Ε.Μ.Π. - ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

ΑΘΗΝΑ, 30 Νοεμβρίου 2022

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ
ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"
Χρήση πληκτρολογίου 4×4 σε θύρα επέκτασης στον AVR

Αναφορά 6ης Εργαστηριακής Άσκησης

Ραπτόπουλος Πέτρος (el19145)
Σαφός Κωνσταντίνος (el19172)

Ζήτημα 6.1:

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε C

```
#define F_CPU 16000000UL
#include <math.h>
#include <util/delay.h>
#include <avr/io.h>

#define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
#define TWI_READ 1             // reading from twi device
#define TWI_WRITE 0            // writing to twi device
#define SCL_CLOCK 100000L     // twi clock in Hz
//Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
#define TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2

// PCA9555 REGISTERS
typedef enum {
    REG_INPUT_0 = 0,
    REG_INPUT_1 = 1,
    REG_OUTPUT_0 = 2,
    REG_OUTPUT_1 = 3,
    REG_POLARITY_INV_0 = 4,
    REG_POLARITY_INV_1 = 5,
    REG_CONFIGURATION_0 = 6,
    REG_CONFIGURATION_1 = 7,
} PCA9555_REGISTERS;

//----------- Master Transmitter/Receiver -------------------
#define TW_START 0x08
#define TW_REP_START 0x10
//---------------- Master Transmitter ----------------------
#define TW_MT_SLA_ACK 0x18
#define TW_MT_SLA_NACK 0x20
#define TW_MT_DATA_ACK 0x28
//---------------- Master Receiver ----------------
#define TW_MR_SLA_ACK 0x40
#define TW_MR_SLA_NACK 0x48
#define TW_MR_DATA_NACK 0x58

#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)

// send one byte divided into 2 (4 bit) parts
void write_2_nibbles(char data) {
    char pinState = PIND; // read 4 LSB and resend them
                          // in order not to alter any previous state
    PORTD = (pinState & 0x0F) | (data & 0xF0) | (1<<PD3); // send 4MSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    PORTD = (pinState & 0x0F) | ((data<<4) & 0xF0) | (1<<PD3); // send 4LSB
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
}

// send one byte of data to the lcd display
void lcd_data(char data) {
    PORTD |= (1<<PD2);  // select data register
    write_2_nibbles(data);
    _delay_us(100);
}

// send one byte of instuction to the lcd display
void lcd_command(char data) {
    PORTD &= (0xFF) & (0<<PD2);  // select command register
    write_2_nibbles(data);
    _delay_us(100);
}
```

```
void lcd_init() {
    _delay_ms(40);              // lcd init procedure
    PORTD = 0x30 | (1<<PD3);    // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x30 | (1<<PD3);    // 8 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    PORTD = 0x20 | (1<<PD3);    // change to 4 bit mode
    PORTD &= (0xFF) & (0<<PD3); // set PD3 to zero, lcd enable pulse
    _delay_us(100);
    lcd_command(0x28); // select character size 5x8 dots and two line display
    lcd_command(0x0c); // enable lcd, hide cursor
    lcd_command(0x01); // clear display
    _delay_us(5000);
    lcd_command(0x06); // enable auto increment of address, disable shift of the display
}

//initialize TWI clock
void twi_init(void) {
    TWSR0 = 0; // PRESCALER_VALUE=1
    TWBR0 = TWBR0_VALUE; // SCL_CLOCK 100KHz
}

// Read one byte from the twi device ( request more data from device)
unsigned char twi_readAck(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

// Issues a start condition and sends address and transfer direction.
// return 0 = device accessible, 1= failed to access device
unsigned char twi_start(unsigned char address) {
    uint8_t twi_status;
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);        // send START condition
    while(!(TWCR0 & (1<<TWINT)));                       // wait until transmission completed
    twi_status = TW_STATUS & 0xF8;                      // check value of TWI Status Register.
    if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
    TWDR0 = address;                                    // send device address
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));   // wail until transmission completed and ACK/NACK has been received
    twi_status = TW_STATUS & 0xF8;  // check value of TWI Status Register.
    if ((twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK)) {
        return 1;
    }
    return 0;
}

// Send start condition, address, transfer direction.
// Use ack polling to wait until device is ready
void twi_start_wait(unsigned char address) {
    uint8_t twi_status;
    while (1) {
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);    // send START condition
    while(!(TWCR0 & (1<<TWINT)));                  // wait until transmission completed
    twi_status = TW_STATUS & 0xF8;                 // check value of TWI Status Register.
    if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;
    TWDR0 = address;   // send device address
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));      // wail until transmission completed
    twi_status = TW_STATUS & 0xF8;     // check value of TWI Status Register.
