Εργαστήριο 9

Ονομα: Μαυρογιώργης Δημήτρης
ΑΜ: 2016030016
ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργατών
Πολυτεχνειο Κρητης

April 19, 2021

Σκοπός

Σκοπός του έβδομου και ένατου εργαστηρίου είναι να γράψουμε κώδικα που να αφαιρεί την αναπήδηση από έναν διακόπτη SPDT με δύο διαφορετικούς τρόπους: α) με δειγματοληψία της εισόδου αρκετά συχνά, δηλαδή με polling βασισμένο στον TIMER0, καθώς και β) με εξωτερικά interrupts

Debouncing Χωρίς Interrupts – με Polling βασισμένο στον TIMER0

Αρχικά, δηλώθηκαν κάποια macro, για να ελέγχουμε αν καποιο bit είναι set ή clear, να θέτουμε κάποιο bit σε "0" ή "1" και για εναλλαγή κάποιου bit από "0" σε "1" και το αντίστροφο. Στο main πρόγραμμα αρχικοποιούμε τον TIMER0, ενεργοποιούμε τα interrupt και τέλος έχουμε ένα while loop στο οποίο δεν κάνουμε τίποτα, καθώς όλα θα γίνονται στον handler του TIMER0.

```c
#include <avr/io.h>
#include <avr/interrupt.h>

#define is_bit_set(byte,bit)    (byte & (1 << bit))
#define is_bit_clear(byte,bit)  (!(byte & (1 << bit)))
#define set_bit(byte,bit)   ((byte) |=  (1<<(bit)))
#define clear_bit(byte,bit) ((byte) &= ~(1<<(bit)))
#define flip_bit(byte,bit)  ((byte) ^=  (1<<(bit)))

#define TRUE 0x01
#define FALSE 0x00

uint8_t count_changes = 0;
uint8_t out_val = 0;

/* The 2 pins of SPDT switch */
uint8_t prev_spdt_pa0 = 0b00000001;
uint8_t prev_spdt_pa1 = 0b00000000;
```

(a) C code for defines

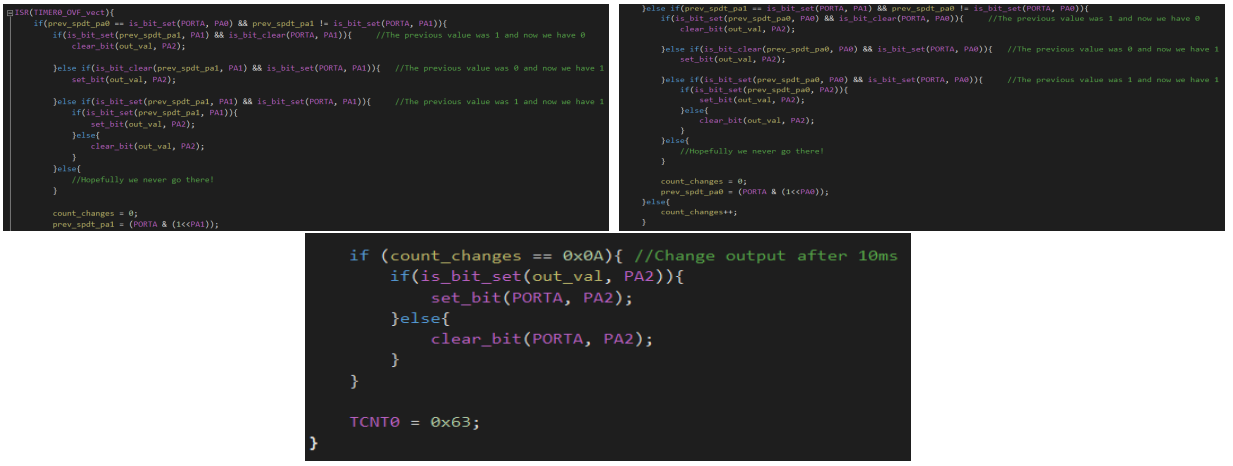
```c
int main(void){
    cli();
    TCNT0_INIT();
    sei();
    PORTA |= prev_spdt_pa0;
    PORTA |= prev_spdt_pa1;
    while(1){
    }
}
```

(b) C code for main

Για το σκοπό της συγκεκριμένης υλοποίησης χρησιμοποιήθηκε ο TIMER0 του 1 ms από τα προηγούμενα εργαστήρια. Στο service routine του TIMER0 κάνουμε τη δειγματοληψία της εισόδου κάθε 1 ms και κάθε φορά αποθηκεύουμε τις τιμές έτσι, ώστε να τις συγκρίνουμε με τις επόμενης περιόδου. Επιπλέον, εκτός από την αποθήκευση, ελέγχουμε τις προηγούμενες τιμές με τις τωρινές και ποιο bit άλλαξε από την προηγούμενη περίοδο, για να δούμε ποιος ακροδέκτης του SPDT έχει τις αναπηδήσεις, και αναλόγως θέτουμε τη έξοδο 0 ή 1.

