ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΑΚΑΔ. ΕΤΟΣ 2022-2023

Αναφορά

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Χρήση πληκτρολογίου 4×4 σε θύρα επέκτασης στον AVR

Ομάδα 1

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197

Ζήτημα 6.1

Στην άσκηση αυτή χρησιμοποιούμε το ολοκληρωμένο επέκτασης θυρών PCA9555 για έλεγχο του 4x4 πληκτρολογίου και εμφανίζουμε στην οθόνη το χαρακτήρα που αντιστοιχεί στο πλήκτρο που πατήθηκε τελευταίο.

Ορίζουμε τον πίνακα με τους χαρακτήρες του 4x4 πληκτρολογίου. Στη συνέχεια, στη συνάρτηση scan_row, όπως ζητείται από την εκφώνηση ελέγχουμε μία γραμμή του πληκτρολογίου ανά εκτέλεση της συνάρτησης. Η συνάρτηση αυτή παίρνει ως όρισμα τη γραμμή που θέλουμε να ελέγξουμε. Αν για παράδειγμα δώσουμε σαν όρισμα τον αριθμό 2, θα ελεγχθεί η γραμμή 2, δίνοντας στη θύρα 1 του PCA9555 το 00001101. Έτσι, θα τεθεί σε χαμηλό το bit εξόδου που αντιστοιχεί στην γραμμή 2. Στη συνέχεια διαβάζουμε την θύρα 1 του PCA9555 ώστε να καταλάβουμε από τα 4 MSB αν είχε πατηθεί κάποιο πλήκτρο (και αν ναι, ποιο). Επιστρέφουμε την στήλη στην οποία αντιστοιχεί το πλήκτρο το οποίο πατήθηκε (1-4) ή 0 αν δεν πατήθηκε κανένα.

Στη συνάρτηση scan_keypad διαβάζουμε διαδοχικά τις σειρές σε μια επανάληψη και αν η scan_row επιστρέψει τιμή διάφορη του 0 (δηλαδή όταν πατηθεί πλήκτρο), σταματά ο έλεγχος των σειρών. Η συνάρτηση scan_keypad επιστρέφει έξοδο τη θέση του πλήκτρου που πατήθηκε στον πίνακα που ορίστηκε παραπάνω. Αν δεν έχει πατηθεί κανένα πλήκτρο θα επιστρέψει την τιμή -1.

Στη συνάρτηση scan_keypad_rising_edge καλούμε 2 φορές την τιμή τη συνάρτηση scan_keypad με χρονοκαθυστέρηση 15ms. Αν οι τιμές είναι ίδιες, επιστρέφονται ως έξοδος της συνάρτησης. Αν διαφέρουν, τότε επιστρέφεται έξοδος η τιμή -1 (σα να μην έχει διαβαστεί καμία τιμή από τη scan_keypad).

Στη συνάρτηση keypad_to_ascii καλούμε τη συνάρτηση scan_keypad_rising_edge και επιστρέφουμε ως έξοδο την τιμή του πίνακα που αντιστοιχεί στην τιμή που παίρνουμε. Αν η scan_keypad_rising_edge επιστρέψει -1 η συνάρτηση επιστρέφει 0.

Στο κυρίως πρόγραμμα αρχικά ορίζουμε την θύρα 0 ως έξοδο καθώς ελέγχει την LCD και για τη θύρα 0, τα MSB ως είσοδο (για να βλέπουμε ποιο πλήκτρο πατήθηκε) και τα LSB ως έξοδο (για να ελέγχουμε μια σειρά). Στη συνέχεια μπαίνουμε σε ένα βρόχο while(1). Σε κάθε επανάληψη καλούμε τη συνάρτηση keypad_to_ascii και εξετάζουμε την έξοδό της. Αν αυτή είναι 0, τότε συνεχίζουμε τις επαναλήψεις περιμένοντας να πατηθεί κάποιο πλήκτρο. Αλλιώς κάνουμε αρχικοποίηση της lcd οθόνης και εμφανίζουμε σε αυτή τον χαρακτήρα που πατήθηκε (με ελάχιστο χρόνο εμφάνισης 500ms).