    if ((twi_status == TW_MT_SLA_NACK ) || (twi_status ==TW_MR_DATA_NACK)) {
        //device busy, send stop condition to terminate write operation
        TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        while(TWCR0 & (1<<TWSTO)); // wait until stop condition is executed and bus released
        continue;
    }
    break;
    }
}
```

```
// Send one byte to twi device, Return 0 if write successful or 1 if write failed
unsigned char twi_write(unsigned char data) {
    TWDR0 = data;                    // send data to the previously addressed device
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));   // wait until transmission completed
    if ((TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
    return 0;
}

// Send repeated start condition, address, transfer direction
//Return: 0 device accessible
// 1 failed to access device
unsigned char twi_rep_start(unsigned char address) {
    return twi_start(address);
}

// Terminates the data transfer and releases the twi bus
void twi_stop(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO); // send stop condition
    while(TWCR0 & (1<<TWSTO));                   // wait until stop condition is executed and bus released
}

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value) {
    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_write(value);
    twi_stop();
}

uint8_t twi_readNak(void) {
    TWCR0 = (1<<TWINT) | (1<<TWEN);
    while(!(TWCR0 & (1<<TWINT)));
    return TWDR0;
}

uint8_t PCA9555_0_read(PCA9555_REGISTERS reg) {
    uint8_t ret_val;
    twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
    twi_write(reg);
    twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);
    ret_val = twi_readNak();
    twi_stop();
    return ret_val;
}

char scan_row(int row) { // row = 0, 1, 2, 3 for IO1_0, IO1_1, IO1_2, IO1_3
    // we set the row we are scanning to 0, while all the rest
    // are set to 1. if a key is pressed the correspondent column
    // would switch to 0. Otherwise it is 1.
    if(row == 0) PCA9555_0_write(REG_OUTPUT_1, 0x0E);
    else if(row == 1) PCA9555_0_write(REG_OUTPUT_1, 0x0D);
    else if(row == 2) PCA9555_0_write(REG_OUTPUT_1, 0x0B);
    else if(row == 3) PCA9555_0_write(REG_OUTPUT_1, 0x07);
    return PCA9555_0_read(REG_INPUT_1) & 0xF0; // read IO1[7:4]
}

void scan_keypad(char* keypad_state) {
// keypad_state holds the current state of all rows
    for(int i = 0; i <= 3; i++) // scan each row
        keypad_state[i] = scan_row(i);
}

int diff_bit(char c1, char c2) {   // c1 holds the old(current) state, c2 holds the new state
    char diff = c1^c2;  // xor of c1, c2  we have 0 if
    char mask = 0x10;   // the ith bit of c1 and c2 are same
    for (int i = 0; i <= 3; ++i) {
        if(((diff & mask) != mask) && ((c2 & mask) == 0))
// we check c2 & mask == 0 in order to be notified only when the button is pressed. If unpressed ignore.
            return i; // different bit found
        mask = mask << 1;   // shift the mask to the left
    }
    return -1; // same
}
```

```
int same(char* array1, char* array2) { // array1, array2 have size 4
    for(int i = 0; i <= 3; ++i)
        if (array1[i]!=array2[i])
            return 0;
    return 1;
}

char keypad_state_current[4];   // hold the current keypad state
int scan_keypad_rising_edge() {
    char keypad_state_1[4];
    char keypad_state_2[4];
    // read two times the keypad, if the states are different then
    // bouncing is under way.
    do {
        scan_keypad(keypad_state_1);
        _delay_ms(15);
        scan_keypad(keypad_state_2);
    } while(same(keypad_state_1, keypad_state_2) == 0);

    int ret = -1;
    for(int i = 0; i <= 3; ++i) {
        int diff = diff_bit(keypad_state_current[i], keypad_state_1[i]);
        // return which bit is different between current and previous keypad state
        keypad_state_current[i] = keypad_state_1[i]; // update current keypad_state
        if (diff != -1) ret = diff + 4*i;            // the place inside chars[] array
    }
    // 0: '*', 1: '0', 2: '#', 3: 'D', 4:'7', ...
    return ret; // same keypad states
}

char keypad_to_ascii() {
    int scan = scan_keypad_rising_edge();
    if(scan == -1) return 0; // no buttons are pressed
    char chars[16] = {'*', '0', '#', 'D', '7', '8', '9', 'C', '4', '5', '6', 'B', '1', '2', '3', 'A'};
    // the sequence of the above chars is the result of the way we scan the keypad
    return chars[scan];
}

int main() {
    DDRD = 0xFF;    // set portD as output (LCD)
    lcd_init();     // init lcd
    _delay_ms(2);   // wait for lcd init
    twi_init();
       PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1_0 as output, 4-7 as input
    while(1) {
        char lcd_state = keypad_to_ascii();
        if(lcd_state != 0) {
            lcd_command(0x01);  // clear display
            _delay_us(2000);
            lcd_data(lcd_state); // display character
        }
        // we want to display the last character pressed
        // if no button is pressed right now display the last character pressed
    }
}
```

