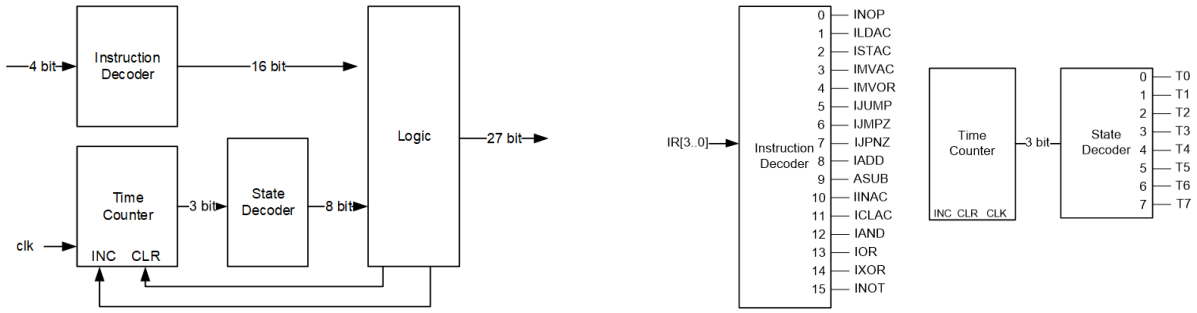
HARDWIRED ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

4

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~~ (**T2 is correct**)	**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 φαίνονται στο Πίνακα 2 που ακολουθεί:

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

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.

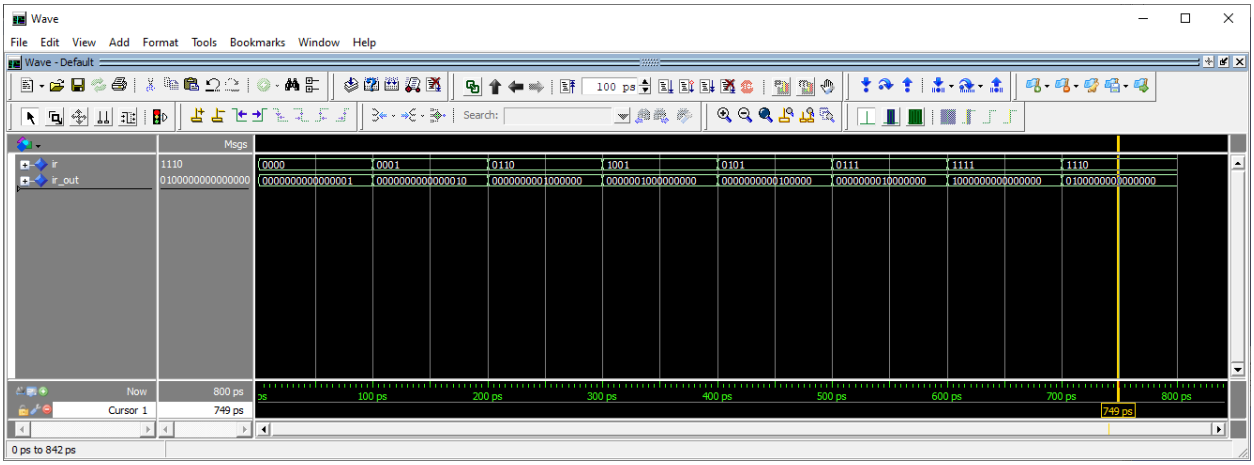
Γράψτε εδώ το πρόγραμμά σας:

Πρόγραμμα 1: *Ο αποκωδικοποιητής εντολών.*

Για να γλυτώσουμε χρόνο και πολυπλοκότητα, θα φτιάξουμε έναν generic αποκωδικοποιητή με τον οποίο θα ελέγχουμε τις εισόδους και εξόδους του με δύο generic μεταβλητές με όνομα: *INPUT_WIDTH* και *OUTPUT_WIDTH*. Έπειτα, για να ορίσουμε το 4x16 αποκωδικοποιητή, αρκεί να βάλουμε 4 και 16 στις generic μεταβλητές.

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

entity decoder_generic is
	generic(
		INPUT_WIDTH  : integer; -- Number of input bits
		OUTPUT_WIDTH : integer  -- Number of output bits (2^{input_width})
	);
	port(
		-- Input(s)
		din  : in std_logic_vector(INPUT_WIDTH-1 downto 0);

		-- Output(s)
		dout : out std_logic_vector(OUTPUT_WIDTH-1 downto 0)
	);
end entity;

architecture rtl of decoder_generic is
begin

	process(din)
	begin

		-- Default: all outputs 0
		dout <= (others => '0');

