

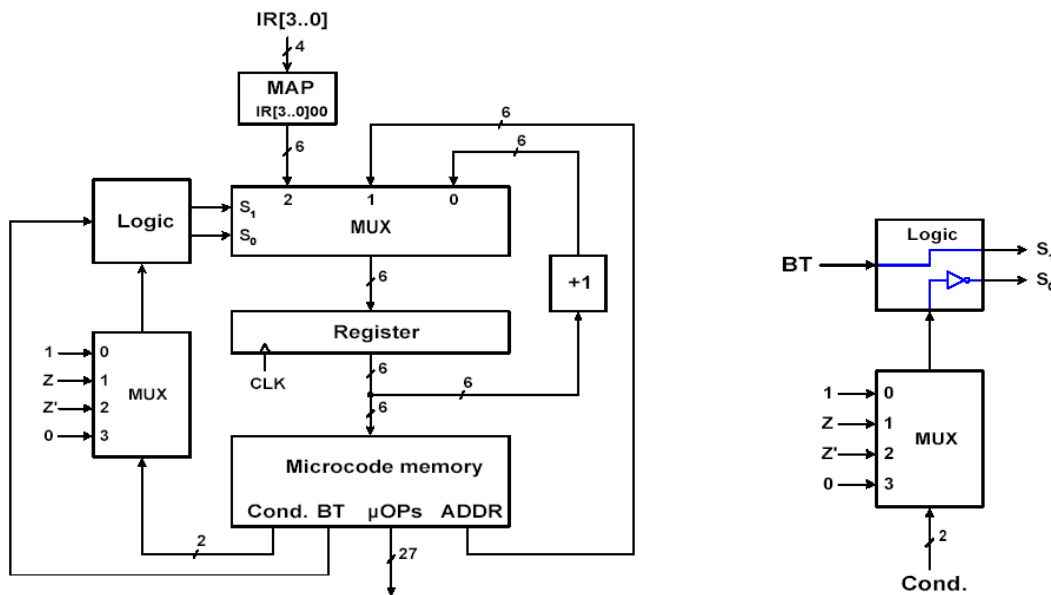
ΜΙΚΡΟΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΜΟΝΑΔΑ ΕΛΕΓΧΟΥ

3

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 μικροπρογραμματιζόμενη (microprogrammed) λογική η οποία θα χρησιμοποιηθεί 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 μικροπρογραμματιζόμενη λογική (microprogrammed logic), η οποία θα υλοποιηθεί με έναν microsequencer (Σχήμα 1).



Σχήμα 1: Λογικό διάγραμμα Μονάδας Ελέγχου και λογική παραγωγής σημάτων ελέγχου S_0, S_1 .

Καταρχάς θα πρέπει να καθοριστεί η δομή μιας μικροεντολής (microinstruction) της ΚΜΕ. Μια μικροεντολή χωρίζεται σε τρία επιμέρους πεδία: SEL, μ OPs και ADDR. Το πεδίο των μικροδιεργασιών (micro-operations) ή μ OPs είναι εκείνο που περιέχει στην ουσία όλα τα σήματα ελέγχου που παρέχονται σε κάθε παλμό στην ΚΜΕ. Έτσι βάσει του διαγράμματος της εσωτερικής οργάνωσης της ΚΜΕ τα σήματα ελέγχου είναι τα ακόλουθα :

