14. Timers/Counters

Οι AVR διαθέτουν μετρητές (Timers) με δυνατότητα σύγκρισης και διακοπής. Ο χρονισμός τους μπορεί να είναι σύγχρονος ή ασύγχρονος, από το ρολόι συστήματος ή από εξωτερική πηγή και υπάρχει δυνατότητα διαίρεσης της συχνότητας χρονισμού με **prescaler** ώστε να επιτευχθεί ο επιθυμητός ρυθμός μέτρησης. Χρησιμοποιούνται στην παραγωγή παλμών διαμόρφωσης πλάτους (PWM) και όπου χρειάζεται ακριβής χρονισμός ή διακοπή μετά το πέρας κάποιου συγκεκριμένου χρονικού διαστήματος.

Ο ATmega16 διαθέτει τρεις μετρητές **Timer/Counter0 – 2**. Οι **Timer/Counter0** και **Timer/Counter2** είναι 8-bit, ενώ ο **Timer/Counter1** είναι 16-bit και είναι διπλός **A** και **B**. Οι **Timer/Counter0** και **Timer/Counter1** έχουν σύγχρονη λειτουργία ενώ ο **Timer/Counter2** ασύγχρονη. Οι μετρητές κάνουν μέτρηση και σύγκριση μεταξύ των καταχωρητών τους. Ο καταχωρητής μέτρησης είναι ο **TCNTn** (n: 0, 1A, 1B, 2) και μεταβάλλει το περιεχόμενό του προς τα πάνω ή προς τα κάτω, από το ελάχιστο όριο **Bottom** έως το μέγιστο **Max** ή **Top**, ανάλογα με τον τρόπο λειτουργίας του μετρητή. Το περιεχόμενό του συγκρίνεται με το περιεχόμενο του καταχωρητή σύγκρισης **OCRn** και όταν υπάρξει ισότητα ή κατά την υπερχείλιση του **TCNTn**, προκαλείται μια διακοπή. Η διακοπές που προκαλούνται και οι σημαίες που τοποθετούνται στα όρια διαμορφώνουν κάποιες λειτουργίες. Το όριο **Bottom** είναι σε όλους τους μετρητές 0 (**0x0**), το όριο **Max** εξαρτάται από τα bit του μετρητή και το **Top** από τον τρόπο λειτουργίας του και από τον **OCRn** ή **ICR1** (**Timer/Counter1**).

Οι τρόποι λειτουργίας των μετρητών καθορίζονται από τα **WGMnx** bit του καταχωρητή **TCCRn**. Ο πιο απλός τρόπος λειτουργίας είναι ο **Normal Mode** όπου ο **TCNTn** μετράει προς τα πάνω μέχρι το μέγιστο και ξεκινάει πάλι από την αρχή. Όταν φτάσει στο μέγιστο τοποθετείται η σημαία υπερχείλισης **TOVn** και προκαλείται η αντίστοιχη διακοπή, αν είναι ενεργοποιημένη από το bit **TOIEn** του καταχωρητή **TIMSK**, η οποία μηδενίζει την **TOVn**.

Στη λειτουργία **Clear Timer on Compare Match (CTC) Mode** ο **TCNTn** μετράει προς τα πάνω και συγκρίνεται με το περιεχόμενο του καταχωρητή **OCRn**. Όταν υπάρξει ισότητα τοποθετείται η σημαία **OCn** και προκαλείται η αντίστοιχη διακοπή, αν είναι ενεργοποιημένη από το bit **OCIEn** του καταχωρητή **TIMSK**, ενώ ταυτόχρονα μηδενίζει ο **TCNTn** και ξεκινάει το μέτρημα ξανά. Την κατάσταση της **OCn** μπορούμε να πάρουμε ως κυματομορφή στο αντίστοιχο pin του PORT, εφόσον το ενεργοποιήσουμε ως έξοδο και ρυθμίσουμε την **OCn** από τα bit **COMnx** του καταχωρητή **TCCRn** να αλλάζει την λογική της κατάσταση στο σημείο σύγκρισης (Toggle). Η κυματομορφή εξόδου θα είναι συμμετρικοί παλμοί όπου η συχνότητα

