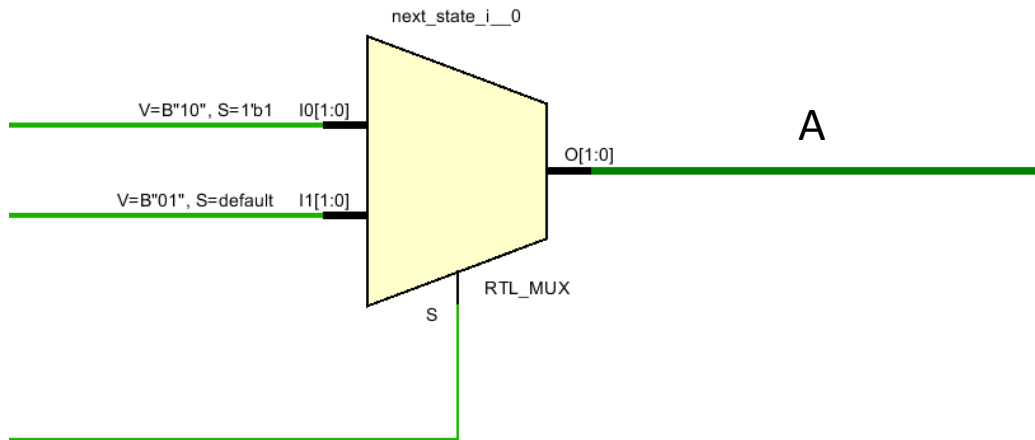
Y		
$S_{(1)} \backslash S_{(0)}$	0	1
0	0	0
1	1	0

$$\begin{aligned} S_{new}(1) &= S_{old}(0)X_{in} \\ S_{new}(0) &= S_{old}(1)'X_{in}' + S_{old}(0)'X_{in}' \\ Y &= S(1)S(0)' \end{aligned}$$

