

# **ΣΧΕΔΙΑΣΗ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ**

## **ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ**

ΚΑΘ. ΑΝΤΩΝΗΣ ΠΑΣΧΑΛΗΣ

**ΥΛΟΠΟΙΗΣΗ RISC ΕΠΕΞΕΡΓΑΣΤΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ MIPS R2000  
ΜΕ ΒΑΣΗ ΣΥΓΚΕΚΡΙΜΕΝΟ ΥΠΟΣΥΝΟΛΟ ΕΝΤΟΛΩΝ**

Α.Μ. : 2015508  
ΕΠΩΝΥΜΟ : ΚΑΡΑΓΙΑΝΝΗΣ  
ΟΝΟΜΑ : ΓΕΩΡΓΙΟΣ

## **Περιεχόμενα**

1.	ΥΠΟΣΥΝΟΛΟ ΕΝΤΟΛΩΝ.....	3
2.	ΣΧΕΔΙΑΣΤΙΚΕΣ ΕΠΙΛΟΓΕΣ .....	3
3.	ΣΧΟΛΙΑΣΜΟΣ ΓΙΑ ΤΗΝ ΔΟΜΗ ΚΑΙ ΤΗΝ ΛΕΙΤΟΥΡΓΙΑ ΚΑΘΕ ΒΑΣΙΚΟΥ ΚΥΚΛΩΜΑΤΟΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ .....	4
4.	ΕΠΑΛΗΘΕΥΣΗ ΟΡΘΗΣ ΣΧΕΔΙΑΣΗΣ ΚΑΘΕ ΒΑΣΙΚΟΥ ΚΥΚΛΩΜΑΤΟΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ .....	30
5.	ΕΠΑΛΗΘΕΥΣΗ ΟΡΘΗΣ ΣΧΕΔΙΑΣΗΣ ΕΠΕΞΕΡΓΑΣΤΗ .....	53
6.	ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΣΥΝΘΕΣΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ .....	67

## 1. ΥΠΟΣΥΝΟΛΟ ΕΝΤΟΛΩΝ

Το σύνολο εντολών που υποστηρίζει ο επεξεργαστής που μου ανατέθηκε να υλοποιήσω σύμφωνα με τον AM μου ( 2015508 ) είναι:

a)	LW	\$r1, 100(\$r2)	Load word
b)	SW	\$r1, 100(\$r2)	Store word
c)	ADDU	\$r1, \$r2, \$r3	Addition Unsigned
d)	SUBU	\$r1, \$r2, \$r3	Subtract Unsigned
e)	XORI	\$r1, \$r2, const	Bitwise Immediate XOR
f)	SLL	\$r1, \$r2, shamt	Shift Left Logical
g)	BNE	\$r1, \$r2, Label	Branch on not equal
h)	SLT	\$r3, \$r2, \$r1	Set less than
i)	JR	\$r1	Jump register
j)	JALR	\$r1	Jump and link register

## 2. ΣΧΕΔΙΑΣΤΙΚΕΣ ΕΠΙΛΟΓΕΣ

Όλη η ανάπτυξη και σχεδίαση του επεξεργαστή έγινε εξολοκλήρου με χρήση της γλώσσας περιγραφής υλικού VHDL. Το Datapath και η IFU σχεδιάστηκαν στο ίδιο επίπεδο.

Τα τμήματα που αφαιρέθηκαν ή τροποποιήθηκαν για την προσαρμογή της διόδου δεδομένων της θεωρίας σε σχέση με την υλοποίηση δίοδο δεδομένων είναι:

- a) Τροποποιήθηκε η ALU. Αφαιρέθηκαν ο πολυπλέκτης στην είσοδο shamt του ολισθητή της ALU και το σήμα επιλογής sv. Η λογική μονάδα αντικαταστάθηκε από μία πύλη xor. Ο ολισθητής τροποποιήθηκε ώστε να εκτελεί μόνο αριστερή λογική ολίσθηση.
- b) Το σήμα Ne/Eq' στην είσοδο του πολυπλέκτη 3 σε 1 που χρησιμοποιείται για την επιλογή της διεύθυνσης της επόμενης εντολής εκτέλεσης αφαιρέθηκε. Η πύλη xor που έπαιρνε ως εισόδους το παραπάνω σήμα και την έξοδο του καταχωρητή Z αντικαταστάθηκε από μία πύλη not που αντιστρέφει την έξοδο του καταχωρητή.
- c) Ο καταχωρητής MDR out παραλήφθηκε αφού η μνήμη δεδομένων υλοποιήθηκε εκμεταλευόμενοι τα διαθέσιμα blocks RAM του FPGA

### 3. ΣΧΟΛΙΑΣΜΟΣ ΓΙΑ ΤΗΝ ΔΟΜΗ ΚΑΙ ΤΗΝ ΛΕΙΤΟΥΡΓΙΑ ΚΑΘΕ ΒΑΣΙΚΟΥ ΚΥΚΛΩΜΑΤΟΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

#### A. ΕΠΙΠΕΔΟ ΔΙΟΔΟΥ ΔΕΔΟΜΕΝΩΝ (DATAPATH)

##### i. Συνολική Περιγραφή

Η Δίοδος Δεδομένων είναι βασική λειτουργική μονάδα του επεξεργαστή που χρησιμοποιείται για την εκτέλεση των εντολών. Η εκτέλεση των εντολών ανάγεται στην εκτέλεση μίας ακολουθίας στοιχειωδών λειτουργιών του datapath, οι οποίες στη συνέχεια ονομάζονται μικρο-λειτουργίες.

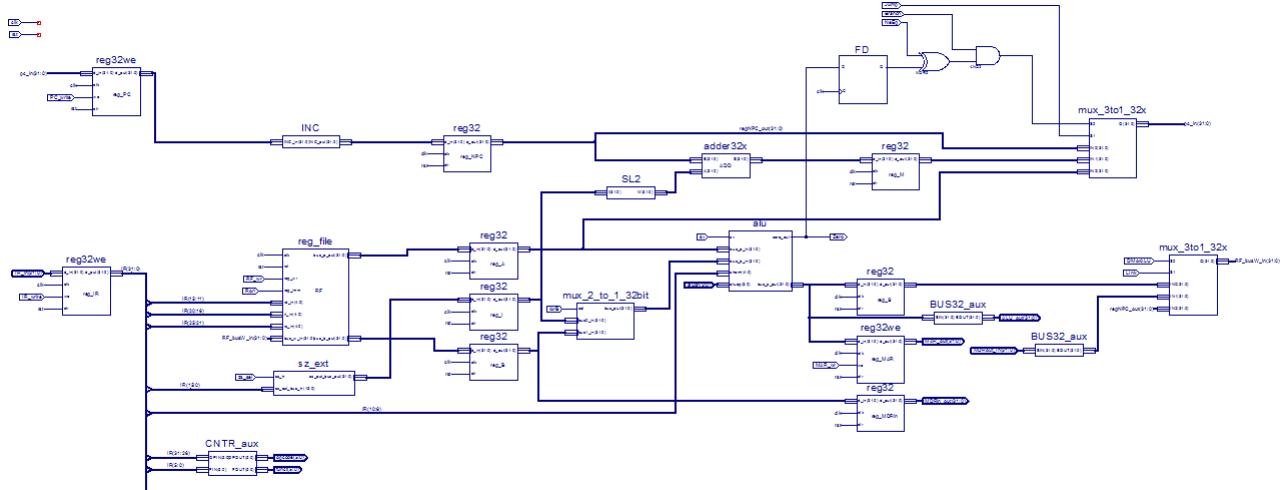


Figure 1. Το block διάγραμμα της Διόδου Δεδομένων (Datapath)

Το datapath απαρτίζεται από λειτουργικές μονάδες, όπως η Αριθμητική και Λογική Μονάδα (ALU), αθροιστές και κυκλώματα επέκτασης, το Αρχείο Καταχωρητών γενικής χρήσης, που αποτελείται από 32 καταχωρητές των 32 bits, εσωτερικές αρτηρίες, για τη διασύνδεση των λειτουργικών μονάδων με τους καταχωρητές γενικής και ειδικής χρήσης, πολυπλέκτες, για τον έλεγχο από τη μονάδα ελέγχου της ροής των πληροφοριών στις εσωτερικές αρτηρίες, και εξωτερικές αρτηρίες, για τη διασύνδεση του datapath με ανώτερα επίπεδα και μνήμες.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library UNISIM;
use UNISIM.VComponents.all;

entity datapath is
  port(
    clk      : in std_logic;
    rst      : in std_logic;

    link_in   : in std_logic;
    jump_in   : in std_logic;
    branch_in : in std_logic;

    sel_regimm_in : in std_logic;
    sel_bimm_in   : in std_logic;
    sel_sigzer_in : in std_logic;
  );
end entity;
  
```

```

sel_dmalu_in  : in std_logic;
wr_pc_in     : in std_logic;
wr_rf_in     : in std_logic;
wr_ir_in     : in std_logic;
wr_mar_in    : in std_logic;

ALUop_in      : in std_logic_vector(3 downto 0);
MDRout_in     : in std_logic_vector(31 downto 0);
IRin_in       : in std_logic_vector(31 downto 0);

Z_out         : out std_logic;

funct_out     : out std_logic_vector(5 downto 0);
opcode_out    : out std_logic_vector(5 downto 0);
PC_out        : out std_logic_vector(31 downto 0);
ALU_out       : out std_logic_vector(31 downto 0);
MAR_out       : out std_logic_vector(31 downto 0);
busW_out      : out std_logic_vector(31 downto 0);
MDRin_out     : out std_logic_vector(31 downto 0)
);
end datapath;

architecture behavioral of datapath is

component reg32 is
port(
clk   : in std_logic;
clr   : in std_logic;
d_in  : in std_logic_vector(31 downto 0);

d_out : out std_logic_vector(31 downto 0)
);
end component;

component reg32we is
port(
clk   : in std_logic;
clr   : in std_logic;
we    : in std_logic;
d_in  : in std_logic_vector(31 downto 0);

d_out : out std_logic_vector(31 downto 0)
);
end component;

component reg_file is
port(
clk : in std_logic;
rst : in std_logic;

reg_wr   : in std_logic;
reg_imm  : in std_logic;
rd_in    : in std_logic_vector(4 downto 0);
rt_in    : in std_logic_vector(4 downto 0);
rs_in    : in std_logic_vector(4 downto 0);
bus_w_in : in std_logic_vector(31 downto 0);

bus_a_out : out std_logic_vector(31 downto 0);
bus_b_out : out std_logic_vector(31 downto 0)
);
end component;

component alu is
port(
bus_a_in  : in std_logic_vector(31 downto 0);
bus_b_in  : in std_logic_vector(31 downto 0);
shamt    : in std_logic_vector(4 downto 0);

```

```

aluop      : in std_logic_vector(3 downto 0);
bus_s_out : out std_logic_vector(31 downto 0);
zero_out   : out std_logic
);
end component;

component ripple_carry_adder_32bit is
port(
cin       : in  std_logic;
bus_a_in  : in  std_logic_vector(31 downto 0);
bus_b_in  : in  std_logic_vector(31 downto 0);

bus_s_out : out std_logic_vector(31 downto 0)
);
end component;

signal regZ : std_logic;
signal alu_z_out_to_regZ_in : std_logic;

signal sz_extension : std_logic_vector(15 downto 0);--ok

signal regIR_to_in           : std_logic_vector(31 downto 0);
signal regA_out_to_in         : std_logic_vector(31 downto 0);
signal regI_out_to_in         : std_logic_vector(31 downto 0);
signal regB_out_to_in         : std_logic_vector(31 downto 0);
signal regNPC_out_to_in       : std_logic_vector(31 downto 0);
signal inc_out_to_npc_in      : std_logic_vector(31 downto 0);
signal sl2_out_to_add_in      : std_logic_vector(31 downto 0);
signal rf_out_to_regA_in       : std_logic_vector(31 downto 0);
signal rf_out_to_regB_in       : std_logic_vector(31 downto 0);
signal szext_out_to_regI       : std_logic_vector(31 downto 0);
signal mux_out_to_alu_in       : std_logic_vector(31 downto 0);
signal alu_busS_out_to_in      : std_logic_vector(31 downto 0);
signal add_out_to_regM_in      : std_logic_vector(31 downto 0);
signal mux_out_to_regPC_in      : std_logic_vector(31 downto 0);
signal regPC_out_to_inc_in      : std_logic_vector(31 downto 0);
signal regM_out_to_pcmux_in      : std_logic_vector(31 downto 0);
signal regS_out_to_rf_mux_in      : std_logic_vector(31 downto 0);
signal mux_out_to_rf_bus_w_in      : std_logic_vector(31 downto 0);

begin

--instantiate components
i_RegPC: reg32we
port map(
clk    => clk,
clr    => rst,
we     => wr_pc_in,
d_in   => mux_out_to_regPC_in,
d_out  => regPC_out_to_inc_in
);

i_RegIR: reg32we
port map(
clk    => clk,
clr    => rst,
we     => wr_ir_in,
d_in   => IRin_in,
d_out  => regIR_to_in
);

--drvie sign/zero entension
sz_extension <= (others=>(regIR_to_in(15) and sel_sigzer_in));
szext_out_to_regI <= sz_extension & regIR_to_in(15 downto 0);

```

```

i_reg_file: reg_file
port map(
    clk => clk,
    rst => rst,

    reg_wr      => wr_rf_in,
    reg_imm     => sel_regimm_in,
    rd_in       => regIR_to_in(15 downto 11),
    rt_in       => regIR_to_in(20 downto 16),
    rs_in       => regIR_to_in(25 downto 21),
    bus_w_in   => mux_out_to_rf_bus_w_in,

    bus_a_out => rf_out_to_regA_in,
    bus_b_out => rf_out_to_regB_in
);

i_regA: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => rf_out_to_regA_in,
    d_out   => regA_out_to_in
);

i_regI: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => szext_out_to_regI,
    d_out   => regI_out_to_in
);

i_regB: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => rf_out_to_regB_in,
    d_out   => regB_out_to_in
);

i_alu: alu
port map(
    bus_a_in  => regA_out_to_in,
    bus_b_in  => mux_out_to_alu_in,
    shamt     => regIR_to_in(10 downto 6),
    aluop     => ALUop_in,

    bus_s_out => alu_busS_out_to_in,
    zero_out  => alu_z_out_to_regZ_in
);

i_regs: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => alu_buss_out_to_in,
    d_out   => regS_out_to_rf_mux_in
);

i_regMDRin: reg32
port map(
    clk      => clk,
    clr      => rst,

```

```

d_in  => regB_out_to_in,
d_out => MDRin_out
);

i_regMAR: reg32we
port map(
    clk      => clk,
    clr      => rst,
    we       => wr_mar_in,
    d_in     => alu_buss_out_to_in,
    d_out   => MAR_out
);

i_inc: ripple_carry_adder_32bit
port map(
    cin      => '0',
    bus_a_in  => regPC_out_to_inc_in,
    bus_b_in  => (2 =>'1', others=>'0'),
    bus_s_out => inc_out_to_npc_in
);

i_regNPC: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => inc_out_to_npc_in,
    d_out   => regNPC_out_to_in
);

--drive SL2
sl2_out_to_add_in <= regI_out_to_in(29 downto 0) & "00";

i_add: ripple_carry_adder_32bit
port map(
    cin      => '0',
    bus_a_in  => regNPC_out_to_in,
    bus_b_in  => sl2_out_to_add_in,
    bus_s_out => add_out_to_regM_in
);

i_regM: reg32
port map(
    clk      => clk,
    clr      => rst,
    d_in     => add_out_to_regM_in,
    d_out   => regM_out_to_pcmux_in
);

--drive regB or RegI to ALU's bus_b_in
alu_mux: process(sel_bimm_in, regI_out_to_in, regB_out_to_in)
begin
    if sel_bimm_in = '0' then
        mux_out_to_alu_in <= regI_out_to_in;
    elsif sel_bimm_in = '1' then
        mux_out_to_alu_in <= regB_out_to_in;
    end if;
end process;

--drive mux_out_to_rf_bus_w_in

```

```

rf_mux: process(link_in, sel_dmalu_in, regNPC_out_to_in, MDRout_in,
regS_out_to_rf_mux_in)
begin
  if link_in = '1' then
    mux_out_to_rf_bus_w_in <= regNPC_out_to_in;
  else
    if sel_dmalu_in = '1' then
      mux_out_to_rf_bus_w_in <= MDRout_in;
    else
      mux_out_to_rf_bus_w_in <= regS_out_to_rf_mux_in;
    end if;
  end if;
end process;

--drive mux_out_to_regPC_in
pc_mux: process(jump_in, regZ, branch_in, regA_out_to_in, regM_out_to_pcmux_in,
regNPC_out_to_in)
begin
  if jump_in = '1' then
    mux_out_to_regPC_in <= regA_out_to_in;
  else
    if ((not regZ) and branch_in) = '1' then
      mux_out_to_regPC_in <= regM_out_to_pcmux_in;
    else
      mux_out_to_regPC_in <= regNPC_out_to_in;
    end if;
  end if;
end process;

--drive regZ (D Flip Flop)
reg_Z: process(clk, rst)
begin
  if rising_edge(clk) then
    if rst = '1' then
      null;
    else
      regZ <= alu_z_out_to_regZ_in;
    end if;
  end if;
end process;

--drive outputs
Z_out <= alu_z_out_to_regZ_in;

funct_out <= regIR_to_in(5 downto 0);
opcode_out <= regIR_to_in(31 downto 26);

PC_out <= regPC_out_to_inc_in;
ALU_out <= alu_buss_out_to_in;
busW_out <= mux_out_to_rf_bus_w_in;

end behavioral;

```

**Listing 1. Ο συνολικός VHDL κώδικας για την υλοποίηση της Διόδου Δεδομένων (Datapath)**

### ii. Αριθμητική και Λογική Μονάδα (ALU)

Η Αριθμητική και Λογική Μονάδα (ALU) είναι υπεύθυνη για όλες τις πράξεις που εκτελούνται μέσα στον επεξεργαστή. Η δομή της αποτελείται από έναν ολισθητή, έναν αθροιστή-αφαιρέτη, τη λογική μονάδα, τη μονάδας και ένα κύκλωμα ανίχνευσης '0' (δένδρο OR). Τα δύο σημαντικότερα κυκλώματα, τα οποία αναπτύχθηκαν ξεχωριστά και ύστερα χρησιμοποιήθηκαν ως components στην υλοποίηση της ALU, είναι ο αθροιστής-αφαιρέτης και ο ολισθητής.

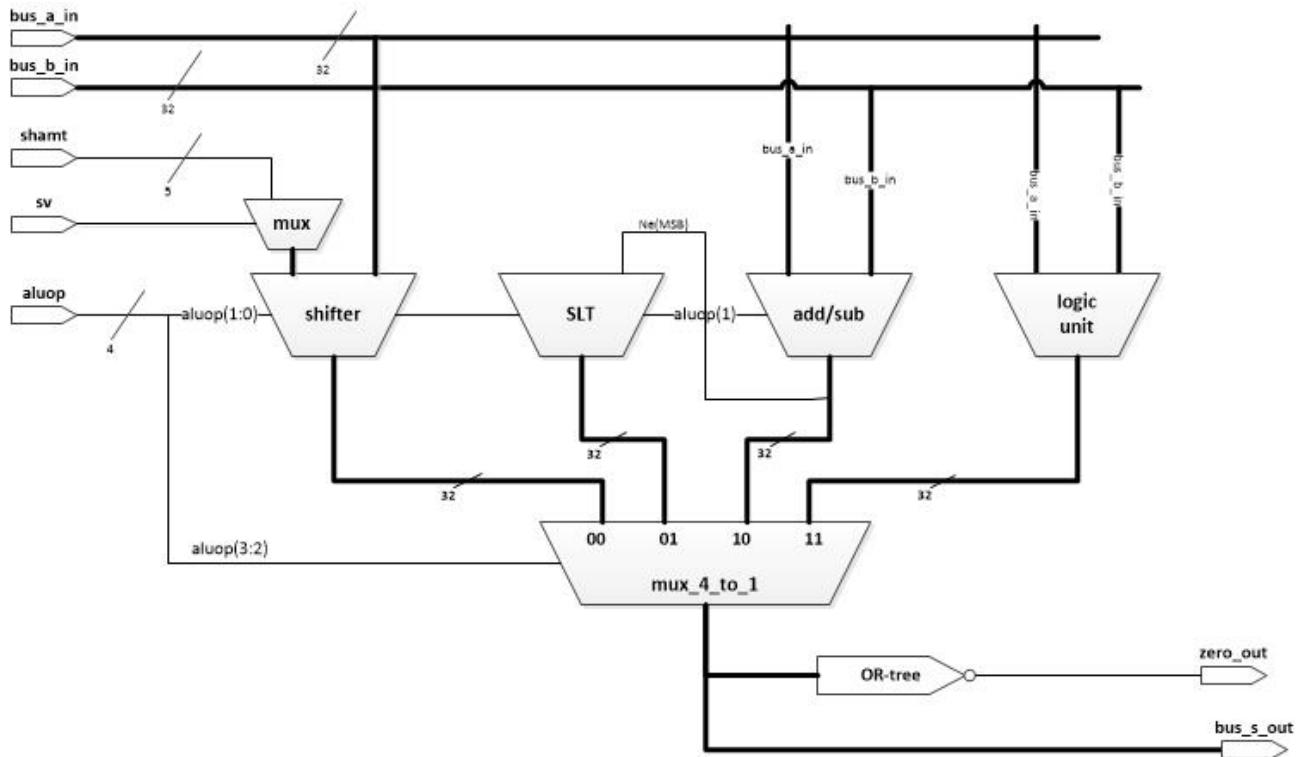


Figure 2. Το block διάγραμμα της Αριθμητικής και Λογικής Μονάδας

Ο αθροιστής-αφαιρέτης αποτελείται από 2 αθροιστές των 16-bits και 32 λογικές πύλες xor. Οι δύο αθροιστές των 16-bits αποτελούν έτοιμα κυκλώματα της βιβλιοθήκης της XILINX και συνθέτουν έναν αθροιστή διάδοσης κρατουμένου των 32 bits, ενώ οι λογικές πύλες υλοποιούν ένα bitwise xor κύκλωμα. Η χρήση των λογικών πυλών xor τροποποιεί κατάλληλα το κύκλωμα του RCA ώστε να εκτελεί και αφαιρεση. Στα ψηφιακά κυκλώματα η αφαιρεση εκτελείται ως μία πράξη πρόσθεσης μεταξύ του αφαιρέτη και του συμπληρώματος ως προς βάση (two's complement) του αφαιρετέου.

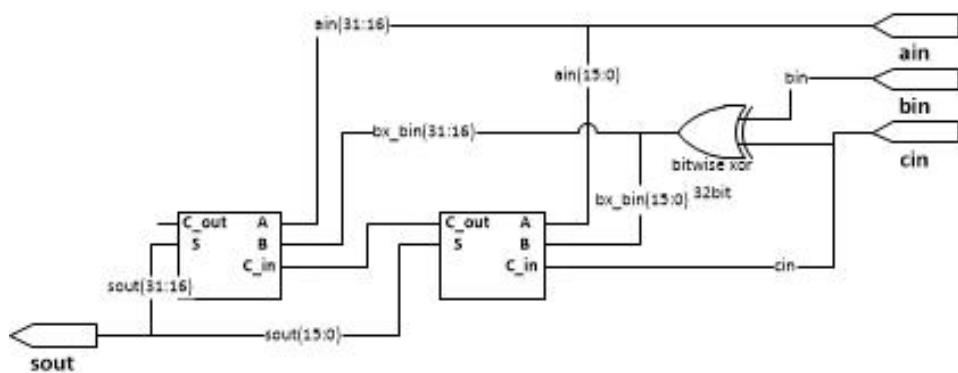


Figure 3. Το block διάγραμμα του αθροιστή αφαιρέτη

Μελετώντας τον πίνακα αλήθειας μίας πύλης xor, καταλήγουμε στο εξής συμπέρασμα. Άς θεωρήσουμε τη μία από τις δύο εισόδους ως ένα σήμα επιλογής. Όταν το σήμα επιλογής παίρνει την τιμή '0', η τιμή του δεύτερου σήματος εισόδου μεταφέρεται ατόφια στην έξοδο της πύλης. Αντίθετα, όταν το σήμα επιλογής παίρνει την τιμή '1', το δεύτερο σήμα εισόδου αντιστρέφεται. Ουσιαστικά, μπορούμε να πούμε ότι μία πύλη xor λειτουργεί σαν μία πύλη not με σήμα επιλογής.

Όπως φαίνεται και στο παραπάνω σχήμα, αν το σήμα cin του αθροιστή-αφαιρέτη λάβει την τιμή '1', τότε η είσοδος B του αθροιστή, που οδηγείται από την έξοδο του bitwise xor κυκλώματος, λαμβάνει το διάνυσμα bin ανεστραμμένο (not bin), ή αλλιώς το συμπλήρωμα ως προς ελαττωμένη βάση (one's complement) του σήματος bin. Για να ολοκληρωθεί ορθά η πράξη της αφαίρεσης, το cin οδηγεί και την είσοδο C\_in του αθροιστή. Έτσι, το κύκλωμα εκτελεί την πρόσθεση ain + (bin)<sub>1'sc</sub> + 1. Καθώς όμως ισχύει (bin)<sub>1'sc</sub> + 1 = (bin)<sub>2'sc</sub>, τελικά το κύκλωμα εκτελεί την πρόσθεση ain + (bin)<sub>2'sc</sub>, δηλαδή την αφαίρεση των διανυσμάτων ain και bin. Αντίθετα, αν το cin λάβει τιμή '0', τότε στην έξοδο του xor κυκλώματος οδηγείται το διάνυσμα bin απόφιο και το κύκλωμα εκτελεί την πράξη ain + bin.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity adder_substractor is
  port(
    ain   :  in std_logic_vector (31 downto 0);
    bin   :  in std_logic_vector (31 downto 0);
    cin   :  in std_logic;
    sout  :  out std_logic_vector (31 downto 0)
  );
end adder_substractor;

architecture dataflow of adder_substractor is

component adder16x is
  port(
    C_in  :  in std_logic;
    A     :  in std_logic_vector(15 downto 0);
    B     :  in std_logic_vector(15 downto 0);

    S     :  out std_logic_vector(15 downto 0);
    C_out :  out std_logic;
    OFL   :  out std_logic
  );
end component;

signal xor_in      : std_logic_vector(31 downto 0);
signal xrbn        : std_logic_vector(31 downto 0);
signal intrnl_c   : std_logic;
signal ofl         : std_logic_vector(1 downto 0);
signal msccarry    : std_logic;

begin

  xor_in  <=  (others => cin);
  xrbn   <=  bin xor xor_in;

  i_lsb_add: adder16x
  port map(
    C_in  => cin,
    A     => ain(15 downto 0),
    B     => xrbn(15 downto 0),
    S     => sout(15 downto 0),
    C_out => intrnl_c,
    OFL   => ofl(0)
  );

  i_msb_add: adder16x
  port map(
    C_in  => intrnl_c,
    A     => bin(15 downto 0),
    B     => xrbn(15 downto 0),
    S     => sout(15 downto 0),
    C_out => intrnl_c,
    OFL   => ofl(0)
  );

```

```

A      => ain(31 downto 16),
B      => xrbm(31 downto 16),
S      => sout(31 downto 16),
C_out => mscarry,
OFL   => ofl(1)
);

end dataflow;

```

**Listing 2. Ο VHDL κώδικας για την υλοποίηση του αθροιστή αφαιρέτη**

Ο ολισθητής, στη συγκεκριμένη υλοποίηση, εκτελεί μόνο αριστερή λογική ολίσθηση. Η δομή του αποτελείται από 5 πολυπλέκτες 2 σε 1 των 32 bits και μία σειρά από buffers.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity mux_32 is
    port(in0      : in  std_logic_vector (31 downto 0);
          in1      : in  std_logic_vector (31 downto 0);
          ctrl     : in  std_logic;
          result   : out std_logic_vector (31 downto 0));
end entity mux_32;

architecture structural of mux_32 is

Begin

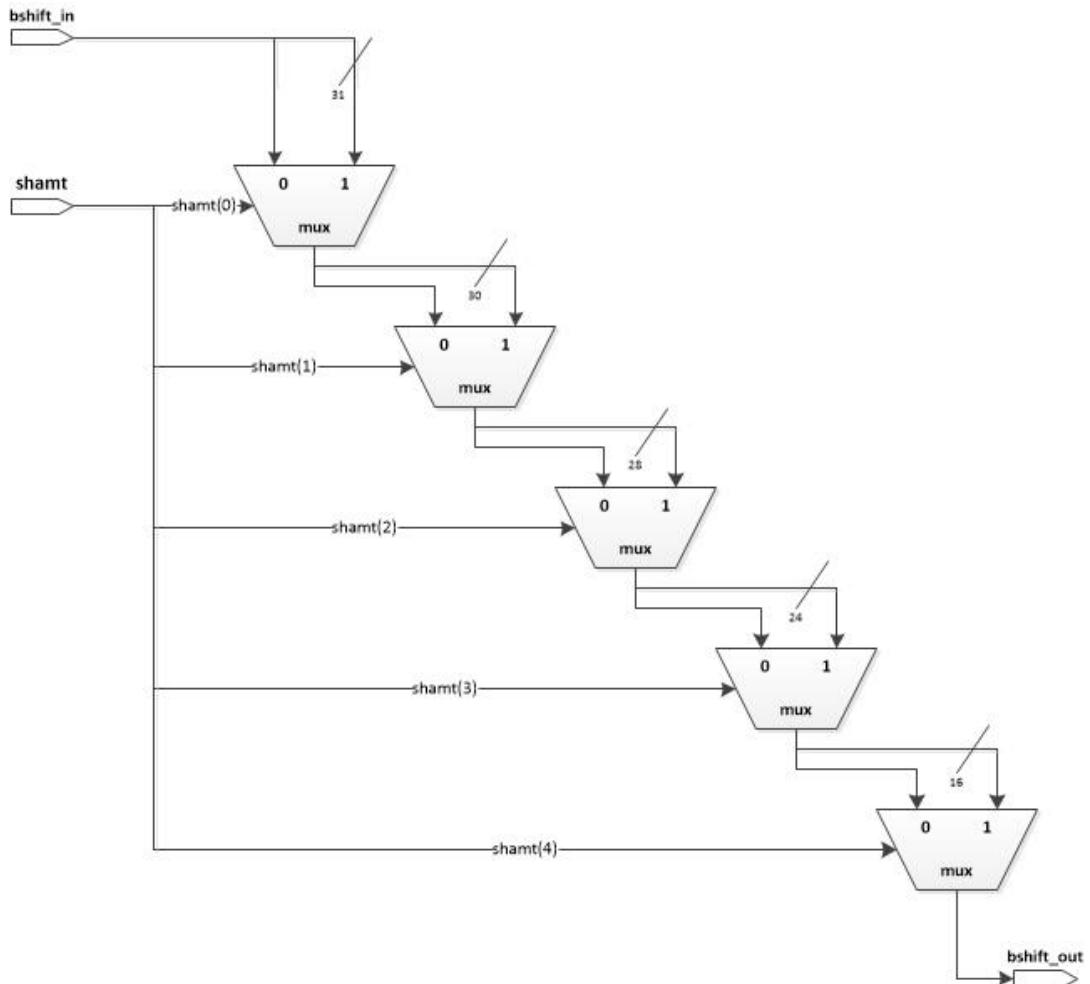
    result <= in1 WHEN ctrl ='1' ELSE in0 ;

end architecture structural;

```

**Listing 3. Ο VHDL κώδικας για την υλοποίηση του πολυπλέκτη 2 σε 1 των 32 bits**

Οι πολυπλέκτες τοποθετούνται σε επίπεδα ο ένας κάτω από τον άλλον. Κάθε πολυπλέκτης ορίζει ένα επίπεδο ολίσθησης  $2^i$  θέσεων, όπου i το επίπεδο του πολυπλέκτη. Το σήμα επιλογής κάθε πολυπλέκτη καθορίζει αν θα πραγματοποιηθεί ολίσθηση στο επίπεδο αυτό ή όχι. Όπως φαίνεται και στο παρακάτω σχήμα, ένας πολυπλέκτης στο επίπεδο x λαμβάνει στη μία εισόδο του την έξοδο του πολυπλέκτη του επιπέδου x-1. Στην δεύτερη είσοδό του συνθέτει ένα σήμα αποτελούμενο από τα  $32 - 2^x$  λιγότερο σημαντικά ψηφία (lsbs) της εξόδου του προηγούμενου πολυπλέκτη, ακολουθούμενο από  $2^x$  '0'. Για την εκτέλεση μίας ολίσθησης  $2^x$  θέσεων, ο πολυπλέκτης στο επίπεδο x λαμβάνει το αντίστοιχο σήμα επιλογής shamt(x) = '1' και προωθεί στην έξοδό του το σήμα που συνθέτει στη δεύτερη είσοδό του.



**Figure 4.** Το block διάγραμμα του barrel shifter

Για την υλοποίηση του ολισθητή χρησιμοποιήθηκε ο barrel shifter που μας δόθηκε από το εργαστήριο, τροποποιημένος ώστε να εκτελεί μόνο αριστερή λογική ολίσθηση. Παρακάτω παρουσιάζεται ο τροποποιημένος κώδικας VHDL για την υλοποίησή του.

