

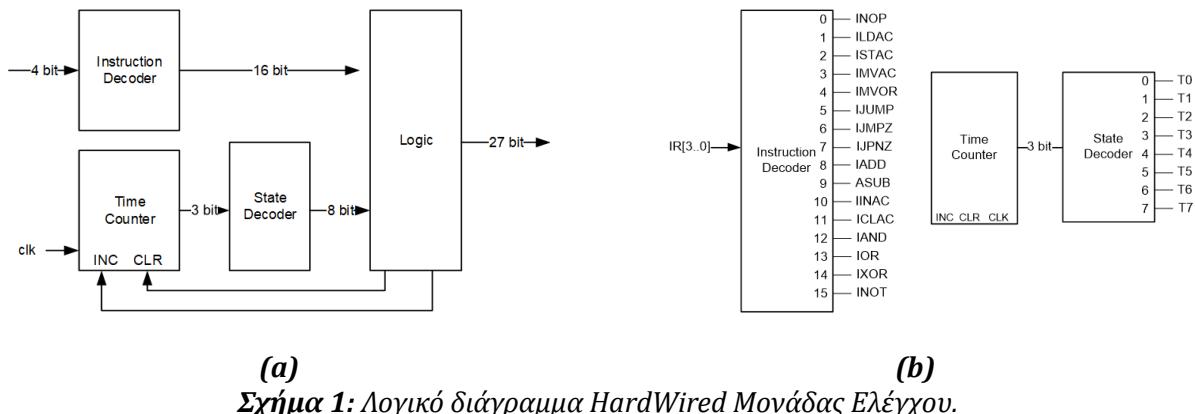
HARDWIRED ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

Νικηφόρος Μεϊχανετζόγλου 21207

Σκοπός

Με αφορμή την σχεδίαση και την εξομοίωση με διάφορους τρόπους, απλών ψηφιακών κυκλωμάτων θα κατακτηθεί το αντικείμενο της άσκησης αυτής που είναι η σχεδίαση της μονάδας ελέγχου, χρησιμοποιώντας την hardwired λογική η οποία θα χρησιμοποιηθεί, εναλλακτικά με την microprogrammed, κατά την τελική σύνθεση της σχετικά απλής KME.

Η μονάδα ελέγχου είναι αυτή που παρέχει στην KME τα απαραίτητα σήματα ελέγχου για τη λειτουργία της. Η λογική σχεδίασής της θα είναι η hardwired λογική, η οποία θα υλοποιηθεί με μία μηχανή πεπερασμένων καταστάσεων – FSM. Η μηχανή καταστάσεων αποτελείται από δύο αποκωδικοποιητές, ένα μετρητή και ένα συνδυαστικό κύκλωμα. Ο πρώτος αποκωδικοποιητής (instruction decoder) παράγει ένα ξεχωριστό σήμα για κάθε εντολή ενώ ο δεύτερος αποκωδικοποιητής (state decoder), με τη βοήθεια ενός απαριθμητή (time counter), παρακολουθεί ποια κατάσταση του κύκλου ανάκλησης η εκτέλεσης κάθε εντολής είναι ενεργή. Τέλος μια μονάδα συνδυαστικής λογικής παράγει μέσα από τα ξεχωριστά σήματα, σήματα ελέγχου για κάθε αποκωδικοποιητή αλλά και για τον απαριθμητή. Μια τέτοια μονάδα ελέγχου θα είχε την ακόλουθη μορφή (Σχήμα 1a).



Η σχεδίαση του αποκωδικοποιητή εντολών είναι σχετικά απλή. Δέχεται σαν είσοδο την έξοδο του καταχωρητή εντολών (IR) ενώ δεδομένου ότι χρησιμοποιούμαι μόνο τα 4 bit του καταχωρητή εντολών για το ρεπερτόριο των 16 εντολών της σχετικά απλής KME είναι προφανές ότι ο αποκωδικοποιητής εντολών είναι ένα αποκωδικοποιητής 4 σε 16. Από την άλλη εφόσον ο μεγιστος αριθμός καταστάσεων για το ρεπερτόριο των 16 εντολών είναι 8 καταστάσεις στη σχεδίαση μας χρησιμοποιούμαι έναν απαριθμητή 3 bit με δυνατότητα αύξησης και μηδενισμού και ένα

αποκωδικοποιητή 3 σε 8. Τα παραπάνω στοιχεία και οι έξοδοι τους φαίνονται με μεγαλύτερη λεπτομέρεια στο Σχήμα 1b.

