Κεφάλαιο 4ο
Σχεδίαση Κυκλωμάτων με χρήση της γλώσσας VHDL

4.1 Εισαγωγή στη VHDL

4.1.1 Θεωρητικό υπόβαθρο

Η VHDL είναι μια γλώσσα που χρησιμοποιείται για την **περιγραφή και μοντελοποίηση** ψηφιακών κυκλωμάτων. Αρχικοποιήθηκε από το DoD (Department of Defense-USA) στις αρχές του 1980.
Πεδία Εφαρμογής της VHDL

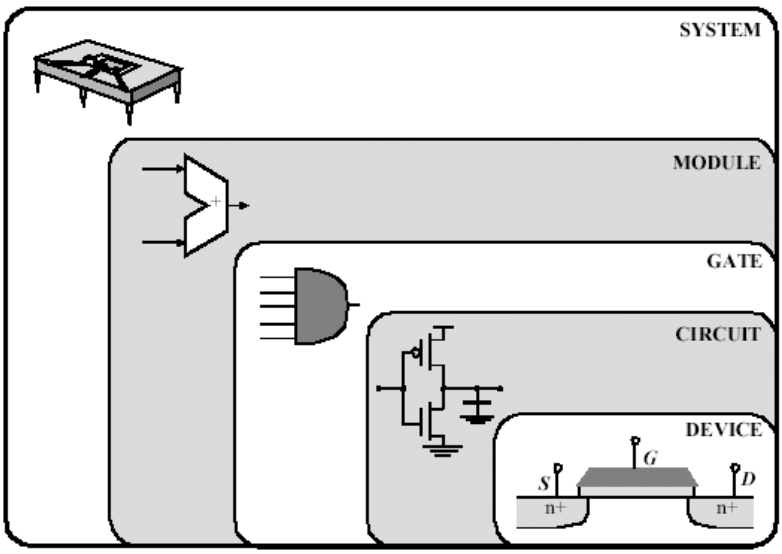
- Εξομοίωση ορθής λειτουργίας (Simulation)
- Σύνθεση ψηφιακών κυκλωμάτων (Synthesis)
- Επιβεβαίωση ορθού σχεδιασμού (DesignVerification)
- Μοντέλα προδιαγραφών (Specification Models)

Τα πλεονεκτήματα της γλώσσας VHDL:

- Παγκόσμιο πρότυπο (IEEE 1076-1987, 1076-1993)
- Υποστήριξη από πληθώρα αναπτυξιακών εμπορικών εργαλείων σχεδιασμού (CAD tools)
- Εύκολη μεταφορά κυκλωματικών περιγραφών σε διαφορετικά αναπτυξιακά περιβάλλοντα
- Δυνατότητα περιγραφής κυκλώματος / συστήματος σε διαφορετικά ιεραρχικά επίπεδα (από επίπεδο πύλης μέχρι επίπεδο συστήματος)
- Υποστήριξη εναλλακτικών σχεδιαστικών μεθοδολογιών (Top-down, Bottom-up, Mixed)
- Ιεραρχικήσχεδίαση (Block Diagrams, Components)
- Επαναχρησιμοποίηση σχεδιασθέντων υπομονάδων (reusable components)
- Χρήση βιβλιοθηκών με σχεδιασθέντα κυκλώματα
- Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος
- Περιγραφή κυκλώματος/συστήματος ανεξάρτητα από την τεχνολογία υλοποίησης
- Επαναχρησιμοποίηση υπάρχουσας κυκλωματικής περιγραφής σε διαφορετικές τεχνολογίες (FPGA xilinx, altera, ASICs)
- Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος / συστήματος
- Υποστήριξη συντρεχουσών και ακολουθιακών δομών (concurrent and sequential constructions). Οι περισσότερες γλώσσες (π.χ. C) υποστηρίζουν μόνο ακολουθιακές δομές. Οι συντρέχουσες δομές είναι απαραίτητες για την περιγραφή της λειτουργίας του υλικού.
- Εύκολη διαχείριση λαθών και επιβεβαίωση ορθής λειτουργίας (Simulation, Error Management, Design Verification)

Στη VHDL διακρίνονται τα εξής επίπεδα σχεδίασης:

- **Functional (system) level**(αρχιτεκτονική)
- **Behaviorallevel** (ανάθεση δομών και πόρων)
- **RTL (Structural) level**(επιλογή τεχνολογίας)
- **Logic (gate) level** + electrical specification
- **Electrical level** + layout requirements
- **Layout level**



Εικόνα 4.1: Τα επίπεδα σχεδίασης

Στο επίπεδο συστήματος προβλέπεται:

- Περιγραφή των προδιαγραφών του συστήματος
- Δεν απαιτείται πληροφορία χρονισμών
- Δεν απαιτείται ακριβής καθορισμός της αρχιτεκτονικής του κυκλώματος / συστήματος
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας

Στο επίπεδο συμπεριφοράς προβλέπεται:

- Αναλυτικότερη περιγραφή της συμπεριφοράς / λειτουργίας του κυκλώματος
- Καθορισμός των απαιτούμενων αλγορίθμων για την ικανοποίηση των προδιαγραφών του συστήματος
- Εμπεριέχει πληροφορίες χρονισμού
- Μη αναλυτικός καθορισμός της αρχιτεκτονικής του κυκλώματος (καταχωρητές, μνήμες, συνδυαστικά κυκλώματα κ.λ.π.)
- Ένα μοντέλο περιγραφής σε επίπεδο συμπεριφοράς αποτελείται από τα λειτουργικά στοιχεία και τη διασύνδεση αυτών
- Κάθε λειτουργικό στοιχείο μπορεί να εμπεριέχει περισσότερα του ενός στοιχεία και πληροφορία χρονισμού

Στα επίπεδα καταχωρητή και πύλης προβλέπεται:

- Επίπεδοκαταχωρητή (Register Transfer Level-RTL)
- Περιγραφή του κυκλώματος με χρήση συνδυαστικών κυκλωμάτων, καταχωρητών, μνημών, σύγχρονων και ασύγχρονων μηχανών πεπερασμένων καταστάσεων
- ΕπίπεδοΠύλης (Gate/Logic-Level)
- Περιγραφή του κυκλώματος σε επίπεδο πύλης
- Χρήση λογικών εξισώσεων (Boolean functions)
- Χρησιμοποιείται κυρίως για το σχεδιασμό βασικών συνδυαστικών κυκλωμάτων (αθροιστές, πολ/στέςκ.λ.π.)
- Υψηλοί χρόνοι σύνθεσης και εξομοίωσης

Στη VHDL, οι υποστηριζόμενοι τρόποι περιγραφής κυκλωμάτων είναι:

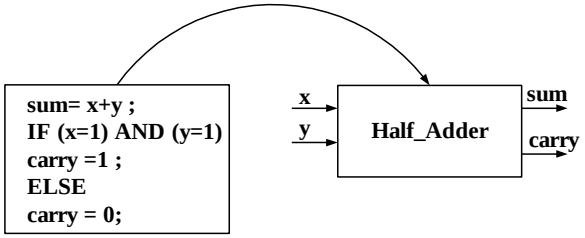
- Συμπεριφοράς (Behavioral VHDL)
- Ροής Δεδομένων (Dataflow VHDL)
- Δομής (Structural VHDL)

Και οι τρεις παραπάνω μέθοδοι περιγραφής μπορούν να χρησιμοποιηθούν σε κάθε επίπεδο της ροής σχεδιασμού. Καθώς μετακινούμαστε από το επίπεδο συμπεριφοράς στο επίπεδο δομής έχουμε:

- Αναλυτικότερη κυκλωματική περιγραφή

- Καλύτερο έλεγχο της σύνθεσης του κυκλώματος
- Μεγαλύτερος κώδικας
- Υψηλότεροι χρόνοι εξομοίωσης

