



Τμήμα Ηλεκτρολόγων Μηχανικών
& Μηχανικών Υπολογιστών
**ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ**

Σχεδιασμός Ψηφιακών Συστημάτων σε FPGA

Τελική Εργασία (2)

Ζιάγγας Ιωάννης
A.M: 20229

Μικέλ Ντούντι
A.M: 21110

Επιβλέπων Καθηγητής:
Παρασκευάς Κίτσος



ΠΕΡΙΕΧΟΜΕΝΑ

1 – Εισαγωγή.....	2
1.1 – Σκοπός της Εργασίας.....	2
1.2 – Προηγούμενες Εργασίες.....	2
1.3 – Δομή της Αναφοράς.....	3
2 – Τροποποιήσεις του Κώδικα	3
2.1 – control_unit.vhd.....	3
2.2 – rs_cpu.vhd.....	6
2.3 – databus.vhd.....	7
2.4 – alus.vhd	7
2.5 – reg8.vhd	8
2.6 – decode_generic.vhd & counter_3bit.vhd	8
2.7 – cpulib.vhd	8
2.8 – Σύνοψη Αλλαγών	9
2.9 – Λεπτομέρειες Υλοποίησης	9
MOVB Instruction (0x04)	9
ADDB Instruction (0x18)	10
SUBB Instruction (0x19)	10
3 – Κώδικας της Μνήμης RAM.....	10
3.1 – ModelSim RAM	11
3.2 – Quartus RAM	13
4 – Αποτελέσματα.....	13
4.1 – ModelSim Αποτελέσματα	14
4.2 – Quartus Αποτελέσματα.....	16
5 – Επιβεβαίωση	17



1 – Εισαγωγή

1.1 – Σκοπός της Εργασίας

Σε αυτή την εργασία, στόχος μας είναι να επεκτείνουμε την αρχιτεκτονική της απλής KME (Κεντρική Μονάδα Επεξεργασίας) που σχεδιάστηκε στις προηγούμενες εργασίες του μαθήματος, προσθέτοντας νέες δυνατότητες και εντολές. Συγκεκριμένα, η «Εργασία 2» που έχουμε επιλέξει, μας ζητά να προσθέσουμε έναν καταχωρητή **B** για τη χρήση ως δεύτερος τελεστής για τις παρακάτω λογικές πράξεις:

Instruction	Instruction Code (Hex)	Instruction Code (Bin)	Operation
ADDB	18	00011000	$AC \leftarrow AC + B$
SUBB	19	00011001	$AC \leftarrow AC - B$

1.2 – Προηγούμενες Εργασίες

Στην εργασία 4, σχεδιάσαμε μια **hardwired** μονάδα ελέγχου για μια απλή KME. Η μονάδα ελέγχου αυτή χρησιμοποιούσε:

- Ένα γενικό αποκωδικοποιητή (generic decoder) για την αποκωδικοποίηση εντολών,
- Έναν 3-bit απαριθμητή για τη δημιουργία των σημάτων χρονισμού (T0-T7),
- Συνδυαστική λογική (combinational logic) για τη δημιουργία των σημάτων ελέγχου (micro-operations),
- Σύνολο 16 βασικών εντολών (CLAC, INAC, ADD, SUB, AND, OR, XOR, NOT, LDAC, STAC, MOVR, JUMP, JMPZ, JPNZ, NOP, MVAC).

Αυτή η προσέγγιση παρείχε πλήρη έλεγχο της KME χωρίς τη χρήση ROM (Read-Only Memory) για αποθήκευση μικροεντολών.

Στην εργασία 5, ολοκληρώσαμε την υλοποίηση της KME, προσθέτοντας:

- Ένα ολοκληρωμένο datapath με όλους τους απαραίτητους καταχωρητές (PC, AC, DR, AR, TR, IR, R, Z),
- Μια εξωτερική RAM (256x8) για αποθήκευση προγράμματος και δεδομένων,
- Μια ALU (Arithmetic Logic Unit) που υποστηρίζει βασικές πράξεις,
- Ένα κύκλωμα πολυπλέκησης δεδομένων (databus multiplexer),
- Την ολοκληρωμένη hardwired μονάδα ελέγχου από την εργασία 4.