		-- Activate one output based on ir value
```

```
            dout(to_integer(unsigned(din))) <= '1';
        end process;
end architecture;
```



Εικόνα 1: Αποτέλεσμα του testbench με όνομα «***decoder_generalTb.vhd***» για τον αποκωδικοποιητή 4 σε 16.

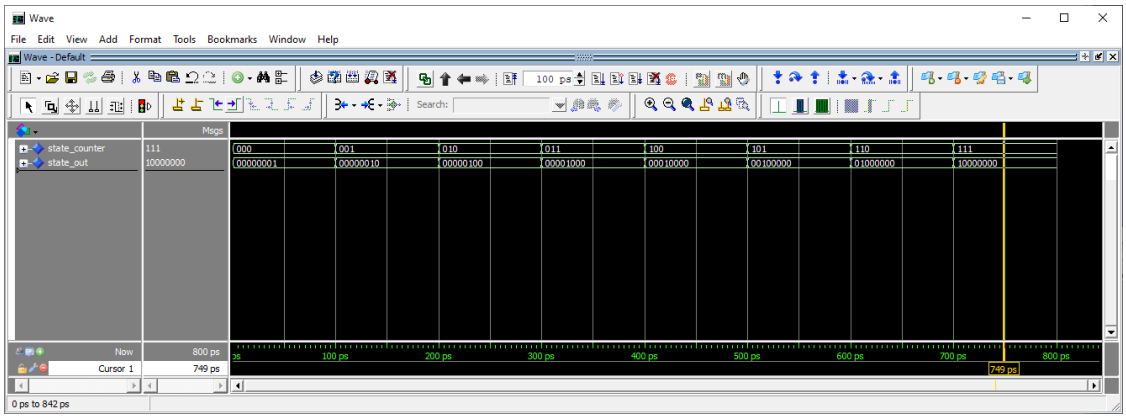
Αποκωδικοποιητής Καταστάσεων

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.*

Για να μην γράφουμε νέο κώδικα, θα χρησιμοποιήσουμε τον κώδικα από το πρόγραμμα 1 στο οποίο έχουμε φτιάξει ένα generic αποκωδικοποιητή και ορίζουμε την είσοδο και έξοδο με τις generic μεταβλητές: *INPUT_WIDTH* και *OUTPUT_WIDTH*. Θα πρέπει η μεταβλητή του *OUTPUT_WIDTH* να είναι: 2^{INPUT_WIDTH}. Άρα για να φτιάξουμε τον 3 σε 8 αποκωδικοποιητή, αρκεί να βάλουμε 3 και 8 στις generic μεταβλητές.



Εικόνα 2: Αποτέλεσμα του testbench με όνομα «***decoder_generalTb.vhd***» για τον αποκωδικοποιητή 3 σε 8.

Βλέπουμε ότι και στις δύο περιπτώσεις, οι αποκωδικοποιητές δουλεύουν όπως το περιμέναμε.

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 .

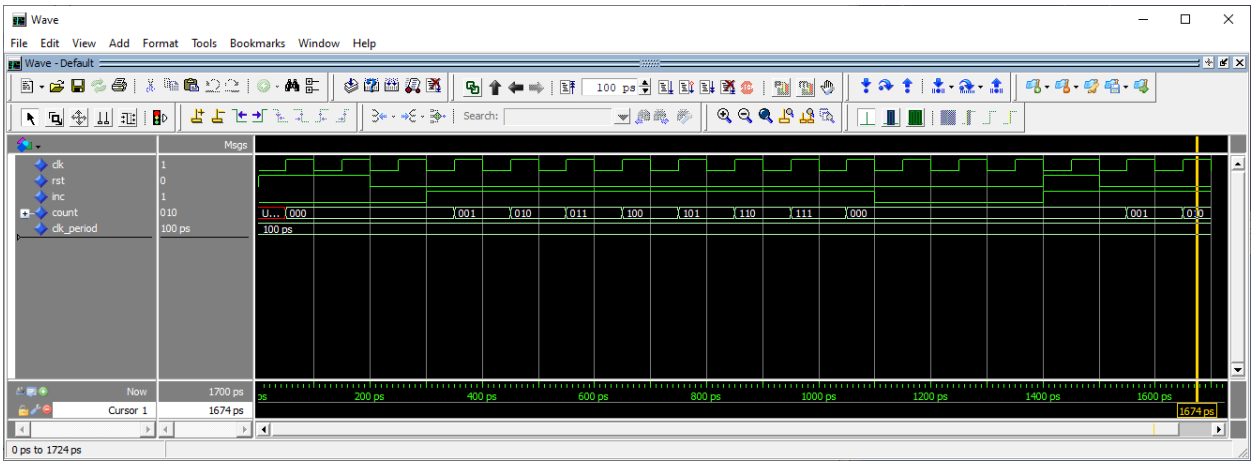
Γράψτε εδώ το πρόγραμμά σας:

Πρόγραμμα 3: *Ο απαριθμητής των 3-bits.*

Για να γράψουμε τον κώδικα του απαριθμητή, θα πάρουμε τον κώδικα του reg1bit από το πρώτο εργαστήριο, και θα το αλλάξουμε κατάλληλα, για να γίνει μετρητής των 3-bits:

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

entity counter_3bit is
	port(

		-- Input(s)
		clk : in std_logic;
		rst : in std_logic;
		inc : in std_logic;

		-- Output(s)
		count : out std_logic_vector(2 downto 0)
	);
end entity;

architecture  rtl  of counter_3bit is

	signal temp : unsigned(2 downto 0);

begin
```

```
process(clk)
begin
        if rising_edge(clk) then
                if rst = '1' then
                        temp <= (others => '0');
                elsif inc = '1' then
                        temp <= temp + 1;
                end if;
        end if;
end process;
count <= std_logic_vector(temp); -- Converting the unsigned temp to std_logic_vector

end architecture;
```