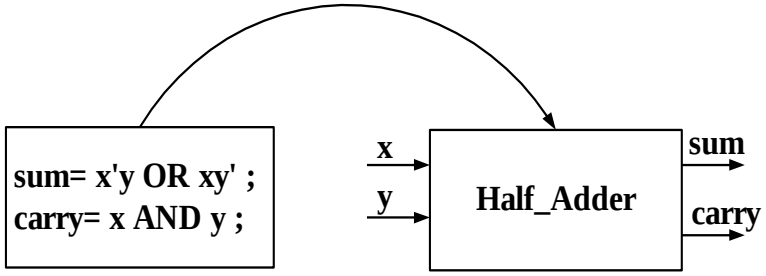
Behavioral VHDL: Χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο. Υποστηρίζει:
Αλγοριθμική περιγραφή της λειτουργίας του κυκλώματος
Δεν περιγράφεται αναλυτικά η κυκλωματική δομή του κυκλώματος
Δεν απαιτούνται αναλυτικές λογικές εξισώσεις
Δυνατότητα εξομοίωσης για επιβεβαίωση ορθής λειτουργίας και κατανόησης (καταρχήν) της λειτουργίας του κυκλώματος
Επαλήθευση μέσω back-annotated πληροφορίας, επιτρέπει επιπλέον να γίνει επιβεβαίωση ορθής λειτουργίας με στοιχεία της τεχνολογίας ολοκλήρωσης



Εικόνα 4.2: Η περιγραφή ενός half_adder με behaviouralVHDL

Data – Flow VHDL: Σε αυτό τον τρόπο περιγραφής έχουμε:

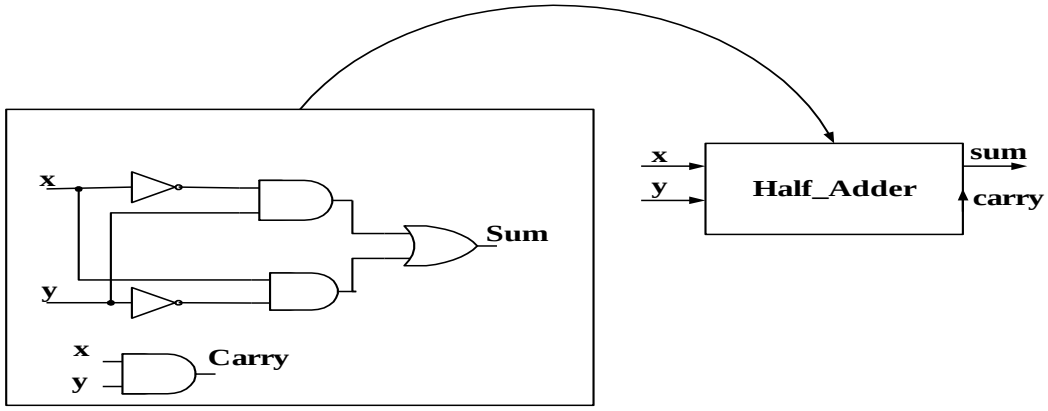
- Αναλυτικότερη περιγραφή της λειτουργίας του κυκλώματος
- Χρήση λογικών εξισώσεων για την μοντελοποίηση της ροής δεδομένων
- Δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας



Εικόνα 4.3: Η περιγραφή ενός half_adder με DataFlowVHDL

Structural VHDL: Χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος.

- Οι δομικές μονάδες ποικίλουν από απλές πύλες μέχρι σύνθετα κυκλώματα
- Προϋποθέτει την ύπαρξη βιβλιοθήκης από σχεδιασθέντα δομικά στοιχεία και την δυνατότητα χρήσης αυτών



Εικόνα 4.4: Η περιγραφή ενός half_adder με StructuralVHDL

Άσκηση Αυτοαξιολόγησης

Διαδραστικό πρόγραμμα 4.1

Επιλέξτε τη σωστή απάντηση:

1. Η VHDL είναι γλώσσα
   A. προγραμματισμού όπως η C και η Java
   B. περιγραφής ιστοσελίδων
   C. περιγραφής ψηφιακών κυκλωμάτων
   D. αντικειμενοστραφούς προγραμματισμού

Επιλέξτε τη σωστή απάντηση:

2. Η VHDL ΔΕΝ χρησιμοποιείται για
   A. Μοντέλα προδιαγραφών
   B. Σύνθεση ψηφιακών κυκλωμάτων
   C. Επιβεβαίωση ορθού σχεδιασμού
   D. Προγραμματισμό διαδικτυακών εφαρμογών

3. Αντιστοιχίστε τα στοιχεία των δύο στηλών

1. Επαναχρησιμοποίηση σχεδιασθέντων υπομονάδων	A. Επαναχρησιμοποίηση υπάρχουσας κυκλωματικής περιγραφής σε διαφορετικές τεχνολογίες
2. Υποστήριξη συντρεχουσών δομών	B. απαραίτητες για την περιγραφή της λειτουργίας του υλικού
3. Περιγραφή κυκλώματος/συστήματος ανεξάρτητα από την τεχνολογία υλοποίησης	C. Υποστήριξη από πληθώρα αναπτυξιακών εμπορικών εργαλείων σχεδιασμού
4. Παγκόσμιο πρότυπο	D. Μικρότερος χρόνος ανάπτυξης βελτιωμένων εκδόσεων του κυκλώματος

4. Βάλτε τα παρακάτω βήματα ανάπτυξης ενός κυκλώματος στη σωστή σειρά
   A. Σύνθεση
   B. Διατύπωση προδιαγραφών
   C. Εξαγωγή αρχείου netlist για προγραμματισμό του ολοκληρωμένου
   D. Ανάπτυξη κώδικα VHDL

5. Αντιστοιχίστε τα στοιχεία των δύο στηλών

1. Ο Behavioral τρόπος περιγραφής στη VHDL	A. Χρησιμοποιεί λογικές εξισώσεις για την μοντελοποίηση της ροής δεδομένων
2. Ο τρόπος περιγραφής dataflow (ροής δεδομένων) στη VHDL	B. Χρησιμοποιείται για την μοντελοποίηση της συμπεριφοράς του κυκλώματος σε υψηλό και αφηρημένο επίπεδο
3. Ο structural τρόπος περιγραφής (περιγραφή δομής) στη VHDL	C. Χρησιμοποιείται για την περιγραφή της διασύνδεσης των δομικών μονάδων του κυκλώματος
4. Η περιγραφή σε επίπεδο συστήματος	D. Παρέχει δυνατότητα εξομοίωσης και επιβεβαίωσης ορθής λειτουργίας

4.2Σχεδίαση απλών συνδυαστικών κυκλωμάτων

4.2.1 Θεωρητικό υπόβαθρο

Για να περιγράψουμε ένα κύκλωμα, φτιάχνουμε ένα αρχείο (στο οποίο δίνουμε ένα όνομα με κατάληξη .vhd). Το αρχείο αυτό οργανώνεται συνήθως σε τρία τμήματα:

1. Το τμήμα δήλωσης των βιβλιοθηκών (library).
2. Το τμήμα δήλωσης της εξωτερικής μορφής του κυκλώματος - οντότητας (entity) όπου ορίζονται το όνομά του και οι διαπεφές του (σήματα εισόδων και εξόδων) του κυκλώματος
3. η περιγραφή της αρχιτεκτονικής της οντότητας (architecture) όπου περιγράφεται η λειτουργία και συμπεριφορά του

Η περιγραφή της **οντότητας** περιλαμβάνει το όνομα αυτής και τα σήματα εισόδου και εξόδου. Η γενικημορφη της δηλωσης μιας οντοτητας είναι:

Περιοχήδήλωσηςοντότητας **(entity)**	
Σύνταξη	**Παράδειγμα**
ENTITYentity_name**IS** **PORT** ([**SIGNAL**] signal_name {,signal,name}:[mode] type_name {; **SIGNAL**] signal_name {,signal_name¨}: [mode] type_name}); **END** entity_name	**ENTITY** example1 **IS** **port** (x1,x2,x3: **IN BIT**; f :**OUT BIT**); **END** example1;

Στον παραπάνω πίνακα οι τονισμένες λέξεις αποτελούν δεσμευμένες λέξεις της γλώσσας.
Η αρχιτεκτονική (architecture) παρέχει τις λεπτομέρειες του κυκλώματος. Η περιγραφή της **αρχιτεκτονικής** μπορεί να γίνει με τους εξής τρόπους:

- behavioral ή μοντέλο συμπεριφοράς το οποίο είναι πιο κοντά στην ανθρώπινη λογική
- structural/gatelevel, ή δομικό που είναι πιο κοντά στο hardware

Αποτελείται από 2 κύρια μέρη:

- την περιοχή δήλωσης των σημάτων η οποία εμφανίζεται πριν τη λέξη κλειδί begin και
- το σώμα της αρχιτεκτονικής (architecture body)

Για τα σήματα εισόδου – εξόδου υποστηρίζονται οι ακόλουθες καταστάσεις:

- **in** : Σήμα εισόδου – Ανάγνωση σήματος
- **out** : Σήμα εξόδου – Εγγραφή σήματος
- **inout** : Σήμα εισόδου/εξόδου – Ανάγνωση και εγγραφή σήματος
  - Σήμα διπλής κατεύθυνσης(bi-directional)
- **buffer** : Διάβασμα, επανεγγραφή και αποθήκευση τιμής
  - Είναι πάντοτε σήμα εξόδου και όχι διπλής κατεύθυνσης
  - Επιτρέπεται μόνο μία ανάθεση εντός της αρχιτεκτονικής

Οι δεσμευμένες λέξεις της γλώσσας φαίνονται στον ακόλουθο πίνακα:

abs	entity	nor	select
access	exit	not	severity
after	file	null	shared
alias	for	of	signal
all	function		sla
and	generate	open	sll

architecture	generic	or	sra
array	guarded	others	srl
assert	if	out	subtype
attribute	impure	package	then
begin	in	port	to
block	inertial	postponed	transport
body	inout	procedure	type
buffer	is	process	unaffected
bus	label	pure	units
case	library	range	until
component	linkage	record	use
configuration	literal	register	variable
constant	loop	reject	wait
disconnect	map	rem	when
downto	mod	report	while
else	nand	return	with
elsif	new	rol	xnor
end	next	ror	xor

Πίνακας 4.1: Δεσμευμένες λέξεις της γλώσσας VHDL

Η γενική μορφή μιας αρχιτεκτονικής είναι:

Περιοχή δήλωσης αρχιτεκτονικής **(architecture)**	
Σύνταξη	**Παράδειγμα**
ARCHITECTUREarchitecture_name**OF**entity_name**IS** [**SIGNAL** declarations] [**CONSTANT** declarations] [**TYPE** declarations] [**ATTRIBUTE** specifications] **BEGIN** {**COMPONENT** instantiation statement ;} {**CONCURENT ASSINGMENT** statement ;} {**PROCESS** statement ;} {**GENERATE** statement ; } **END** [architecture_name] ;	**ARCHITECTURE**logicFunc**OF** example1 **IS** **BEGIN** f <= (x1 AND x2) OR (NOT x2 AND x3); **END**logicFunc

Επαλήθευση ορθής λειτουργίας κυκλώματος

Για να επαληθεύσουμε τη λειτουργία ενός κυκλώματος χρησιμοποιούμε ένα αρχείο δοκιμών που ονομάζεται test-bench. (Το αρχείο δοκιμών συνήθως ονομάζεται όπως και το αρχείο που δηλώνει το κύκλωμα προσθέτοντας την ένδειξη _TB στο τέλος. Παράδειγμα: comb1_TB.vhd). Το αρχείο δοκιμών δίνει τιμή στις εισόδους του κυκλώματος (όπως κάνατε στον πάγκο με τους διακόπτες) προκειμένου με την προσομοίωση να ελεγχθεί η ορθή λειτουργία του κυκλώματος (όπως στον πάγκο με τα led). Η γενική μορφή ενός αρχείου δοκιμών είναι:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use IEEE.std_logic_textio.all;
use ieee.std_logic_arith.all;

library work;
```

```
use work.all;
```

```
entity example1_tb is
end example1_tb;
```

```
architecture example1_tb_a of example1_tb is

  component example1 is
     port (
              x1,x2,x3: IN  BIT;
                  f             :OUT BIT);
                       );

  end component;

   signal x1,x2,x3, f: std_logic;

begin

example1_inst: example1
     port map (
x1,x2,x3, f );
```

```
stimulus_proc : process is
begin
x1<='0';
x2<='0';
x3<='0';
wait for 12ns;

x1<='0';
x2<='0';
x3<='1';
wait for 2ns;

x1<='0';
x2<='1';
x3<='0';
wait for 3ns;

x1<='0';
x2<='1';
x3<='1';
wait for 7ns;

x1<='1';
x2<='0';
x3<='0';
wait for 2ns;
  wait;

end process stimulus_proc;
end architecture example1_tb_a;
```

4.2.2 Πειραματικό μέρος

Το παρακάτω πρόγραμμα περιγράφει μια πύλη AND δύο εισόδων:

```
library IEEE;
  use IEEE.std_logic_1164.all;
  USE ieee.std_logic_arith.all ;
```

```
  USE ieee.std_logic_unsigned.all ;
  use ieee.std_logic_textio.all;   -- in order to use hread( )
library work;
use work.all;
```

```
entity AND_gate is
      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );
end AND_gate ;
```

```
architecture rtl of AND_gate is

begin
process (x, y)
  begin
   z<=x AND y;
endprocess;

endrtl ;
```

Τα προγράμματα της VHDL αποτελούνται από 3 τμήματα:

- Το τμήμα δήλωσης των βιβλιοθηκών (library).
- Το τμήμα δήλωσης του διεπαφών (εισόδων και εξόδων) του κυκλώματος (entity)
- Το τμήμα δήλωσης της συμπεριφοράς - αρχιτεκτονικής του κυκλώματος.

Επαλήθευση ορθής λειτουργίας κυκλώματος

Για να επαληθεύσουμε τη λειτουργία ενός κυκλώματος χρησιμοποιούμε ένα αρχείο δοκιμών που ονομάζεται test-bench. (Το αρχείο δοκιμών συνήθως ονομάζεται όπως και το αρχείο που δηλώνει το κύκλωμα προσθέτοντας την ένδειξη _TB στο τέλος. Παράδειγμα: AND_gate_TB.vhd).

Το αρχείο δοκιμών έχει την ακόλουθη μορφή:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use IEEE.std_logic_textio.all;
use ieee.std_logic_arith.all;

library work;
use work.all;
```

```
entity AND_gate_tb is
end AND_gate_tb;
```

```
architecture AND_gate_tb_a of AND_gate_tb is
component AND_gate is
      port (
            x: in std_logic;
            y: in std_logic;
            z: out std_logic
                  );
  end component;
   signal X, y, z: std_logic;
begin
AND_gate_inst: AND_gate
     port map (
           x, y, z       );
```

```
stimulus_proc : process is
```

```
begin
x<='0';y<='0';
wait for 12ns;
x<='1';  y<='0';
wait for 2ns;
x<='1'; y<='1';
wait for 3ns;
x<='0'; y<='1';
wait for 7ns;
x<='0'; y<='0';
wait for 2ns;
  wait;
end process stimulus_proc;
end architecture AND_gate_tb_a;
```

Το testbench αποτελείται από 4 μέρη:

- Δήλωση βιβλιοθηκών
- Δήλωση οντότητας (παρατηρείστε ότι δεν υπάρχουν είσοδοι και έξοδοι αφού δεν πρόκειται για κύκλωμα με εισόδους και εξόδους αλλά για ένα περιβάλλον δοκιμής)
- Την αρχιτεκτονική η οποία αποτελείται από 2 τμήματα
    - Τμήμα δήλωσης του κυκλώματος που θέλουμε να ελέγξουμε και τον ονομάτων που δίνουμε στα σήματα αυτού
    - Τους συνδυασμούς εισόδων που θέλουμε να δοκιμάσουμε.