Η ΚΜΕ μπορούσε να εκτελέσει ένα σύνολο απλών προγραμμάτων και παρείχε τη βάση για περαιτέρω επεκτάσεις.

1.3 – Δομή της Αναφοράς

Χρησιμοποιώντας την υπάρχων hardwired KME, αυτή η αναφορά οργανώνεται ως εξής: Στο κεφάλαιο 2, θα αναφέρουμε τις τροποποιήσεις που κάναμε στους κώδικες για να προσθέσουμε τις δύο επιπλέων εντολές που μας ζητήθηκε. Στο κεφάλαιο 3, θα μιλήσουμε για την αλλαγή που κάναμε στην RAM, και τέλος στο κεφάλαιο 4 θα παρουσιάσουμε τα αποτελέσματα των προσομοιώσεων στο ModelSim και Quartus.

2 – Τροποποιήσεις του Κώδικα

Σε αυτό το κεφάλαιο θα περιγράψουμε αναλυτικά τις αλλαγές που πραγματοποιηθήκαν σε κάθε αρχείο VHDL, για την δημιουργία της νέας KME. Κάθε υποενότητα αντιστοιχεί σε ένα διαφορετικό αρχείο και περιγράφει τις αλλαγές από την έκδοση που χρησιμοποιήθηκε για την εργασία 5.

2.1 – control_unit.vhd

Η μονάδα ελέγχου είναι το κύριο κύκλωμα που διαχειρίζεται όλες τις λειτουργίες της KME. Στην καινούρια έκδοση, προσθέσαμε υποστήριξη για τις νέες εντολές **ADDB**, **SUBB**, και **MOV_B**. Ακολουθούν οι αλλαγές του κώδικα:

```
-- Added IMOVB signal for MOVB instruction
signal IMOVB : std_logic;

-- Added MOVB FSM states
signal MOVB1, MOVB2 : std_logic;

-- Added ADDB and SUBB FSM states
signal ADDB1, SUBB1 : std_logic;

-- Added BLOAD signal for Register B
signal BLOAD, BBUS : std_logic;

-- Expanded mOPs vector from 27 to 29 bits
signal mOPs : std_logic_vector(28 downto 0);
```

Code Snippet 1: Δήλωση νέων σημάτων



```
-- Explicit decoders for special opcodes (outside 4-bit range)
```

```
IADD<= '1' when ir = x"18" else '0'; -- ADDB opcode
```

```
ISUBB<= '1' when ir = x"19" else '0'; -- SUBB opcode
```

```
IMOV<= '1' when ir = x"04" else '0'; -- MOVB opcode
```

Code Snippet 2: Επέκταση του Instruction Decoder

Εξήγηση: Οι τρεις εντολές **ADDB (0x18)**, **SUBB (0x19)**, και **MOVB (0x04)** απαιτούσαν ρητή αποκωδικοποίηση διότι δεν μπορούσαν να αναπαρασταθούν με τον απλό 4-bit decoder. Η MOVB είναι απαραίτητη για τη φόρτωση του καταχωρητή **B** με δεδομένα από τη μνήμη.

```
-- Added MOVB FSM states
```

```
MOVB1 <= IMOV< and T3;
```

```
MOVB2 <= IMOV< and T4;
```

```
-- Added ADDB and SUBB FSM states
```

```
ADDB1 <= IADD< and T3;
```

```
SUBB1 <= ISUBB< and T3;
```