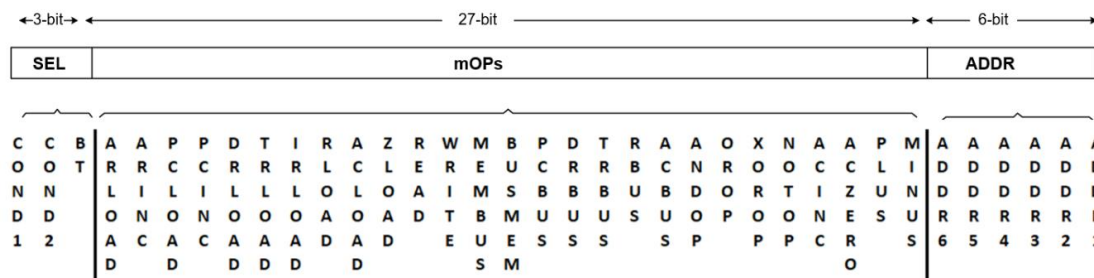
- | | |
|---------------------------------|---|
| ✓ Σήματα καταχώρησης | <i>ARLOAD,PCLOAD,DRLOAD,IRLOAD,TRLOAD,RLOAD,ACLOAD</i> |
| ✓ Σήματα προσαύξησης | <i>ARINC, PCINC</i> |
| ✓ Σήματα ελέγχου της ALU | <i>ALUS 1, ALUS 2,...,ALUS 7</i> |
| ✓ Σήματα που αφορούν τη μνήμη | <i>READ, WRITE</i> |
| ✓ Σήματα ελέγχου των απομονωτών | <i>PCBUS, DRBUS, TRBUS, RBUS, ACBUS, MEMBUS, BUSMEM</i> |

Τα σήματα αυτά μπορούν να ομαδοποιηθούν σε μία ψηφιολέξη, που θα αποτελεί το πεδίο των μικροδιεργασιών (μOPs) σε κάθε μικροεντολή. Στην πράξη όμως θα ακολουθηθεί μια διαφορετική προσέγγιση, όπου αντί των σημάτων ελέγχου της ALU (ALUS[1..7]) θα χρησιμοποιηθούν στο πεδίο των μικροδιεργασιών τα εξής οκτώ σήματα: ANDOP, OROP, XOROP, NOTOP, ACINC, ACZERO, PLUS και MINUS. Τα σήματα αυτά δεν αποτελούν απευθείας σήματα ελέγχου της ΚΜΕ, αλλά μέσω κάποιας λογικής παράγονται από αυτά τα επιθυμητά (σήματα). Κάθε ένα από αυτά αντιστοιχεί σε περισσότερες από μία μικροδιεργασίες.

Έτσι για παράδειγμα ενεργοποίηση του σήματος ADDOP, σημαίνει ότι λαμβάνει χώρα η διεργασία της πρόσθεσης (ADD) μεταξύ δύο αριθμών των 8-bits, η οποία για να υλοποιηθεί θα πρέπει να γίνουν κάποιες μικροδιεργασίες, μεταξύ των οποίων ενεργοποίηση των σημάτων ALUS 1 & 3 της ALU. Αντίστοιχα, η ενεργοποίηση του σήματος MINUS που αναφέρεται στην πράξη της αφαίρεσης, θα σημαίνει παράλληλα ενεργοποίηση των σημάτων ALUS 1, 2 & 4 μέσω κάποιας λογικής που θα παρεμβάλλεται μεταξύ της μονάδας ελέγχου και της ALU. Η ίδια λογική ισχύει και για τα υπόλοιπα σήματα που αντιστοιχούν σε διεργασίες που αφορούν την ALU και τον συσσωρευτή (AC). Έτσι τελικά το πεδίο των μOPs θα έχει εύρος 27-bits.

Θα πρέπει όμως να σημειωθεί ότι η επιλογή των παραπάνω οκτώ (8) σημάτων αντί των ALUS[1..7], δεν είναι η καλύτερη από πλευράς σχεδίασης, αλλά απλά επιλέχθηκε σαν μια διαφορετική προσέγγιση. Σε πιο πολύπλοκες όμως ΚΜΕ όπου τα σήματα ελέγχου θα είναι πολύ περισσότερα, η επιλογή έμμεσων κατά κάποιο τρόπο σημάτων ελέγχου (όπως τα ANDOP, OROP, ..κτλ), αποτρέπει το φαινόμενο το πεδίο των μικροδιεργασιών (μΟΡs) να είναι πολύ μεγάλου εύρους, ανάλογο του πλήθους των σημάτων ελέγχου.

