12. Διακοπές – Interrupts (IRQ)

Πίνακας Ι. Χειρισμός διακοπών στον ATmega16.

A/A	Program address	Source	Vector	Interrupt definition
1	$000	RESET		External Pin, Power-on Reset, Brown-outReset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	EXT_INT0_vect	External Interrupt Request 0
3	$004	INT1	EXT_INT1_vect	External Interrupt Request 1
4	$006	TIMER2 COMP	TIM2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	TIM2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	TIM1_CAPT_vect	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	TIM1_COMPA_vect	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	TIM1_COMPB_vect	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	TIM1_OVF_vect	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	TIM0_OVF_vect	Timer/Counter0 Overflow
11	$014	SPI, STC	SPI_STC_vect	Serial Transfer Complete
12	$016	USART, RXC USART, Rx	USART_RXC_vect	Complete
13	$018	USART, UDRE USART	USART_UDRE_vect	Data Register Empty
14	$01A	USART, TXC USART, Tx	USART_TXC_vect	Complete
15	$01C	ADC	ADC_vect	ADC Conversion Complete
16	$01E	EE_RDY	EE_RDY_vect	EEPROM Ready
17	$020	ANA_COMP	ANA_COMP_vect	Analog Comparator
18	$022	TWI	TWSI_vect	Two-wire Serial Interface
19	$024	INT2	EXT_INT2_vect	External Interrupt Request 2
20	$026	TIMER0 COMP	TIM0_COMP_vect	Timer/Counter0 Compare Match
21	$028	SPM_RDY	SPM_RDY_vect	Store Program Memory Ready

Οι AVR υποστηρίζουν διακοπές κατά τις οποίες το πρόγραμμα διακόπτει[1] την κανονική ροή του και εκτελεί κάποιες λειτουργίες που ορίζονται από το διάνυσμα της διακοπής. Οι διακοπές μπορεί να είναι είτε εξωτερικές είτε εσωτερικές. Οι εξωτερικές διακοπές προέρχονται από σήματα από εξωτερικές πηγές οι οποίες είναι συνδεδεμένες σε εισόδους του μικροελεγκτή ενώ οι εσωτερικές από τα εσωτερικά περιφερειακά του μικροελεγκτή. Κάθε διακοπή έχει ένα διάνυσμα στη μνήμη προγράμματος. Η διακοπή, το διάνυσμα και η διεύθυνσή του παρουσιάζονται στον Πίνακα Ι .Όσο μικρότερη είναι η διεύθυνση του

[1] *Με τη διακοπή RESET το πρόγραμμα επανεκκινεί ενώ με τις υπόλοιπες συνεχίζει από το σημείο που σταμάτησε κατά την έναρξη της διακοπής.*

διανύσματος, τόσο μεγαλύτερη είναι η προτεραιότητα της διακοπής. Από το διάνυσμα της διακοπής διευθυνσιοδοτείται ο **Program Counter** για να εκτελέσει τη ρουτίνα της διακοπής.

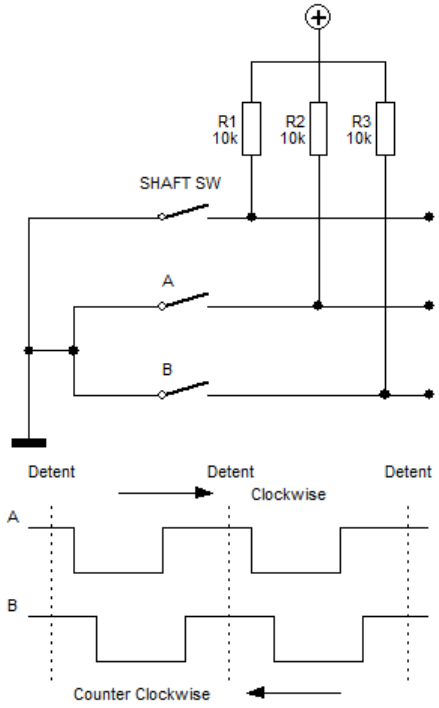
Οι διακοπές έχουν το δικό τους bit ενεργοποίησης στον αντίστοιχο καταχωρητή κατάστασής τους, το οποίο πρέπει να γίνει **1** προκειμένου να είναι διαθέσιμη να προκληθεί από την αιτία της και ένα γενικό bit[1] για όλες τις διακοπές, το οποίο πρέπει και αυτό να γίνει **1**. Αν υπάρξει η αιτία για μια διακοπή γίνεται **1** η σημαία της διακοπής και αν είναι ενεργοποιημένες οι διακοπές από το γενικό bit, ξεκινάει να εκτελείται και καθαρίζεται η σημαία. Στη συνέχεια οι διακοπές απενεργοποιούνται[2] καθαρίζοντας το γενικό bit και ενεργοποιούνται ξανά όταν το πρόγραμμα βγει από τη διακοπή. Αν δεν είναι ενεργοποιημένες οι διακοπές, για οποιοδήποτε λόγο, τότε παραμένει σε αναμονή με τη σημαία της **1** έως ότου ενεργοποιηθούν. Αν εν τω μεταξύ υπάρξουν αιτίες και για άλλες διακοπές, θα γίνουν οι σημαίες τους **1** και θα είναι όλες σε αναμονή. Η διακοπή που θα εκτελεστεί πρώτη είναι αυτή με την υψηλότερη προτεραιότητα.