```

library IEEE;
use IEEE.std_logic_1164.all;

Entity bshift is    -- barrel shifter
  port (
    shift   : in  std_logic_vector(4 downto 0);  -- shift count
    bshift_in  : in  std_logic_vector (31 downto 0);
    bshift_out : out std_logic_vector (31 downto 0)
  );
end entity bshift;

architecture circuits of bshift is

  signal L1s  : std_logic_vector(31 downto 0);
  signal L2s  : std_logic_vector(31 downto 0);
  signal L4s  : std_logic_vector(31 downto 0);
  signal L8s  : std_logic_vector(31 downto 0);
  signal L16s : std_logic_vector(31 downto 0);
  signal L1   : std_logic_vector(31 downto 0);
  signal L2   : std_logic_vector(31 downto 0);
  signal L4   : std_logic_vector(31 downto 0);
  signal L8   : std_logic_vector(31 downto 0);

```

```

component mux_32
  port(in0 : in std_logic_vector (31 downto 0);
       in1 : in std_logic_vector (31 downto 0);
       ctl : in std_logic;
       result : out std_logic_vector (31 downto 0));
end component;

begin -- circuits
L1w: L1s <= bshift_in(30 downto 0) & '0'; -- just wiring
L1m: mux_32 port map (in0=>bshift_in, in1=>L1s, ctl=>shift(0), result=>L1);
L2w: L2s <= L1(29 downto 0) & "00"; -- just wiring
L2m: mux_32 port map (in0=>L1, in1=>L2s, ctl=>shift(1), result=>L2);
L4w: L4s <= L2(27 downto 0) & "0000"; -- just wiring
L4m: mux_32 port map (in0=>L2, in1=>L4s, ctl=>shift(2), result=>L4);
L8w: L8s <= L4(23 downto 0) & "000000000"; -- just wiring
L8m: mux_32 port map (in0=>L4, in1=>L8s, ctl=>shift(3), result=>L8);
L16w: L16s <= L8(15 downto 0) & "0000000000000000"; -- just wiring
L16m: mux_32 port map (in0=>L8, in1=>L16s, ctl=>shift(4), result=>bshift_out);

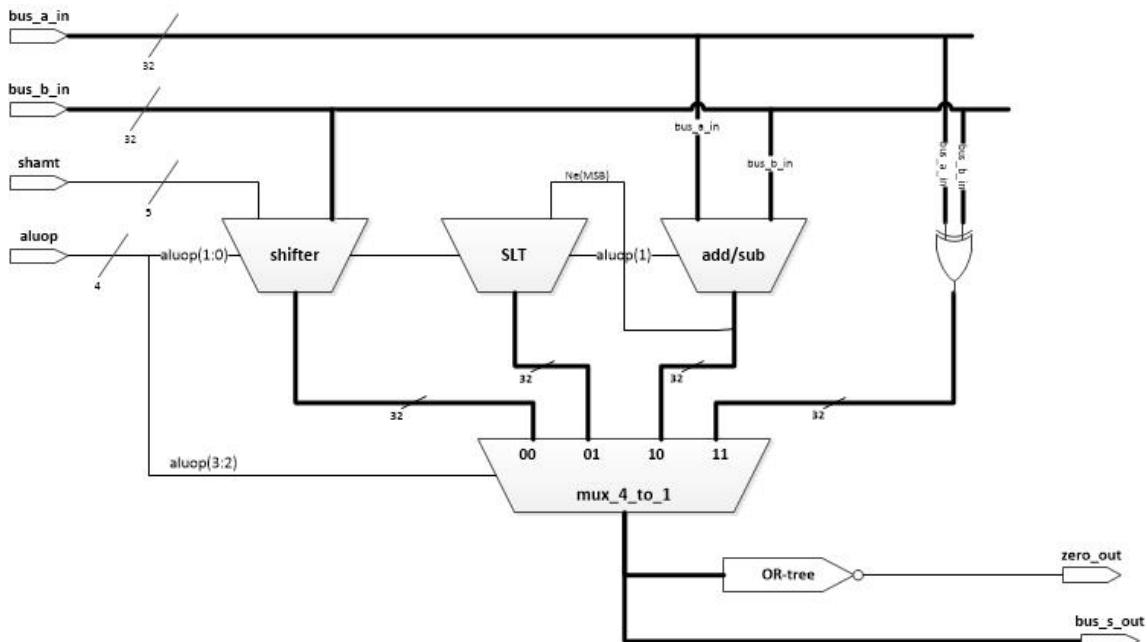
end architecture circuits; -- of bshift

```

**Listing 4. Ο VHDL κώδικας για την υλοποίηση του ολισθητή**

Ο barrel shifter που μας δόθηκε από το εργαστήριο είχε τη δυνατότητα εκτέλεσης αριστερών και δεξιών λογικών, καθώς και αριθμητικών ολισθήσεων. Ωστόσο, το σύνολο εντολών που χρειάστηκε να υλοποιηθεί, απαιτούσε την εκτέλεση μόνο αριστερών λογικών ολισθήσεων. Για το λόγο αυτό αφαιρέθηκαν οι επιπλέον πολυπλέκτες που υλοποιούσαν τις υπόλοιπες ολισθήσεις καθώς και οι πολυπλέκτες επιλογής εξόδου του αποτελέσματος. Στην ίδια λογική, απομακρύνθηκαν και οι είσοδοι “left” και “logical” που οδηγούν τα σήματα επιλογής των δύο τελευταίων πολυπλεκτών.

Η μονάδα SLT, για τη σύγκριση διανυσμάτων, καθώς και η λογική μονάδα, που εκτελεί μόνο την πράξη xor, απλοποιήθηκαν και υλοποιήθηκαν με τη μορφή ενός buffer και μίας xor πύλης των 32 bits, αντίστοιχα. Η τελική μορφή της ALU φαίνεται στο παρακάτω σχήμα.



**Figure 5. Το τελικό block διάγραμμα της Αριθμητικής και Λογικής Μονάδας**

Παρακάτω παρουσιάζεται ο συνολικός VHDL κώδικας για την υλοποίηση της ALU. Αξίζει να σημειωθούν τα εξής:

- η υλοποίηση της μονάδας SLT με χρήση ενός 32 bit buffer, ο οποίος έχει σταθερά την τιμή '0' στα 31 MSBs του, ενώ το bit 0 οδηγείται από το MBB της εξόδου της ALU.

```
signal slt_out      : std_logic_vector(31 downto 0);
begin
  slt_out <= (0 => addsub_out(31), others=>'0');
```

- η υλοποίηση της λογικής μονάδας με μία πύλη xor των 32 bits. Η εξόδος της οποίας οδηγείται σε έναν 32bit buffer

```
signal xor_out      : std_logic_vector(31 downto 0);
begin
  xor_out <= bus_a_in xor bus_b_in;
```

- η υλοποίηση του ολισθητή με χρήση του τροποποιημένου στοιχείου barrel shifter. Στην αρχική σχεδίαση, η υλοποίηση διατηρούσε ακέραια τη μορφή του ολισθητή. Για να εκτελεί μόνο αριστερή λογική ολίσθηση, η διασύνδεση έγινε όπως φαίνεται παρακάτω.

```
i_bshift: bshift
port map(
  shift      => shamt,
  bshift_in => bus_b_in,
  bshift_out => shift_out
);
```

Κατά τη βελτιστοποίηση, οι επιπλέον πολυπλέκτες αφαιρέθηκαν, ενώ το interface, και κατά συνέπεια και η διασύνδεση, του barrel shifter τροποποιήθηκαν.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity alu is
  port(
    bus_a_in      : in std_logic_vector(31 downto 0);
    bus_b_in      : in std_logic_vector(31 downto 0);
    shamt         : in std_logic_vector(4 downto 0);
    aluop         : in std_logic_vector(3 downto 0);

    bus_s_out     : out std_logic_vector(31 downto 0);
    zero_out      : out std_logic
  );
end alu;

architecture behavioral of alu is

component bshift is -- barrel shifter
  port (
    shift      : in  std_logic_vector(4 downto 0); -- shift count
    bshift_in : in  std_logic_vector (31 downto 0);

    bshift_out : out std_logic_vector (31 downto 0)
  );
end component;
```

```

component adder_substractor is
  port(
    ain   :  in std_logic_vector (31 downto 0);
    bin   :  in std_logic_vector (31 downto 0);
    cin   :  in std_logic;
    sout  :  out std_logic_vector (31 downto 0)
  );
end component;

--mux_4-to-1 signals
--input signals
signal shift_out  : std_logic_vector(31 downto 0);
signal slt_out    : std_logic_vector(31 downto 0);
signal addsub_out : std_logic_vector(31 downto 0);
signal xor_out    : std_logic_vector(31 downto 0);
--output signals
signal mux_out    : std_logic_vector(31 downto 0);

begin

  --Components instantiation
  i_bshift: bshift
  port map(
    shift      => shamt,
    bshift_in => bus_b_in,
    bshift_out  => shift_out
  );

  i_adder_substractor: adder_substractor
  port map(
    ain      => bus_a_in,
    bin      => bus_b_in,
    cin      => aluop(1),
    sout    => addsub_out
  );

  --internal signals from "components"
  slt_out <= (0 => addsub_out(31), others=>'0');
  xor_out <= bus_a_in xor bus_a_in;

  alu_bhv:process(bus_a_in, bus_b_in, shamt, aluop)
    variable op  : std_logic_vector(1 downto 0);
  begin
    op := aluop(3 downto 2);

    if op = "00" then
      mux_out <= shift_out;
    elsif op = "01" then
      mux_out <= slt_out;
    elsif op = "10" then
      mux_out <= addsub_out;
    elsif op = "11" then
      mux_out <= xor_out;
    else
      null;
    end if;

  end process alu_bhv;

  --drive output signals
  bus_s_out <= mux_out;
  zero_out  <= not ( mux_out(0) or mux_out(1) or mux_out(2) or mux_out(3) or mux_out(4)
or mux_out(5) or mux_out(6) or mux_out(7) or mux_out(8) or mux_out(9) or

```

```

        mux_out(10) or mux_out(11) or mux_out(12) or mux_out(13) or mux_out(14) or
mux_out(15) or mux_out(16) or mux_out(17) or mux_out(18) or mux_out(19) or
mux_out(20) or mux_out(21) or mux_out(22) or mux_out(23) or mux_out(24) or
mux_out(25) or mux_out(26) or mux_out(27) or mux_out(28) or mux_out(29) or
mux_out(30) or mux_out(31) );

end behavioral;

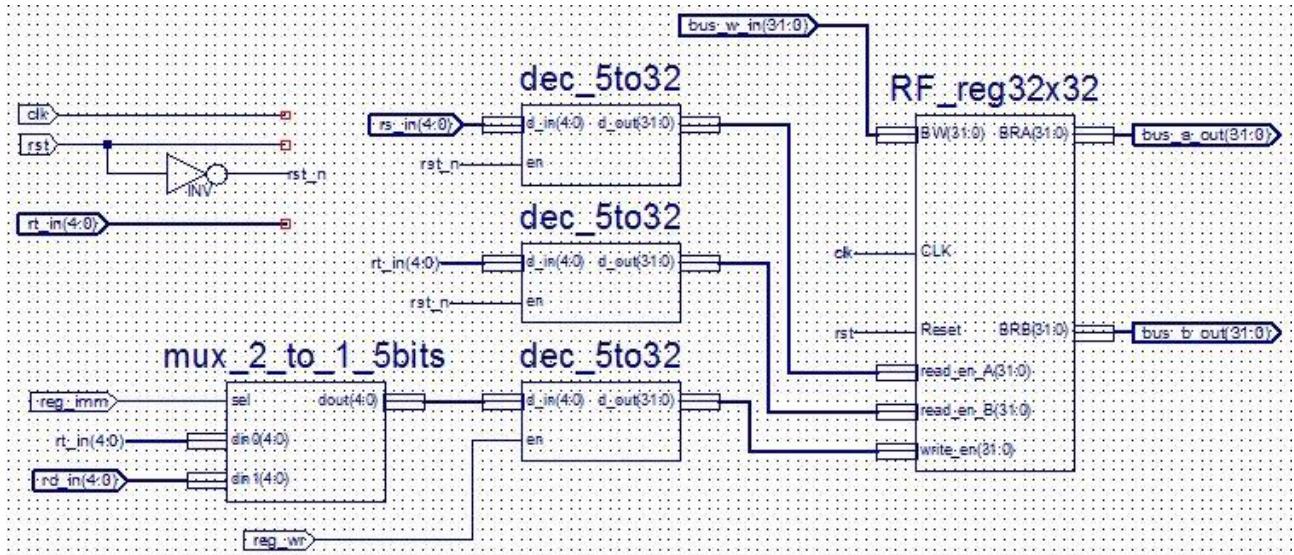
```

**Listing 5. Ο VHDL κώδικας για την υλοποίηση της Αριθμητικής και Λογικής Μονάδας (ALU)**

### iii. Αρχείο Καταχωρητών (RF)

Το αρχείο καταχωρητών αποτελείται από μία συστοιχία 32 καταχωρητών των 32 bits με σήμα enable, 3 αποκωδικοποιητές 5 σε 32 και έναν πολυπλέκτη 2 σε 1 των 5 bits επίσης. Ο ένας από τους 3 αποκωδικοποιητές χρησιμοποιείται ως αποπλέκτης 1 σε 32 και οδηγεί το σήμα εγγραφής (reg\_wr) σε καταχωρητή, με βάση το σήμα επιλογής που λαμβάνει από την είσοδο. Ο καταχωρητής στον οποίον θα πραγματοποιηθεί εγγραφή καθορίζεται από τα πεδία rt ή rd της εντολής που βρίσκεται στον Instruction Register, ενώ τα δεδομένα οδηγούνται μέσω της εισόδου bus\_w\_in(31:0).

Η αρχιτεκτονική του register file επιτρέπει την ανάγνωση δύο καταχωρητών ταυτόχρονα, μέσω των εξόδων bus\_a\_out(31:0) και bus\_b\_out(31:0). Καθένας από τους άλλους δύο αποκωδικοποιητές λαμβάνει στην είσοδο του ένα σήμα επιλογής. Αυτά τα σήματα επιλογής αποτελούν τη binary απεικόνιση της διεύθυνσης ενός καταχωρητή και προέρχονται από τα πεδία rs και rt, αντίστοιχα, της εντολής που βρίσκεται στον Instruction Register. Οι διευθύνσεις αποκωδικοποιούνται και η έξοδος των dec\_5to32 οδηγεί τα σήματα read\_en\_A(31:0) και read\_en\_B(31:0), αντίστοιχα, τα οποία με τη σειρά τους οδηγούν τα σήματα TA και TB του κάθε καταχωρητή. Έτσι, επιτυγχάνεται η ανάγνωση του περιεχομένου του καταχωρητή που βρίσκεται στη συγκεκριμένη διεύθυνση, ενώ το περιεχόμενό του οδηγείται στην έξοδο bus\_a\_out(31:0) ή/και bus\_b\_out(31:0), ανάλογα με το πεδίο-σήμα (rs\_in(4:0) ή/και rt\_in(4:0) αντίστοιχα) που καλεί την ανάγνωσή του.



**Figure 6. Το block διάγραμμα του Αρχείου Καταχωρητών (RF)**

Η υλοποίηση του αρχείου καταχωρητών έγινε με περιγραφή υλικού. Ο κώδικας για την υλοποίηση του αρχείου καταχωρητών παρουσιάζεται στο επόμενο πλαίσιο.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity reg_file is
  port(
    clk : in std_logic;
    rst : in std_logic;

    reg_wr      : in std_logic;
    reg_imm     : in std_logic;
    rd_in       : in std_logic_vector(4 downto 0);
    rt_in       : in std_logic_vector(4 downto 0);
    rs_in       : in std_logic_vector(4 downto 0);
    bus_w_in    : in std_logic_vector(31 downto 0);

    bus_a_out   : out std_logic_vector(31 downto 0);
    bus_b_out   : out std_logic_vector(31 downto 0)
  );
end reg_file;

architecture behavioral of reg_file is

component dec_5to32 is
  port(
    d_in   : in std_logic_vector(4 downto 0);
    en     : in std_logic;
    d_out  : out std_logic_vector(31 downto 0)
  );
end component;

component RF_reg32x32
  port(
    BW      : in std_logic_vector (31 downto 0);
    read_en_A : in std_logic_vector (31 downto 0);
    read_en_B : in std_logic_vector (31 downto 0);
    CLK     : in std_logic;
    write_en : in std_logic_vector (31 downto 0);
    Reset   : in std_logic;
    BRA    : out std_logic_vector (31 downto 0);
    BRB    : out std_logic_vector (31 downto 0)
  );
end component;

signal rst_n : std_logic;
signal mux_out_to_demux_d_in : std_logic_vector(4 downto 0);
signal demux_out_to_rf_write_en_in : std_logic_vector(31 downto 0);
signal dec_out_to_rf_read_en_A_in : std_logic_vector(31 downto 0);
signal dec_out_to_rf_read_en_B_in : std_logic_vector(31 downto 0);

begin

  rst_n <= not rst;

  i_demux: dec_5to32
  port map(
    d_in  => mux_out_to_demux_d_in,
    en    => reg_wr,
    d_out => demux_out_to_rf_write_en_in
  );

```

```

i_decA: dec_5to32
port map(
  d_in  => rs_in,
  en    => rst_n,
  d_out => dec_out_to_rf_read_en_A_in
);

i_decB: dec_5to32
port map(
  d_in  => rt_in,
  en    => rst_n,
  d_out => dec_out_to_rf_read_en_B_in
);

i_regf: RF_reg32x32
port map(
  BW  => bus_w_in,
  read_en_A => dec_out_to_rf_read_en_A_in,
  read_en_B => dec_out_to_rf_read_en_B_in,
  CLK => clk,
  write_en  => demux_out_to_rf_write_en_in,
  Reset => rst,
  BRA => bus_a_out,
  BRB => bus_b_out
);

mux: process(rd_in, rt_in, reg_imm)
  variable rdh_or_rtl : std_logic;
begin
  rdh_or_rtl := reg_imm;

  if reg_imm = '1' then
    mux_out_to_demux_d_in <= rd_in;
  else
    mux_out_to_demux_d_in <= rt_in;
  end if;
end process mux;
end behavioral;

```

**Listing 6. Ο VHDL κώδικας για την υλοποίηση του αρχείου καταχωρητών (register file)**

#### iv. Μονάδα Προσκόμισης Εντολής (IFU)

Η ΜΠΕ χρησιμοποιείται για τον υπολογισμό και την επιλογή της διεύθυνσης της εντολής που πρόκειται να εκτελεσθεί σε 3 κύκλους. Αποτελείται από 3 καταχωρητές, έναν ολισθητή, δύο αθροιστές (όλα των 32 bits) και ένα D flip flop.

Ο καταχωρητής reg\_PC αποθηκεύει τη διεύθυνση της επόμενης εντολής που ακολουθεί. Η έξοδος του reg\_PC αποτελεί και έξοδο του datapath προς την control unit, ενώ ταυτόχρονα οδηγείται και σε έναν αθροιστή (INC). Ο INC αυξάνει τη διεύθυνση κατά 4 και το αποτέλεσμα (PC + 4) αποθηκεύεται στον προσωρινό καταχωρητή NPC.

Σε περίπτωση εκτέλεσης εντολής διακλάδωσης με συνθήκη (BNE, BEQ – εντολές τύπου I), το πεδίο immediate (16 bits) υποδεικνύει το πλήθος m των εντολών που έπονται της εντολής στη διεύθυνση PC + 4 και πρέπει να “παρακαμφθούν”. Η τιμή m, αφού υποστεί επέκταση προσήμου (στο επίπεδο του datapath) αποθηκεύεται στον προσωρινό καταχωρητή I και από εκεί οδηγείται στον ολισθητή SL2. Ο ολισθητής εκτελεί αριστερή ολίσθηση δύο θέσεων, πολλαπλασιάζοντας το πλήθος m επί το μέγεθος των εντολών σε bytes. Το αποτέλεσμα (4m) αντιπροσωπεύει την απόσταση σε bytes της διεύθυνσης της

εντολής που θα πρέπει να εκτελεστεί, στη μνήμη εντολών. Στη συνέχεια, η διαφορά 4m, σε bytes, προστίθεται στη διεύθυνση PC + 4, μέσω ενός δεύτερου αθροιστή των 32 bits, και η νέα διεύθυνση PC + 4 + 4m (διεύθυνση προορισμού διακλάδωσης) αποθηκεύεται στον προσωρινό καταχωρητή M.

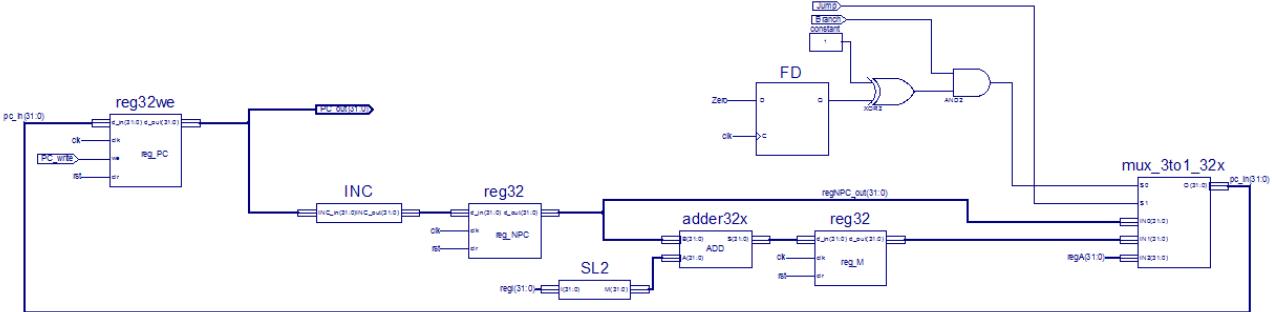


Figure 7. To block διάγραμμα της Μονάδας Προσκόμισης Εντολής (IFU)

Στο τελευταίο στάδιο λειτουργίας της IFU, επιλέγεται η διεύθυνση της επόμενης εντολής που θα εκτελεστεί μέσω ενός πολυπλέκτη 3 σε 1 των 32 bits. Ο πολυπλέκτης διαθέτει τρία σήματα επιλογής (“jump\_in”, “branch\_in” και “sel\_neeq\_in”) τα οποία καθορίζουν τη διεύθυνση που θα οδηγηθεί στον καταχωρητή PC, όπως φαίνεται και στο παρακάτω τμήμα από το συνολικό κώδικα του datapath. Αξίζει να σημειωθεί ότι το σήμα επιλογής “sel\_neeq\_in” έχει παραληφθεί, καθώς η μόνη εντολή διακλάδωσης που υλοποιήθηκε είναι η BNE, η οποία εκτελείται μόνο όταν τα σήματα επιλογής παίρνουν τις εξής τιμές: jump\_in = 0, branch\_in = 1 και sel\_neeq\_in = 1. Θεωρώντας λοιπόν ότι το σήμα επιλογής sel\_neeq\_in είναι πάντα ίσο με 1, η πύλη xor αντικαταστάθηκε με μία πύλη not που αντιστρέφει το σήμα zero\_out από την ALU στο σήμα επιλογής του πολυπλέκτη, μέσω ενός D flip flop. Η υλοποίηση του πολυπλέκτη φαίνεται στο παρακάτω απόσπασμα του συνολικού κώδικα του datapath.

