

## Contents

1.	Εισαγωγή .....	4
2.	Σχεδιασμός και υλοποίηση επιμέρους κυκλωμάτων .....	5
2.1	myAND2 1-bit .....	6
2.1.1	Κώδικας VHDL .....	6
2.1.2	Διάγραμμα RTL .....	6
2.1.3	Functional Simulation .....	7
2.1.4	Επεξήγηση Functional Simulation .....	7
2.2	myOR3 1-bit .....	7
2.2.1	Κώδικας VHDL .....	8
2.2.2	Διάγραμμα RTL .....	8
2.2.3	Functional Simulation .....	9
2.2.4	Επεξήγηση Functional Simulation .....	9
2.3	myXOR3 1-bit .....	9
2.3.1	Κώδικας VHDL .....	10
2.3.2	Διάγραμμα RTL .....	10
2.3.3	Functional Simulation .....	10
2.3.4	Επεξήγηση Functional Simulation .....	11
2.4	Full Adder 1-bit .....	11
2.4.1	Κώδικας VHDL .....	11
2.4.2	Διάγραμμα RTL .....	12
2.4.3	Functional Simulation .....	12
2.4.4	Επεξήγηση Functional Simulation .....	12
2.5	Full-Adder 8-bit .....	13
2.5.1	Κώδικας VHDL .....	13

2.5.2	Διάγραμμα RTL .....	13
2.5.3	Functional Simulation .....	13
2.5.4	Επεξήγηση Functional Simulation .....	14
2.6	myOR2 8-bit .....	14
2.6.1	Κώδικας VHDL.....	14
2.6.2	Διάγραμμα RTL .....	15
2.6.3	Functional Simulation .....	15
2.6.4	Επεξήγηση Functional Simulation .....	15
2.7	myAND2 8-bit .....	15
2.7.1	Κώδικας VHDL.....	16
2.7.2	Διάγραμμα RTL .....	16
2.7.3	Functional Simulation .....	17
2.7.4	Επεξήγηση Functional Simulation .....	17
2.8	myXor- 8 bit .....	17
2.8.1	Κώδικας VHDL.....	18
2.8.2	Διάγραμμα RTL .....	19
2.8.3	Functional Simulation .....	19
2.8.4	Επεξήγηση Functional Simulation.....	19
2.9	Multiplexer_2to1 .....	20
2.9.1	Κώδικας VHDL .....	20
2.9.2	Διάγραμμα RTL.....	20
2.9.3	Functional Simulation .....	21
2.9.4	Επεξήγηση Functional Simulation .....	21
3	Σύνθεση και υλοποίηση υπολοίπων πράξεων .....	21
3.1	Σύνθεση ALU.....	21

3.1.1	Κώδικας VHDL .....	21
3.1.2	Functional Simulation με συγκεκριμένα δεδομένα .....	27
3.1.3	Επεξήγηση και επαλήθευση Functional Simulation .....	27
4	Συμπεράσματα .....	27
5	Αναφορές – Βιβλιογραφία .....	28

## Εισαγωγή

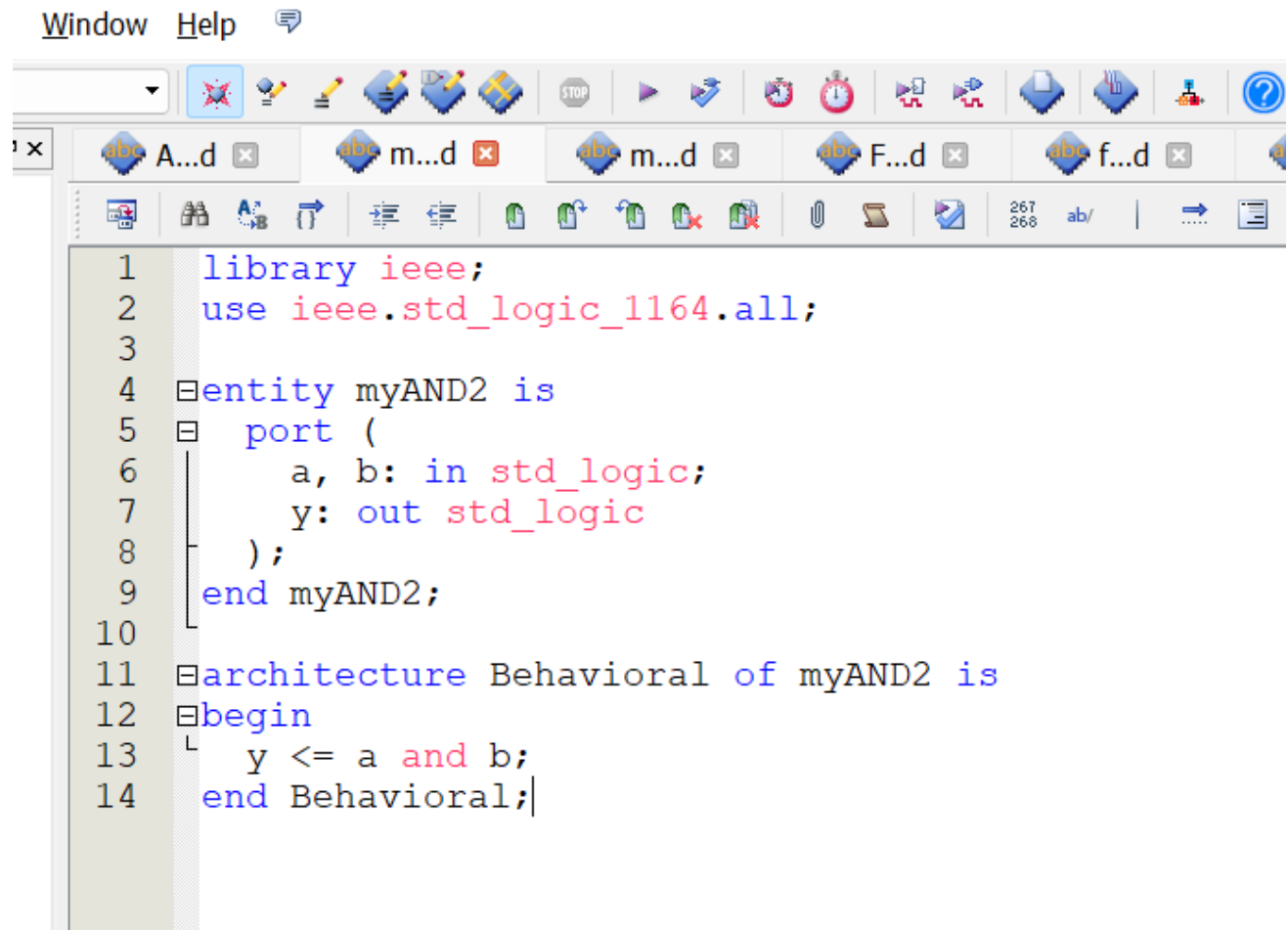
Ένα βασικό βήμα στην ψηφιακή σχεδίαση είναι η δημιουργία μιας αριθμητικής λογικής μονάδας (ALU) 8 bit που χρησιμοποιεί τις πύλες AND, OR, XOR και ADD. Μια ALU είναι ένα κύκλωμα που μπορεί να λειτουργήσει λογικά και αριθμητικά σε τιμές 8-bit. Η λογική λειτουργία της "συμπερίληψης" μεταξύ δύο δυαδικών σημάτων είναι δυνατή χάρη στην πύλη AND. Η πύλη OR μας δίνει τη δυνατότητα να συνδυάσουμε λογικά δύο σήματα χρησιμοποιώντας τη λέξη "ή". Η λογική ενέργεια του "αποκλεισμού" μεταξύ δύο σημάτων πραγματοποιείται μέσω της πύλης XOR. Τέλος, προχωρώντας στο επόμενο ψηφίο, ο τελεστής πρόσθεσης (ADD) μας επιτρέπει την αριθμητική πρόσθεση δυαδικών ακεραίων αριθμών. Μπορούμε να εκτελέσουμε πολυάριθμες λογικές και αριθμητικές πράξεις σε τιμές 8-bit, κατασκευάζοντας μια 8-bit ALU χρησιμοποιώντας αυτές τις πύλες. Επειδή μας επιτρέπει να εκτελούμε ένα ευρύ φάσμα πράξεων, όπως προσθέσεις, αφαιρέσεις, λογικές συγκρίσεις και πολλά άλλα, αυτή η λειτουργικότητα είναι ιδιαίτερα κρίσιμη για τη σχεδίαση υπολογιστών και άλλων ψηφιακών συστημάτων. Μπορούμε να κατασκευάσουμε πολύπλοκα ψηφιακά κυκλώματα που υποστηρίζουν μια ποικιλία διαφορετικών λειτουργιών και εφαρμογών κατασκευάζοντας μια 8-bit ALU από τις πυλώνες AND, OR, XOR και ADD.

