ΗΜΥ-210: Σχεδιασμός Ψηφιακών Συστημάτων

Χειμερινό Εξάμηνο 2009

VHDL για Σχεδιασμό Ακολουθιακών Κυκλωμάτων

Διδάσκουσα: Μαρία Κ. Μιχαήλ

Πανεπιστήμιο Κύπρου
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

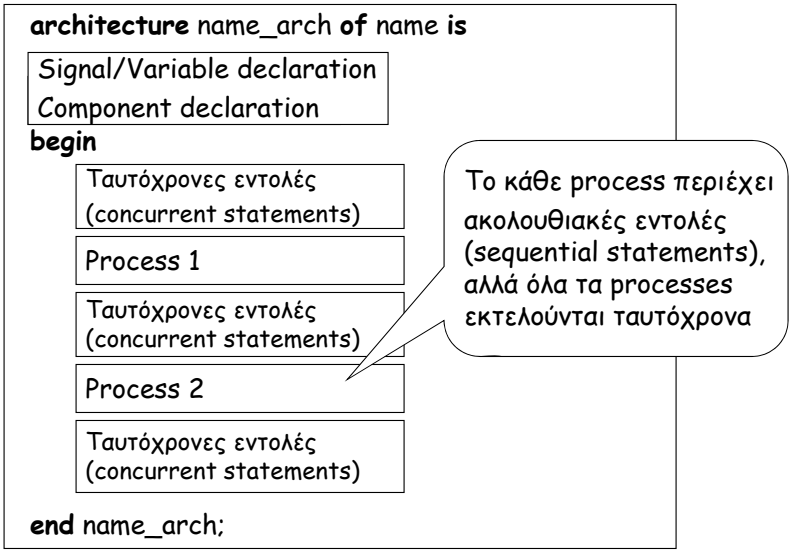
Περίληψη

- VHDL Processes
- Εντολές If-Then-Else και CASE
- Περιγραφή Flip-Flop με VHDL
- Περιγραφή Ακολουθιακού Κυκλώματος (βάσει πίνακα/διαγράμματος καταστάσεων) με VHDL
- Συγχρονισμός μεταξύ Processes

VHDL Process

- Ένα σύνολο από εντολές VHDL το οποίο «εκτελείται» όταν ένα signal (από ένα συγκεκριμένο σύνολο) αλλάζει τιμή.
- Ο «κορμός» ("body") ενός process υλοποιεί ένα *σειριακό* (*sequential*) πρόγραμμα, δηλ. οι τιμές των *signals* ανανεώνονται μόνο όταν ολοκληρωθεί η εκτέλεση του προγράμματος.
- Μπορεί επίσης να χρησιμοποιήσει *variables*, των οποίων η τιμή ανανεώνεται αμέσως.

Αρχιτεκτονική VHDL



VHDL Process

```
P1: process (<sensitivity list>)
<variable declarations>
begin
    <sequential statements>
end process P1;
```

Προαιρετική σήμανση

Μέσα σε ένα process:

- Ανάθεση μεταβλητών (variables) με **:=** και **άμεση** ενημέρωση.
- Ανάθεση σημάτων (signals) με <= και η ενημέρωση γίνετε στο **τέλος του process**.

Signals Vs Variables σε ένα Process

Θεωρείστε ότι A, B, και C είναι ακέραιοι με A=1, B=5, και C=10.

A, B, C: ***signals***	A, B, C: ***variables***
begin process … B <= A; C <= B; … end process;	begin process … B := A; C := B; … end process;
B = 1 και C = 5 (χρησιμοποιεί την αρχική τιμή του B (=5) όταν υπολογίζει το C)	B = 1 και C = 1 (χρησιμοποιεί την νέα τιμή του B (=1) όταν υπολογίζει το C)

Εντολή If-Then-Else

```
[if_label:]
if boolean_expression then
      { sequential_statement; }
{ elsif boolean_expression then
      { sequential_statement; } }
[ else
      { sequential_statement; } ]
end if  [ if_label ];
```