```
--drive mux_out_to_regPC_in
pc_mux: process(jump_in, regZ, branch_in, regA_out_to_in, regM_out_to_pcmux_in,
regNPC_out_to_in)
begin
  if jump_in = '1' then
    mux_out_to_regPC_in <= regA_out_to_in;
  else
    if ((not regZ) and branch_in) = '1' then
      mux_out_to_regPC_in <= regM_out_to_pcmux_in;
    else
      mux_out_to_regPC_in <= regNPC_out_to_in;
    end if;
  end if;
end process;
```

Listing 7. Ο VHDL κώδικας για την υλοποίηση του πολυπλέκτη 3 σε 1 των 32 bits για την επιλογή της διεύθυνσης της επόμενης εντολής

## B. ΕΠΙΠΕΔΟ ΕΠΕΞΕΡΓΑΣΤΗ (PROCESSOR – TOP LEVEL)

### i. Συνολική Περιγραφή

Όπως φαίνεται και από το παρακάτω σχήμα, το πιο πάνω ierarχικό επίπεδο αποτελείται από όλα τα δομικά κυκλώματα που υλοποιήσαμε μέχρι στιγμής. Πιο συγκεκριμένα, ο επεξεργαστής, σε αυτό το επίπεδο, απαρτίζεται από τη δίοδο δεδομένων (datapath), την μανάδα ελέγχου (control unit) και τις μνήμες εντολών και δεδομένων. Η μονάδα ελέγχου σχηματίζεται από δύο block, ένα συνδυαστικό (cu\_comb) και ένα ακολουθιακό (cu\_fsm).

Η μονάδα ελέγχου επικοινωνεί με τη δίοδο δεδομένων. Οι είσοδοι opcode\_in και funct\_in της μονάδας ελέγχου οδηγούνται από τις αντίστοιχες εξόδους της διόδου δεδομένων. Από την άλλη, οι έξοδοι της μονάδας ελέγχου οδηγούν τις εισόδους της διόδου δεδομένων, παρέχοντας τα κατάλληλα σήματα επιλογής και ενεργοποίησης εγγραφής.

Τέλος, η δίδοδος δεδομένων επικοινωνεί με τη μνήμη εντολών (IMEM) μέσω των αρτηριών PC\_out και IR\_in, καθώς και με τη μνήμη δεδομένων μέσω των αρτηριών MAR\_out, MDRin\_out και MDRout\_in.

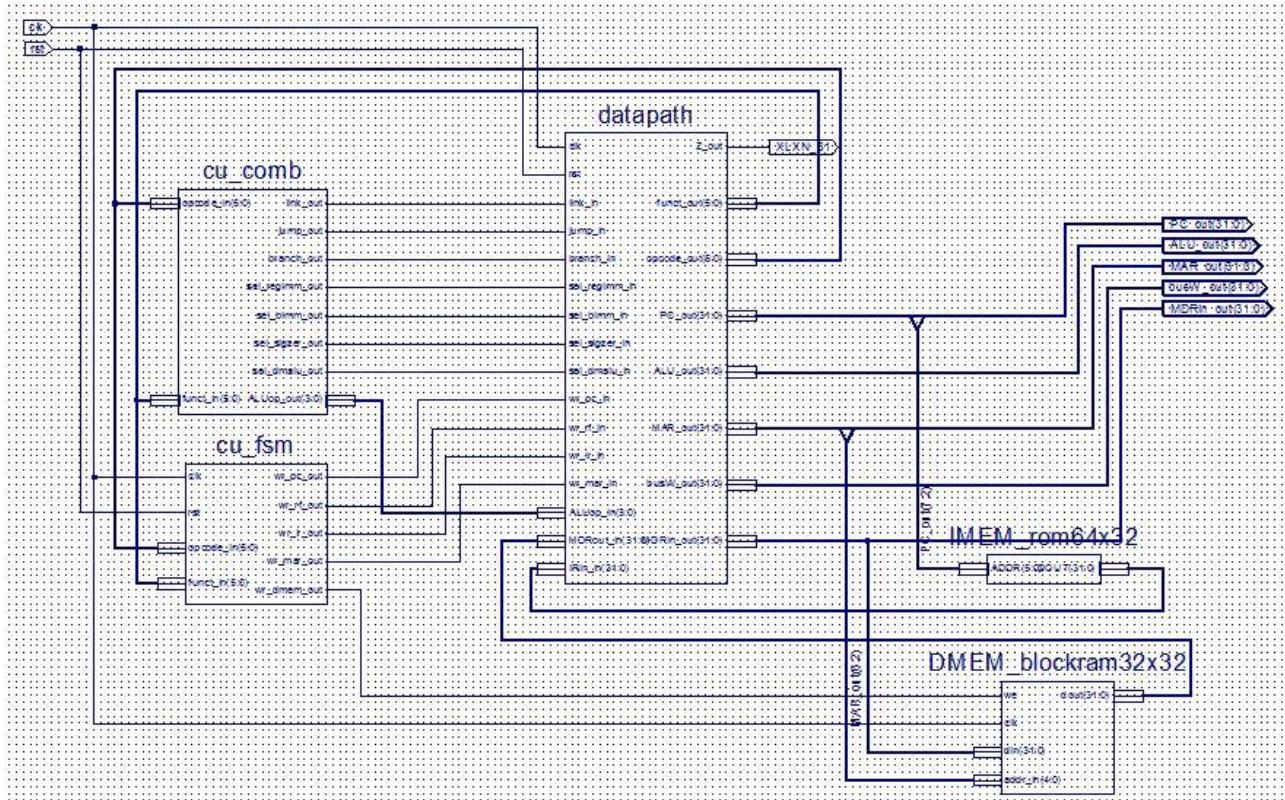


Figure 8 To block διάγραμμα του επεξεργαστή

### ii. Συνδυαστική Μονάδα Ελέγχου

Η Συνδυαστική Μονάδα Ελέγχου είναι υπεύθυνη για την παραγωγή των σημάτων ελέγχου, τα οποία παραμένουν σταθερά κατά την εκτέλεση μίας εντολής, σε όλους τους κύκλους. Τα σήματα αυτά προσδιορίζουν όλες τις μικρολειτουργίες που εκτελούν οι λειτουργικές μονάδες της διόδου δεδομένων, κατά την εκτέλεση της εντολής, και καθορίζουν τη ροή των δεδομένων μέσα στη δίοδο. Οι τιμές των σημάτων αυτών προσδιορίζουν τον πίνακα αλήθειας της Συνδυαστικής Μονάδας Ελέγχου, όπως φαίνεται παρακάτω.

Instruction	Opcode	Funct	Link	Jump	Branch	RorI	BorI	SorZ	DMorALU	ALUop	sv	NEorEQ
SLL	000000 (00)	000000 (00)	0	0	0	1	1		0	0000 (00)	0	
JR	000000 (00)	001000 (08)		1								
JALR	000000 (00)	001001 (09)	1	1		1						
ADDU	000000 (00)	100001 (21)	0	0	0	1	1		0	1001 (09)		
SUBU	000000 (00)	100011 (23)	0	0	0	1	1		0	1011 (2B)		
SLT	000000 (00)	101010 (2A)	0	0	0	1	1		0	0110 (06)		
BNE	000101 (05)			0	1		1	1		1011 (0B)		1
XORI	001110 (0E)		0	0	0	0	0	0	0	1110 (0E)		
LW	100011 (23)		0	0	0	0	0	1	1	1001 (09)		
SW	101011 (2B)			0	0		0	1	1	1001 (09)		

Table 1 Ο Πίνακας Αλήθειας (Πίνακας Λειτουργίας) της Συνδυαστικής Μονάδας Ελέγχου

Ο VHDL κώδικας για την υλοποίηση της Συνδυαστικής Μονάδας Ελέγχου αναπτύχθηκε με βάση τον παραπάνω πίνακα αλήθειας. Σ' αυτόν, περιγράφεται η συμπεριφορά της μονάδας, καθορίζοντας τις εξόδους της, για κάθε έγκυρο ζεύγος εισόδων. Η περιγραφή συμπεριφοράς της μονάδας γίνεται με χρήση μίας διεργασίας (process), η οποία βασίζεται σε ένθετες δομές case που ελέγχουν τις δύο εισόδους της μονάδας. Η περιγραφή της Συνδυαστικής Μονάδας παρουσιάζεται παρακάτω. Αξίζει να σημειωθεί ότι τα σήματα sv και NEorEQ έχουν παραληφθεί στην περιγραφή και την υλοποίηση καθώς έχουν αφαιρεθεί από το συνολικό design του επεξεργαστή.

```

library ieee;
use ieee.std_logic_1164.all;

entity cu_comb is
  port(
    opcode_in : in std_logic_vector(5 downto 0);
    funct_in  : in std_logic_vector(5 downto 0);

    link_out   : out std_logic;
    jump_out   : out std_logic;
    branch_out : out std_logic;

    sel_regimm_out : out std_logic;
    sel_bimm_out   : out std_logic;
    sel_sigzer_out : out std_logic;
    sel_dmalu_out  : out std_logic;

    ALUop_out : out std_logic_vector(3 downto 0)
  );
end cu_comb;

architecture behavioral of cu_comb is

-- "opcode" codes definition as constants
constant RTYPE : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant LW    : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SW    : std_logic_vector(5 downto 0) := "101011"; -- 0x2B
constant XORI  : std_logic_vector(5 downto 0) := "001110"; -- 0x0E
constant BNE   : std_logic_vector(5 downto 0) := "000101"; -- 0x05

-- "funct" codes definition as constants
constant SLLR : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant ADDU : std_logic_vector(5 downto 0) := "100001"; -- 0x21
constant SUBU : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SLT  : std_logic_vector(5 downto 0) := "101010"; -- 0x2A
constant JR   : std_logic_vector(5 downto 0) := "001000"; -- 0x08
constant JALR : std_logic_vector(5 downto 0) := "001001"; -- 0x09

begin

cu_comb: process(opcode_in, funct_in)
begin
  --output initialization
  link_out  <= '0';
  jump_out  <= '0';
  branch_out <= '0';

  sel_regimm_out  <= '0';
  sel_bimm_out    <= '0';
  sel_sigzer_out  <= '0';
  sel_dmalu_out   <= '0';

  ALUop_out <= "0000";

  case opcode_in is

```

```

when LW =>
    link_out      <= '0';
    jump_out     <= '0';
    branch_out   <= '0';

    sel_regimm_out <= '0';
    sel_bimm_out   <= '0';
    sel_sigzer_out <= '1';
    sel_dmalu_out  <= '1';

    ALUop_out <= "1001";

when SW =>
    link_out      <= '-';
    jump_out     <= '0';
    branch_out   <= '0';

    sel_regimm_out <= '-';
    sel_bimm_out   <= '0';
    sel_sigzer_out <= '1';
    sel_dmalu_out  <= '-';

    ALUop_out <= "1001";

when XORI =>
    link_out      <= '0';
    jump_out     <= '0';
    branch_out   <= '0';

    sel_regimm_out <= '0';
    sel_bimm_out   <= '0';
    sel_sigzer_out <= '0';
    sel_dmalu_out  <= '0';

    ALUop_out <= "1110";

when BNE =>
    link_out      <= '-';
    jump_out     <= '0';
    branch_out   <= '1';

    sel_regimm_out <= '-';
    sel_bimm_out   <= '1';
    sel_sigzer_out <= '1';
    sel_dmalu_out  <= '-';

    ALUop_out <= "1011";

when RTYPE =>

    case funct_in is
        when SLLR =>
            link_out      <= '0';
            jump_out     <= '0';
            branch_out   <= '0';

            sel_regimm_out <= '1';
            sel_bimm_out   <= '1';
            sel_sigzer_out <= '-';
            sel_dmalu_out  <= '0';

            ALUop_out <= "0000";

        when ADDU =>
            link_out      <= '0';
            jump_out     <= '0';
            branch_out   <= '0';

```

```

        sel_regimm_out  <= '1';
        sel_bimm_out    <= '1';
        sel_sigzer_out  <= '-';
        sel_dmalu_out    <= '0';

        ALUop_out <= "1001";

when SUBU =>
    link_out      <= '0';
    jump_out     <= '0';
    branch_out   <= '0';

    sel_regimm_out  <= '1';
    sel_bimm_out    <= '1';
    sel_sigzer_out  <= '-';
    sel_dmalu_out    <= '0';

    ALUop_out <= "1011";

when SLT  =>
    link_out      <= '0';
    jump_out     <= '0';
    branch_out   <= '0';

    sel_regimm_out  <= '1';
    sel_bimm_out    <= '1';
    sel_sigzer_out  <= '-';
    sel_dmalu_out    <= '0';

    ALUop_out <= "0110";

when JR   =>
    link_out      <= '-';
    jump_out     <= '1';
    branch_out   <= '-';

    sel_regimm_out  <= '-';
    sel_bimm_out    <= '-';
    sel_sigzer_out  <= '-';
    sel_dmalu_out    <= '-';

    ALUop_out <= "----";

when JALR =>
    link_out      <= '1';
    jump_out     <= '1';
    branch_out   <= '-';

    sel_regimm_out  <= '1';
    sel_bimm_out    <= '-';
    sel_sigzer_out  <= '-';
    sel_dmalu_out    <= '-';

    ALUop_out <= "----";

when others =>
    link_out      <= '-';
    jump_out     <= '-';
    branch_out   <= '-';

    sel_regimm_out  <= '-';
    sel_bimm_out    <= '-';
    sel_sigzer_out  <= '-';
    sel_dmalu_out    <= '-';

    ALUop_out <= "----";

end case; -- funct_in

```

```

when others =>
    link_out      <= '-';
    jump_out     <= '-';
    branch_out   <= '-';

    sel_regimm_out <= '-';
    sel_bimm_out   <= '-';
    sel_sigzer_out <= '-';
    sel_dmalu_out  <= '-';

    ALUop_out <= "----";
end case; -- opcode_in
end process;
end behavioral;

```

**Listing 8 Ο VHDL κώδικας για την υλοποίηση της Συνδυαστικής Μονάδας Ελέγχου**

### iii. Ακολουθιακή Μονάδα Ελέγχου

Η Ακολουθιακή Μονάδα Ελέγχου είναι υπεύθυνη για την παραγωγή των σημάτων ελέγχου ενεργοποίησης εγγραφής (write enable) των καταχωρητών του datapath και των μνημών του επεξεργαστή. Τα σήματα αυτά πρέπει να είναι ενεργά (να έχουν τιμή 1 στη θετική λογική) σε ένα συγκεκριμένο κύκλο κατά την εκτέλεση μίας εντολής, έτσι ώστε, στην επόμενη ακμή του ρολογιού, να είναι δυνατή η εγγραφή του σχετικού καταχωρητή ή της σχετικής μνήμης.

Τα σήματα, καθώς και η χρονική περίοδος κατά την οποία τα οποία αυτά θα πρέπει να είναι ενεργά, προκύπτουν από την ανάλυση των εντολών στις επιμέρους μικρολειτουργίες τους. Για κάθε μία από τις υποστηριζόμενες εντολές του επεξεργαστή, σχεδιάστηκε η αντίστοιχη μηχανή πεπερασμένων καταστάσεων (FSM) που περιγράφει τις διακριτές καταστάσεις εκτέλεσης, τα σήματα ενεργοποίησης εγγραφής που παράγονται σε κάθε κατάσταση και τις μεταβάσεις μεταξύ αυτών. Οι FSMs που προέκυψαν, μελετήθηκαν και, αφού προέκυψαν όλες οι διακριτές καταστάσεις τους, ενοποιήθηκαν σε μία ευρύτερη FSM που ικανοποιεί όλο το σύνολο των υποστηριζόμενων εντολών και εξυπηρετεί τον επεξεργαστή. Το τελικό διάγραμμα καταστάσεων που περιγράφει την FSM της ακολουθιακής μονάδας του επεξεργαστή, παρουσιάζεται στην επόμενη σελίδα.

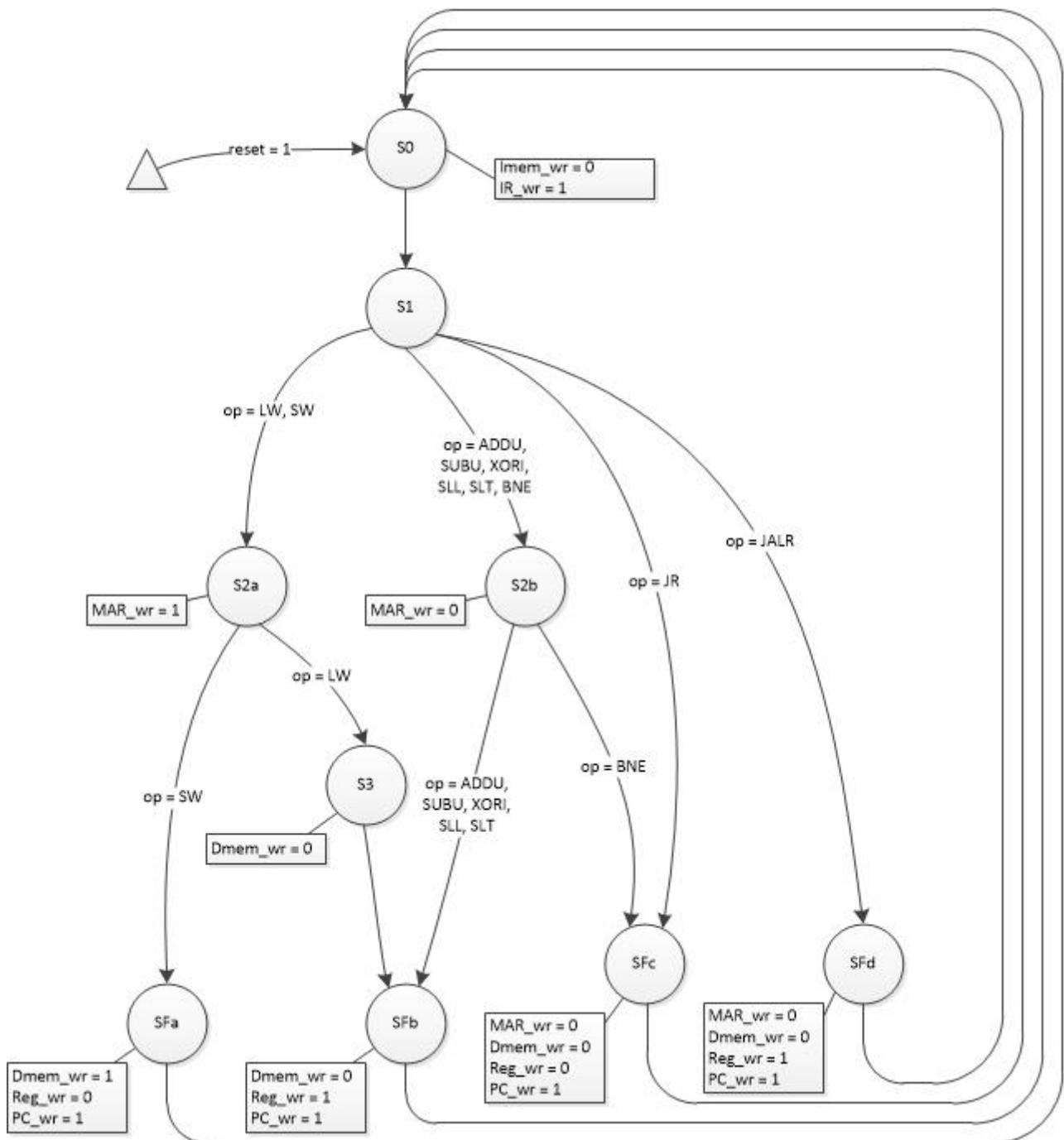


Figure 9 Το διάγραμμα καταστάσεων της Ακολουθιακής Μονάδας Ελέγχου

Ο VHDL κώδικας για την υλοποίηση της Ακολουθιακής Μονάδας Ελέγχου αναπτύχθηκε ώστε να περιγράψει την παραπάνω FSM που προέκυψε. Η περιγραφή συμπεριφοράς της μονάδας γίνεται με τη μέθοδο σχεδίασης των δύο διεργασιών (two process design method) που είναι κατάλληλη για την περιγραφή ακολουθιακών κυκλωμάτων με ένα ρολόι, όπως οι FSMs. Η αρχιτεκτονική αποτελείται από δύο μέρη. Το τμήμα της ακολουθιακής λογικής (synchronous process) υλοποιεί τον καταχωρητή κατάστασης (cur\_state) που οδηγείται από τα σήματα clk και rst. Το τμήμα της συνδυαστικής λογικής (asynchronous process) παράγει την επόμενη κατάσταση και τις εξόδους του κυκλώματος, συναρτήσει της τρέχουσας κατάστασης και των εισόδων (μηχανή πεπερασμένων

καταστάσεων τύπου Mealy). Η περιγραφή-ανάπτυξη γίνεται με τη χρήση ένθετων δομών case, οι οποίες ελέγχουν όλους τους πιθανούς συνδυασμούς κατάστασης και εισόδων.

```

library ieee;
use ieee.std_logic_1164.all;

entity cu_fsm is
  port(
    clk      : in std_logic;
    rst      : in std_logic;
    opcode_in : in std_logic_vector (5 downto 0);
    funct_in : in std_logic_vector (5 downto 0);

    wr_pc_out      : out std_logic;
    wr_rf_out      : out std_logic;
    wr_ir_out      : out std_logic;
    wr_mar_out     : out std_logic;
    wr_dmem_out    : out std_logic
  );
end cu_fsm;

architecture behavioral of cu_fsm is
-- state definition
type fsm_states is
  (S0, S1, S2a, S2b, S3, SFa, SFb, SFc, SFd);
signal cur_state  : fsm_states;
signal nxt_state  : fsm_states;

-- "opcode" codes definition as constants
constant RTYPE   : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant LW       : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SW       : std_logic_vector(5 downto 0) := "101011"; -- 0x2B
constant XORI    : std_logic_vector(5 downto 0) := "001110"; -- 0x0E
constant BNE     : std_logic_vector(5 downto 0) := "000101"; -- 0x05
-- "funct" codes definition as constants
constant SLLR    : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant ADDU    : std_logic_vector(5 downto 0) := "100001"; -- 0x21
constant SUBU    : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SLT     : std_logic_vector(5 downto 0) := "101010"; -- 0x2A
constant JR      : std_logic_vector(5 downto 0) := "001000"; -- 0x08
constant JALR    : std_logic_vector(5 downto 0) := "001001"; -- 0x09

begin
-- synchronous process where states are changing
SYNCHR: process(clk, rst)
begin
  if rst = '1' then
    cur_state <= S0;           -- initial state
  elsif rising_edge(clk) then
    cur_state <= nxt_state;
  end if;
end process;

-- asynchronous process where
ASYNCHR: process(cur_state, opcode_in, funct_in)
begin
-- initial state and output initialization
  nxt_state <= S0; -- initial state
  wr_pc_out    <= '0';
  wr_rf_out    <= '0';
  wr_ir_out    <= '0';
  wr_mar_out   <= '0';
  wr_dmem_out  <= '0';

-- next states and outputs on state transition
  case cur_state is

```

```

when S0      => wr_ir_out <= '1';
                  nxt_state <= S1;

when S1      => case opcode_in is
                  when LW|SW      => nxt_state <= S2A;
                  when BNE|XORI   => nxt_state <= S2B;
                  when RTYPE     => case funct_in is
                                         when ADDU|SUBU|SLLR|SLT => nxt_state <= S2b;
                                         when JR          => nxt_state <= SFc;
                                         when JALR        => nxt_state <= SFd;
                                         when others      => null;
                                         end case; -- funct_in
                  when others    => null;
end case; -- opcode_in

when S2A     => wr_mar_out <= '1';
                  case opcode_in is
                  when SW      => nxt_state <= SFa;
                  when LW      => nxt_state <= S3;
                  when others  => null;
end case; -- opcode_in

when S2B     => wr_mar_out <= '0';
                  case opcode_in is
                  when XORI    => nxt_state <= SFb;
                  when RTYPE   => case funct_in is
                                         when ADDU|SUBU|SLLR|SLT => nxt_state <= SFb;
                                         when others      => null;
                                         end case; -- funct_in
                  when BNE     => nxt_state <= SFc;
                  when others  => null;
end case; -- opcode_in

when S3      => wr_dmem_out <= '0';
                  nxt_state <= SFb;

when SFa     => wr_dmem_out <= '1';
                  wr_rf_out  <= '1';
                  wr_pc_out  <= '1';
                  nxt_state <= S0;

when SFb     => wr_dmem_out <= '0';
                  wr_rf_out  <= '1';
                  wr_pc_out  <= '1';
                  nxt_state <= S0;

when SFc     => wr_mar_out <= '0';
                  wr_dmem_out <= '0';
                  wr_rf_out  <= '0';
                  wr_pc_out  <= '1';
                  nxt_state <= S0;

when SFd     => wr_mar_out <= '0';
                  wr_dmem_out <= '0';
                  wr_rf_out  <= '1';
                  wr_pc_out  <= '1';
                  nxt_state <= S0;

when others  => null;
end case; -- cur_state
end process;
end behavioral;

```

**Listing 9 Ο VHDL κώδικας για την υλοποίηση της Ακολουθιακής Μονάδας Ελέγχου**

## 4. ΕΠΑΛΗΘΕΥΣΗ ΟΡΩΣ ΣΧΕΔΙΑΣΗΣ ΚΑΘΕ ΒΑΣΙΚΟΥ ΚΥΚΛΩΜΑΤΟΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

### A. ΕΠΠΕΔΟ ΔΙΟΔΟΥ ΔΕΔΟΜΕΝΩΝ (DATAPATH)

#### i. Αριθμητική και Λογική Μονάδα (ALU)

Για τον έλεγχο ορθής σχεδίασης της Αριθμητικής και Λογικής Μονάδας, αναπτύχθηκε κατάλληλο test bench με χρήση εσωτερικού αρχείου διανυσμάτων δοκιμής. Επιπλέον, κατά την προσομοίωση του test bench γίνεται αυτόματος έλεγχος των αποκρίσεων της ALU, ενώ η συμπεριφορά της ελέγχεται και για μη υποστηριζόμενες λειτουργίες.

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

entity alu_tb is
end alu_tb;

architecture behavior of alu_tb is

-- test-bench constants
constant test_pattern_no : integer := 27;
constant propagation_delay : time := 30 ns;
constant loop_delay : time := 100 ns;

constant ADDU : std_logic_vector(3 downto 0) := "1001";
constant SUBU : std_logic_vector(3 downto 0) := "1011";
constant SLLR : std_logic_vector(3 downto 0) := "0000";
constant SLT : std_logic_vector(3 downto 0) := "0110";
constant XORI : std_logic_vector(3 downto 0) := "1110";

type tb_signals is record
    bus_a : std_logic_vector(31 downto 0);
    bus_b : std_logic_vector(31 downto 0);
    shamt : std_logic_vector(4 downto 0);
    aluop : std_logic_vector(3 downto 0);
    sv : std_logic;
    bus_s : std_logic_vector(31 downto 0);
    zero : std_logic;
end record;

type tb_file is array (0 to test_pattern_no-1) of tb_signals;

constant tb_pattern_table : tb_file :=(
    ADDU,   (bus_a => X"10228080", bus_b => X"10020080", shamt => "00110", aluop =>
    "0"), -- ADDU
    SUBU,   (bus_a => X"97623B51", bus_b => X"7C95C360", shamt => "10100", aluop =>
    "0"), -- SUBU
    XORI,   (bus_a => X"16C75A94", bus_b => X"92D786F4", shamt => "10111", aluop =>
    "0"), -- XORI
    SLLR,   (bus_a => X"4189A25C", bus_b => X"D457E1F0", shamt => "01100", aluop =>
    "0"), -- SLL
    SLT,    (bus_a => X"621D65A2", bus_b => X"297369B3", shamt => "01101", aluop =>
    "1"), -- SLT is not satisfied,
    check critical path
    (bus_a => X"19174832", bus_b => X"76113468", shamt => "00010", aluop =>
    "0011", sv => '-', bus_s => X"00000000", zero => '0'), -- UNSUPPORTED OPERATION
    (bus_a => X"7890120E", bus_b => X"05A998F1", shamt => "10011", aluop =>
    ADDU,   sv => '-', bus_s => X"7E39AAFF", zero => '0'), -- ADDU
);
```

```

        (bus_a => X"24088000", bus_b => X"80244004", shamt => "01101", aluop =>
SUBU, sv => '-', bus_s => X"A3E43FFC", zero => '0'),-- SUBU
        (bus_a => X"A49B314E", bus_b => X"B9147E4A", shamt => "01111", aluop =>
XORI, sv => '-', bus_s => X"1D8F4F04", zero => '0'),-- XORI
        (bus_a => X"4189A25C", bus_b => X"D457E1F0", shamt => "11100", aluop =>
SLLR, sv => '-', bus_s => X"00000000", zero => '1'),-- SLL
        (bus_a => X"1792B693", bus_b => X"368A7921", shamt => "11001", aluop =>
SLT, sv => '-', bus_s => X"00000001", zero => '0'),-- SLT is satisfied,
check critical path
        (bus_a => X"61F56DF2", bus_b => X"7C7C8493", shamt => "01011", aluop =>
"1100", sv => '-', bus_s => X"00000000", zero => '0'),-- UNSUPPORTED OPERATION

        (bus_a => X"00000000", bus_b => X"00000000", shamt => "11110", aluop =>
SLT, sv => '-', bus_s => X"00000000", zero => '1'),-- SLT
        (bus_a => X"00000000", bus_b => X"00000000", shamt => "11101", aluop =>
XORI, sv => '-', bus_s => X"00000000", zero => '1'),-- XORI
        (bus_a => X"00000000", bus_b => X"00000000", shamt => "11011", aluop =>
ADDU, sv => '-', bus_s => X"00000000", zero => '1'),-- ADDU
        (bus_a => X"00000000", bus_b => X"00000000", shamt => "10111", aluop =>
SLLR, sv => '-', bus_s => X"00000000", zero => '1'),-- SLL
        (bus_a => X"00000000", bus_b => X"00000000", shamt => "01111", aluop =>
SUBU, sv => '-', bus_s => X"00000000", zero => '1'),-- SUBU

        (bus_a => X"00000000", bus_b => X"FFFFFFFF", shamt => "11010", aluop =>
ADDU, sv => '-', bus_s => X"FFFFFFFF", zero => '0'),-- ADDU
        (bus_a => X"00000000", bus_b => X"FFFFFFFE", shamt => "11111", aluop =>
SLLR, sv => '-', bus_s => X"00000000", zero => '1'),-- SLL check critical path
        (bus_a => X"00000000", bus_b => X"FFFFFFFF", shamt => "01110", aluop =>
XORI, sv => '-', bus_s => X"FFFFFFFF", zero => '0'),-- XORI check critical
path
        (bus_a => X"00000000", bus_b => X"FFFFFFFF", shamt => "11001", aluop =>
SUBU, sv => '-', bus_s => X"00000001", zero => '0'),-- SUBU
        (bus_a => X"00000000", bus_b => X"FFFFFFFF", shamt => "01010", aluop =>
SLT, sv => '-', bus_s => X"00000000", zero => '1'),-- SLT check critical path

        (bus_a => X"FFFFFFFF", bus_b => X"FFFFFFFF", shamt => "10000", aluop =>
SLT, sv => '-', bus_s => X"00000000", zero => '1'),-- SLT
        (bus_a => X"FFFFFFFF", bus_b => X"FFFFFFFF", shamt => "00001", aluop =>
ADDU, sv => '-', bus_s => X"FFFFFFFE", zero => '0'),-- ADDU check critical
path
        (bus_a => X"FFFFFFFF", bus_b => X"FFFFFFFF", shamt => "00010", aluop =>
SUBU, sv => '-', bus_s => X"00000000", zero => '1'),-- SUBU check critical
path
        (bus_a => X"FFFFFFFF", bus_b => X"FFFFFFFF", shamt => "10011", aluop =>
SLLR, sv => '-', bus_s => X"FFF80000", zero => '0'),-- SLL
        (bus_a => X"FFFFFFFF", bus_b => X"FFFFFFFF", shamt => "01000", aluop =>
XORI, sv => '-', bus_s => X"00000000", zero => '1') -- XORI check critical
path
    );
-- Component Declaration for the Unit Under Test (UUT)
component alu
port(
    bus_a_in : in std_logic_vector(31 downto 0);
    bus_b_in : in std_logic_vector(31 downto 0);
    shamt : in std_logic_vector(4 downto 0);
    aluop : in std_logic_vector(3 downto 0);
    sv : in std_logic;

    bus_s_out : out std_logic_vector(31 downto 0);
    zero_out : out std_logic
);
end component;

--Inputs

```

```

signal bus_a_in : std_logic_vector(31 downto 0) := (others => '0');
signal bus_b_in : std_logic_vector(31 downto 0) := (others => '0');
signal shamt    : std_logic_vector(4 downto 0)  := (others => '0');
signal aluop    : std_logic_vector(3 downto 0)  := (others => '0');
signal sv       : std_logic := '0';

--Outputs
signal bus_s_out : std_logic_vector(31 downto 0);
signal zero_out : std_logic;

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: alu PORT MAP (
    bus_a_in => bus_a_in,
    bus_b_in => bus_b_in,
    shamt => shamt,
    aluop => aluop,
    sv => sv,
    bus_s_out => bus_s_out,
    zero_out => zero_out
);

-- Stimulus process
stim_proc: process
begin
    for i in 0 to test_pattern_no-1 loop
        bus_a_in  <= tb_pattern_table(i).bus_a;
        bus_b_in  <= tb_pattern_table(i).bus_b;
        shamt     <= tb_pattern_table(i).shamt;
        aluop     <= tb_pattern_table(i).aluop;
        sv        <= tb_pattern_table(i).sv;
        wait for propagation_delay;

        assert (bus_s_out = tb_pattern_table(i).bus_s and zero_out =
tb_pattern_table(i).zero)
            report "Failed Respond"
            severity error;

        wait for loop_delay - propagation_delay;
    end loop;

    wait;
end process;

END;

```

**Listing 10 To VHDL test bench για τον έλεγχο λειτουργίας της Αριθμητικής και Λογικής Μονάδας**

Από την εκτέλεση των Behavioral και Post-Route Simulation προέκυψαν σύμφωνα αποτελέσματα. Και οι δύο προσομοιώσεις (Behavioral και Post-Route) που έγιναν με το παραπάνω test bench, επαληθεύουν την ορθή λειτουργία της Αριθμητικής και Λογικής Μονάδας. Τα αποτελέσματα των δύο προσομοιώσεων παρουσιάζονται στα παρακάτω σχήματα.

