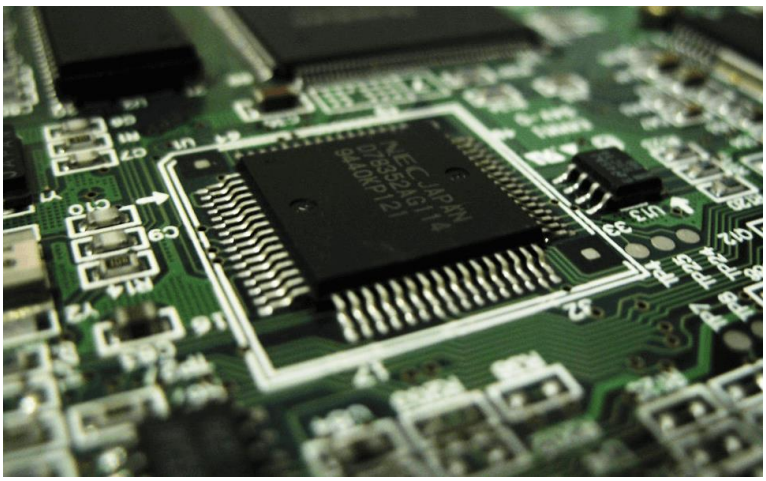
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

ΕΞΑΜΗΝΙΑΙΕΣ ΕΡΓΑΣΙΕΣ

ΓΙΑ ΤΟ ΜΑΘΗΜΑ
ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΕΣ

ΕΚΠΟΝΗΘΗΚΕ ΑΠΟ ΤΗΝ
ΒΑΣΙΛΕΙΟΥ ΟΛΓΑ, 01691, 3ο ΕΤΟΣ

ΔΙΔΑΣΚΩΝ ΚΑΘΗΓΗΤΗΣ
ΚΑΛΟΒΡΕΚΤΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ



2020-2021

ΠΕΡΙΕΧΟΜΕΝΑ

Εισαγωγή

Το έγγραφο περιέχει την εκπόνηση των εργασιών εργαστηρίου και θεωρίας του μαθήματος Μικροεπεξεργαστές. Συγκεκριμένα, περιλαμβάνει την επίλυση της πρώτης εργασίας της κατηγορίας Α, καθώς και τον κώδικα και την περιγραφή αυτού για την δεύτερη εργασία της ίδιας κατηγορίας.

Κατηγορία Α

Ακολουθούν οι υλοποιήσεις των δύο πρώτων, υποχρεωτικών εργασιών.

Εργασία 1Α

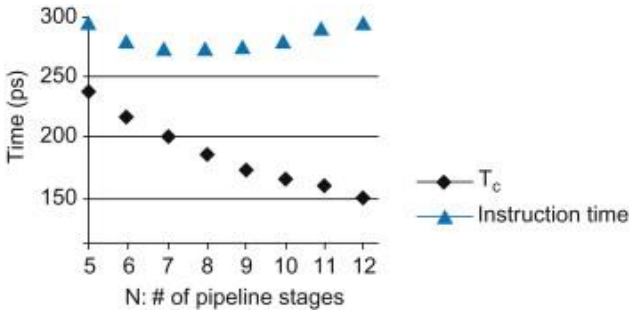
Σύμφωνα με την επιστήμη των υπολογιστών, η τεχνική διοχέτευσης (pipeline) επιτρέπει την ταυτόχρονη επεξεργασία πολλαπλών εντολών[1]. Καθώς ο υπολογιστής επεξεργάζεται μία εντολή, μπορεί ταυτόχρονα να βρίσκεται στο στάδιο επεξεργασία και άλλων εντολών. Χωρίς την τεχνική της διοχέτευσης, δεν θα μπορούσαμε να έχουμε καν πρόσβαση στις υπόλοιπες εντολές πριν τελειώσει η επεξεργασία της πρώτης[1]. Ο χρόνος κάθε σταδίου οφείλει να είναι παρόμοιος και συχνά απαιτείται buffering μεταξύ σταδίων λόγω διαφορών μεταξύ του χρόνου εκτέλεσής τους[2]. Στην κλασσική τεχνική διοχέτευσης RISC 5 σταδίων, τα στάδια είναι τα εξής[3]:

1. *Instruction Fetch (IF)* – Προσκόμιση εντολής: Η εντολή είναι αποθηκευμένη στη θέση μνήμης με διεύθυνση το περιεχόμενο του μετρητή προγράμματος (Program Counter – PC), και προσκομίζεται στον καταχωρητή εντολών. Κατόπιν, ο μετρητής προγράμματος γίνεται PC = PC + 1 για να προχωρήσει στην επόμενη εντολή.
2. *Instruction Decode (ID)* – Αποκωδικοποίηση εντολών και ανάγνωση καταχωρητών: Πραγματοποιείται ανάγνωση της εντολής από τον καταχωρητή εντολών και αποκωδικοποίηση των τελεστών σύμφωνα με την πράξη της εντολής.
3. *Execute (EX)* – Εκτέλεση εντολών: Όταν η εντολή είναι μία εντολή εκτέλεσης αριθμητικής ή λογικής πράξης, η εν λόγω πράξη εκτελείται στην αριθμητική λογική μονάδα (Arithmetic Logic Unit - ALU). Όταν η εντολή είναι μία εντολή προσπέλασης της μνήμης με σχετικό τρόπο διευθυνσιοδότησης, υπολογίζεται η διεύθυνση προσπέλασης της μνήμης. Όταν η εντολή είναι μία εντολή διακλάδωσης, ελέγχεται η ισχύς της συνθήκης και αν ισχύει, αποθηκεύεται η διεύθυνση διακλάδωσης στον PC.
4. *Memory Access (MEM)* – Προσπέλαση μνήμης: Όταν η εντολή είναι εντολή προσπέλασης μνήμης, τα δεδομένα διαβάζονται από τη μνήμη ή αποθηκεύονται στην μνήμη, και το αποτέλεσμα της ALU αποθηκεύεται απευθείας στο στάδιο αυτό.
5. *Write Back (WB)* – Αποθήκευση αποτελεσμάτων: Το αποτέλεσμα της λειτουργίας γράφεται σε κάποιον καταχωρητή γενικού σκοπού (General Purpose Register - GPR).

Μία τεχνική διοχέτευσης μεγαλύτερης των 5 σταδίων, είναι το pipeline 6 σταδίων. Τα στάδια της τεχνικής αυτής είναι τα ίδια με αυτά του pipeline 5 σταδίων, εκτός από το στάδιο αποκωδικοποίησης εντολών και ανάγνωσης καταχωρητών, που διασπάται σε δύο στάδια, αυτό της αποκωδικοποίησης εντολών (Instruction Decode) και αυτό της ανάγνωσης καταχωρητών (Register Read)[5]. Κατά την ανάγνωση καταχωρητών

πραγματοποιείται ο υπολογισμός διεύθυνσης του καταχωρητή ανάγνωσης και η αποθήκευσή της σε έναν καταχωρητή γενικού σκοπού (GPR). Κατόπιν, τα δεδομένα του καταχωρητή γενικού σκοπού διαβάζονται και προωθούνται στο επόμενο στάδιο.

Ο ευκολότερος τρόπος να μειωθεί ο χρόνος ενός κύκλου ρολογιού είναι να διασπαστεί η διοχέτευση σε περισσότερα στάδια. Όσα περισσότερα στάδια έχει μία διοχέτευση, τόσο λιγότερη λογική διαθέτει, με αποτέλεσμα να εκτελείται γρηγορότερα[4]. Παρόλα αυτά, όπως φαίνεται στην παρακάτω εικόνα, όσο αυξάνονται τα στάδια της διοχέτευσης και μειώνεται ο χρόνος ρολογιού, παρατηρείται σταδιακή αύξηση του χρόνου εκτέλεσης της εντολής. Επιπρόσθετα, τα επιπλέον στάδια απαιτούν περισσότερους καταχωρητές και υλικό (hardware) με σκοπό να αντιμετωπιστούν οι εξαρτήσεις (hazards), οι οποίες λόγω των παραπάνω σταδίων, αυξάνονται. Υπάρχουν τριών ειδών εξαρτήσεις, οι δομικές εξαρτήσεις (Structural/Resourse Conflicts), οι εξαρτήσεις από δεδομένα (Data Dependencies) και οι διαδικαστικές εξαρτήσεις (Control/Procedural Dependencies)[5].



