Μικροεπεξεργαστές και Περιφερειακά

05/06/2019

Εγκατάσταση πλήρωσης υλικού

Υπατία Δάμη,8606

Ιάσων Καλαμποκίδης,8953

Σκοπός εργασίας

Σκοπός της εργασίας μας ήταν η δημιουργία ενός προγράμματος για τον έλεγχο λειτουργίας μιας εγκατάστασης πλήρωσης υλικού. Το πρόγραμμα αναπτύχθηκε με βάση τα χαρακτηριστικά λειτουργίας του μικροελεγκτή AVR ATMEGA16, και της δοκιμαστικής πλακέτας STK500. Στόχος μας ήταν η βέλτιστη προσομοίωση της λειτουργίας της εγκατάστασης, μέσω της χρήσης LEDS , διακοπτών , χρονομέτρων και αναλογικών διακοπτών. Με τα LEDS της STK500 (PORTB)απεικονίζονται τα διάφορα στάδια λειτουργίας του σιλό , καθώς και έκτακτες καταστάσεις όπως έλλειψη τροφοδοσίας υλικού. Με τους διακόπτες της πλακέτας(PORTD) αναπαρίστανται τα κουμπιά εκκίνησης και σταματημού της λειτουργίας , επιβεβαίωσης έκτακτης κατάστασης καθώς και ορισμένοι αισθητήρες, όπως αισθητήρες υπερθέρμανσης των κινητήρων της γραμμής. Ως χρονόμετρο για κάποιες διαδικασίες χρησιμοποιήσαμε τον εσωτερικό timer1 του AVR. Τέλος με τους αναλογικούς διακόπτες που συνδέονται στην PORTA του ΜΕ, αποσκοπούμε στην προσομοίωση των αισθητήρων βάρους, που είναι υπεύθυνοι για τον έλεγχο της παροχής του υλικού στην εγκατάσταση.

Υλοποίηση προγράμματος

Το πρόγραμμα ξεκινάει με τον ορισμό των διευθύνσεων διακοπών και reset, και στη συνέχεια αρχικοποιείται ο Stack Pointer και τίθεται η portb (LEDS) ως έξοδος , η portd(διακόπτες και αισθητήρες) ως είσοδος και η portc(αναλογικοί διακόπτες) ως είσοδος. Ως καταχηρητής προσωρινής αποθήκευσης δεδομένων (temp) έχει οριστεί ο R27 και επίσης έχει οριστεί ο R28 ως καταχωρητής επεξεργασίας των leds(led_manipulator), ο οποίος περιέχει πάντα την κατάσταση των leds ανάλογα με την λειτουργία του συστήματος.Στη συνέχεια αρχικοποιούνται οι καταχωρητές ελέγχου των interrupts και του timer1. Στο πρόγραμμα χρησιμοποιούμε την διακοπή INT0 με διπλή χρήση. Ενεργοποιείται σε περίπτωση που θέλουμε να διακόψουμε την λειτουργία της εγκατάστασης προσωρινά (STOP BUTTON), αλλα και για να δηλώσουμε την επιβεβαίωση μιας κατάστασης έκτακτης ανάγκης (ACK BUTTON). Τα δύο αυτά γεγονότα είναι αμοιβαίως αποκλειόμενα , επομένως για εξοικονόμιση πόρων υλοποιούμε και τις δύο λειτουργίες με την ίδια διακοπή. Η αναγνώριση της πηγής της διακοπής γίνεται μέσα στη ρουτίνα εξυπηρέτισης της διακοπής , όπως θα εξηγηθεί παρακάτω.

Μέσω των παρακάτω αναθέσεων , ορίζεται ο εντοπισμός του interrupt να γίνεται στο falling edge του σήματος (MCUCR) και κάνουμε enable του interrupt1 μέσω του GICR. Τέλος με την εντολή sei θέτουμε 1 στη σημαία των global interrupt του SREG και επιτρέπουμε τα interrupts να διακόψουν το πρόγραμμά μας.

```
ldi r17, (1 << ISC11)

sts MCUCR , r17

ldi  r17, (1<<INT1)

out GICR, r17

sei
```

Στη συνέχεια γίνεται αρχικοποίηση των καταχωρητών ελέγχου του timer1. Χρησιμοποιούμε τον timer1 δύο φορές στο πρόγραμμα, και τις δύο σε κανονική λειτουργία με overflow. Αρχικά χρησιμοποιείται για να μετρήσει 7 δευτερόλεπτα ,αφού έχει πατηθεί το κουμπί εκκίνησης της διαδικασίας, για να σταθεροποιηθεί η ταχύτητα του ιμάντα. Η δεύτερη λειτουργία του είναι να αναβοσβήνει ένα led με περίοδο 1 sec όταν το σύστημα βρεθεί σε κατάσταση acknowledge , μετά από μία έκτακτη κατάσταση.