Η ρουτίνα FETCH είναι η μόνη ρουτίνα η οποία δεν χρησιμοποιείται από το αποκωδικοποιητή εντολών. Δεδομένου ότι κατά τη ρουτίνα αυτή η προς εκτέλεση εντολή ανακαλείται από τη μνήμη η έξοδος του αποκωδικοποιητή μπορεί να είναι οποιαδήποτε. Σε αυτή μας τη σχεδίαση αναθέτουμε την κατάσταση T0 στην FETCH1 θέλοντας να εκμεταλλευτούμε το γεγονός ότι αυτή είναι προσπελάσιμη καθαρίζοντας (clear) τον απαριθμητή καταστάσεων. Όμοια αναθέτουμε την κατάσταση T1 και T2 στην FETCH2 και FETCH3 αντίστοιχα. Οι καταστάσεις των προς εκτέλεση εντολών εξαρτώνται αφενός από το opcode κάθε εντολής και αφετέρου από την τιμή του απαριθμητή καταστάσεων. Η T3 είναι η πρώτη χρονικά κατάσταση κάθε εντολής, η T4 η δεύτερη και ούτω καθεξής. Η μονάδα ελέγχου συνδέοντας με λογική and την κατάλληλη τιμή του απαριθμητή καταστάσεων με την έξοδο του αποκωδικοποιητή εντολών παράγει τις επιμέρους καταστάσεις για κάθε εντολή. Για παράδειγμα οι δύο πρώτες καταστάσεις της εντολής LDAC είναι:

$$\begin{aligned} \text{LDAC1} &= \text{ILDAC} \wedge \text{T3} \\ \text{LDAC2} &= \text{ILDAC} \wedge \text{T4} \end{aligned}$$

Η συνολική λίστα των επιμέρους καταστάσεων για όλες τις εντολές δίνεται στο πίνακα Γ.4.1 που ακολουθεί.

κατάσταση	λειτουργία	κατάσταση	λειτουργία
FETCH1	T0	JMPZY1	IJMPZ \wedge Z \wedge T3
FETCH2	T1	JMPZY2	IJMPZ \wedge Z \wedge T4
FETCH3	T3	JMPZY3	IJMPZ \wedge Z \wedge T5
NOP1	INOP \wedge T3	JMPZN1	IJMPZ \wedge Z' \wedge T3
LDAC1	ILDAC \wedge T3	JMPZN2	IJMPZ \wedge Z' \wedge T4
LDAC2	ILDAC \wedge T4	JPNZY1	IJPNZ \wedge Z' \wedge T3
LDAC3	ILDAC \wedge T5	JPNZY2	IJPNZ \wedge Z' \wedge T4
LDAC4	ILDAC \wedge T6	JPNZY3	IJPNZ \wedge Z' \wedge T5
LDAC5	ILDAC \wedge T7	JPNZN1	IJPNZ \wedge Z \wedge T3
STAC1	ISTAC \wedge T3	JPNZN2	IJPNZ \wedge Z \wedge T4
STAC2	ISTAC \wedge T4	ADD1	IADD \wedge T3
STAC3	ISTAC \wedge T5	SUB1	ISUB \wedge T3
STAC4	ISTAC \wedge T6	INAC1	IINAC \wedge T3
STAC5	ISTAC \wedge T7	CLAC1	ICLAC \wedge T3
MVAC1	IMVAC \wedge T3	AND1	IAND \wedge T3
MOVR1	IMOVR \wedge T3	OR1	IOR \wedge T3
JUMP1	IJUMP \wedge T3	XOR1	IXOR \wedge T3
JUMP2	IJUMP \wedge T4	NOT1	INOT \wedge T3
JUMP3	IJUMP \wedge T5		