Εικόνα 1. Συνάρτηση αριθμού σταδίων διοχέτευσης - χρόνου ρολογιού

Ακολουθεί η εκτέλεση εντολών (**LDI R1, 0(R11)** και **ADDI R2, R1, 10**) με διοχέτευση 5 σταδίων και 6 σταδίων:

1. Τεχνική διοχέτευσης 5 σταδίων

LDI R1, 0(R(11))	IF	ID	EX	MEM	WB			
ADDI R2, R1, 10		IF	STALL	STALL	ID	EX	MEM	WB

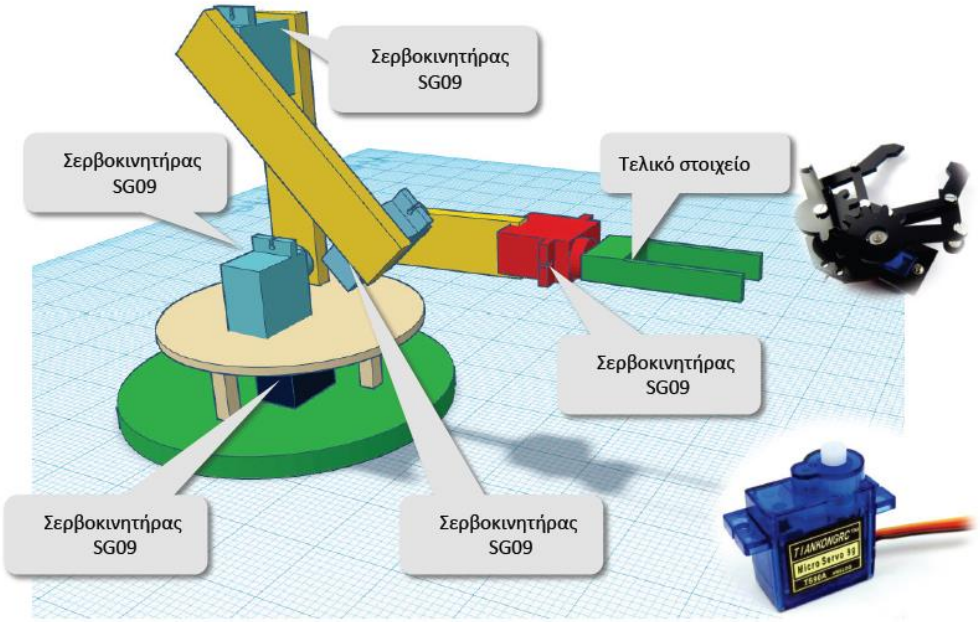
2. Τεχνική διοχέτευσης 6 σταδίων:

LDI R1, 0(R(11))	IF	ID	RR	EX	MEM	WB			
ADDI R2, R1, 10		IF	ID	STALL	STALL	RR	EX	MEM	WB

Παρατηρώ ότι η διοχέτευση 5 σταδίων απαιτεί 8 κύκλους ρολογιού, λόγω καθυστερήσεων που απαιτούνται για να μπορέσει η εντολή **ADDI** να διαβάσει το περιεχόμενο του καταχωρητή **R1** και να εκτελέσει την πράξη. Το ίδιο συμβαίνει και στη διοχέτευση 6 σταδίων, μόνο που η διάσπαση του 2ου σταδίου σε ID και RR φέρει ως αποτέλεσμα την ολοκλήρωση της επεξεργασίας μετά από 9, και όχι 8, κύκλους ρολογιού. Επομένως, παρόλο που τα στάδια είναι περισσότερα και ο χρόνος επεξεργασίας του κάθε σταδίου είναι μικρότερος από αυτόν στη διοχέτευση 5 σταδίων, ο αριθμός των κύκλων ρολογιού, και κατ' επέκταση ο χρόνος εκτέλεσης του κώδικα, είναι μεγαλύτερος.