Οι εξωτερικές διακοπές μπορούν να προγραμματιστούν ώστε να προκληθούν από μέτωπο παλμού ή από χαμηλή λογική κατάσταση. Οι διακοπές **INT0** και **INT1** όταν είναι ενεργοποιημένες ως διακοπές μετώπου παλμού, απαιτούν την παρουσία ρολογιού (I/O clock), ενώ όταν είναι ενεργοποιημένες ως διακοπές χαμηλής λογικής η διακοπή θα δρα όσο υπάρχει χαμηλή λογική στην είσοδο και δεν απαιτούν την παρουσία ρολογιού. Εκτελούνται ακόμα και αν ο μικροελεγκτής βρίσκεται σε sleep mode όπου δεν υπάρχει λειτουργία ρολογιού και για αυτό μπορούν να χρησιμοποιηθούν για να τον επαναφέρουν από αυτή την κατάσταση. Η διακοπή **INT2** μπορεί να προγραμματιστεί μόνο ως διακοπή μετώπου παλμού και δρα ασύγχρονα.

[1] *Στον καταχωρητή SREG το Bit 7 με ονομασία I: Global Interrupt Enable.*

[2] *Αν θέλουμε να είναι ενεργοποιημένες ώστε να είναι δυνατό να δράσει μια άλλη διακοπή κατά τη διάρκεια εκτέλεσης των εντολών της διακοπής που έχει ήδη σε εξέλιξη, πρέπει να ενεργοποιήσουμε ξανά τις διακοπές με την εντολή* `sei()`.

13. Rotary Encoder (ROT)



Ο Rotary encoder είναι ένας μηχανικός κωδικοποιητής περιστροφής incremental encoder[1]. Αποτελείται από ένα άξονα, ο οποίος περιστρέφεται και δύο διακόπτες οι οποίοι κλείνουν – ανοίγουν με μια συγκεκριμένη σειρά που εξαρτάται από τη φορά περιστροφής του άξονα. Ο άξονας έχει ένα μηχανισμό με θέσεις συγκράτησης (detents) κατά την περιστροφή. Οι δύο διακόπτες (A, B) έχουν το ένα άκρο τους κοινό και το άλλο ελεύθερο. Το κοινό άκρο συνδέεται συνήθως στη γείωση και τα ελεύθερα άκρα A και B σε εισόδους του μικροελεγκτή με ενεργοποιημένες τις εσωτερικές pull up αντιστάσεις ή με συνδεδεμένες εξωτερικές. Συνήθως υπάρχει και ένας τρίτος ανεξάρτητος διακόπτης που κλείνει με πάτημα του άξονα και έχει δύο ελεύθερους ακροδέκτες, οι οποίοι συνδέονται με τον ίδιο τρόπο, ο ένας στη γείωση και ο άλλος στον μικροελεγκτή. Αν αρχίσουμε να περιστρέφουμε τον άξονα συνεχώς, οι δύο διακόπτες θα παράγουν τετραγωνικούς παλμούς με διαφορά φάσης. Όταν ο αριθμός των θέσεων συγκράτησης είναι ίσος με τον αριθμό των παλμών, τότε έχουμε τον έναν τύπο ROT (detents = pulses) όπου στις θέσεις συγκράτησης οι διακόπτες είναι πάντα ανοιχτοί. Όταν ο αριθμός των θέσεων συγκράτησης είναι διπλάσιος από τον αριθμό των

[1] *Η άλλη κατηγορία μηχανικών κωδικοποιητών είναι absolute encoder.*

παλμών, έχουμε το δεύτερο τύπο ROT (detents = 2*pulses) όπου στις θέσεις συγκράτησης εναλλάσσονται είτε και οι δύο ανοιχτοί είτε και οι δύο κλειστοί. Στο σχήμα παρουσιάζεται ο πρώτος τύπος ROT όπου δείχνονται οι θέσεις συγκράτησης. Για τον δεύτερο τύπο υπάρχουν και ενδιάμεσες θέσεις.