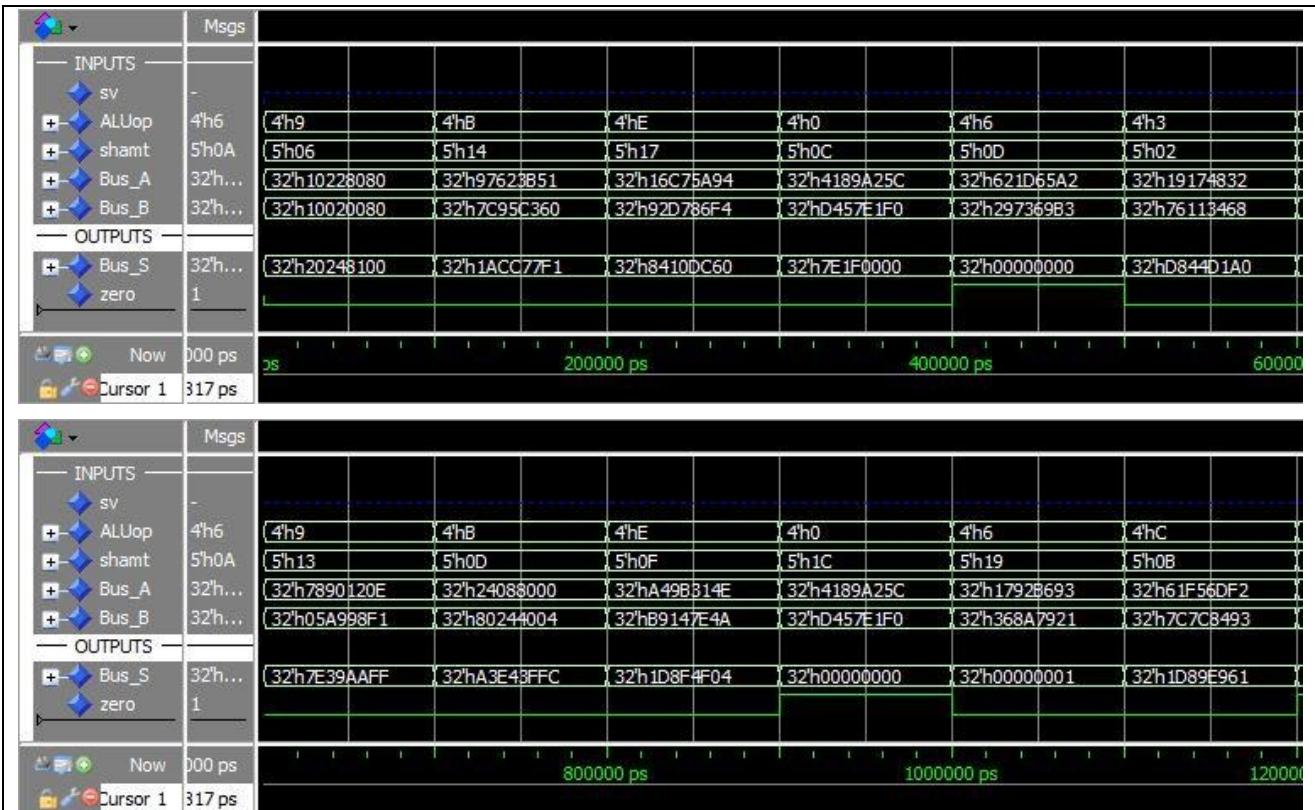


Figure 10. Τα αποτελέσματα του Behavioral simulation της Αριθμητικής και Λογικής Μονάδας (ALU) - απόσπασμα

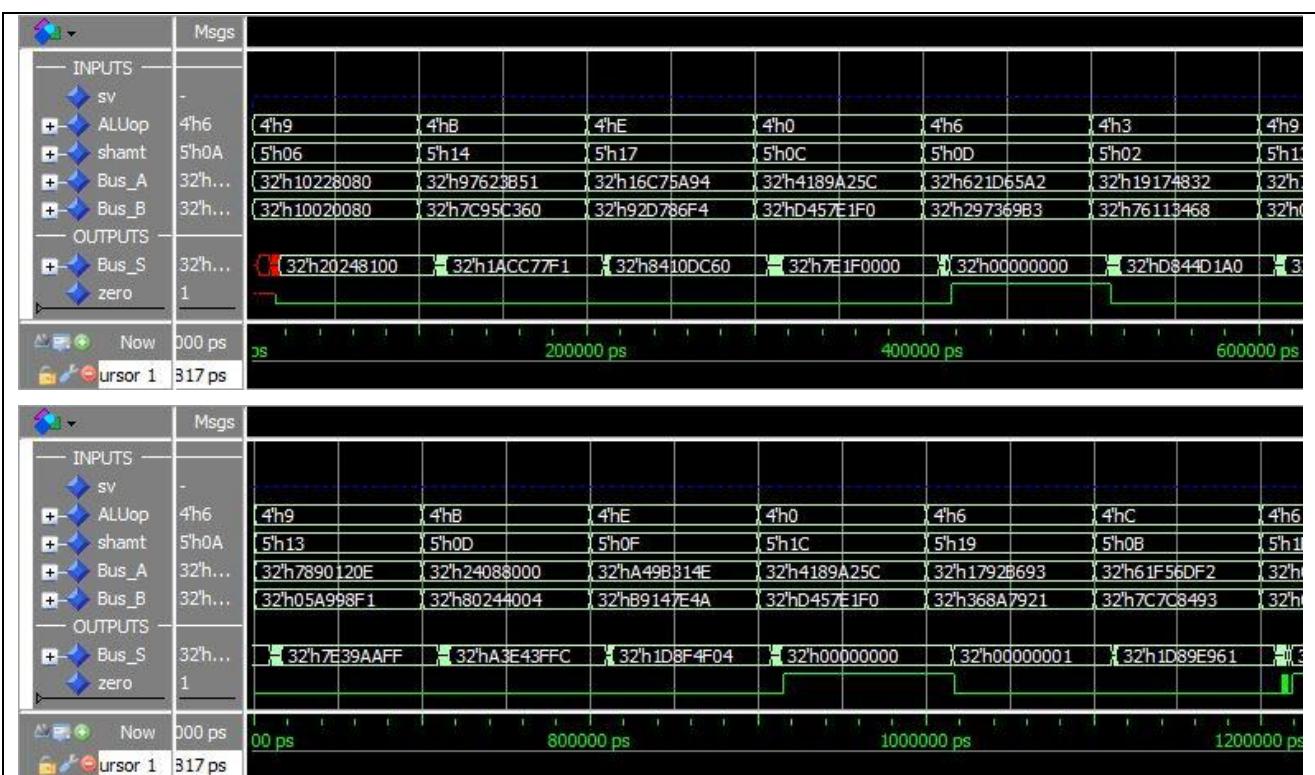


Figure 11. Τα αποτελέσματα του Post-Route simulation της Αριθμητικής και Λογικής Μονάδας (ALU) - απόσπασμα

Η υλοποίηση της Αριθμητικής και Λογικής Μονάδας παράγει το Post Place and Route Stating Timing Report της Αριθμητικής και Λογικής Μονάδας, από το οποίο προκύπτει το critical path. Σύμφωνα με το report, το critical path της ALU εντοπίζεται μεταξύ της εισόδου shamt(0) και της εξόδου zero\_out, με χρονική καθυστέρηση διάδοσης 20.139 ns. Από την κυματομορφή του Post-Route Simulation η αντίστοιχη καθυστέρηση υπολογίζεται στα 17.020 ns, τιμή αρκετά κοντά στην προβλεπόμενη.

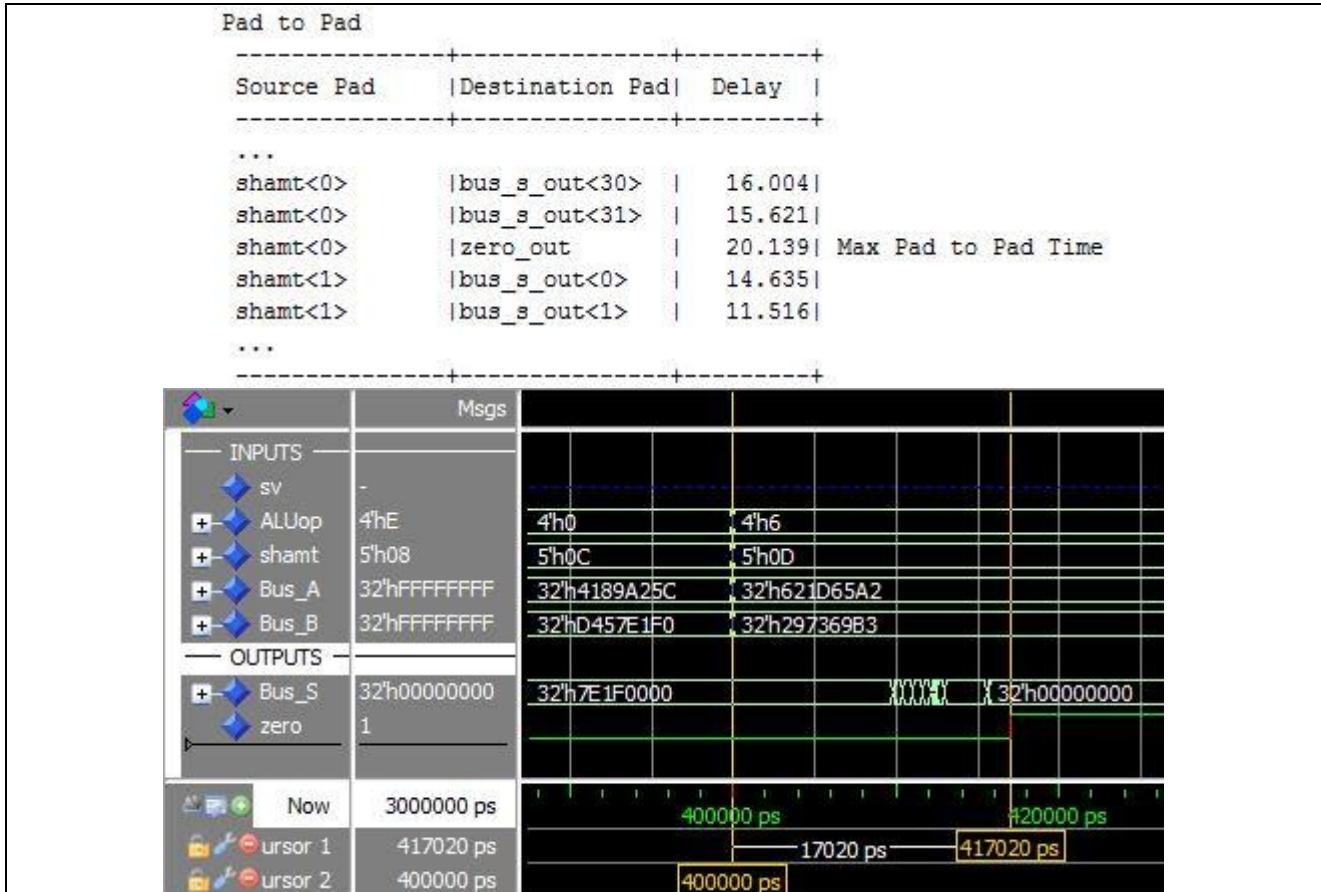


Figure 12 Η καθυστέρηση διάδοσης του critical path της Αριθμητικής και Λογικής Μονάδας

### ii. Αρχείο Καταχωρητών (RF)

Για τον έλεγχο ορθής σχεδίασης του Αρχείου Καταχωρητών, αναπτύχθηκε κατάλληλο test bench με χρήση δύο δομών for loops. Κατά την πρώτη for loop, πραγματοποιούνται εγγραφές στους καταχωρητές reg[rt = i] και reg[rd = 1+16], εναλλάξ, ενώ μετά από κάθε εγγραφή, το σήμα ενεργοποίησης εγγραφής reg\_write, απενεργοποιείται (τίθεται σε 0, για θετική λογική), ώστε να αναγνωστεί το νέο περιεχόμενο των καταχωρητών reg[i] και reg[i+16]. Μετά την ολοκλήρωση των εγγραφών σε όλους τους καταχωρητές, μια δεύτερη for loop προσομοιώνει την ανάγνωση όλων των καταχωρητών, επιβεβαιώνοντας την ορθή εγγραφή και ανάγνωση όλων των καταχωρητών του Register File.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY reg_file_tb IS
END reg_file_tb;

```

```

ARCHITECTURE behavior OF reg_file_tb IS

-- Component Declaration for the Unit Under Test (UUT)
COMPONENT reg_file
PORT (
    clk : IN std_logic;
    rst : IN std_logic;
    reg_wr : IN std_logic;
    reg_imm : IN std_logic;
    rd_in : IN std_logic_vector(4 downto 0);
    rt_in : IN std_logic_vector(4 downto 0);
    rs_in : IN std_logic_vector(4 downto 0);
    bus_w_in : IN std_logic_vector(31 downto 0);
    bus_a_out : OUT std_logic_vector(31 downto 0);
    bus_b_out : OUT std_logic_vector(31 downto 0)
);
END COMPONENT;

--Inputs
signal clk : std_logic;
signal rst : std_logic;
signal reg_wr : std_logic;
signal reg_imm : std_logic;
signal rd_in : std_logic_vector(4 downto 0);
signal rt_in : std_logic_vector(4 downto 0);
signal rs_in : std_logic_vector(4 downto 0);
signal bus_w_in : std_logic_vector(31 downto 0);

--Outputs
signal bus_a_out : std_logic_vector(31 downto 0);
signal bus_b_out : std_logic_vector(31 downto 0);

-- Clock period definitions
constant clk_period : time := 100 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
--HDL
    uut: reg_file PORT MAP (
        clk => clk,
        rst => rst,
        reg_wr => reg_wr,
        reg_imm => reg_imm,
        rd_in => rd_in,
        rt_in => rt_in,
        rs_in => rs_in,
        bus_w_in => bus_w_in,
        bus_a_out => bus_a_out,
        bus_b_out => bus_b_out
    );

-- Clock process definitions
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period - clk_period/2;
end process;

rst <= '1', '0' after clk_period/2;

-- Stimulus process
stim_proc: process

```

```

begin

    reg_wr      <=  '0';

    -- write reg
    for i in 0 to 15 loop
        wait until rising_edge(clk);
        wait for clk_period/10; -- test vector remain stable for 10 ns after clk's
    rising edge
        -- write reg[rt = i+16] value i
        reg_wr      <=  '1';
        reg_imm    <=  '0';
        rd_in      <=  conv_std_logic_vector(i, 5);          -- does not used on write
    operation
        rt_in      <=  conv_std_logic_vector(i+16, 5);     -- write bus_w_in value on
    reg[i+16]
        rs_in      <=  conv_std_logic_vector(i, 5);
        bus_w_in   <=  conv_std_logic_vector(i, 32);

        wait for clk_period;
        reg_wr      <=  '0';

        wait until rising_edge(clk);
        wait for clk_period/10; -- test vector remain stable for 10 ns after clk's
    rising edge
        -- write reg[rd = i] value i+16
        reg_wr      <=  '1';
        reg_imm    <=  '1';
        rd_in      <=  conv_std_logic_vector(i, 5);          -- write bus_w_in value on
    reg[i]
        rt_in      <=  conv_std_logic_vector(i+16, 5);     -- does not used on write
    operation
        rs_in      <=  conv_std_logic_vector(i, 5);
        bus_w_in   <=  conv_std_logic_vector(i+16, 32);

        wait for clk_period;
        reg_wr      <=  '0';

    end loop;

    wait for 2*clk_period;

    -- read reg
    for i in 0 to 15 loop
        wait until rising_edge(clk);
        wait for clk_period/10; -- test vector remain stable for 10 ns after clk's
    rising edge
        reg_imm    <=  '0';
        rd_in      <=  conv_std_logic_vector(i+16, 5);
        rt_in      <=  conv_std_logic_vector(i, 5);
        rs_in      <=  conv_std_logic_vector(i+16, 5);
        bus_w_in   <=  conv_std_logic_vector(16732, 32);
    end loop;

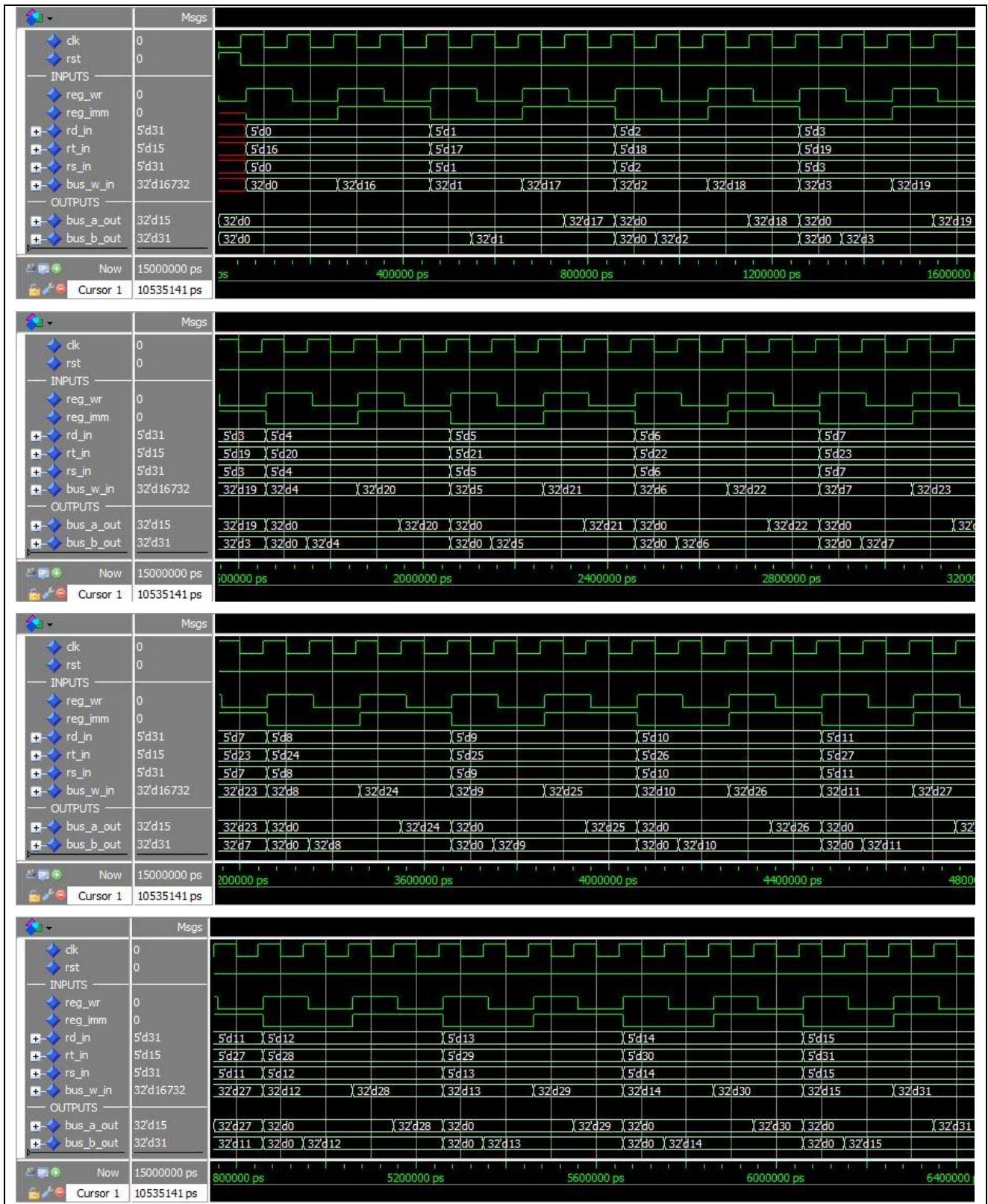
    wait;
end process;