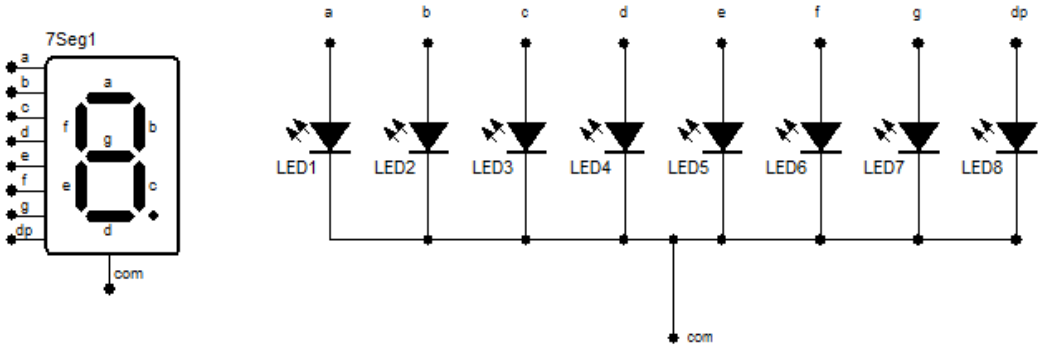
εξαρτάται από τη συχνότητα του ρολογιού του συστήματος, τον prescaler και από την τιμή του **OCRn**.

Στη λειτουργία **Fast PWM Mode** ο **TCNTn** μετράει προς τα πάνω και συγκρίνεται με το περιεχόμενο του καταχωρητή **OCRn**. Όταν υπάρξει ισότητα τοποθετείται (ή μηδενίζει στην ανάστροφη λειτουργία) η σημαία **OCn** και προκαλείται η αντίστοιχη διακοπή, αν είναι ενεργοποιημένη από το bit **OCIEn** του καταχωρητή **TIMSK**, χωρίς να μηδενίζει ο **TCNTn**, ο οποίος συνεχίζει το μέτρημα μέχρι το **Max**. Όταν φτάσει στο **Max** μηδενίζει (ή τοποθετείται) η **OCn**, μηδενίζει ο **TCNTn** και ξεκινάει το μέτρημα ξανά, τοποθετείται η σημαία υπερχείλισης **TOVn** και προκαλείται η αντίστοιχη διακοπή αν είναι ενεργοποιημένη, η οποία μηδενίζει την **TOVn**. Την κατάσταση της **OCn** μπορούμε να πάρουμε ως κυματομορφή στο αντίστοιχο pin του PORT, εφόσον το ενεργοποιήσουμε ως έξοδο και ρυθμίσουμε κατάλληλα την **OCn** από τα bit **COMnx** του καταχωρητή **TCCRn** σε κανονική, ανάστροφη ή Toggle λειτουργία. Η κυματομορφή εξόδου θα είναι παλμοί με σταθερή συχνότητα και διαμόρφωση εύρους (duty cycle) από 0-100%. Η συχνότητα εξαρτάται από τη συχνότητα του ρολογιού του συστήματος και τον prescaler ενώ το duty cycle από την τιμή του **OCRn**.

Η λειτουργία **Phase Correct PWM Mode** είναι σχεδόν ίδια με την **Fast PWM** με τη διαφορά ότι όταν ο **TCNTn** φτάσει στην ανώτερη τιμή δε μηδενίζει, αλλά αρχίζει να μετράει αντίστροφα (ελαττώνεται) μέχρι να φτάσει στο μηδέν όπου ξεκινάει να αυξάνει ξανά. Όταν υπάρξει ισότητα τοποθετείται (ή μηδενίζει στην ανάστροφη λειτουργία) η **OCn** όταν ο **TCNTn** βρίσκεται στην ανοδική φάση και το αντίθετο στην καθοδική. Η **TOVn** τοποθετείται όταν κατά την ελάττωση φτάσει στο μηδέν.

Ο **Timer/Counter1** διαθέτει περισσότερους τρόπους λειτουργίας από τους άλλους δύο Timers και για αυτό το λόγο ο έλεγχος γίνεται από 4 bit (**WGM13:0**). Επίσης, μολονότι είναι διπλός (**A** και **B**) διαθέτει έναν καταχωρητή μέτρησης **TCNT1**. Οι επιπλέον τρόπου λειτουργίας εντοπίζονται κυρίως στις **PWM** λειτουργίες όπου ο **TCNT1** αυξάνει μέχρι το **Top** το οποίο μπορεί να είναι 8, 9 ή 10 bit ή να εξαρτάται από τον **OCR1A/B** ή **ICR1**. Υπάρχει επίσης η λειτουργία **Phase and Frequency Correct PWM Mode** όπου η μοναδική διαφορά της με την **Phase Correct PWM Mode** είναι ότι στην πρώτη ο καταχωρητής **OCR1A/B** ενημερώνεται από τον αντίστοιχο buffer όταν ο **TCNT1** βρίσκεται στο **Top** ενώ στη δεύτερη στο **Bottom**.

