ΑΡΙΘΜΗΤΙΚΗ & ΛΟΓΙΚΗ ΜΟΝΑΔΑ - ALU

2

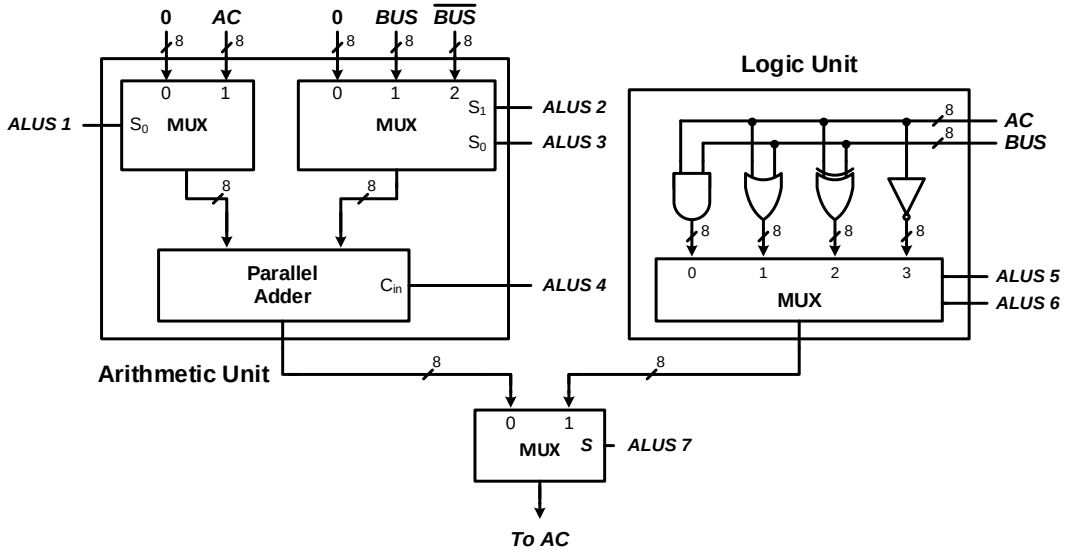
Σκοπός

Με αφορμή την σχεδίαση και την εξομοίωση με διάφορους τρόπους, απλών ψηφιακών κυκλωμάτων θα κατακτηθεί το αντικείμενο της άσκησης αυτής που είναι η σχεδίαση μιας αριθμητικής λογική μονάδας (ALU) η οποία θα χρησιμοποιηθεί κατά την τελική σύνθεση της σχετικά απλής ΚΜΕ.

Συνοπτικά οι λειτουργίες που θα πρέπει να υλοποιεί η ALU είναι δέκα (10) συνολικά, οι οποίες μπορούν να χωριστούν σε τρεις κατηγορίες :

- Λογικές πράξεις μεταξύ δύο αριθμών των 8-bits : **AND, OR, XOR**
- Αριθμητικές πράξεις μεταξύ δύο αριθμών των 8-bits : **ADD, SUB**
- Διεργασίες που αφορούν το συσσωρευτή : **NOT, CLAC, INAC, MOVR, LDAC**

Το Σχήμα 1 απεικονίζει αναλυτικά την ALU.



Σχήμα 1: *Λογικό διάγραμμα Αριθμητικής & Λογικής Μονάδας.*

Για την υλοποίησή της ALU θα γραφεί κώδικας περιγραφής υλικού σε VHDL για κάθε υποκύκλωμα της ξεχωριστά και στη συνέχεια αυτά να χρησιμοποιηθούν σαν στοιχεία (components) για τη σύνθεση του τελικού κώδικα. Τα στοιχεία που απαιτούνται βάσει του παραπάνω σχήματος (**Σχήμα**

1), είναι τέσσερις (4) πολυπλέκτες και ένας παράλληλος αθροιστής (parallel adder) των 8-bits με αρχικό κρατούμενο. Για τις λογικές συναρτήσεις δεν απαιτείται η συγγραφή ξεχωριστού στοιχείου αφού αποτελούν τελεστές της γλώσσας VHDL.

Παράλληλος Αθροιστής

Η υλοποίηση σε VHDL του παράλληλου αθροιστή των 8-bits θα γίνει με το structural μοντέλο περιγραφής. Αυτό σημαίνει ότι θα πρέπει αρχικά να γραφεί κώδικας για το στοιχείο του πλήρη αθροιστή, το οποίο στη συνέχεια θα κληθεί οκτώ (8) φορές στο πρόγραμμα περιγραφής του παράλληλου αθροιστή των 8-bits.