Σύμβαση:
[ ] -- προαιρετικό
{ } – επανάληψη δυνατή

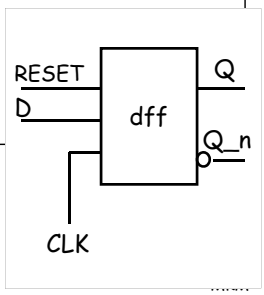
Εντολή CASE

```
[case_label:]
case expression is
 { when choice =>
     { sequential statement; } }
 [ when others =>
     { sequential statement; } ]
end case [case_label];
```

Περιγραφή Flip-Flop με VHDL:
Θετικά Ακμοπυροδοτούμενο D-FF με Ασύγχρονη Αρχικοποίηση

- Δήλωση Οντότητας:

```
-- Positive Edge-Triggered D Flip-Flop with Reset:
-- VHDL Process Description
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port(CLK, RESET, D: in std_logic;
       Q, Q_n: out std_logic);
end dff;
```

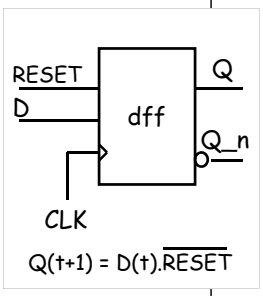


Περιγραφή Flip-Flop με VHDL:
Θετικά Ακμοπυροδοτούμενο D-FF με Ασύγχρονη Αρχικοποίηση

- Αρχιτεκτονική:

```
architecture pet_pr of dff is
-- Implements positive edge-triggered bit state storage
-- with asynchronous reset.
signal state: std_logic;
begin
   Q <= state;
   Q_n <= not state;
   process (CLK, RESET)
   begin
     if (RESET = '1') then
       state <= '0';
     else
        if (CLK'event and CLK = '1') then
           state <= D;
        end if;
     end if;
   end process;
end pet_pr;
```

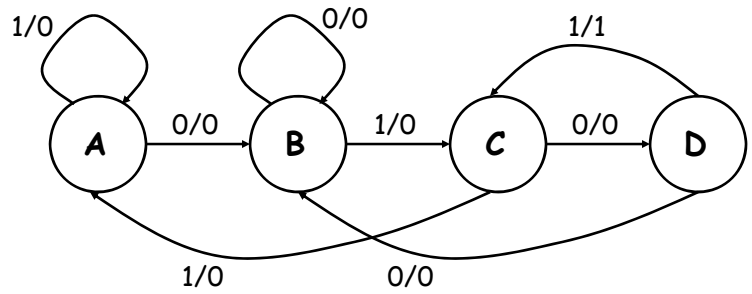
Καθορίζει τον τύπο πυροδότησης του FF



Ακολουθιακό Κύκλωμα σε VHDL Ανιχνευτής Ακολουθίας

- Θυμηθείτε το παράδειγμα του ανιχνευτή της ακολουθίας "0101", με μία είσοδο (X) και μία έξοδο (Z).

Διάγραμμα Καταστάσεων:



Ανιχνευτής Ακολουθίας σε VHDL (συν.)

- Χρησιμοποιούμε 3 ξεχωριστά processes, τα οποία εκτελούνται παράλληλα.
- Ο συγχρονισμός μεταξύ των διαφόρων processes επιτυγχάνεται από την αλληλοεπίδραση κοινών signals.
- 3 processes:
  - Στοιχεία Μνήμης (storage cct) – `state_register`
  - Λογική Εισόδων FFs (next state logic) – `next_state_func`
  - Λογική Εξόδων (primary output logic) – `output_func`

Ανιχνευτής Ακολουθίας σε VHDL (συν.)

- Ορίζουμε ένα scalar enumeration `type` για αναπαράσταση των $4^{ων}$ καταστάσεων:
  - **type** state_type **is** (A, B, C, D);
- Ακολούθως, δηλώνουμε signals ή variables για την παρούσα και την επόμενη κατάσταση:
  - **signal** state, next_state : state_type;
- `state` και `next_state` μπορούν να πάρουν μόνο την τιμή `A`, `B`, `C`, ή `D`. Προσπάθεια ανάθεσης οποιασδήποτε άλλης τιμής θα δώσει λάθος στην συμβολομετάφραση.