15. Seven Segment Display. Πολυπλεγμένη λειτουργία



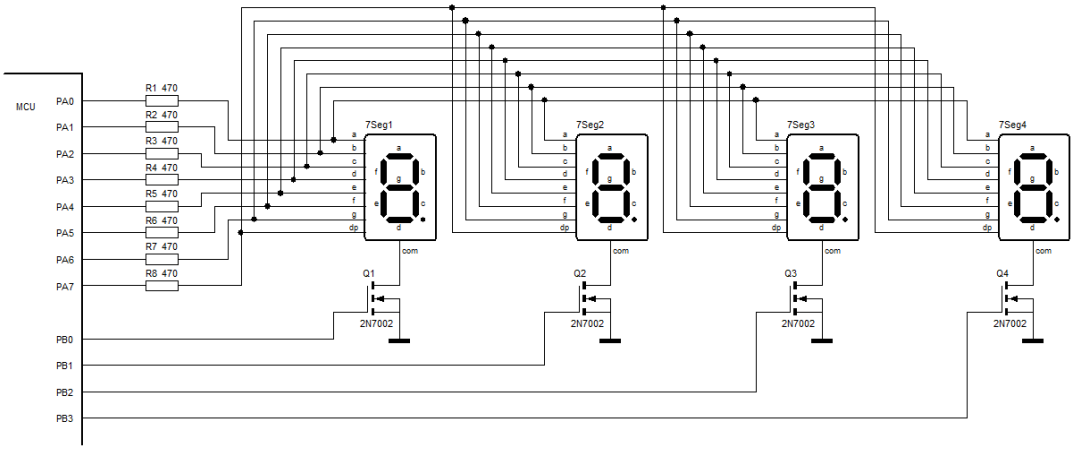
Τα Seven Segment Display με LED είναι ίσως η πιο συνηθισμένη μέθοδος απεικόνισης αριθμών. Η διακριτική τους ικανότητα, η υψηλή φωτεινότητα σε συνάρτηση με το μεγάλο μέγεθος, την ποικιλία των χρωματικών αποχρώσεων και την μικρή κατανάλωση ρεύματος τα κάνει ελκυστικά σε διάφορες εφαρμογές. Το κάθε display μπορεί να απεικονίσει τους αριθμούς και ορισμένα γράμματα[1] με συνδυασμό επτά LED σε κατάλληλη διάταξη και ένα όγδοο για την τελεία (ως υποδιαστολή ή ένα διπλό για άνω – κάτω τελεία). Το κάθε LED χαρακτηρίζεται από ένα γράμμα και το ένα άκρο του είναι ελεύθερο σε ακροδέκτη ενώ το άλλο συνδέεται εσωτερικά με τα υπόλοιπα σε έναν κοινό ακροδέκτη. Ο κοινός ακροδέκτης μπορεί να είναι οι άνοδοι ή οι κάθοδοι των LED και με αυτόν τον τρόπο χαρακτηρίζεται ο τύπος του display, κοινής ανόδου ή καθόδου αντίστοιχα. Οι αριθμοί σχηματίζονται αν ανάψουν συγκεκριμένα LED δίνοντας τάση με κατάλληλη πόλωση (μια αντίσταση περιορισμού του ρεύματος για κάθε είναι απαραίτητη). Για να σχηματιστεί το 2 για παράδειγμα πρέπει να ανάψουν τα a, b, g, e, d.