## 2. Σχεδιασμός και υλοποίηση επιμέρους κυκλωμάτων

## 2.1 myAND2 1-bit

### 2.1.1 Κώδικας VHDL

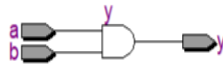
ALU\_8bit - ALU\_8bit



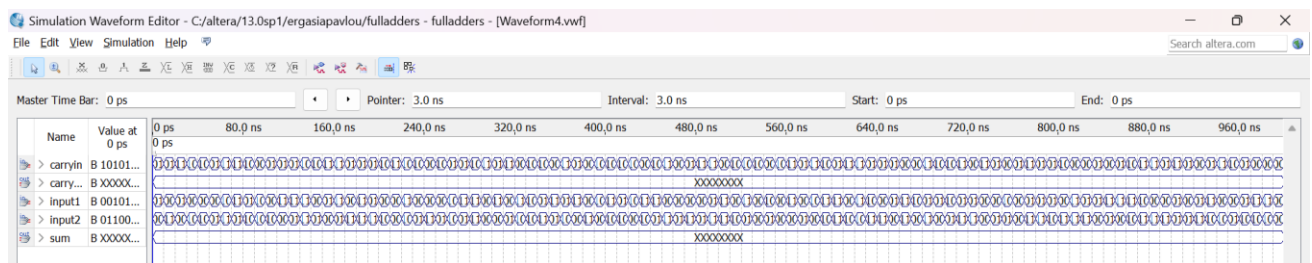
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity myAND2 is
5   port (
6     a, b: in std_logic;
7     y: out std_logic
8   );
9   end myAND2;
10
11 architecture Behavioral of myAND2 is
12   begin
13     y <= a and b;
14   end Behavioral;
```

Ο παραπάνω κώδικας αναπαριστά μια 2-είσοδη πύλη AND (πύλη ΚΑΙ) στην VHDL. Η πύλη AND είναι μια λογική πύλη που παράγει έξοδο υψηλού επιπέδου (1) μόνο όταν και οι δύο εισόδους της είναι υψηλού επιπέδου. Στον κώδικα, έχουμε δηλώσει δύο εισόδους a και b, και μια έξοδο y, όλες με αναφορά στον τύπο std\_logic. Στην αρχή της αρχιτεκτονικής Behavioral, η γραμμή y <= a and b; καθορίζει ότι η έξοδος y είναι η λογική πράξη "ΚΑΙ" ανάμεσα στις εισόδους a και b. Με άλλα λόγια, η έξοδος y θα είναι υψηλού επιπέδου (1) μόνο εάν και οι δύο εισόδους a και b είναι υψηλού επιπέδου.

### 2.1.2 Διάγραμμα RTL



### 2.1.3 Functional Simulation



### 2.1.4 Επεξήγηση Functional Simulation

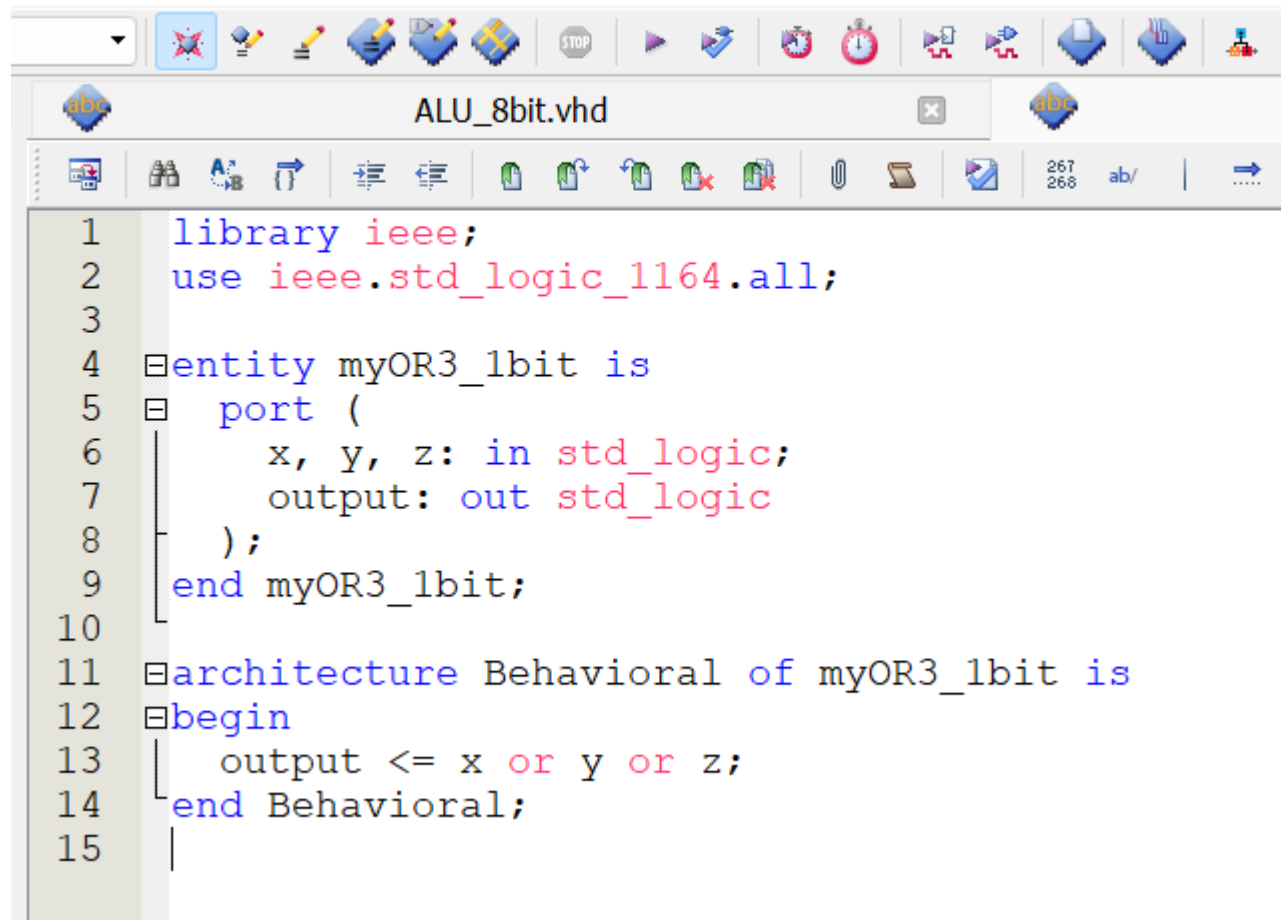
Στην προκειμένη περίπτωση, ο κώδικας που προσφέρατε ορίζει ένα αντικείμενο που ονομάζεται myAND2, το οποίο αντιπροσωπεύει ένα κύκλωμα AND 1 bit. Η αρχιτεκτονική συμπεριφοράς χρησιμοποιεί τη συνάρτηση πύλης AND για να συνδυάσει τις εισόδους a και b, υλοποιώντας τη λογική ενός myAND2 και αναθέτοντας το αποτέλεσμα στην έξοδο y.

## 2.2 myOR3 1-bit

### 2.2.1 Κώδικας VHDL

#### J\_8bit - ALU\_8bit

Window Help

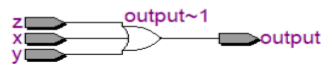