Γράψτε τον κώδικα για τον πλήρη αθροιστή με σήματα εισόδου a, b, c_{in} και σήματα εξόδου s και c_{out} εύρους 1 bit. Το κύκλωμα αυτό όπως είναι γνωστό ότι υλοποιεί την αριθμητική πράξη της πρόσθεση ανάμεσα στις εισόδου (a) και (b) λαμβάνοντας υπόψιν και την τιμή του κρατουμένου εισόδου (c_{in}) και υπολογίζει το άθροισμα (s) και το κρατούμενο εξόδου(c_{out}). Να δώσετε όνομα στην entity adder1bit.

Γράψτε εδώ το πρόγραμμά σας:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY adder1bit IS

PORT (a, b: IN STD_LOGIC;

cin: IN STD_LOGIC;

s: OUT STD_LOGIC;

cout: OUT STD_LOGIC);

END adder1bit;

-------------------------------------------------

ARCHITECTURE adder_rtl OF adder1bit IS

SIGNAL xor_out, and1_out, and2_out : STD_LOGIC; -- δήλωση των εσωτερικών σημάτων

BEGIN

xor_out <= a XOR b; -- Περιγραφή του εσωτερικού σήματος xor_out

and1_out <= a AND b; -- Περιγραφή του εσωτερικού σήματος and1_out

and2_out <= cin AND xor_out; -- Περιγραφή του εσωτερικού σήματος and2_out

s <= xor_out XOR cin; -- Περιγραφή της εξόδου του αθροίσματος S

cout <= and1_out OR and2_out; -- Περιγραφή του κρατούμενου εξόδου Co

END adder_rtl;
```

Πρόγραμμα 1: *Ο πλήρης αθροιστής.*

Χρησιμοποιώντας το παραπάνω κύκλωμα του πλήρη αθροιστή γράψτε τον κώδικα περιγραφής για τον παράλληλο αθροιστή των 8 bit. Να δώσετε όνομα στην entity adder8bit.

ΣΗΜΕΙΩΣΗ *Μπορείτε να κάνετε χρήση της εντολής generate για να καλέσετε το κύκλωμα του πλήρη αθροιστή 8 φορές.*

Γράψτε εδώ το πρόγραμμά σας:

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


ENTITY adder8bit IS
GENERIC(NUM_BITS: natural :=8);
PORT (a, b: IN STD_LOGIC_VECTOR(NUM_BITS-1 DOWNTO 0);
    cin: IN STD_LOGIC;
    s: OUT STD_LOGIC_VECTOR(NUM_BITS-1 DOWNTO 0;
    cout: OUT STD_LOGIC);
END adder8bit;
-----------------------------------------------
ARCHITECTURE strl OF adder8bit IS
SIGNAL carry: STD_LOGIC_VECTOR(0 TO NUM_BITS);
BEGIN
carry(0)<=cin;
gen_adder: FOR i IN 0 TO NUM_BITS-1 GENERATE
adder: ENTITY work.adder1bit
PORT MAP (a(i),b(i),carry(i),s(i),carry(i+1));
END GENERATE;
cout<=carry(NUM_BITS);
END strl;
```