Ζήτημα 6.2:

Παρακάτω φαίνεται ο κώδικας σε C που υλοποιεί τα ζητούμενα της άσκησης.

Η ορθή λειτουργία των προγραμμάτων έχει ελεγχθεί στο περιβάλλον προσομοίωσης MPLAB X, καθώς και στην αναπτυξιακή πλακέτα του εργαστηρίου.

Σημείωση: Ο ακριβής τρόπος λειτουργίας του προγράμματος υποδεικνύεται μέσω σχολίων σε εντολές του κώδικα.

Κώδικας σε C

```
int main() {
    DDRB = 0xFF;    // set portD as output (LCD)
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1_0 as output, 4-7 as input
    char code_1stDigit = '5';
    char code_2ndDigit = '9';
    char pressed_1stDigit = ' '; // init
    char pressed_2ndDigit = ' ';
    int i = 1; // 1st digit
    while(1) {
        char temp = keypad_to_ascii();
        if(temp != 0) { // when some key is pressed
            char state = temp;
            do { // while pressed do nothing
                temp = keypad_to_ascii();
                _delay_ms(10);
            } while(temp == state);
            if(i == 1) {
                pressed_1stDigit = state;
                i = 2;
            } else if (i == 2) {
                pressed_2ndDigit = state;
                i = 1;
                if(code_1stDigit == pressed_1stDigit && code_2ndDigit == pressed_2ndDigit) {
                    PORTB = 0x3F;
                    _delay_ms(4000);
                    PORTB = 0x00;
                    _delay_ms(1000);
                } else { // wrong password
                    // light up the leds
                    for (int j = 0; j < 10; j++) {  // 2 Hz => T = 0.5 sec, DC = 50% => 0.25 sec leds on
                       PORTB = 0x3F;
                       _delay_ms(250);
                       PORTB = 0x00;
                       _delay_ms(250);
                    }
                }
            }
        }
    }
}
```

Σημείωση: Οι υπόλοιπες βοηθητικές συναρτήσεις ταυτίζονται με αυτές του ζητήματος 6.1. Συνεπώς δεν ξαναπαρουσιάζονται στο τρέχον ζητούμενο.

Διάγραμμα Ροής