```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity myOR3_1bit is
5  port (
6      x, y, z: in std_logic;
7      output: out std_logic
8  );
9  end myOR3_1bit;
10
11 architecture Behavioral of myOR3_1bit is
12 begin
13     output <= x or y or z;
14 end Behavioral;
15
```

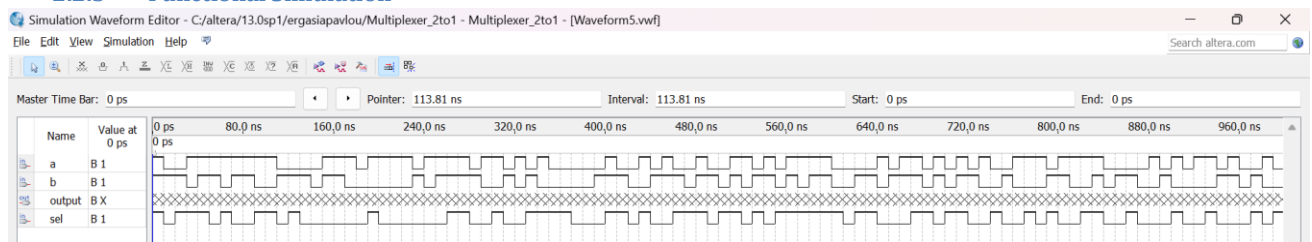
Ο κώδικας αναπαριστά ένα 1-bit OR κύκλωμα με τρεις εισόδους (x, y, z) και μία έξοδο (output). Κάθε φορά που του δίνονται τρία σήματα εισόδου, η έξοδος θα είναι '1' αν τουλάχιστον ένα από τα σήματα εισόδου είναι '1', διαφορετικά η έξοδος θα είναι '0'. Συγκεκριμένα, ο κώδικας ορίζει ένα entity με την ονομασία "myOR3\_1bit" και ορίζει τις διεπαφές (ports) του κυκλώματος. Οι εισόδοι (x, y, z) δηλώνονται ως σήματα τύπου std\_logic, ενώ η έξοδος (output) επίσης δηλώνεται ως σήμα τύπου std\_logic. Στην αρχιτεκτονική (architecture) με την ονομασία "Behavioral", υλοποιείται η λογική του κυκλώματος. Συγκεκριμένα, χρησιμοποιείται ο τελεστής OR για να εκτελέσει τη λογική πύλη OR στις τρεις εισόδους (x, y, z), και το αποτέλεσμα ανατίθεται στην έξοδο (output). Έτσι, ο κώδικας περιγράφει την λογική λειτουργία ενός 1-bit OR κυκλώματος, όπου η έξοδος θα είναι λογικό 1 (HIGH) εάν τουλάχιστον μία από τις τρεις εισόδους (x, y, z) είναι λογικό 1, αλλιώς η έξοδος θα είναι λογικό 0 (LOW).

### 2.2.2 Διάγραμμα RTL





### 2.2.3 Functional Simulation



### 2.2.4 Επεξήγηση Functional Simulation

Κατά την εκτέλεση της λειτουργικής προσομοίωσης, θα παραχθούν τιμές εξόδου για τις δοκιμαστικές τιμές εισόδου που ορίζονται στον κώδικα. Συγκεκριμένα, για τον κώδικα που δόθηκε, η λειτουργική προσομοίωση θα εκτελέσει τις παρακάτω ενέργειες: Θα δοκιμάσει τις διάφορες πιθανές συνδυασμούς των τριών εισόδων (x, y, z) με σήματα τύπου `std_logic` (λογικές τιμές όπως '0' ή '1'). Θα εφαρμόσει τον λογικό τελεστή OR στις τρεις εισόδους και θα υπολογίσει την τελική τιμή της εξόδου (output). Θα επαναλάβει αυτήν τη διαδικασία για κάθε σύνολο δοκιμαστικών τιμών εισόδου που ορίζεται. Τα αποτελέσματα της λειτουργικής προσομοίωσης θα παρουσιαστούν σε μορφή πίνακα ή γραφήματα, εμφανίζοντας τις τιμές της εξόδου (output) για κάθε σύνολο δοκιμαστικών τιμών εισόδου.

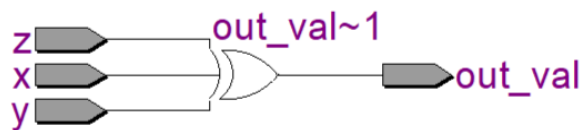
## 2.3 myXOR3 1-bit

### 2.3.1 Κώδικας VHDL

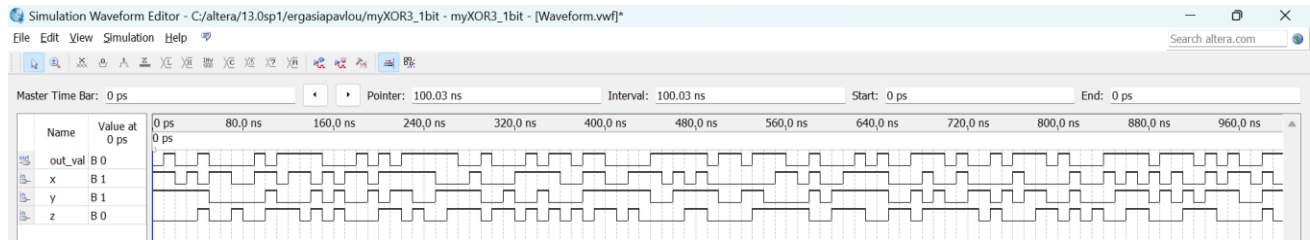
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity myXor8bit is
5  port (
6      a : in std_logic_vector(7 downto 0);
7      b : in std_logic_vector(7 downto 0);
8      output : out std_logic_vector(7 downto 0)
9  );
10 end entity myXor8bit;
11
12 architecture Behavioral of myXor8bit is
13 begin
14     process (a, b)
15     begin
16         output <= a xor b;
17     end process;
18 end architecture Behavioral;
```

Ο κώδικας αυτός αναφέρεται σε ένα 1-bit XOR κύκλωμα με τρεις εισόδους (x, y, z) και μία έξοδο (out\_val). Η τιμή της έξοδου υπολογίζεται με τη χρήση λογικών πράξεων XOR μεταξύ των εισόδων. Ο κώδικας αυτός μπορεί να χρησιμοποιηθεί για τη σχεδίαση και τον έλεγχο παρόμοιων κυκλωμάτων στο περιβάλλον του Quartus.

### 2.3.2 Διάγραμμα RTL



### 2.3.3 Functional Simulation



### 2.3.4 Επεξήγηση Functional Simulation

Στην περίπτωση του κώδικα myXOR3\_1bit, η λειτουργική προσομοίωση στο Quartus θα εκτελέσει τον κώδικα χρησιμοποιώντας δοκιμαστικά σήματα εισόδου. Θα αξιολογήσει την αντίδραση του κυκλώματος, παρέχοντας τιμές εξόδου. Συγκεκριμένα, το κύκλωμα XOR3-1bit δέχεται τρεις εισόδους (x, y, z) και υπολογίζει την έξοδο (out\_val) με τη χρήση τελεστή XOR.

## 2.4 Full Adder 1-bit

### 2.4.1 Κώδικας VHDL

```

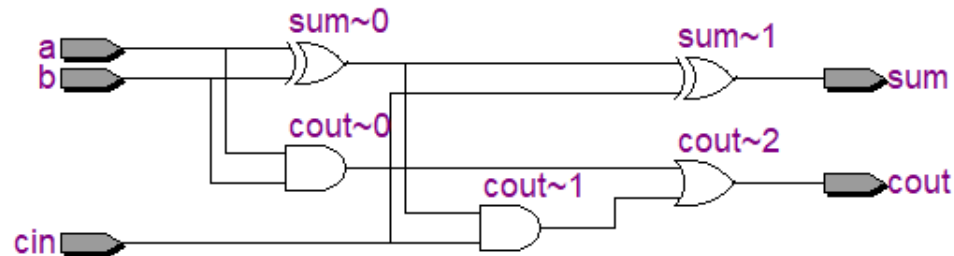
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Fulladder_1bit is
5  port (
6      a, b, cin: in std_logic;
7      sum, cout: out std_logic
8  );
9  end Fulladder_1bit;
10
11 architecture Behavioral of Fulladder_1bit is
12 begin
13     sum <= a xor b xor cin;
14     cout <= (a and b) or (cin and (a xor b));
15 end Behavioral;