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.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)

//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;
}

// Read one byte from the twi device (dont request more data from device)
unsigned char twi_readNak(void)
{
  TWCR0 = (1<<TWINT) | (1<<TWEN);
  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;
  // send START condition
  TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
  // wait until transmission completed
  while(!(TWCR0 & (1<<TWINT)));
  // check value of TWI Status Register.
  twi_status = TW_STATUS & 0xF8;
  if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
  // send device address
  TWDR0 = address;
  TWCR0 = (1<<TWINT) | (1<<TWEN);
  // wail until transmission completed and ACK/NACK has been received
  while(!(TWCR0 & (1<<TWINT)));
  // check value of TWI Status Register.
  twi_status = TW_STATUS & 0xF8;
  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 )
  {
    // send START condition
    TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
```

```
      // wait until transmission completed
      while(!(TWCR0 & (1<<TWINT)));
      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;
      // send device address
      TWDR0 = address;
      TWCR0 = (1<<TWINT) | (1<<TWEN);
      // wail until transmission completed
      while(!(TWCR0 & (1<<TWINT)));
      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      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);
         // wait until stop condition is executed and bus released
         while(TWCR0 & (1<<TWSTO));
         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 )
{
   // send data to the previously addressed device
   TWDR0 = data;
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // wait until transmission completed
   while(!(TWCR0 & (1<<TWINT)));
   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)
{
   // send stop condition
   TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
   // wait until stop condition is executed and bus released
   while(TWCR0 & (1<<TWSTO));
}
```

```
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 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;
}

void write_2_nibbles(uint8_t c) {
   uint8_t temp = c;
   uint8_t prev = PCA9555_0_read(REG_INPUT_0);
   prev &= 0x0F;
   c &= 0xF0;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   c = temp;
   c &= 0x0F;
   c = c << 4;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   return;
}

void lcd_data(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp |= 0x04;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
   _delay_us(100);
   return;
}

void lcd_command(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp &= 0xFB;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
```

```
    _delay_us(100);
    return;
}

void lcd_init(void) {
    _delay_ms(40);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x20);
    PCA9555_0_write(REG_OUTPUT_0, 0x28);
    PCA9555_0_write(REG_OUTPUT_0, 0x20);

    _delay_us(100);

    lcd_command(0x28);
    lcd_command(0x0C);
    lcd_command(0x01);
    _delay_us(5000);

    lcd_command(0x06);
    return;
}

static volatile unsigned char key_array[16] =       {'*','0','#','D',
                                                     '7','8','9','C',
                                                     '4','5','6','B',
                                                     '1','2','3','A'};

static volatile unsigned char output;

// returns 0 if no pressed button was detected, or the position of the button
//1 for first, 2 for second, 3 for third, 4 for fourth
uint8_t scan_row(uint8_t row)
{
    output = 0x01 << (row - 1);
    output = (~output);
    output = output & 0x0F;
    PCA9555_0_write(REG_OUTPUT_1, output);
    _delay_us(20);
    uint8_t temp = (0x0F | PCA9555_0_read(REG_INPUT_1));
    temp = ~temp;
    temp = (temp >> 4);
    if ((temp & 0x01) == 0x01) {
        return 1;
    } else if ((temp & 0x02) == 0x02) {
        return 2;
```

```
} else if ((temp & 0x04) == 0x04) {
      return 3;
   } else if ((temp & 0x08) == 0x08) {
      return 4;
   }
   return 0; //if no button was pressed
}

int scan_keypad(void)
{
   uint8_t column;
   uint8_t row = 1;
   while (row < 5)
   {
      column = scan_row(row);
      if(column != 0) break;
      row++;
   }
   if (column != 0)
   {
      return (row-1) * 4 + (column-1);    // position of the letter in the array
   } else {
      return -1;
   }
}

int scan_keypad_rising_edge(void)
{
   int result1;
   int result2;
   result1 = scan_keypad();
   _delay_ms(15);
   result2 = scan_keypad();
   if (result1 == result2) {
      return result1;
   } else {
      return -1;
   }
}

unsigned char keypad_to_ascii(void)
{
   int pos = scan_keypad_rising_edge();
   if (pos == -1) {
      return 0;
   } else {
      return key_array[pos];
   }
}

static volatile unsigned char c;