Πρόγραμμα 2: *Ο παράλληλος αθροιστής.*

Πολυπλέκτες

Οι πολυπλέκτες που θα χρησιμοποιηθούν είναι τέσσερις συνολικά, από τους οποίους οι δύο πρώτοι αφορούν την αριθμητική μονάδα (Arithmetic Unit) και ο τρίτος την Λογική μονάδα (Logic Unit). Πιο αναλυτικά θα χρησιμοποιηθούν: α) ένας πολυπλέκτης 2-σε-1 για την επιλογή είτε του περιεχομένου του AC, είτε του σήματος "00000000", β) ένας πολυπλέκτης 3-σε-1 για την επιλογή ενός από τα τρία σήματα BUS, BUS' και "00000000", γ) ένας πολυπλέκτης 4-σε-1 για την επιλογή λογικής πράξης (and, or, xor και not) και δ) ένας πολυπλέκτης 2-σε-1 για την επιλογή μιας από τις εξόδους, είτε της Αριθμητικής είτε της Λογικής μονάδας.
Θα πρέπει να σημειωθεί ότι ο πολυπλέκτης 3-σε-1 στην πραγματικότητα είναι 4-σε-1, αφού για την επιλογή ενός από τρία σήματα εισόδου απαιτούνται δύο σήματα ελέγχου. Επιπλέον, οι πολυπλέκτες 4-σε-1 και 3-σε-1 είναι πανομοιότυποι (4 είσοδοι των 8-bits & 2 σήματα ελέγχου), όπως και οι δύο πολυπλέκτες 2-σε-1 (2 είσοδοι των 8-bits & 1 σήμα ελέγχου). Αυτό σημαίνει ότι θα μπορούσε να γραφεί κώδικας μόνο για αυτά τα δύο είδη πολυπλεκτών, τα οποία θα χρησιμοποιούνταν δύο φορές το καθένα. Να δώσετε όνομα στις entities mux4 και mux2 αντίστοιχα.

Γράψτε τον κώδικα για ένα πολυπλέκτη 2 σε 1.

Γράψτε εδώ το πρόγραμμά σας:

```
LIBRARY ieee ;

USE ieee.std_logic_1164.all ;


ENTITY mux2 IS

PORT (input1, input2: IN STD_LOGIC ;

sel: IN STD_LOGIC ;

output: OUT STD_LOGIC ) ;

END mux2 ;


ARCHITECTURE structural OF mux2 IS

BEGIN

output <= input1 WHEN sel = '0' ELSE

input2 WHEN sel = '1';

END structural ;
```

Πρόγραμμα 3: *Ο πολυπλέκτης 2 σε 1.*

Γράψτε τον κώδικα για ένα πολυπλέκτη 4 σε 1.

Γράψτε εδώ το πρόγραμμά σας:

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

ENTITY mux4 IS
PORT (input1, input2, input3, input4: IN STD_LOGIC ;
sel: IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
output: OUT STD_LOGIC ) ;
END mux4 ;

ARCHITECTURE structural OF mux4 IS
PROCESS(input1, input2, input3, input4, sel)
BEGIN
CASE sel IS
WHEN "00" => output <= input1;
WHEN "01" => output <= input2;
WHEN "10" => output <= input3;
WHEN "11" => output <= input4;
WHEN others => output <='0';
END CASE;
END PROCESS;
END structural;
```

Πρόγραμμα 4: *Ο πολυπλέκτης 4 σε 1.*

Αριθμητική & Λογική Μονάδα

Έχοντας ολοκληρώσει τη συγγραφή του κώδικα για τα επιμέρους στοιχεία που συνθέτουν την ALU και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γραφεί το συνολικό πρόγραμμα περιγραφής της. Τα σήματα ελέγχου της (ALUS 1, ALUS 2, … ALUS 7), θεωρούνται σαν μία ψηφιολέξη των 7-bits, όπου κάθε ένα bit αντιστοιχεί σε ένα σήμα από αυτά.

Γράψτε τον κώδικα για τη βιβλιοθήκη (package), με το όνομα alulib, η οποία θα περιέχει τα επιμέρους στοιχεία που συνθέτουν την ALU.

Γράψτε εδώ το πρόγραμμά σας:

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

PACKAGE alulib IS

  COMPONENT adder1bit
PORT (a, b: IN STD_LOGIC;
cin: IN STD_LOGIC;
s: OUT STD_LOGIC;
cout: OUT STD_LOGIC);
  END COMPONENT;

  COMPONENT adder8bit
GENERIC(NUM_BITS: natural :=8);
PORT (a, b: IN STD_LOGIC_VECTOR(NUM_BITS-1 DOWNTO 0);
     cin: IN STD_LOGIC;
     s: OUT STD_LOGIC_VECTOR(NUM_BITS-1 DOWNTO 0;
     cout: OUT STD_LOGIC);
  END COMPONENT;

  COMPONENT mux2
PORT (input1, input2: IN STD_LOGIC ;
sel: IN STD_LOGIC ;
output: OUT STD_LOGIC ) ;
  END COMPONENT;

  COMPONENT mux4
PORT (input1, input2, input3, input4: IN STD_LOGIC ;
```

```
sel: IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
output: OUT STD_LOGIC ) ;
  END COMPONENT;

END PACKAGE alulib;
```