Ειδικότερα, αν άλλαξε το bit PA1, ελέγχουμε ποια ήταν η προηγούμενη τιμή του. Αν άλλαξε από "1" σε "0". Τότε θερωρούμε ότι σε αυτό τον κύκλο έκανε bounce η είσοδος PA1 και θέτουμε το bit PA2 στο "0". Στην αντίστροφη, περίπτωση διατηρούμε την έξοδο "1". Αν τώρα έχουμε και στις δύο εισόδους "1", απλά αποθηκεύουμε ως έξοδο την προηγούμενη τιμή. Τέλος, η κατάσταση "0" και στους δύο ακροδέκτες δεν είναι δυνατή, γιατί δεν γίνεται να είναι ενεργοποιημένοι και οι δύο ακροδέκτες του SPDT ταυτόχρονα. Ομοίως με τα παραπάνω γίνονται οι ίδιο ελεγχοι για το αν άλλαξε το PA0 τιμή από την προηγούμενη δειγματοληψία.

Τέλος, αν δεν άλλαξε κάποιος ακροδέκτης τιμή, απλώς αυξάνουμε την τιμή ενός counter και μόλις η τιμή του φτάσει το 0x0A, δηλαδή έχουν περάσει 10 ms, βγάζουμε την έξοδο.



```
ISR(TIMER0_OVF_vect){
    if(prev_spdt_pa0 == is_bit_set(PORTA, PA0) && prev_spdt_pa1 != is_bit_set(PORTA, PA1)){
        if(is_bit_set(prev_spdt_pa1, PA1) && is_bit_clear(PORTA, PA1)){     //The previous value was 1 and now we have 0
            clear_bit(out_val, PA2);

        }else if(is_bit_clear(prev_spdt_pa1, PA1) && is_bit_set(PORTA, PA1)){   //The previous value was 0 and now we have 1
            set_bit(out_val, PA2);

        }else if(is_bit_set(prev_spdt_pa1, PA1) && is_bit_set(PORTA, PA1)){     //The previous value was 1 and now we have 1
            if(is_bit_set(prev_spdt_pa1, PA1)){
                set_bit(out_val, PA2);
            }else{
                clear_bit(out_val, PA2);
            }
        }else{
            //Hopefully we never go there!
        }

        count_changes = 0;
        prev_spdt_pa1 = (PORTA & (1<<PA1));
    }else if(prev_spdt_pa1 == is_bit_set(PORTA, PA1) && prev_spdt_pa0 != is_bit_set(PORTA, PA0)){
        if(is_bit_set(prev_spdt_pa0, PA0) && is_bit_clear(PORTA, PA0)){     //The previous value was 1 and now we have 0
            clear_bit(out_val, PA2);

        }else if(is_bit_clear(prev_spdt_pa0, PA0) && is_bit_set(PORTA, PA0)){   //The previous value was 0 and now we have 1
            set_bit(out_val, PA2);

        }else if(is_bit_set(prev_spdt_pa0, PA0) && is_bit_set(PORTA, PA0)){     //The previous value was 1 and now we have 1
            if(is_bit_set(prev_spdt_pa0, PA0)){
                set_bit(out_val, PA2);
            }else{
                clear_bit(out_val, PA2);
            }
        }else{
            //Hopefully we never go there!
        }

        count_changes = 0;
        prev_spdt_pa0 = (PORTA & (1<<PA0));
    }else{
        count_changes++;
    }

    if (count_changes == 0x0A){ //Change output after 10ms
        if(is_bit_set(out_val, PA2)){
            set_bit(PORTA, PA2);
        }else{
            clear_bit(PORTA, PA2);
        }
    }

    TCNT0 = 0x63;
}
```

Figure 2: C code for TIMER0 inerrupt handler

Προσομοίωση Αποτελεσμάτων

Αρχικά, προσομοιώνουμε τα bounches και από 1-1 που είναι οι δύο είσοδοι, γίνονται 1-0 και στη συνέχεια ξανά 1-1. Όπως φαίνεται στις δύο πρώτες εικόνες, η έξοδος από "0" γίνεται "1", αφού κατασταλάξει η έξοδος και περάσουν τα 10 ms. Ομοίως, στις επόμενες δύο εικόνες, η προσομοίωση γίνεται με περισσότερα bounches από την αρχική και βλέπουμε ότι η έξοδος από "1" που ήταν, άλλαξε σε "0" μετά από επίσης 10 ms και αφού σταμάτησαν τα bounches.