Εκτός όμως από το πεδίο των μικροδιεργασιών, κάθε μικροεντολή περιλαμβάνει και τα πεδία SEL και ADDR, τα οποία χρησιμοποιούνται από τον microsequencer για τη λειτουργία του. Το πεδίο ADDR αποτελεί τη διεύθυνση της μνήμης μικροκώδικα της επόμενης μικροεντολής που θα εκτελεστεί. Για την ακρίβεια είναι μία από τις πιθανές διευθύνσεις της επόμενης μικροεντολής. Το τρίτο και τελευταίο πεδίο μιας μικροεντολής είναι το πεδίο SEL εύρους τριών (3) bits, τα οποία χρησιμοποιούνται σαν σήματα επιλογής διεύθυνσης της επόμενης μικροεντολής που θα εκτελεστεί.



Σχήμα 2: Δομή μιας μικροεντολής (microinstruction).

Ολοκληρώνοντας την περιγραφή της μονάδας ελέγχου κάθε μικροεντολή έχει εύρος 36-bits, από τα οποία τα 27-bits στο πεδίο των μικροδιεργασιών (μOPs) θα αποτελούν τα σήματα ελέγχου της ΚΜΕ. Βάσει των μικροδιεργασιών σε κάθε κατάσταση, προκύπτει το σύνολο των μικροεντολών για τη σχετικά απλή ΚΜΕ, το οποίο φαίνεται στον παρακάτω πίνακα.

SEL				μOPs																																A D D R (dec)
State	A D D R E S S	C O N D 1	C O N D 2	B T	A R R L O A D	A R R I N C	P C L O A D	P C I N C	D R L O A D	T R L O A D	I R L O A D	R L O A D	A C L O A D	Z L O A D	R E A D	W R I T E	M E M B U S	B U S M E M	P C B U S	D R B U S	T R B U S	R B U S	A C B U S	A N D O P	O R O P	X O R O P	N O T O P	A C I N C	A C Z E R O	P L U S	M I N U S					
FETCH 1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2			
FETCH 2	2	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3			
FETCH 3	3	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	X			
NOP 1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1			
LDAC 1	4	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5			
LDAC 2	5	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	6			
LDAC 3	6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	7			
LDAC 4	7	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33			
LDAC 5	33	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1			
STAC 1	8	0	0	0	0	1	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9			
STAC 2	9	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	10			
STAC 3	10	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	11			
STAC 4	11	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	34			
STAC 5	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1			
MVAC	12	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1			
MOVR	16	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1			
JUMP 1	20	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21			
JUMP 2	21	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	22			
JUMP 3	22	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1			
JMPZ 1	24	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41			
JMPZY 1	25	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26			
JMPZY 2	26	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	27			
JMPZY 3	27	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1			
JMPZN 1	41	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	42			
JMPZN 2	42	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1			
JPNZ 1	28	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45			
JPNZY 1	29	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30			
JPNZY 2	30	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	31			
JPNZY 3	31	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1			
JPNZN 1	45	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46			
JPNZN 2	46	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1			
ADD 1	32	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1			
SUB 1	36	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1			
INAC 1	40	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1			
CLAC 1	44	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1			
AND 1	48	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1			
OR 1	52	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1			
XOR 1	56	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1			
NOT 1	60	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1			