Πρόγραμμα 5: *βιβλιοθήκη στοιχείων για την ALU.*

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 6) γράψτε τον κώδικα περιγραφής για ALU

Γράψτε εδώ το πρόγραμμά σας:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.alulib.all;

entity alu is
	generic (n : integer := 8);
	port  (
		ac     : in std_logic_vector(n-1 downto 0);
		db     : in std_logic_vector(n-1 downto 0);
		alus   : in std_logic_vector(7 downto 1);
		dout   : out std_logic_vector(n-1 downto 0)
	);
end alu;

architecture arch of alu is

   signal mux2_out   : std_logic_vector(n-1 downto 0);
   signal mux3_out   : std_logic_vector(n-1 downto 0);
   signal arith_out  : std_logic_vector(n-1 downto 0);
   signal and_out    : std_logic_vector(n-1 downto 0);
   signal or_out     : std_logic_vector(n-1 downto 0);
   signal xor_out    : std_logic_vector(n-1 downto 0);
   signal not_out    : std_logic_vector(n-1 downto 0);
   signal logic_out  : std_logic_vector(n-1 downto 0);
   signal carry      : std_logic;

begin
```

```
--ARITHMETIC UNIT
-- mux2: επιλέγει 0 ή ac με βάση alus1

mux2_gen : for i in 0 to n-1 generate
   m2: entity work.mux2
   port map(
      input1 => '0',         -- 0
      input2 => ac(i),       -- AC
      sel    => alus(1),     -- alus1
      output => mux2_out(i)
   );
end generate;



--mux3 (με mux4): επιλέγει 0, db, db', με alus2=s1, alus3=s0

mux3_gen : for i in 0 to n-1 generate
   m3 : entity work.mux4
   port map(
      input1 => '0',         -- 00 -> 0
      input2 => db(i),       -- 01 -> db
      input3 => not db(i),   -- 10 -> db'
      input4 => '0',         -- 11 -> δεν χρησιμοποιείται
      sel    => alus(3 downto 2), -- s1=alus2, s0=alus3
      output => mux3_out(i)
   );
end generate;
```

```
--adder8bit

arith: entity work.adder8bit
generic map(NUM_BITS => n)
port map(
   a    => mux2_out,
   b    => mux3_out,
   cin  => alus(4),          -- alus4 = carry in
   s    => arith_out,
   cout => carry
);

         --LOGIC UNIT
logic_and:  and_out  <= ac and db;
logic_or:   or_out   <= ac or  db;
logic_xor:  xor_out  <= ac xor db;
logic_not:  not_out  <= not ac;

--MUX4 για επιλογή πράξης logic: alus5=s1, alus6=s0
logic_gen: for i in 0 to n-1 generate
   m4 : entity work.mux4
   port map(
      input1 => and_out(i),
      input2 => or_out(i),
      input3 => xor_out(i),
      input4 => not_out(i),
      sel    => alus(6 downto 5),
```

```
      output => logic_out(i)
    );
  end generate;



  --Τελικός mux2: επιλέγει arithmetic ή logic result,alus7 = select
  ------------------------------------------------------------------
  final_gen: for i in 0 to n-1 generate
    fm : entity work.mux2
    port map(
      input1 => arith_out(i),   -- 0 -> arithmetic
      input2 => logic_out(i),   -- 1 -> logic
      sel    => alus(7),
      output => dout(i)
    );
  end generate;