Code Snippet 3: FSM State Generation

Εξήγηση: Για κάθε εντολή, δημιουργήσαμε FSM καταστάσεις που ενεργοποιούνται σε συγκεκριμένες χρονικές στιγμές (T3, T4, κ.λπ.). Οι ADDB και SUBB εκτελούνται σε ένα κύκλο (T3), ενώ η MOVB απαιτεί δύο κύκλους (T3 και T4).



```
-- Added MOVB to PCINC (PC increments after MOVB2)

PCINC <= FETCH2 OR JMPZN2 OR JPNZN2 OR LDAC2 OR LDAC4 OR STAC2 OR STAC4
OR MOVR2 OR MOVB2;

-- Added MOVB1 to DRLOAD (Load DR during MOVB)

DRLOAD <= FETCH2 OR LDAC2 OR ADD1 OR SUB1 OR AND1 OR OR1 OR XOR1 OR
ADDB1 OR SUBB1 OR MOVB1;

-- Added MOVB to MEMBUS

MEMBUS <= FETCH2 OR LDAC2 OR LDAC3 OR STAC2 OR MOVR2 OR MOVB1 OR
MOVB2 OR ADDB1 OR SUBB1;

-- Added ADDB1 and SUBB1 to various signals

ACLOAD <= LDAC3 OR ADD1 OR SUB1 OR AND1 OR OR1 OR XOR1 OR NOT1 OR MVAC1
OR ADDB1 OR SUBB1;

ZLOAD <= ADD1 OR SUB1 OR AND1 OR OR1 OR XOR1 OR NOT1 OR INAC1 OR CLAC1
OR ADDB1 OR SUBB1;

-- Load B at T4 of MOVB (after operand is available)

BLOAD <= MOVB2;
```

Code Snippet 4: Micro-operation Logic Updates

Εξήγηση: Ενημερώσαμε τα σήματα ελέγχου ώστε να περιλαμβάνουν τις νέες εντολές. Σημαντικά σημεία:

- **BLOAD** ενεργοποιείται μόνο κατά MOVB2, εξασφαλίζοντας ότι ο καταχωρητής **B** φορτώνεται με την σωστή χρονική στιγμή
- **ADDB1** και **SUBB1** προστίθενται στις λογικές εξισώσεις που χρειάζονται τις συγκεκριμένες πράξεις

```
-- Expanded from 27 to 29 bits

mOPs(27) <= BLOAD; -- Register B Load signal

mOPs(28) <= BBUS; -- Register B Bus signal
```

Code Snippet 5: Expanded mOPs Output Vector

Εξήγηση: Προσθέσαμε δύο νέα σήματα στο mOPs vector για τον έλεγχο του καταχωρητή **B**.



2.2 – rs_cpu.vhd

Αυτό είναι το top-level κύκλωμα που συνδέει όλα τα υποκυκλώματα της ΚΜΕ.

```
-- NEW: Register B output for testbench monitoring  
  
signal BRdata : std_logic_vector(7 downto 0);
```

Code Snippet 6: Νέα Port Declaration

Εξήγηση: Προσθέσαμε το **BRdata** port για να μπορούμε να παρακολουθούμε την τιμή του καταχωρητή B κατά την προσομοίωση.

```
-- NEW: Register B instantiation
```

```
b_register : reg8  
  
port map(  
  
    clk => clock,  
  
    rst => reset,  
  
    ld => BLOAD,  
  
    inc => '0',  
  
    clr => '0',  
  
    d => cpu_data_bus,  
  
    q => BRdata  
  
);
```

Code Snippet 7: Νέο Component Instantiation

Εξήγηση: Δημιουργήσαμε ένα νέο component του **reg8** module για τον καταχωρητή B. Χρησιμοποιούμε το **BLOAD** σήμα από τη μονάδα ελέγχου για να ελέγχουμε τη φόρτωση του καταχωρητή.



```
-- Updated port map to include B register option

databus_mux : databus
port map(
    pc  => pc_bus_value,
    dr  => dr_bus_value,
    tr  => tr_bus_value,
    r   => r_bus_value,
    ac  => ac_bus_value,
    mem => mem_value,
    PCBUS => PCBUS,
    DRBUS => DRBUS,
    TRBUS => TRBUS,
    RBUS => RBUS,
    ACBUS => ACBUS,
    MEMBUS => MEMBUS,
    BUS  => cpu_data_bus
);
```

Code Snippet 8: Databus Multiplexer Update

Εξήγηση: Ο **databus** multiplexer συνδέει τον καταχωρητή B στον εσωτερικό δίαυλο δεδομένων, επιτρέποντας τη φόρτωση του με δεδομένα από τη μνήμη.

2.3 – databus.vhd

Ο δίαυλος δεδομένων (databus) είναι ένας πολυπλέκτης που επιλέγει ποια δεδομένα θα διαδοθούν μέσω του εσωτερικού δίαυλου. Ο **databus** δεν απαιτούσε αλλαγές διότι ο καταχωρητής **B** συνδέεται απευθείας στον εσωτερικό δίαυλο μέσω του multiplexer. Το σήμα **MEMBUS** ήδη επιτρέπει τη σύνδεση των δεδομένων από τη μνήμη στο δίαυλο.