Πίνακας 1: Το σύνολο των μικροεντολών για τη σχετικά απλή CPU

Μνήμη μικροκώδικα

Με τη βοήθεια οποιουδήποτε Editor συντάξετε και αποθηκεύστε το πρόγραμμα 1 που ακολουθεί με όνομα "rs_microcode.mif". Το αρχείο αυτό θα χρησιμοποιηθεί για την αρχικοποίηση της μνήμης μικροκώδικα (μεταβλητή lpm_file) που θα δημιουργήσουμε στο επόμενο βήμα και περιέχει το σύνολο των μικροεντολών, εύρους 36-bits η κάθε μία, σε 39 θέσεις της μνήμης (αντίστοιχες των 39 καταστάσεων) μικροκώδικα.

```

0: 1100000000000000000000000000000000000001; -- NOP1
1: 00010000000000000010000000000000000010; -- FETCH1
2: 0000001100000101000000000000000000011; -- FETCH2
3: 00110000010000000100000000000000000000; -- FETCH3
4: 000010110000010100000000000000000000101; -- LDAC1
5: 00000011100001010010000000000000000110; -- LDAC2
6: 00010000000000000001100000000000000111; -- LDAC3
7: 1100000100000101000000000000000100001; -- LDAC4
8: 000010110000010100000000000000000001001; -- STAC1
9: 0000001110000101001000000000000001010; -- STAC2
10: 0001000000000000001100000000000001011; -- STAC3
11: 110000010000000000000100000000100010; -- STAC4
12: 11000000010000000001000000000000000001; -- MVAC1
16: 110000000001100000001000000000000000001; -- MOV1R
20: 00001001000001010000000000000000010101; -- JUMP1
21: 00000001100001010010000000000000010110; -- JUMP2
22: 11000100000000000001100000000000000001; -- JUMP3
24: 010000000000000000000000000000000101001; -- JMPZ1
25: 00001001000001010000000000000000011010; -- JMPZY1
26: 00000001100001010010000000000000011011; -- JMPZY2
27: 110001000000000000001100000000000000001; -- JMPZY3
28: 100000000000000000000000000000000101101; -- JPNZ1
29: 00001001000001010000000000000000011110; -- JPNZY1
30: 00000001100001010010000000000000011111; -- JPNZY2
31: 11000100000000000001100000000000000001; -- JPNZY3
32: 110000000001100000001000000010000001; -- ADD1
33: 11000000000110000100000000000000000001; -- LDAC5
34: 110000000000000101010000000000000000001; -- STAC5
36: 110000000001100000001000000001000001; -- SUB1

```

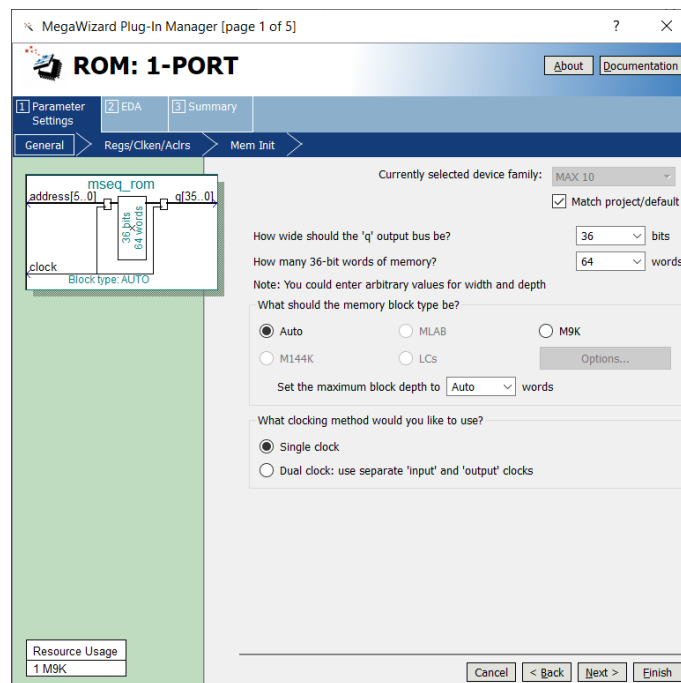
```

40: 110000000001100000000000001000000001; -- INAC1
41: 00000010000000000000000000000000101010; -- JMPZN1
42: 11000010000000000000000000000000000001; -- JMPZN2
44: 11000000000110000000000000000000000001; -- CLAC1
45: 000000100000000000000000000000000000101110; -- JPNZN1
46: 1100001000000000000000000000000000000001; -- JPNZN2
48: 11000000000110000000101000000000000001; -- AND1
52: 11000000000110000000100100000000000001; -- OR1
56: 11000000000110000000100010000000000001; -- XOR1
60: 11000000000110000000000000100000000001; -- NOT1
END;

