

HARDWIRED ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

ΕΠΩΝΥΜΟ: ΣΤΑΜΑΤΑΚΗΣ

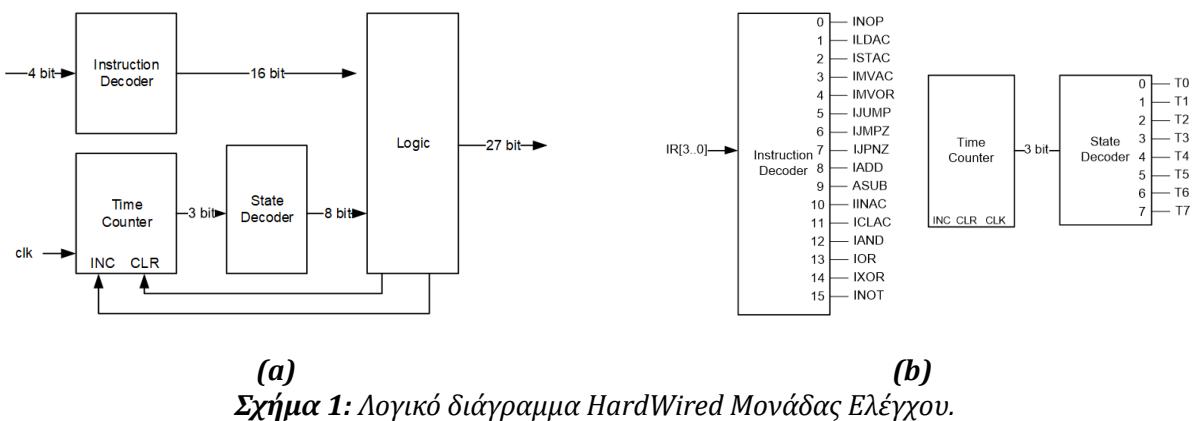
ΟΝΟΜΑ : ΦΩΤΗΣ

ΑΜ: 21048

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).



Σχήμα 1: Λογικό διάγραμμα HardWired Μονάδας Ελέγχου.
(a) (b)

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 μέγιστος 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

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

επιτύχουμε αυτό συνδέομε με λογική ορ 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 ορ όλες τις υπόλοιπες καταστάσεις (πλην της τελευταίας) κάθε εντολής. Τέλος, η συνδιαστική λογική που χρειάζεται για να παραχθούν τα κατάλληλα σήματα ελέγχου, για τα επιμέρους τμήματα της KME φαίνονται στο Πίνακα 2 που ακολουθεί:

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:

Πρόγραμμα 1: Ο αποκωδικοποιητής εντολών.

--FOTIS STAMATAKIS--

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

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

architecture rtl of instruction_decoder is

begin

process(Din)

begin

Dout <= (others => '0');

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'; -- ASUB
when "1010"=> Dout(10) <= '1'; -- INAC
when "1011"=> Dout(11) <= '1'; -- CALC
when "1100"=> Dout(12) <= '1'; -- AND
when "1101"=> Dout(13) <= '1'; -- OR
when "1110"=> Dout(14) <= '1'; -- XOR
when "1111"=> Dout(15) <= '1'; -- NOT
when others => null;
end case;
end process;
end rtl;

```

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:

Πρόγραμμα 2: Ο αποκωδικοποιητής καταστάσεων.

--FOTIS STAMATAKIS--

library ieee;

use ieee.std_logic_1164.all;

```
entity state_decoder is
  port(
    Din : in std_logic_vector(2 downto 0);
    Dout : out std_logic_vector(7 downto 0)
  );
end state_decoder;
```

```
architecture rtl of state_decoder is
begin
  process(Din)
  begin
    Dout <= (others => '0');
    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 "111" => Dout(7) <= '1'; -- T7
      when others => null;
    end case;
  end process;
end rtl;
```

```
end process;  
end rtl;
```

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:

Πρόγραμμα 3: Ο απαριθμητής των 3-bits.

--FOTIS STAMATAKIS--

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity counter3 is  
port(  
    clock : in std_logic;  
    rst : in std_logic;  
    inc : in std_logic;  
    count : out std_logic_vector(2 downto 0)  
);  
end counter3;
```

```

architecture rtl of counter3 is
    signal cnt : std_logic_vector(2 downto 0);
begin
    process(clock)
        begin
            if rising_edge(clock) then
                if rst = '1' then
                    cnt <= "000";
                elsif inc = '1' then
                    cnt <= cnt + 1;
                end if;
            end if;
        end process;

        count <= cnt;
    end rtl;

```

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.