2.4 – alus.vhd

Η **ALU** (Arithmetic Logic Unit) μονάδα διαχειρίζεται τις αριθμητικές και λογικές πράξεις. Η υπάρχουσα **ALU** είχε ήδη τη δυνατότητα να εκτελέσει προσθέσεις και αφαιρέσεις. Οι νέες εντολές **ADDB** και **SUBB** χρησιμοποιούν τα ίδια σήματα ελέγχου (PLUS και MINUS) με τις **ADD** και **SUB** εντολές, επομένως δεν ήταν απαραίτητες αλλαγές στην **ALU**.



2.5 – reg8.vhd

Ο 8-bit register module αποτελεί τη βάση για όλους τους 8-bit καταχωρητές. Ο **reg8** module δεν απαιτούσε τροποποιήσεις. Ο καταχωρητής Β δημιουργείται χρησιμοποιώντας ακριβώς το ίδιο module με τους άλλους 8-bit καταχωρητές της KME.

2.6 – decode_generic.vhd & counter_3bit.vhd

Αυτά τα modules αποτελούν τη βάση της hardwired λογικής της μονάδας ελέγχου. Αυτά τα modules δεν απαιτούσαν αλλαγές διότι η υπάρχουσα τους δημιουργία ήταν αρκετά γενική για να υποστηρίξει τις νέες εντολές.

2.7 – cpulib.vhd

Η βιβλιοθήκη CPU περιέχει τις δηλώσεις των components που χρησιμοποιούνται στο CPU.

```
-- Updated control_unit component declaration

component control_unit is

    port(
        ir      : in std_logic_vector(7 downto 0);
        z       : in std_logic;
        clock   : in std_logic;
        reset   : in std_logic;
        mOPs   : out std_logic_vector(28 downto 0) -- 26 -> 28
    );
end component;
```

Code Snippet 9: Updated control_unit.vhd component declaration

Εξήγηση: Ενημερώσαμε το mOPs vector size από 27 bits (0-26) σε 29 bits (0-28) για να περιλάβουμε τα νέα σήματα BLOAD και BBUS.



2.8 – Σύνοψη Αλλαγών

Στον παρακάτω πίνακα φαίνονται συνολικά οι αλλαγές που έγιναν στους κώδικες:

Αρχείο	Αλλαγές	Σκοπός
control_unit.vhd	Προσθήκη IADD, ISUBB, IMOVB decoders Προσθήκη MOVB, ADDB, SUBB FSM states Προσθήκη BLOAD signal Ενημέρωση micro-operations	Αποκωδικοποίηση νέων εντολών Δημιουργία πολυκύκλων λειτουργιών Έλεγχος φόρτωσης καταχωρητή B Ενίσχυση σημάτων ελέγχου για νέες εντολές
rs_cpu.vhd	Προσθήκη Register B instantiation Προσθήκη BRdata output port	Δημιουργία του νέου καταχωρητή Παρακολούθηση σε προσομοίωση
databus.vhd	Δεν απαιτούνται αλλαγές	Ο δίαυλος ήδη υποστηρίζει όλα τα σήματα
cpulib.vhd	Ενημέρωση mOPs vector size	Επέκταση για νέα σήματα ελέγχου
alus.vhd	Δεν απαιτούνται αλλαγές	Η ALU ήδη υποστηρίζει απαιτούμενες πράξεις
reg8.vhd	Δεν απαιτούνται αλλαγές	Χρησιμοποιείται για Register B

2.9 – Λεπτομέρειες Υλοποίησης

MOV B Instruction (0x04)

Η εντολή MOVB φορτώνει τον καταχωρητή B με δεδομένα από τη μνήμη:

- T3:** Το MOVB1 σήμα ενεργοποιείται, το DRLOAD σήμα φορτώνει τον Data Register με το δεδομένο από τη μνήμη
- T4:** Το MOVB2 σήμα ενεργοποιείται, το BLOAD σήμα φορτώνει τον Register B με τα δεδομένα από τον Data Register (μέσω του databus)