Όταν θέλουμε να ελέγξουμε ένα συνδυαστικό κύκλωμα 2 εισόδων δοκιμάζουμε και τους 4 δυνατούς συνδυασμούς των εισόδων, δηλαδή τον πίνακα αληθείας.
Ερώτηση: Ελέγχει όλους τους συνδυασμούς τιμών εισόδων το παραπάνω test-bench; (Απάντηση: ΟΧΙ)
Ο έλεγχος ορθής λειτουργίας γίνεται με τη βοήθεια εργαλείων προσομοίωσης. Ένα τέτοιο εργαλείο είναι το Modelsim.
Οδηγίες συντακτικού ελέγχου κώδικα και προσομοίωσης

1. Κάνουμε συντακτικό έλεγχο του αρχείου που περιγράφει το κύκλωμα. (επιλέγουμε το αρχείο και με δεξί click επιλέγουμε “compile”).
2. Στο κάτω τμήμα της οθόνης (παράθυρο transcript) με πράσινα γράμματα φαίνεται το επιτυχές αποτέλεσμα του συντακτικού ελέγχου ενώ με κόκκινα αν η προσπάθεια δεν ήταν επιτυχής. Στη δεύτερη περίπτωση πρέπει να κάνουμε διπλό click στα κόκκινα γράμματα για να μας δώσει περισσότερες λεπτομέρειες για τα λάθη μας. Οπότε:
    a. Ανοίγουμε το αρχείο περιγραφής του κώδικα
    b. Πηγαίνουμε στη γραμμή που μας υποδεικνύει το Modelsim και διορθώνουμε το λάθος
    c. Αποθηκεύουμε το αρχείο και
    d. Επαναλαμβάνουμε το συντακτικό έλεγχο
3. Αποτο menu τουmodelsimεπιλέγουμε simulate/start simulation/design και επιλέγουμε work/and_gate_tb
4. Απο το παράθυρο objects επιλέγουμε όλα τα σήματα και με δεξί click επιλέγουμε addtowave.
5. Μεγιστοποιούμε το παράθυρο wave και επιλέγουμε run. Τώρα μπορούμε να δούμε στην οθόνη πως αλλάζει η έξοδος z για διαφορετικές τιμές των εισόδων, όπως θα βλέπαμε και στο led στον πάγκο.

4.2.3 Τεστ αυτοαξιολόγησης

Άσκηση 1
Αναπτύξτε το παραπάνω πρόγραμμα με το όνομα AND_gate.vhd. Αναπτύξτε κύκλωμα πύλης OR και πύλης NOT. Τα αντίστοιχα αρχεία να αποθηκευτούν με τις ονομασίες OR_gate.vhd και NOT_gate.vhd αντίστοιχα.
Λύση

Διαδραστικό πρόγραμμα 4.2

Άσκηση 2
Χρησιμοποιώντας το testbench της πύλης AND (αρχείο AND_gate_TB.vhd), δοκιμάστε 40ns μετά την τελευταία αλλαγή τιμών εισόδου το συνδυασμό x=1, y=1 και αποθηκεύστε το νέο testbench με το όνομα AND_gate_TB_v2.vhd
Λύση

Διαδραστικό πρόγραμμα 4.3

Άσκηση 3
Αναπτύξτε test-bench για την πύλη OR που φτιάξατε με τη γλώσσα VHDL. Επαληθεύστε συντακτικά και λειτουργικά την πύλη σας.
(*Υπόδειξη*:

1. προσθέστε τα νέα αρχεία
2. Ελέγξτε τα συντακτικά
3. Επαναλάβετε τα βήματα 3, 4, 5 που φαίνονται παραπάνω για το νέο κύκλωμα.)

Άσκηση 4
Σας δίνεται το αρχείο example1.vhd και το σχετικό testbendh (example1_TB.vhd).

Διαδραστικό πρόγραμμα 4.4

Προσομοιώστε τη λειτουργία του κυκλώματος με το δοσμένο test-bench και απαντήστε:
Τη χρονική στιγμή 20ns, τι τιμή έχει:

- Η είσοδος x1 ☐ Σωστό 0
- Η είσοδος x2 ☐ Σωστό 1
- Η είσοδος x3 ☐ Σωστό 1
- Η έξοδος f ☐ Σωστό 0

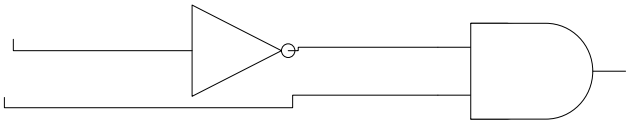
Έλεγχος απαντήσεων

Διαδραστικό πρόγραμμα 4.5

Άσκηση 5
Επαυξήστε το παραπάνω test-bench, προκειμένου να ελέγχει όλες τις πιθανές περιπτώσεις και απαντήστε:
Τι τιμή έχει η έξοδος όταν:
Οι είσοδοι έχουν τιμή 101 ☐ Σωστό 1
Οι είσοδοι έχουν τιμή 111 ☐ Σωστό 1
Λύση

Διαδραστικό πρόγραμμα 4.6

Άσκηση 6
Στο παρακάτω σχήμα φαίνεται ένα συνδυαστικό κύκλωμα.



a) Αναπτύξτε το αρχείο περιγραφής του κυκλώματος αυτού με χρήση της γλώσσας VHDL.
b) Υπολογίστε τον πίνακα αληθείας
c) Αναπτύξτε το αρχείο δοκιμής του κυκλώματος (testbench) με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο δοκιμών με το όνομα comb_v3_TB.vhd

Άσκηση 7
Δίνεται η έξοδος F ενός κυκλώματος η οποία εκφράζεται από τη συνάρτηση F=xy'+z.
a) Συμπληρώστε τον πίνακα αληθείας με βάση θεωρητικούς υπολογισμούς

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

έλεγχος

b) Αναπτύξτε το αρχείο περιγραφής του κυκλώματος αυτού με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο με το όνομα comb.vhd. Λύση
c) Αναπτύξτε το αρχείο δοκιμής του κυκλώματος (testbench) με χρήση της γλώσσας VHDL. Αποθηκεύστε το αρχείο δοκιμών με το όνομα comb_TB.vhd Λύση

Λύση

Διαδραστικό πρόγραμμα 4.7

4.3 Οι δομές IF – THEN – ELSE και CASE στη VHDL

4.3.1 Θεωρητικό υπόβαθρο

Η σύνταξη της ροής **IF** είναι:

```
[if label : ]   IF boolean expression THEN
                    {sequential statement}
                ELSIFboolean expression THEN
                    {sequential statement}
                ELSE
                    {sequential statement}
                END IF [if label];
```

ΠΡΟΣΟΧΗ!!! Είναι ακολουθιακή δήλωση => Η σειρά εμφάνισης των συνθηκών είναι σημαντική

Το ισοδύναμο κύκλωμα έχει προτεραιότητες ανάλογα με τη σειρά εμφάνισης των συνθηκών στον κώδικα.

Παράδειγμα:

```
IFsel = '1' THEN
    c <= d;
ELSE
    c<=a;
endif ;
```

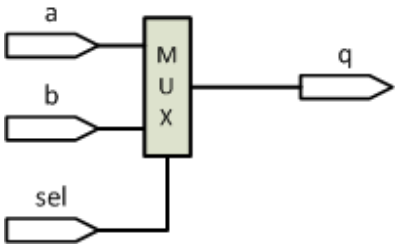
Η σύνταξη της ροής **CASE** είναι:

```
[case label : ]   CASE expression IS
                      WHEN choise1 =>
                          {sequential statement}
                      WHEN choise2 =>
                          {sequential statement}
                          . . .
                          . . .
                  END CASE;
```

Παράδειγμα:

```
CASE sel IS
    WHEN '0' => c<=a;
    WHEN '1' => c<=b;
ENDCASE;
```

Δείτε το κυκλωματικό ισοδύναμο



Σημείωση: Η παραπάνω case είναι ισοδύναμη με το παράδειγμα για την IF.
Παρατηρήσεις για την CASE:

- Η έκφραση πρέπει να είναι διακριτού τύπου
- Όλες οι δυνατές τιμές πρέπει να απαριθμούνται
- Χρήση others στην περίπτωση που αυτές είναι πολλές
- Όταν δεν πρέπει να αλλάξει κάποια τρέχουσα τιμή, δηλαδή καμία ενέργεια να μη λάβει χώρα (donothing), χρησιμοποιείται η δήλωση null.
  Παράδειγμα:
  CASE a IS
  WHEN "00" => q1<= '1';
  "01" =>q1 <='0';
  WHENOTHERS =>null;

4.3.2 Πειραματικό μέρος

Άσκηση 1
Περιγράψτε έναν πολυπλέκτη 4 εισόδων δεδομένων (2 σήματα επιλογής/διεύθυνσης) σε VHDL χρησιμοποιώντας τη δομή IF-ELSE
Άσκηση 2
Αναπτύξτε το σχετικό αρχείο δοκιμών και επαληθεύστε τη λειτουργία του πολυπλέκτη.

