HARDWIRED
ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

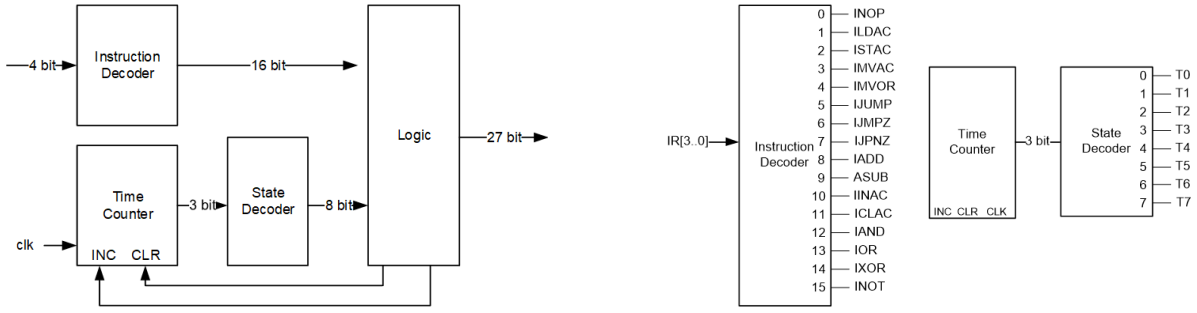
4

Δημήτρης Βαβουλιωτης, 21144

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 ΚΜΕ.

Η μονάδα ελέγχου είναι αυτή που παρέχει στην ΚΜΕ 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).



(a) ***(b)***

Σχήμα 1: *Λογικό διάγραμμα HardWired Μονάδας Ελέγχου.*

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 ΚΜΕ 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:

LDAC1 = ILDAC ^ T3
LDAC2 = ILDAC ^ T4

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

Πίνακας 1: *Παραγωγή καταστάσεων για τη σχετικά απλή ΚΜΕ*

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 συνδέουμε 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 ΚΜΕ

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

φαίνονται στο Πίνακα 2 που ακολουθεί:

Πίνακας 2: *Παραγωγή σημάτων ελέγχου για τη σχετικά απλή ΚΜΕ*

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;

-- ==========================================
-- 4-to-16 Instruction Decoder
-- Μετατρέπει ένα 4-bit opcode σε one-hot έξοδο
-- ==========================================

entity Decoder_4to16 is
   Port (
      Din  : in  STD_LOGIC_VECTOR(3 downto 0);   -- 4-bit είσοδος (opcode εντολής)
      Dout : out STD_LOGIC_VECTOR(15 downto 0)   -- 16-bit έξοδος (one-hot)
   );
end Decoder_4to16;

architecture Behavioral of Decoder_4to16 is
begin

   -- Συνδυαστική λογική:
   -- Κάθε αλλαγή στο Din ενημερώνει άμεσα την έξοδο Dout
   process(Din)
   begin
      case Din is

         -- Κάθε περίπτωση ενεργοποιεί ΜΟΝΟ ένα bit της Dout
         when "0000" =>
            Dout <= "0000000000000001";  -- INOP
```

```
when "0001" =>
   Dout <= "0000000000000010";  -- ILDAC

when "0010" =>
   Dout <= "0000000000000100";  -- ISTAC

when "0011" =>
   Dout <= "0000000000001000";  -- IMVAC

when "0100" =>
   Dout <= "0000000000010000";  -- IMOVR

when "0101" =>
   Dout <= "0000000000100000";  -- IJUMP

when "0110" =>
   Dout <= "0000000001000000";  -- IJMPZ

when "0111" =>
   Dout <= "0000000010000000";  -- IJPNZ

when "1000" =>
   Dout <= "0000000100000000";  -- IADD

when "1001" =>
   Dout <= "0000001000000000";  -- ISUB

when "1010" =>
   Dout <= "0000010000000000";  -- IINAC
```

```
        when "1011" =>
            Dout <= "0000100000000000";  -- ICLAC

        when "1100" =>
            Dout <= "0001000000000000";  -- IAND

        when "1101" =>
            Dout <= "0010000000000000";  -- IOR

        when "1110" =>
            Dout <= "0100000000000000";  -- IXOR

        when "1111" =>
            Dout <= "1000000000000000";  -- INOT

        -- Ασφάλεια για μη έγκυρες τιμές (X, U κλπ.)
        when others =>
            Dout <= (others => '0');