int main(void)
{
   DDRB |= 0x3F;
   twi_init();
```

```
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
    while(1) {
        c = keypad_to_ascii();
        if (c == 0) {
            continue;
        } else {
            lcd_init();
            _delay_ms(2);
            lcd_data(c);
            _delay_ms(500);
        }
    }
}
```

Ζήτημα 6.2

Στην άσκηση αυτή, ελέγχουμε πάλι το πληκτρολόγιο για είσοδο και αν πληκτρολογηθεί ο διψήφιος αριθμός της ομάδας μας, δηλαδή 01, τότε θα ανάψουν για 4 sec τα led της θύρας B, αλλιώς θα αναβοσβήνουν για 5 sec. Χρησιμοποιούμε τις συναρτήσεις που ορίστηκαν στο ζήτημα 6.1.

Στο ζήτημα αυτό ορίζεται και η συνάρτηση blinky, η οποία αναβοσβήνει τα led της θύρας B με τη ζητούμενη συχνότητα για 5 sec συνολικά.

Ορίζουμε τον πίνακα submitted_code, που θα αποθηκεύουμε το συνδυασμό που πληκτρολογήθηκε.

Στο κυρίως πρόγραμμα, ορίζουμε πίνακα με τιμές αυτές του αριθμού της ομάδας μας, τη θύρα B ως έξοδο και τα led σβηστά. Στην επανάληψη, περιμένουμε να πατηθεί κάποιο πλήκτρο, όταν πατηθεί, το αποθηκεύουμε στον πίνακα submitted_code (με αναμονή μέχρι να αφεθεί, για να μην καταχωρηθεί ξανά παρακάτω η ίδια τιμή). Στη συνέχεια, περιμένουμε, σε νέα επανάληψη, να πατηθεί το δεύτερο πλήκτρο, όπως πριν και το αποθηκεύουμε στην επόμενη θέση του πίνακα submitted_code. Αν τα 2 πλήκτρα που πατήθηκαν σχηματίζουν τον αριθμό της ομάδας μας (01), όπως θα προκύψει από τη σύγκριση των 2 πινάκων, τότε ανάβουν για 4 sec τα led της PORTB και μετά παραμένουν σβηστά για 1 sec. Αλλιώς καλείται η συνάρτηση blinky. Και στις 2 περιπτώσεις το πρόγραμμα δε δέχεται είσοδο για 5 sec. Στη συνέχεια τερματίζεται η εσωτερική επανάληψη και περιμένουμε από την αρχή νέο συνδυασμό.

```
#define F_CPU 16000000UL

#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.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)

//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;
}

// Read one byte from the twi device (dont request more data from device)
unsigned char twi_readNak(void)
{
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   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;
   // send START condition
   TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
   // wait until transmission completed
   while(!(TWCR0 & (1<<TWINT)));
   // check value of TWI Status Register.
   twi_status = TW_STATUS & 0xF8;
   if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
   // send device address
   TWDR0 = address;
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // wail until transmission completed and ACK/NACK has been received
   while(!(TWCR0 & (1<<TWINT)));
   // check value of TWI Status Register.
   twi_status = TW_STATUS & 0xF8;
   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 )
   {
      // send START condition
      TWCR0 = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
```

```
      // wait until transmission completed
      while(!(TWCR0 & (1<<TWINT)));
      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) continue;
      // send device address
      TWDR0 = address;
      TWCR0 = (1<<TWINT) | (1<<TWEN);
      // wail until transmission completed
      while(!(TWCR0 & (1<<TWINT)));
      // check value of TWI Status Register.
      twi_status = TW_STATUS & 0xF8;
      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);
         // wait until stop condition is executed and bus released
         while(TWCR0 & (1<<TWSTO));
         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 )
{
   // send data to the previously addressed device
   TWDR0 = data;
   TWCR0 = (1<<TWINT) | (1<<TWEN);
   // wait until transmission completed
   while(!(TWCR0 & (1<<TWINT)));
   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)
{
   // send stop condition
   TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
   // wait until stop condition is executed and bus released
   while(TWCR0 & (1<<TWSTO));
}
```