```

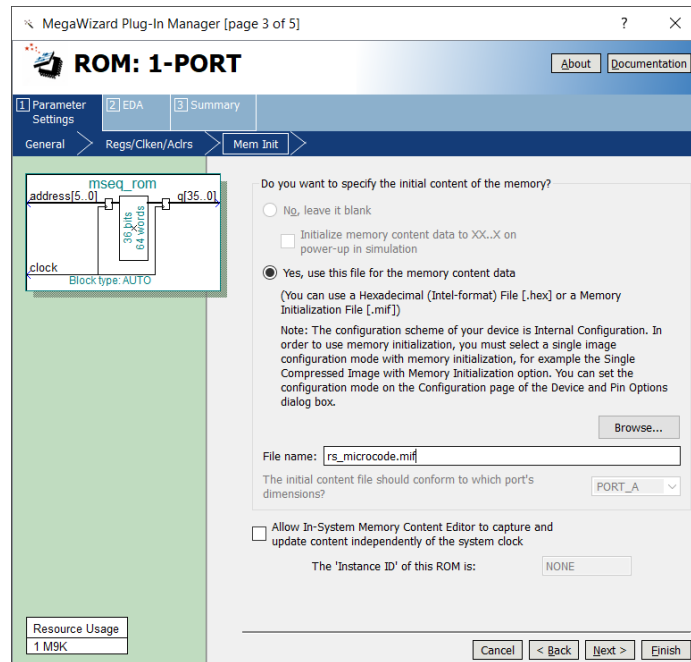
Πρόγραμμα 1: Περιεχόμενα μνήμης μικροκώδικα

Στη συνέχεια "καλέστε" τον MegaWizard Manager μέσω του IP Catalog (πάνω δεξιά περιοχή στο Quartus) αφού κάνετε διπλό click στην επιλογή Library|Basic Functions|On Chip Memory|ROM: 1-PORT και επιλέξετε στο παράθυρο που εμφανίζεται το όνομα (προτείνεται το mseq_rom.vhd), το path και το τύπο του αρχείου(VHDL) ξεκινάει η διαδικασία ορισμού των παραμέτρων της μνήμης:



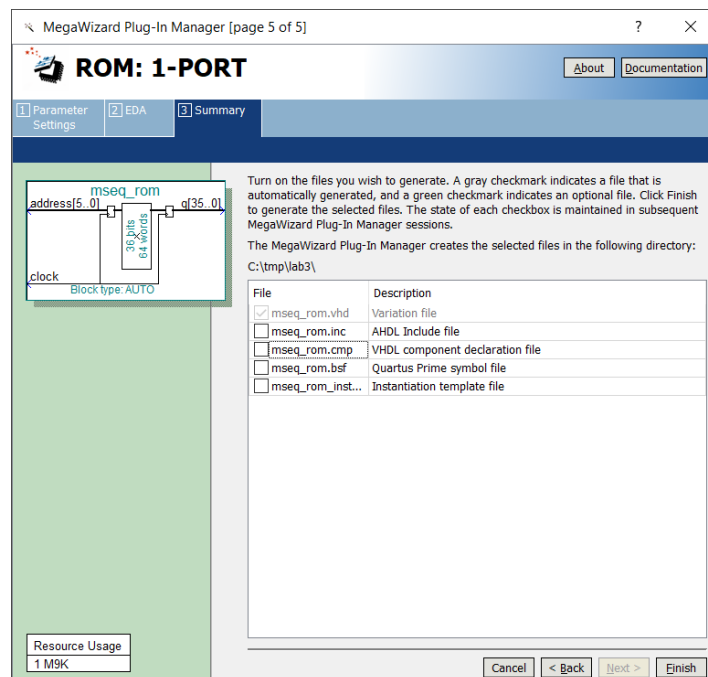
Εικόνα 1

Στο παράθυρο που εμφανίζεται εφαρμόζουμε τις ρυθμίσεις από την Εικόνα 1, πατάμε next και προσπερνάμε το επόμενο παράθυρο (page 2 of 5).