Υπάρχουν πολλά ηλεκτρονικά κυκλώματα που υποστηρίζουν Seven Segment Display καθώς και ολοκληρωμένα όπως τα **CD4511** και **MC14495**, τα οποία είναι αποκωδικοποιητές **BCD** και **Hexadecimal** σε **Seven Segment Display (with latch)** αντίστοιχα, ή το **MAX6954**, το οποίο είναι προγραμματιζόμενος αποκωδικοποιητής με κύκλωμα πολύπλεξης. Επίσης μπορεί να γίνει χρήση μικροελεγκτή για αποκωδικοποίηση και απεικόνιση. Οι απαιτήσεις σε λογισμικό είναι ελάχιστες καθώς δε χρειάζονται ειδικές βιβλιοθήκες και ο κώδικας περιορίζεται σε λίγες μόνο εντολές, με αποτέλεσμα να μπορεί να χρησιμοποιηθεί ο ίδιος ο

[1] *Μπορούν να απεικονιστούν όλοι οι αριθμοί και γράμματα όπως τα A, b, C, c, d, E, F, H, i, J, L, n, o, P, r, t, U, u.*

μικροελεγκτής που εκτελεί την εφαρμογή η οποία χρειάζεται συσκευή απεικόνισης, αρκεί να έχει ελεύθερες εξόδους.

Αν χρησιμοποιήσουμε περισσότερα από ένα Seven Segment Display τα σήματα προς αυτά είναι πολυπλεγμένα. Συνδέονται μεταξύ τους παράλληλα, εκτός από την κοινή κάθοδο (ή άνοδο), η οποία ελέγχεται μέσω ενός τρανζίστορ – διακόπτη ξεχωριστά για το κάθε display. Ένα μόνο display ενεργοποιείται κάθε φορά μέσω του τρανζίστορ εμφανίζοντας το νούμερο που αντιστοιχεί στην κατάλληλη θέση. Η εναλλαγή γίνεται με υψηλή ταχύτητα ώστε το ανθρώπινο μάτι να αντιλαμβάνεται όλα τα ψηφία αναμμένα. Η τεχνική αυτή απλουστεύει το σχεδιασμό αφού χρειάζονται λιγότερες έξοδοι (8 για τα 7 segment και την τελεία και από μια για κάθε ένα display). Αν χρησιμοποιήσουμε εξωτερικούς αποκωδικοποιητές χρειάζονται ακόμα λιγότεροι έξοδοι, για παράδειγμα με έναν αποκωδικοποιητή **BCD to Seven Segment Display (74LS347)** και έναν αποκωδικοποιητή **4-Line to 16-Line Decoder/Demultiplexer (74LS154)** αρκούν 8 έξοδοι (ένα port) για 16 display. Ο έλεγχος γίνεται σε όλες τις περιπτώσεις από τον μικροελεγκτή. Ένα τυπικό κύκλωμα οθόνης με 4 Seven Segment Display τύπου LED παρουσιάζεται στο παρακάτω σχήμα.



Για την υποστήριξη του παραπάνω κυκλώματος χρειάζεται να χρησιμοποιηθεί ένας timer για την εναλλαγή των display σε σταθερά χρονικά διαστήματα, ώστε είναι σταθερή η φωτεινότητα και να μη γίνεται αισθητή διακοπή ή τρεμοπαίξιμο. Με αυτόν τον τρόπο η εναλλαγή είναι ανεξάρτητη από την εκτέλεση του προγράμματος. Οι απαραίτητες εντολές για την εναλλαγή βρίσκονται στο διάνυσμα της διακοπής που προκαλείται από τον timer με αποτέλεσμα στον κυρίως κώδικα να χρειάζεται μόνο να τροποποιείται η μεταβλητή που χρησιμοποιείται ως buffer για τα display. Στη συνέχεια παρουσιάζεται ένα παράδειγμα γραμμένο σε γλώσσα προγραμματισμού C:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

uint8_t test[]={'0','1','2','3','4','5','6','7','8','9','A','b','C','d','E',
                'F','-','.',' ',0};
uint8_t buffer[4];
uint8_t display[4];
uint8_t i1=0;
uint8_t digit=0;

void ato7sd (unsigned char buffer2)//ASCII to 7 Seg Disp conversion routine
{
    switch (buffer2)
    {
        case '0':
        display[i1] = 0b00111111;        //FEDCBA      ABCDEF
        break;

        case '1':
        display[i1] = 0b00000110;        //CB          CB
        break;

        case '2':
        display[i1] = 0b01011011;        //GEDBA       ABDEG
        break;

        case '3':
        display[i1] = 0b01001111;        //GDCBA       ABCDG
        break;

        case '4':
        display[i1] = 0b01100110;        //GFCB        BCFG
        break;

        case '5':
        display[i1] = 0b01101101;        //GFDCA       ACDFG
        break;

        case '6':
        display[i1] = 0b01111101;        //GFEDCA      ACDEFG
        break;