Στις περισσότερες εφαρμογές η κίνηση του ROT αυξάνει ή ελαττώνει ένα μετρητή, ο οποίος χρησιμοποιείται στη συνέχεια σε κάποια λειτουργία. Το λογισμικό αναγνωρίζει την κίνηση του άξονα αν είναι κατά τη φορά δεικτών του ρολογιού (CW) ή αντίθετη (CCW). Η αναγνώριση γίνεται ελέγχοντας την κατάσταση των A, B. Ο πιο ασφαλής τρόπος είναι τα A, B να προκαλούν το καθένα εξωτερική διακοπή στον μικροελεγκτή και η ρουτίνα της διακοπής να διαχειρίζεται τη μεταβολή του μετρητή. Στη συνέχεια παρουσιάζεται ένα παράδειγμα χρήσης ROT γραμμένο σε γλώσσα προγραμματισμού C:

```
#include <avr/interrupt.h>
volatile int i1;
// Has to be "volatile" if other routine than ISR(INTx_vect) affects "reg1bin"

void InitROT (void)                         // Start up, initialization routine
{
    MCUCR |= (1<<ISC01);                    // Interrupt by the falling edge INT0
    MCUCR |= (1<<ISC11);                    // Interrupt by the falling edge INT1
    GICR |= (3<<6);                         // Enable INT0, INT1 interrupt
    sei();                                  // Enable interrupts
    DDRD &= ~((1<<2)|(1<<3));               // Port D1, D2 inputs
    PORTD |= (3<<PORTD2);                   // Port D1, D2 Enable pull up
}

ISR(INT0_vect)      //INT0, PD2 Rot A, CW Increase, start moving (A->0, B=1)
{
    //sei();                                // Enable interrupts (if necessary)
    //cli();                                // Disable interrupts (if necessary)
    _delay_ms(1);
    while (!(bit_is_set(PIND,2)));          // A=0
    _delay_ms(1);
    if ((bit_is_set(PIND,2)) && (!(bit_is_set(PIND,3))))      // A=1, B=0
    i1++;                                   // Increase
    while (!(bit_is_set(PIND,3)));          // B=0
    _delay_ms(1)
}

ISR(INT1_vect)      //INT1, PD3 Rot B, CCW Decrease, start moving (B->0, A=1)
{
    //sei();                                // Enable interrupts (if necessary)
    //cli();                                // Disable interrupts (if necessary)
    _delay_ms(1);
    while (!(bit_is_set(PIND,3)));          // B=0
    _delay_ms(1);
    if ((bit_is_set(PIND,3)) && (!(bit_is_set(PIND,2))))      // B=1, A=0
    i1--;                                   // Decrease
    while (!(bit_is_set(PIND,2)));          // A=0
    _delay_ms(1);
}
```

Οι έξοδοι A, B του ROT είναι συνδεδεμένοι στις εισόδους **INT0**, **INT1** (PD2, PD3) του μικροελεγκτή. Η βιβλιοθήκη **avr/interrupt** είναι απαραίτητη για τη λειτουργία των διακοπών και η μεταβλητή **i1** πρέπει να είναι volatile αν θέλουμε να υπάρχει δυνατότητα να αλλάξει η τιμή της και από άλλες ρουτίνες. Η ρουτίνα **InitROT(void)** ενεργοποιεί τις διακοπές στο κατερχόμενο μέτωπο και προγραμματίζει τα PD2, PD3 εισόδους με ενεργοποιημένες τις εσωτερικές pull up αντιστάσεις. Οι δύο επόμενες ρουτίνες, **ISR(INT0_vect)** και **ISR(INT1_vect)**, εκτελούνται με την αντίστοιχη διακοπή.

Αν περιστρέψουμε τον άξονα κάποιο από τα A, B θα περάσει πρώτο σε λογικό **0** και το κατερχόμενο μέτωπο θα προκαλέσει τη διακοπή **INT0** ή **INT1** ανάλογα με τη φορά περιστροφής. Αν το A προκαλέσει τη διακοπή αρχίζουν να εκτελούνται οι εντολές της ρουτίνας **ISR(INT0_vect)**. Σε περίπτωση που θέλουμε να υπάρχει δυνατότητα κάποια άλλη διακοπή να εμπλακεί μπορούμε να ενεργοποιήσουμε τις διακοπές (στο παράδειγμα η εντολή είναι σε σχόλιο). Εκτελείται αρχικά μια καθυστέρηση για τους παρασιτικούς παλμούς του διακόπτη και στη συνέχεια το πρόγραμμα παραμένει στη **while** για όσο το A βρίσκεται σε λογικό **0**. Καθώς συνεχίζει να περιστρέφεται ο άξονας το B θα περάσει σε λογικό **0** και αυτό πριν το A περάσει σε λογικό **1** ξανά. Όταν το A περάσει σε λογικό **1** το πρόγραμμα βγαίνει από τη **while** και εκτελείται μια καθυστέρηση για τους παρασιτικούς παλμούς. Αν το A είναι σε λογικό **1** και το B σε λογικό **0** επιβεβαιώνεται τότε η φορά περιστροφής CW και αυξάνει ο μετρητής **i1**. Το πρόγραμμα περιμένει να περάσει και το B σε λογικό **1**, εκτελείται μια καθυστέρηση για τους παρασιτικούς παλμούς, ενεργοποιούνται ξανά οι διακοπές αν είχαν απενεργοποιηθεί και βγαίνει από τη ρουτίνα διακοπής δίνοντας τον έλεγχο στο κυρίως πρόγραμμα. Αν η περιστροφή είναι αντίθετη (CCW), το B θα προκαλέσει τη διακοπή **INT1** και θα αρχίσει να εκτελείται η ρουτίνα **ISR(INT1_vect)**, η οποία εκτελεί ίδιες εντολές που όμως το A βρίσκεται στη θέση του B και το αντίθετο και ελαττώνει τον μετρητή **i1**.