Πίνακας 1: Παραγωγή καταστάσεων για τη σχετικά απλή KME

Έχοντας δημιουργήσει τις επιμέρους καταστάσεις για κάθε εντολή είναι ανάγκη να δημιουργήσουμε τα σήματα που θα οδηγούν τις εισόδους inc και clr του απαριθμητή καταστάσεων. Για να το επιτύχουμε αυτό συνδέοντας με λογική or την τελευταία κατάσταση κάθε εντολής για να δημιουργήσουμε το σήμα που θα οδηγήσει την είσοδο clr. Δεδομένου ότι η είσοδος inc πρέπει να είναι ενεργοποιημένη σε κάθε άλλη κατάσταση, μπορεί να υλοποιηθεί συνδέοντας με λογική or όλες τις υπόλοιπες καταστάσεις (πλην της τελευταίας) κάθε εντολής. Τέλος, η συνδυαστική λογική που

χρειάζεται για να παραχθούν τα κατάλληλα σήματα ελέγχου , για τα επιμέρους τμήματα της KME φαίνονται στο Πίνακα 2 που ακολουθεί:

Σήμα	Συνδιαστική Λογική
ARLOAD	FETCH1 \vee FETCH3 \vee LDAC3 \vee STAC3
ARINC	LDAC1 \vee STAC1 \vee JMPZY1 \vee JPNZY1
PCLOAD	JUMP3 \vee JMPZY3 \vee JPNZY3
PCINC	FETCH2 \vee LDAC1 \vee LDAC2 \vee STAC1 \vee STAC2 \vee JMPZN1 \vee JMPZN2 \vee JPNZN1 \vee JPNZN2
DRLOAD	FETCH2 \vee LDAC1 \vee LDAC2 \vee LDAC4 \vee STAC1 \vee STAC2 \vee STAC4 \vee JUMP1 \vee JUMP2 \vee JMPZY1 \vee JMPZY2 \vee JPNZY1 \vee JPNZY2
TRLOAD	LDAC2 \vee STAC2 \vee JUMP2 \vee JMPZY2 \vee JPNZY2
IRLOAD	FETCH3
RLOAD	MVAC1
ACLOAD	LDAC5 \vee MOVR1 \vee ADD1 \vee SUB1 \vee INAC1 \vee CLAC1 \vee AND1 \vee OR1 \vee XOR1 \vee NOT1
ZLOAD	LDAC5 \vee MOVR1 \vee ADD1 \vee SUB1 \vee INAC1 \vee CLAC1 \vee AND1 \vee OR1 \vee XOR1 \vee NOT1
READ	FETCH2 \vee LDAC1 \vee LDAC2 \vee LDAC4 \vee STAC1 \vee STAC2 \vee JUMP1 \vee JUMP2 \vee JMPZY1 \vee JMPZY2 \vee JPNZY1 \vee JPNZY2
WRITE	STAC5
MEMBUS	FETCH2 \vee LDAC1 \vee LDAC2 \vee LDAC4 \vee STAC1 \vee STAC2 \vee JUMP1 \vee JUMP2 \vee JMPZY1 \vee JMPZY \vee JPNZY1 \vee JPNZY2
BUSMEM	STAC5
PCBUS	FETCH1 or FETCH3
DRBUS	LDAC2 \vee LDAC3 \vee LDAC5 \vee STAC2 \vee STAC3 \vee STAC5 \vee JUMP2 \vee JUMP3 \vee JMPZY2 \vee JMPZY3 \vee JPNZY2 \vee JPNZY3
TRBUS	LDAC3 \vee STAC3 \vee JUMP3 \vee JMPZY3 \vee JPNZY3
RBUS	MOVR1 \vee ADD1 \vee SUB1 \vee AND1 \vee OR1 \vee XOR1
ACBUS	STAC4 \vee MVAC1
ANDOP	AND1
OROP	OR1
XOROP	XOR1
NOTOP	NOT1
ACINC	INAC1
ACZERO	CLAC1
PLUS	ADD1
MINUS	SUB1

Πίνακας 2: Παραγωγή σημάτων ελέγχου για τη σχετικά απλή KME

Αποκωδικοποιητής Εντολών

Γράψτε τον κώδικα για τον αποκωδικοποιητή 4 σε 16 με σήμα εισόδου D_{in} εύρους 4 bit και σήμα εξόδου D_{out} εύρους 16 bit. Το κύκλωμα αυτό όπως είναι γνωστό θα αντιστοιχεί την τιμή (opcode) κάθε μιας από τις 16 εντολές που εμφανίζεται στην είσοδο του σε μία από τις 16 εξόδους του.