Εικόνα 3: Αποτέλεσμα του testbench με όνομα «***counter_3bitTb.vhd***» για τον μετρητή 3-bit.

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.*

Η βιβλιοθήκη *hardwiredlib.vhd* θα περιέχει τα παρακάτω components:

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

package hardwiredlib is

    component decoder_generic is
        generic(
            INPUT_WIDTH  : integer; -- Number of input bits
            OUTPUT_WIDTH : integer  -- Number of output bits (2^{input_width})
        );
        port(
            -- Input(s)
            din  : in std_logic_vector(INPUT_WIDTH-1 downto 0);

            -- Output(s)
            dout : out std_logic_vector(OUTPUT_WIDTH-1 downto 0)
        );
    end component;

    component counter_3bit is
        port(
            -- Input(s)
            clk : in std_logic;
            rst : in std_logic;
            inc : in std_logic;

            -- Output(s)
            count : out std_logic_vector(2 downto 0)
        );
    end component;
```

end package 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.*

```
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 rtl of hardwired is


            -- Internal signals for decoded instructions

   signal inst_decode : std_logic_vector(15 downto 0);

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

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


            -- Internal signals for decoded states from the counter (T0-T7)

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

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

        -- Counter signals
    signal counter_out : std_logic_vector(2 downto 0); -- 3-bit output of the state counter
    signal counter_inc : std_logic;                    -- Signal to increment the counter
    signal counter_clr : std_logic;                    -- Signal to clear (reset) the counter

        -- FSM state signals (combinational logic based on T-states and instruction)
    signal FETCH1, FETCH2, FETCH3          : std_logic;
    signal 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          : std_logic;
    signal JMPZN1, JMPZN2                  : std_logic;
    signal JPNZY1, JPNZY2, JPNZY3          : std_logic;
    signal JPNZN1, JPNZN2                  : std_logic;
    signal ADD1, SUB1, INAC1, CLAC1        : std_logic;
    signal AND1, OR1, XOR1, NOT1           : std_logic;

        -- Individual control signals (micro-operations)
    signal ARLOAD, ARINC, PCLOAD, PCINC, DRLOAD, TRLOAD, IRLOAD, RLOAD : std_logic;
    signal ACLOAD, ZLOAD, RD, WR, MEMBUS, BUSMEM : std_logic;
    signal PCBUS, DRBUS, TRBUS, RBUS, ACBUS      : std_logic;
    signal ANDOP, OROP, XOROP, NOTOP       : std_logic;
    signal ACINC, ACZERO, PLUS, MINUS      : std_logic;

        -- Utility signals
    signal Z_not : std_logic;

begin
```

```
        -- Instruction decoder (4 to 16)
inst_decoder : decoder_generic
   generic map(INPUT_WIDTH => 4, OUTPUT_WIDTH => 16)
   port map(din => ir, dout => inst_decode);

        -- State decode (3 to 8)
state_decoder : decoder_generic
   generic map(INPUT_WIDTH => 3, OUTPUT_WIDTH => 8)
   port map(din => counter_out, dout => state_decode);

        -- 3-bit counter
        -- Increments on 'counter_inc' and clears on 'counter_clr'.
-- NOTE: This component uses a SYNCHRONOUS reset.
counter : counter_3bit
   port map(clk => clock, rst => counter_clr, inc => counter_inc, count => counter_out);

-- Assign decoded instruction signals for readability
INOP  <= inst_decode(0);
ILDAC <= inst_decode(1);
ISTAC <= inst_decode(2);
IMVAC <= inst_decode(3);
IMOVR <= inst_decode(4);
IJUMP <= inst_decode(5);
IJMPZ <= inst_decode(6);
IJPNZ <= inst_decode(7);
IADD  <= inst_decode(8);
ISUB  <= inst_decode(9);
IINAC <= inst_decode(10);
ICLAC <= inst_decode(11);
IAND  <= inst_decode(12);
IOR   <= inst_decode(13);
IXOR  <= inst_decode(14);
INOT  <= inst_decode(15);
```