        case '7':
        display[i1] = 0b00000111;        //CBA         ABC
        break;

        case '8':
        display[i1] = 0b01111111;        //GFEDCBA     ABCDEFG
        break;

        case '9':
        display[i1] = 0b01101111;        //GFDCBA      ABCDFG
        break;

        case '.':
        display[i1] = 0b10000000;        //H
        break;

        case '-':
        display[i1] = 0b01000000;        //G
        break;
```

```
        case 'A':
        display[i1] = 0b01110111;        //GFECBA      ABCEFG
        break;

        case 'b':
        display[i1] = 0b01111100;        //GFEDC       CDEFG
        break;

        case 'C':
        display[i1] = 0b00111001;        //FEDA        ADEF
        break;

        case 'd':
        display[i1] = 0b01011110;        //GEDCB       BCDEG
        break;

        case 'E':
        display[i1] = 0b01111001;        //GFEDA       ADEFG
        break;

        case 'F':
        display[i1] = 0b01110001;        //GFEA        AEFG
        break;

        case 'o':
        display[i1] = 0b01011100;        //GEDC        CDEG
        break;

        case 'r':
        display[i1] = 0b01010000;        //GE          EG
        break;

        case ' ':
        display[i1] = 0;
        break;
    }
}

ISR(TIMER1_COMPA_vect)
{
    if (i1 >= 4)                  //For 4 displays
    {
        i1 = 0;                   //Reset display number after 4th display
        digit = 1;                //Reset display number after 4th display
    }
    ato7sd(buffer[i1]);           //Call ASCII to 7 Seg Disp conversion routine
    PORTB = 0;                    //Turn off display
    PORTA = display[i1];          //Send 7 Seg Disp format character to port
    PORTB = digit;                //Turn on display
    i1++;                         //Shift display in next cycle
    digit = digit<<1;             //Shift display in next cycle
}

int main(void)
{
    DDRA = 0b11111111;            //Segment port all outputs
    DDRB = 0b00001111;            //Displays port all outputs (PORTx0:3)

    TCCR1B |= (1 << WGM12);       //Configure timer 1 for CTC mode
    TCCR1B |= (3 << CS10);        //Start timer at Fcpu/64
    OCR1A = 500;                  //Set CTC compare for 4ms at 8MHz AVR clock
```

```
    TIMSK |= (1 << OCIE1A);        //Enable CTC interrupt
    sei();                         //Enable global interrupts

    while (1)
    {
        for (uint8_t i2 = 0; i2 < 17; i2++)
        {
            buffer[0]=test[i2];
            buffer[1]=test[i2+1];
            buffer[2]=test[i2+2];
            buffer[3]=test[i2+3];
            _delay_ms(500);
        }