4.3.3 Τεστ αυτοαξιολόγησης

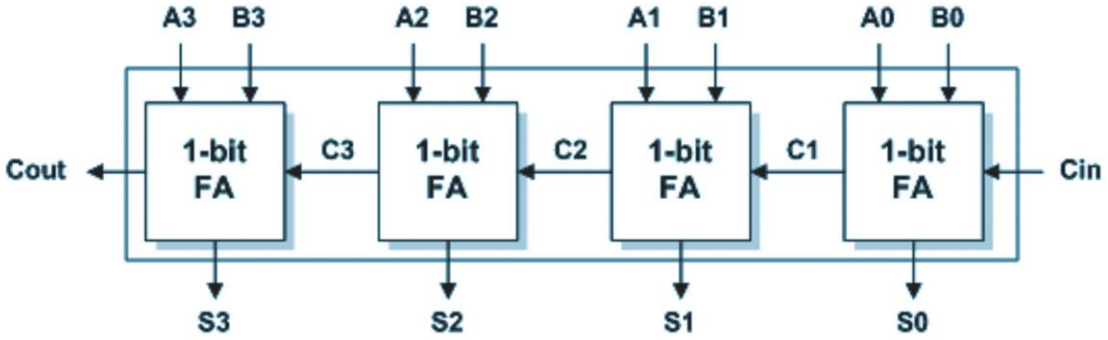
Περιγράψτε έναν πολυπλέκτη 4 εισόδων δεδομένων (2 σήματα επιλογής/διεύθυνσης) σε VHDL χρησιμοποιώντας τη δομή CASE
Λύση

Διαδραστικό πρόγραμμα 4.8

4.4 Διεπίπεδη λογική σε συνδυαστικά κυκλώματα

4.4.1 Θεωρητικό υπόβαθρο

Όπως γνωρίζουμε από την άσκηση σχεδίασης αθροιστών, ένας αθροιστής 4 bit υλοποείται με 4 πλήρης αθροιστές ή τρεις πλήρης αθροιστές και έναν ημιαθροιστή. Το σχετικό κύκλωμα φαίνεται στο ακόλουθο σχήμα, όπου είναι φανερό ότι το δομικό στοιχείο για την κατασκευή του κυκλώματος αθροιστή 4-μπιτων αριθμών είναι ο πλήρης αθροιστής. Το μοντέλο αυτό ονομάζεται δομικό.



Εικόνα 1: Αθροιστής τετραψήφιων δυαδικών αριθμών που αποτελείται από τέσσερις πλήρεις αθροιστές

Θα πρέπει λοιπόν πρώτα να σχεδιαστεί σε VHDL ένας πλήρης αθροιστής του ενός bit (1-bitFA) και στη συνέχεια να κληθεί αυτός 4 φορές ως υποκύκλωμα. Κάθε στοιχειώδες κύκλωμα (ή υποκύκλωμα) που αποτελεί τμήμα ενός πιο σύνθετου κυκλώματος καλείται *στοιχείο* (component). Κάθε στοιχείο σχεδιάζεται σε VHDL ξεχωριστά ως αυτόνομο κύκλωμα.

Στο πρόγραμμα σχεδίασης του συνολικού κυκλώματος (και συγκεκριμένα στο τμήμα της αρχιτεκτονικής) το δομικό στοιχείο – υποκύκλωμα δηλώνεται ως COMPONENT, ενώ καλείται μέσω της δήλωσης PORTMAP.

Στον ακόλουθο πίνακα φαίνεται η σύνταξη της δήλωσης στοιχείου και το παράδειγμα για έναν πλήρη αθροιστή.

δήλωσης στοιχείου **(component)**	
Σύνταξη	**Παράδειγμα**
COMPONENTόνομα_στοιχείου**IS** **PORT(** όνομα_σήματος {,όνομα_σήματος} : mode_σήματος τύπος_δεδομένων; … όνομα_σήματος {,όνομα_σήματος} : mode_σήματος τύπος_δεδομένων**);** **END COMPONENT** ;	**COMPONENT** FA **PORT** (A, B, Cin : IN std_logic; S, Cout : OUT std_logic); **END COMPONENT**;

Η κλήση ενός *στοιχείου* (component) γίνεται με τη δήλωση PORTMAP, της οποίας η γενική σύνταξη φαίνεται στον ακόλουθο πίνακα.

Δήλωση PORT MAP	
Σύνταξη	**Παράδειγμα**
ετικέτα:όνομα_στοιχείου **PORTMAP** (αντιστοίχιση σημάτων του port);	**FA_1:** FA **PORT MAP** (A(0), B(0), C(0), Sum(0), C(1));

Κάθε δήλωση PORTMAP ξεκινά υποχρεωτικά με ένα όνομα ή *ετικέτα* (label), ακολουθούμενο από το όνομα του στοιχείου το οποίο καλείται και τη δήλωση PORT MAP αμέσως μετά. Ακολουθεί μέσα σε παρένθεση η αντιστοίχιση των σημάτων του PORT της δήλωσης

COMPONENT με τα σήματα που χρησιμοποιούνται πλέον στο τελικό πρόγραμμα. Με τον τρόπο αυτό γίνεται ουσιαστικά η σύνδεση των δομικών στοιχείων και του τελικού κυκλώματος.

4.4.2 Πειραματικό μέρος

Χρησιμοποιώντας το πλήρη αθροιστή που σας δίνεται εδώ και την ακόλουθη περιγραφή ενός αθροιστή 4-bit, προσομοιώστε έναν αθροιστή 4-bit.

```
entity Adder_4bit is
        port(a,b: in std_logic_vector(3 downto 0);
        enable: in std_logic;
        sum: out std_logic_vector(3 downto 0);
        cout: out std_logic);
end Adder_4bit;
architecturestruct of Adder_4bit is
signal c : std_logic_vector(2 downto 0);
signal s : std_logic_vector(3 downto 0);
component And_2
port( i1, i2 : in std_logic;
        o1: out std_logic);
end component;
component FA
port(a,b,cin: in std_logic;
        s, cout: out std_logic);
end component;
begin
        FA0: FA port map(a(0), b(0), 0, s(0), c(0));
        FA1: FA port map(a(1), b(1), c(0), s(1), c(1));
        FA2: FA port map(a(2), b(2), c(1), s(2), c(2));
        FA3: FA port map(a(3), b(3), c(2), s(3), c(3));
        g0: And_2 port map (s(0), enable, sum(0));
        g1: And_2 port map (s(1), enable, sum(1));
        g2: And_2 port map (s(2), enable, sum(2));
        g3: And_2 port map (s(3), enable, sum(3));
endstruct;
```

4.4.3 Τεστ αυτοαξιολόγησης

Έχοντας αναπτύξει στην προηγούμενη άσκηση τον πολυπλέκτη 4 σε 1, αναπτύξτε πολυπλέκτη 16 σε 1 χρησιμοποιώντας πολλαπλούς πολυπλέκτες 4 σε 1.