    end case;
  end process;

end Behavioral;
```

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.

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

-- ================================================
-- 3-to-8 State Decoder
-- Αποκωδικοποιεί την τιμή του μετρητή (T-state counter)
-- σε one-hot σήματα καταστάσεων T0 έως T7
-- ================================================

entity Decoder_3to8 is
   Port (
      Din  : in  STD_LOGIC_VECTOR(2 downto 0);  -- 3-bit είσοδος από τον απαριθμητή
      Dout : out STD_LOGIC_VECTOR(7 downto 0)   -- 8-bit έξοδος (T0–T7, one-hot)
   );
end Decoder_3to8;

architecture Behavioral of Decoder_3to8 is
begin

   -- Συνδυαστική λογική:
   -- Κάθε αλλαγή στο Din ενεργοποιεί άμεσα
   -- μία και μόνο μία κατάσταση T0–T7
   process(Din)
   begin
      case Din is

         -- Για κάθε δυαδική τιμή του Din
         -- ενεργοποιείται το αντίστοιχο T-state
```

```
    when "000" =>
        Dout <= "00000001";  -- T0: αρχική κατάσταση (FETCH1)

    when "001" =>
        Dout <= "00000010";  -- T1: δεύτερη φάση fetch (FETCH2)

    when "010" =>
        Dout <= "00000100";  -- T2: τρίτη φάση fetch (FETCH3)

    when "011" =>
        Dout <= "00001000";  -- T3: πρώτη μικροκατάσταση εντολής

    when "100" =>
        Dout <= "00010000";  -- T4: δεύτερη μικροκατάσταση

    when "101" =>
        Dout <= "00100000";  -- T5: τρίτη μικροκατάσταση

    when "110" =>
        Dout <= "01000000";  -- T6: τέταρτη μικροκατάσταση

    when "111" =>
        Dout <= "10000000";  -- T7: πέμπτη / τελευταία μικροκατάσταση

    -- Ασφαλής κατάσταση για μη έγκυρες τιμές
    when others =>
        Dout <= (others => '0');

end case;
```

```
    end process;

end Behavioral;
```

Τι κάνει συνοπτικά αυτός ο decoder

- ***Λαμβάνει την τιμή του 3-bit μετρητή.***
- ***Παράγει 8 σήματα καταστάσεων (T0–T7).***
- ***Μόνο μία κατάσταση είναι ενεργή κάθε φορά (one-hot).***
- ***Αποτελεί τη ραχοκοκαλιά της FSM της μονάδας ελέγχου.***

Γράψτε εδώ το πρόγραμμά σας:

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 .

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

-- ================================================
-- 3-bit Synchronous Counter (Time-State Counter)
-- Χρησιμοποιείται στη μονάδα ελέγχου για την
-- παραγωγή των χρονικών καταστάσεων T0–T7
-- ================================================

entity Counter is
    Port (
```

```
    clk   : in  STD_LOGIC;              -- Ρολόι συστήματος
    rst   : in  STD_LOGIC;              -- Σύγχρονο reset (επιστροφή στην T0)
    clr   : in  STD_LOGIC;              -- Clear από τη μονάδα ελέγχου
    inc   : in  STD_LOGIC;              -- Αύξηση κατά 1 (inc)
    count : out STD_LOGIC_VECTOR(2 downto 0)  -- Έξοδος 3-bit (τιμή απαριθμητή)
  );
end Counter;

architecture Behavioral of Counter is

  -- Εσωτερικός καταχωρητής 3-bit
  -- Αποθηκεύει την τρέχουσα τιμή της χρονικής κατάστασης
  signal counter_value : STD_LOGIC_VECTOR(2 downto 0) := "000";

begin

  -- Σύγχρονο process: ενεργοποιείται στη θετική ακμή του clock
  process(clk)
  begin
    if clk'event and clk = '1' then

      -- Προτεραιότητα 1: Reset
      -- Όταν rst = '1', ο απαριθμητής μηδενίζεται
      if rst = '1' then
        counter_value <= "000";

      -- Προτεραιότητα 2: Clear
      -- Ενεργοποιείται στην τελευταία μικροκατάσταση
      -- κάθε εντολής για επιστροφή στο FETCH (T0)
```

```
        elsif clr = '1' then
            counter_value <= "000";