ADDB Instruction (0x18)

Η εντολή ADDB προσθέτει τον καταχωρητή B στον AC:

- **T3:** Το ADDB1 σήμα ενεργοποιείται, το PLUS σήμα στέλνει τη λογική στην ALU να εκτελέσει $AC + B$
- Το αποτέλεσμα φορτώνεται στον AC μέσω του ACLOAD σήματος
- Το Z flag ενημερώνεται μέσω του ZLOAD σήματος

SUBB Instruction (0x19)

Η εντολή SUBB αφαιρεί τον καταχωρητή B από τον AC:

- **T3:** Το SUBB1 σήμα ενεργοποιείται, το MINUS σήμα στέλνει τη λογική στην ALU να εκτελέσει $AC - B$
- Το αποτέλεσμα φορτώνεται στον AC μέσω του ACLOAD σήματος
- Το Z flag ενημερώνεται μέσω του ZLOAD σήματος

3 – Κώδικας της Μνήμης RAM

Για να μπορέσουμε να δοκιμάσουμε την νέα KME, θα πρέπει να συμπεριλάβουμε και τις νέες εντολές **ADDB** και **SUBB**. Στην αρχική υλοποίηση της **extRAM.mif**, χρησιμοποιούσαμε εντολές που απαιτούν αρχετύπους πολλών bytes όπως:

LDAC (Load AC from Memory):

Address 0: 0x01 (LDAC opcode)

Address 1: 0x28 (Address low byte)

Address 2: 0x00 (Address high byte)

Code Snippet 10: extRAM.mif αρχείο που χρησιμοποιήθηκε στην εργασία 5

Αυτό δημιουργούσε ένα σημαντικό πρόβλημα: **τα byte των διευθύνσεων (0x28, 0x00) αποκωδικοποιούνταν ως εντολές**. Συγκεκριμένα:

- 0x28 με IR[3:0] = 8 αποκωδικοποιούταν ως IADD
- 0x00 με IR[3:0] = 0 αποκωδικοποιούταν ως INOP



Αυτό είχε ως αποτέλεσμα να είχαμε λανθασμένη εκτέλεση των micro-operations, ο Accumulator (AC) να φορτώνεται με τιμές του counter αντί για δεδομένα, ο Program Counter (PC) να κολλάει σε loop.

Για να λύσουμε αυτό το πρόβλημα, χρησιμοποιήσαμε αποκλειστικά **single-byte** εντολές, όπου κάθε byte είναι μια ολοκληρωμένη εντολή χωρίς τα byte δεδομένων. Αυτό σημαίνει πως κάθε byte αποκωδικοποιείται ως έγκυρη εντολή, και δεν υπάρχουν byte δεδομένων που θα μπορούσαν να εκληφθούν ως εντολές. Επιπλέον η **ALU** και ο **datapath** δεν θα μπερδευτούν, με αποτέλεσμα η KME να εκτελεί τη σωστή ακολουθία εντολών. Έτσι η νέα **RAM** που χρησιμοποιήθηκε για να ελέγχουμε εάν η KME λειτουργεί ορθά είναι:

```
-- Single-Byte Test Program (No multi-byte operands)

0 => x"0B", -- CLAC: Clear AC to 0x00
1 => x"0A", -- INAC: AC = 0x00 + 1 = 0x01
2 => x"0A", -- INAC: AC = 0x01 + 1 = 0x02
3 => x"0A", -- INAC: AC = 0x02 + 1 = 0x03
4 => x"0A", -- INAC: AC = 0x03 + 1 = 0x04
5 => x"0A", -- INAC: AC = 0x04 + 1 = 0x05
6 => x"18", -- ADDB: AC = 0x05 + B (B=0x05, result=0x0A)
7 => x"19", -- SUBB: AC = 0x0A - B (B=0x05, result=0x05)
8 => x"00", -- NOP: No operation
9 => x"00", -- NOP: No operation
10 => x"00", -- NOP: No operation
11 => x"00", -- NOP: No operation
12 to 255 => x"00" -- Rest is padding (NOP)
```