END;

```

**Listing 11 To VHDL test bench για τον έλεγχο λειτουργίας του Αρχείου Καταχωρητών**

Τα αποτελέσματα που προέκυψαν από την εκτέλεση των Behavioral και Post-Route Simulation, παρουσιάζονται στις επόμενες σελίδες. Όπως φαίνεται και στις δύο κυματομορφές, το παραπάνω test bench επαληθεύει την ορθή σχεδίαση και λειτουργία του Αρχείου Καταχωρητών που υλοποιήθηκε.



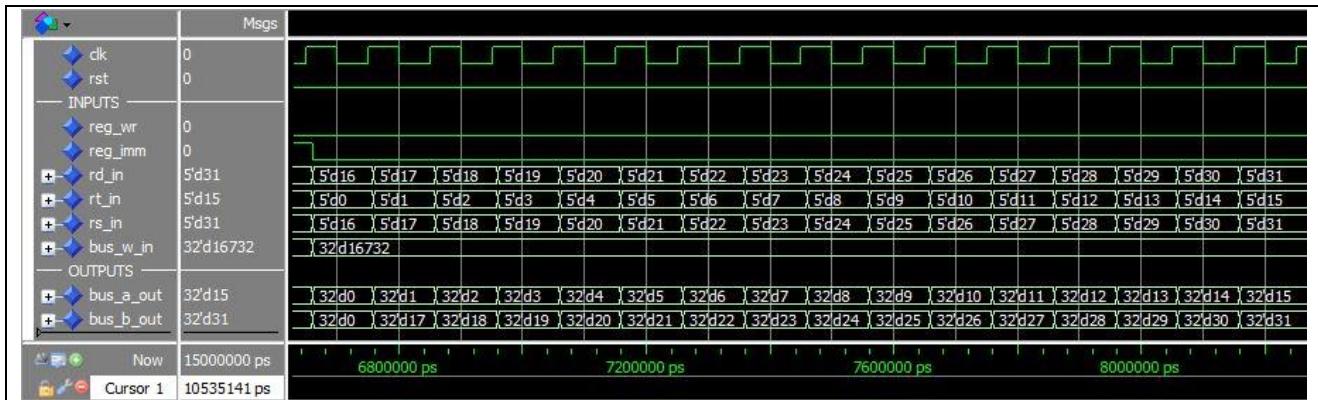
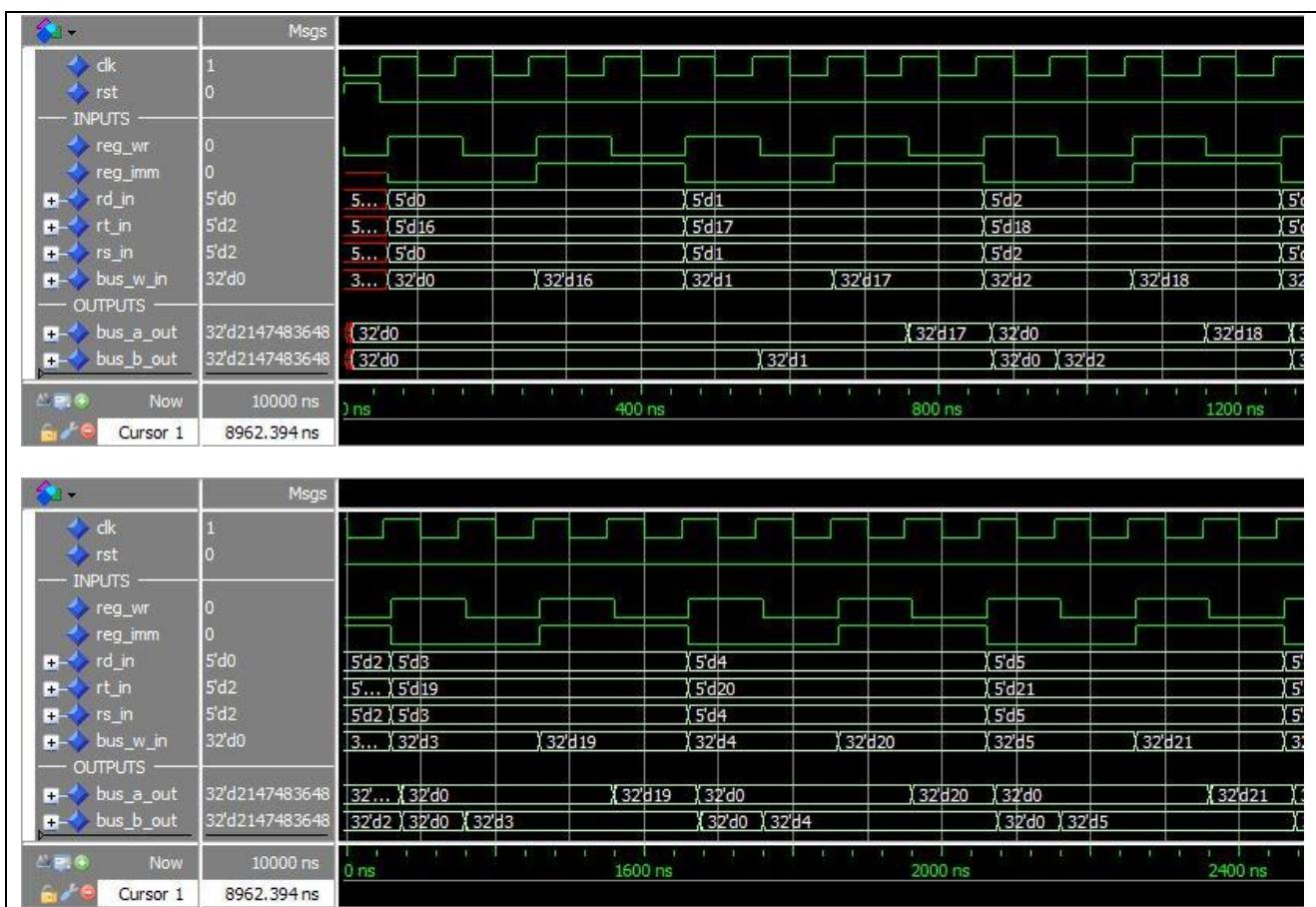
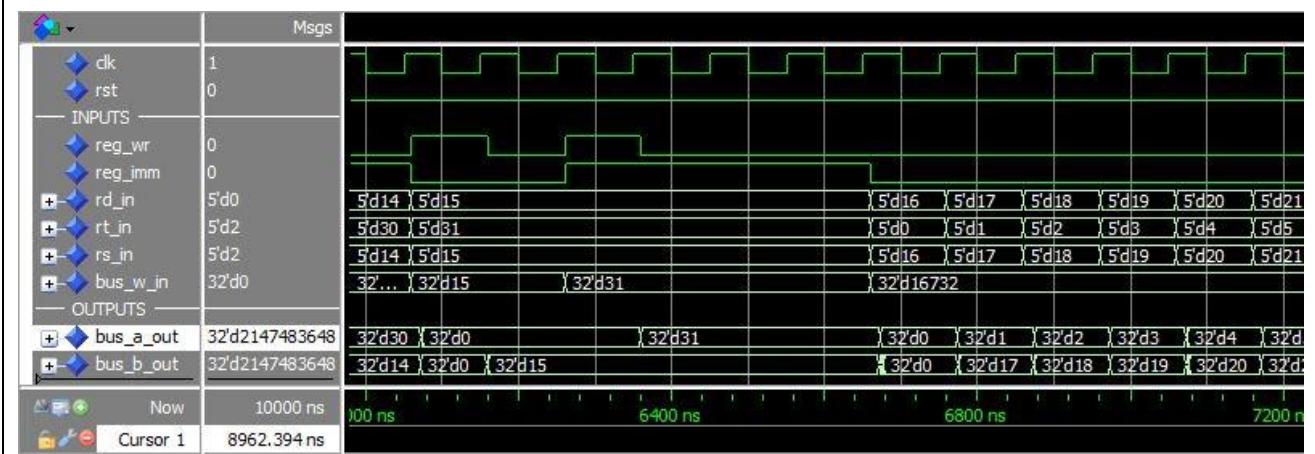
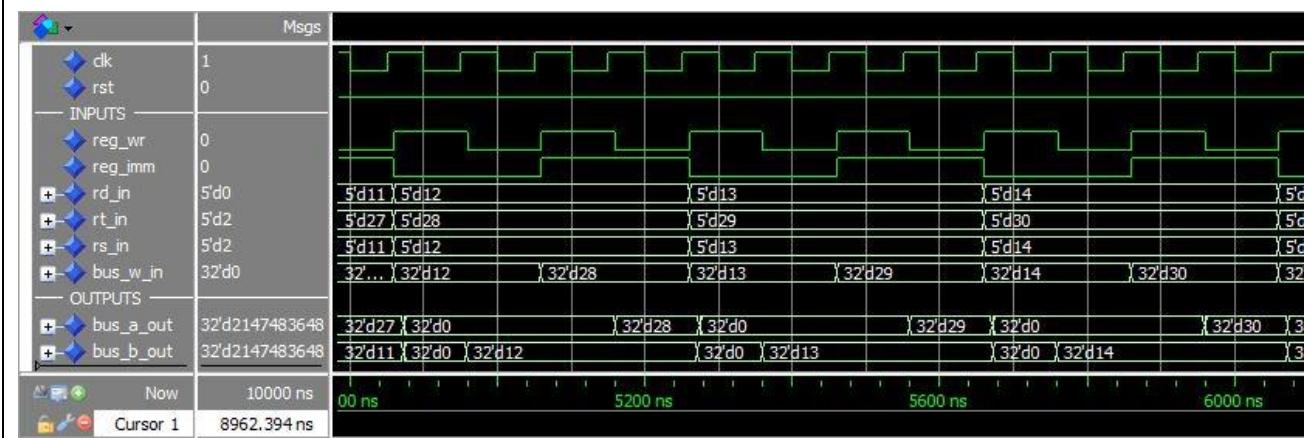
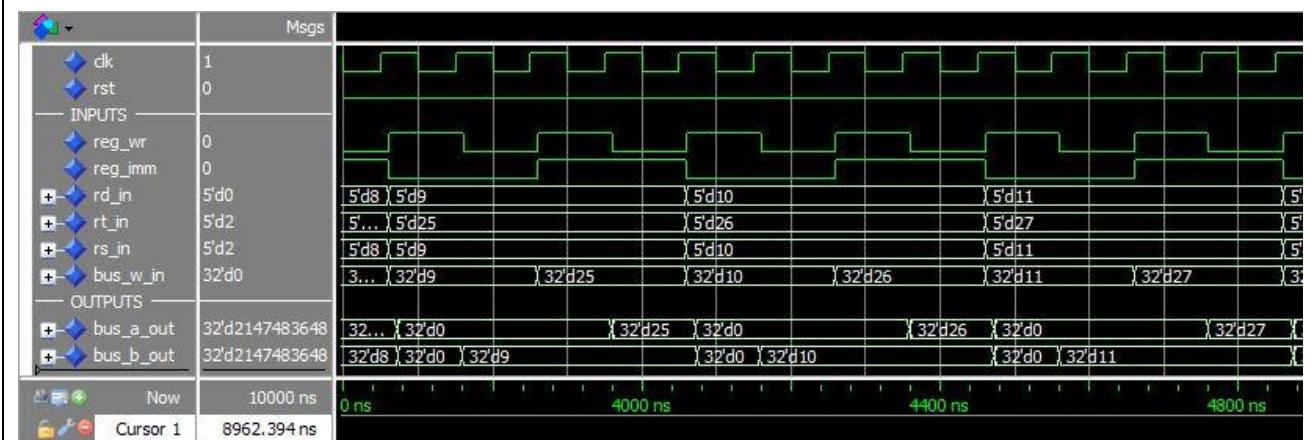
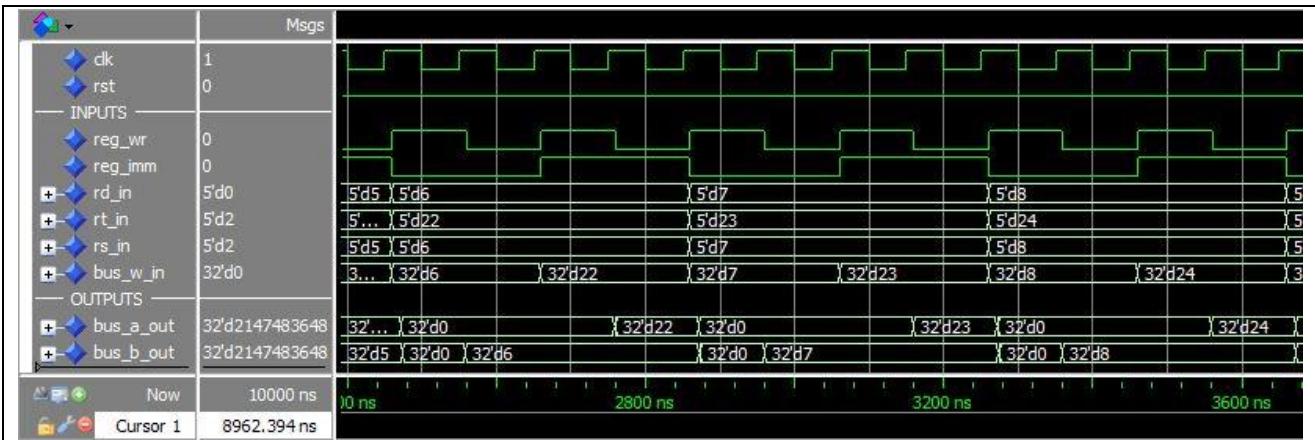


Figure 13 Τα αποτελέσματα του Behavioral simulation του Αρχείου Καταχωρητών (Top-Level Register File)





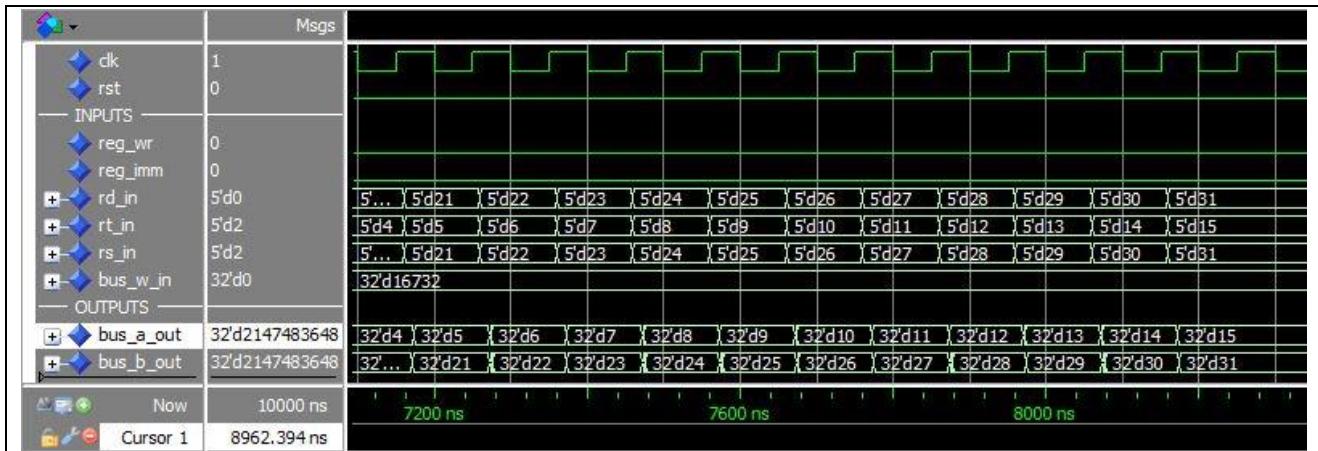


Figure 14 Τα αποτελέσματα του Post-Route Simulation του Αρχείου Καταχωρητών (Top-Level Register File)

Όλες οι πληροφορίες για τις καθυστερήσεις και τον ορθό χρονισμό του Αρχείου Καταχωρητών, δίνονται από το Post, Place and Route Static Timing Report που παράγεται κατά την υλοποίηση. Σύμφωνα με το report, για τον ορθό χρονισμό του κυκλώματος, η είσοδος απαιτείται να εφαρμόζεται τουλάχιστον για 12.258 ns πριν την ανερχόμενη ακμή του ρολογιού (max Set-up time), και να παραμένει σταθερή (stable) για τουλάχιστον 1.613 ns (max Hold time) πριν/μετά (?) την ανερχόμενη ακμή του ρολογιού.

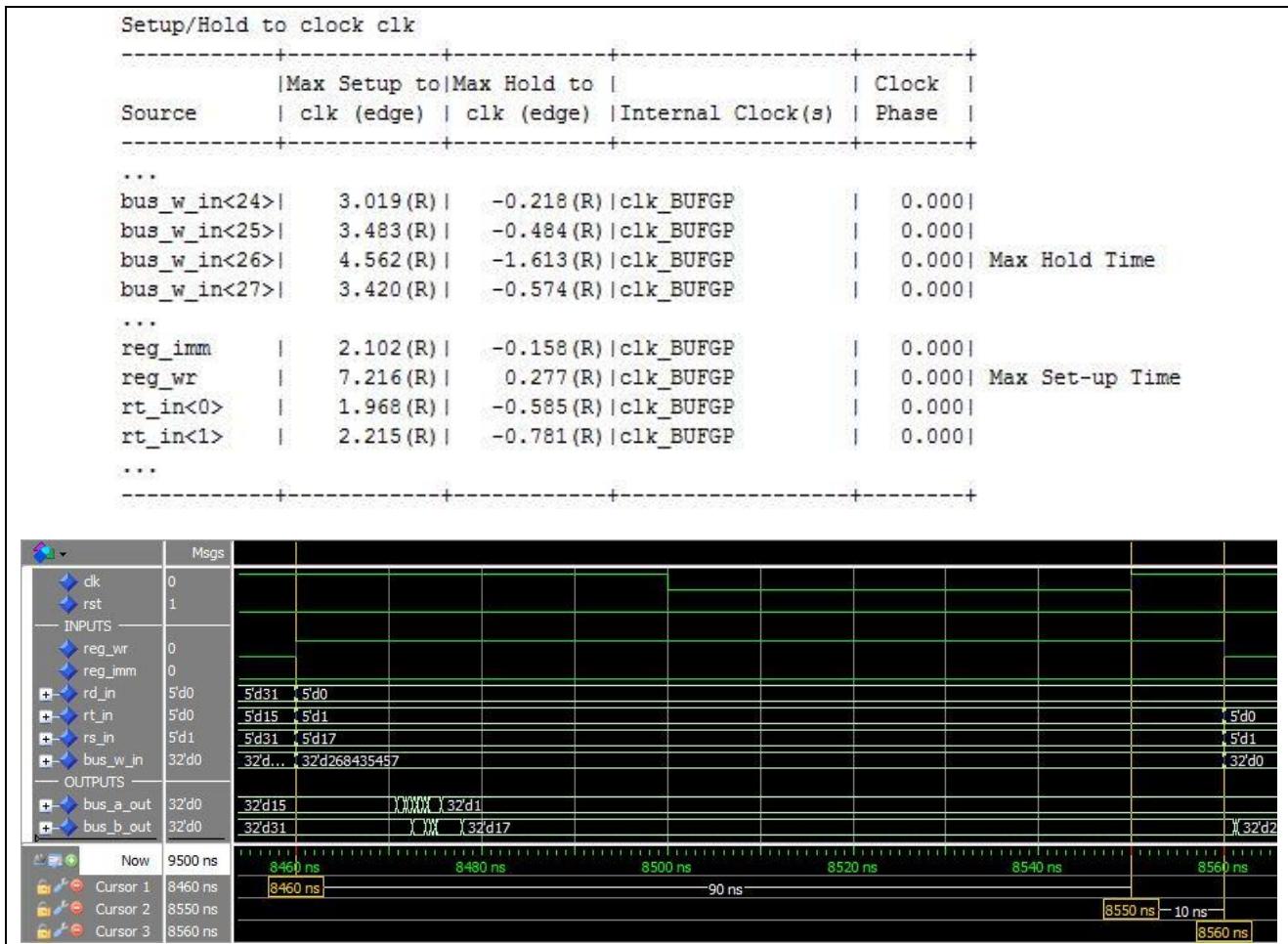


Figure 15 Οι παράμετροι χρονισμού του Αρχείου Καταχωρητών (Top-Level Register File)

Από την ανάλυση του Timing Report, προκύπτουν επίσης και οι καθυστερήσεις διάδοσης. Η μέγιστη καθυστέρηση διάδοσης από την ανερχόμενη ακμή του ρολογιού, για την έξοδο bus\_a\_out(31), υπολογίζεται στα 14.205 ns. Αντίστοιχα, το critical path του κυκλώματος, μεταξύ των σήματος rst (Reset) και της εξόδου bus\_a\_out(28), παρουσιάζει μέγιστη καθυστέρηση διάδοσης στα 21.300 ns.

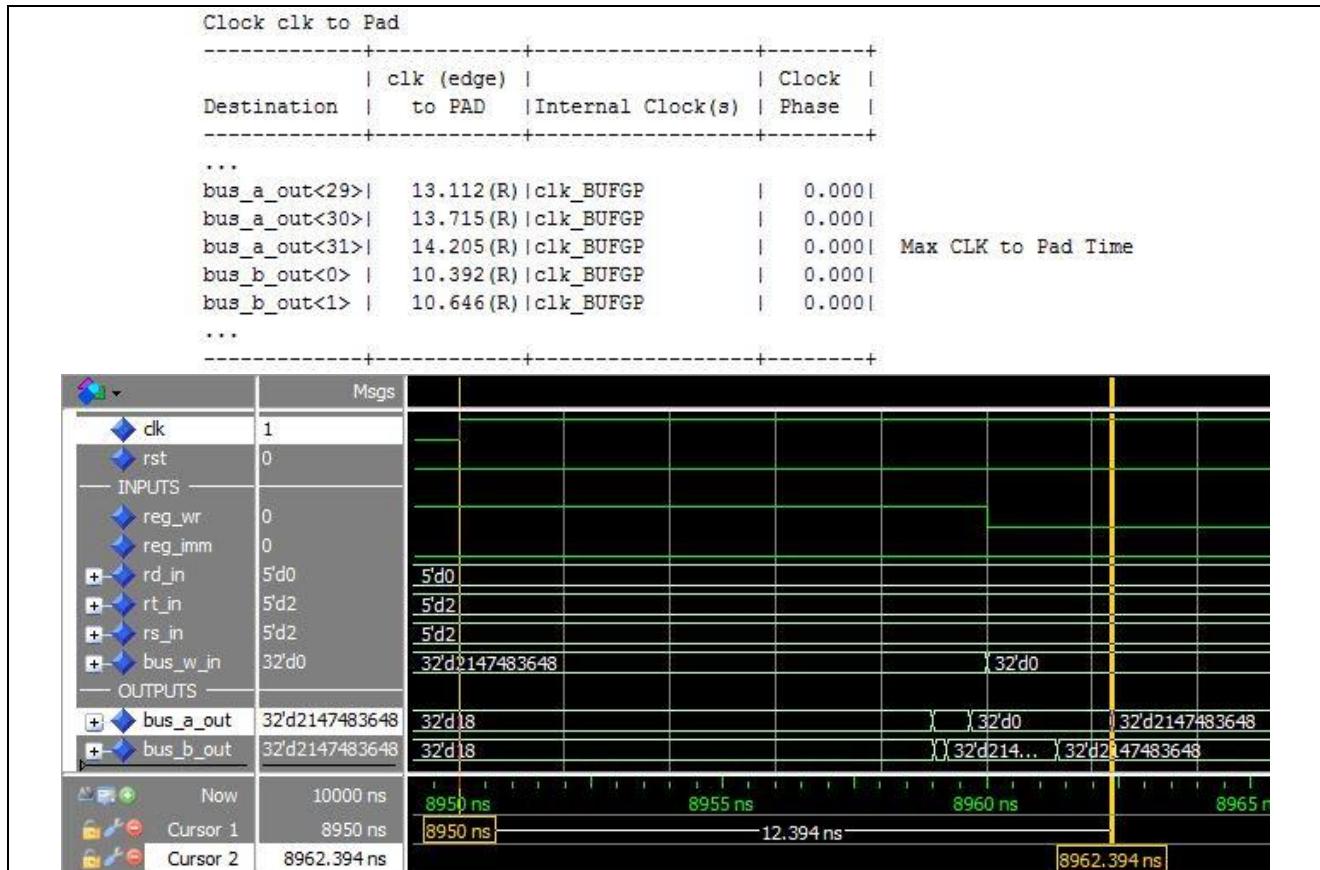


Figure 16 Η μέγιστη καθυστέρηση διάδοσης από την ανερχόμενη ακμή του ρολογιού του Αρχείου Καταχωρητών

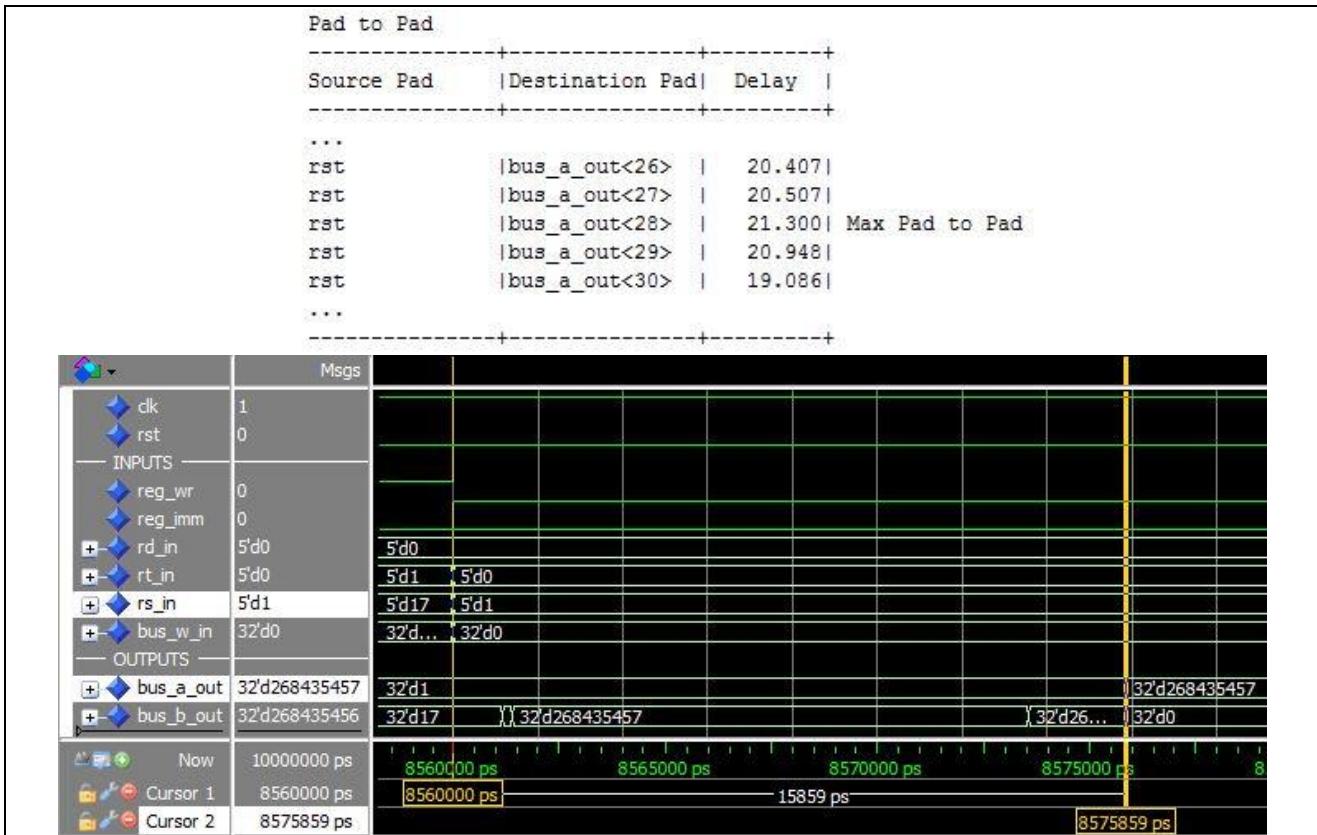


Figure 17 Η καθυστέρηση διάδοσης του critical path του Αρχείου Καταχωρητών

## B. ΕΠΙΠΕΔΟ ΕΠΕΞΕΡΓΑΣΤΗ (PROCESSOR – TOP LEVEL)

### i. Συνδυαστική Μονάδα Ελέγχου

Για τον έλεγχο ορθής σχεδίασης της Συνδυαστικής Μονάδας Ελέγχου, αναπτύχθηκε κατάλληλο test bench, επίσης, με χρήση εσωτερικού αρχείου διανυσμάτων δοκιμής. Κατά την προσομοίωση του test bench ελέγχεται η συμπεριφορά της Συνδυαστικής Μονάδας και για μη υποστηριζόμενες λειτουργίες.

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY cu_comb_tb IS
END cu_comb_tb;

ARCHITECTURE behavior OF cu_comb_tb IS

    -- "opcode" codes definition as constants
    constant RTYPE : std_logic_vector(5 downto 0) := "000000"; -- 0x00
    constant LW : std_logic_vector(5 downto 0) := "100011"; -- 0x23
    constant SW : std_logic_vector(5 downto 0) := "101011"; -- 0x2B
    constant XORI : std_logic_vector(5 downto 0) := "001110"; -- 0x0E
    constant BNE : std_logic_vector(5 downto 0) := "000101"; -- 0x05

    -- "funct" codes definition as constants
    constant SLLR : std_logic_vector(5 downto 0) := "000000"; -- 0x00

```

```

constant ADDU : std_logic_vector(5 downto 0) := "100001"; -- 0x21
constant SUBU : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SLT : std_logic_vector(5 downto 0) := "101010"; -- 0x2A
constant JR : std_logic_vector(5 downto 0) := "001000"; -- 0x08
constant JALR : std_logic_vector(5 downto 0) := "001001"; -- 0x09

-- test-bench constants
constant test_pattern_no : integer := 15;
constant propagation_delay : time := 10 ns;
constant loop_delay : time := 100 ns;

type tb_signals is record
    opcode_in : std_logic_vector(5 downto 0);
    funct_in : std_logic_vector(5 downto 0);

    link_out : std_logic;
    jump_out : std_logic;
    branch_out : std_logic;

    sel_regimm_out : std_logic;
    sel_bimm_out : std_logic;
    sel_sigzer_out : std_logic;
    sel_dmalu_out : std_logic;

    ALUop_out : std_logic_vector(3 downto 0);
end record;

type tb_file is array (0 to test_pattern_no-1) of tb_signals;

constant tb_pattern_table : tb_file :=(
    (opcode_in => LW,          funct_in => "-----", link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '0', sel_bimm_out => '0',
sel_sigzer_out => '1', sel_dmalu_out => '1', ALUop_out => "1001"),--01 LW
    (opcode_in => SW,          funct_in => "-----", link_out => '-', jump_out
=> '0', branch_out => '0', sel_regimm_out => '-', sel_bimm_out => '0',
sel_sigzer_out => '1', sel_dmalu_out => '-', ALUop_out => "1001"),--02 SW
    (opcode_in => XORI,        funct_in => "-----", link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '0', sel_bimm_out => '0',
sel_sigzer_out => '0', sel_dmalu_out => '0', ALUop_out => "1110"),--03 XORI
    (opcode_in => "001100",   funct_in => "-----", link_out => '-', jump_out
=> '-', branch_out => '-', sel_regimm_out => '-', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--04 ANDI not
supported
    (opcode_in => BNE,         funct_in => "-----", link_out => '-', jump_out
=> '0', branch_out => '1', sel_regimm_out => '-', sel_bimm_out => '1',
sel_sigzer_out => '1', sel_dmalu_out => '-', ALUop_out => "1011"),--05 BNE
    (opcode_in => RTYPE,       funct_in => "100110", link_out => '-', jump_out
=> '-', branch_out => '-', sel_regimm_out => '-', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--06 XOR not
supported
    (opcode_in => RTYPE,       funct_in => SLLR,      link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '1', sel_bimm_out => '1',
sel_sigzer_out => '-', sel_dmalu_out => '0', ALUop_out => "0000"),--07 SLL
    (opcode_in => RTYPE,       funct_in => ADDU,       link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '1', sel_bimm_out => '1',
sel_sigzer_out => '-', sel_dmalu_out => '0', ALUop_out => "1001"),--08 ADDU
    (opcode_in => "000100",   funct_in => "-----", link_out => '-', jump_out
=> '-', branch_out => '-', sel_regimm_out => '-', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--09 BEQ not
supported
    (opcode_in => RTYPE,       funct_in => SUBU,      link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '1', sel_bimm_out => '1',
sel_sigzer_out => '-', sel_dmalu_out => '0', ALUop_out => "1011"),--10 SUBU
    (opcode_in => RTYPE,       funct_in => SLT,        link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '1', sel_bimm_out => '1',
sel_sigzer_out => '-', sel_dmalu_out => '0', ALUop_out => "0110"),--11 SLT
);

```

```

        (opcode_in => RTYPE,      funct_in => "000110", link_out => '-', jump_out
=> '-', branch_out => '-', sel_regimm_out => '-', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--12 SRLV not
supported
        (opcode_in => RTYPE,      funct_in => JR,      link_out => '-', jump_out
=> '1', branch_out => '-', sel_regimm_out => '-', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--13 JR
        (opcode_in => RTYPE,      funct_in => JALR,     link_out => '1', jump_out
=> '1', branch_out => '-', sel_regimm_out => '1', sel_bimm_out => '-',
sel_sigzer_out => '-', sel_dmalu_out => '-', ALUop_out => "----"),--14 JALR
        (opcode_in => XORI,       funct_in => "-----", link_out => '0', jump_out
=> '0', branch_out => '0', sel_regimm_out => '0', sel_bimm_out => '0',
sel_sigzer_out => '0', sel_dmalu_out => '0', ALUop_out => "1110") --15 XORI
);

-- Component Declaration for the Unit Under Test (UUT)
component cu_comb is
    port(
        opcode_in : in std_logic_vector(5 downto 0);
        funct_in : in std_logic_vector(5 downto 0);

        link_out : out std_logic;
        jump_out : out std_logic;
        branch_out : out std_logic;

        sel_regimm_out : out std_logic;
        sel_bimm_out : out std_logic;
        sel_sigzer_out : out std_logic;
        sel_dmalu_out : out std_logic;

        ALUop_out : out std_logic_vector(3 downto 0)
    );
end component;

--Inputs
signal opcode_in : std_logic_vector(5 downto 0) := (others => '0');
signal funct_in : std_logic_vector(5 downto 0) := (others => '0');

--Outputs
signal link_out : std_logic;
signal jump_out : std_logic;
signal branch_out : std_logic;
signal sel_regimm_out : std_logic;
signal sel_bimm_out : std_logic;
signal sel_sigzer_out : std_logic;
signal sel_dmalu_out : std_logic;
signal ALUop_out : std_logic_vector(3 downto 0);

BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: cu_comb
port map (
    opcode_in => opcode_in,
    funct_in => funct_in,
    link_out => link_out,
    jump_out => jump_out,
    branch_out => branch_out,
    sel_regimm_out => sel_regimm_out,
    sel_bimm_out => sel_bimm_out,
    sel_sigzer_out => sel_sigzer_out,
    sel_dmalu_out => sel_dmalu_out,
    ALUop_out => ALUop_out
);

```

```

-- Stimulus process
stim_proc: process
begin
    for i in 0 to test_pattern_no-1 loop
        opcode_in <= tb_pattern_table(i).opcode_in;
        funct_in <= tb_pattern_table(i).funct_in;
        wait for propagation_delay;

        assert (jump_out = tb_pattern_table(i).jump_out)
            report "Failed Respond"
            severity warning;

        wait for loop_delay - propagation_delay;
    end loop;

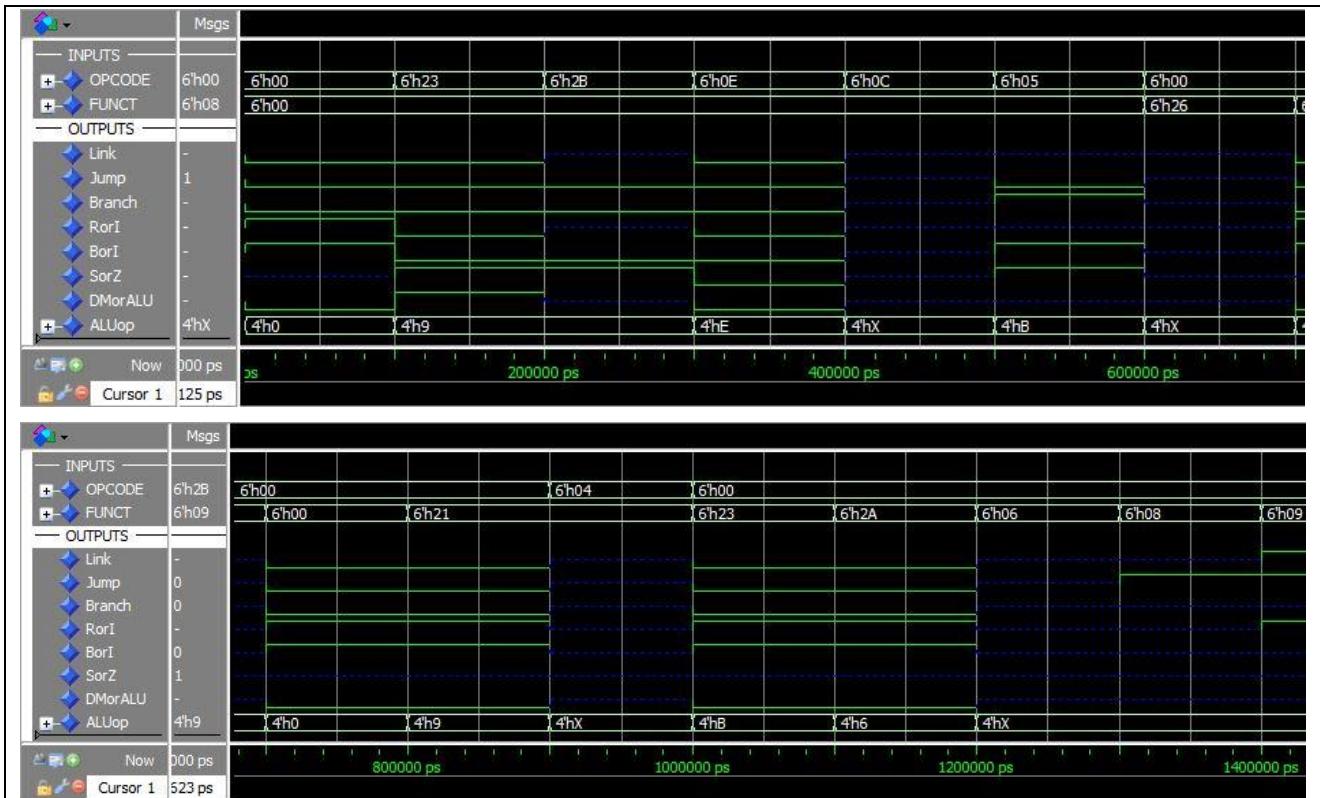
    wait;
end process;

END;

```

**Listing 12 To VHDL test bench για τον έλεγχο λειτουργίας της Συνδυαστικής Μονάδας Ελέγχου**

Από την εκτέλεση των Behavioral και Post-Route Simulation προέκυψαν σύμφωνα αποτελέσματα. Και οι δύο προσομοιώσεις (Behavioral και Post-Route) που έγιναν με το παραπάνω test bench, επαληθεύουν την ορθή λειτουργία της Συνδυαστικής Μονάδας Ελέγχου. Οι κυματομορφές που προέκυψαν από τις δύο προσομοιώσεις παρουσιάζονται στα επόμενα σχήματα.