Λύση

Διαδραστικό πρόγραμμα 4.9

Δίνεται το ακόλουθο τμήμα κώδικα VHDL

```
ARCHITECTURE structural OF xor3 IS
SIGNAL U1_OUT: STD_LOGIC;

COMPONENT xor2 IS
    PORT(
            I1 : IN STD_LOGIC;
            I2 : IN STD_LOGIC;
            Y : OUT STD_LOGIC
        );
END COMPONENT;

BEGIN
    U1: xor2 PORT MAP (I1 => A,
                       I2 => B,
                       Y  => U1_OUT);

    U2: xor2 PORT MAP (I1 => U1_OUT,
                       I2 => C,
                       Y  => Result);

END structural;
```

labels

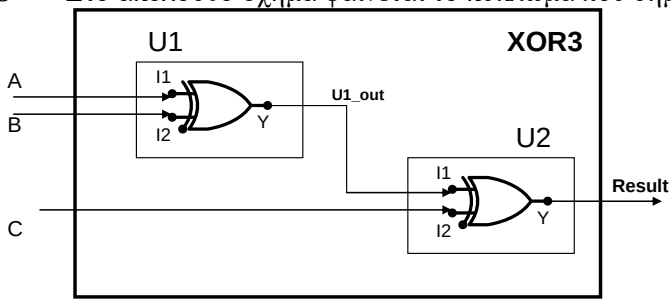
1 Ποιο είναι το δομικό στοιχείο που χρησιμοποιείται στο παράδειγμα? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

2 Πόσες φορές χρησιμοποιείται το δομικό στοιχείο? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

3 Ποιο είναι το label την πρώτη φορά? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

4 Ποιο είναι το όνομα του κυκλώματος που δημιουργείται με την παραπάνω αρχιτεκτονική; Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

5 Στο ακόλουθο σχήμα φαίνεται το κύκλωμα που δημιουργείται. Παρατηρείστε ότι:



- Συμπληρώστε τα κενά.
- Η είσοδος I1 του U1 συνδέεται στην είσοδο .....του κυκλώματος XOR3 ενώ η είσοδος I1 του U2 συνδέεται στην ..... του .....
- Το σήμα U1_out αποτελεί ταυτόχρονα..... του U1, ... του U2 και εσωτερικό σήμα για το XOR3.

Διαδραστικό πρόγραμμα 4.10

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

ENTITY dec4to16 IS
PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
En : IN STD_LOGIC ;
y : OUT STD_LOGIC_VECTOR(0 TO 15) ) ;
```

```
END dec4to16 ;

ARCHITECTURE Structure OF dec4to16 IS
      COMPONENT dec2to4
      PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
      En : IN STD_LOGIC ;
      y : OUT STD_LOGIC_VECTOR(0 TO 3) ) ;
      END COMPONENT ;
      SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
G1: FOR i IN 0 TO 3 GENERATE
      Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
     G2: IF i=3 GENERATE
Dec_left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
     END GENERATE ;
END GENERATE ;
END Structure;
```

- Ποιο είναι το δομικό στοιχείο που χρησιμοποιείται στο παράδειγμα? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής
- Πόσες φορές χρησιμοποιείται το δομικό στοιχείο? Κουτί για να συμπληρώσει ο χρήστης/φοιτητής

Διαδραστικό πρόγραμμα 4.11

4.5 Σχεδίαση απλών ακολουθιακών κυκλωμάτων

4.5.1 Θεωρητικό υπόβαθρο

Τα σύγχρονα ακολουθιακά κυκλώματα μοντελοποιούνται/περιγράφονται με clockedprocesses. Για τη διέγερση τους απαιτείται αλλαγή της τιμής του σήματος ρολογιού. Ένα ακολουθιακό κύκλωμα αλλάζει κατάσταση μόνο στις χρονικές στιγμές αλλαγής του ρολογιού.
Οι εναλλακτικές περιγραφές του ρολογιού φαίνονται στον ακόλουθο πίνακα:

1	Alt 1: **PROCESS**(CLK) **BEGIN** **IF**clk'event**AND**clk='1' **THEN** q<=d; **ENDIF**; **ENDPROCESS**;
2	Alt 2: **PROCESS**(CLK) **BEGIN** **IF**clk='1' **THEN** q<=d; **ENDIF**; **ENDPROCESS**;
3	Alt 3: **PROCESS**(CLK) **BEGIN** **IF**clk'event**AND**clk='1' **AND**clk'last_value='0' **THEN** q<=d; **ENDIF**; **ENDPROCESS**;

4	Alt 4: **PROCESS** **BEGIN** **WAITUNTIL**clk='1'; q<=d; **ENDPROCESS**;
5	Alt 5: **PROCESS** **BEGIN** **WAITUNTIL**rising_edge(clk); q<=d; **ENDPROCESS**;

Πίνακας 4.2: Οι εναλλακτικές περιγραφές του ρολογιού

4.5.2 Πειραματικό μέρος

Δίνεται ο ακόλουθος μετρητής

```
architecturertlofcounteris
signalz_int: integer;
begin
z<=z_int;
process(rst, clk)
begin
ifrst='0' then
z_int<=0;
elsifclk='1' and clk'event then
z_int<=z_int+1;
end if;
endprocess;
endrtl;
```

Και το σχετικό αρχείο δοκιμών

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use IEEE.std_logic_textio.all;
use ieee.std_logic_arith.all;

library work;
use work.all;
entity counter_tb is
end counter_tb;
architecture counter_tb_a of counter_tb is

component counter is

      port (
rst: in std_logic;
clk: in std_logic;
            z  : out integer
                  );
end component;

constant clk_hp : time := 3000 ps;
```

```
  signal rst      :  std_logic;
  signal clk      :  std_logic;
  signal Z        :  INTEGER   ;

begin

counterinst: counter
port map
          (rst ,
clk,
           z    );
--------------------------------------------------------------------
clock_gen_proc : process is
begin
clk<= '1';
  wait for clk_hp;

clk<= '0';
  wait for clk_hp;

end process clock_gen_proc;
--------------------------------------------------------------------
stimulus_proc : process is
begin
rst<='0';


wait for 2* clk_hp;
wait for 0.5 ns ;
rst<='1';
wait for 60* clk_hp;
rst<='0';
wait for 2* clk_hp;
rst<='1';
wait for 2* clk_hp;
wait for 20* clk_hp;
  wait;

endprocessstimulus_proc;
```

Άσκηση 1

1. Ακολουθείστε τις οδηγίες προσομοίωσης, και ελέγξτε τη λειτουργία του παραπάνω κυκλώματος counter.
2. Αναπτύξτε κώδικα που να περιγράφει κύκλωμα μετρητή που μετράει από 1 ως 9 κυκλικά και επαληθεύστε τη λειτουργία του.

4.5.3 Τεστ αυτοαξιολόγησης

1. Αναπτύξτε κώδικα που να περιγράφει μετρητή με είσοδο x ο οποίος για x=0 μετράει από 1 έως 7 ανά δύο ενώ x=1 από 6 έως 12 ανά 1. (Υπόδειξη: ο μετρητής να μηδενίζεται όταν ενεργοποιείται το reset και να ελέγχεται πρώτα η περίπτωση x=0 και μετά η τιμή x=1.)

Λύση

Διαδραστικό πρόγραμμα 4.12

2. Χρησιμοποιώντας το ακόλουθο test-bench απαντήστε:

Ποια τιμή έχει ο μετρητής, τη χρονική στιγμή:

a. 3ns
b. 30ns
c. 70ns
d. 240ns

Έλεγχος

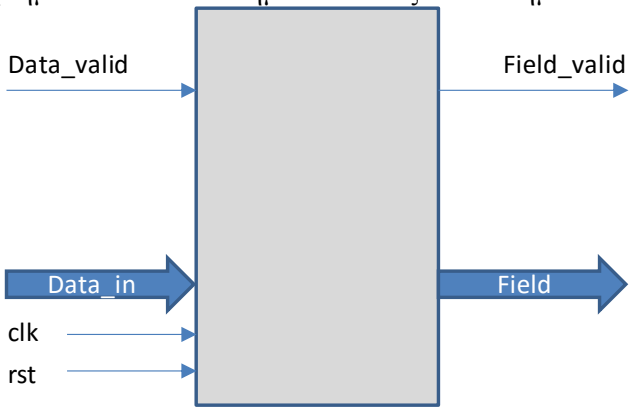
Διαδραστικό πρόγραμμα 4.13

3. Αναπτύξτε κώδικα που να περιγράφει κύκλωμα μετρητή που μετράει από 3 ως 53 κυκλικά και επαληθεύστε τη λειτουργία του.