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

package hardwiredlib is

constant INOP_IDX : integer := 0;
constant ILDAC_IDX : integer := 1;
constant ISTAC_IDX : integer := 2;
constant IMVAC_IDX : integer := 3;
constant IMOVR_IDX : integer := 4;
constant IJUMP_IDX : integer := 5;
constant IJMPZ_IDX : integer := 6;
constant IJPNZ_IDX : integer := 7;
constant IADD_IDX : integer := 8;
constant ISUB_IDX : integer := 9;
constant IINAC_IDX : integer := 10;
constant ICLAC_IDX : integer := 11;
constant IAND_IDX : integer := 12;
constant IOR_IDX : integer := 13;
constant IXOR_IDX : integer := 14;
constant INOT_IDX : integer := 15;

constant AR_LOAD : integer := 0;
constant AR_INC : integer := 1;
constant PC_LOAD : integer := 2;
constant PC_INC : integer := 3;
constant DR_LOAD : integer := 4;
constant TR_LOAD : integer := 5;
```

```

constant IR_LOAD : integer := 6;
constant R_LOAD : integer := 7;
constant AC_LOAD : integer := 8;
constant Z_LOAD : integer := 9;
constant READ_OP : integer := 10;
constant WRITE_OP: integer := 11;
constant MEM_BUS : integer := 12;
constant BUS_MEM : integer := 13;
constant PC_BUS : integer := 14;
constant DR_BUS : integer := 15;
constant TR_BUS : integer := 16;
constant R_BUS : integer := 17;
constant AC_BUS : integer := 18;
constant AND_OP : integer := 19;
constant OR_OP : integer := 20;
constant XOR_OP : integer := 21;
constant NOT_OP : integer := 22;
constant AC_INC : integer := 23;
constant AC_ZERO : integer := 24;
constant PLUS_OP : integer := 25;
constant MINUS_OP: integer := 26;

```

end package hardwiredlib;

package body hardwiredlib is

end package body hardwiredlib;

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 5) γράψτε τον κώδικα περιγραφής για της μονάδας ελέγχου, δηλαδή της μηχανής πεπερασμένων καταστάσεων, έτσι όπως 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:

Πρόγραμμα 5: Μονάδα Ελέγχου.

--FOTIS STAMATAKIS--

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

entity hardwired is

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

architecture arc of hardwired is

component instruction_decoder

```
port(  
    Din : in std_logic_vector(3 downto 0);  
    Dout : out std_logic_vector(15 downto 0)
```

```

    );
end component;

component state_decoder
port(
    Din : in std_logic_vector(2 downto 0);
    Dout : out std_logic_vector(7 downto 0)
);
end component;

component counter3
port(
    clock : in std_logic;
    rst : in std_logic;
    inc : in std_logic;
    count : out std_logic_vector(2 downto 0)
);
end component;

signal IR_reg : std_logic_vector(3 downto 0);
signal instr_out : std_logic_vector(15 downto 0);
signal state_out : std_logic_vector(7 downto 0);
signal count : std_logic_vector(2 downto 0);
signal inc, clr : std_logic;
signal mOPs_int : std_logic_vector(26 downto 0);
signal NOP, LDAC, STAC, MVAC, MOVR, JUMP, JMPZ, JPNZ, ADD, SUB, INAC, CLAC, IAND, IOR, IXOR,
INOT : std_logic;
signal FETCH1, FETCH2, FETCH3, NOP1 : std_logic;
signal LDAC1, LDAC2, LDAC3, LDAC4, LDAC5 : std_logic;

```