        -- Προτεραιότητα 3: Increment
        -- Όταν inc = '1', ο απαριθμητής αυξάνει κατά 1
        elsif inc = '1' then
            counter_value <= counter_value + 1;

        -- Αν κανένα από τα παραπάνω δεν ισχύει,
        -- ο απαριθμητής διατηρεί την τρέχουσα τιμή του
        end if;

    end if;
  end process;

  -- Αντιστοίχιση της εσωτερικής τιμής στην έξοδο
  count <= counter_value;

end Behavioral;
```

Γράψτε εδώ το πρόγραμμά σας:

Τι κάνει συνοπτικά αυτός ο Counter

- ***Είναι σύγχρονος απαριθμητής 3-bit.***
- ***Μετράει τις χρονικές καταστάσεις T0–T7.***
- ***Ο inc τον προχωρά στην επόμενη κατάσταση.***
- ***Το clr και το rst τον επαναφέρουν στην αρχική κατάσταση (T0).***
- ***Χρησιμοποιείται ως time-state generator της FSM.***

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:

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


-- ================================================
-- Package: ask4lib
-- Περιέχει τις δηλώσεις components που
-- χρησιμοποιούνται στη μονάδα ελέγχου (hardwired)
-- ================================================


package ask4lib is


  -- ------------------------------------------
  -- Component: Decoder_4to16
  -- Αποκωδικοποιητής εντολών 4 → 16
  -- Μετατρέπει το opcode (IR[3:0]) σε one-hot σήματα
  -- ------------------------------------------
  component Decoder_4to16 is
    Port (
      Din  : in  STD_LOGIC_VECTOR(3 downto 0);   -- 4-bit είσοδος (opcode)
      Dout : out STD_LOGIC_VECTOR(15 downto 0)   -- 16 one-hot έξοδοι εντολών
    );
  end component;
```

```
-- ------------------------------------------------
-- Component: Decoder_3to8
-- Αποκωδικοποιητής καταστάσεων 3 → 8
-- Μετατρέπει την έξοδο του απαριθμητή
-- σε one-hot χρονικές καταστάσεις T0–T7
-- ------------------------------------------------
component Decoder_3to8 is
   Port (
      Din  : in  STD_LOGIC_VECTOR(2 downto 0);   -- 3-bit είσοδος από counter
      Dout : out STD_LOGIC_VECTOR(7 downto 0)    -- One-hot T0–T7
   );
end component;


-- ------------------------------------------------
-- Component: Counter
-- Σύγχρονος απαριθμητής 3-bit
-- Παράγει την ακολουθία χρονικών καταστάσεων
-- για τη μονάδα ελέγχου
-- ------------------------------------------------
component Counter is
   Port (
      clk   : in  STD_LOGIC;                -- Ρολόι συστήματος
      rst   : in  STD_LOGIC;                -- Σύγχρονο reset
      clr   : in  STD_LOGIC;                -- Clear από FSM (επιστροφή σε T0)
      inc   : in  STD_LOGIC;                -- Αύξηση κατά 1
      count : out STD_LOGIC_VECTOR(2 downto 0)   -- Έξοδος απαριθμητή
   );
end component;
```

end package ask4lib;

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 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:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library lpm;
use lpm.lpm_components.all;
use work.ask4lib.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 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, JUMP1 ,JUMP2 ,JUMP3 : std_logic ;
signal JMPZ1 ,JMPZY1, JMPZY2, JMPZY3 ,JMPZN1, JMPZN2 : std_logic ;
signal JPNZ1 ,JPNZY1 ,JPNZY2, JPNZY3 ,JPNZN1 ,JPNZN2 : std_logic ;
signal ADD1, SUB1 ,INAC1, CLAC1, AND1, OR1, XOR1, NOT1 : std_logic ;
signal cdata :  std_logic_vector(2 downto 0);
```

```
signal dout2 :  std_logic_vector(7 downto 0);

signal dout1 :  std_logic_vector(15 downto 0);

signal notz ,inc, clear : std_logic ;


signal INOP,ILDAC,ISTAC,IMVAC,IMOVR,IJMPZ,IJUMP,IADD,ISUB,IINAC,ICLAC,IAND,IOR,IXOR,INOT,IJPNZ   :
std_logic ;

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


begin


notZ<= not Z;



INOP <= dout1(0);

ILDAC <= dout1(1);

ISTAC <= dout1(2);