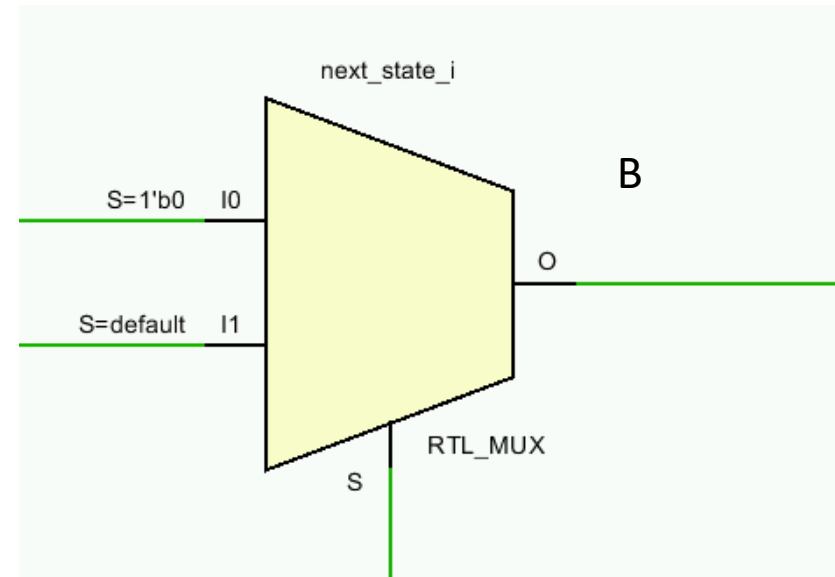
VHDL – Finite State Machines

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

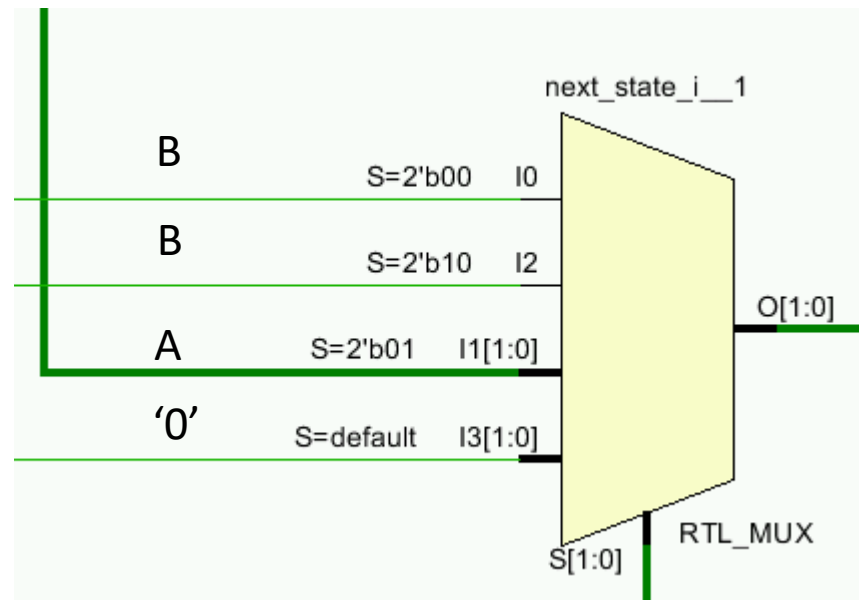
Παράδειγμα – Ανίχνευση ακολουθίας 01 – Moore – Πίνακας Αληθείας

$X_{in}=S$	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

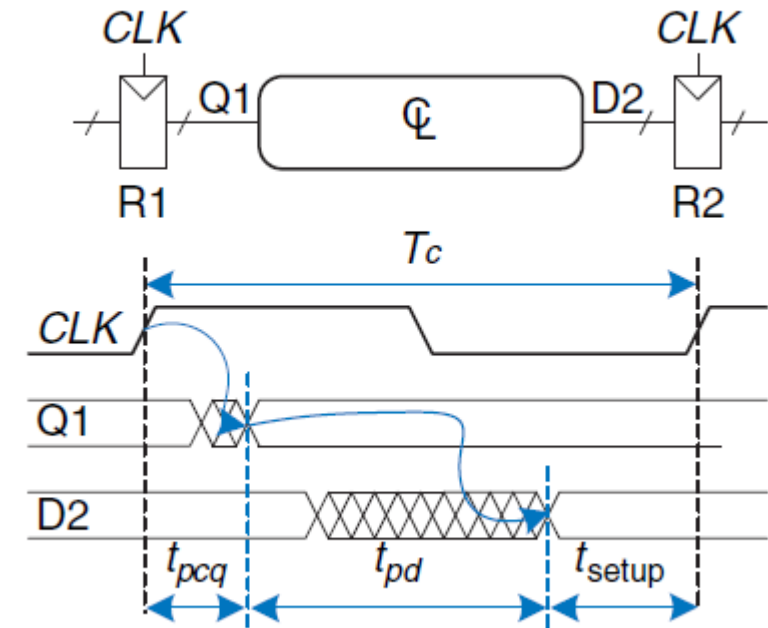


S_{old}	X_{in}	S_{new}
00(0)	0	S_1
01(1)	0	S_1
10(2)	0	S_1
11(X)	0	S_0
00(0)	1	S_0
01(1)	1	S_2
10(2)	1	S_0
11(X)	1	S_0

Sequential - Timing

$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

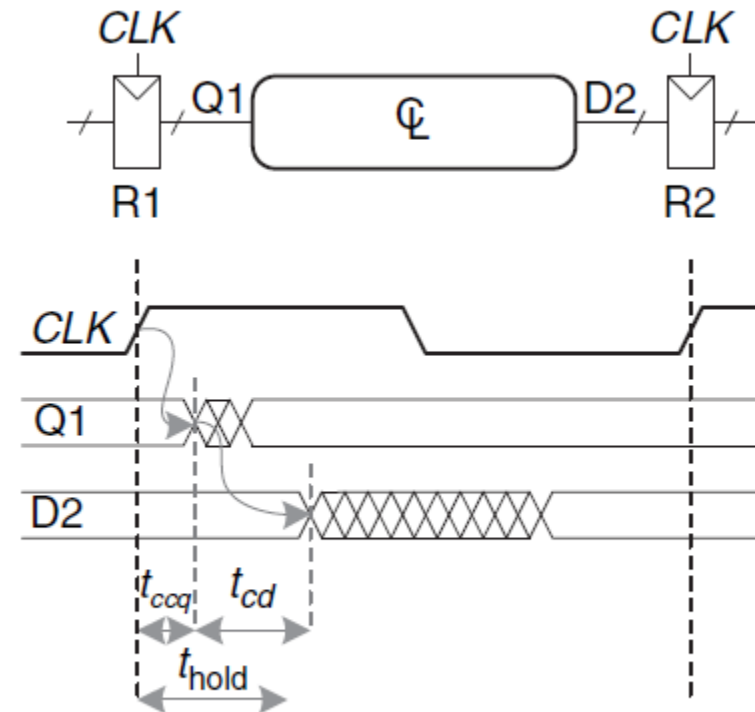
Περίοδος ρολογιού > χρόνος για πάρει τη σωστή τιμή η έξοδος του Reg
+χρόνος διάδοσης
+χρόνος setup



