ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

6η Εργαστηριακή Αναφορά στο μάθημα "**ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ**" του 7ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Ευάγγελου Περσόπουλου, Α.Μ.: 03121701

1η Άσκηση: -> micro_lab06_ex01.c

Το ζητούμενο πρόγραμμα κάνει χρήση των συναρτήσεων scan_row, scan_keypad, scan_keypad_rising_edge και keypad_to_ascii που υλοποιήθηκαν και εκτυπώνει κάθε φορά στην LCD οθόνη το πλήκτρο που πατήθηκε τελευταίο. Όσον αφορά την υλοποίηση κάθε υπορουτίνας έχουμε τα εξής:

- **scan_row:** ρουτίνα η οποία θέτει κάθε φορά σε λογικό 0 την γραμμή που θέλουμε να διαβαστεί και στην συνέχεια επιστρέφει την τιμή των ακροδεκτών της θύρας IO1.
- **scan_keypad:** ρουτίνα η οποία διαβάζει μέσω της scan_row την κατάσταση και των 4 γραμμών, ελέγχοντας έτσι ολόκληρο το πληκτρολόγιο.
- **scan_keypad_rising_edge:** ρουτίνα η οποία καλεί την scan_keypad 2 φορές με ενδιάμεση καθυστέρηση για την αντιμετώπιση του σπινθιρισμού και στην συνέχεια ελέγχει κατά πόσο οι 2 τιμές είναι ίδιες η όχι, επιστρέφοντας κάθε φορά την τελική τιμή.
- **keypad_to_ascii:** ρουτίνα η οποία μετατρέπει τις τιμές που λαμβάνει από την scan_keypad_rising_edge και τις αντιστοιχεί σε χαρακτήρες προς εκτύπωση στην lcd οθόνη

Αξίζει να σημειωθεί ότι η αποθήκευση της προηγούμενης κατάστασης γίνεται εντός της int main(), έτσι ώστε να αποφευχθεί η διαρκής αποτύπωση του πλήκτρου που πατήθηκε στην LCD οθόνη, όταν αυτό μένει πατημένο παρατεταμένα.

C Program:

```
#define F_CPU 16000000UL //running
#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, read is followed by a stop
condition
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(char x)
{
      char y=PIND & 0x0f;
      char x1=x & 0xf0;
      x1=x1+y;
      PORTD=x1;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      x=x<<4 | x>>4;
      x=x & 0xf0;
      PORTD=x+y;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
}

void lcd_data(char x)
{
      PORTD=PORTD | (1<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_command(char x)
{
      PORTD=PORTD | (0<<PD2);
      write_2_nibbles(x);
      _delay_us(40);
}

void lcd_init (void)
{
      _delay_ms(40);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
```