Εργασία 2Α

Η διαμόρφωση εύρους παλμών (Pulse Width Modulation – PWM) είναι μία τεχνική διαμόρφωσης που παράγει παλμούς μεταβλητού πλάτους για την αναπαράσταση του πλάτους ενός αναλογικού σήματος εισόδου[6]. Η περιοδική αυτή τετραγωνική κυματομορφή έχει δύο τμήματα: το τμήμα ON στο οποίο η κυματομορφή έχει την μέγιστη τιμή της (high signal) και το τμήμα OFF στο οποίο έχει την τιμή μηδέν (low signal)[7]. Η τεχνική PWM χρησιμοποιείται και για την κίνηση σερβοκινητήρων. Συγκεκριμένα, η παραγωγή PWM σημάτων προσδιορίζει τη θέση των σερβοκινητήρων[8]. Στην εργασία αυτή, το ελάχιστο όριο λογικής κατάστασης του παλμού είναι 0.5ms, που αντιστοιχεί σε 0^o, η ενδιάμεση κατάσταση είναι 1.5ms, δηλαδή 90^o, και το μέγιστο όριο είναι 2.5ms, που αντιστοιχεί σε 180^o.



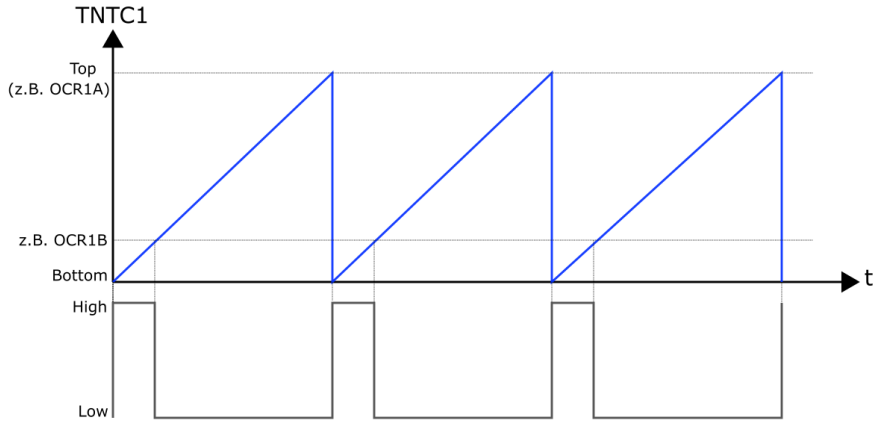
Εικόνα 2. Ένας ρομποτικός μηχανισμός 4 DOF με 5 σερβοκινητήρες

Για την δημιουργία κώδικα ελέγχου του παραπάνω ρομποτικού μηχανισμού, θα πρέπει να προσδιορίσω το duty cycle , δηλαδή το άνω και κάτω όριο κύκλου εργασίας, με βάση τον παρακάτω τύπο, όπου $\mathbf{T_{on}}$: χρόνος για high signal (ON), $\mathbf{T_{off}}$: χρόνος για low signal (OFF) και $\mathbf{T_{total} = T_{on} + T_{off}}$

$$D = \frac{T_{on}}{(T_{on} + T_{off})} = \frac{T_{on}}{T_{total}}$$

Αρχικοποίηση

Αρχικά, όμως, πρέπει να ορίσω τη σωστή κυματομορφή (Waveform Generation Mode - WGM). Για την κίνηση σερβοκινητήρων προτιμώ τη μέθοδο Fast PWM, όπου ο timer αυξάνεται κι έπειτα επαναφέρεται στο μηδέν, όταν φτάσει την μέγιστη απαιτούμενη τιμή που έχω θέσει[9]. Για να ορίσω τη σωστή μέγιστη τιμή, διαιρώ την περίοδο PWM με την περίοδο του ρολογιού του ATmega328, που ισούται με 1μs. Γνωρίζω ότι η περίοδος PWM $\mathbf{T = T_{total}}$ = 20ms, σύμφωνα με την εκφώνηση. Συνεπώς, η μέγιστη τιμή TOP = 20000. Σύμφωνα με το εγχειρίδιο του ATmega328[10], μπορώ να χρησιμοποιήσω τον καταχωρητή ICR1 για να δηλώσω την περίοδο και να καθορίσω την τιμή TOP.



Στη συνέχεια, πρέπει να επιλέξω μέθοδο λειτουργίας, που θα καθορίσει τη συμπεριφορά του Timer/Counter και τα Pins σύγκρισης εξόδου. Για την συγκεκριμένη περίπτωση, χρησιμοποιώ ένα 16-bit Timer/Counter unit, τον TCNT1, επειδή επιτρέπει την ακριβή διαχείριση γεγονότων, την παραγωγή κυμάτων και την μέτρηση χρονικού σήματος. Ο μετρητής TCNT1 περιέχει τον καταχωρητή σύγκρισης OCR1A, τον καταχωρητή καταγραφής εισόδου ICR1, και καταχωρητές ελέγχου TCCR1A και TCCR1B[10].