IMVAC <= dout1(3);

IMOVR <= dout1(4);

IJUMP <= dout1(5);

IJMPZ <= dout1(6);

IJPNZ <= dout1(7);

IADD <= dout1(8);

ISUB <= dout1(9);

IINAC <= dout1(10);

ICLAC <= dout1(11);

IAND <= dout1(12);

IOR <= dout1(13);

IXOR <= dout1(14);

INOT <= dout1(15);



T0 <= dout2(0);
```

```
T1 <= dout2(1);
T2 <= dout2(2);
T3 <= dout2(3);
T4 <= dout2(4);
T5 <= dout2(5);
T6 <= dout2(6);
T7 <= dout2(7);
```

```
FETCH1<=T0;
FETCH2<=T1;
FETCH3<=T2;
NOP1<=INOP and T3;
LDAC1<=ILDAC and T3;
LDAC2<=ILDAC and T4;
LDAC3<=ILDAC and T5;
LDAC4<=ILDAC and T6;
LDAC5<=ILDAC and T7;
STAC1<=ISTAC and T3;
STAC2<=ISTAC and T4;
STAC3<=ISTAC and T5;
STAC4<=ISTAC and T6;
STAC5<=ISTAC and T7;
MVAC1<=IMVAC and T3;
MOVR1<=IMOVR and T3;
JUMP1<=IJUMP and T3;
JUMP2<=IJUMP and T4;
JUMP3<=IJUMP and T5;
```

JMPZY1<=IJMPZ and Z and T3;

JMPZY2<=IJMPZ and Z and T4;

JMPZY3<=IJMPZ and Z and T5;

JMPZN1<=IJMPZ and notZ and T3;

JMPZN2<=IJMPZ and notZ and T4;

JPNZY1<=IJPNZ and notZ and T3;

JPNZY2<=IJPNZ and notZ and T4;

JPNZY3<=IJPNZ and notZ and T5;

JPNZN1<=IJPNZ and Z and T3;

JPNZN2<=IJPNZ and Z and T4;

ADD1<=IADD and T3;

SUB1<=ISUB and T3;

INAC1<=IINAC and T3;

CLAC1<=ICLAC and T3;

AND1<=IAND and T3;

OR1<=IOR and T3;

XOR1<=IXOR and T3;

NOT1<=INOT and T3;

mOPS(26)<=FETCH1 or FETCH3 or LDAC3 or STAC3;

mOPS(25)<=LDAC1 or STAC1 or JMPZY1 or JPNZY1;

mOPS(24)<=JUMP3 or JMPZY3 or JPNZY3;

mOPS(23)<=FETCH2 or LDAC1 or LDAC2 or STAC1 or STAC2 or JMPZN1 or JMPZN2 or JPNZN1 or JPNZN2;

mOPS(22)<=FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or STAC4 or JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;

mOPS(21)<=LDAC2 or STAC2 or JUMP2 or JMPZY2 or JPNZY2;

mOPS(20)<=FETCH3;

mOPS(19)<=MVAC1;

mOPS(18)<=LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or XOR1 or NOT1;

mOPS(17)<=LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or AND1 or OR1 or XOR1 or NOT1;

mOPS(16)<=FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;

```
mOPS(15)<=STAC5;

mOPS(14)<=FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or JUMP1 or JUMP2 or JMPZY1 or
JMPZY1  or JPNZY1 or JPNZY2;

mOPS(13)<=STAC5;

mOPS(12)<=FETCH1  or  FETCH3;

mOPS(11)<=LDAC2 or LDAC3 or LDAC5 or STAC2 or STAC3 or STAC5 or JUMP2 or JUMP3 or JMPZY2 or
JMPZY3 or JPNZY2 or JPNZY3;

mOPS(10)<=LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;

mOPS(9)<=MOVR1 or ADD1 or SUB1 or AND1 or  OR1 or XOR1;

mOPS(8)<=STAC4 or MVAC1;

mOPS(7)<=AND1;

mOPS(6)<=OR1;

mOPS(5)<=XOR1;

mOPS(4)<=NOT1;

mOPS(3)<=INAC1;

mOPS(2)<=CLAC1;

mOPS(1)<=ADD1;

mOPS(0)<=SUB1;