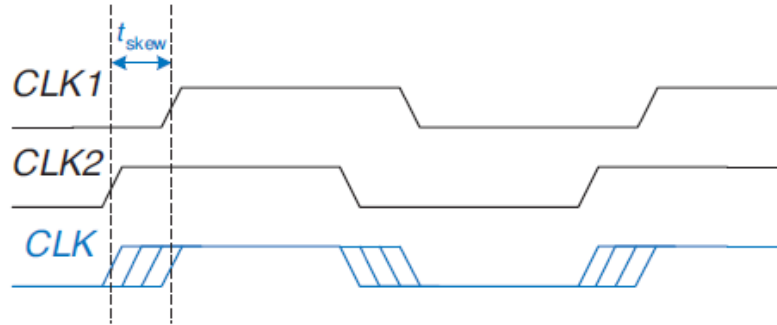
Sequential - Timing

$$t_{hold} < t_{ccq} + t_{cd}$$

Χρόνος hold < Χρόνος μόλυνσης καταχωρητή +
+χρόνος μόλυνσης συνδυαστικού



Sequential - Timing

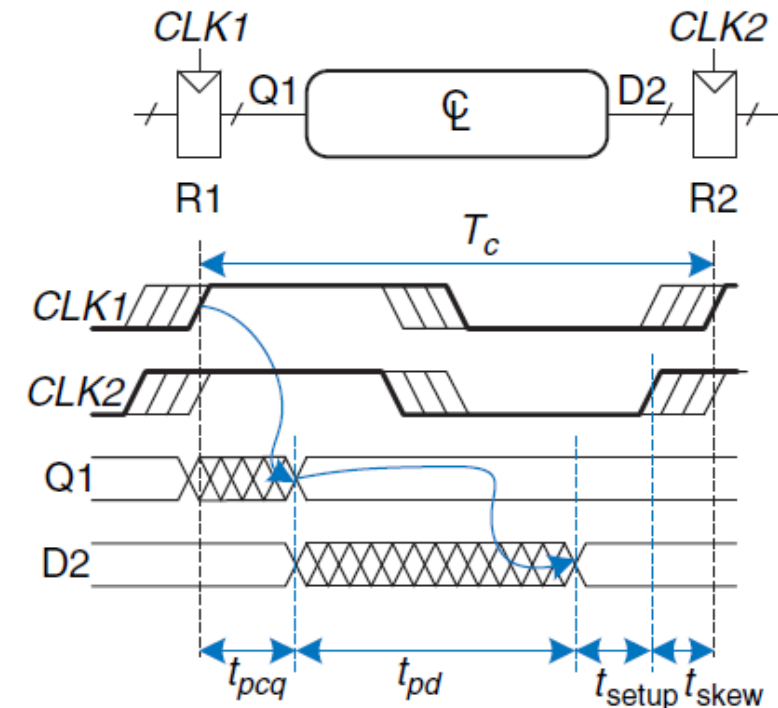


$$T_c \geq t_{pcq} + t_{pd} + t_{setup} + t_{skew}$$

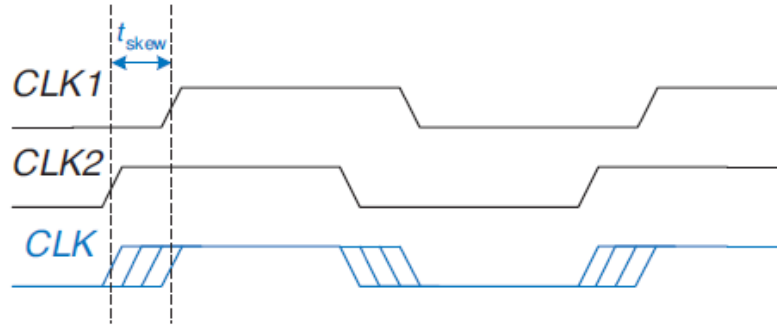
Περίοδος ρολογιού > χρόνος για πάρει τη σωστή τιμή η έξοδος του Reg
 +χρόνος διάδοσης συνδυαστικού
 +χρόνος setup
 +χρόνος skew