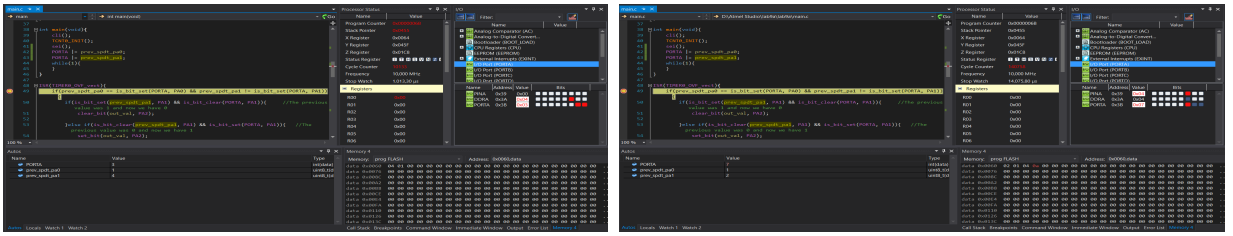


Figure 3: Results from Atmel Studio 7 - Change from "0" to "1"

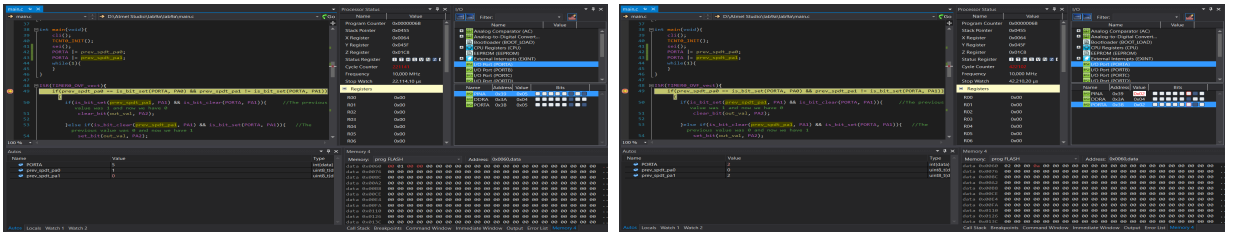


Figure 4: Results from Atmel Studio 7 - Change from "1" to "0"

Debouncing με Interrupts

Ομοίως με την περίπτωση χωρίς external interrupt, δηλώθηκαν κάποια macro, για να ελέγχουμε αν καποιο bit είναι set ή clear, να θέτουμε κάποιο bit σε "0" ή "1" και για εναλλαγή κάποιου bit από "0" σε "1" και το αντίστροφο. Επιπλέον, για τη χρήση external interrupt ενεργοποιούμε τα bit INT0 και INT1 στον καταχωρητή GICR, ενώ για την ανίχνευση των αναπηδήσεων, χρειαζόμαστε να ανιχνεύουμε falling edges. Γι' αυτό θέτουμε σε "1" τα bit ISC11 και ISC01.

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 10000000

#define is_bit_set(byte,bit)    (byte & (1 << bit))
#define is_bit_clear(byte,bit)  (!(byte & (1 << bit)))
#define set_bit(byte,bit)   ((byte) |=  (1<<(bit)))
#define clear_bit(byte,bit) ((byte) &= ~(1<<(bit)))
#define flip_bit(byte,bit)  ((byte) ^=  (1<<(bit)))

#define out_val 0b00000100
```

(a) C code for defines

```c
void INIT_EXT_INTERRUPT(){
    GICR |= (1<<INT0)|(1<<INT1);    //Enable external interrupt INT0, INT1
    MCUCR = (1<<ISC11)|(1<<ISC01); //Falling edge will generate an interrupt for both inputs PD2 and PD3
    DDRA |= (1<<PA2);
    return;
}

int main(void){
    cli();
    INIT_EXT_INTERRUPT();
    sei();
    PORTA = out_val;
    while(1){

    }
}
```

(b) C code for main

Για την αλλαγή ανίχνευση των αναπηδήσεων, ελέγχουμε μόλις έρθει κάποιο falling edge την τιμή της άλλης εισόδου, πχ για το INT0, το οποίο εκτελείται όταν βρεθέι κάποιο falling edge στο PD2 έχουμε δύο περιπτώσεις: Αν το PD3 ήταν "1" τότε σημαίνει ότι τα bounches βρίσκονται στο PD2. Οπότε κάνουμε την έξοδο "0". Αν τώρα το PD3 είναι "0" σε κάποιο falling edge, σημαίνει ότι στο PD2 υπήρξε μια αυξομείωση τάσης και σε αυτό το σενάριο διατηρούμε "1" την έξοδο. Ομοίως, και στον δεύτερο handler ελέγχουμε για τις ίδιες περιπτώσεις με μοναδική διαφορά ότι σε αυτή την περίπτωση αν το PD2 είναι "1" και έχουμε falling edge στο PD3, τότε διατηρούμε την έξοδο "1", αλλιώς την κάνουμε "0".