Για να επιλέξω τη μέθοδο non-inverted, κατά την οποία το σήμα PWM ξεκινά με παλμό και αργότερα γίνεται μηδέν, ενεργοποιώ τον καταχωρητή OCR1A, ορίζοντας COM1A1 = 1 και COM1A0 = 0 (το δεύτερο bit δεν χρειάζεται να οριστεί γιατί είναι ήδη ίσο με μηδέν). Στα high byte και low byte του καταχωρητή αποθηκεύω τα high και low bytes της αρχικής τιμής του duty cycle (0.5ms), αντίστοιχα.

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM1[3:0] = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation); for all other WGM1 settings, normal port operation, OC1A/OC1B disconnected
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

Εικόνα 3. Πίνακας bits του καταχωρητή σύγκρισης OCR1A

Για να επιλέξω την τεχνική Fast PWM με μέγιστη τιμή TOP = ICR1 (Mode 14), ορίζω ίσα με την μονάδα τα bits WGM13 και WGM12 του καταχωρητή ελέγχου TCCR1B και το bit WGM11 του καταχωρητή TCCR1A. Παράλληλα, θέτω το CS10 bit του TCCR1B ίσο με τη μονάδα, για να επιλέξω τη λειτουργία no prescaling, αφού ο timer είναι 16-bit και η τιμή TOP δεν ξεπερνά τα 16 bits (0xFFFF).

Bit No	7	6	5	4	3	2	1	0	
Identifier	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

Εικόνα 4. Πίνακας bits του καταχωρητή ελέγχου TCCR1A

Bit No	7	6	5	4	3	2	1	0	
Identifier	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Εικόνα 5. Πίνακας bits του καταχωρητή ελέγχου TCCR1B

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV Flag set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Εικόνα 6. Πίνακας Waveform Generation Mode Bit

CS12	CS11	CS10	Description
0	0	0	No Clock Source
0	0	1	System Clock
0	1	0	Prescaler = 8
0	1	1	Prescaler = 64
1	0	0	Prescaler = 256
1	0	1	Prescaler = 1024
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Εικόνα 7. Πίνακας Clock Select Bit

Τέλος, για την αποθήκευση του αποτελέσματος, χρησιμοποιώ την Port D (0xFF), ενώ για την αύξηση και μείωση του duty cycle, φορτώνω τον καταχωρητή DDRB και χρησιμοποιώ από το PINB, PIN0 και PIN1 αντίστοιχα.

Main

Ο βασικός κώδικας αποτελείται από έναν βρόχο *main* που επαναλαμβάνεται μέχρι να πατηθεί ο διακόπτης, δηλαδή να μετατραπεί είτε το PINB1 είτε το PINB0 από μηδέν σε 1. Ο βρόχος αυτός περιέχει τρεις βρόχους: δύο βρόχους, *scan0* και *scan1*, για τον έλεγχο των PINB0 και PINB1 αντίστοιχα, και έναν βρόχο *move* για την χρονική καθυστέρηση που απαιτείται για την κίνητη του σερβοκινητήρα.