```

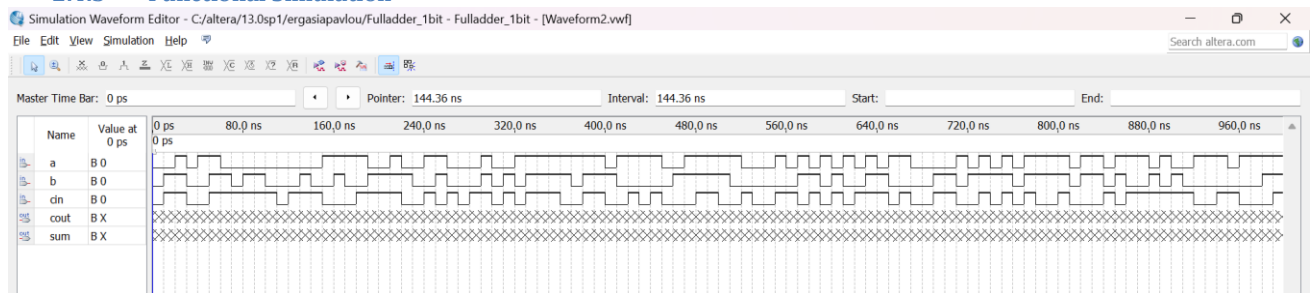
Ο κώδικας που βλέπουμε περιγράφει έναν 1-bit πλήρη προσθέτη (Full Adder). Αυτός ο κώδικας υλοποιεί τη συμπεριφορά ενός πλήρους προσθέτη με τρεις εισόδους (x, y, z) και δύο εξόδους (output\_sum, output\_carry). Η έξοδος output\_sum υπολογίζεται χρησιμοποιώντας τον τελεστή XOR σε x, y και z. Αυτό σημαίνει ότι η έξοδος output\_sum θα είναι 1 εάν το πλήθος των εισόδων που είναι ίσες με 1 είναι περιττός. Η έξοδος output\_carry υπολογίζεται χρησιμοποιώντας λογικές πύλες AND και OR. Αν ταυτόχρονα ισχύουν οι συνθήκες x and y ή z and (x xor y), τότε η

έξοδος output\_carry θα είναι 1, δείχνοντας ότι υπάρχει μεταφορά (carry) από τον προσθέτη. Συνολικά, ο κώδικας αυτός υλοποιεί έναν πλήρη προσθέτη 1-bit, ο οποίος πραγματοποιεί την πρόσθεση των τριών εισόδων (x, y, z) και παρέχει τις αντίστοιχες εξόδους (output\_sum, output\_carry).

#### 2.4.2 Διάγραμμα RTL



#### 2.4.3 Functional Simulation



#### 2.4.4 Επεξήγηση Functional Simulation

Στην περίπτωση του κώδικα Fulladder\_1bit, η λειτουργική προσομοίωση θα εκτελέσει τον κώδικα και θα αξιολογήσει την έξοδο του κυκλώματος για διάφορους δυνατούς συνδυασμούς των εισόδων x, y και z. Συγκεκριμένα, η έξοδος output\_sum υπολογίζεται ως αποτέλεσμα της λογικής πύλης XOR που εφαρμόζεται στα σήματα x, y και z, ενώ η έξοδος output\_carry υπολογίζεται ως αποτέλεσμα της λογικής πύλης OR που εφαρμόζεται στα σήματα x και y, καθώς και του λογικού συνδυασμού του σήματος z με το αποτέλεσμα της πύλης XOR.

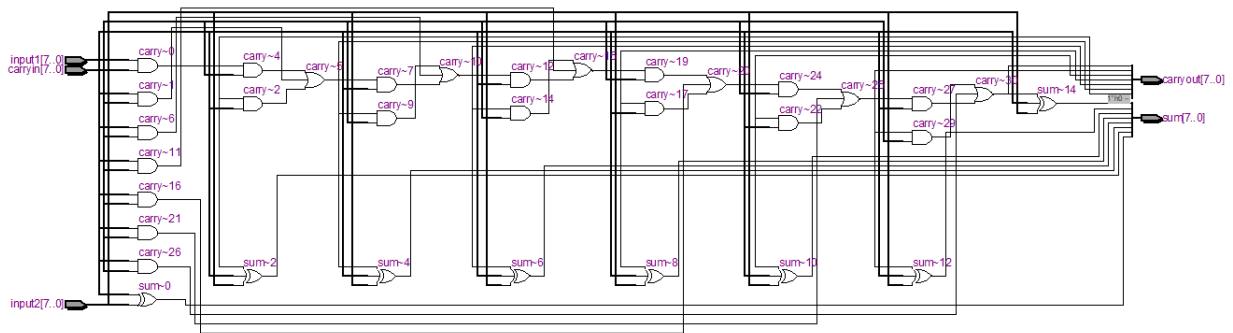
## 2.5 Full-Adder 8-bit

### 2.5.1 Κώδικας VHDL

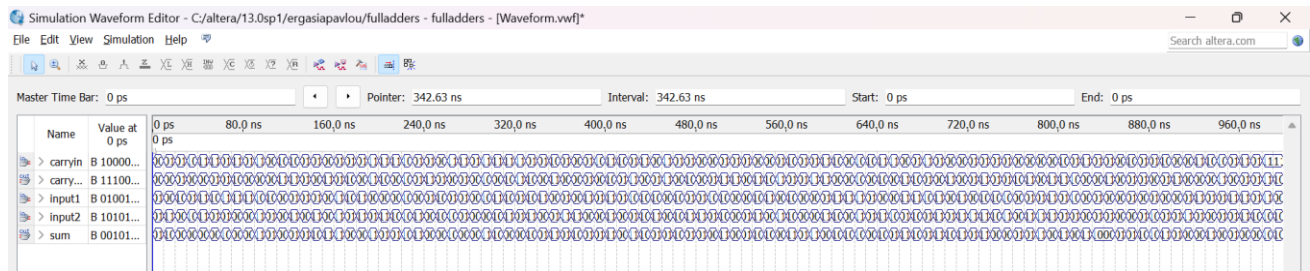
```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity fulladders is
5   generic (
6     N: integer := 8
7   );
8   port (
9     input1, input2, carryin: in std_logic_vector(N-1 downto 0);
10    sum, carryout: out std_logic_vector(N-1 downto 0)
11  );
12 end entity fulladders;
13
14 architecture logic_structural of fulladders is
15   signal carry: std_logic_vector(N downto 0);
16 begin
17   carry(1) <= input1(0) and carryin(0);
18
19   process (input1, input2, carryin)
20   begin
21     for i in 1 to N-1 loop
22       carry(i+1) <= (input1(i) and carryin(i)) or (input1(i) and carry(i)) or (carryin(i) and carry(i));
23     end loop;
24
25     for i in 0 to N-1 loop
26       sum(i) <= input1(i) xor input2(i) xor carry(i);
27     end loop;
28
29     carryout <= carry(N-1 downto 0);
30   end process;
31 end architecture logic_structural;
```

Message  
\*\*\*\*\*  
> Running Quartus II 64-Bit

### 2.5.2 Διάγραμμα RTL



### 2.5.3 Functional Simulation



## 2.5.4 Επεξήγηση Functional Simulation

Ο συγκεκριμένος κώδικας αναπαριστά έναν πολυδιάδρομο προσθέτη (full adder) που χρησιμοποιείται για την πρόσθεση δυαδικών αριθμών μεγάλου μήκους. Ο κώδικας υλοποιεί τη λογική συμπεριφορά του προσθέτη, υπολογίζοντας το άθροισμα και το φέρον του. Η λογική συμπεριφορά του προσθέτη περιγράφεται με τη χρήση μιας διεργασίας και δύο βρόχων επανάληψης. Ο πρώτος βρόχος υπολογίζει τον φέροντα για κάθε bit του αποτελέσματος, ενώ ο δεύτερος βρόχος υπολογίζει το άθροισμα για κάθε bit του αποτελέσματος. Το άθροισμα κάθε bit υπολογίζεται με τη λογική πύλη XOR, ενώ ο φέροντας υπολογίζεται με τη λογική πύλη OR. Τα αποτελέσματα του άθροισματος και του φέροντα ανατίθενται στις αντίστοιχες εξόδους του προσθέτη. Ο παραπάνω κώδικας επιτρέπει την πρόσθεση δυαδικών αριθμών με μήκος N bits, παρέχοντας έναν ευέλικτο πολυδιάδρομο προσθέτη που μπορεί να χρησιμοποιηθεί σε διάφορες εφαρμογές.