Με την παρακάτω ανάθεση στον TIMSK ενεργοποιούμε την διακοπή overflow του timer1.

```
ldi r17 ,(1<< TOIE1)

out TIMSK ,r17
```

Έπειτα το πρόγραμμα ξεκινάει και εκτελεί ρουτίνες ελέγχου, για την ύπαρξη αποδεκτής ποσότητας υλικού στα σιλό και συνθήκες ασφάλειας και κατάστασης των κινητήρων . Η μέτρηση των τιμών των αισθητήρων βάρους γίνεται με polling . Η ρουτίνα ανακτά μια τιμή με τη σειρά από τον κάθε αισθητήρα και αποθηκεύει τις τιμές σε καταχωρητή για έλεγχό τους σε διαφορετική ρουτίνα. Αν είναι αποδεκτές συνεχίζεται η ροή κανονικά.

Στον παρακάτω κώδικα φαίνεται η μετατροπή μιας τιμής από έναν αισθητήρα και το σετάρισμα των καταχωρητών ADC. Ορίζουμε στον ADMUX την τάση εσωτερική 2,56 V και left adgusted μετατροπή του αποτελέσματος. (κρατάμε μόνο τα 8 σημαντικότερα bit από τα 10.). Στον ADSCRA κάνουμε enable και start conversion και στη συνέχεια ελέγχουμε συνεχώς το flag ένδειξης τέλους μετατροπής .Αποθηκεύουμε την τιμή στο high byte του ADC και έπειτα στον καταχωρητή A1 για έλεγχο αργότερα. Συνεχίζουμε την ίδια διαδικασία και για τους υπόλοιπους 4 αισθητήρες.

```
ldi r16,0b11100000

out ADMUX,r16

sbi ADCSRA,7
```

```
sbi ADCSRA,6

cnv0:

sbic ADCSRA,6

jmp cnv0

in r18,ADCH

mov A1,r18

ldi r16,0b11100001

out ADMUX,r16

sbi ADCSRA,7

sbi ADCSRA,6
```

Έπειτα τσεκάρονται οι υπόλοιπες συνθήκες σωστής κατάστασης για έναρξη λειτουργίας(πίεση πλήκτρου start και αγωγός εκφόρτωσης στο σιλό 1.) Αν όλα είναι εντάξει τότε θέτουμε τον timer σε λειτουργία δίνοντας τιμή (1024)στον prescaler του (TCCR1B καταχωρητής και θέτουμε την επιθυμητή τιμή στον TCCNT1 ώστε όταν κάνει overflow να έχουν περάσει 7 δευτερόλεπτα.Οι μεταθέσεις αυτές φαίνονται στον παρακάτω κώδικα:

```
ldi r17,HIGH(10845)

out TCNT1H,r17

ldi r17,LOW(10845)

out TCNT1L,r17

ldi r17, (1 << CS10)

out TCCR1B ,r17

ldi r17 ,(0 << CS11)

out TCCR1B ,r17

ldi r17 ,(1 << CS12)

out TCCR1B ,r17
```

Όταν ο timer κάνει overflow καλείται η ρουτίνα εξυπηρέτησής του , στην οποία αποθηκεύονται στη στοίβα ο καταχωρητής που θα χρησιμοποιηθεί και ο SREG. Μέσω του καταχωρητή led_manipulator εξακριβώνουμε αν η διακοπή προήλθε από

τη μέτρηση 7 δετερολέπτων ή από την ack ρουτίνα. (αν το led0 (error)είναι κλειστό τότε πρόκειται για τα 7 sec).Έπειτα ανακτούνται οι τιμές από τη στοίβα ενεργοποιούμε τα interrupts και επιστρέφουμε στο label timer1_finished , όπου ανάβουν τα κατάλληλα led λειτουργίας και ξεκινάει ο ιμάντας.

```
 timer1Handler:

push r17

 in r17, SREG

 push r17

mov r17,led_manipulator

andi r17, 0b00000001 ;

 cpi r17,0b00000000

breq 1_sec

7_secs:

pop r17

out SREG, r17

pop r17

sei

jmp timer1_finished
```

