# Εργαστήριο 9

# 17/12/2020

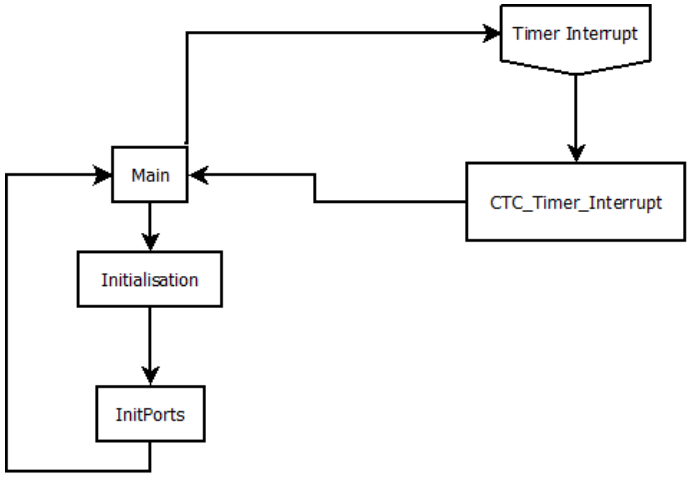
# Αφαιρώντας την Αναπήδηση Διακοπτών χωρίς και με Εξωτερικές Διακοπές (Interrupts)

Ντουνέτας Δημήτρης
ΑΜ: 2016030141

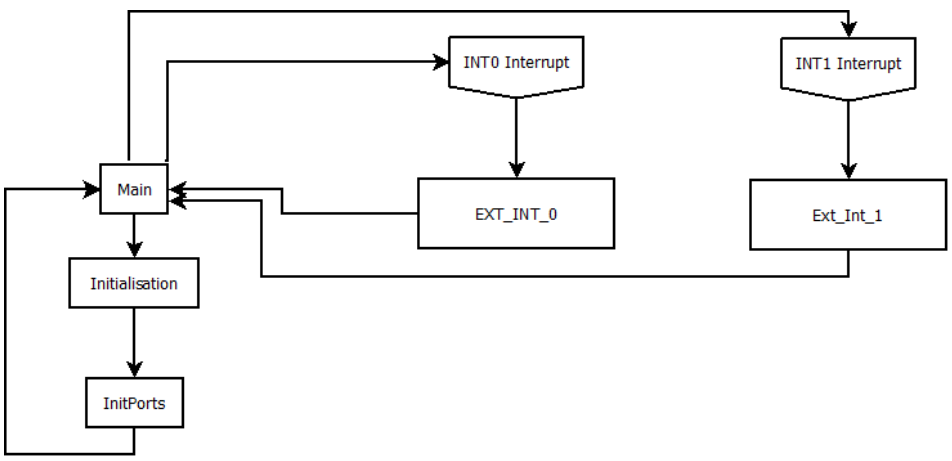
## Εισαγωγή

Σκοπός του εργαστηρίου είναι η δημιουργία 2 απλών προγραμμάτων αφαίρεσης αναπήδησης από διακόπτες SPDT. Επίσης η παραπάνω εξοικείωση με την τεχνική του polling και των External Interrupts. Στο τέλος πραγματοποιήθηκαν 2 προγράμματα και εξομοιώθηκαν με χρήση Stimuli File.

## Block Diagram Προγράμματος Polling




## Block Diagram Προγράμματος Interrupts



# Αρχικοποίηση του προγράμματος σε Γλώσσα C

Αρχικά το πρόγραμμα ξεκινάει από την συνάρτηση main που βρίσκεται στο αρχείο main.c . Η main συνάρτηση είναι υπεύθυνη για την αρχικοποίηση του προγράμματος και της μνήμης καθώς και την λειτουργία του βρόγχου. Για γίνει η αρχικοποίηση, η Main καλεί τη συνάρτηση initialization. Ο Stack Pointer δεν χρειάζεται να αρχικοποιηθεί αφού τον αρχικοποιεί ο C Compiler από μόνος του ώστε να μπορούμε να επιστρέφουμε σωστά από τις ρουτίνες και τις συναρτήσεις που καλούμε.

Η Initialization στην συνέχεια καλεί την initPorts συνάρτηση η οποία κάνει την αρχικοποίηση μνήμης , καταχωρητών και Ports όπως έχουν αναφερθεί και στα προηγούμενα εργαστήρια.