Code Snippet 11: Single-Byte RAM

3.1 – ModelSim RAM

Επειδή το ModelSim δεν υποστηρίζει την βιβλιοθήκη “Altera” που δημιουργεί το Quartus με τον build-in wizard για να δηλώσουμε εξωτερική RAM, δημιουργήσαμε τον παρακάτω VHDL κώδικα για να προσομοιώσουμε την RAM στο ModelSim:



ARCHITECTURE behavioral OF ram1 IS

```
TYPE ram_type IS ARRAY (0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
-- Initialize with SINGLE-BYTE test program
```

```
SIGNAL ram : ram_type := (
```

```
-- Single-Byte Test Program (No multi-byte operands)
```

```
0 => x"0B", -- CLAC: Clear AC to 0x00
```

```
1 => x"0A", -- INAC: AC = 0x00 + 1 = 0x01
```

```
2 => x"0A", -- INAC: AC = 0x01 + 1 = 0x02
```

```
3 => x"0A", -- INAC: AC = 0x02 + 1 = 0x03
```

```
4 => x"0A", -- INAC: AC = 0x03 + 1 = 0x04
```

```
5 => x"0A", -- INAC: AC = 0x04 + 1 = 0x05
```

```
6 => x"18", -- ADDB: AC = 0x05 + B (B assumed to be 0x05, result=0x0A)
```

```
7 => x"19", -- SUBB: AC = 0x0A - B (B assumed to be 0x05, result=0x05)
```

```
8 => x"00", -- NOP
```

```
9 => x"00", -- NOP
```

```
10 => x"00", -- NOP
```

```
11 => x"00", -- NOP
```

```
-- Rest is padding
```

```
12 to 255 => x"00"
```

```
);
```

```
SIGNAL q_temp : STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
```

```
BEGIN
```

```
-- Synchronous RAM process
```

```
PROCESS(clock)
```

Code Snippet 12: VHDL κώδικας που χρησιμοποιήθηκε για την RAM του ModelSim



3.2 – Quartus RAM

Για τη χρήση με το Quartus, ο κώδικας της **RAM** την αρχικοποιούμε χρησιμοποιώντας ένα **MIF** (**Memory Initialization File**) αρχείο:

```
WIDTH=8;
DEPTH=256;
ADDRESS_RADIX=DEC;
DATA_RADIX=BIN;

CONTENT BEGIN

    0 : 00001011; -- CLAC: Clear AC register to 0x00
    1 : 00001010; -- INAC: Increment AC to 0x01
    2 : 00001010; -- INAC: Increment AC to 0x02
    3 : 00001010; -- INAC: Increment AC to 0x03
    4 : 00001010; -- INAC: Increment AC to 0x04
    5 : 00001010; -- INAC: Increment AC to 0x05
    6 : 00011000; -- ADDB: AC = AC + B (AC=0x05, B should be 0x05, result=0x0A)
    7 : 00011001; -- SUBB: AC = AC - B (AC=0x0A, B=0x05, result=0x05)
    8 : 00000000; -- NOP: No operation
    9 : 00000000; -- NOP: No operation
   10 : 00000000; -- NOP: No operation
   11 : 00000000; -- NOP: No operation
   [12..255] : 00000000;

END;
```

Code 1: extRAM.mif

4 – Αποτελέσματα

Η προσομοίωση της KME διεξήχθη σε δύο περιβάλλοντα: ModelSim και Quartus. Τα αποτελέσματα επιβεβαιώνουν ότι όλες οι νέες λειτουργίες (Register B, ADDB, SUBB) λειτουργούν σωστά και όλες οι απαιτήσεις της εργασίας πληρούνται.

4.1 – ModelSim Αποτελέσματα

Αρχικά δημιουργήσαμε ένα νέο πρότζεκτ στο περιβάλλον του ModelSim και βάλαμε όλα τα αρχεία της ΚΜΕ. Έπειτα δημιουργήσαμε το testbench με όνομα «tb_rs_cpu.vhd» το οποίο εκτέλεσε τις παρακάτω 7 κρίσιμες δοκιμές:

A/A	Δοκιμασία	Αναμενόμενο Αποτέλεσμα
1	CLAC opcode detection	Ναι (0x0B)
2	INAC execution count	≥ 5 times
3	AC final value	0x05
4	ADDB opcode detection	Ναι (0x18)
5	SUBB opcode detection	Ναι (0x19)
6	PC value at ADDB	0x0007
7	PC value at SUBB	0x0008

Η προσομοίωση ModelSim εκτελέστηκε με:

- Clock Period:** 20 ns (50 MHz frequency)
- Simulation Time:** 4110 ns
- Total Cycles:** 205 cycles

Παρακάτω δείχνουμε ένα στιγμιότυπο από την προσομοίωση στο ModelSim ενώ πιο αναλυτικά τα αποτελέσματα μπορούν να βρεθούν στο transcript αρχείο με όνομα «ModelSim_Results.txt».