4.6 Διεπίπεδη λογική σε ακολουθιακά κυκλώματα

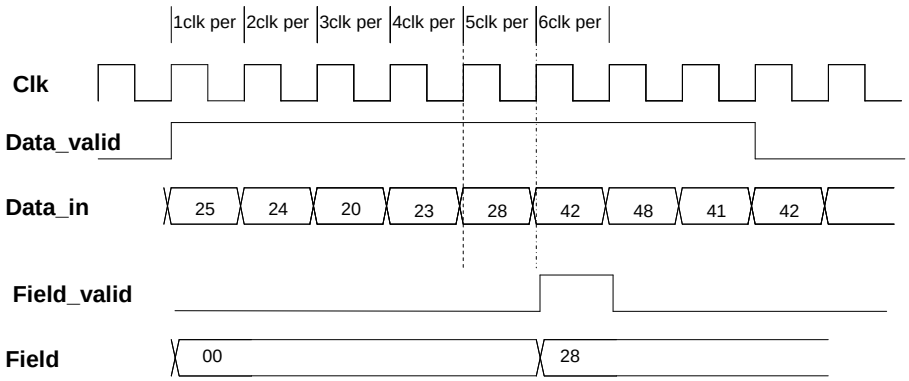
Θα εξετάσουμε την περιγραφή με VHDL διεπίπεδης λογικής με βάση ένα παράδειγμα: ένα κύκλωμα που υλοποιείται συχνά στο δέκτη ενός τηλεπικοινωνιακού συστήματος. Σε ένα τηλεπικοινωνιακό σύστημα, στην πλευρά του δέκτη, όταν λαμβάνεται ένα πακέτο πληροφορίας, αυτό προωθείται για επεξεργασία. Για παράδειγμα, πρέπει να βρεθεί η διεύθυνση του παραλήπτη προκειμένου αυτό να προωθηθεί κατάλληλα. Η θέση στην οποία βρίσκεται κάθε πληροφορία (π.χ. διεύθυνση παραλήπτη) είναι προκαθορισμένη από τα διεθνή πρότυπα (standards). Έτσι στη συσκευή του δέκτη ένα σύνολο ψηφιακών κυκλωμάτων αναλαμβάνουν την επεξεργασία του πακέτου. Υποθέτουμε ότι έχει φτάσει στο δέκτη ένα πακέτο πληροφορίας, και **πρέπει να βρούμε το περιεχόμενο (την πληροφορία) που βρίσκεται στο 5ºbyte του πακέτου** και να το προωθήσουμε σε ένα άλλο κύκλωμα για επεξεργασία. Ζητείται η σχεδίαση του κυκλώματος field_filtering το οποίο θα υλοποιεί την παραπάνω λειτουργία, σύμφωνα με τις παρακάτω προδιαγραφές:

Η εξωτερική μορφή του κυκλώματος field_filtering φαίνεται στο ακόλουθο σχήμα. (Σημειώνεται ότι το σήμα data όπως και το σήμα field έχουν εύρος 8 bit.)

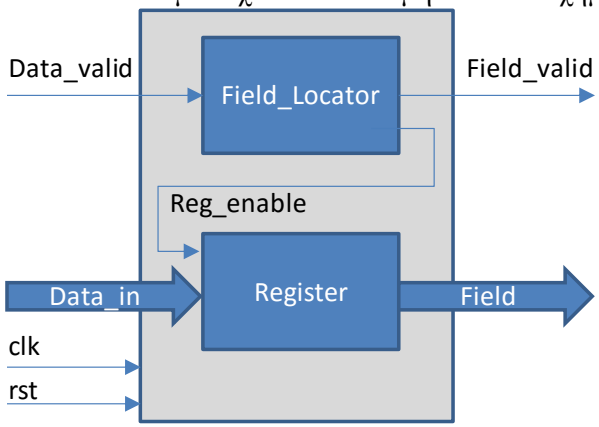


Εικόνα 4.6: Διεπαφές μονάδας field_filtering

Η επιθυμητή (εσωτερική) λειτουργία του κυκλώματος φαίνεται στο ακόλουθο σχήμα.

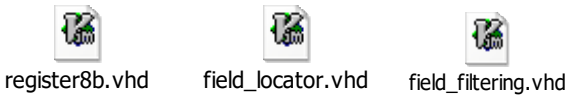


Το κύκλωμα field_filtering να δημιουργηθεί με δομικά στοιχεία (components) έναν καταχωρητή (register) και ένα κύκλωμα field_locator που εντοπίζει τον παλμό ρολογιού στον οποίο το κύκλωμα δέχεται το επιθυμητό πεδίο. Σχηματικά αυτό φαίνεται στο ακόλουθο σχήμα.



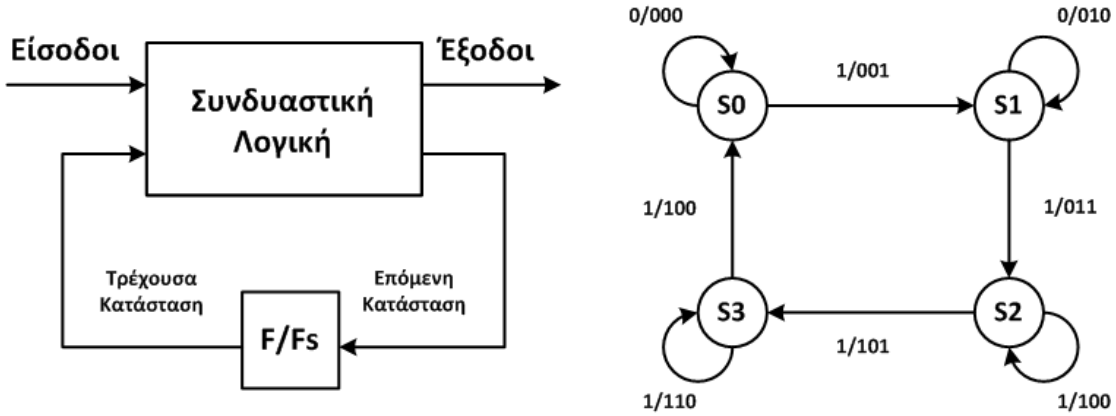
Εικόνα 4.7: Οργάνωση μονάδας field_filtering

Αναπτύξτε το σχετικό αρχείο δοκιμών και προσομοιώστε τη λειτουργία.



4.7 Σχεδίαση μηχανών καταστάσεων

Οι Μηχανές Πεπερασμένων Καταστάσεων (FiniteStateMachines, FSMs) χρησιμοποιούνται, για την υλοποίηση μιας λογικής ελέγχου (controllogic), η οποία είναι πάντοτε γρηγορότερη από την υλοποίηση αυτής σε επεξεργαστή μέσω προγράμματος. Αποτελούνται από συνδυαστικό κύκλωμα και F/Fs και μπορούν να περιγραφούν από ένα διάγραμμα εναλλαγής καταστάσεων (StateTransitionDiagram – STD).



Εικόνα 4.8: Παράδειγμα οργάνωσης μηχανής πεπερασμένων καταστάσεων και διαγράμματος εναλλαγής καταστάσεων (StateTransitionDiagram – STD)

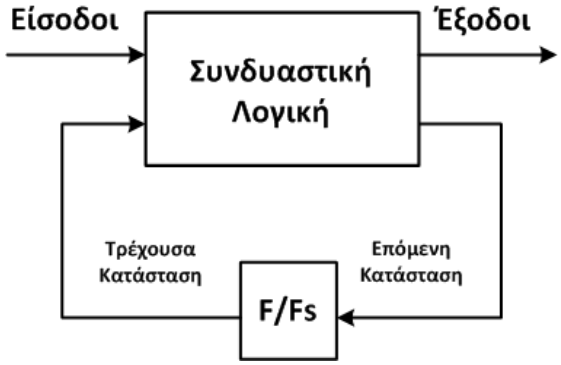
Για τη σωστή λειτουργία μιας μηχανής απαιτείται οι είσοδοι να έχουν σταθεροποιηθεί πριν την άφιξη του ρολογιού. Η καθυστέρηση καθορίζεται από το μέγιστο χρόνο υπολογισμού της νέας κατάστασης και κατά συνέπεια από το μέγιστο χρόνος απόκρισης του συνδυαστικού κυκλώματος.