```c
ISR(INT0_vect){
    if(is_bit_set(PORTD,PD3)){
        clear_bit(PORTA,PA2);

    }else if(is_bit_clear(PORTD,PD3)){
        set_bit(PORTA,PA2);

    }else{
        // Hopefully never goes there
    }
}
```

```c
ISR(INT1_vect){
    if(is_bit_set(PORTD,PD2)){
        set_bit(PORTA,PA2);

    }else if(is_bit_clear(PORTD,PD2)){
        clear_bit(PORTA,PA2);

    }else{
        // Hopefully never goes there
    }
}
```

Figure 6: C code for the two external interrupt handlers

Προσομοίωση Αποτελεσμάτων

Σε αυτή την περίπτωση βλέπουμε ότι με το που ανιχνευτεί κάποιο falling edge εκτελείται ο αντίστοιχος handler και ελέγχει σε ποια περίπτωση βρισκόμαστε από αυτες που αναλύθηκαν παραπάνω. Στις πρώτες δύο εικόνες, βλέπουμε ότι η έξοδος από "1" άλλαξε σε "0" μετά από περίπου 25000 κύκλους, όπως συμβαίνει και στο stimuli file όπου για 25000 η έξοδος είναι "1" και έπειτα αλλάζει σε "0". Ομοίως, στις επόμενες δύο εικόνες βλέπουμε ότι εκτελείται ο δεύτερος handler και η έξοδος αλλάζει από "0" σε "1". Επομένως, βλέπουμε ότι με αυτό τον τρόπο η ανίχνευση των bounches είναι γρηγορότερη σε σχέση με τη μεθοδο του polling και TIMER0, καθώς με το που βρεθέι κάποιο falling edge που σημαίνει κατ' επέκταση αλλαγή κάποιας εισόδου, μπορούμε να αλλάξουμε απευθείας την έξοδο ανάλογα με τις τιμές που έχουν αυτές οι είσοδοι.

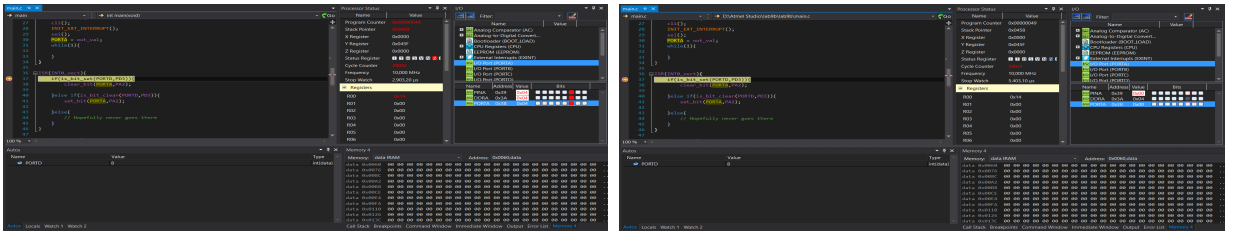


Figure 7: Results from Atmel Studio 7 - Change from "0" to "1"

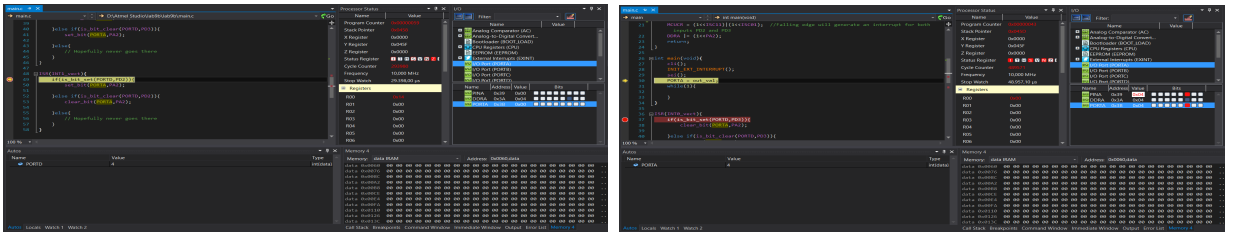


Figure 8: Results from Atmel Studio 7 - Change from "1" to "0"