Εικόνα 2

Εφαρμόζουμε τις ρυθμίσεις από την Εικόνα 2, πατάμε next και προσπερνάμε το επόμενο παράθυρο (page 4 of 5).



Εικόνα 3

Τέλος καταλήγουμε στο τελευταίο παράθυρο (Εικόνα 3) στο οποίο επιλέγουμε την δημιουργία του αρχείου VHDL, το οποίο είναι και η default επιλογή, και πατάμε Finish για να ολοκληρωθεί. Κατά το τέλος της διαδικασίας το Quartus μας ρωτάει αν επιθυμούμε να προσθέσουμε το Quartus IP File στο

project. Εδώ επιλέγουμε όχι καθώς θα χρειαστούμε μόνο το αρχείο κώδικα της μνήμης το οποίο και θα προσθέσουμε αργότερα στο project.

Μονάδα Ελέγχου.

Έχοντας ολοκληρώσει τη συγγραφή του κώδικα για τα επιμέρους στοιχεία που συνθέτουν την μονάδα ελέγχου, με δεδομένο ότι έχουμε στη διάθεση μας τους κώδικες για τον καταχωρητή n-bits και του πολυπλέκτη 4 σε 1 από προηγούμενες ασκήσεις και αφού όλα συγκεντρωθούν σε μία βιβλιοθήκη, μπορεί πλέον να γραφεί το συνολικό πρόγραμμα περιγραφής της μονάδας ελέγχου. Σημειώνεται εδώ ότι δεδομένου ότι κύκλωμα παραγωγής των σημάτων ελέγχου S_1 και S_0 (σχήμα 1) είναι εξαιρετικά απλό δεν είναι απαραίτητη η συγγραφή ξεχωριστού στοιχείου για αυτό.

Γράψτε τον κώδικα για τη βιβλιοθήκη (package), με το όνομα mseqlib, 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 mseqlib is
```

```
    component reg6
```

```
    port (
```

```
        clk, reset : in std_logic;
```

```
        d : in std_logic_vector(5 downto 0);
```

```
        q : out std_logic_vector(5 downto 0)
```

```
    );
```

```
end component;
```

```
    component mux4_6bit
```

```
    port (
```

```
        sel : in std_logic_vector(1 downto 0);
```

```
        in0, in1, in2, in3 : in std_logic_vector(5 downto 0);
```

```
        output : out std_logic_vector(5 downto 0)
```

```
    );
```

```
end component;
```

```
component plus_one
```

```
port (  
    input_addr : in std_logic_vector(5 downto 0);  
    output_addr : out std_logic_vector(5 downto 0)  
);
```

```
end component;
```

```
component map_logic
```

```
port (  
    ir_opcode : in std_logic_vector(3 downto 0);  
    map_out : out std_logic_vector(5 downto 0)  
);
```

```
end component;
```

```
component cond_mux
```

```
port (  
    sel : in std_logic_vector(1 downto 0);  
    in0, in1, in2, in3 : in std_logic;  
    cond_out : out std_logic  
);
```

```
end component;
```

```
component mseq_rom
```

```
port (  
    address : in std_logic_vector(5 downto 0);  
    clock : in std_logic;  
    q : out std_logic_vector(35 downto 0)
```