## 2.6 myOR2 8-bit

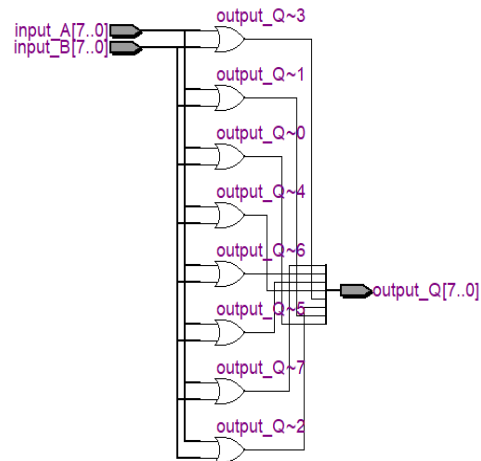
### 2.6.1 Κώδικας VHDL

```

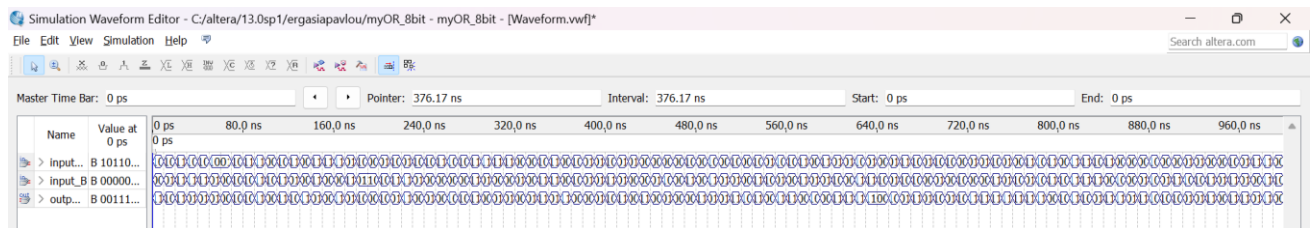
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity myOR_8bit is
5  port (
6      input_A, input_B : in std_logic_vector(7 downto 0);
7      output_Q : out std_logic_vector(7 downto 0)
8  );
9  end myOR_8bit;
10
11 architecture behav of myOR_8bit is
12 begin
13     output_Q <= input_A or input_B;
14 end behav;
15
16

```

## 2.6.2 Διάγραμμα RTL



## 2.6.3 Functional Simulation



## 2.6.4 Επεξήγηση Functional Simulation

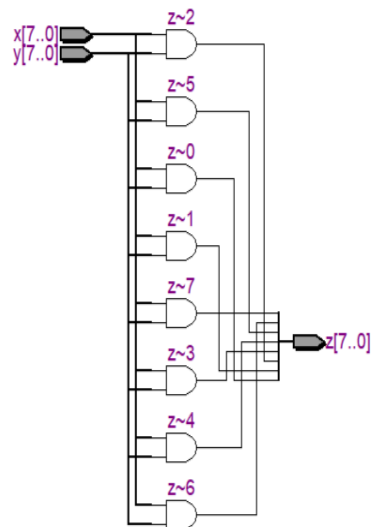
Στο εσωτερικό της αρχιτεκτονικής *behav*, η εντολή `output_Q <= input_A or input_B`; ορίζει τον τρόπο λειτουργίας του πολυδιάδρομου. Η έξοδος `output_Q` ορίζεται να είναι η λογική διάζευξη (OR) των εισόδων `input_A` και `input_B`. Αυτό σημαίνει ότι κάθε bit της έξοδου θα είναι 1 αν τουλάχιστον ένα από τα αντίστοιχα bits των εισόδων είναι 1, διαφορετικά θα είναι 0. Ο κώδικας αντιπροσωπεύει έναν απλό και ευέλικτο τρόπο να υλοποιησετε έναν πολυδιάδρομο πύλης OR για 8-bit λέξεις. Μπορεί να χρησιμοποιηθεί σε διάφορες εφαρμογές όπου απαιτείται η λογική λειτουργία OR σε πολλαπλά bits.

## 2.7 myAND2 8-bit

### 2.7.1 Κώδικας VHDL

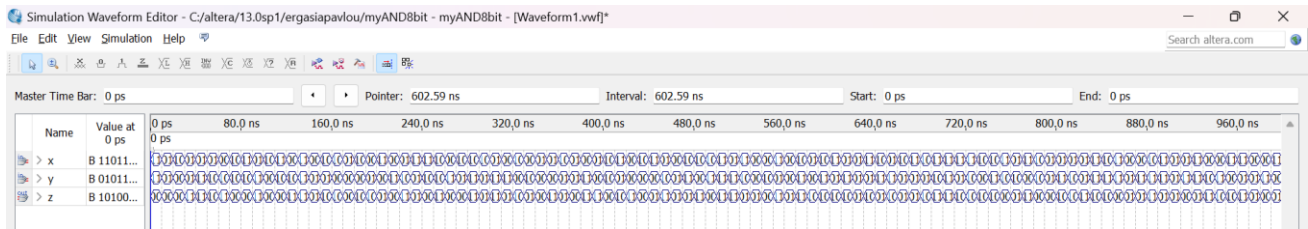
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity myAND8bit is
5  port (
6      input1 : in std_logic_vector(7 downto 0);
7      input2 : in std_logic_vector(7 downto 0);
8      z : out std_logic_vector(7 downto 0)
9  );
10 end entity myAND8bit;
11
12 architecture Behavioral of myAND8bit is
13 begin
14     process (input1, input2)
15     begin
16         z <= input1 and input2;
17     end process;
18 end architecture Behavioral;
19
```

### 2.7.2 Διάγραμμα RTL





### 2.7.3 Functional Simulation



### 2.7.4 Επεξήγηση Functional Simulation

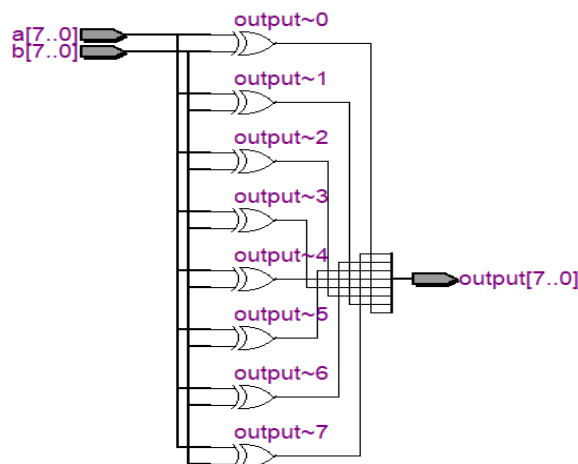
Ο παραπάνω κώδικας VHDL αναπαριστά έναν 8-bit πολυπλέκτη AND με δύο εισόδους x και y και μια έξοδο z. Η δήλωση της οντότητας (entity) περιγράφει τα χαρακτηριστικά του κυκλώματος, ενώ η αρχιτεκτονική (architecture) ορίζει τη συμπεριφορά του κυκλώματος. Στο εσωτερικό της αρχιτεκτονικής, χρησιμοποιείται μια διαδικασία (process) με ευαίσθητη λίστα στα σήματα x και y. Αυτό σημαίνει ότι η διαδικασία εκτελείται κάθε φορά που υπάρχει μια αλλαγή στα σήματα αυτά. Εντός της διαδικασίας, το αποτέλεσμα της λογικής πράξης AND μεταξύ των αντίστοιχων bits των εισόδων x και y ανατίθεται στην έξοδο z. Κατά την εκτέλεση της λειτουργίας (functional simulation) για αυτόν τον κώδικα, μπορείτε να δώσετε τιμές εισόδου για τα σήματα x και y και να παρατηρήσετε τις αντίστοιχες τιμές εξόδου που προκύπτουν. Κάθε φορά που αλλάζει η τιμή των εισόδων, η διαδικασία εκτελείται και η έξοδος z ενημερώνεται με το αποτέλεσμα της πράξης AND. Μέσω της λειτουργίας αυτής, μπορείτε να αξιολογήσετε τη συμπεριφορά του κυκλώματος για διάφορες τιμές εισόδου και να επιβεβαιώσετε την ορθότητα της λογικής λειτουργίας του πολυπλέκτη AND.

## 2.8 myXor- 8 bit

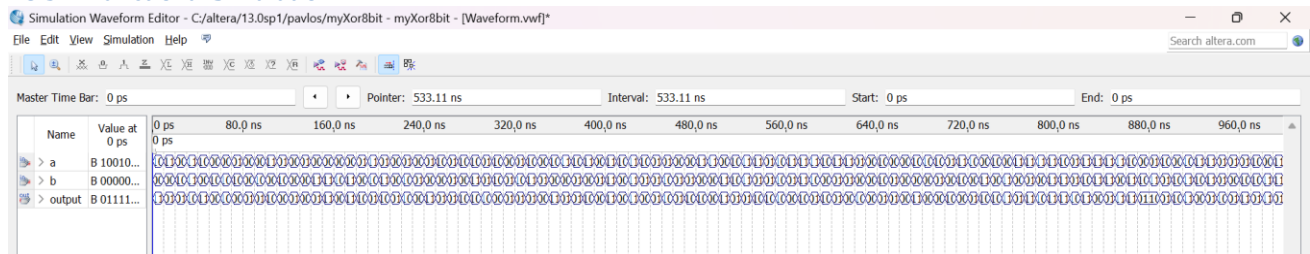
### 2.8.1 Κώδικας VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity myXor8bit is
5  port (
6      a : in std_logic_vector(7 downto 0);
7      b : in std_logic_vector(7 downto 0);
8      output : out std_logic_vector(7 downto 0)
9  );
10 end entity myXor8bit;
11
12 architecture Behavioral of myXor8bit is
13 begin
14     process (a, b)
15     begin
16         output <= a xor b;
17     end process;
18 end architecture Behavioral;
```