```
-- Assign decoded state signals for readability
T0 <= state_decode(0);
T1 <= state_decode(1);
T2 <= state_decode(2);
T3 <= state_decode(3);
T4 <= state_decode(4);
T5 <= state_decode(5);
T6 <= state_decode(6);
T7 <= state_decode(7);

        -- Invert the Z flag for use in JMPZN and JPNZY instructions
Z_not <= not z;

        -- FETCH CYCLE STATES
-- T0, T1, and T2 are dedicated to the fetch cycle.
FETCH1 <= T0;
FETCH2 <= T1;
FETCH3 <= T2;

        -- INSTRUCTION EXECUTION STATES
-- All instruction execution starts at T3.
        -- Table 1
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 Z_not and T3;

JMPZN2 <= IJMPZ and Z_not and T4;


JPNZY1 <= IJPNZ and Z_not and T3;

JPNZY2 <= IJPNZ and Z_not and T4;

JPNZY3 <= IJPNZ and Z_not 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;


        -- Control Signal Generation from Table 2.
```

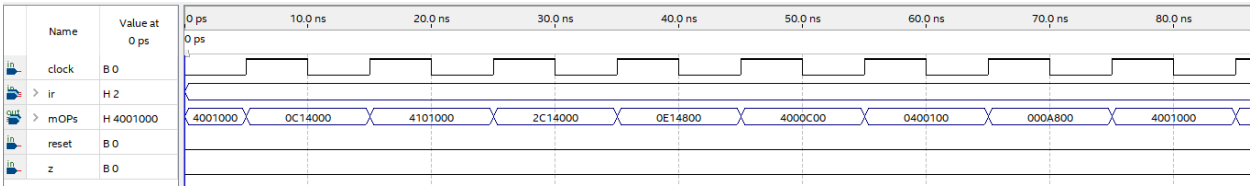
```
        -- Register Control
ARLOAD          <= FETCH1 or FETCH3 or LDAC3 or STAC3;
ARINC  <= LDAC1 or STAC1 or JMPZY1 or JPNZY1;
PCLOAD          <= JUMP3 or JMPZY3 or JPNZY3;
PCINC  <= FETCH2 or LDAC1 or LDAC2 or STAC1 or STAC2 or
      JMPZN1 or JMPZN2 or JPNZN1 or JPNZN2;
DRLOAD          <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or STAC4 or
      JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;
TRLOAD          <= LDAC2 or STAC2 or JUMP2 or JMPZY2 or JPNZY2;
IRLOAD          <= FETCH3;
RLOAD           <= MVAC1;
ACLOAD          <= LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or
      AND1 or OR1 or XOR1 or NOT1;
ZLOAD           <= LDAC5 or MOVR1 or ADD1 or SUB1 or INAC1 or CLAC1 or
      AND1 or OR1 or XOR1 or NOT1;

        -- Memory Control
RD              <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or
                        JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;
WR              <= STAC5;
MEMBUS  <= FETCH2 or LDAC1 or LDAC2 or LDAC4 or STAC1 or STAC2 or
      JUMP1 or JUMP2 or JMPZY1 or JMPZY2 or JPNZY1 or JPNZY2;
BUSMEM          <= STAC5;

        -- Bus Control
PCBUS  <= FETCH1 or FETCH3;
DRBUS <= LDAC2 or LDAC3 or LDAC5 or STAC2 or STAC3 or STAC5 or
      JUMP2 or JUMP3 or JMPZY2 or JMPZY3 or JPNZY2 or JPNZY3;
TRBUS  <= LDAC3 or STAC3 or JUMP3 or JMPZY3 or JPNZY3;
RBUS    <= MOVR1 or ADD1 or SUB1 or AND1 or OR1 or XOR1;
ACBUS <= STAC4 or MVAC1;

        -- ALU operations
```

```
    ANDOP               <= AND1;
    OROP   <= OR1;
    XOROP <= XOR1;
    NOTOP               <= NOT1;
    ACINC  <= INAC1;
    ACZERO              <= CLAC1;
    PLUS    <= ADD1;
    MINUS               <= SUB1;

    -- Increment on all non-terminal states. The counter advances the FSM
    -- through the fetch and execute cycles.
            counter_inc <= FETCH1 or FETCH2 or FETCH3 or
                    LDAC1 or LDAC2 or LDAC3 or LDAC4 or
                    STAC1 or STAC2 or STAC3 or STAC4 or
                    JUMP1 or JUMP2 or
                    JMPZY1 or JMPZY2 or JMPZN1 or
                    JPNZY1 or JPNZY2 or JPNZN1;