Γράψτε εδώ το πρόγραμμά σας:

library ieee;

use ieee.std_logic_1164.all;

```

entity decoder4to16 is
    port (
        Din : in std_logic_vector(3 downto 0);
        Dout : out std_logic_vector(15 downto 0)
    );
end decoder4to16;

```

```

architecture behavior of decoder4to16 is
begin
    process(Din)
    begin
        case Din is
            when "0000" => Dout(0) <= '1'; -- NOP
            when "0001" => Dout(1) <= '1'; -- LDAC
            when "0010" => Dout(2) <= '1'; -- STAC
            when "0011" => Dout(3) <= '1'; -- MVAC
            when "0100" => Dout(4) <= '1'; -- MOVR
            when "0101" => Dout(5) <= '1'; -- JUMP
            when "0110" => Dout(6) <= '1'; -- JMPZ
            when "0111" => Dout(7) <= '1'; -- JPNZ
            when "1000" => Dout(8) <= '1'; -- ADD
            when "1001" => Dout(9) <= '1'; -- SUB
            when "1010" => Dout(10) <= '1'; -- INAC
            when "1011" => Dout(11) <= '1'; -- CLAC
            when "1100" => Dout(12) <= '1'; -- AND
            when "1101" => Dout(13) <= '1'; -- OR
            when "1110" => Dout(14) <= '1'; -- XOR
            when others => Dout(15) <= '1'; -- NOT
        end case;
    end process;
end behavior;

```

```
end case;  
end process;  
end behavior;
```

Πρόγραμμα 1: Ο αποκωδικοποιητής εντολών.

Αποκωδικοποιητής Καταστάσεων

Γράψτε τον κώδικα για τον αποκωδικοποιητή 3 σε 8 με σήμα εισόδου D_{in} εύρους 3 bit και σήμα εξόδου D_{out} εύρους 8 bit. Το κύκλωμα αυτό θα αντιστοιχεί την τιμή μέτρησης από τον μετρητή που εμφανίζεται στην είσοδο του σε μία από τις 8 εξόδους του η οποίες και θα συμβολίζουν την παρούσα κατάσταση.

Γράψτε εδώ το πρόγραμμά σας:

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity decoder3to8 is  
port (  
    Din : in std_logic_vector(2 downto 0);  
    Dout : out std_logic_vector(7 downto 0)  
);  
end decoder3to8;
```

architecture behavior of decoder3to8 is

```
begin  
process(Din)  
begin  
    case Din is  
        when "000" => Dout(0) <= '1'; -- T0  
        when "001" => Dout(1) <= '1'; -- T1  
        when "010" => Dout(2) <= '1'; -- T2  
        when "011" => Dout(3) <= '1'; -- T3  
        when "100" => Dout(4) <= '1'; -- T4
```

```

when "101" => Dout(5) <= '1'; -- T5
when "110" => Dout(6) <= '1'; -- T6
when others => Dout(7) <= '1'; -- T7
end case;
end process;
end behavior;

```

Πρόγραμμα 2: Ο αποκωδικοποιητής καταστάσεων.

Απαριθμητής

Γράψτε τον κώδικα για έναν μετρητή με εύρος 3-bits με σήματα εισόδου/ελέγχου inc για την αύξηση κατά ένα και rst για εκκαθάριση και σήμα εξόδου count .

Γράψτε εδώ το πρόγραμμά σας:

```

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

entity counter3bit is
port (
    clock : in std_logic;
    rst : in std_logic;
    inc : in std_logic;
    clr : in std_logic;
    count : out std_logic_vector(2 downto 0)
);
end counter3bit;

```

architecture behavior of counter3bit is