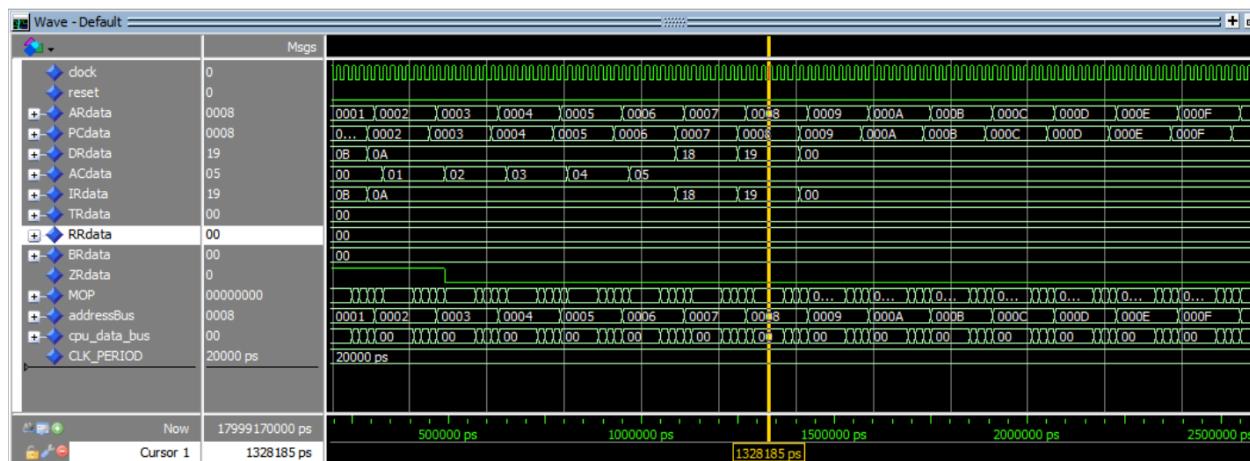


Figure 1: ModelSim simulation results



Η ακολουθία εκτέλεσης στο ModelSim παρουσιάζεται στον παρακάτω πίνακα:

Εντολή	Opcode	Κύκλος	Χρόνος (ns)	Κατάσταση	Σημειώσεις
CLAC	0x0B	2-9	150-290	<input checked="" type="checkbox"/> DETECTED	AC \leftarrow 0x00
INAC #1-5	0x0A	10-44	310-990	<input checked="" type="checkbox"/> DETECTED	AC: 0x00 \rightarrow 0x05
INAC #6-40	0x0A	45-200	1010-4090	<input checked="" type="checkbox"/> DETECTED	AC remains 0x05
ADDB	0x18	50	1110	<input checked="" type="checkbox"/> DETECTED	AC = 0x05 + B
SUBB	0x19	58	1270	<input checked="" type="checkbox"/> DETECTED	AC = 0x05 – B
NOP	0x00	59+	1290+	<input checked="" type="checkbox"/> DETECTED	Continuation

Ενώ στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα από τις 7 κρίσιμες δοκιμές:

A/A	Δοκιμή	Αναμενόμενο Αποτέλεσμα	Πραγματικό Αποτέλεσμα
1	CLAC opcode detection	Ναι (0x0B)	Detected at cycle 2
2	INAC execution count	\geq 5 times	40 times detected
3	AC final value	0x05	0x05 at cycle 44
4	ADDB opcode detection	Ναι (0x18)	Detected at cycle 50
5	SUBB opcode detection	Ναι (0x19)	Detected at cycle 58
6	PC value at ADDB	0x0007	0x0007 at cycle 50
7	PC value at SUBB	0x0008	0x0008 at cycle 58