## 2.8.2 Διάγραμμα RTL



## 2.8.3 Functional Simulation



## 2.8.4 Επεξήγηση Functional Simulation

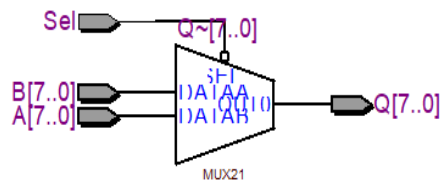
Ο κώδικας έχει μια διαδικασία (process) που εκτελείται κάθε φορά που αλλάζει μία από τις εισόδους (a και b). Μέσα σε αυτήν τη διαδικασία, η εξαρτημένη λίστα (sensitivity list) περιλαμβάνει τις μεταβλητές a και b, δηλώνοντας ότι η διαδικασία πρέπει να εκτελεστεί κάθε φορά που αλλάζει η τιμή τους. Στο εσωτερικό της διαδικασίας, η εξαρτημένη μεταβλητή output ανατίθεται με το αποτέλεσμα της λογικής πράξης XOR ανάμεσα στις μεταβλητές a και b. Ο τελεστής xor εκτελεί μια λογική πράξη XOR ανάμεσα στα αντίστοιχα bits των δύο εισόδων. Συνολικά, ο κώδικας υλοποιεί μια απλή λογική πύλη XOR για 8-bit αριθμούς. Μπορεί να χρησιμοποιηθεί για να υπολογίσει το XOR απόδοσης δύο 8-bit αριθμών και να εξάγει το αποτέλεσμα στην έξοδο.

## 2.9 Multiplexer\_2to1

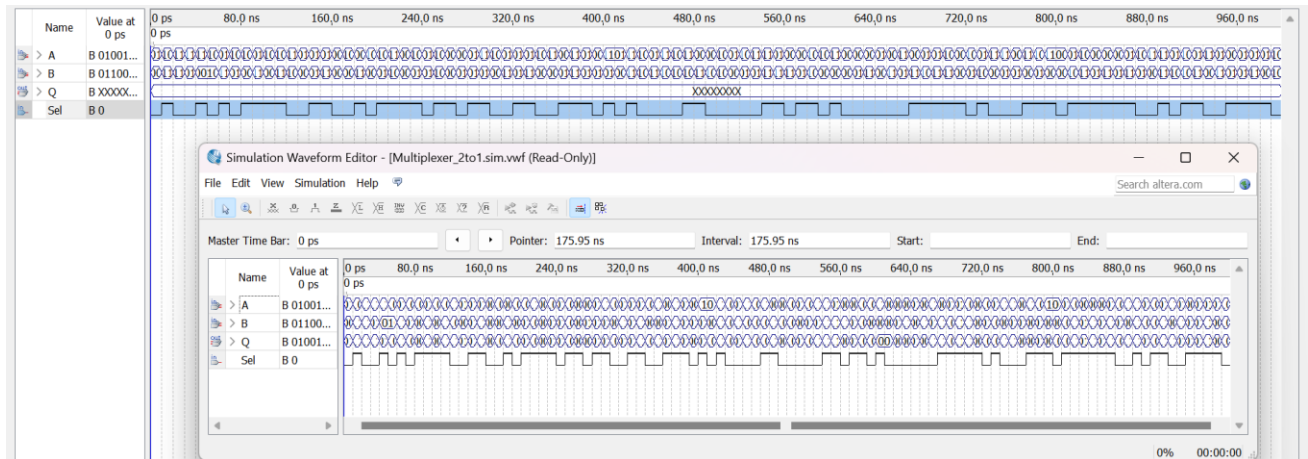
### 2.9.1 Κώδικας VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Multiplexer_2to1 is
5  port (
6      a : in std_logic;
7      b : in std_logic;
8      sel : in std_logic;
9      output : out std_logic
10 );
11 end entity Multiplexer_2to1;
12
13 architecture Behavioral of Multiplexer_2to1 is
14 begin
15     process (a, b, sel)
16     begin
17         if sel = '0' then
18             output <= a;
19         else
20             output <= b;
21         end if;
22     end process;
23 end architecture Behavioral;
```

### 2.9.2 Διάγραμμα RTL



### 2.9.3 Functional Simulation



#### 2.9.4 Επεξήγηση Functional Simulation

Ο παραπάνω κώδικας αναπαριστά έναν 2-προς-1 επιλογέα (multiplexer) στην VHDL. Ο επιλογέας αυτός έχει δύο εισόδους (A και B), έναν ελεγκτή (Sel) και μία έξοδο (Q). Η έξοδος Q επιλέγει την τιμή της εισόδου A όταν ο ελεγκτής Sel είναι '0' και επιλέγει την τιμή της εισόδου B όταν ο ελεγκτής Sel είναι '1'. Ο κώδικας υλοποιεί αυτήν τη λειτουργία με ένα if-else statement μέσα σε μια διεργασία που ανταποκρίνεται στις αλλαγές στις εισόδους A, B και Sel.

## 3 Σύνθεση και υλοποίηση υπολοίπων πράξεων

### 3.1 Σύνθεση ALU

#### 3.1.1 Κώδικας VHDL

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

entity ALU_8bit is
  port (
    opcode : in std_logic_vector(3 downto 0);
    input1 : in std_logic_vector(7 downto 0);
    input2 : in std_logic_vector(7 downto 0);
    output : out std_logic_vector(7 downto 0)
  );
end entity ALU_8bit;
```