Στη συνέχεια ο ιμάντας ξεκινάει και βρισκόμαστε στη λειτουργία του σιλό 1. Το πρόγραμμα καλεί συνεχώς τις ρουτίνες ελέγχου αισθητήρων υλικού. Έτσι όταν το σιλό 1 αδείάζει καλείται η ρουτίνα που μεταφέρει τον παροχέα υλικού στο σιλό 2. Όταν αδειάσει και αυτό ανάβει το led3 και το πρόγραμμα τελειώνει. Αν κατά τη διάρκεια της διαδικασίας ενεργοποιηθούν οι αισθητήρες υπερθέρμανσης Q1,Q1 είτε πατήσουμε εκκίνηση ενώ το κύριο σιλό είναι άδειο, τότε το πρόγραμμα μεταφέρεται στη ρουτίνα χειρισμού της σειρήνας, η οποία είναι συνδεδεμένη στην PORTC και ηχεί μόλις κληθεί η ρουτίνα. Για να σταματήσει να ηχεί πρέπει να πατηθεί το πλήκτρο ACK που το έχουμε συνδέσει με τον INTERRUPT1 (pin3 της portd).Όταν πατήσουμε το sw3 το int1 διακόπτει το πρόγραμμα και καλέι τη ρουτίνα εξυπηρέτησης της διακοπής,όπου αναγνωρίζεται μέσω του led_manipulator(όπως και στον timer), αν πρόκειται για διακοπή λόγω stop ή λόγω ack. (error led checked).Στη συγκεκριμένη περίπτωση γίνεται branch στο label ack_pressed, όπου σβήνουμε τη σειρήνα και το πρόγραμμα επιστρέφει .

```
int1handler:
push r17
in r17, SREG
push r17
mov r17,led_manipulator
andi r17,0b00000001
cpi r17,0b00000000
breq ack_pressed
……
ack_pressed:
 ldi r17,0b00000000
out PORTC,r17
pop r17
out SREG, r17
pop r17
 reti
```

Το πρόγραμμα επιστρέφει στο βρόχο αναμονής που το έχουμε εγκλωβίσει , wait_ack και περιμένει το πάτημα του sw6, έτσι ώστε να επιβεβαιωθεί το acknowledge και να αρχίσει να αναβοσβήνει το led1.Όταν γίνει αυτό γίνεται branch στο label siren_stopped , όπου φορτώνουμε τον timer1 με τιμή που κάνει overflow se 1 sec και τον θέτουμε σε λειτουργία(θέτουμε TCCR1B).Αφού γίνει overflow, καλείται η ρουτίνα εξυπηρέτησης του timer1 αυτή τη φορά όμως κάνει branch στο label One_sec. Εκεί ελέγχεται η τιμή του καταχωρητή r20, στην οποία αποθηκεύουμε την κατάσταση του led1 έτσι ώστε να αλλάζει κάθε φορά που εκτελείται η ρουτίνα και να αναβοσβήνει .Ελέγχεται στη συνέχεια αν έχει πατηθεί το πλήκτρο start για έξοδο από τη διαδικασία. Αν δεν έχει πατηθεί φορτώνεται ο timer ξανά , επιστρέφει η ροή στο βρόχο αναμονης και μετράει πάλι ένα δευτερόλεπτο μέχρι να ξαναγίνει interrupt που θα αλλάξει την κατάσταση του led1. Αν πάλι πατηθεί το πλήκτρο start το πρόγραμμα μεταφέρεται στην αρχική λειτουργία εκκίνησης, αφού πρώτα σταματήσουμε τον timer1(μηδενισμός TCCR1B) . Παρακάτω φαίνεται ο κώδικας του timer1handler από το label one_sec και έπειτα.

```
one_sec:
cpi r20,0b11111110
breq close
cpi r20,0b11011110
breq close
open:
ldi r20,0b11111110
andi r20,0b11011111
out PORTB,r20
jmp loop
close:
ldi r20,0b11111111
andi r20,0b11011111
out PORTB,r20
loop:
in r17,PIND
andi r17,0b00000001
cpi r17,0b00000000
breq endd
ldi r17,HIGH(57723)
 out TCNT1H,r17
 ldi r17, LOW(57723)
 out TCNT1L , r17
 ldi r20,0b11111111
ldi r17, (1 << CS10)
out TCCR1B ,r17
```

```
ldi r17 ,(0 << CS11)

out TCCR1B ,r17

ldi r17 ,(1 << CS12)

out TCCR1B ,r17

pop r17

out SREG, r17

pop r17

endd:

reti
```

Τέλος, αν προκληθεί interrupt από το sw3 για διακοπή της λειτουργίας(STOP) , καλείται και πάλι η ρουτίνα εξυπηρέτησης του INT1 , αλλά αυτή τη φορά γίνεται branch sto label stop1.Σβήνουμε τα απαραίτητα leds και συνεχίζουμε να καλούμε τις ρουτίνες ελέγχου βάρους , για να χτυπήσει η σειρήνα σε περίπτωση που πάμε να ξαναξεκινήσουμε τον ιμάντα και η ποσότητα υλικού δεν είναι σωστή. Ελέγχουμε συνέχώς αν πατήθηκε το πλήκτρο start και αν ναι επιστρέφουμε στην λειτουργία εκκίνησης του προγράμματος. Παρακάτω φαίνεται η ρουτίνα εξυπηρέτησης INT1 από το label stop1 και έπειτα.

```
stop1:

ori led_manipulator,0b11010100

out PORTB,led_manipulator

call Input_Pot

call check_A1_to_open_led

call check_Y

wait_return:

in r17, PIND

andi r17,0b00000001

cpi r17,0b00000000

breq reset1

jmp stop1
```

```
reset1:

andi led_manipulator,0b01111111   ;poio labaki na valw

out PORTB,led_manipulator

pop r17

out SREG, r17

pop r17

reti
```