    -- Clear (reset) the counter only on the terminal state of each instruction.
    -- This causes the FSM to loop back to T0 (FETCH1) for the next instruction.
            counter_clr <=  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;

    -- The individual control signals are combined into the final 27-bit mOPs word.
    -- The order of concatenation is critical and must match the system design.
    mOPs <= ARLOAD & ARINC & PCLOAD & PCINC & DRLOAD & TRLOAD & IRLOAD & RLOAD &
          ACLOAD & ZLOAD & RD & WR & MEMBUS & BUSMEM & PCBUS & DRBUS &
          TRBUS & RBUS & ACBUS & ANDOP & OROP & XOROP & NOTOP & ACINC &
          ACZERO & PLUS & MINUS;

end architecture;
```

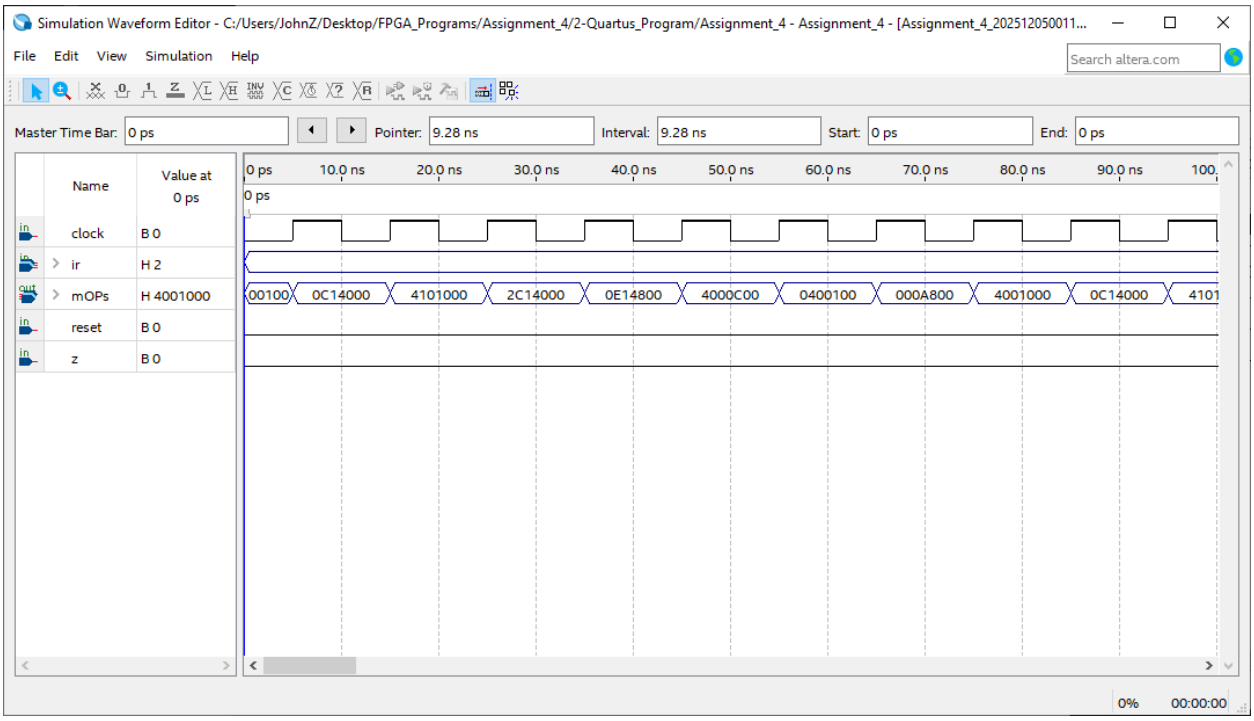
Εξομοίωση της Μονάδας Ελέγχου.

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).



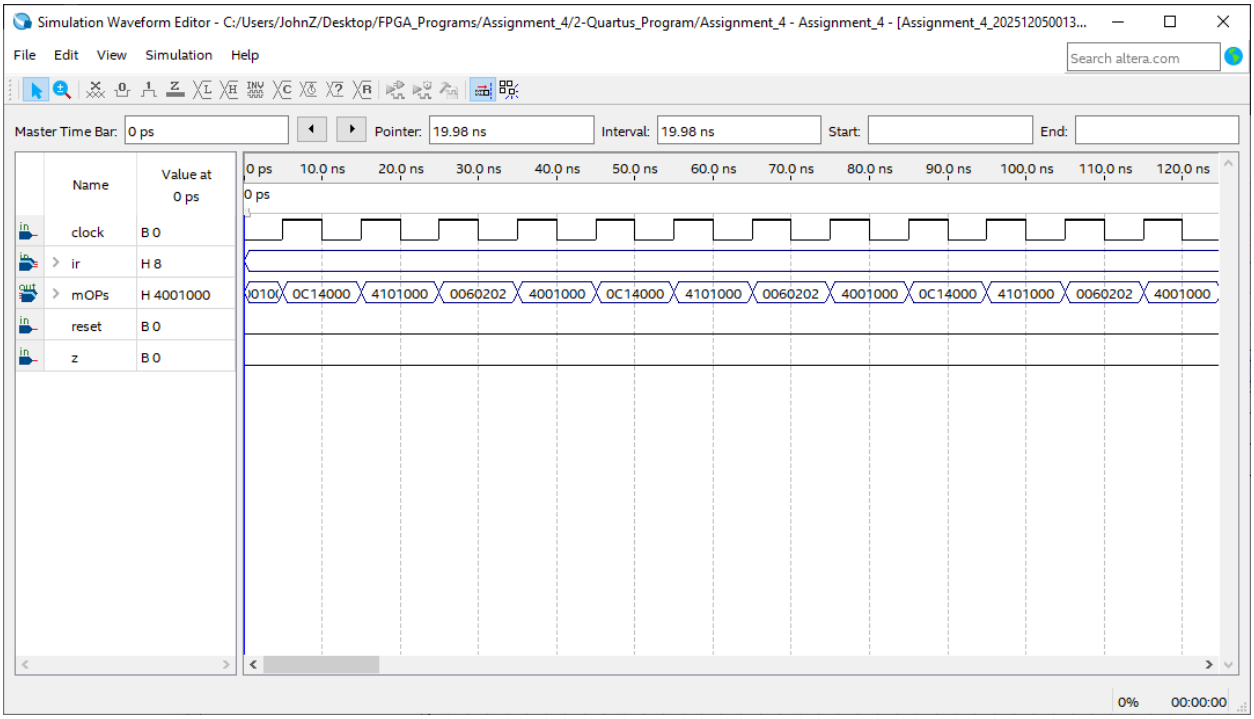
Εικόνα 4: *Κυματομορφές εξομοίωσης εντολής STAC.*



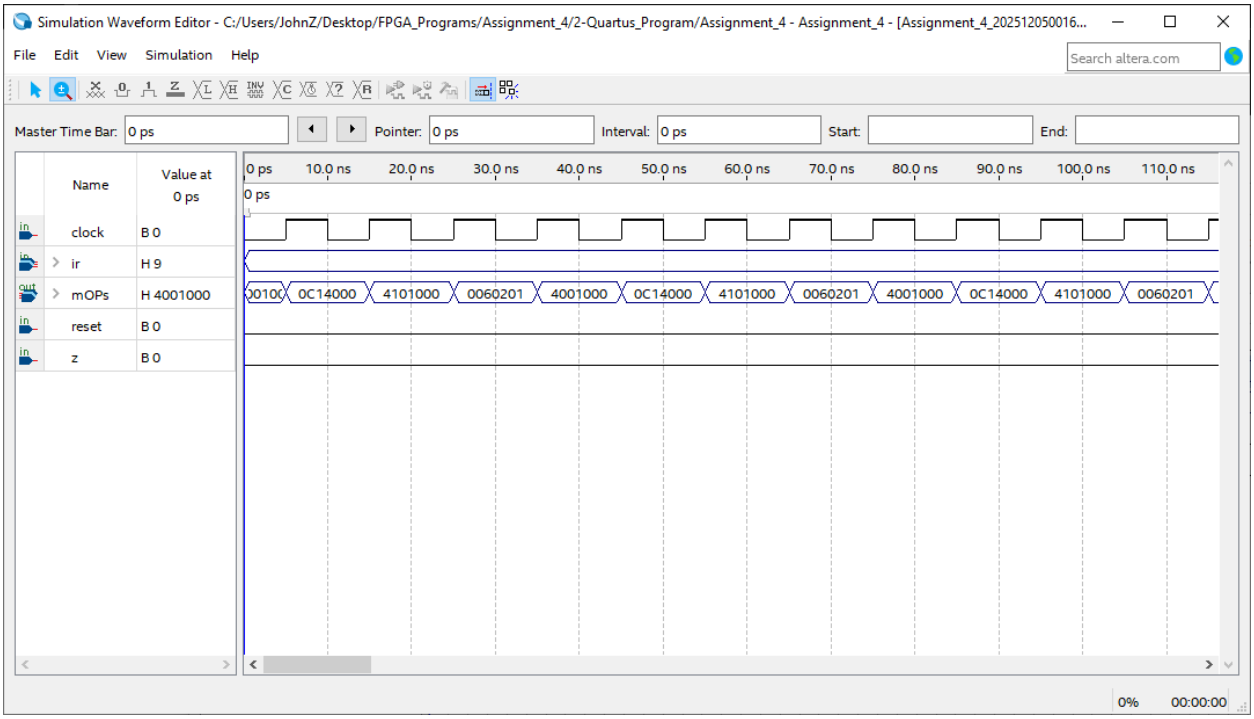
Τοποθετήστε εδώ τις κυματομορφές σας:

Εικόνα 5: *Κυματομορφές εξομοίωσης της μονάδας ελέγχου*