clear<= NOP1 OR LDAC5 OR STAC5 OR MVAC1 OR MOVR1 OR JUMP3 OR JMPZY3 OR JMPZN2 OR JPNZY3
        OR JPNZN2 OR ADD1 OR SUB1 OR INAC1 OR CLAC1 OR AND1 OR OR1 OR XOR1 OR NOT1;



 block0: Decoder_4to16 port map(Din=>ir, Dout=>dout1);

 block1: Counter

   port map(clk   => clock,rst   => reset,  clr   => clear, inc   => inc,count => cdata);

 block2: Decoder_3to8 port map(Din=>cdata, Dout=>dout2);

end arc;
```

Πρόγραμμα 5: *Μονάδα Ελέγχου.*

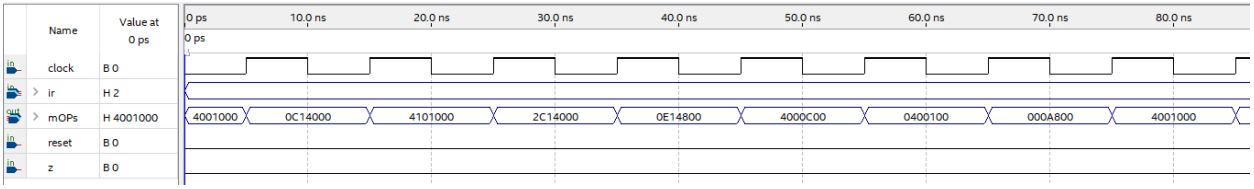
Ο παραπάνω κώδικας υλοποιεί τη hardwired μονάδα ελέγχου (Control Unit) μιας απλής ΚΜΕ. Χρησιμοποιεί έναν αποκωδικοποιητή εντολών 4→16 για τη μετατροπή του opcode σε one-hot σήματα εντολών, έναν απαριθμητή 3-bit και έναν αποκωδικοποιητή καταστάσεων

3→8 για την παραγωγή των χρονικών καταστάσεων T0–T7. Οι μικροκαταστάσεις κάθε εντολής παράγονται με συνδυαστική λογική (AND εντολής και T-state), ενώ τα σήματα ελέγχου mOPs προκύπτουν ως λογικές συναρτήσεις αυτών των μικροκαταστάσεων. Το σήμα inc επιτρέπει την πρόοδο του απαριθμητή στις ενδιάμεσες φάσεις, ενώ το clear μηδενίζει τον απαριθμητή στην τελευταία μικροκατάσταση κάθε εντολής, εξασφαλίζοντας την επιστροφή στη φάση FETCH. Συνολικά, ο κώδικας υλοποιεί μια πλήρη hardwired FSM για τον έλεγχο της ΚΜΕ.

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 ΚΜΕ , της επιλογής σας.

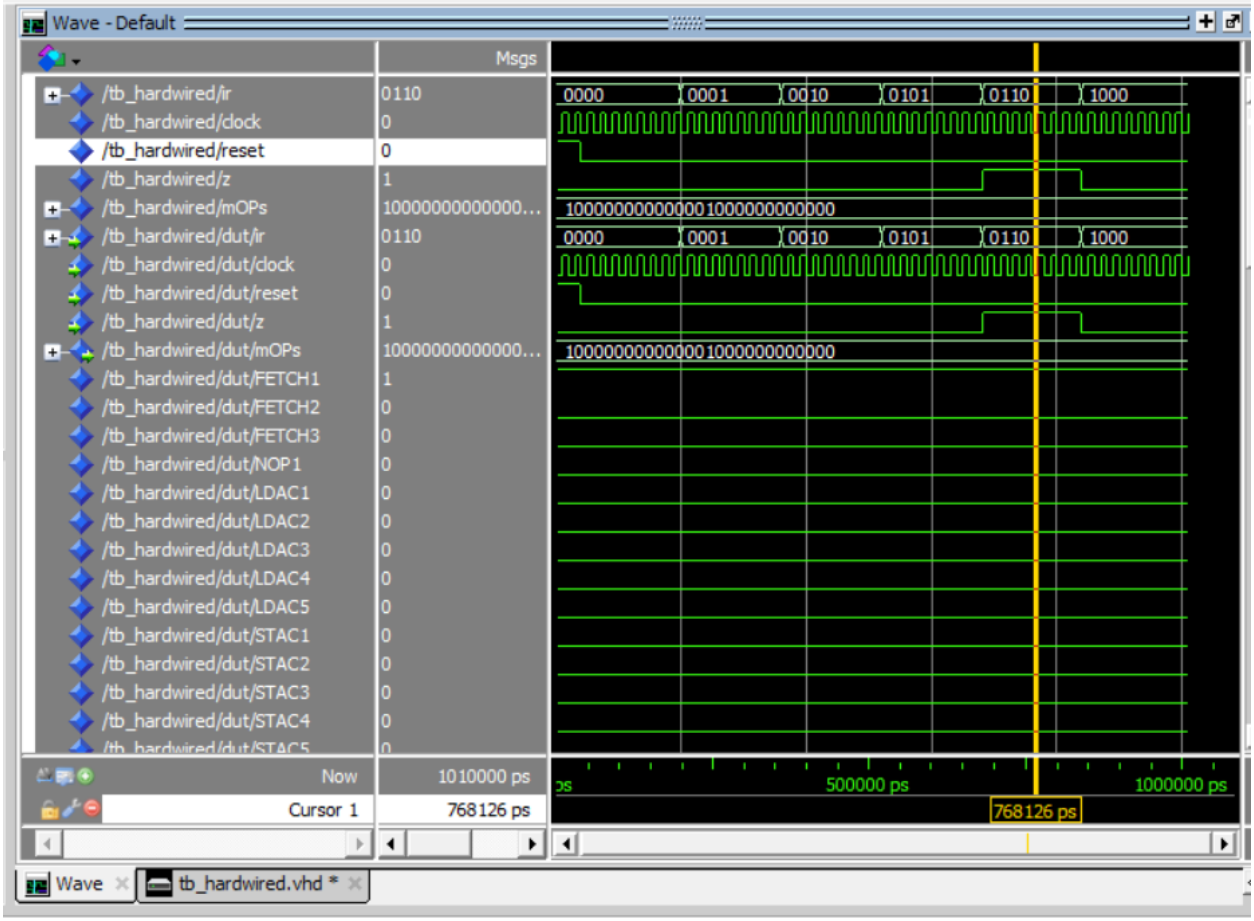
Σαν παράδειγμα ακολουθούν οι κυματομορφές εξομοίωσης για την εντολή STAC (ir=0x2).



Εικόνα 1: *Κυματομορφές εξομοίωσης εντολής STAC.*

Τοποθετήστε εδώ τις κυματομορφές σας:

Εικόνα 2: *Κυματομορφές εξομοίωσης της μονάδας ελέγχου*



tb_hardwired.vhd