end arch;
```

Πρόγραμμα 6: *Αριθμητική & Λογική Μονάδα.*

Εξομοίωση Αριθμητικής & Λογικής Μονάδας

Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της ALU με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Σαν σήματα εισόδου εκτός των σημάτων ελέγχου, θεωρούνται δύο τυχαίοι αριθμοί, οι "01001001" και "10011001" που αντιστοιχούν στις δύο εισόδους της ALU (ac & db). Στον πίνακα που ακολουθεί, παρουσιάζονται τα θεωρητικά αναμενόμενα αποτελέσματα για κάθε μια από τις διεργασίες που σχετίζονται με την ALU και τον AC.

State	ALUS	result	state	ALUS	Result
AND	1000000	00001001	**INAC**	0001001	01001010
OR	1100000	11011001	**ADD**	0000101	11100010
NOT	1110000	10110110	**SUB**	0001011	10110000
XOR	1010000	11010000	**MOV**	0000100	10011001

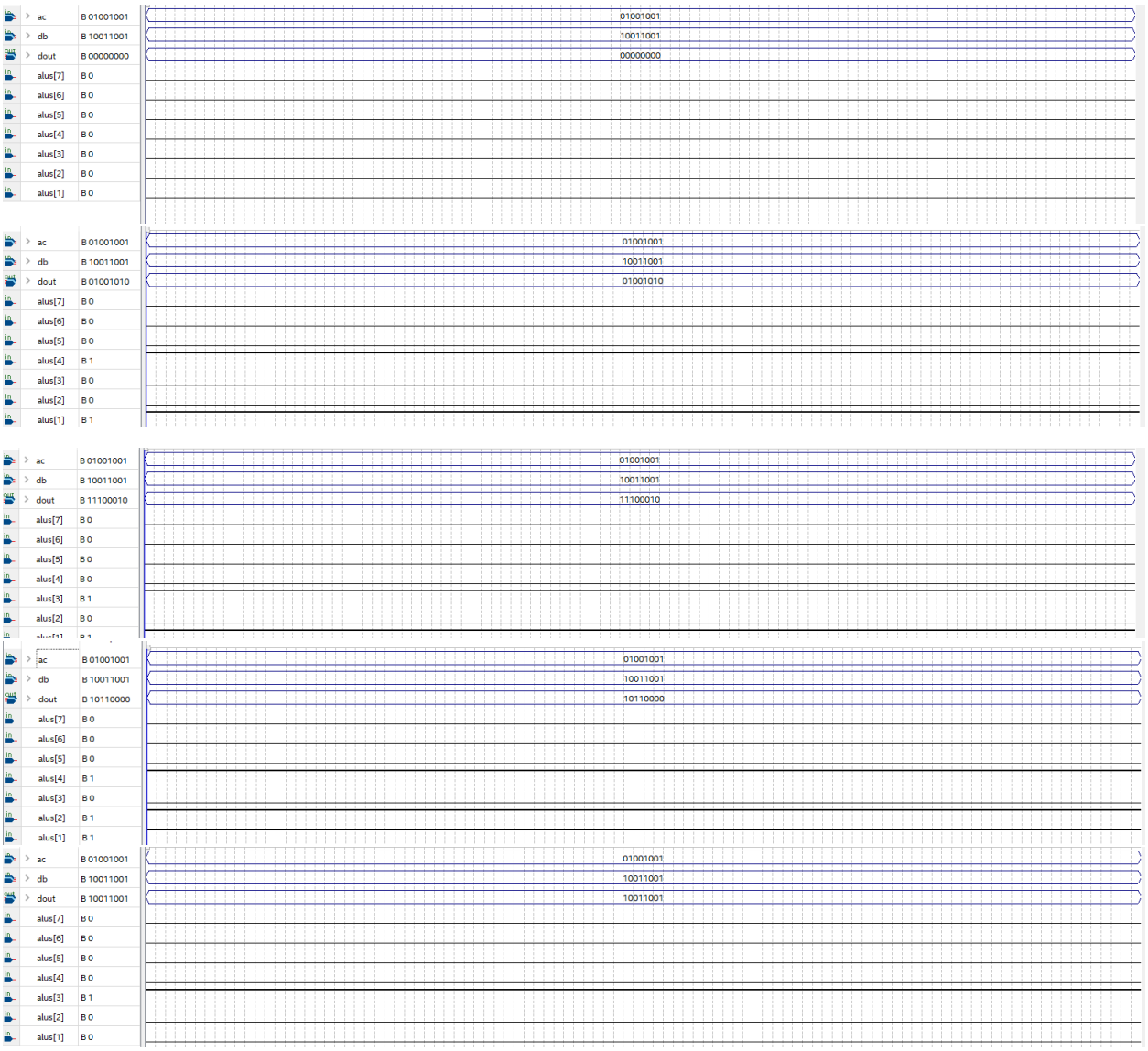
CLAC	0000000	00000000	**LDAC 5**	0000100	0000000

Πίνακας 1: Θεωρητικά αποτελέσματα για τις διεργασίες που σχετίζονται με την ALU και τον AC.

Με οδηγό την άσκηση 1 δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της ALU με τη βοήθεια του Waveform Editor και επιβεβαιώστε τα θεωρητικά αποτελέσματα.

***ΠΡΟΣΟΧΗ**: Εισάγετε τα nodes alus ξεχωριστά δηλαδή alus[1] έως alus[7] για να μην έχετε προβλήματα (error) κατά την εξομοίωση.*

Τοποθετήστε εδώ τις κυματομορφές σας:



Εικόνα 1: *Κυματομορφές εξομοίωσης της ALU*

Έλεγχος στην αναπτυξιακή πλατφόρμα DE10Lite

Δημιουργήστε δύο αρχεία κώδικα vhdl , με όνομα alu_dut_lib.vhd και alu_dut.vhd, τα οποία περιέχουν τα προγράμματα 7 και 8 αντίστοιχα. Προσθέστε τα στο project σας και ορίστε ως top level entity το αρχείο alu_dut.vhd.

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

package alu_dut_lib is

component alu is
generic (n : integer := 8);
port  ( ac      : in std_logic_vector(n-1 downto 0) ;
```

```
        db      : in std_logic_vector(n-1 downto 0) ;
        alus    : in std_logic_vector(7 downto 1) ;
        dout    : out std_logic_vector(n-1 downto 0) );
end component ;

end package alu_dut_lib;
```