Οι Μηχανές Πεπερασμένων Καταστάσεων ακολουθούν μια λειτουργία δύο φάσεων:

- Φάση 1: Υπολογισμός της νεάς κατάστασης
- Φάση 2: Δειγματοληψία της νέας κατάστασης

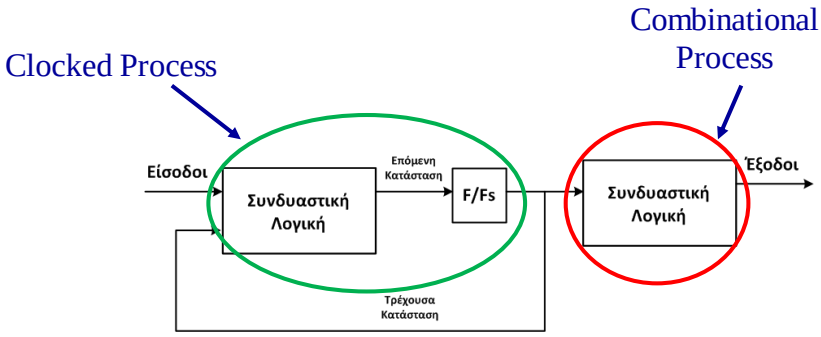
Υπάρχουν δύο μέθοδοι υλοποίησης:

- MealyFSMs : Οι έξοδοι είναι συνάρτηση των εισόδων και της τρέχουσας κατάστασης
-



Εικόνα 4.9: Οργάνωση μηχανής πεπερασμένων καταστάσεωνMealy

- MooreFSMs : Οι έξοδοι είναι συνάρτηση **μόνο** της τρέχουσας κατάστασης



Εικόνα 4.10: Οργάνωση μηχανής πεπερασμένων καταστάσεωνMoore

Οι MooreFSMs απαιτούν ένα κύκλο ρολογιού παραπάνω: ένα κύκλο υπολογισμού της κατάστασης και ένα κύκλο για τον υπολογισμό της εξόδου.

Στην VHDL, **ο σχεδιαστής μπορεί να δώσει ονόματα στις καταστάσεις** αφήνοντας την κωδικοποίηση αυτών (στο δυαδικό σύστημα) στα εργαλεία. Τα ονόματα των καταστάσεων δηλώνονται στην αρχή της αρχιτεκτονικής.
Παράδειγμα:

```
architecturemy_arch of flag_detector is
    type state is (zero,one,two,three,four,five,six,seven,eight);
signalpr_state, nx_state: state;
```

Με αυτό τον τρόπο έχουμε ορίσει 9 καταστάσεις.
Η αρχιτεκτονική ενός τέτοιου κυκλώματος περιγράφεται από δύο process:

- μια που υλοποιεί τις μεταβάσεις στην ακμή του ρολογιού.

Παράδειγμα:

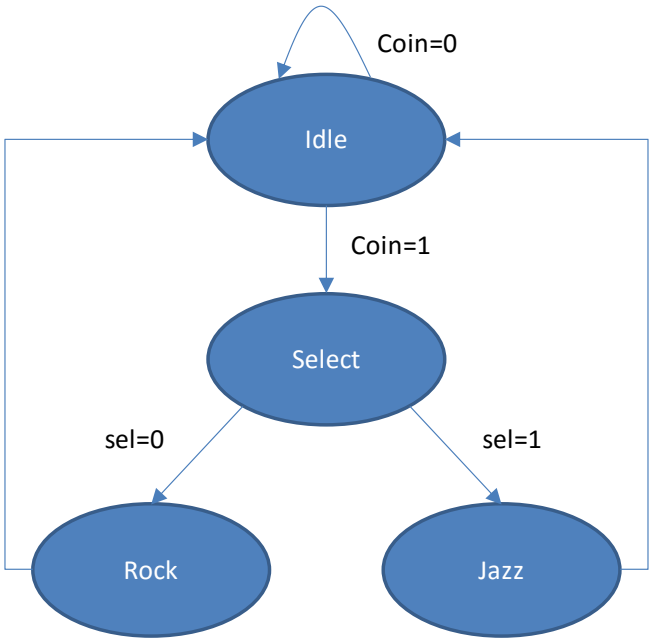
```
PROCESS (rst,clk)
begin
if (rst='0') then
pr_state<= zero;
elsif (clk'event and clk='1') then
        pr_state<= nx_state;
endif;
endprocess;
```

- μια που υλοποιεί το συνδυαστικό κύκλωμα και συνήθως περιγράφεται με την εντολή CASE. Η σύνταξη της εντολής case περιγράφεται παρακάτω ενώ επίσης ακολουθεί και ένα παράδειγμα. Στη θέση της «έκφραση» στην περίπτωση των ακολουθιακών κυκλωμάτων συνήθως μπαίνει η παρούσα κατάσταση.
-

Σύνταξη εντολής CASE	**Παράδειγμα**

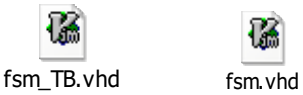
CASE έκφραση IS
WHEN , σταθερη_τιμή =>
εντολή ;
[εντολή;]
WHEN , σταθερη_τιμή =>
εντολή ;
[εντολή;]
WHENOTHERS =>
εντολή ;
[εντολή;]
END CASE
	process (d, pr_state)
	begin
	case pr_state is
	when zero =>
	if (d='0') then
	nx_state<= one;
	else
	nx_state<= zero;
	end if;
	
	when one =>
	if (d='1') then
	nx_state<= two;
	else
	nx_state<= one;
	end if;
	.....
	when eight =>
	if (d='1') then
	nx_state<= two;
	else
	nx_state<= one;
	end if;
	end case;
	end process;
	end my_arch;

Υλοποιείστε με χρήση της γλώσσας VHDL την ακόλουθη μηχανή καταστάσεων. Πρόκειται για μια μηχανή, η οποία ελέγχει την αναπαραγωγή διαφορετικών ειδών μουσικής (Rock ή Jazz) σε μία συσκευή αναπαραγωγής μουσικής κατόπιν πληρωμής (τύπου jukebox).



Εικόνα 4.11: Παράδειγμα μηχανής πεπερασμένων καταστάσεων

Η μηχανή βρίσκεται στην κατάσταση αναμονής (idle) και ενεργοποιείται με την εισαγωγή του κατάλληλου κέρματος (το οποίο σηματοδοτείται με την μετάβαση του σήματος Coin σε λογικό 1). Στην κατάσταση Select ο πελάτης επιλέγει το είδος μουσικής που θέλει με το σήμα sel. Στις καταστάσεις Rock και Jazz εκτελείται το αντίστοιχο μουσικό πρόγραμμα και η συσκευή επιστρέφει στην κατάσταση αναμονής idle.



4.8 Βιβλιογραφία

P. Ashenden, "*The Designer's Guide to VHDL*", Morgan Kaufman Publishers, 1996
S. Sjoholm and L. Lennart, "*VHDL for Designers*", Prentice Hall, 1997
B. Cohen, "*VHDL Coding Styles and Methodologies*", Kluwer Academic Publishers, 1999
Z. Navabi, "*VHDL-Analysis and Modeling of Digital Systems*", Mc Graw-Hill, 1993
On-line μαθήματα σε ASIC/VHDL design:
http://www.dacafe.com/ASICs.htm
http://www.ecs.umass.edu/ece/vspgroup/burleson/courses/558/
http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.vhd/html/vhdl-all-e.html
http://www.eas.asu.edu/~yong598d/VHDL_links.html
Tutorials/Papers:
http://www.vhdl.org/fmf/wwwpages/FMF_ECL_models_paper.html
http://www.bluepc.com/download.html#downloadtop
http://www.symphonyeda.com/products.htm
http://www.angelfire.com/electronic/in/vlsi/vhdl.html#vhdl
Άλλες πηγές:
http://www.ieee.org
http://www.acm.org
http://www.vhdl.org
http://tech-www.informatik.uni-hamburg.de/vhdl/
http://www.eeglossary.com/vhdl.htm
http://www.ent.ohiou.edu/~starzyk/network/Class/ee514/intro.html