```
library ieee;

use ieee.std_logic_1164.all;


entity tb_hardwired is

end entity;


architecture sim of tb_hardwired is

  signal ir    : std_logic_vector(3 downto 0) := "0000";

  signal clock : std_logic := '0';

  signal reset : std_logic := '1';
```

```
 signal z     : std_logic := '0';
 signal mOPs  : std_logic_vector(26 downto 0);

 constant Tclk : time := 20 ns;
begin

 -- DUT
 dut: entity work.hardwired
  port map(
    ir    => ir,
    clock => clock,
    reset => reset,
    z     => z,
    mOPs  => mOPs
  );

 -- CLOCK: συνεχίζει για πάντα (ΔΕΝ σταματά)
 clk_proc : process
 begin
  while true loop
    clock <= '0';
    wait for Tclk/2;
    clock <= '1';
    wait for Tclk/2;
  end loop;
 end process;

 -- STIMULI: reset + 6 εντολές (8 κύκλοι η κάθε μία)
 stim_proc : process
```

```
begin
  -- reset 2 κύκλοι
  reset <= '1';
  z <= '0';
  ir <= "0000";
  wait for 2*Tclk;   -- 40 ns
  reset <= '0';

  -- 1) NOP
  ir <= "0000"; z <= '0';
  wait for 8*Tclk;

  -- 2) LDAC
  ir <= "0001"; z <= '0';
  wait for 8*Tclk;

  -- 3) STAC
  ir <= "0010"; z <= '0';
  wait for 8*Tclk;

  -- 4) JUMP
  ir <= "0101"; z <= '0';
  wait for 8*Tclk;

  -- 5) JMPZ με Z=1
  ir <= "0110"; z <= '1';
  wait for 8*Tclk;

  -- 6) ADD
```

```
    ir <= "1000"; z <= '0';
    wait for 8*Tclk;

    wait; -- τερματισμός stimuli
  end process;

end architecture;
```