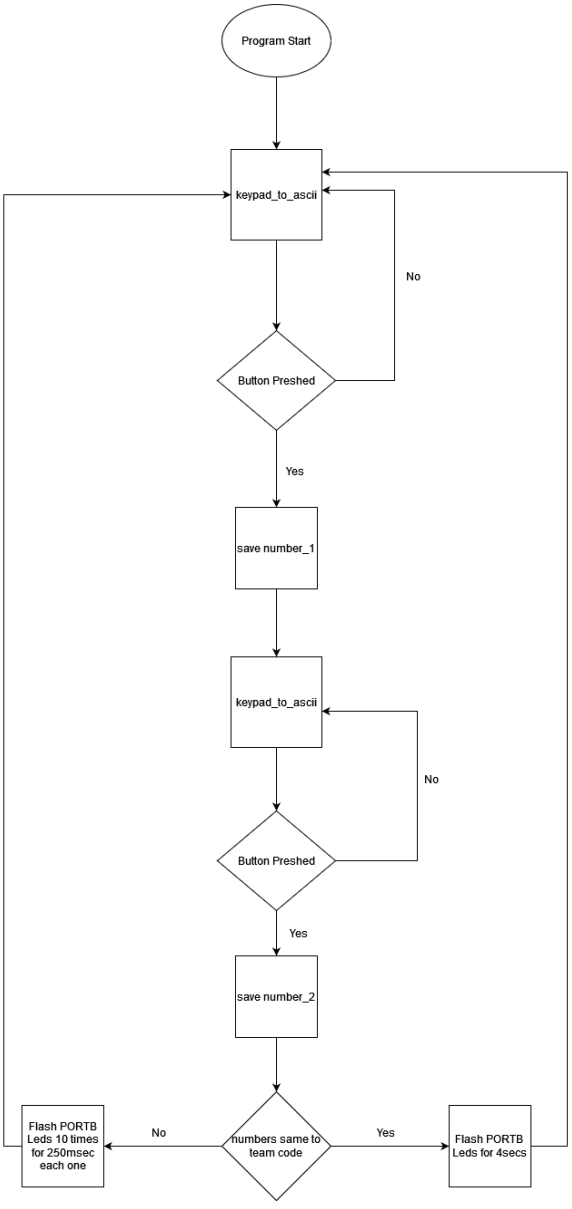
```
      _delay_us(38);
      PORTD=0x30;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      PORTD=0x20;
      PORTD=PORTD | (1<<PD3);
      PORTD=PORTD & (0<<PD3);
      _delay_us(38);
      lcd_command(0x28);
      lcd_command(0x0c);
      lcd_command(0x01);
      _delay_ms(500);
      lcd_command(0x06);
}


unsigned char scan_row(unsigned int row)
{
    PCA9555_0_write(REG_OUTPUT_1, row);
    //_delay_ms(5);
    unsigned char value = PCA9555_0_read(REG_INPUT_1);
    return value;
}

unsigned char scan_keypad(void)
{
    unsigned char key=0;
    key = scan_row(0xFE); //check row 1
    if (key != 0xFE) return key;
    key = scan_row(0xFD); //check row 2
    if (key != 0xFD) return key;
    key = scan_row(0xFB); //check row 3
    if (key != 0xFB) return key;
    key = scan_row(0xF7); //check row 4
    return key;
}

unsigned char scan_keypad_rising_edge(void)
{
     unsigned char check_1 = scan_keypad();
    _delay_ms(10);
    unsigned char check_2 = scan_keypad();  //check for sparkle effect
    if (check_1 != check_2)
    {
        //_delay_ms(20);
        return check_2;
    }
    else return check_1;
}

unsigned char keypad_to_ascii(void)
```

```
{
 unsigned char x = scan_keypad_rising_edge();
 //1st row
 if (x == 0b11101110) return '*';
 if (x == 0b11011110) return '0';
 if (x == 0b10111110) return '#';
 if (x == 0b01111110) return 'D';
 //2nd row
 if (x == 0b11101101) return '7';
 if (x == 0b11011101) return '8';
 if (x == 0b10111101) return '9';
 if (x == 0b01111101) return 'C';
 //3rd row
 if (x == 0b11101011) return '4';
 if (x == 0b11011011) return '5';
 if (x == 0b10111011) return '6';
 if (x == 0b01111011) return 'B';
 //4th row
 if (x == 0b11100111) return '1';
 if (x == 0b11010111) return '2';
 if (x == 0b10110111) return '3';
 if (x == 0b01110111) return 'A';

 return 0xF7;   //if nothing is pressed
}

int main(void)
{
    DDRD |= 0b11111111;  // output for LCD
    lcd_init();
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1's
bit4-7 as input and bit0-3 as output
    unsigned char button, prev=0;
    while(1)
    {
        button = keypad_to_ascii();
        if ((button == 0xF7) || (button == prev)) continue;
        lcd_command(0x01);
        _delay_ms(50);
        lcd_data(button);
        prev = button;
    }
    return 0;
}
```