## RAM MAP

### Debouncing με Polling

| 0x80 | 0x81 | 0x82 |
|---|---|---|
| switch1 | prev | count |

### Debouncing με Interrupts

| 0x80 | 0x81 | 0x82 |
|---|---|---|
| count | Count1 | lastBounced |

## Αρχικοποίηση Timer Counter για Polling

Για την αρχικοποίηση θέτουμε το prescale του TIMER/COUNTER1 που χρησιμοποιούμε για το πρόγραμμα που σε 1/8 βάζοντας άσσο στο bit CS11 του καταχωρητή TCCR1B και αλλάζουμε την τιμή του Καταχωρητή OCR1A σε 625 ώστε με ρολόι 10Mhz να μετράει για χρόνο 0.5ms.

## Αρχικοποίηση External Interrupts

Για την ενεργοποίηση των External Interrupts στο πρόγραμμα ενεργοποιούμε τα bits ISC10 και ISC00 στον Καταχωρητή MCUCR για την ρύθμιση λήψεως Interrupt σε κάθε αλλαγή λογικής τιμής στα PIND2 και PIND3. Τέλος, ενεργοποιούμε τα bits INT0 και INT1 στον Καταχωρητή GICR για την ενεργοποίηση των Interrupts INT0 και INT1.

# Λειτουργία του κύριου προγράμματος με Polling

Η λειτουργία του συγκεκριμένου προγράμματος είναι πολύ απλή. Αφού αρχικοποιηθεί το πρόγραμμα ελέγχει κάθε 0.5ms τις τιμές των PINA0 και PINA1. Αν αυτές είναι ίδιες τότε μηδενίζει τον counter που έχουμε στη μνήμη. Αν είναι διαφορετικές τότε αυξάνει τον counter και αποθηκεύει τις τιμές που έχει στην μνήμη. Αν για 10 συνεχόμενες φορές δηλαδή συνολικά για 10* 0.5ms = 5ms βρε τις τιμές να είναι ίδιες βγάζει στο PORTA3 την τιμή του PINA0.

```
void CTC_Timer_Interrupt(){

//  bit 0 = A and bit 1 = A'
    *switch1 = (PINA&(1<<PA0))|(PINA&(1<<PA1));

//  if both switch values are the same reset counter
    if ((*switch1 == 3) || (*switch1 == 0))
    {
        *count = 0;
    }
//  if both switch values are the same start counting
    if (((*switch1 == 1)||(*switch1 == 2))&&(*prev == *switch1)){
        *count +=1;
    }
//  if for 10 consecutive times we have the same values then the switch is stable
    if (*count == 10)
    {
        // Output the correct output in PORTA bit 2 including previous values of PORTA.
        PORTA= ((PINA&(1<<PA0)) << PA2);

        *count = 0;
    }
//  Save the switch values to evaluate next time
    *prev = *switch1;
    return;
}
```

# Λειτουργία του κύριου προγράμματος με External Interrupts

Σε αυτό το πρόγραμμα, λαμβάνουμε interrupts κάθε φορά που διενεργείται μια αλλαγή λογικής στα bits PIND2 και PIND3. Όταν η αλλαγή γίνεται στο PIND2 ενεργοποιείται ένα INT0 interrupt και αυξάνουμε τον counter. Αντίθετα όταν γίνεται στο PIND3 η αλλαγή ενεργοποιείται ένα INT1 interrupt και αυξάνουμε τον μετρητή count1. Όταν και οι 2 μετρητές είναι θετικοί του μηδενίζουμε. Έτσι κάθε φορά όταν θα έχει η μία πλευρά του διακόπτη bouncing θα γνωρίζουμε ότι είναι η πλευρά που δεν είναι μηδενική. Τελικά, όταν εντοπίσουμε ποια πλευρά έχει το bouncing αρκεί να βγάλουμε στην έξοδο την τιμή της αντίθετης της. Για να έχουμε τελικές τιμές στην έξοδο 0 και 1. Όταν το Α είναι σταθερό και το Α’ κάνει αναπηδήσεις βγάζουμε στην έξοδο PORTA3 το Α. Ενώ όταν το Α΄είναι σταθερό και το Α κάνει αναπηδήσεις βγάζουμε στην έξοδο το συμπληρωματικό του Α’.

```
void Ext_Int_0(){

    if (*count > *count1 && *count >=2 && *count1 == 0 && *lastBounced == 1)
    {
        // Means that A in PD2 is bouncing PD3(A') => Stable
        PORTA = 0b00000100&(~(PIND&(1<<PD3)))>> 1);
        *lastBounced = 0;
    }
    else if(*count > 0 && *count1 > 0)
    {
        *count = 0;
        *count1 = 0;
    }
    else
    {
        *count += 1;
    }
}
void Ext_Int_1(){

    if (*count1 > *count && *count1 >=2 && *count == 0 && *lastBounced == 0)
    {
        // Means that A' in PD3 is bouncing PD2(A) => Stable
        PORTA = 0b00000100 &((PIND&(1<<PD2)));
        *lastBounced = 1;
    }
    else if(*count > 0 && *count1 > 0)
    {
        *count = 0;
        *count1 = 0;
    }
    else{
        *count1 += 1;
    }
}
```