```

signal temp_count : std_logic_vector(2 downto 0);

```

```

begin
  process(clock, rst)
    begin
      if rst = '1' then
        temp_count <= (others => '0');
      elsif rising_edge(clock) then
        if clr = '1' then
          temp_count <= (others => '0');
        elsif inc = '1' then
          temp_count <= temp_count + 1;
        end if;
      end if;
    end process;

    count <= temp_count;
  end behavior;

```

Πρόγραμμα 3: Ο απαριθμητής των 3-bits.

Μονάδα Ελέγχου.

Έχοντας ολοκληρώσει τη συγγραφή του κώδικα για τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γραφεί το συνολικό πρόγραμμα περιγραφής της μονάδας ελέγχου. Σημειώνεται εδώ ότι δεδομένου ότι το κύκλωμα παραγωγής των σημάτων ελέγχου τόσο της ΚΜΕ όσο και του μετρητή καταστάσεων (σχήμα 1) είναι εξαιρετικά απλό δεν είναι απαραίτητη η συγγραφή ξεχωριστού στοιχείου για αυτό.

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

Γράψτε εδώ το πρόγραμμά σας:

Πρόγραμμα 4: βιβλιοθήκη στοιχείων για την μονάδα ελέγχου.

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 5) γράψτε τον κώδικα περιγραφής για της μονάδας ελέγχου, δηλαδή της μηχανής πεπερασμένων καταστάσεων, έτσι ώστε διαμορφώνεται από τα επιμέρους στοιχεία και το σχήμα 1. Τα σήματα που θα δέχεται σαν είσοδο το κύκλωμα, εκτός των σημάτων clock και reset, θα είναι τα τέσσερα (4) λιγότερο σημαντικά bit του καταχωρητή εντολών

(ir) και η τιμή του καταχωρητή σημαίας (z). Σαν έξοδοι λαμβάνεται το σήμα mOPs που αντιστοιχεί στην κάθε μικροεντολή εύρους 3627-bits.

Γράψτε εδώ το πρόγραμμά σας:

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

library lpm;
use lpm.lpm_components.all;
use work.hardwiredlib.all;

entity hardwired is
port (
    ir      : in std_logic_vector(3 downto 0);
    clock, reset : in std_logic;
    z       : in std_logic;
    mOPs    : out std_logic_vector(26 downto 0)
);
end hardwired;
```

architecture arc of hardwired is

```
signal count_out : std_logic_vector(2 downto 0);
signal cnt_inc  : std_logic;
signal cnt_clr  : std_logic;

signal I_dec    : std_logic_vector(15 downto 0); -- Instruction Signals
signal T_dec    : std_logic_vector(7 downto 0); -- Time/State Signals

signal INOP, ILDAC, ISTAC, IMVAC, IMOVR, IJUMP, IJMPZ, IJPNZ : std_logic;
```

```

signal IADD, ISUB, IINAC, ICLAC, IAND, IOR, IXOR, INOT    : std_logic;

signal T0, T1, T2, T3, T4, T5, T6, T7 : std_logic;

signal LDAC, STAC : std_logic_vector(4 downto 0);
signal MVAC, MOVR, NOP : std_logic;

signal FETCH, JUMP, JMPZY, JPNZY : std_logic_vector(2 downto 0);
signal JMPZN, JPNZN : std_logic_vector(1 downto 0);

signal ADD_OP, SUB_OP, INAC, CLAC, AND_OP, OR_OP, XOR_OP, NOT_OP : std_logic;

begin

-- 1. Instantiation of Components

block0: decoder4to16 port map (
    Din => ir,
    Dout => I_dec
);

block1: decoder3to8 port map (
    Din => count_out,
    Dout => T_dec
);

block2: counter3bit port map (
    clock => clock,
    rst  => reset,
    inc  => cnt_inc,