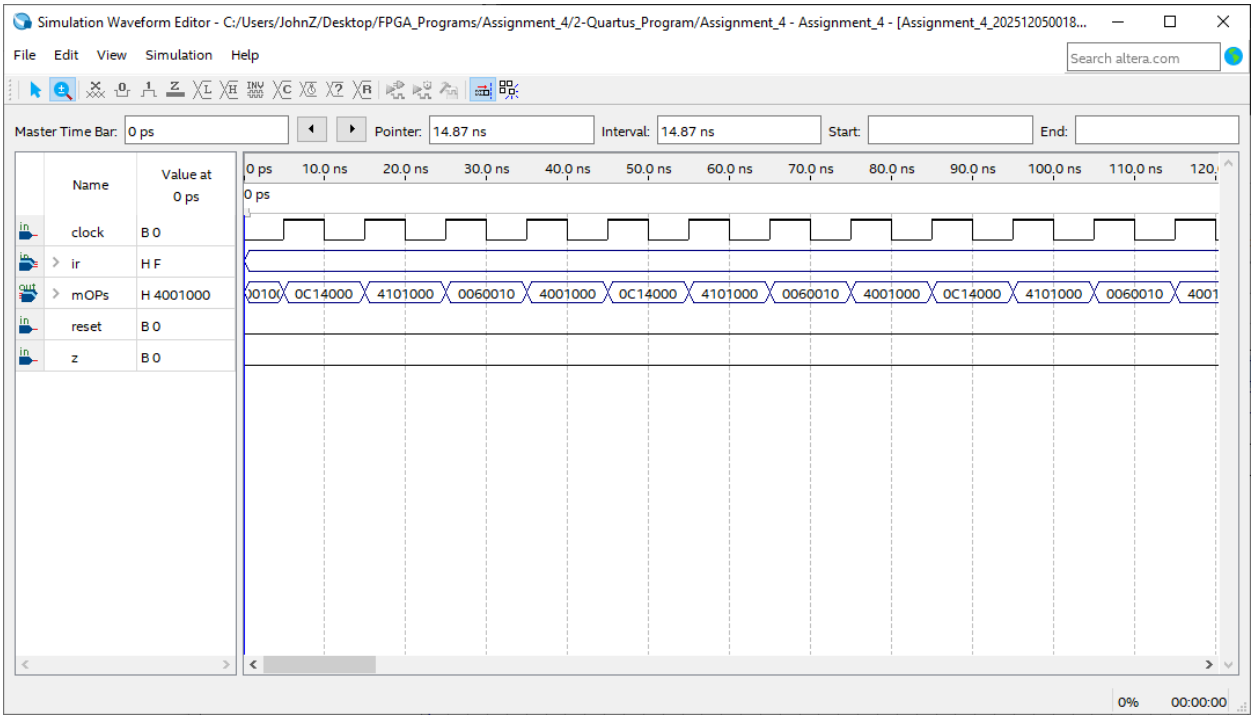
1. ADD $(8)_{16} = (8)_{10}$



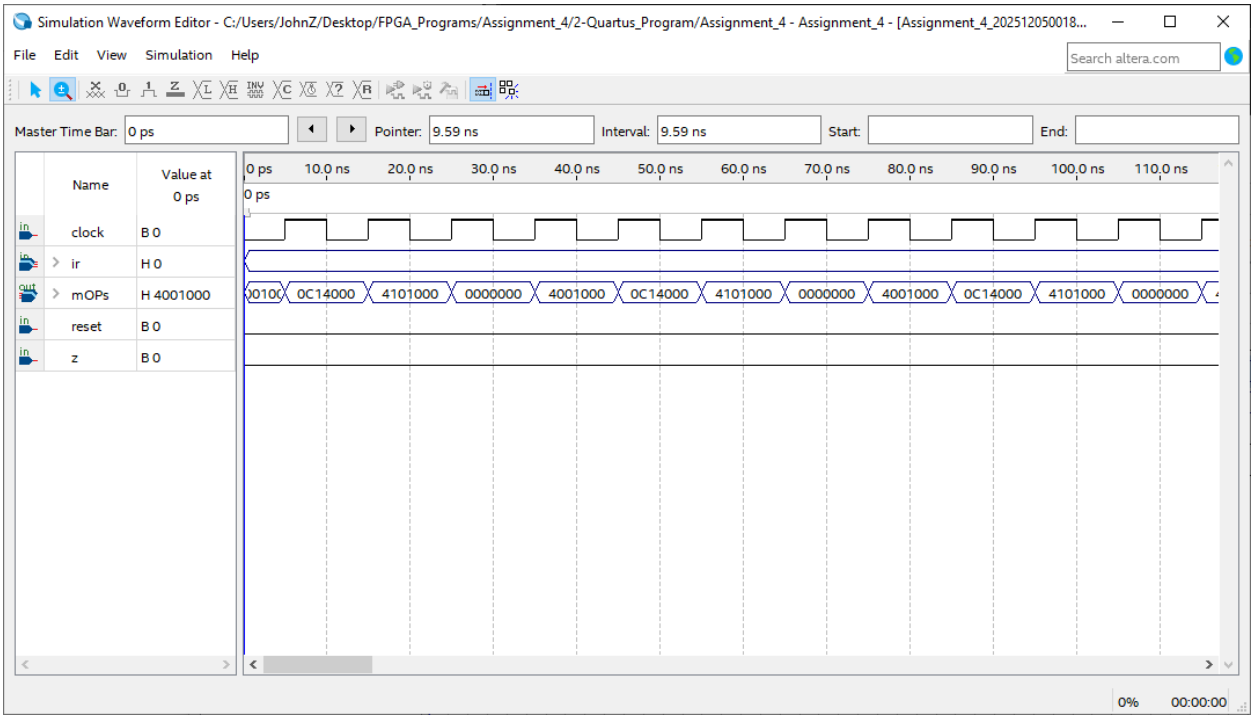
2. SUB $(9)_{16} = (9)_{10}$



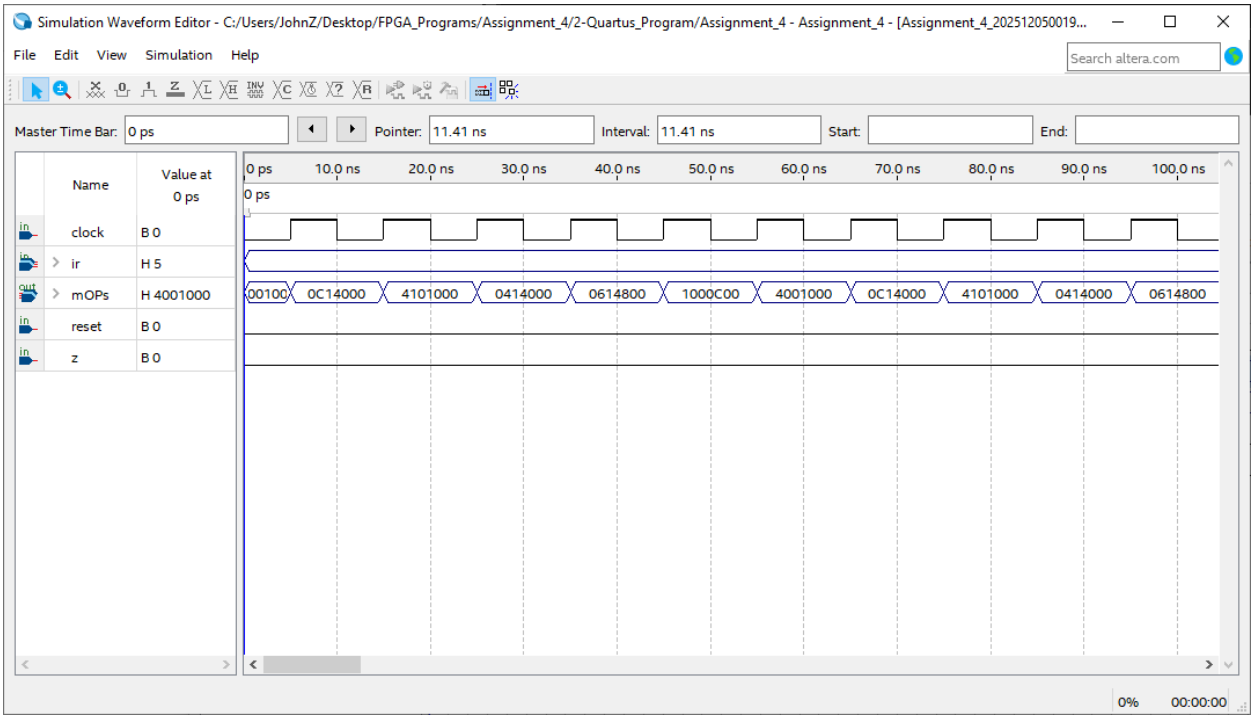
3. NOT $(F)_{16} = (15)_{10}$



4. NOP $(0)_{16} = (0)_{10}$



5. JUMP $(5)_{16} = (5)_{10}$



6. MVOR $(4)_{16} = (4)_{10}$