```
);
```

```
end component;
```

```
end mseqlib;
```

Πρόγραμμα 2: βιβλιοθήκη στοιχείων για την μονάδα ελέγχου.

Με βάση το σκελετό που ακολουθεί (πρόγραμμα 3) γράψτε τον κώδικα περιγραφής για της μονάδας ελέγχου, δηλαδή του microsequencer, έτσι όπως διαμορφώνεται από τα επιμέρους στοιχεία και το σχήμα 1. Τα σήματα που θα δέχεται σαν είσοδο ο microsequencer 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 λαμβάνονται τόσο το πεδίο των μικροδιεργασιών (μΟΡs) που περιέχει τα σήματα ελέγχου της CPU, όσο και το σήμα code που αντιστοιχεί στην κάθε μικροεντολή εύρους 36-bits. Ο λόγος που λαμβάνεται το σήμα code, είναι για να διακρίνεται πιο εύκολα στην εξομοίωση της CPU ποια μικροεντολή εκτελείται σε κάθε παλμό.

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.mseqlib.all;
```

```
entity mseq is
```

```
    port( ir      : in std_logic_vector(3 downto 0);
```

```
          clock, reset : in std_logic;
```

```
          z      : in std_logic;
```

```
          code    : out std_logic_vector(35 downto 0);
```

```
          mOPs    : out std_logic_vector(26 downto 0));
```

```
end mseq;
```

```
architecture arc of mseq is
```

```
signal cur_address : std_logic_vector(5 downto 0);
signal nxt_address : std_logic_vector(5 downto 0);
signal inc_address : std_logic_vector(5 downto 0);
signal map_address : std_logic_vector(5 downto 0);
```

```
-- rom signals
```

```
signal microinstruction : std_logic_vector(35 downto 0);
signal cond             : std_logic_vector(1 downto 0);
signal BT               : std_logic;
signal ADDR             : std_logic_vector(5 downto 0);
```

```
signal S1, S0           : std_logic;
signal S_vector          : std_logic_vector(1 downto 0);
```

```
signal mux_cond_out     : std_logic;
signal not_Z             : std_logic;
```

```
begin
```

```
-- Instruction splitting
```

```
code <= microinstruction;
```

```
cond <= microinstruction(35 downto 34);
```

```
BT  <= microinstruction(33);
```

```
mOPs <= microinstruction(32 downto 6);
```

```
ADDR <= microinstruction(5 downto 0);
```

```
not_Z <= not z;
```

-- Control Logic

S1 <= BT;

S0 <= not mux_cond_out;

S_vector <= S1 & S0;

block_0: mseq_rom port map (

 address => cur_address,

 clock => clock,

 q => microinstruction

);

block_1: reg6 port map (

 clk => clock,

 reset => reset,

 d => nxt_address,

 q => cur_address

);

-- Incrementer

block_2: plus_one port map (

 input_addr => cur_address,

 output_addr => inc_address

);

block_3: map_logic port map (

 ir_opcode => ir,

 map_out => map_address

```
);
```

```
block_4: cond_mux port map (
```

```
    sel    => cond,
```

```
    in0    => '1',
```

```
    in1    => z,
```

```
    in2    => not_Z,
```

```
    in3    => '0',
```

```
    cond_out => mux_cond_out
```

```
);
```

```
block_5: mux4_6bit port map (
```

```
    sel    => S_vector,
```

```
    in0    => inc_address,
```

```
    in1    => ADDR,
```

```
    in2    => map_address,
```

```
    in3    => (others => '0'),
```

```
    output => nxt_address
```

```
);
```

```
end arc;
```

Πρόγραμμα 3: Αριθμητική & Λογική Μονάδα.

Εξομοίωση της Μονάδας Ελέγχου.