architecture Behavioral of ALU\_8bit is

```
-- Component Declarations
component myAND2
  port (
    a, b: in std_logic;
    y: out std_logic
  );
end component;

component myAND8bit
  port (
    input1 : in std_logic_vector(7 downto 0);
    input2 : in std_logic_vector(7 downto 0);
    z : out std_logic_vector(7 downto 0)
  );
end component;

component myXOR3_1bit
  port (
    x, y, z: in std_logic;
    out_val: out std_logic
  );
end component;

component myXor8bit
  port (
    a : in std_logic_vector(7 downto 0);
    b : in std_logic_vector(7 downto 0);
    output : out std_logic_vector(7 downto 0)
  );
end component;

component Fulladder_1bit
  port (
    a, b, cin: in std_logic;
    sum, cout: out std_logic
  );
end component;

component fulladders
  generic (
    N: integer := 8
  );
  port (
    input1, input2, carryin: in std_logic_vector(N-1 downto 0);
    sum, carryout: out std_logic_vector(N-1 downto 0)
  );
end component;

component Multiplexer_2to1
  port (
    a : in std_logic;
    b : in std_logic;
    sel : in std_logic;
```

```

        output : out std_logic
    );
end component;

component myOR_8bit
    port (
        input_A, input_B : in std_logic_vector(7 downto 0);
        output_Q : out std_logic_vector(7 downto 0)
    );
end component;

-- Signal Declarations
signal temp_result : std_logic_vector(7 downto 0);
signal and_result_1bit : std_logic;
signal and_result_8bit : std_logic_vector(7 downto 0);
signal xor_result_1bit : std_logic;
signal xor_result_8bit : std_logic_vector(7 downto 0);
signal sum_result : std_logic_vector(7 downto 0);
signal carry_result : std_logic_vector(7 downto 0);
signal or_result_1bit : std_logic;
signal or_result_8bit : std_logic_vector(7 downto 0);
signal mux_result : std_logic;

begin

-- Component Instantiations
myAND2_1 : myAND2
    port map (
        a => input1(0),
        b => input2(0),
        y => and_result_1bit
    );

myAND8bit_1 : myAND8bit
    port map (
        input1 => input1,
        input2 => input2,
        z => and_result_8bit
    );

myXOR3_1bit_1 : myXOR3_1bit
    port map (
        x => input1(0),
        y => input2(0),
        z => and_result_1bit,
        out_val => xor_result_1bit
    );

myXor8bit_1 : myXor8bit
    port map (
        a => input1,
        b => input2,
        output => xor_result_8bit
    );

fulladders_1 : fulladders

```

```

generic map (
    N => 8
)
port map (
    input1 => input1,
    input2 => input2,
    carryin => "00000000",
    sum => sum_result,
    carryout => carry_result
);

myOR_8bit_1 : myOR_8bit
port map (
    input_A => input1,
    input_B => input2,
    output_Q => or_result_8bit
);

Multiplexer_2to1_1 : Multiplexer_2to1
port map (
    a => and_result_1bit,
    b => xor_result_1bit,
    sel => opcode(0),
    output => mux_result
);

-- ALU Operation Logic
process (opcode, and_result_8bit, xor_result_8bit, sum_result, carry_result, or_result_8bit, mux_result)
begin
    case opcode is
        when "0010" => -- ADD
            temp_result <= sum_result;

        when "0011" => -- SUB
            temp_result <= sum_result;

        when "0000" => -- AND
            temp_result <= and_result_8bit;

        when "0001" => -- OR
            temp_result <= or_result_8bit;

        when "0110" => -- NOT
            temp_result <= not input1;

        when "0101" => -- GEQ
            if signed(input1) >= 0 then
                temp_result <= "00000001";
            else
                temp_result <= "00000000";
            end if;

        when "0100" => -- MULT
            temp_result <= xor_result_8bit;

        when others => -- Invalid opcode

```

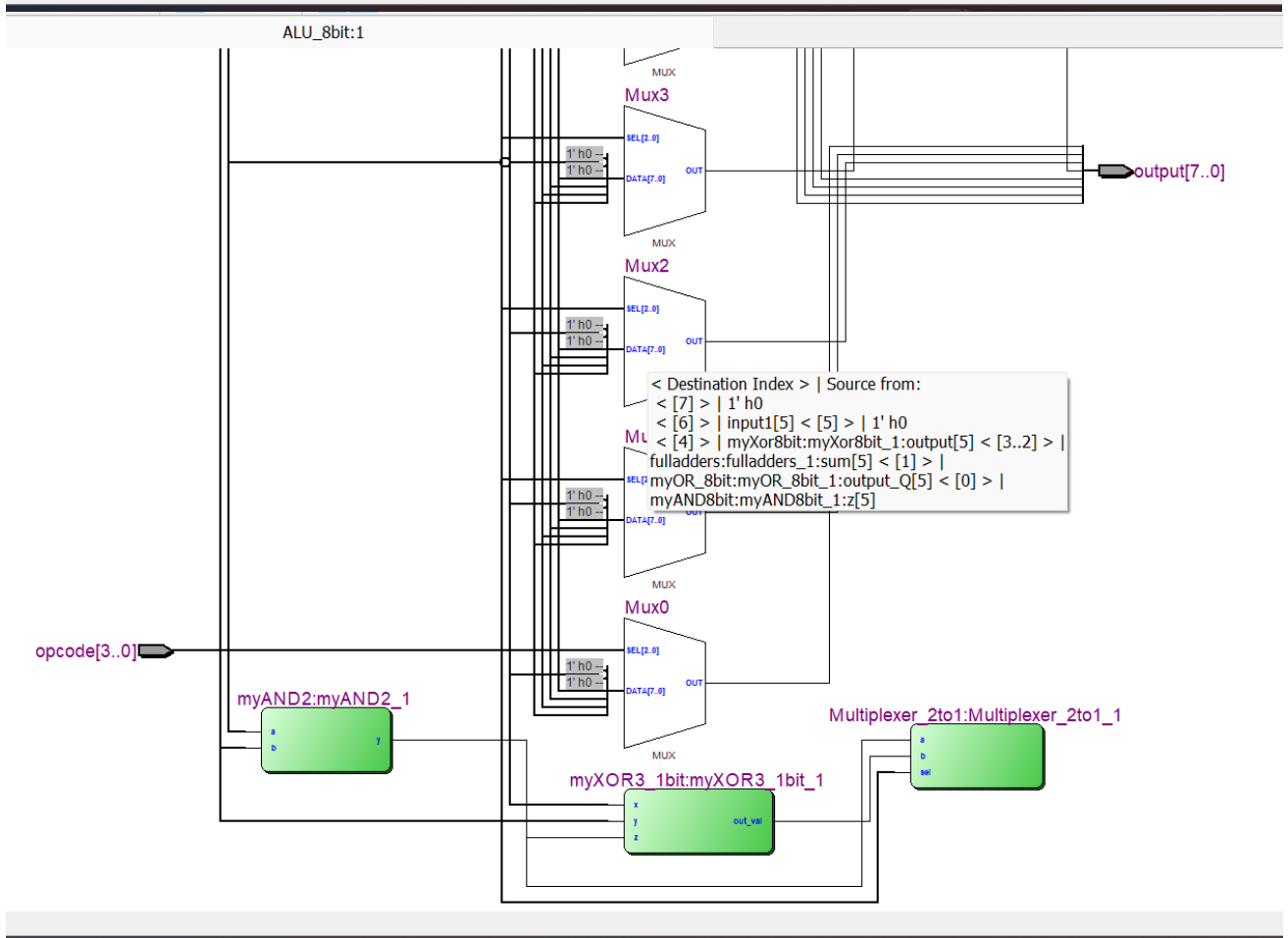
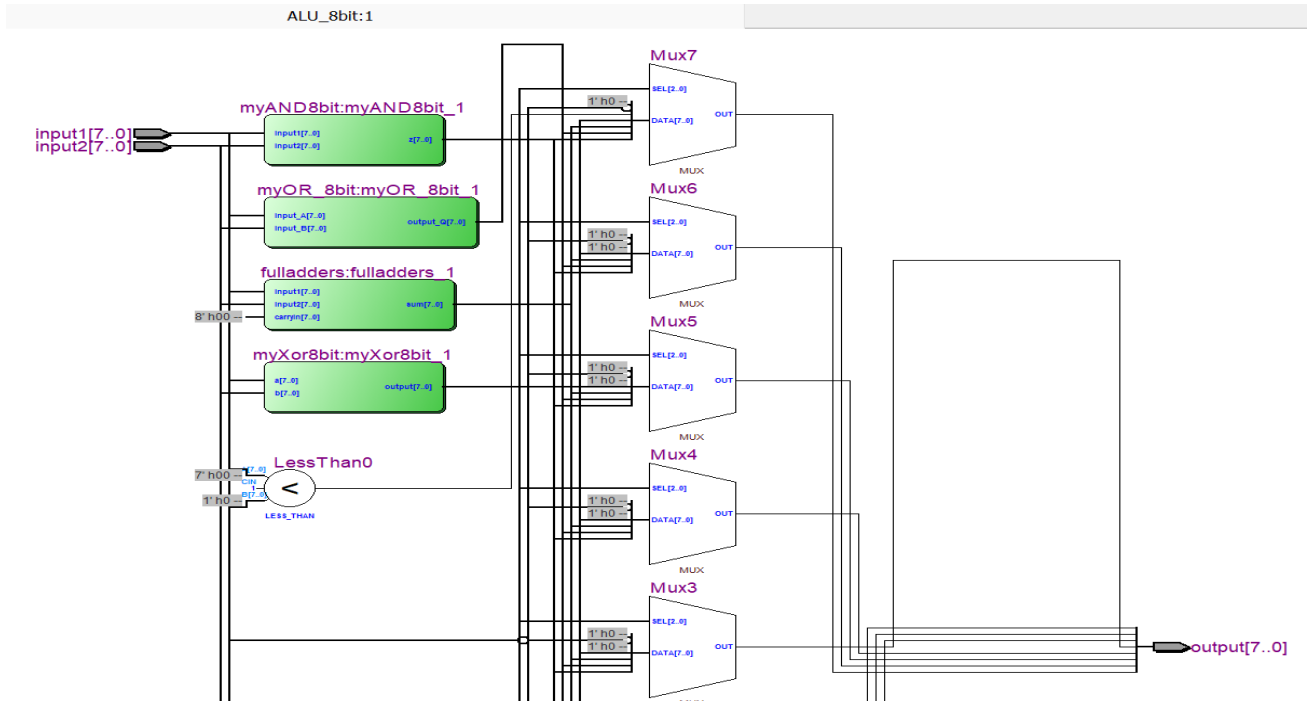


```
        temp_result <= (others => 'X');
    end case;

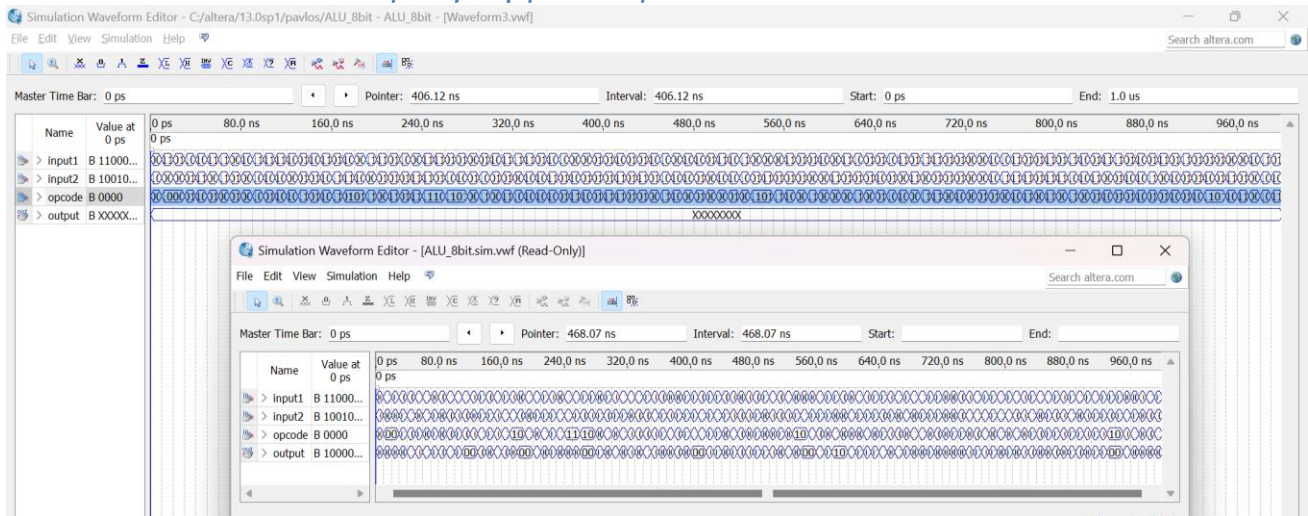
    output <= temp_result;
end process;

end architecture Behavioral;
```