## STIMULI FILE για τον έλεγχο των προγραμμάτων

Με το παρακάτω Stimuli File μπορούμε να προσομοιώσουμε την αλλαγή του διακόπτη από μια θέση σε μια άλλη και ύστερα πάλι πίσω στην αρχική.

```
$log PORTA
$startlog lab9Switch.log
// Correct output 0
PIND = 0b00000100
#4000
PIND = 0b00000000
#4000
PIND = 0b00001000
#4000
PIND = 0b00000000
#4000
PIND = 0b00001000
#4000
PIND = 0b00000000
#4000
PIND = 0b00001000
```

```
#100000
// Correct output 1
#1000
PIND = 0b00000000
#4000
PIND = 0b00000100
#4000
PIND = 0b00000000
#4000
PIND = 0b00000100
#4000
PIND = 0b00000000
#4000
PIND = 0b00000100
#4000
PIND = 0b00000000
#4000
PIND = 0b00000100
```

```
#100000
// Correct output 0
#1000
PIND = 0b00000000
#4000
PIND = 0x00001000
#4000
PIND = 0b00000000
#4000
PIND = 0b00001000
#8000
PIND = 0b00000000
#4000
PIND = 0b00001000
#4000
PIND = 0b00000000
#4000
PIND = 0b00001000
```

## Ποσοστά χρήση πόρων στις 2 λύσεις

Για την υλοποίηση με χρήση polling. Στην δική μας περίπτωση που ελέγχουμε ανά 0.5 ms τα σήματα και με χρήση κύκλων 71 ανά κλήση της συνάρτησης υλοποιήσης στη χειρότερη περίπτωση μπορούμε να υποθέσουμε ότι θα κληθεί 20 φορές άρα θα καταναλώσει 1420 κύκλους ρολογιού από τις 100000. Έτσι η χρήση πόρων ανέρχεται σε 1.42%. Η χρήση πόρων είναι αρκετά μεγάλη για αυτή τη διαδικασία όμως καθώς το πρόγραμμα δεν εκτελεί άλλες διεργασίες θεωρήθηκε καλή τακτική να αυξηθεί η ευαισθησία. Για τη μείωση χρήσης πόρων μπορούμε να αυξήσουμε την τιμή μέτρησης του μετρητή και να μειώσουμε τον αριθμό συνεχόμενων ίδιων τιμών. Αυτό όμως θα μειώσει και την ανάλυση της μέτρησής μας.

Για την υλοποίηση με χρήση Interrupts η απάντηση αυτού του ερωτήματος δεν είναι τόσο εύκολη καθώς εξαρτάται άμεσα από τον αριθμό των bounce που θα εκτελέσει ο διακόπτης. Στην καλύτερη περίπτωση θα κληθεί 4 φορές η συνάρτηση του Interrupt μας. Με μέσο αριθμό κύκλων 50 ανά κλήση αυτό σημαίνει ότι αποφασίζουμε με χρήση πόρων 0,2%. Συνήθως όμως οι αναπηδήσεις συνεχίζονται και αφού έχουμε αποφασίσει. Έτσι έχουμε χρήση 2 κλήσεις ανά bounce. Κατά γενικό κανόνα έχουμε 2n + 2 κλήσεις ,όπου n ο αριθμός των bounces. Για το παράθυρο των 10ms μπορούμε να δώσουμε το ποσοστό χρήσης του Μικροελεγκτή ως : $\frac{100n+100}{100000}$ = X % .
Τέλος αν πάρουμε ως πιο πιθανή την τιμή του n να είναι 3 δηλαδή 3 αναπηδήσεις τότε το ποσοστό χρήση θα είναι περίπου 0.04%

## Τελικό Αποτέλεσμα

Τα προγράμματα τρέχουν συνεχόμενα χωρίς να σταματάνε αφού ως βασικό μέρος έχει έναν ατέρμονα βρόγχο. Διαβάζουν τα PIN που ορίζονται ως είσοδοι σε κάθε περίπτωση και ενεργούν κατάλληλα βγάζοντας την σωστή είσοδο. Μπορούμε να δούμε την έξοδο που παράγει το πρόγραμμα στο PortA3. Το πρόγραμμα τρέχει σωστά είτε θεωρήσουμε ότι το εσωτερικό έλασμα είναι γειωμένο και άρα έχει την τιμή 0 και οι επαφές είναι στην τιμή 1 με αντίσταση στα άκρα τους, είτε το αντίθετο. Οπότε μπορούμε να συνδέσουμε τον διακόπτη όπως επιθυμούμε.