Πρόγραμμα 7: *βιβλιοθήκη που περιέχει την ALU.*

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
use work.alu_dut_lib.all;

entity alu_dut is
generic (n : integer := 8);
port  ( alus    : in std_logic_vector(7 downto 1) ;
        dout    : out std_logic_vector(n-1 downto 0) );
end alu_dut ;

architecture arc of alu_dut is

signal ac, db : std_logic_vector(n-1 downto 0) ;
begin

ac  <= "01001001";
db  <= "10011001";

alu_i: alu port map(ac,db,alus,dout);

end arc;
```

Πρόγραμμα 8: *κώδικας ελέγχου για την ALU..*

Ρυθμίστε τα pin της ALU μέσω του μενού **Assigments|Pin Planer** ώστε να οδηγήσετε τα σήματα εισόδου alus[1] - alus[7] στα sw[0] – sw[6] και εξόδου dout[0] – dout[7] στα led[0] – led[7] της πλακέτας DE10Lite.

Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard	Reserved	rrent Streng	Slew Rate
alus[7]	Input	PIN_A13	7	B7_N0	PIN_T3	3.3-V LVTTL		8mA (default)	
alus[6]	Input	PIN_B12	7	B7_N0	PIN_U4	3.3-V LVTTL		8mA (default)	
alus[5]	Input	PIN_A12	7	B7_N0	PIN_U1	3.3-V LVTTL		8mA (default)	
alus[4]	Input	PIN_C12	7	B7_N0	PIN_N1	3.3-V LVTTL		8mA (default)	
alus[3]	Input	PIN_D12	7	B7_N0	PIN_U5	3.3-V LVTTL		8mA (default)	
alus[2]	Input	PIN_C11	7	B7_N0	PIN_V3	3.3-V LVTTL		8mA (default)	
alus[1]	Input	PIN_C10	7	B7_N0	PIN_U3	3.3-V LVTTL		8mA (default)	
dout[7]	Output	PIN_D14	7	B7_N0	PIN_P8	3.3-V LVTTL		8mA (default)	2 (default)
dout[6]	Output	PIN_E14	7	B7_N0	PIN_P1	3.3-V LVTTL		8mA (default)	2 (default)
dout[5]	Output	PIN_C13	7	B7_N0	PIN_N9	3.3-V LVTTL		8mA (default)	2 (default)
dout[4]	Output	PIN_D13	7	B7_N0	PIN_W1	3.3-V LVTTL		8mA (default)	2 (default)
dout[3]	Output	PIN_B10	7	B7_N0	PIN_V1	3.3-V LVTTL		8mA (default)	2 (default)
dout[2]	Output	PIN_A10	7	B7_N0	PIN_N8	3.3-V LVTTL		8mA (default)	2 (default)
dout[1]	Output	PIN_A9	7	B7_N0	PIN_R7	3.3-V LVTTL		8mA (default)	2 (default)
dout[0]	Output	PIN_A8	7	B7_N0	PIN_U2	3.3-V LVTTL		8mA (default)	2 (default)
<<new node>>									

Εικόνα 2: *ρυθμίσεις pin για την ALU*

Ελέγξτε τη λειτουργία της ALU με τη βοήθεια των διακοπτών και επιβεβαιώστε τα θεωρητικά αποτελέσματα (πίνακας 1).