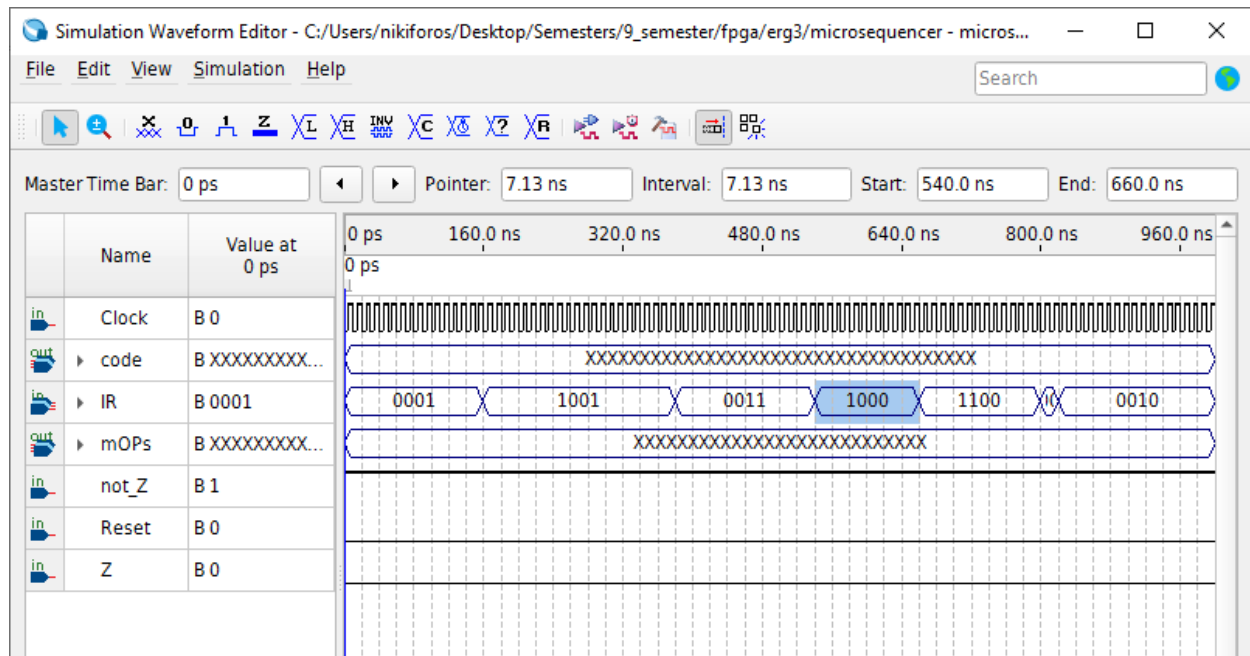
Το επόμενο στάδιο περιλαμβάνει την εξομοίωση της μονάδας ελέγχου με τον Waveform Editor με σκοπό τον έλεγχο της λειτουργίας της. Με οδηγό τις προηγούμενες ασκήσεις, δημιουργήστε ένα καινούργιο project και εξομοιώστε τη λειτουργία της μονάδας ελέγχου με τη βοήθεια του Waveform Editor για έξι (6) εντολές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

ΠΡΟΣΟΧΗ: Μέσω του μενού **Assignments/Device/Device and Pin Options../Configuration** ρυθμίστε δώστε στο **Configuration mode** την τιμή

Single Uncompressed Image with Memory Initialization (256Kbits UFM)
για να μην έχετε προβλήματα – λάθη κατά το compilation του project.

Τοποθετήστε εδώ τις κυματομορφές σας:

Όπως σας ανάφερα και σε μήνυμα, δεν μπόρεσα να χρησιμοποιήσω την έκδοση του quartus που χρησιμοποιούμε (20.1.1) ούτε σε windows, ούτε σε linux. Για αυτό το λόγο κατέβασα την τρέχουσα έκδοση (25.1) η οποία όμως δεν έχει το modelsim. Δεν μπόρεσα να τρέξω το simulation λόγω της απουσίας του modelsim (εκτυπώνεται το μήνυμα modelsim executable not found). Δοκίμασα να εγκαταστήσω το questa αλλά μάλλον χρειάζεται κάποιο license για να δουλέψει.



Εικόνα 4: Κυματομορφές εξομοίωσης της μονάδας ελέγχου