**Figure 18 Τα αποτελέσματα του Behavioral simulation της Συνδυαστικής Μονάδας Ελέγχου**

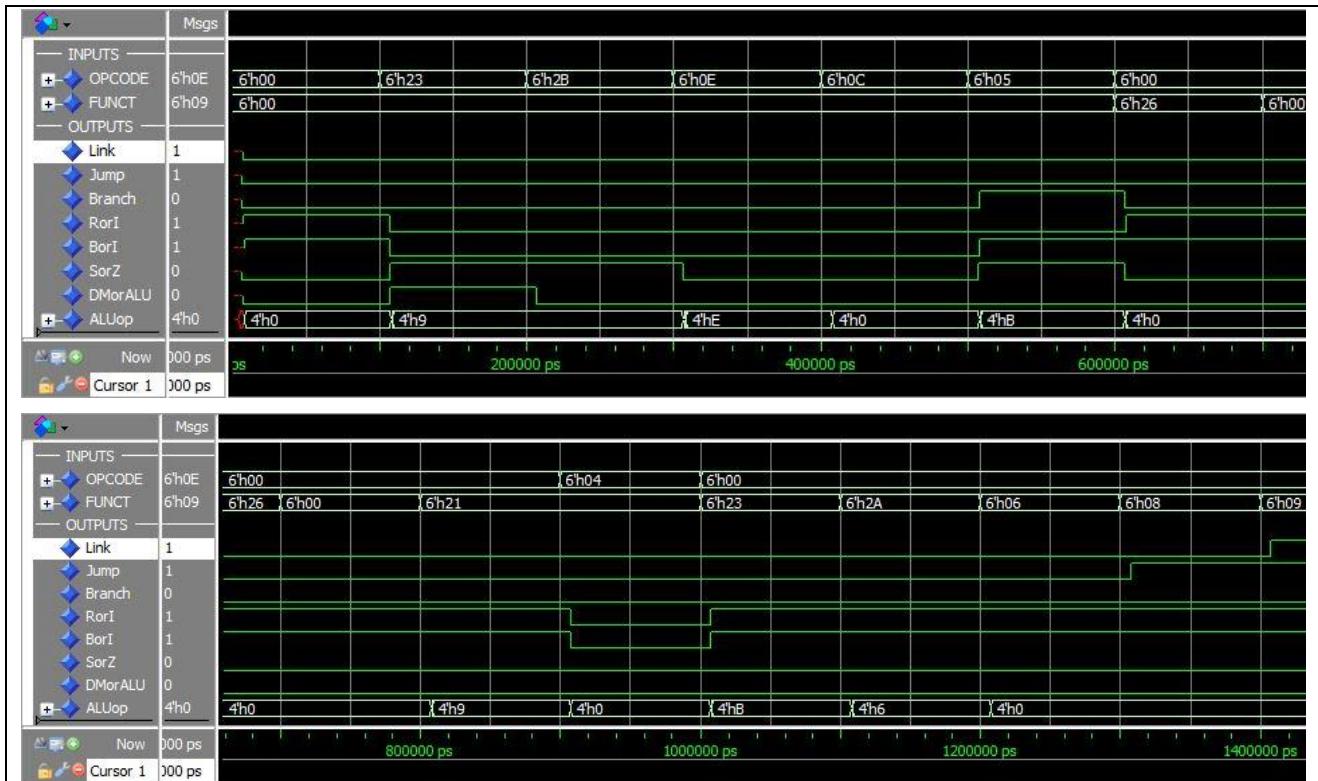


Figure 19 Τα αποτελέσματα του Post-Route simulation της Συνδυαστικής Μονάδας Ελέγχου

Όπως προκύπτει από τη μελέτη του Post Place and Route Stating Timing Report της Συνδυαστικής Μονάδας Ελέγχου, το critical path της εντοπίζεται μεταξύ της εισόδου opcode\_in(1) και της εξόδου link\_out, με χρονική καθυστέρηση διάδοσης 9.658 ns. Σύμφωνα με τις κυματομορφές που παρήχθησαν κατά την εκτέλεση του Post-Route Simulation, η αντίστοιχη καθυστέρηση υπολογίζεται στα 9.331 ns, τιμή που σχεδόν συμφωνεί με το PPR Static Timing Report.

Pad to Pad			
Source Pad	Destination Pad	Delay	
...			
opcode_in<1>	branch_out	7.544	
opcode_in<1>	jump_out	8.889	
opcode_in<1>	link_out	9.658	Max Pad to Pad Time
opcode_in<1>	sel_bimm_out	7.838	
opcode_in<1>	sel_dmalu_out	6.776	
...			

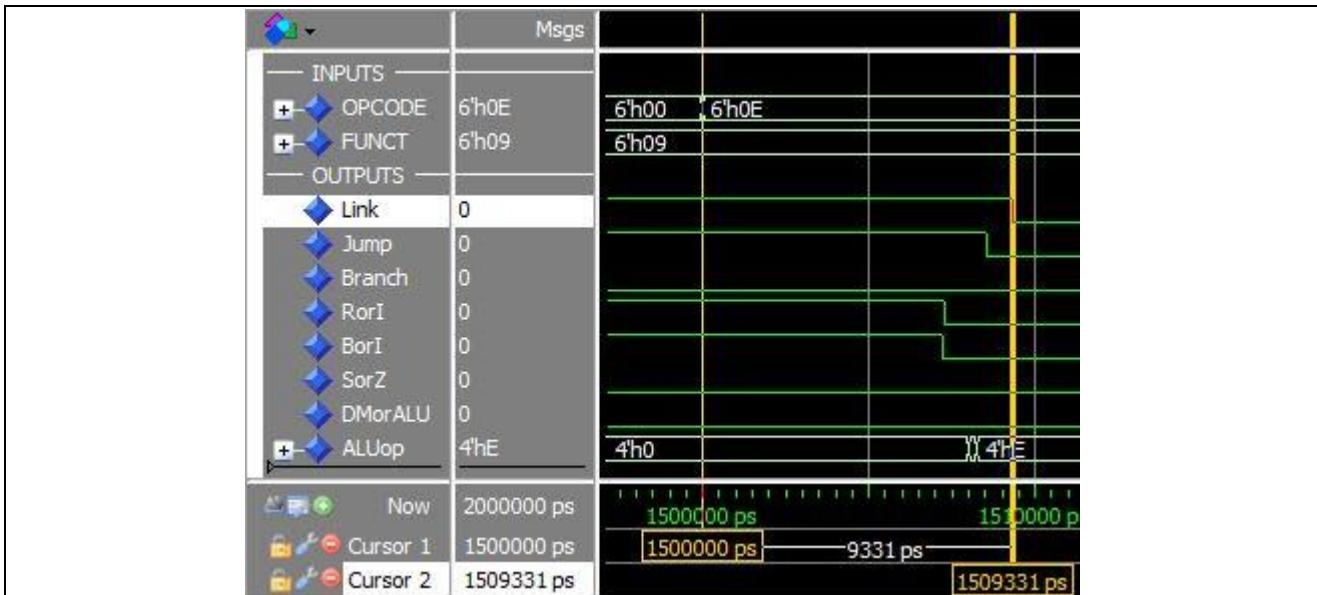


Figure 20 Η καθυστέρηση διάδοσης των critical path της Συνδυαστικής Μονάδας Ελέγχου

### ii. Ακολουθιακή Μονάδα Ελέγχου

Για τον έλεγχο ορθής σχεδίασης της Συνδυαστικής Μονάδας Ελέγχου, αναπτύχθηκε κατάλληλο test bench με χρήση ακολουθιακών εντολών ανάθεσης σημάτων, εφαρμόζοντας συγκεκριμένα διανύσματα δοκιμής. Η προσομοίωση του test bench ελέγχει τη συμπεριφορά της Ακολουθιακής Μονάδας και για μη υποστηριζόμενες λειτουργίες (ασφαλής λειτουργία).

```

library ieee;
use ieee.std_logic_1164.ALL;

entity cu_fsm_tb is
end cu_fsm_tb;

architecture behavior of cu_fsm_tb is

-- "opcode" codes definition as constants
constant RTYPE : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant LW    : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SW    : std_logic_vector(5 downto 0) := "101011"; -- 0x2B
constant XORI  : std_logic_vector(5 downto 0) := "001110"; -- 0xE0
constant BNE   : std_logic_vector(5 downto 0) := "000101"; -- 0x05

-- "funct" codes definition as constants
constant SLLR : std_logic_vector(5 downto 0) := "000000"; -- 0x00
constant ADDU : std_logic_vector(5 downto 0) := "100001"; -- 0x21
constant SUBU : std_logic_vector(5 downto 0) := "100011"; -- 0x23
constant SLT  : std_logic_vector(5 downto 0) := "101010"; -- 0x2A
constant JR   : std_logic_vector(5 downto 0) := "001000"; -- 0x08
constant JALR : std_logic_vector(5 downto 0) := "001001"; -- 0x09

-- Clock period definition
constant clk_period : time := 100 ns;

-- Component Declaration for the Unit Under Test (UUT)
component cu_fsm is
port(
  clk    : in std_logic;
  rst_n : in std_logic;

```

```

        opcode_in : in std_logic_vector (5 downto 0);
        funct_in  : in std_logic_vector (5 downto 0);

        wr_pc_out   : out std_logic;
        wr_rf_out   : out std_logic;
        wr_ir_out   : out std_logic;
        wr_mar_out  : out std_logic;
        wr_dmem_out : out std_logic
    );
end component;

--Inputs
signal clk : std_logic := '0';
signal rst_n : std_logic := '0';
signal opcode_in : std_logic_vector(5 downto 0);
signal funct_in : std_logic_vector(5 downto 0);

--Outputs
signal wr_pc_out : std_logic;
signal wr_rf_out : std_logic;
signal wr_ir_out : std_logic;
signal wr_mar_out : std_logic;
signal wr_dmem_out : std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: cu_fsm
    port map (
        clk      => clk,
        rst_n   => rst_n,
        opcode_in => opcode_in,
        funct_in  => funct_in,
        wr_pc_out   => wr_pc_out,
        wr_rf_out   => wr_rf_out,
        wr_ir_out   => wr_ir_out,
        wr_mar_out  => wr_mar_out,
        wr_dmem_out => wr_dmem_out
    );

    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period - clk_period/2;
    end process;

    -- Reset generation
    rst_n  <=  '1', '0' after clk_period/2;

    -- Stimulus process
    stim_proc: process
    begin

        -- insert stimulus here
        wait until wr_ir_out = '1'; -- on clk's rising edge

        -- 01) LW
        wait for clk_period/25;
        opcode_in <= LW;
        funct_in  <= "-----";

        -- 02) SW
    end process;

```

```

    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= SW;
    funct_in <= "-----";

-- 03) XORI
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= XORI;
    funct_in <= "-----";

-- 04) ANDI not supported
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= "001100";
    funct_in <= "-----";

-- 05) BNE
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= BNE;
    funct_in <= "-----";

-- 06) XOR not supported
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= "100110";

-- 07) SLL
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= SLLR;

-- 08) ADDU
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= ADDU;

-- 09) BEQ not supported
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= "000100";
    funct_in <= "-----";

-- 10) SUBU
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= SUBU;

-- 11) SLT
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= SLT;

-- 12) SRLV not supported
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in <= "000110";

-- 13) JR
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;

```

```

        opcode_in <= RTYPE;
        funct_in  <= JR;

-- 14) JALR
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= RTYPE;
    funct_in  <= JALR;

-- 15) XORI
    wait until wr_ir_out = '1'; -- on clk's rising edge
    wait for clk_period/25;
    opcode_in <= XORI;
    funct_in  <= "-----";

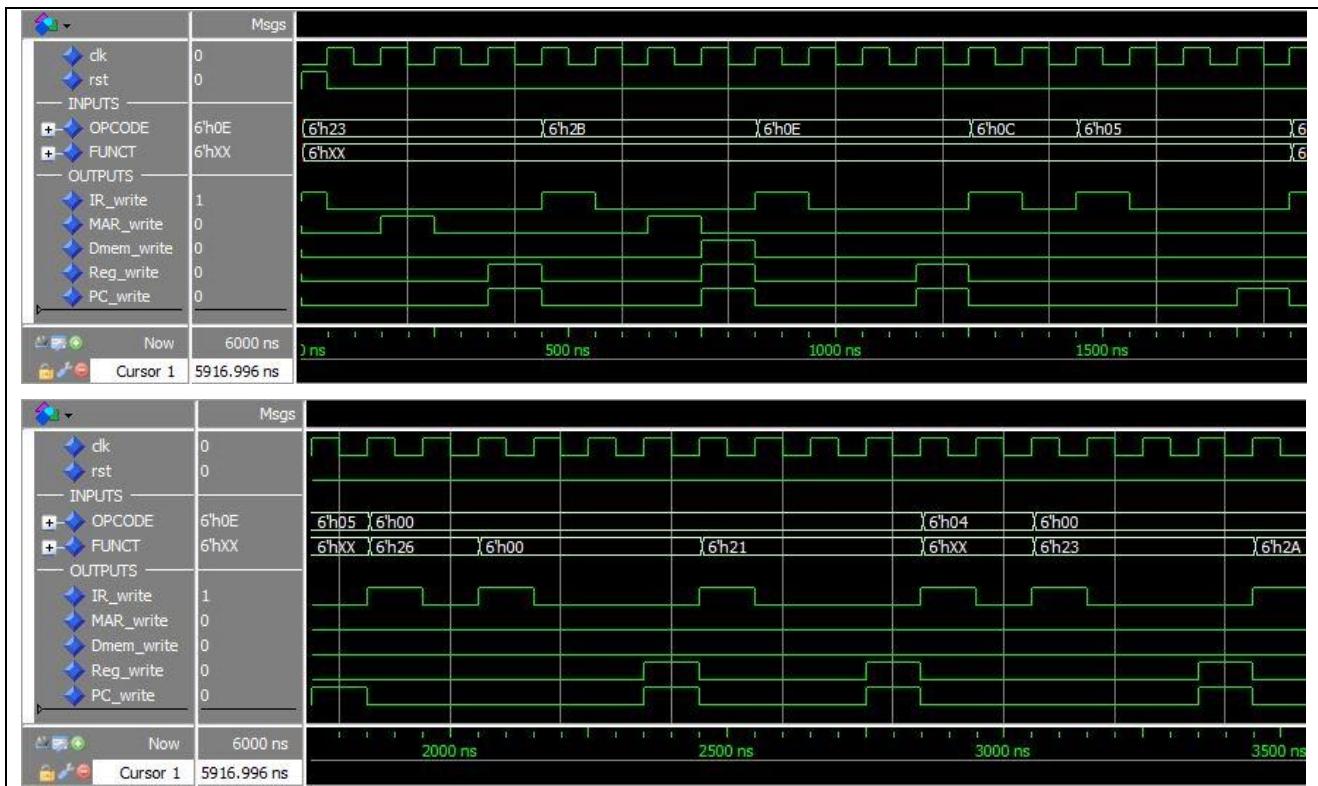
    wait;
end process;

END;

```

**Listing 13 To VHDL test bench για τον έλεγχο λειτουργίας της Ακολουθιακής Μονάδας Ελέγχου**

Όπως φαίνεται και στις παρακάτω κυματομορφές, τα αποτελέσματα των Behavioral και Post-Route Simulation συμφωνούν και επαληθεύουν την ορθή λειτουργία της Συνδυαστικής Μονάδας Ελέγχου.



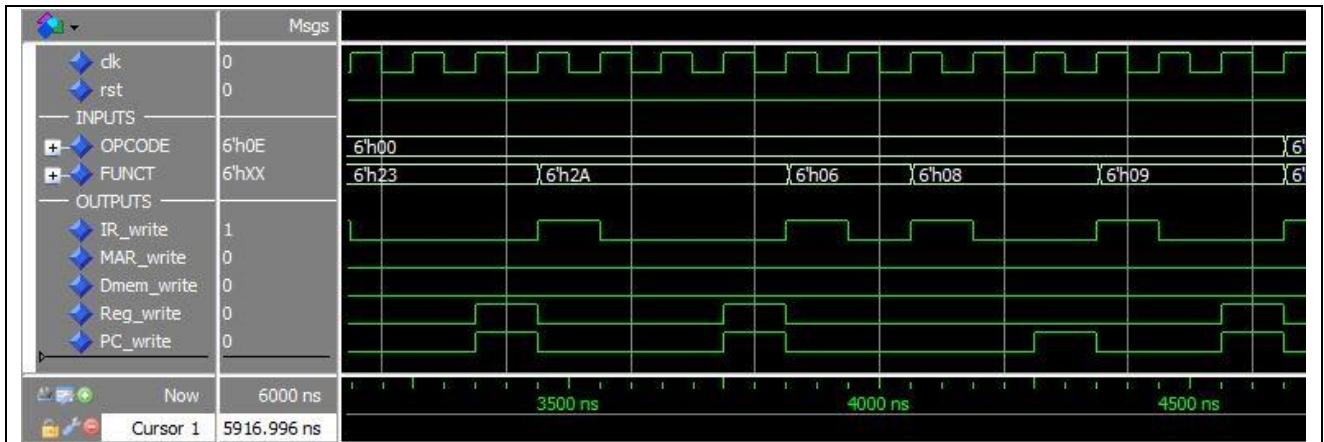


Figure 21 Τα αποτελέσματα του Behavioral simulation της Ακολουθιακής Μονάδας Ελέγχου



Figure 22 Τα αποτελέσματα του Post-Route simulation της Ακολουθιακής Μονάδας Ελέγχου

Από τη μελέτη του Post Place and Route Stating Timing Report που παράγθηκε κατά την υλοποίηση, προκύπτουν και οι παράμετροι χρονισμού της Ακολουθιακής Μονάδας Ελέγχου. Σύμφωνα με το report, οι μέγιστοι χρόνοι setup time και hold time, καθορίζονται από την είσοδο funct\_in(2), η οποία απαιτεί το σήμα να παραμείνει σταθερό (stable) για 5.697 ns (setup time), και μέχρι 2.123 ns πριν/μετά(?) την ανερχόμενη ακμή του ρολογιού.

Setup/Hold to clock clk				
Source	Max Setup to clk (edge)	Max Hold to clk (edge)	Internal Clock(s)	Clock Phase
funct_in<0>	4.390 (R)	0.242 (R)	clk_BUFGP	0.000
funct_in<1>	4.141 (R)	-0.783 (R)	clk_BUFGP	0.000
funct_in<2>	5.697 (R)	-2.123 (R)	clk_BUFGP	0.000   Max Set-up Time/Max Hold Time
funct_in<3>	4.320 (R)	-0.926 (R)	clk_BUFGP	0.000
funct_in<4>	4.896 (R)	-1.482 (R)	clk_BUFGP	0.000
funct_in<5>	3.647 (R)	-0.388 (R)	clk_BUFGP	0.000
opcode_in<0>	4.138 (R)	-0.724 (R)	clk_BUFGP	0.000
opcode_in<1>	5.296 (R)	-0.528 (R)	clk_BUFGP	0.000
opcode_in<2>	4.897 (R)	-0.624 (R)	clk_BUFGP	0.000
opcode_in<3>	5.154 (R)	-0.091 (R)	clk_BUFGP	0.000
opcode_in<4>	3.027 (R)	0.037 (R)	clk_BUFGP	0.000
opcode_in<5>	3.368 (R)	-0.321 (R)	clk_BUFGP	0.000

Figure 23 Οι παράμετροι χρονισμού της Ακολουθιακής Μονάδας Ελέγχου

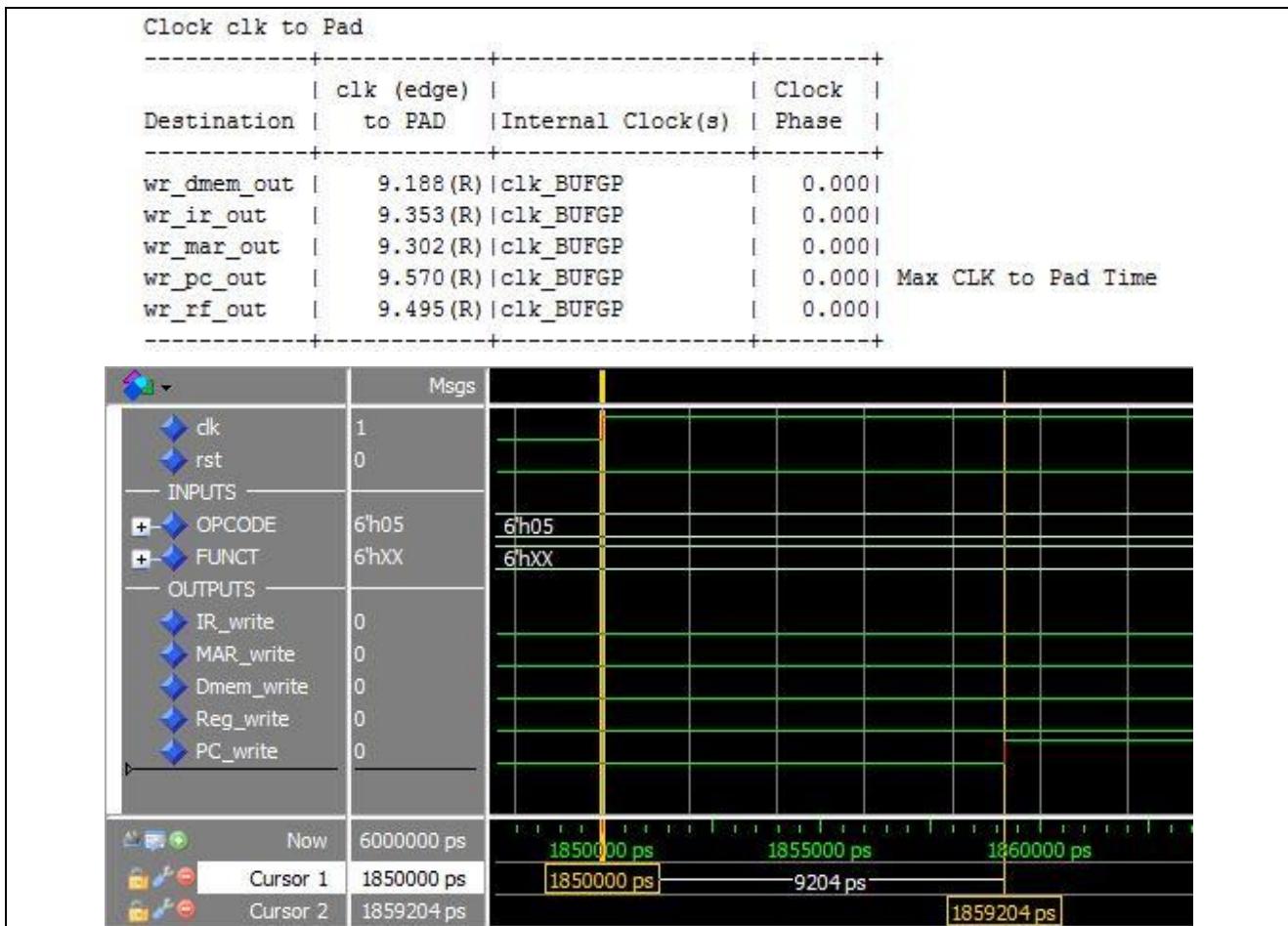


Figure 24 Η μέγιστη καθυστέρηση διάδοσης από την ανερχόμενη ακμή του ρολογιού της Ακολουθιακής Μονάδας Ελέγχου

## 5. ΕΠΑΛΗΘΕΥΣΗ ΟΡΩΗΣ ΣΧΕΔΙΑΣΗΣ ΕΠΕΞΕΡΓΑΣΤΗ

### A. Πρόγραμμα σε γλώσσα Assembly

Το πρόγραμμα σε assembly που χρησιμοποίησα είναι το ακόλουθο:

```
.text      # text section
.globl main # call main by SPIM

main:

    xori    $1, $0, 0x1111  # Reg[01] = x"00001111", Bus_w = x"00001111", PC=00 x"00"
    xori    $2, $0, 0x2222  # Reg[02] = x"00002222", Bus_w = x"00002222", PC=04 x"04"
    addu   $3, $2, $1       # Reg[03] = x"00003333", Bus_w = x"00003333", PC=08 x"08"

    sw     $3, 0($0)        # DM(0) = x"00003333", Bus_w = x"XXXXXXXX", PC=12 x"0C"
    lw     $17, 0($0)       # Reg[17] = x"00003333", Bus_w = x"00003333", PC=16 x"10"

    sll   $21, $17, 5       # Reg[21] = x"00066660", Bus_w = x"00066660", PC=20 x"14"
    sw     $21, 4($0)        # DM(1) = x"00066660", Bus_w = x"00066660", PC=24 x"18"
    lw     $18, 4($0)       # Reg[18] = x"00066660", Bus_w = x"00066660", PC=28 x"1C"

    bne   $14, $0, label1 # condition not satisfied, Bus_w = x"XXXXXXXX", PC=32 x"20"
    subu $25, $18, $1       # Reg[25] = x"0006554F", Bus_w = x"0006554F", PC=36 x"24"

    bne   $25, $0, label1 # satisfied, Bus_w = x"XXXXXXXX", PC=40 x"28"

label2:
    slt   $16, $18, $21    # Reg[16] = x"00000001", Bus_w = x"00000001", PC=44 x"2C"
    slt   $1, $3, $17       # Reg[10] = x"00000000", Bus_w = x"00000000", PC=48 x"30"

label1:
    xori  $27, $0, 0x44    # Reg[27] = x"00000044", Bus_w = x"00000044", PC=52 x"34"
    jalr  $27               # Reg[31] = x"0000003C", PC =68 x"44",
                           # Bus_w = x"0000003C", PC = 56 x"38"
    bne   $21, $18, label2
                           # satisfied on 2nd loop, Bus_w = x"XXXXXXXX", PC=60 x"3C"
    subu $18, $18, $25    # Reg[18] = x"00001111", dec by 0x6554F,
                           # Bus_w = x"0006332D", PC=64 x"40"
    jr    $31               # PC = "3C", loop, Bus_w = x"XXXXXXXX", PC=68 x"44"
```

Τα αποτελέσματα του προγράμματος είναι τα ακόλουθα:

Το πρόγραμμα αρχικά φορτώνει τους καταχωρητές 1 και 2 με τις τιμές x”1111” και x”2222”, αντίστοιχα, με χρήση της εντολής xor. Στη συνέχεια, υπολογίζει το άθροισμά μεταξύ των περιεχομένων των καταχωρητών αυτών και αποθηκεύει το αποτελέσμα στον καταχωρητή 3. Το περιεχόμενο του τελευταίου, αποθηκεύεται στη διεύθυνση x”0000” της μνήμης δεδομένων, και από εκεί αντιγράφεται στον καταχωρητή 17. Ο καταχωρητής 21, λαμβάνει το αποτέλεσμα της αριστερής λογικής ολίσθησης 5 θέσεων, του περιεχομένου του καταχωρητή 17, και ύστερα αποθηκεύεται στη διεύθυνση x”0004” της διεύθυνσης δεδομένων. Το περιεχόμενο της μνήμης στη διεύθυνση αυτή, μεταφέρεται στη συνέχεια στο καταχωρητή 18.