1. *scan0:* Αρχικά, γίνεται έλεγχος του PINB0 και αν ισούται με 1, σημαίνει πως ζητήθηκε από το πρόγραμμα να αυξηθεί το duty cycle. Στη συνέχεια, φορτώνεται ο καταχωρητής OCR1A και συγκρίνονται τα high και low bytes του με τα αντίστοιχα της μέγιστης τιμής (2.5ms). Αν η σύγκριση δείξει ότι είναι ήδη επιλεγμένο το max duty cycle, γίνεται jump στον βρόχο *scan1* για να μειωθεί το duty cycle. Αν δεν ισχύει η προηγούμενη υπόθεση, φορτώνεται το βήμα (0.1ms) με το οποίο αυξάνεται ο παλμός, και το αποτέλεσμα των προσθέσεων αποθηκεύεται πίσω στον OCR1A. Τέλος, ακολουθείται η διακλάδωση στον βρόχο *move* για να πραγματοποιηθεί η κίνηση του σερβοκινητήρα μετά την απαραίτητη καθυστέρηση.
2. *scan1:* Ακολουθώ παρόμοια διαδικασία με αυτήν του βρόχου *scan0.* Αντί της σύγκρισης με την μέγιστη τιμή, πραγματοποιείται σύγκριση με την ελάχιστη τιμή, για να μεταφερθεί ο μετρητής προγράμματος στον προηγούμενο βρόχο σε περίπτωση που έχουμε ήδη min duty cycle. Επίσης, αντί να προσθέσω το βήμα στον παλμό, το αφαιρώ.
3. *move:* Στον βρόχο αυτό υπάρχουν δύο διακλαδώσεις καθυστέρησης (*delay1* και *delay2*). Η δημιουργία του κώδικα για τις καθυστερήσεις βασίζεται στον τον εξής τρόπο[11]:
    - Υπολογισμός καθυστέρησης: $T_{delay} = (N_{mc} \times L_{cnt} - 1)t_{mc}$ όπου T_{delay} η χρονική καθυστέρηση που δημιουργείται στον βρόχο, N_{mc} το σύνολο των κύκλων, L_{cnt} το σύνολο των τρεχουσών επαναλήψεων, και t_{mc} η περίοδος ενός κύκλου = $1/f_{CLK}$.= 1/1MHz = 1μs Οι κύκλοι που απαιτούνται για κάθε εντολή βρίσκονται στο Instruction Set του εγχειριδίου του ATmega328.
    - Υπολογισμός σύνολο επαναλήψεων για δεδομένη καθυστέρηση: Επιλέγω να έχω 100ms καθυστέρηση. Άρα, $L_{cnt} = (T_{delay} / t_{mc} + 1)/ N_{mc} = (100 \times 10^{-3} / 10^{-6} + 1)/4 = 250$ δηλαδή 1ms. Το σύνολο των κύκλων (4) υπολογίζεται από τον κώδικα.
    - Υπολογισμός εξωτερικού βρόχου καθυστέρησης: Για να έχω συνολικό χρόνο καθυστέρησης 100ms, θέτω την εξωτερική καθυστέρηση ίση με 100 και την πολλαπλασιάζω με την εσωτερική καθυστέρηση σε κάθε επανάληψη[12].