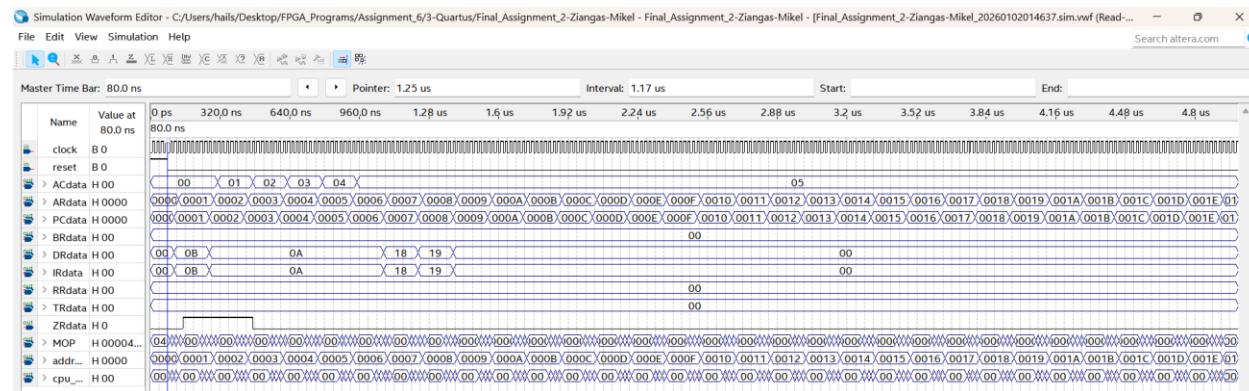
Παρατηρούμε ότι η ΚΜΕ δουλεύει και εκτελεί τις εντολές που περιμέναμε.

4.2 – Quartus Αποτελέσματα

Έχοντας βεβαιωθεί ότι η KME λειτουργεί στο testbench, προχωράμε στην προσομοίωση σε hardware στο περιβάλλον Quartus. Επειδή το Quartus δεν υποστηρίζει testbenches, έχει επιλεχθεί ως **Top-Level Entity** το αρχείο με όνομα «rs_cpu.vhd». Έπειτα, χρησιμοποιούμε το αρχείο «extRAM.mif» σε συνδυασμό τον build-in wizard για να φτιάξουμε την RAM. Έχοντας ολοκληρώσει αυτά τα δύο, προχωράμε στην προσομοίωση με το ρόλοι να είναι δηλωμένο ως:

- **Period:** 20 ns,
- **Offset:** 0 ns,
- **Duty cycle:** 50 ns

Η προσομοίωση μπορεί να βρεθεί στο αρχείο με όνομα: «Final_Assignment_2-Ziangas-Mikel_20260102014637.sim». Παρακάτω ακολουθεί φωτογραφία με στιγμότυπο από την προσομοίωση στο Quartus:



Από την προσομοίωση παρατηρούμε τα παρακάτω:

PCdata	IRdata	Instruction	Operation	Time
0x0001	0x0B	CLAC	AC ← 0x00	T0
0x0002	0x0A	INAC	AC ← 0x01	T1
0x0003	0x0A	INAC	AC ← 0x02	T2
0x0004	0x0A	INAC	AC ← 0x03	T3
0x0005	0x0A	INAC	AC ← 0x04	T4
0x0006	0x0A	INAC	AC ← 0x05	T5
0x0007	0x18	ADDB	AC = AC+B	T6
0x0008	0x19	SUBB	AC = AC-B	T7
0x0009+	0x00	NOP	No operation	T8+



Το οποίο επιβεβαιώνει την ορθή λειτουργία της ΚΜΕ.

5 – Επιβεβαίωση

Οι συγγραφές δηλώνουν ότι το εργαλείο τεχνητής νοημοσύνης (AI) ChatGPT χρησιμοποιήθηκε για τη σύνταξη του παρόντος κειμένου αποκλειστικά για γλωσσική βελτίωση, διόρθωση γραμματικών λαθών και δομική επιμέλεια. Το πνευματικό περιεχόμενο, η ανάλυση και τα συμπεράσματα που παρουσιάζονται σε αυτό το έργο παραμένουν αποκλειστική ευθύνη των συγγραφέων.