2η Άσκηση: -> micro_lab06_ex02.c

Το ζητούμενο πρόγραμμα λαμβάνει κάθε φορά έναν διψήφιο κωδικό και σε περίπτωση που αυτός αντιστοιχεί με τον κωδικό της ομάδας (στην προκειμένη περίπτωση 17), κρατά αναμμένα τα leds στο PORTB για 5sec. Αν δοθεί λανθασμένος 2ψήφιος κωδικός, τότε το πρόγραμμα αναβοσβήνει τα leds στο PORTB ανά 250msec. Αυτό ισχύει, μιας και ζητείται συχνότητα 2Hz και duty cycle στο 50% που αντιστοιχεί σε $\boldsymbol{T_{up} = T_{down} = 50\% * (1/2) = 0.25.}$ Οι συναρτήσεις για την ανάγνωση από το πληκτρολόγιο είναι οι ίδιες με αυτές της 1ης άσκησης.

Παρακάτω ακολουθεί το διάγραμμα ροής της άσκησης καθώς και ο κώδικας σε C:



C Program:

```
#define F_CPU 16000000UL //running
#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, read is followed by a stop
condition
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;
}

unsigned char scan_row(unsigned int row)
{
    PCA9555_0_write(REG_OUTPUT_1, row);
    _delay_ms(2);
    unsigned char value = PCA9555_0_read(REG_INPUT_1);
    return value;
}

unsigned char scan_keypad(void)
{
    unsigned char key=0;
    key = scan_row(0xFE); //chech row 1
    if (key != 0xFE) return key;
    key = scan_row(0xFD); //chech row 2
    if (key != 0xFD) return key;
    key = scan_row(0xFB); //chech row 3
```

```
    if (key != 0xFB) return key;
    key = scan_row(0xF7); //chech row 4
    return key;
}

unsigned char scan_keypad_rising_edge_2(void)
{
    unsigned char check_1 = scan_keypad();
    _delay_ms(10);
    unsigned char check_2 = scan_keypad();
    while (check_1 == check_2)
    {
        //_delay_ms(20);
        check_2 = scan_keypad();
    }
    return check_1;
}

unsigned char keypad_to_ascii(void)
{
 unsigned char x = scan_keypad_rising_edge_2();
 //1st row
 if (x == 0b11101110) return '*';
 if (x == 0b11011110) return '0';
 if (x == 0b10111110) return '#';
 if (x == 0b01111110) return 'D';
 //2nd row
 if (x == 0b11101101) return '7';
 if (x == 0b11011101) return '8';
 if (x == 0b10111101) return '9';
 if (x == 0b01111101) return 'C';
 //3rd row
 if (x == 0b11101011) return '4';
 if (x == 0b11011011) return '5';
 if (x == 0b10111011) return '6';
 if (x == 0b01111011) return 'B';
 //4th row
 if (x == 0b11100111) return '1';
 if (x == 0b11010111) return '2';
 if (x == 0b10110111) return '3';
 if (x == 0b01110111) return 'A';

 return 0xF7;   //if nothing is pressed
}

int main(void)
{
    DDRB |= 0b11111111;
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1's
bit4-7 as input and bit0-3 as output
    int i;
```

```
    unsigned char number_1, number_2;
    PORTB = 0x00;
    while(1)
    {
        number_1 = keypad_to_ascii();
        while (number_1 == 0xF7)
        {
            number_1 = keypad_to_ascii();
        }
        number_2 = keypad_to_ascii();
        while (number_2 == 0xF7)
        {
            number_2 = keypad_to_ascii();
        }
        if ((number_1 == '1') && (number_2 == '7'))
        {
                PORTB = 0xFF;
                _delay_ms(4000);
                PORTB = 0x00;
        }
        else
        {
                for(i=0; i<10; i++)
                {
                    PORTB = 0xFF;
                     _delay_ms(250);
                    PORTB = 0x00;
                    _delay_ms(250);
                }

        }
    _delay_ms(5000);
    }
    return 0;
}
```