Το πρόγραμμα συνεχίζει εκτελώντας έλεγχο ανισότητας μεταξύ των περιεχομένων του καταχωρητή 14 και 0. Η συνθήκη ικανοποιείται (και οι δύο καταχωρητές περιέχουν την τιμή 0) και η εκτέλεση συνεχίζει κανονικά. Το αποτέλεσμα της αφαίρεσης του περιεχομένου του καταχωρητή 1, από το περιεχόμενο του καταχωρητή 18, αποθηκεύεται στον καταχωρητή 25. Ένας νέος έλεγχος μεταξύ των καταχωρητών 0 και 25, μεταφέρει την εκτέλεση του προγράμματος στη διεύθυνση x”34” της μνήμης εντολών, όπου και πάλι με τη χρήση μίας εντολής xor, φορτώνει τον αριθμό x”44” στον καταχωρητή 27.

Στη συνέχεια, καλείται μία διαδικασία, μέσω της εντολής jalr, η οποία τοποθετεί στον καταχωρητή 31 την αμέσως επόμενη διεύθυνση εντολών x”3C”, και μεταφέρει την εκτέλεση στη διεύθυνση x”44”, που υποδεικνύει ο καταχωρητής 17. Από εκεί, η εντολή jr, επιστρέφει την εκτέλεση στη διεύθυνση που υποδεικνύει ο καταχωρητής 31, όπου και εκτελείται η εντολή διακλάδωσης bne, ελέγχοντας το περιεχόμενο των καταχωρητών 21 και 18. Η σύγκριση δεν ικανοποιείται και η εκτέλεση συνεχίζει στην εκτελώντας μία αφαίρεση, η οποία ελαττώνει το περιεχόμενο του καταχωρητή 18, κατά το περιεχόμενο του καταχωρητή 25.

Η διαδικασία μεταπήδησης επαναλαμβάνεται εκτελώντας την εντολή jr και στη συνέχεια την εντολή διακλάδοσης, η οποία πλέον ικανοποιείται, καθώς το περιεχόμενο των δύο καταχωρητών, πλέον, δεν είναι ίσο. Έτσι, η εκτέλεση μεταφέρεται στη διεύθυνση x”2C”. Εκεί εκτελούνται δύο συνεχόμενες συγκρίσεις του καταχωρητή 18 με τον 21, και του καταχωρητή 3 με τον 17, αντίστοιχα. Τα αποτέλεσματα αποθηκεύονται στους καταχωρητές 16 και 1, αντίστοιχα. Από εκεί και πέρα το πρόγραμμα σχηματίζει έναν ατέρμων βρόγχο κατά τον οποίο, στον καταχωρητή 17 φορτώνεται η διεύθυνση x”44”, το πρόγραμμα καλεί μία διαδικασία, κατά την οποία ο έλεγχος ανισότητας (bne) ικανοποιείται πλέον πάντα, και η εκτέλεση μεταφέρεται και πάλι πίσω στις διαδοχικές συγκρίσεις των καταχωρητών.

Το προκύπτον μετά τη μεταγλώττιση αρχείο DISASM είναι το ακόλουθο:

```
test_program:      file format elf32-bigmips

No symbols in "test_program".
Disassembly of section .text:
0x00000000 38011111      xori    $at,$zero,0x1111
0x00000004 38022222      xori    $v0,$zero,0x2222
0x00000008 00411821      addu   $v1,$v0,$at
0x0000000c ac030000      sw      $v1,0($zero)
0x00000010 8c110000      lw      $s1,0($zero)
0x00000014 0011a940      sll     $s5,$s1,0x5
```

0x00000018	ac150004	sw	\$s5,4(\$zero)
0x0000001c	15c00006	bnez	\$t6,0x00000038
0x00000020	8c120004	lw	\$s2,4(\$zero)
0x00000024	0241c823	subu	\$t9,\$s2,\$at
0x00000028	17200003	bnez	\$t9,0x00000038
0x0000002c	00000000	nop	
0x00000030	0255802a	slt	\$s0,\$s2,\$s5
0x00000034	0071082a	slt	\$at,\$v1,\$s1
0x00000038	381b0044	xori	\$k1,\$zero,0x44
0x0000003c	0360f809	jalr	\$k1
0x00000040	00000000	nop	
0x00000044	16b2ffffa	bne	\$s5,\$s2,0x00000030
0x00000048	00000000	nop	
0x0000004c	03e00008	jr	\$ra
0x00000050	02599023	subu	\$s2,\$s2,\$t9

Οι απαιτούμενες αλλαγές στο αρχείο DISASM είναι οι ακόλουθες:

```
test_program:      file format elf32-bigmips

No symbols in "test_program".
Disassembly of section .text:
0x00000000 38011111      xori   $at,$zero,0x1111
0x00000004 38022222      xori   $v0,$zero,0x2222
0x00000008 00411821      addu   $v1,$v0,$at
0x0000000c ac030000      sw     $v1,0($zero)
0x00000010 8c110000      lw     $s1,0($zero)
0x00000014 0011a940      sll    $s5,$s1,0x5
0x00000018 ac150004      sw     $s5,4($zero)
0x0000001c 8c120004      lw     $s2,4($zero)
0x00000020 15c00004      bnez  $t6,0x00000034
0x00000024 0241c823      subu  $t9,$s2,$at
0x00000028 17200002      bnez  $t9,0x00000034
0x0000002c 0255802a      slt    $s0,$s2,$s5
0x00000030 0071082a      slt    $at,$v1,$s1
0x00000034 381b0044      xori  $k1,$zero,0x44
0x00000038 0360f809      jalr  $k1
0x0000003c 16b2ffffb     bne   $s5,$s2,0x0000002c
0x00000040 02599023      subu  $s2,$s2,$t9
0x00000044 03e00008      jr    $ra
```

Ο κώδικας VHDL της μνήμης εντολών είναι ο ακόλουθος:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity IMEM_rom64x32 is
    Port ( ADDR : in std_logic_vector(5 downto 0);
            DOUT : out std_logic_vector(31 downto 0));
end IMEM_rom64x32;

architecture Behavioral of IMEM_rom64x32 is

subtype Imem_word is bit_vector(31 downto 0);
type Imem_array is array (0 to 63)of Imem_word;

constant Imem : Imem_array := (
X"38011111", X"38022222", X"00411821", X"ac030000", X"8c110000", X"0011a940",
X"ac150004", X"8c120004", X"15c00004", X"0241c823", X"17200002", X"0255802a",
X"0071082a", X"381b0044", X"0360f809", X"16b2ffffb", X"02599023", X"03e00008",
X"00000000", X"00000000", X"00000000", X"00000000", X"00000000", X"00000000",
X"00000000", X"00000000", X"00000000", X"00000000", X"00000000", X"00000000");

```

**Begin**

```

DOUT <= To_X01(Imem(conv_integer(ADDR)));

```

**End Behavioral;**

Ο πίνακας ελέγχου είναι ο ακόλουθος:

a.o.	εντολή	clk(ns)	PC	IR	ALUout	Zero	MAR	MDRin	W	Σχόλια
1	xori \$1, \$0, 0x1111	0	x"00"	x"38011111"						
		100								
		200			x"00001111"	0				
		300					x"00000000"	x"00000000"	0x"00001111"	
2	xori \$2, \$0, 0x2222	400	x"04"	x"38022222"						
		500								
		600			x"00002222"	0				
		700					x"00000000"	x"00000000"	x"00002222"	
3	addu \$3, \$2, \$1	800	x"08"	x"00411821"						
		900								
		1000			x"00003333"	0				
		1100					x"00000000"	x"00001111"	x"00003333"	
4	sw \$3, 0(\$0)	1200	x"0C"	x"ac030000"						
		1300								
		1400			x"00000000"	1				
		1500					x"00000000"	x"00003333"	x"00000000"	
5	lw \$17, 0(\$0)	1600	x"10"	x"8c110000"						
		1700								
		1800			x"00000000"					
		1900					x"00000000"	x"00000000"		
		2000							x"00003333"	

a.a.	εντολή	clk(ns)	PC	IR	ALUout	Zero	MAR	MDRin	W	Σχόλια
6	sll \$21, \$17, 5	2100	x"14"	x"0011a940"						
		2200								
		2300			x"00066660"	0				
		2400					x"00000000"	x"00003333"	x"00066660"	
7	sw \$21, 4(\$0)	2500	x"18"	x"ac150004"						
		2600								
		2700			0x"00000004"	0				
		2800					x"00000004"	x"00066660"	x"00000004"	
8	lw \$18, 4(\$0)	2900	x"1C"	x"8c120004"						
		3000								
		3100								
		3200					x"00000004"	x"00000000"		
		3300							x"00066660"	
9	bne \$14, \$0, label1	3400	x"20"	x"15c00004"						
		3500								
		3600			0x"00000000"	1				
		3700					x"00000004"	x"00000000"	x"00000000"	
10	subu \$25, \$18, \$1	3800	x"24"	x"0241c823"						
		3900								
		4000			0x"0006554F"	0				
		4100					x"00000004"	x"00001111"	x"0006554F"	

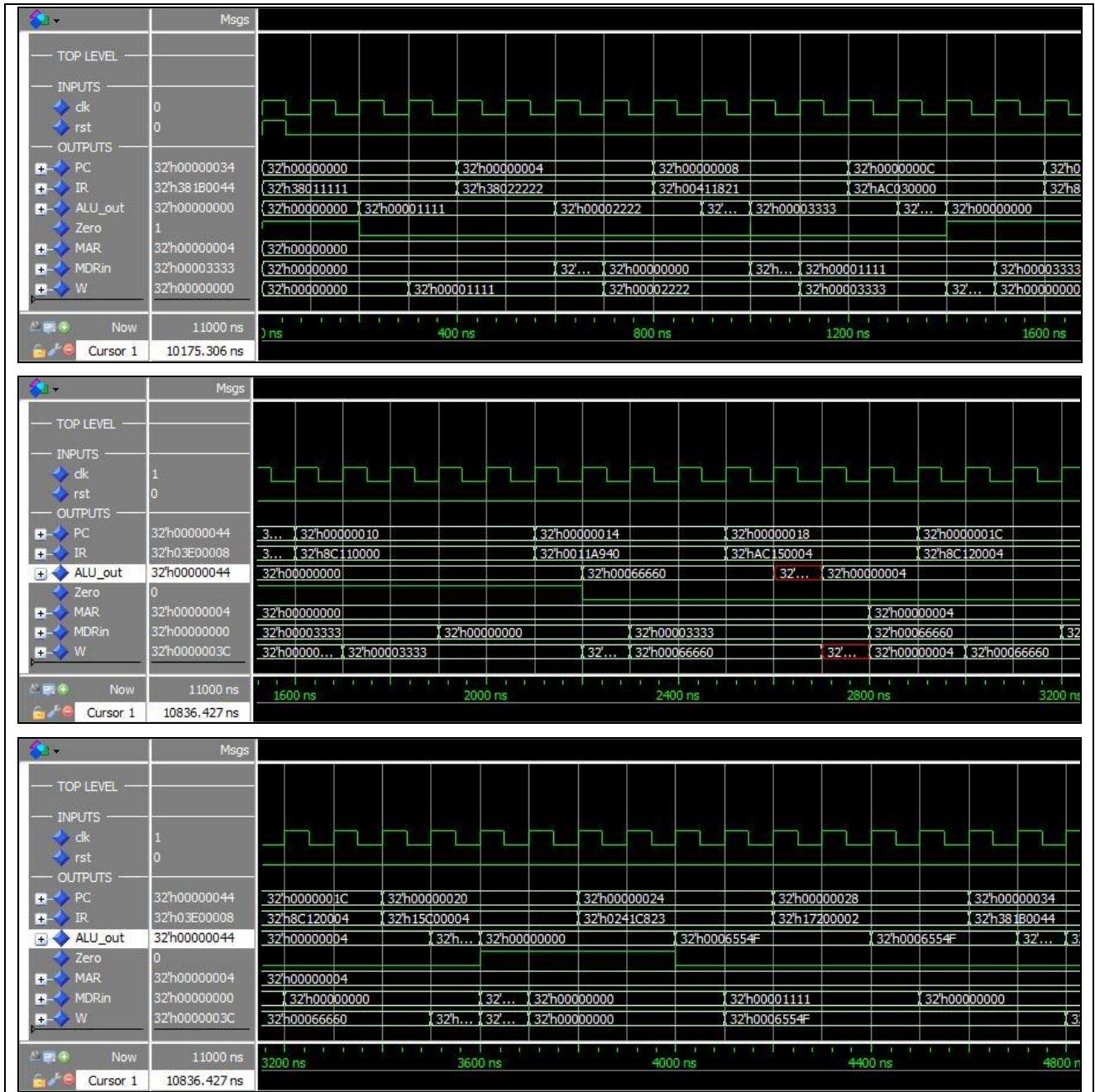
a.a.	εντολή		clk(ns)	PC	IR	ALUout	Zero	MAR	MDRin	W	Σχόλια
11	bne	\$25, \$0, label1	4200	x"28"	x"17200002"						
			4300								
			4400			0x"0006554F"	0				
			4500					x"00000004"	x"00000000"	x"0006554F"	
12	xori	\$27, \$0, 0x44	4600	x"34"	x"381B0044"						
			4700								
			4800			0x"00000044"	0				
			4900					x"00000004"	x"00000000"	x"00000044"	
13	jalr	\$27	5000	x"38"	x"0360F809"						
			5100								
			5200								
14	jr	\$31	5300	x"44"	x"03E00008"						
			5400								
			5500								
15	bne	\$21, \$18, label2	5600	x"3C"	x"16B2FFFB"						
			5700								
			5800			0x"00000000"	1				
			5900					x"00000004"	x"00066660"	x"00000000"	
16	subu	\$18, \$18, \$25	6000	x"40"	x"02599023"						
			6100								
			6200			0x"00001111"	0				
			6300					x"00000004"	x"0006554F"	x"00001111"	





## Αποτελέσματα προσομοιώσεων στο επίπεδο του επεξεργαστή

### i . Behavioral Simulation



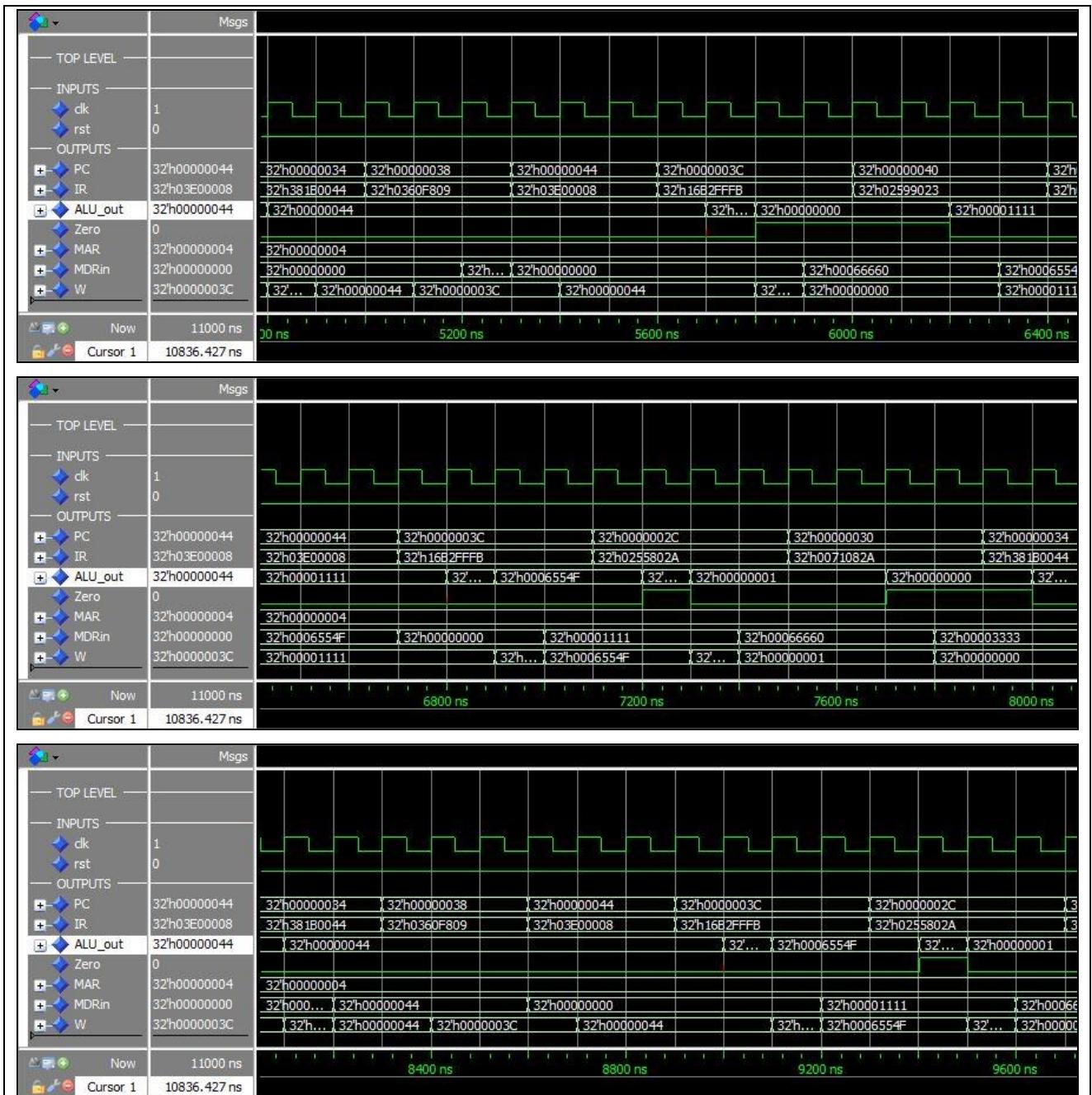
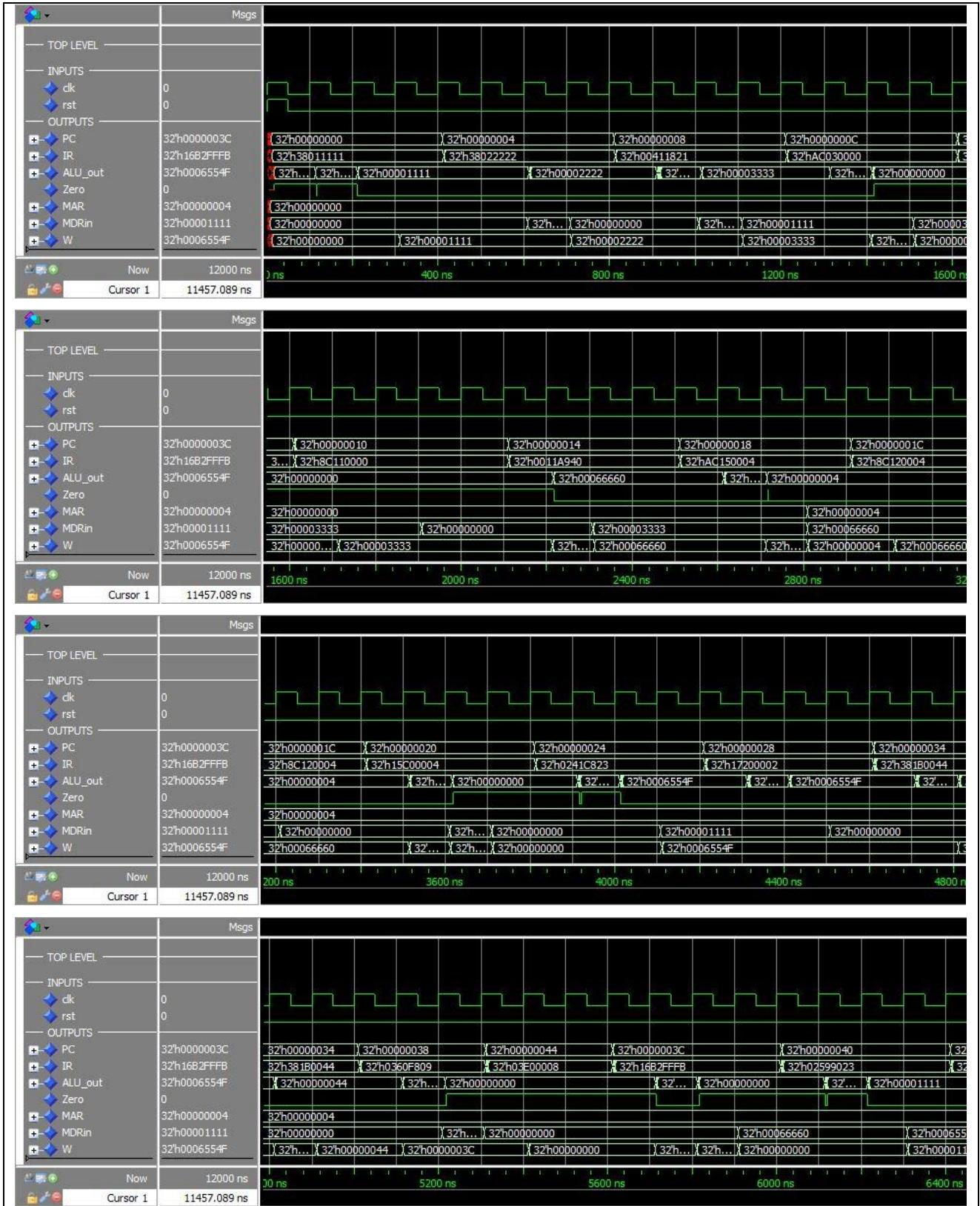


Figure 25 Τα αποτελέσματα του Behavioral simulation του επεξεργαστή

## ii . Post-Route Simulation



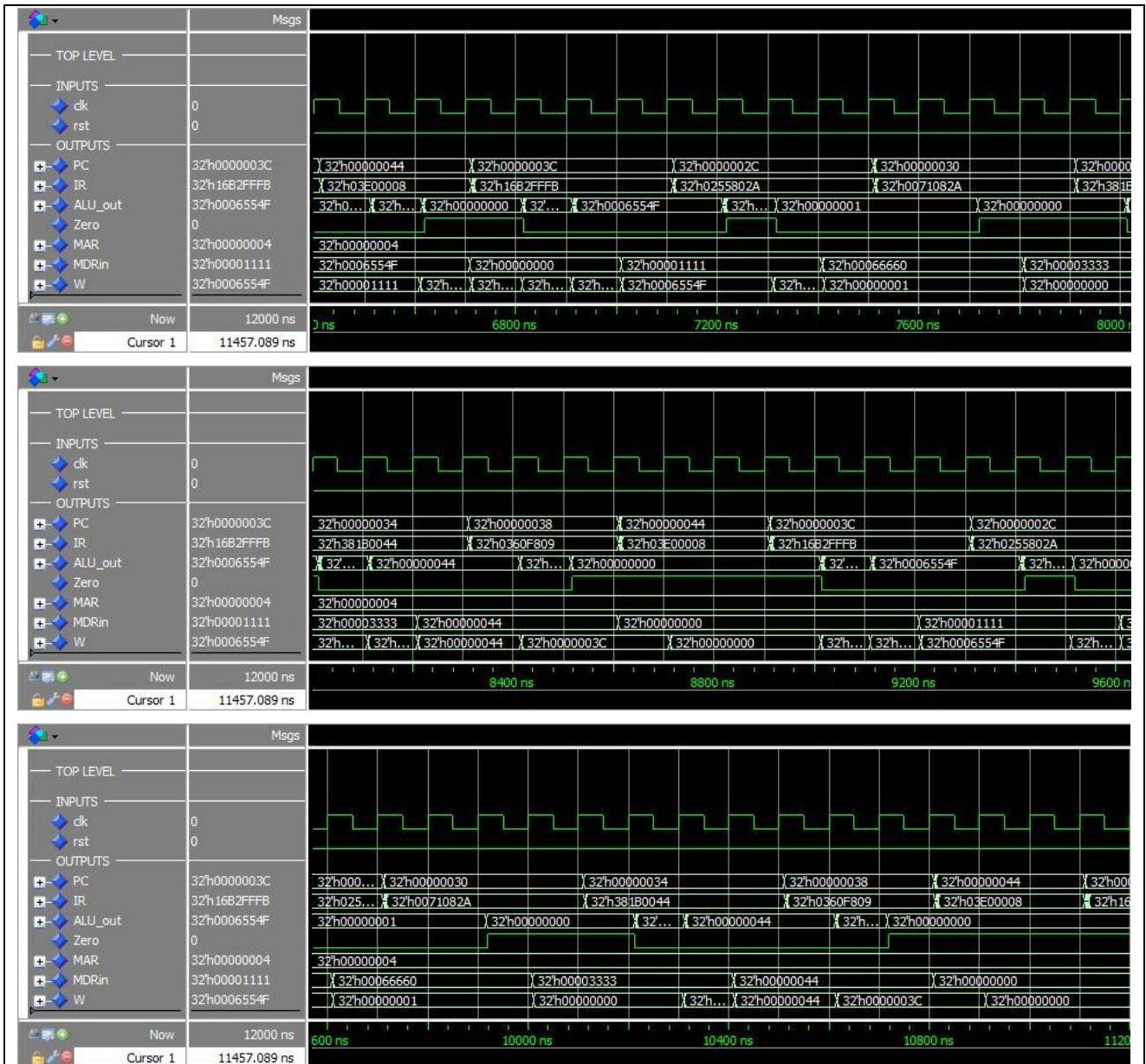


Figure 26 Τα αποτελέσματα του Post-Route simulation του επεξεργαστή

Από το Static Timing Report που προέκυψε μετά την υλοποίηση του επεξεργαστή, η μέγιστη καθυστέρηση διάδοσης, από την ανερχόμενη ακμή του ρολογιού, εντοπίζεται στην έξοδο ALUout(30) και ισούται με 32.740 ns. Η αντίστοιχη καθυστέρηση που εμφανίζεται στο Post-Route suimulation του επεξεργαστή είναι 18.168 ns, τιμή πολύ κοντά στην αναμενόμενη.

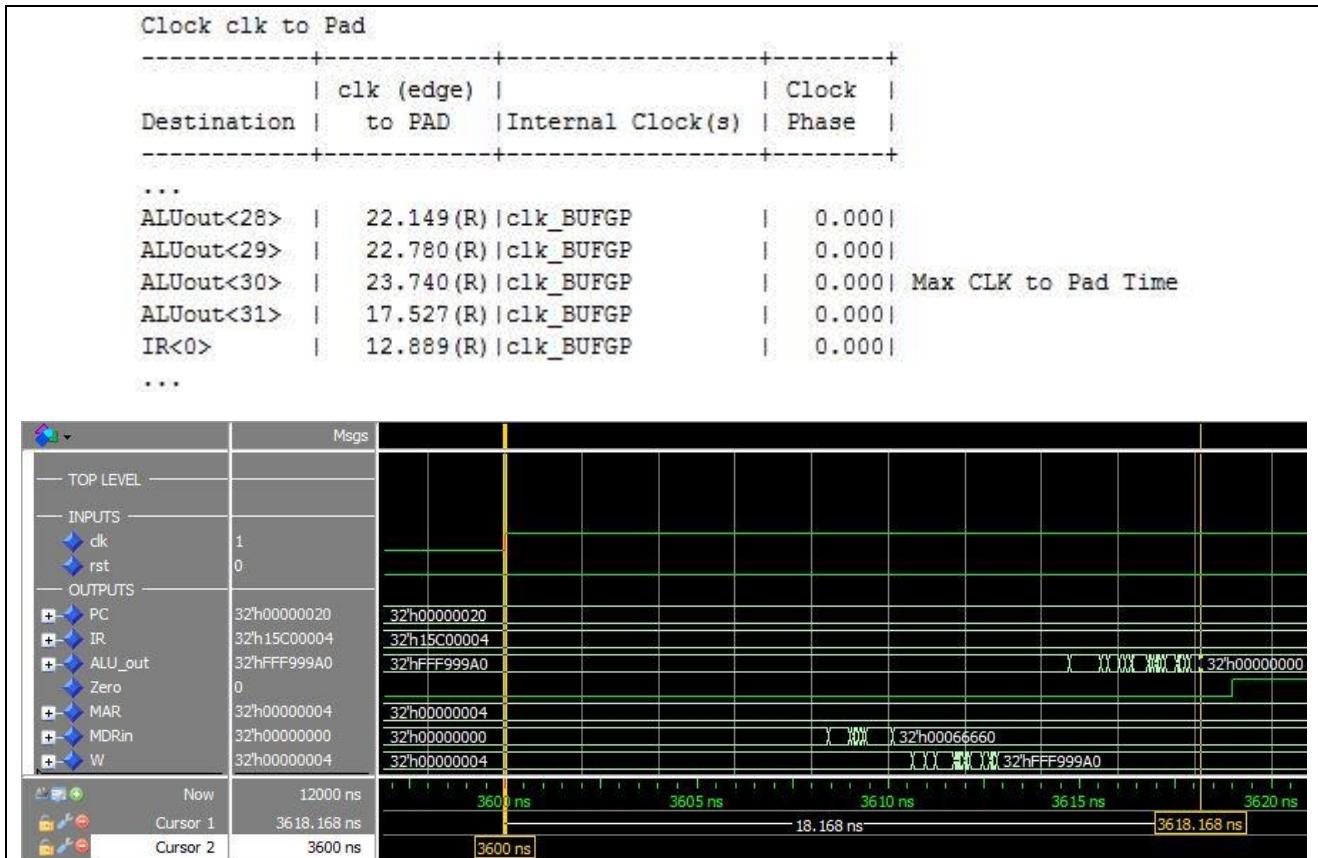


Figure 27 Η μέγιστη καθυστέρηση διάδοσης από την ανερχόμενη ακμή του ρολογιού του επεξεργαστή

Ποια είναι η συχνότητα λειτουργίας του επεξεργαστή;

Με τη χρήση “Timing Constraints”, η τελική συχνότητα του επεξεργαστή έφτασε τα 74,294 MHz, όπως φαίνεται και στην παρακάτω εικόνα.

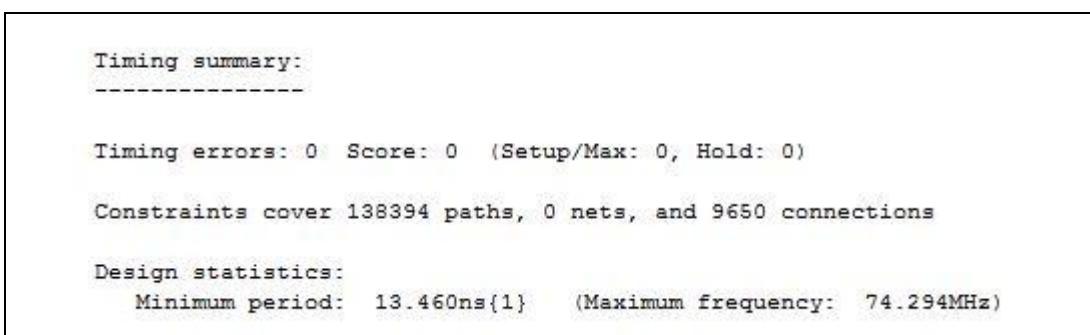


Figure 28 Η τελική συχνότητα λειτουργίας του επεξεργαστή

## 6. ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΣΥΝΘΕΣΗΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗΣ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

### A. Design Summary

mips_r2000 Project Status (03/22/2016 - 01:47:00)				
Project File:	MIPS_R2000.xise	Parser Errors:		
Module Name:	mips_r2000	Implementation State:	Placed and Routed	
Target Device:	xc3s1000-5fg320	• Errors:	No Errors	
Product Version:	ISE 14.7	• Warnings:	25 Warnings (25 new, 0 filtered)	
Design Goal:	Balanced	• Routing Results:	All Signals Completely Routed	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	All Constraints Met	
Environment:	System Settings	• Final Timing Score:	0 ( <a href="#">Timing Report</a> )	

Device Utilization Summary					[1]
Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Flip Flops	1,316	15,360	8%		
Number of 4 input LUTs	1,898	15,360	12%		
Number of occupied Slices	1,552	7,680	20%		
Number of Slices containing only related logic	1,552	1,552	100%		
Number of Slices containing unrelated logic	0	1,552	0%		
Total Number of 4 input LUTs	1,930	15,360	12%		
Number used as logic	1,898				
Number used as a route-thru	32				
Number of bonded IOBs	195	221	88%		
Number of RAMB16s	1	24	4%		
Number of BUFGMUXs	1	8	12%		
Number of RPM macros	6				
Average Fanout of Non-Clock Nets	4.52				

Figure 29 To Design Summary του επεξεργαστή

## B. Post, Place and Route Static Timing Report

```
-----  
-----  
Release 14.7 Trace (nt)  
Copyright (c) 1995-2013 Xilinx, Inc. All rights reserved.  
  