```

```
    clr => cnt_clr,  
    count => count_out  
);  
  
INOP <= I_dec(0); ILDAC <= I_dec(1); ISTAC <= I_dec(2); IMVAC <= I_dec(3);  
IMOVR <= I_dec(4); IJUMP <= I_dec(5); IJMPZ <= I_dec(6); IJPNZ <= I_dec(7);  
IADD <= I_dec(8); ISUB <= I_dec(9); IINAC <= I_dec(10); ICLAC <= I_dec(11);  
IAND <= I_dec(12); IOR <= I_dec(13); IXOR <= I_dec(14); INOT <= I_dec(15);
```

```
T0 <= T_dec(0); T1 <= T_dec(1); T2 <= T_dec(2); T3 <= T_dec(3);  
T4 <= T_dec(4); T5 <= T_dec(5); T6 <= T_dec(6); T7 <= T_dec(7);
```

```
FETCH(0) <= T0;  
FETCH(1) <= T1;  
FETCH(2) <= T2;
```

```
NOP <= INOP AND T3;
```

```
LDAC(0) <= ILDAC AND T3;  
LDAC(1) <= ILDAC AND T4;  
LDAC(2) <= ILDAC AND T5;  
LDAC(3) <= ILDAC AND T6;  
LDAC(4) <= ILDAC AND T7;
```

```
STAC(0) <= ISTAC AND T3;  
STAC(1) <= ISTAC AND T4;  
STAC(2) <= ISTAC AND T5;  
STAC(3) <= ISTAC AND T6;
```

STAC(4) <= ISTAC AND T7;

MVAC <= IMVAC AND T3;

MOVR <= IMOVR AND T3;

JUMP(0) <= IJUMP AND T3;

JUMP(1) <= IJUMP AND T4;

JUMP(2) <= IJUMP AND T5;

JMPZY(0) <= IJMPZ AND Z AND T3;

JMPZY(1) <= IJMPZ AND Z AND T4;

JMPZY(2) <= IJMPZ AND Z AND T5;

JMPZN(0) <= IJMPZ AND (NOT Z) AND T3;

JMPZN(1) <= IJMPZ AND (NOT Z) AND T4;

JPNZY(0) <= IJPNZ AND (NOT Z) AND T3;

JPNZY(1) <= IJPNZ AND (NOT Z) AND T4;

JPNZY(2) <= IJPNZ AND (NOT Z) AND T5;

JPNZN(0) <= IJPNZ AND Z AND T3;

JPNZN(1) <= IJPNZ AND Z AND T4;

ADD_OP <= IADD AND T3;

SUB_OP <= ISUB AND T3;

INAC <= IINAC AND T3;

CLAC <= ICLAC AND T3;

AND_OP <= IAND AND T3;

OR_OP <= IOR AND T3;

XOR_OP <= IXOR AND T3;

NOT_OP <= INOT AND T3;

cnt_clr <= NOP OR LDAC(4) OR STAC(4) OR MVAC OR MOVR

OR JUMP(2) OR JMPZY(2) OR JMPZN(1)

OR JPNZY(2) OR JPNZN(1)

OR ADD_OP OR SUB_OP OR INAC OR CLAC

OR AND_OP OR OR_OP OR XOR_OP OR NOT_OP;

cnt_inc <= '1';

-- Based on Table 2

mOPs(26) <= FETCH(0) OR FETCH(2) OR LDAC(2) OR STAC(2); -- ARLOAD

mOPs(25) <= LDAC(0) OR STAC(0) OR JMPZY(0) OR JPNZY(0); -- ARINC

mOPs(24) <= JUMP(2) OR JMPZY(2) OR JPNZY(2); -- PCLOAD

mOPs(23) <= FETCH(1) OR LDAC(0) OR LDAC(1) OR STAC(0) OR STAC(1) OR JMPZN(0) OR JMPZN(1) OR JPNZN(0) OR JPNZN(1); -- PCINC

mOPs(22) <= FETCH(1) OR LDAC(0) OR LDAC(1) OR LDAC(3) OR STAC(0) OR STAC(1) OR STAC(3) OR JUMP(0) OR JUMP(1) OR JMPZY(0) OR JMPZY(1) OR JPNZY(0) OR JPNZY(1); -- DRLOAD