$$t_{hold} + t_{skew} < t_{ccq} + t_{cd}$$

Κονσόλα (tcl): report_timing_summary -datasheet

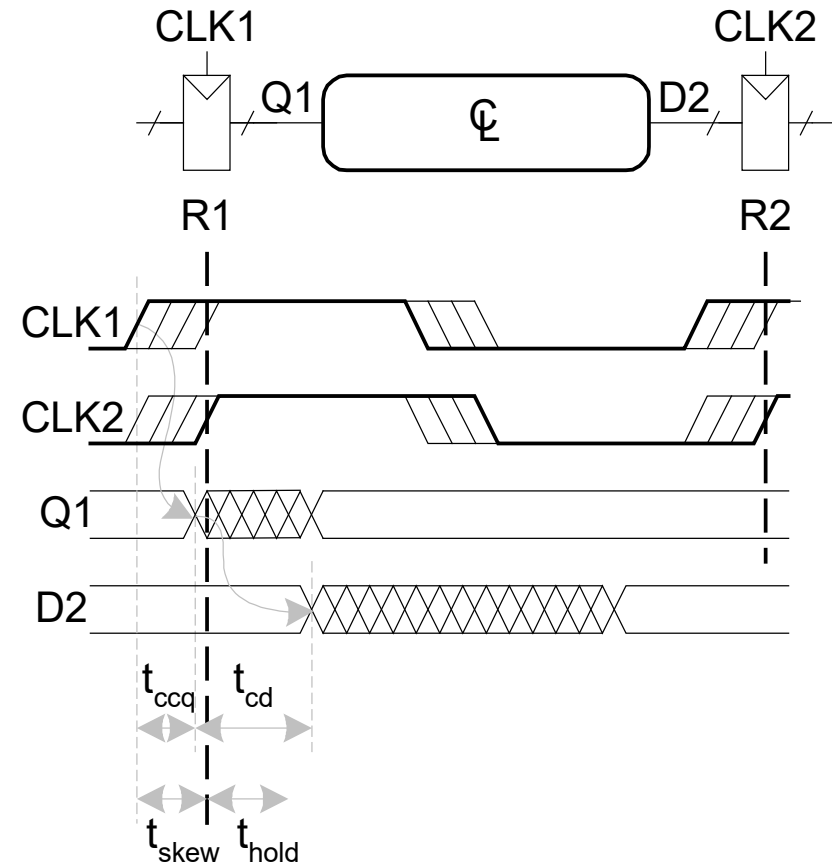


Sequential - Timing



$$t_{hold} + t_{skew} < t_{ccq} + t_{cd}$$

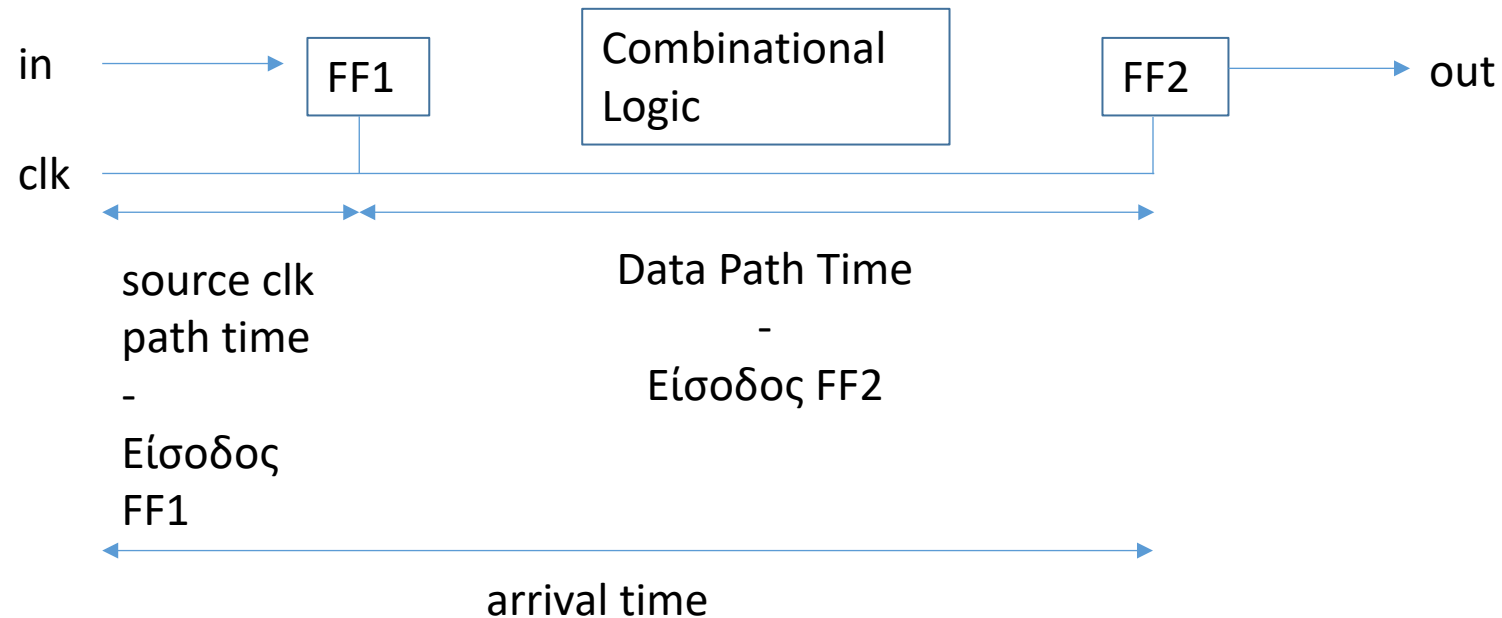