C:\Xilinx\14.7\ISE_DS\ISE\bin\nt\unwrapped\trce.exe -filter  
  
C:/Users/George/Desktop/architecture_project/MIPS_R2000/iseconfig/filter.filter  
-intstyle ise -v 3 -s 5 -n 3 -fastpaths -xml mips_r2000.twx  
mips_r2000.ncd -o  
mips_r2000.twr mips_r2000.pcf  
  
Design file: mips_r2000.ncd  
Physical constraint file: mips_r2000.pcf  
Device, package, speed: xc3s1000,fg456,-5 (PRODUCTION 1.39 2013-  
10-13)  
Report level: verbose report  
  
Environment Variable Effect  
----- -----  
NONE No environment variables were set  
-----  
-----  
  
INFO:Timing:2698 - No timing constraints found, doing default enumeration.  
  
INFO:Timing:3412 - To improve timing, see the Timing Closure User Guide (UG612).  
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths  
option. All paths that are not constrained will be reported in the  
unconstrained paths section(s) of the report.  
INFO:Timing:3339 - The clock-to-out numbers in this timing report  
are based on  
a 50 Ohm transmission line loading model. For the details of  
this model,
```

and for more information on accounting for different loading conditions,

please see the device datasheet.

INFO:Timing:3390 - This architecture does not support a default System Jitter

value, please add SYSTEM\_JITTER constraint to the UCF to modify the Clock

Uncertainty calculation.

INFO:Timing:3389 - This architecture does not support 'Discrete Jitter' and

'Phase Error' calculations, these terms will be zero in the Clock

Uncertainty calculation. Please make appropriate modification to

SYSTEM\_JITTER to account for the unsupported Discrete Jitter and Phase

Error.

Data Sheet report:

-----  
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk

Source	Max Setup to	Max Hold to	Internal Clock(s)	Clock	Phase
rst	clk (edge)	-1.713 (R)	clk_BUFGP		0.000

Clock clk to Pad

Destination	clk (edge)	Internal Clock(s)	Clock	Phase
ALUout<0>	19.386 (R)	clk_BUFGP		0.000
ALUout<1>	16.808 (R)	clk_BUFGP		0.000
ALUout<2>	18.153 (R)	clk_BUFGP		0.000

ALUout<3>		18.430 (R)  clk_BUFGP		0.000
ALUout<4>		16.561 (R)  clk_BUFGP		0.000
ALUout<5>		17.376 (R)  clk_BUFGP		0.000
ALUout<6>		18.295 (R)  clk_BUFGP		0.000
ALUout<7>		18.685 (R)  clk_BUFGP		0.000
ALUout<8>		17.314 (R)  clk_BUFGP		0.000
ALUout<9>		18.759 (R)  clk_BUFGP		0.000
ALUout<10>		21.252 (R)  clk_BUFGP		0.000
ALUout<11>		16.854 (R)  clk_BUFGP		0.000
ALUout<12>		19.861 (R)  clk_BUFGP		0.000
ALUout<13>		18.867 (R)  clk_BUFGP		0.000
ALUout<14>		17.159 (R)  clk_BUFGP		0.000
ALUout<15>		18.793 (R)  clk_BUFGP		0.000
ALUout<16>		18.131 (R)  clk_BUFGP		0.000
ALUout<17>		18.537 (R)  clk_BUFGP		0.000
ALUout<18>		17.546 (R)  clk_BUFGP		0.000
ALUout<19>		18.736 (R)  clk_BUFGP		0.000
ALUout<20>		19.486 (R)  clk_BUFGP		0.000
ALUout<21>		20.017 (R)  clk_BUFGP		0.000
ALUout<22>		17.901 (R)  clk_BUFGP		0.000
ALUout<23>		19.378 (R)  clk_BUFGP		0.000
ALUout<24>		18.163 (R)  clk_BUFGP		0.000
ALUout<25>		19.538 (R)  clk_BUFGP		0.000
ALUout<26>		21.586 (R)  clk_BUFGP		0.000
ALUout<27>		19.097 (R)  clk_BUFGP		0.000
ALUout<28>		22.149 (R)  clk_BUFGP		0.000
ALUout<29>		22.780 (R)  clk_BUFGP		0.000
ALUout<30>		23.740 (R)  clk_BUFGP		0.000
ALUout<31>		17.527 (R)  clk_BUFGP		0.000
IR<0>		12.889 (R)  clk_BUFGP		0.000
IR<1>		14.993 (R)  clk_BUFGP		0.000
IR<2>		12.087 (R)  clk_BUFGP		0.000
IR<3>		13.392 (R)  clk_BUFGP		0.000
IR<4>		12.677 (R)  clk_BUFGP		0.000
IR<5>		14.418 (R)  clk_BUFGP		0.000
IR<6>		12.990 (R)  clk_BUFGP		0.000
IR<7>		13.216 (R)  clk_BUFGP		0.000
IR<8>		14.314 (R)  clk_BUFGP		0.000

IR<9>		13.284 (R)  clk_BUFGP		0.000
IR<10>		13.855 (R)  clk_BUFGP		0.000
IR<11>		12.629 (R)  clk_BUFGP		0.000
IR<12>		13.936 (R)  clk_BUFGP		0.000
IR<13>		14.580 (R)  clk_BUFGP		0.000
IR<14>		12.616 (R)  clk_BUFGP		0.000
IR<15>		14.299 (R)  clk_BUFGP		0.000
IR<16>		13.036 (R)  clk_BUFGP		0.000
IR<17>		11.661 (R)  clk_BUFGP		0.000
IR<18>		12.677 (R)  clk_BUFGP		0.000
IR<19>		11.671 (R)  clk_BUFGP		0.000
IR<20>		14.154 (R)  clk_BUFGP		0.000
IR<21>		15.063 (R)  clk_BUFGP		0.000
IR<22>		12.647 (R)  clk_BUFGP		0.000
IR<23>		14.264 (R)  clk_BUFGP		0.000
IR<24>		14.537 (R)  clk_BUFGP		0.000
IR<25>		12.714 (R)  clk_BUFGP		0.000
IR<26>		11.701 (R)  clk_BUFGP		0.000
IR<27>		12.443 (R)  clk_BUFGP		0.000
IR<28>		12.838 (R)  clk_BUFGP		0.000
IR<29>		12.887 (R)  clk_BUFGP		0.000
IR<31>		12.092 (R)  clk_BUFGP		0.000
MAR<0>		9.446 (R)  clk_BUFGP		0.000
MAR<1>		9.380 (R)  clk_BUFGP		0.000
MAR<2>		10.020 (R)  clk_BUFGP		0.000
MAR<3>		9.210 (R)  clk_BUFGP		0.000
MAR<4>		10.555 (R)  clk_BUFGP		0.000
MAR<5>		10.057 (R)  clk_BUFGP		0.000
MAR<6>		9.718 (R)  clk_BUFGP		0.000
MAR<7>		9.527 (R)  clk_BUFGP		0.000
MAR<8>		9.111 (R)  clk_BUFGP		0.000
MAR<9>		8.636 (R)  clk_BUFGP		0.000
MAR<10>		8.754 (R)  clk_BUFGP		0.000
MAR<11>		8.903 (R)  clk_BUFGP		0.000
MAR<12>		8.340 (R)  clk_BUFGP		0.000
MAR<13>		8.961 (R)  clk_BUFGP		0.000
MAR<14>		9.503 (R)  clk_BUFGP		0.000
MAR<15>		9.322 (R)  clk_BUFGP		0.000

MAR<16>		8.284 (R)  clk_BUFGP		0.000
MAR<17>		9.231 (R)  clk_BUFGP		0.000
MAR<18>		9.441 (R)  clk_BUFGP		0.000
MAR<19>		8.655 (R)  clk_BUFGP		0.000
MAR<20>		8.711 (R)  clk_BUFGP		0.000
MAR<21>		9.156 (R)  clk_BUFGP		0.000
MAR<22>		9.805 (R)  clk_BUFGP		0.000
MAR<23>		8.816 (R)  clk_BUFGP		0.000
MAR<24>		9.191 (R)  clk_BUFGP		0.000
MAR<25>		9.238 (R)  clk_BUFGP		0.000
MAR<26>		8.798 (R)  clk_BUFGP		0.000
MAR<27>		9.196 (R)  clk_BUFGP		0.000
MAR<28>		8.937 (R)  clk_BUFGP		0.000
MAR<29>		8.637 (R)  clk_BUFGP		0.000
MAR<30>		9.001 (R)  clk_BUFGP		0.000
MAR<31>		8.206 (R)  clk_BUFGP		0.000
MDRin<0>		8.862 (R)  clk_BUFGP		0.000
MDRin<1>		9.538 (R)  clk_BUFGP		0.000
MDRin<2>		9.537 (R)  clk_BUFGP		0.000
MDRin<3>		8.716 (R)  clk_BUFGP		0.000
MDRin<4>		9.069 (R)  clk_BUFGP		0.000
MDRin<5>		9.022 (R)  clk_BUFGP		0.000
MDRin<6>		9.181 (R)  clk_BUFGP		0.000
MDRin<7>		9.795 (R)  clk_BUFGP		0.000
MDRin<8>		8.470 (R)  clk_BUFGP		0.000
MDRin<9>		9.631 (R)  clk_BUFGP		0.000
MDRin<10>		9.122 (R)  clk_BUFGP		0.000
MDRin<11>		10.244 (R)  clk_BUFGP		0.000
MDRin<12>		9.066 (R)  clk_BUFGP		0.000
MDRin<13>		10.080 (R)  clk_BUFGP		0.000
MDRin<14>		9.764 (R)  clk_BUFGP		0.000
MDRin<15>		9.466 (R)  clk_BUFGP		0.000
MDRin<16>		9.034 (R)  clk_BUFGP		0.000
MDRin<17>		8.961 (R)  clk_BUFGP		0.000
MDRin<18>		9.805 (R)  clk_BUFGP		0.000
MDRin<19>		9.056 (R)  clk_BUFGP		0.000
MDRin<20>		9.781 (R)  clk_BUFGP		0.000
MDRin<21>		9.399 (R)  clk_BUFGP		0.000

MDRin<22>		9.466 (R)  clk_BUFGP		0.000
MDRin<23>		8.835 (R)  clk_BUFGP		0.000
MDRin<24>		8.353 (R)  clk_BUFGP		0.000
MDRin<25>		8.841 (R)  clk_BUFGP		0.000
MDRin<26>		8.536 (R)  clk_BUFGP		0.000
MDRin<27>		8.641 (R)  clk_BUFGP		0.000
MDRin<28>		8.261 (R)  clk_BUFGP		0.000
MDRin<29>		9.130 (R)  clk_BUFGP		0.000
MDRin<30>		7.466 (R)  clk_BUFGP		0.000
MDRin<31>		9.158 (R)  clk_BUFGP		0.000
PC<0>		10.387 (R)  clk_BUFGP		0.000
PC<1>		9.776 (R)  clk_BUFGP		0.000
PC<2>		10.266 (R)  clk_BUFGP		0.000
PC<3>		10.658 (R)  clk_BUFGP		0.000
PC<4>		12.403 (R)  clk_BUFGP		0.000
PC<5>		11.120 (R)  clk_BUFGP		0.000
PC<6>		12.284 (R)  clk_BUFGP		0.000
PC<7>		12.389 (R)  clk_BUFGP		0.000
PC<8>		9.741 (R)  clk_BUFGP		0.000
PC<9>		10.022 (R)  clk_BUFGP		0.000
PC<10>		9.696 (R)  clk_BUFGP		0.000
PC<11>		10.220 (R)  clk_BUFGP		0.000
PC<12>		11.544 (R)  clk_BUFGP		0.000
PC<13>		9.742 (R)  clk_BUFGP		0.000
PC<14>		9.991 (R)  clk_BUFGP		0.000
PC<15>		9.094 (R)  clk_BUFGP		0.000
PC<16>		10.342 (R)  clk_BUFGP		0.000
PC<17>		9.391 (R)  clk_BUFGP		0.000
PC<18>		9.432 (R)  clk_BUFGP		0.000
PC<19>		9.396 (R)  clk_BUFGP		0.000
PC<20>		9.241 (R)  clk_BUFGP		0.000
PC<21>		8.876 (R)  clk_BUFGP		0.000
PC<22>		9.036 (R)  clk_BUFGP		0.000
PC<23>		10.144 (R)  clk_BUFGP		0.000
PC<24>		9.246 (R)  clk_BUFGP		0.000
PC<25>		8.795 (R)  clk_BUFGP		0.000
PC<26>		9.065 (R)  clk_BUFGP		0.000
PC<27>		9.092 (R)  clk_BUFGP		0.000

PC<28>		8.869 (R)  clk_BUFGP		0.000
PC<29>		9.323 (R)  clk_BUFGP		0.000
PC<30>		10.785 (R)  clk_BUFGP		0.000
PC<31>		8.868 (R)  clk_BUFGP		0.000
W<0>		17.670 (R)  clk_BUFGP		0.000
W<1>		18.885 (R)  clk_BUFGP		0.000
W<2>		16.071 (R)  clk_BUFGP		0.000
W<3>		16.865 (R)  clk_BUFGP		0.000
W<4>		16.374 (R)  clk_BUFGP		0.000
W<5>		18.583 (R)  clk_BUFGP		0.000
W<6>		17.443 (R)  clk_BUFGP		0.000
W<7>		17.001 (R)  clk_BUFGP		0.000
W<8>		16.929 (R)  clk_BUFGP		0.000
W<9>		16.705 (R)  clk_BUFGP		0.000
W<10>		17.573 (R)  clk_BUFGP		0.000
W<11>		17.188 (R)  clk_BUFGP		0.000
W<12>		16.374 (R)  clk_BUFGP		0.000
W<13>		17.077 (R)  clk_BUFGP		0.000
W<14>		16.476 (R)  clk_BUFGP		0.000
W<15>		16.579 (R)  clk_BUFGP		0.000
W<16>		15.784 (R)  clk_BUFGP		0.000
W<17>		15.511 (R)  clk_BUFGP		0.000
W<18>		17.669 (R)  clk_BUFGP		0.000
W<19>		16.975 (R)  clk_BUFGP		0.000
W<20>		14.706 (R)  clk_BUFGP		0.000
W<21>		15.130 (R)  clk_BUFGP		0.000
W<22>		14.887 (R)  clk_BUFGP		0.000
W<23>		17.125 (R)  clk_BUFGP		0.000
W<24>		16.612 (R)  clk_BUFGP		0.000
W<25>		16.392 (R)  clk_BUFGP		0.000
W<26>		17.489 (R)  clk_BUFGP		0.000
W<27>		16.716 (R)  clk_BUFGP		0.000
W<28>		16.929 (R)  clk_BUFGP		0.000
W<29>		17.441 (R)  clk_BUFGP		0.000
W<30>		17.457 (R)  clk_BUFGP		0.000
W<31>		17.408 (R)  clk_BUFGP		0.000
Zero		20.885 (R)  clk_BUFGP		0.000
<hr/>				

```

Clock to Setup on destination clock clk
-----+-----+-----+-----+
      | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
-----+-----+-----+-----+
clk          |    13.185|           |           |
-----+-----+-----+-----+
Analysis completed Tue Mar 22 00:06:13 2016
-----
-----
Trace Settings:
-----
Trace Settings
-----
Peak Memory Usage: 152 MB

```

**Figure 30 Post-Place & Route Static Timing Report**

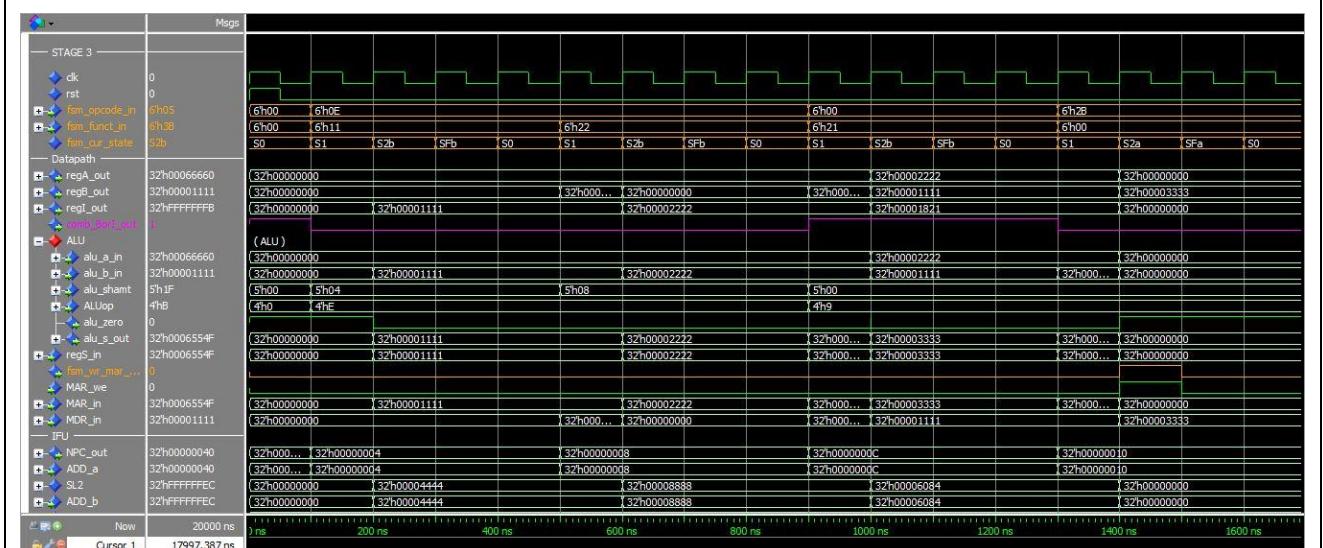
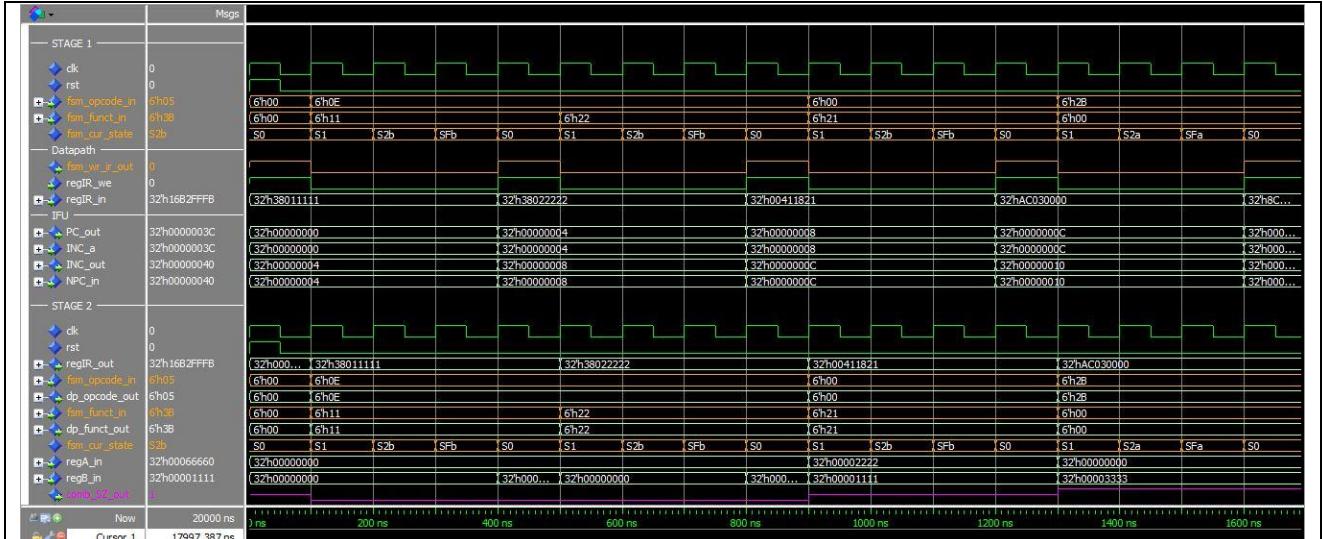
### C. Σχολιασμός των αποτελεσμάτων

Από το Design Summary του project και το Static Timing Report εξάγονται όλα τα στοιχεία ολοκλήρωσης του επεξεργαστή. Για το implementation του επεξεργαστή επιλέχθηκε το chip xc3s1000, της οικογένειας Spartan 3 των FPGA της XILINX. Το utilization φτάνει μόλις το 20% των διαθέσιμων slices του FPGA. Οι πόροι που καταλαμβάνει ο επεξεργαστής αναλύονται σε 1316 flip-flops για την υλοποίηση όλων των register (utilization 8%), 1898 4 input LUTs που χρησιμοποιούνται για τη λογική του κυκλώματος (utilization 12%), ακόμη 32 4 input LUTs για την οδήγηση σημάτων και ένα από τα 24 διαθέσιμα blocks μνήμης RAM (utilization 4%). Επιπλέον, ο επεξεργαστής χρησιμοποιεί 195 IOBs, από τα συνολικά 221 διαθέσιμα (utilization 88%), για την οδήγηση των σημάτων εισόδου και εξόδου. Περισσότερες λεπτομέρειες για τη σχεδίαση και την υλοποίηση στο chip, μπορούν να εξαχθούν από τη μελέτη των Synthesis, Translation, Map και Place and Route Reports.

Όσο αφορά τις παραμέτρους χρονισμού και τις καθυστερήσεις στο επίπεδο του επεξεργαστή, αυτά εξάγονται από το Static Timing Report. Ενδεικτικά αναφέρεται ότι η ελάχιστη περίοδος ρολογιού είναι στα 13.460 ns, με τη μέγιστη συχνότητα λειτουργίας του επεξεργαστή να φτάνει τα 74.294 MHz.

Τέλος, αξίζει να σημειωθεί ότι υπήρξαν κάποιες αλλαγές στα παραδοτέα του πρώτου μέρους. Κατά την ανάπτυξη του δεύτερου μέρους του project, παρατηρήθηκαν αρκετά λάθη στη περιγραφή της δίοδου δεδομέων. Προκειμένου να εντοπιστούν και να διορθωθούν όλα τα λάθη, που οδηγούσαν σε απροσάρμοστη λειτουργία του

επεξεργαστή, στο πιο υψηλό ιεραρχικά επίπεδο, χρειάστηκε να γίνει ένας εξαντλητικός και διεξοδικός έλεγχος όλων των εσωτερικών σημάτων του datapath. Για το λόγο αυτό, το test bench του επεξεργαστή εκτελέστηκε και χρησιμοποιήθηκε για την διεξοδική μελέτη όλων των εσωτερικών σημάτων και όλων των μονοπατιών στο εσωτερικό του datapath. Η χρήση του terst bench ήταν καθοριστική σε αυτό το σημείο, καθώς χωρίς αυτό δε θα ήταν δυνατό να γίνει η διόρθωση του design. Αποσπάσματα των κυματομορφών που ελέγχησαν, παρουσιάζονται στις επόμενες εικόνες.



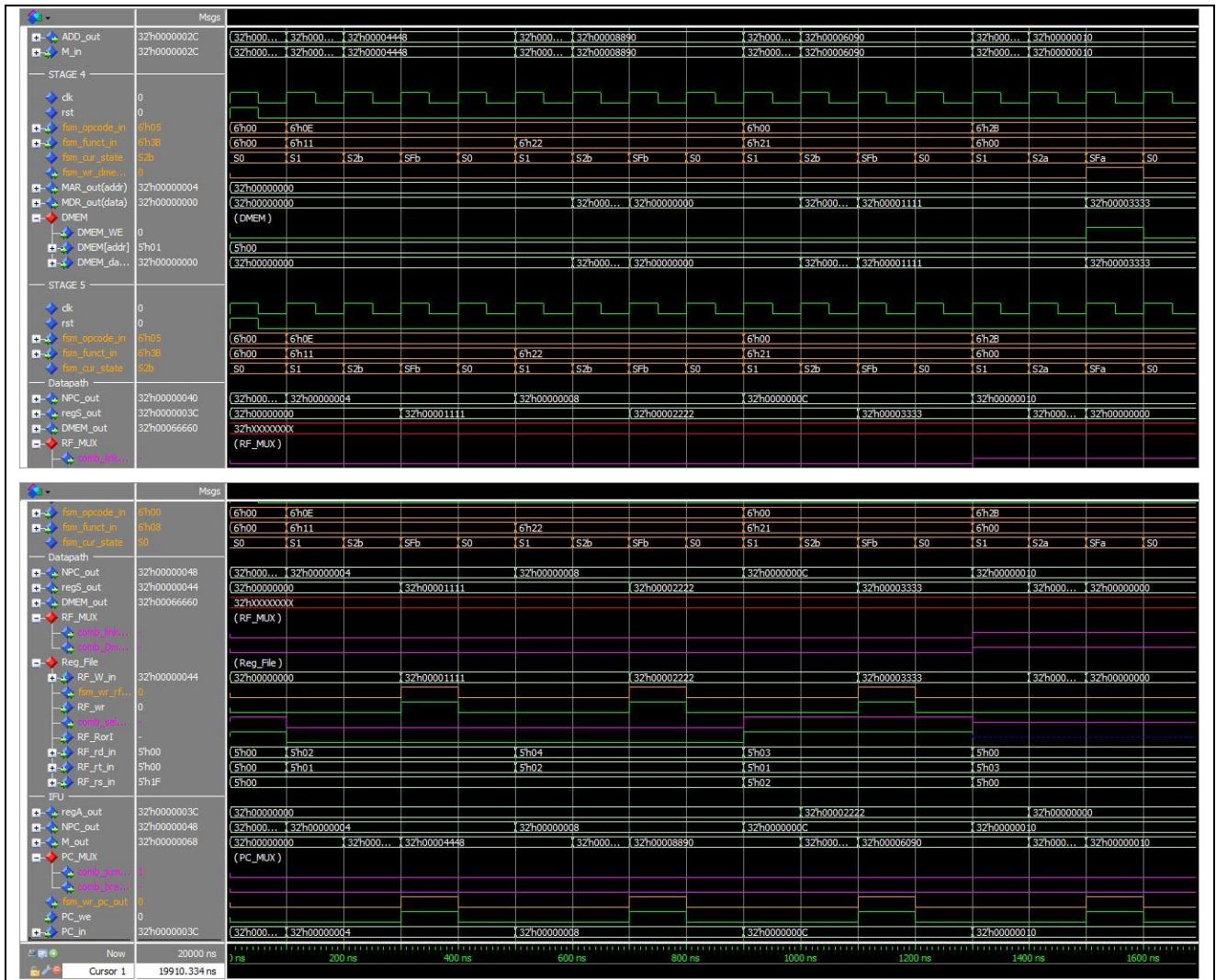


Figure 31 Απόσπασμα του Behavioral simulation που μελετήθηκε κατά το debugging του datapath