Δήλωση Οντότητας & Αρχιτεκτονικής

```
-- Sequence Recognizer: VHDL Process Description

library ieee;
use ieee.std_logic_1164.all;
entity seq_rec is
   port(CLK, RESET, X: in std_logic;
        Z: out std_logic);
end seq_rec;

architecture process_3 of seq_rec is
   type state_type is (A, B, C, D);
   signal state, next_state : state_type;
begin

…

end;
```

Process για Στοιχεία Μνήμης (State Register)

```
-- Process 1 - state_register: implements positive edge-triggered
-- state storage with asynchronous reset.
   state_register: process (CLK, RESET)
   begin
     if (RESET = '1') then
        state <= A;
     else
        if (CLK'event and ClK = '1') then
           state <= next_state;
        end if;
     end if;
   end process;
```

- Πόσα FFs; Εξαρτάται από τον ***αριθμό*** των τιμών που μπορούν να πάρουν τα signals `state` & `next_state`!
  Για αυτή την περίπτωση, υπάρχουν 4 δυνατές καταστάσεις (`A, B, C, D`) και, επομένως, θα χρησιμοποιηθούν 2 FFs.

Process για Συναρτήσεις Επόμενης Κατάστασης (Next State Functions)

```
-- Process 2 - next_state_function: implements
-- next state as function of input X and state.
   next_state_func: process (X, state)
   begin
      case state is
        when A =>
            if X = '1' then
                next_state <= B;
            else
                next_state <= A;
            end if;
        when B =>
             if X = '1' then
                next_state <= C;
            else
                next_state <= A;
            end if;
        when C =>
            if X = '1' then
                next_state <= C;
            else
                next_state <= D;
            end if;
        when D =>
            if X = '1' then
               next_state <= B;
            else
               next_state <= A;
            end if;
      end case;
   end process;
```

Πίνακας Καταστάσεων

	next_state		Z	
state	X=1	X=0	X=1	X=0
A	B	A	0	0
B	C	A	0	0
C	C	D	0	0
D	B	A	1	0

Process για Συνάρτηση Εξόδου (Output State Function)

```
-- Process 3 - output_function:
-- implements output as function of
-- input X and state.
   output_func: process (X, state)
   begin
      case state is
         when A =>
              Z <= '0';
         when B =>
              Z <= '0';
         when C =>
              Z <= '0';
         when D =>
              if X = '1' then
                 Z <= '1';
              else
                 Z <= '0';
              end if;
      end case;
   end process;
```

Υπονοεί μοντέλο Mealy

Πίνακας Καταστάσεων

	next_state		Z	
state	X=1	X=0	X=1	X=0
A	B	A	0	0
B	C	A	0	0
C	C	D	0	0
D	B	A	1	0

Συγχρονισμός μεταξύ των processes

- state_register: **process** (CLK, RESET)
  - state ανανεώνεται βάσει των CLK και RESET
- next_state_func: **process** (X, state)
  - next_state ανανεώνεται βάσει των X και state
- output_func: **process** (X, state)
  - Z ανανεώνεται βάσει των X και state

Συγχρονισμός (συν.)

- Θεωρείστε την ακόλουθη περίπτωση:

χρόνος t_0	χρόνος t_1	χρόνος t_2
state = D	X=0	X=1
X=1	⇩	⇩
Z=1	next_state = A	next_state = ?
RESET=0	Z=0	Z=?
CLK=0	RESET=0	RESET=0
	CLK=0	CLK=0

Χρησιμοποιεί τις τιμές των state και X στον χρόνο t_0
για να υπολογίσει το next_state

Συγχρονισμός (συν.)

- Θεωρείστε την ακόλουθη περίπτωση :

χρόνος t_0	χρόνος t_1	χρόνος t_2
state = D	X=0	X=1
X=1	⇩	⇩
Z=1	next_state = A	next_state = A
RESET=0	Z=0	Z=0
CLK=0	RESET=0	RESET=0
	CLK=0	CLK=0

Χρησιμοποιεί τις τιμές των state και X στον χρόνο t_0
για να υπολογίσει το next_state αφού CLK $\neq \uparrow$