```
signal STAC1, STAC2, STAC3, STAC4, STAC5 : std_logic;  
signal MVAC1, MOVR1 : std_logic;  
signal JUMP1, JUMP2, JUMP3 : std_logic;  
signal JMPZY1, JMPZY2, JMPZY3, JMPZN1, JMPZN2 : std_logic;  
signal JPNZY1, JPNZY2, JPNZY3, JPNZN1, JPNZN2 : std_logic;  
signal ADD1, SUB1, INAC1, CLAC1, AND1, OR1, XOR1, NOT1 : std_logic;  
signal any_last_state : std_logic;
```

```
alias t0 is state_out(0);
```

```
alias t1 is state_out(1);
```

```
alias t2 is state_out(2);
```

```
alias t3 is state_out(3);
```

```
alias t4 is state_out(4);
```

```
alias t5 is state_out(5);
```

```
alias t6 is state_out(6);
```

```
alias t7 is state_out(7);
```

```
begin
```

```
process(clock)
```

```
begin
```

```
if rising_edge(clock) then
```

```
if mOPs_int(IR_LOAD) = '1' then
```

```
IR_reg <= ir;
```

```
end if;
```

```
end if;
```

```
end process;
```

U_Counter : counter3

```
port map ( clock => clock, rst => clr, inc => inc, count => count );
```

U_StateDecoder : state_decoder

```
port map ( Din => count, Dout => state_out );
```

U_InstructionDecoder : instruction_decoder