report_timing_summary -datasheet



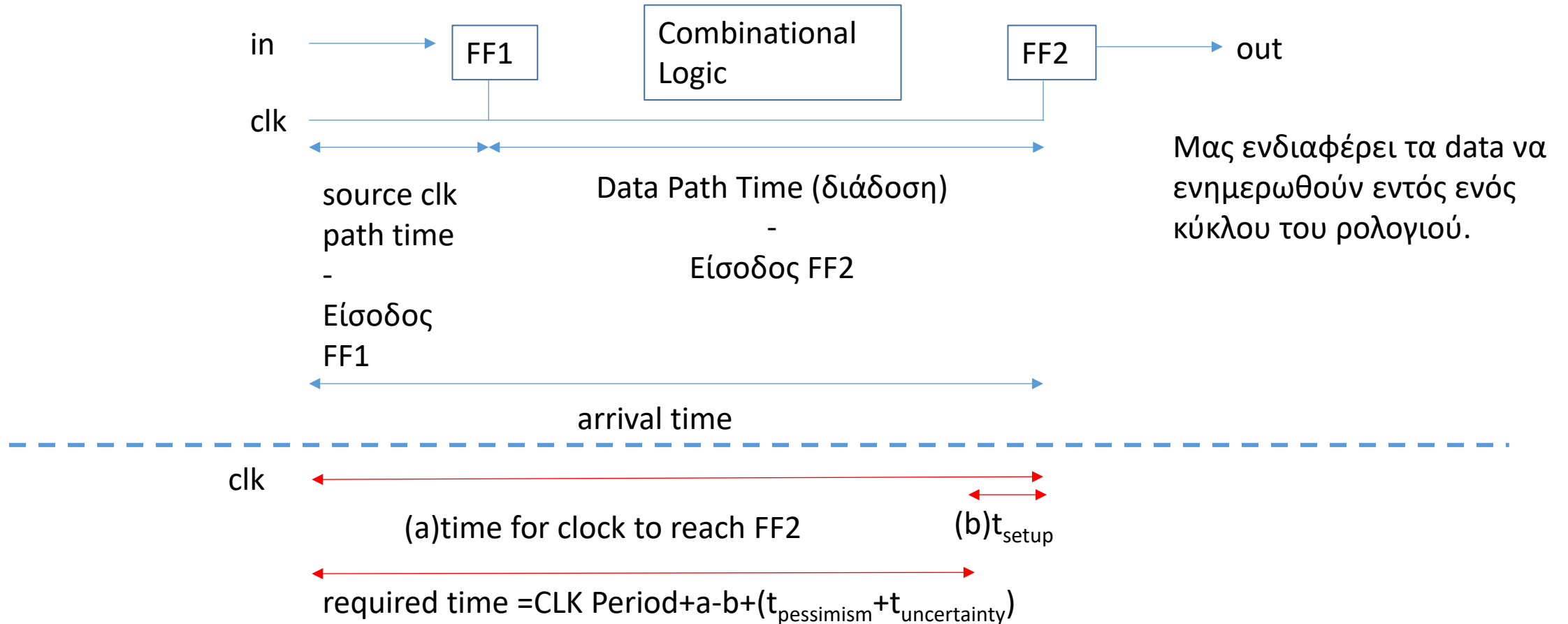
Sequential – Timing – Vivado analysis

Συνδυαστικό: Μόνο data path delay

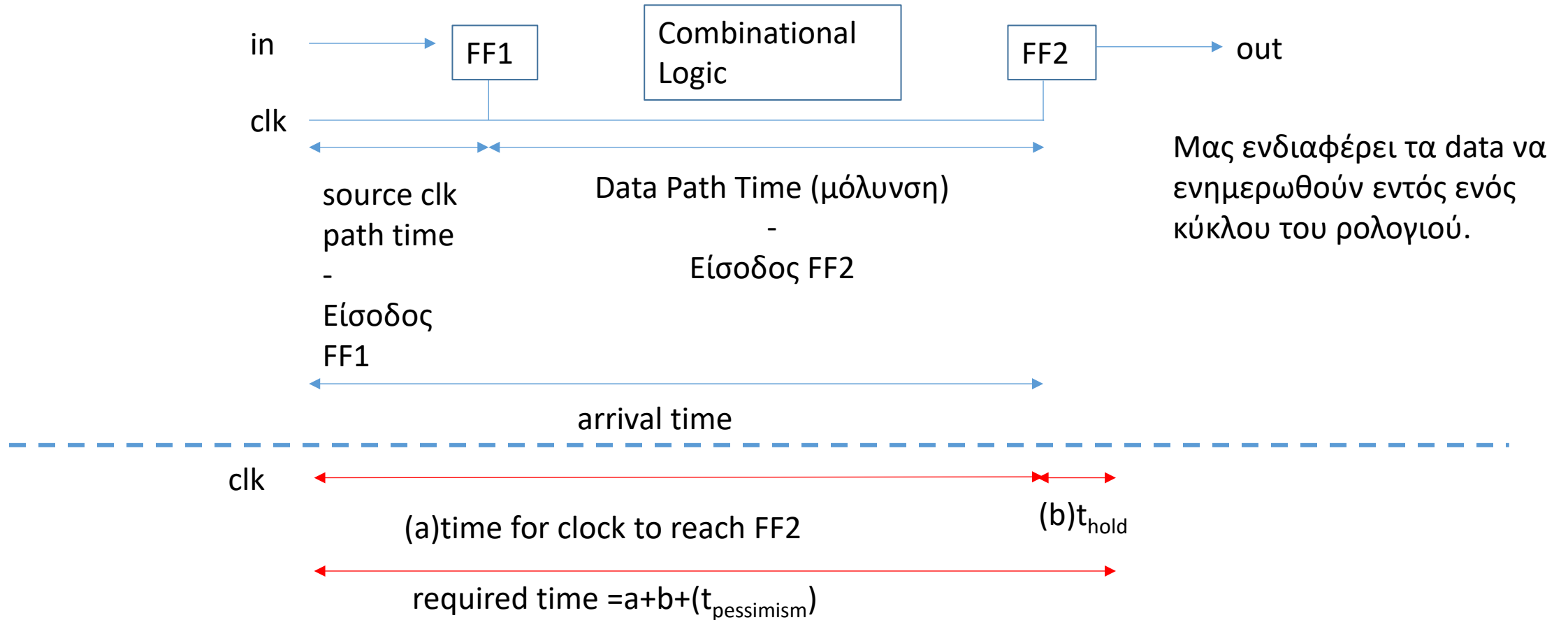
Ακολουθιακό: Clock delay, path delay



Sequential – Timing – Vivado Analysis



Sequential – Timing – Vivado Analysis



Περίληψη

Harris: Παράγραφοι 3.4, 3.5 και 4.6

Ashenden: Παράγραφοι 4.3 και 4.4

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/>