```
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 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;
}

void write_2_nibbles(uint8_t c) {
   uint8_t temp = c;
   uint8_t prev = PCA9555_0_read(REG_INPUT_0);
   prev &= 0x0F;
   c &= 0xF0;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   c = temp;
   c &= 0x0F;
   c = c << 4;
   c |= prev;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c |= 0x08;
   PCA9555_0_write(REG_OUTPUT_0, c);
   c &= 0xF7;
   PCA9555_0_write(REG_OUTPUT_0, c);

   return;
}

void lcd_data(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp |= 0x04;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
   _delay_us(100);
   return;
}

void lcd_command(uint8_t c) {
   uint8_t temp = PCA9555_0_read(REG_INPUT_0);
   temp &= 0xFB;
   PCA9555_0_write(REG_OUTPUT_0, temp);
   write_2_nibbles(c);
```

```
    _delay_us(100);
    return;
}

void lcd_init(void) {
    _delay_ms(40);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x30);
    PCA9555_0_write(REG_OUTPUT_0, 0x38);
    PCA9555_0_write(REG_OUTPUT_0, 0x30);

    _delay_us(100);

    PCA9555_0_write(REG_OUTPUT_0, 0x20);
    PCA9555_0_write(REG_OUTPUT_0, 0x28);
    PCA9555_0_write(REG_OUTPUT_0, 0x20);

    _delay_us(100);

    lcd_command(0x28);
    lcd_command(0x0C);
    lcd_command(0x01);
    _delay_us(5000);

    lcd_command(0x06);
    return;
}

static volatile unsigned char key_array[16] =       {'*','0','#','D',
                                                     '7','8','9','C',
                                                     '4','5','6','B',
                                                     '1','2','3','A'};

static volatile unsigned char output;

// returns 0 if no pressed button was detected, or the position of the button
//1 for first, 2 for second, 3 for third, 4 for fourth
uint8_t scan_row(uint8_t row)
{
    output = 0x01 << (row - 1);
    output = (~output);
    output = output & 0x0F;
    PCA9555_0_write(REG_OUTPUT_1, output);
    _delay_us(20);
    uint8_t temp = (0x0F | PCA9555_0_read(REG_INPUT_1));
    temp = ~temp;
    temp = (temp >> 4);
    if ((temp & 0x01) == 0x01) {
        return 1;
    } else if ((temp & 0x02) == 0x02) {
        return 2;
```

```
} else if ((temp & 0x04) == 0x04) {
      return 3;
   } else if ((temp & 0x08) == 0x08) {
      return 4;
   }
   return 0; //if no button was pressed
}

int scan_keypad(void)
{
   uint8_t column;
   uint8_t row = 1;
   while (row < 5)
   {
      column = scan_row(row);
      if(column != 0) break;
      row++;
   }
   if (column != 0)
   {
      return (row-1) * 4 + (column-1);    // position of the letter in the array
   } else {
      return -1;
   }
}

int scan_keypad_rising_edge(void)
{
   int result1;
   int result2;
   result1 = scan_keypad();
   _delay_ms(15);
   result2 = scan_keypad();
   if (result1 == result2) {
      return result1;
   } else {
      return -1;
   }
}

unsigned char keypad_to_ascii(void)
{
   int pos = scan_keypad_rising_edge();
   if (pos == -1) {
      return 0;
   } else {
      return key_array[pos];
   }
}

void blinky(void) {
   for(int i = 0; i < 10; i++) {
      PORTB = 0xFF;
      _delay_ms(250);
      PORTB = 0x00;
      _delay_ms(250);
   }
}
```

```
unsigned char submitted_code[2];

int main(void)
{
   unsigned char c;
   unsigned char code[2] = {'0','1'};
   DDRB = 0xFF;
   PORTB = 0x00;
   twi_init();
   PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
   PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);

   // read first button

   while(1) {
      c = keypad_to_ascii();
      if (c == 0) {
         continue;
      } else {
         while (!(keypad_to_ascii()==0)); // wait for the button to be unpressed
         submitted_code[0] = c;          // store first character
      }

      while(1) {                    // same procedure for the second character
         c = keypad_to_ascii();
         if (c == 0) {
            continue;
         } else {
            while (!(keypad_to_ascii()==0)); // wait for the button to be unpressed
            submitted_code[1] = c;          // store second character
            if ((submitted_code[0] == code[0]) && (submitted_code[1] == code[1])) {
               PORTB = 0xFF;
               _delay_ms(4000);
               PORTB = 0x00;
               _delay_ms(1000);
               break;
            } else {
               blinky();
               break;
            }
         }
      }

   }
}
```