```
port map ( Din => IR_reg, Dout => instr_out );
```

$NOP \leqslant instr_out(0); LDAC \leqslant instr_out(1); STAC \leqslant instr_out(2); MVAC \leqslant instr_out(3);$
 $MOVR \leqslant instr_out(4); JUMP \leqslant instr_out(5); JMPZ \leqslant instr_out(6); JPNZ \leqslant instr_out(7);$
 $ADD \leqslant instr_out(8); SUB \leqslant instr_out(9); INAC \leqslant instr_out(10); CLAC \leqslant instr_out(11);$
 $IAND \leqslant instr_out(12); IOR \leqslant instr_out(13); IXOR \leqslant instr_out(14); INOT \leqslant instr_out(15);$
 $FETCH1 \leqslant t0; FETCH2 \leqslant t1; FETCH3 \leqslant t2;$
 $NOP1 \leqslant NOP \text{ and } t3;$
 $LDAC1 \leqslant LDAC \text{ and } t3; LDAC2 \leqslant LDAC \text{ and } t4; LDAC3 \leqslant LDAC \text{ and } t5; LDAC4 \leqslant LDAC \text{ and } t6;$
 $LDAC5 \leqslant LDAC \text{ and } t7;$
 $STAC1 \leqslant STAC \text{ and } t3; STAC2 \leqslant STAC \text{ and } t4; STAC3 \leqslant STAC \text{ and } t5; STAC4 \leqslant STAC \text{ and } t6; STAC5 \leqslant STAC \text{ and } t7;$
 $MVAC1 \leqslant MVAC \text{ and } t3; MOVR1 \leqslant MOVR \text{ and } t3;$
 $JUMP1 \leqslant JUMP \text{ and } t3; JUMP2 \leqslant JUMP \text{ and } t4; JUMP3 \leqslant JUMP \text{ and } t5;$
 $JMPZY1 \leqslant JMPZ \text{ and } Z \text{ and } t3; JMPZY2 \leqslant JMPZ \text{ and } Z \text{ and } t4; JMPZY3 \leqslant JMPZ \text{ and } Z \text{ and } t5;$
 $JMPZN1 \leqslant JMPZ \text{ and } (\text{not } Z) \text{ and } t3; JMPZN2 \leqslant JMPZ \text{ and } (\text{not } Z) \text{ and } t4;$
 $JPNZY1 \leqslant JPNZ \text{ and } (\text{not } Z) \text{ and } t3; JPNZY2 \leqslant JPNZ \text{ and } (\text{not } Z) \text{ and } t4; JPNZY3 \leqslant JPNZ \text{ and } (\text{not } Z) \text{ and } t5;$
 $JPNZN1 \leqslant JPNZ \text{ and } Z \text{ and } t3; JPNZN2 \leqslant JPNZ \text{ and } Z \text{ and } t4;$
 $ADD1 \leqslant ADD \text{ and } t3; SUB1 \leqslant SUB \text{ and } t3; INAC1 \leqslant INAC \text{ and } t3; CLAC1 \leqslant CLAC \text{ and } t3;$
 $AND1 \leqslant IAND \text{ and } t3; OR1 \leqslant IOR \text{ and } t3; XOR1 \leqslant IXOR \text{ and } t3; NOT1 \leqslant INOT \text{ and } t3;$

$\text{any_last_state} \leq \text{NOP1 or LDAC5 or STAC5 or MVAC1 or MOVR1 or JUMP3 or JMPZN2 or}$
 $\text{JMPZY3 or JPNZN2 or JPNZY3 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or XOR1 or NOT1};$
 $\text{clr} \leq \text{reset or any_last_state};$
 $\text{inc} \leq \text{not any_last_state};$

$\text{process(FETCH1, FETCH2, FETCH3, LDAC1, LDAC2, LDAC3, LDAC4, LDAC5,}$
 $\text{STAC1, STAC2, STAC3, STAC4, STAC5, MVAC1, MOVR1, JUMP1, JUMP2, JUMP3,}$
 $\text{JMPZY1, JMPZY2, JMPZY3, JMPZN1, JMPZN2, JPNZY1, JPNZY2, JPNZY3, JPNZN1, JPNZN2,}$
 $\text{ADD1, SUB1, INAC1, CLAC1, AND1, OR1, XOR1, NOT1})$

begin

$mOPs_int \leq (\text{others} \Rightarrow '0');$
 $mOPs_int(\text{AR_LOAD}) \leq \text{FETCH1 or FETCH3 or LDAC3 or STAC3};$
 $mOPs_int(\text{AR_INC}) \leq \text{LDAC1 or STAC1 or JMPZY1 or JPNZY1};$
 $mOPs_int(\text{PC_LOAD}) \leq \text{JUMP3 or JMPZY3 or JPNZY3};$
 $mOPs_int(\text{PC_INC}) \leq \text{FETCH2 or LDAC1 or LDAC2 or STAC1 or STAC2 or JMPZN1 or JMPZN2 or}$
 $\text{JPNZN1 or JPNZN2};$
 $mOPs_int(\text{DR_LOAD}) \leq \text{FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or STAC4 or}$
 $\text{JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2};$
 $mOPs_int(\text{TR_LOAD}) \leq \text{LDAC2 or STAC2 or JUMP2 or JMPZY2 or JPNZY2};$
 $mOPs_int(\text{IR_LOAD}) \leq \text{FETCH3};$
 $mOPs_int(\text{R_LOAD}) \leq \text{MVAC1};$
 $mOPs_int(\text{AC_LOAD}) \leq \text{LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or}$
 $\text{XOR1 or NOT1};$
 $mOPs_int(\text{Z_LOAD}) \leq mOPs_int(\text{AC_LOAD});$
 $mOPs_int(\text{READ_OP}) \leq \text{FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or JUMP1 or}$
 $\text{JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2};$
 $mOPs_int(\text{WRITE_OP}) \leq \text{STAC5};$
 $mOPs_int(\text{MEM_BUS}) \leq mOPs_int(\text{READ_OP});$
 $mOPs_int(\text{BUS_MEM}) \leq \text{STAC5};$
 $mOPs_int(\text{PC_BUS}) \leq \text{FETCH1 or FETCH3};$
 $mOPs_int(\text{DR_BUS}) \leq \text{LDAC2 or LDAC3 or LDAC5 or STAC2 or STAC3 or STAC5 or JUMP2 or}$
 $\text{JUMP3 or JMPZY2 or JMPZY3 or JPNZY2 or JPNZY3};$