Κώδικας

```
.DEF DL1 = R24      ; define register for delay1
.DEF DL2 = R25      ; define register for delay2
.EQU MIN = 0x01F4   ; min value of duty cycle = 0.5ms (500 microseconds)
.EQU MAX = 0x09C4   ; max value of duty cycle = 2.5ms (2500 microseconds)
.EQU STEP = 0x64    ; step by 0.1ms (100 microseconds)

LDI R16, 0XFF
STS DDRD, R16       ; port D is output (255)

LDI R16, HIGH(MIN)
STS OCR1AH, R16     ; store high byte
LDI R17, LOW(MIN)
STS OCR1AL, R17     ; store low byte

; WGM10 = 0, WGM11 = WGM12 = WGM13 = 1  -> Fast PWM where TOP = ICR1
; COM1A1 = 1, COM1A0 = 0 -> non-inverted mode
; CS10 = 1, CS11 = CS12 = 0 -> no prescaling
; COM1A0, CS11 and CS12 already 0 so we don't set them

LDI R16, (1<<WGM11) | (1<<COM1A1)
STS TCCR1A, R16
LDI R17, (1<<WGM13) | (1<<WGM12) | (1<<CS10)
STS TCCR1B, R17

CBI DDRB, 0     ; PINB0 will be used to increase duty cycle
CBI DDRB, 1     ; PINB1 will be used to decrease duty cycle
```

```
main:           ; loop between scan0 and scan1 till switch is pressed
                ; and if so, go to move delay and then loop again
    scan0:
        SBIS PINB, 0        ; skip rjmp if PINB0 = 1 (if switch is pressed)
        RJMP scan1          ; PC jump to check if PINB1 of switch is pressed
        LDI R16, OCR1AH     ; R16 contains high byte of OCR1A
        LDI R17, OCR1AL     ; R17 contains low byte of OCR1A
        CPI R16, HIGH(MAX)  ; compare high bytes (OCR1AH and HIGH(MAX))
        CPI R17, LOW(MAX)   ; compare low bytes (OCR1AL and LOW(MAX))
        BREQ scan1          ; if we have max duty cycle, branch to scan1
        LDI R18, STEP       ; load 0x64 to R18
        ADD R16, R18        ; R16 += 0x64
        LDI R18, 0X00       ; R18 = 0X00
        ADD R17, R18        ; R17 += 0X00
        STS OCR1AH, R17     ; store R17 to high byte of OCR1A
        STS OCR1AL, R16     ; store R16 to low byte of OCR1A
        RJMP move           ; jump to move the servo
```

```
    scan1:
        SBIS PINB, 1        ; skip rjmp if PINB1 = 1 (if switch is pressed)
        RJMP scan0          ; PC jump to check if PINB0 of switch is pressed
        LDI R16, OCR1AH
        LDI R17, OCR1AL
        CPI R16, HIGH(MIN)  ; compare high bytes (OCR1AH and HIGH(MIN))
        CPI R17, LOW(MIN)   ; compare low bytes (OCR1AL and LOW(MIN))
        BREQ scan0          ; if we have min duty cycle, branch to scan0
        LDI R18, STEP       ; load 0x64 to R18
        SUB R16, R18        ; R16 -= 0X64
        LDI R18, 0X00       ; R18 = 0X00
        SUB R17, R18        ; R17 -= 0X00
        STS OCR1AH, R17     ; store R17 to high byte of OCR1A
        STS OCR1AL, R16     ; store R16 to low byte of OCR1A
        RJMP move           ; jump to move the servo

    move:                   ; time delay in order to move the servo
        LDI DL1, 100        ; total delay time = DL1*DL2 = 100*1ms = 100ms
        delay1:
            LDI DL2, 250    ; DL2 = 250(hex) = 1(decimal) so DL2 = 1ms
        delay2:             ;
            NOP             ; 1 clock cycle wait
            DEC DL2         ; 1 clock cycle (DL2 = DL2 - 1)
            BRNE delay2     ; 2 clock cycles if true, 1 cycle if false
                            ; repeat until counter DL2 = 0
            DEC DL1         ; 1 clock cycle (DL1 = DL1 - 1)
            BRNE delay1     ; 2 clock cycles if true, 1 cycle if false
                            ; repeat until counter DL1 = 0
    RJMP main       ; jump back to beginning of main
```

ΑΝΑΦΟΡΕΣ - ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Per. Christensson, *Pipeline Definition,* TechTerms. Sharpened Productions, 2006

[2] Αριστείδης Ευθυμίου, *Υπόβαθρο: Διοχέτευση, Αρχιτεκτονική Υπολογιστών,* Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής, Πολυτεχνική Σχολή - Πανεπιστήμιο Ιωαννίνων, 2014

[3] Δημήτριος Β. Νικολός, *Αρχιτεκτονική Υπολογιστών,* Εκδόσεις Β. Γκιούρδας, ISBN: 978-960-387-795-0, 2008

[4] Sarah L. Harris, David Money Harris, *Digital Design and Computer Architecture: Arm Edition*, Εκδόσεις Morgan Kaufmann, ISBN 9780128000564, 2016

[5] Nishant Kumar, Ekta Aggrawal, *General Purpose Six-Stage Pipelined Processor,* International Journal of Scientific & Engineering Research, Volume 4, Issue 12, ISSN 2229-5518, 2013

[6] Robert D. Christ, Robert L. Wernli, *The ROV Manual (Second Edition)*, Εκδόσεις Butterworth-Heinemann, ISBN 9780080982885, 2014

[7] *Pulse Width Modulation in AVR Microcontroller*, JavaTpoint, 2011-2018

[8] *AVR Microcontrollers*, ScienceProg, 2006-2020

[9] *Controlling a Hobby Servo using an Arduino,* PHD Robotics, NewbieHack, 2020

[10] Atmel Corporation, *ATmega328P Datasheet,* San Jose, USA, 2015

[11] Grant M. Hill, Ph.D., *AVR Control Transfer – AVR Looping*, Department of Kinesiology, California State University, Long Beach, USA, 2009

[12] Muhammad Ali Mazidi, Sarmad Naimi, Sepehr Naimi, *The AVR Microcontroller and Embedded System*, Εκδόσεις Pearson, ISBN-10 0-13-800331-9, 2011