mOPs(21) <= LDAC(1) OR STAC(1) OR JUMP(1) OR JMPZY(1) OR JPNZY(1); -- TRLOAD

mOPs(20) <= FETCH(2); -- IRLOAD

mOPs(19) <= MVAC; -- RLOAD

mOPs(18) <= LDAC(4) OR MOVR OR ADD_OP OR SUB_OP OR INAC OR CLAC OR AND_OP OR OR_OP OR XOR_OP OR NOT_OP; -- ACLOAD

mOPs(17) <= LDAC(4) OR MOVR OR ADD_OP OR SUB_OP OR INAC OR CLAC OR AND_OP OR OR_OP OR XOR_OP OR NOT_OP; -- ZLOAD

mOPs(16) <= FETCH(1) OR LDAC(0) OR LDAC(1) OR LDAC(3) OR STAC(0) OR STAC(1) OR JUMP(0) OR JUMP(1) OR JMPZY(0) OR JMPZY(1) OR JPNZY(0) OR JPNZY(1); -- READ

mOPs(15) <= STAC(4); -- WRITE

mOPs(14) <= FETCH(1) OR LDAC(0) OR LDAC(1) OR LDAC(3) OR STAC(0) OR STAC(1) OR JUMP(0) OR JUMP(1) OR JMPZY(0) OR JMPZY(1) OR JPNZY(0) OR JPNZY(1); -- MEMBUS

mOPs(13) <= STAC(4); -- BUSMEM
 mOPs(12) <= FETCH(0) OR FETCH(2); -- PCBUS
 mOPs(11) <= LDAC(1) OR LDAC(2) OR LDAC(4) OR STAC(1) OR STAC(2) OR STAC(4) OR JUMP(1) OR JUMP(2) OR JMPZY(1) OR JMPZY(2) OR JPNZY(1) OR JPNZY(2); -- DRBUS
 mOPs(10) <= LDAC(2) OR STAC(2) OR JUMP(2) OR JMPZY(2) OR JPNZY(2); -- TRBUS
 mOPs(9) <= MOVR OR ADD_OP OR SUB_OP OR AND_OP OR OR_OP OR XOR_OP; -- RBUS
 mOPs(8) <= STAC(3) OR MVAC; -- ACBUS
 mOPs(7) <= AND_OP; -- ANDOP
 mOPs(6) <= OR_OP; -- OROP
 mOPs(5) <= XOR_OP; -- XOROP
 mOPs(4) <= NOT_OP; -- NOTOP
 mOPs(3) <= INAC; -- ACINC
 mOPs(2) <= CLAC; -- ACZERO
 mOPs(1) <= ADD_OP; -- PLUS
 mOPs(0) <= SUB_OP; -- MINUS

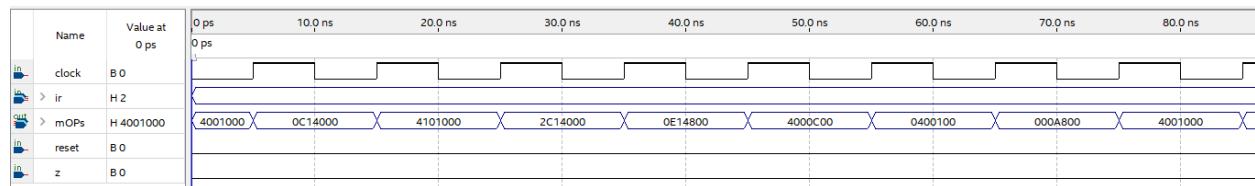
end arc;

Πρόγραμμα 5: Μονάδα Ελέγχου.

Εξομοίωση της Μονάδας Ελέγχου.

Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της μονάδας ελέγχου με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Με οδηγό τις προηγούμενες ασκήσεις, δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της μονάδας ελέγχου με τη βοήθεια του Waveform Editor για έξι (6) εντολές της KME , της επιλογής σας.

Σαν παράδειγμα ακολουθούν οι κυματομορφές εξομοίωσης για την εντολή STAC (ir=0x2).



Εικόνα 1: Κυματομορφές εξομοίωσης εντολής STAC.

Toπoθeτήστε εδώ τις κυματομορφές σας:

Όπως σας είπα και από κοντά, έχω εγκαταστήσει την έκδοση 25.1 του quartus prime lite edition και δεν μπορώ να κάνω εξομοιώσεις καθώς το questa χρειάζεται license για να λειτουργήσει.

Εικόνα 2: Κυματομορφές εξομοιώσης της μονάδας ελέγχου