```

mOPs_int(TR_BUS) <= LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;
mOPs_int(R_BUS) <= MOVR1 or ADD1 or SUB1 or AND1 or OR1 or XOR1;
mOPs_int(AC_BUS) <= STAC4 or MVAC1;
mOPs_int(AND_OP) <= AND1;
mOPs_int(OR_OP) <= OR1;
mOPs_int(XOR_OP) <= XOR1;
mOPs_int(NOT_OP) <= NOT1;
mOPs_int(AC_INC) <= INAC1;
mOPs_int(AC_ZERO) <= CLAC1;
mOPs_int(PLUS_OP) <= ADD1;
mOPs_int(MINUS_OP) <= SUB1;
end process;

```

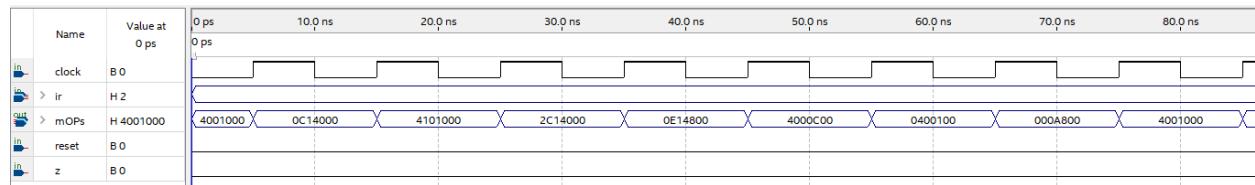
mOPs <= mOPs_int;

end arc;

Εξομοίωση της Μονάδας Ελέγχου.

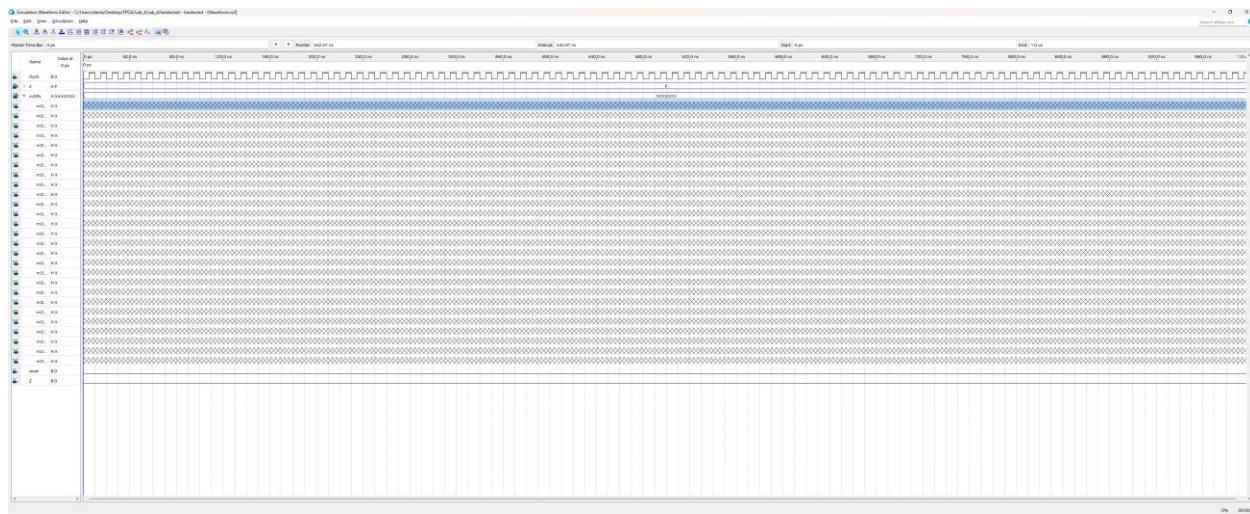
Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της μονάδας ελέγχου με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Με οδηγό τις προηγούμενες ασκήσεις, δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της μονάδας ελέγχου με τη βοήθεια του Waveform Editor για έξι (6) εντολές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

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



Εικόνα 1: Κυματομορφές εξομοίωσης εντολής STAC.

Τοποθετήστε εδώ τις κυματομορφές σας:



Εικόνα 2: Κυματομορφές εξομοίωσης της μονάδας ελέγχου