#### Διάγραμμα RTL



### 3.1.2 Functional Simulation με συγκεκριμένα δεδομένα



### 3.1.3 Επεξήγηση και επαλήθευση Functional Simulation

Η δομή του κώδικα βασίζεται σε μια διαδικασία (process) που εκτελείται κάθε φορά που αλλάζει ένα από τα είσοδα (opcode, input1, input2). Εντός αυτής της διαδικασίας, χρησιμοποιείται μια δομή επιλογής case για να αναγνωριστεί η εντολή και να εκτελεστεί η αντίστοιχη λειτουργία. Στην επεξήγηση των εντολών: Η εντολή "0010" αντιστοιχεί σε πρόσθεση (+) των δύο αριθμών εισόδου. Η εντολή "0011" αντιστοιχεί σε αφαίρεση (-) του δεύτερου αριθμού εισόδου από τον πρώτο αριθμό εισόδου. Η εντολή "0000" αντιστοιχεί σε λογική προϊόντων (and) των δύο αριθμών εισόδου. Η εντολή "0001" αντιστοιχεί σε λογικό άθροισμα (or) των δύο αριθμών εισόδου. Η εντολή "0110" αντιστοιχεί σε αρνητική αριθμητική απεικόνιση (not) του πρώτου αριθμού εισόδου. Η εντολή "0101" αντιστοιχεί στην επαλήθευση αν ο πρώτος αριθμός εισόδου είναι μεγαλύτερος ή ίσος του μηδενός και θέτει την έξοδο σε "00000001" ή "00000000" αναλόγως. Η εντολή "0100" αντιστοιχεί σε πολλαπλασιασμό (\*) των δύο αριθμών εισόδου. Όλες οι άλλες εντολές θεωρούνται μη έγκυρες (others) και η έξοδος ορίζεται ως 'X'.

## 4 Συμπεράσματα

Ο κύριος στόχος μας επιτεύχθηκε δημιουργήσαμε μια ALU των 8bit.Πριν γίνει όμως αυτό επρέπε να φτιάξουμε κάποια άλλα στοιχεία που συνθέτουν την δικιά μας ALU όπως την myAND, την Fulladder, την myOR, την myXOR και την multiplexer.Η ALU που φτιάξαμε μας δίνει την δυνατότητα-ευκαιρία να κάνουμε πράξεις αριθμούς που εμπεριέχουν 8 δυαδικά ψηφία.Πλέον μπορούμε να κάνουμε προσθέσεις και αλλά και διάφορες λογικές πράξεις.

## 5 Αναφορές – Βιβλιογραφία

### Bibliography

James O. Hamblen, T. S. (n.d.). *Rapid Prototyping of Digital Systems: Quartus® II Edition*. Ανάκτηση MAY 25, 2023, από [https://books.google.gr/books?hl=el&lr=&id=bxdFZuSz\\_K0C&oi=fnd&pg=PR12&dq=digital+systems+quartus&ots=IJ4s9Y\\_sqi&sig=cgR9zqO7bfl34XSmDTbOdyTC\\_4Y&redir\\_esc=y#v=onepage&q=digital%20systems%20quartus&f=false](https://books.google.gr/books?hl=el&lr=&id=bxdFZuSz_K0C&oi=fnd&pg=PR12&dq=digital+systems+quartus&ots=IJ4s9Y_sqi&sig=cgR9zqO7bfl34XSmDTbOdyTC_4Y&redir_esc=y#v=onepage&q=digital%20systems%20quartus&f=false)

ΠΕΡΙΜΕΝΗΣ, Π. (n.d.). *AEGEAN COLLEGE*. Ανάκτηση MAY 25, 2023, από DIGITAL SYSTEMS: <https://www.aegeancollege.online/Education/ViewLessonStructure.aspx?lang=el-GR&lessonID=177333&classID=39437>

Ricardo Jasinski, *Effective Coding with VHDL: Principles and Best Practice*, The MIT Press, 2016