    }
}
```

Τα segment των display είναι συνδεδεμένα στο **PORTA** του μικροελεγκτή ενώ οι κοινές κάθοδοι, μέσω τρανζίστορ – διακοπτών, είναι συνδεδεμένες στο **PORTB3:0**, όπως δείχνεται στο θεωρητικό κύκλωμα. Η βιβλιοθήκη **avr/interrupt.h** είναι απαραίτητη για τη λειτουργία των διακοπών. Η μεταβλητή **test[]** είναι ένας πίνακας που χρησιμοποιείται μόνο για την επίδειξη των χαρακτήρων. Η μεταβλητή **buffer[4]** χρησιμοποιείται ως πίνακας που τοποθετούνται σε κωδικοποίηση ASCII οι χαρακτήρες που εμφανίζονται στα display. Η μεταβλητή **display[4]** χρησιμοποιείται ως πίνακας που τοποθετούνται σε κωδικοποίηση Seven Segment Display οι χαρακτήρες που εμφανίζονται στα display, ώστε να οδηγηθούν στη συνέχεια στο **PORTA**. Η μεταβλητή **i** χρησιμοποιείται για την απαρίθμηση των display κατά την εναλλαγή τους και η μεταβλητή **digit** χρησιμοποιείται για την εναλλαγή των display.

Στην αρχή του κώδικα εμφανίζεται η ρουτίνα **ato7sd** η οποία αναλαμβάνει τη μετατροπή της κωδικοποίησης των χαρακτήρων από ASCII σε Seven Segment Display. Περιέχει μια εντολή **switch** όπου στις διάφορες **case** αποθηκεύεται, στο στοιχείο της μεταβλητής **display[i]**, η τιμή που πρέπει να οδηγηθεί στο PORT όταν θα είναι η σειρά του αντίστοιχου ψηφίου να εμφανιστεί.

Στη συνέχεια, στο μπλοκ κάτω από τον τίτλο του διανύσματος **ISR(TIMER1_COMPA_vect)**, βρίσκεται ο κώδικας που εκτελείται από το διάνυσμα της διακοπής του Timer. Αρχικά γίνεται έλεγχος της μεταβλητής **i** και μηδενισμός της μετά την τιμή 3 (επιτρεπτές τιμές 0, 1, 2, 3 για χρήση τεσσάρων display αριθμημένα από αριστερά προς δεξιά). Στη συνέχεια γίνεται κλήση της ρουτίνας **ato7sd** όπου είσοδος της ρουτίνας είναι η μεταβλητή **buffer[i]** και έξοδος η μεταβλητή **display[i]**. Με την επόμενη εντολή σβήνουν όλα τα display ώστε να προετοιμαστεί η αλλαγή τους, η οποία γίνεται στη συνέχεια αποδίδοντας στο **PORTA** την τιμή της **display[i]** και στο **PORTB** η τιμή της **digit** ώστε να ανάψει το νέο display με την αντίστοιχη τιμή του. Ακολουθεί η προετοιμασία για τον επόμενο κύκλο με αύξηση της **i** και

αριστερή ολίσθηση της `digit`. Το σβήσιμο των display είναι απαραίτητο, γιατί δεν είναι δυνατό να φορτωθούν ταυτόχρονα και στα δύο PORT οι νέες τιμές με αποτέλεσμα να φαίνεται αχνά στο display η τιμή του επόμενου (ή του προηγούμενου, ανάλογα με το ποιο PORT θα ενημερωθεί πρώτο).

Ακολουθεί το κυρίως μέρος του προγράμματος `int main(void)`. Προγραμματίζονται τα δύο PORT ως έξοδοι, ρυθμίζεται ο **Timer/Counter1A** για λειτουργία **CTC** με **prescaler Fcpu/64** και όριο σύγκρισης 500 και ενεργοποιούνται η διακοπή **CTC** και οι γενικές διακοπές. Το μπλοκ της `while(1)` περιέχει κώδικα μόνο για επίδειξη των display και δε χρειάζεται να συμπεριληφθεί στις διάφορες εφαρμογές (ομοίως και η μεταβλητή `test[]`). Για να εμφανιστεί ένας χαρακτήρας στο κατάλληλο ψηφίο πρέπει να γίνει απόδοση της τιμής του σε κωδικοποίηση ASCII στην αντίστοιχη θέση της μεταβλητής `buffer[i]` κάπου μέσα στο μπλοκ της `while(1)`, σε οποιοδήποτε σημείο θέλουμε του κώδικά μας.

Η κλήση της ρουτίνας `ato7sd` δεν είναι υποχρεωτικό να καλείται μέσα από το διάνυσμα της διακοπής, αλλά μπορεί να καλείται μέσα από τον κυρίως κώδικα κάθε φορά που θέλουμε να αλλάξουμε τιμή σε κάποιο ψηφίο. Είναι δυνατό μάλιστα να αφαιρεθεί και η μεταβλητή `buffer[i]` αν ως είσοδο της `ato7sd` δίνουμε σε κωδικοποίηση ASCII τον επιθυμητό χαρακτήρα. Με αυτόν τον τρόπο ελαττώνεται το μέγεθος του κώδικα και ελαφραίνει το φορτίο του μικροελεγκτή, αφού δεν εκτελεί τη ρουτίνα σε κάθε διακοπή. Αυξάνει όμως η πολυπλοκότητα του προγράμματος στον κυρίως κώδικα, αφού θα πρέπει πάντα να θυμόμαστε να καλούμε τη ρουτίνα για να αποδώσουμε νέα τιμή. Επίσης γίνεται δυσκολότερη η φορητότητα του κώδικα πολύπλεξης, αφού θα πρέπει να προσαρμόζεται στις νέες απαιτήσεις σε περισσότερα σημεία.

Η επιλογή του **prescaler** και του ορίου σύγκρισης έγινε ώστε να είναι κάθε display αναμμένο 4ms και η συχνότητα σάρωσης να είναι λίγο πάνω από 60Hz, ώστε να μην είναι ορατή η εναλλαγή και να είναι ικανοποιητική η φωτεινότητα των display.