Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών ΕΜΠ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΑΚΑΔ. ΕΤΟΣ 2024-2025

6ή εργαστηριακή άσκηση.

Ομάδα : 58
Ονοματεπώνυμο : Φέζος Κωνσταντίνος / Α.Μ : 03118076
Ονοματεπώνυμο : Τσουκνίδας Οδυσσέας Αρθούρος Ρήγας / Α.Μ : 03120043

Σε αυτήν την εργαστηριακή άσκηση θα χρησιμοποιήσουμε την διεπαφή TWI0 ώστε να επικοινωνήσουμε με πληκτρολόγιο που είναι συνδεδεμένο στην εξωτερική θύρα IO1 του ολοκληρωμένου PCA9555.

Καταρχάς θα εισάγουμε header file με τις απαραίτητες συναρτήσεις για τον χειρισμό των καταχωρητών του PCA9555 καθώς και τον κώδικα σε C που τις υλοποιεί. Τα αρχεία κώδικα είναι τα παρακάτω :

```
#ifndef HEADER61
#define      HEADER61


#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)
//MY COMMENT: Here pre-scalar is taken as 1 - it will be set so in twi_init
#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

// For reading only TWS[7:3] from TWSR0
#define TW_STATUS_MASK 0b11111000
#define TW_STATUS (TWSR0 & TW_STATUS_MASK)
```

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdbool.h>
#include <stdint.h>

void twi_init(void);
unsigned char twi_readAck(void);
unsigned char twi_readNak(void);
unsigned char twi_start(unsigned char address);
void twi_start_wait(unsigned char address);
unsigned char twi_write( unsigned char data );
unsigned char twi_rep_start(unsigned char address);
void twi_stop(void);

void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value);
uint8_t PCA9555_0_read(PCA9555_REGISTERS reg);


#endif /* HEADER61 */
```

```c
#include "61header.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


//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)
//TWEA =1 -> ACK
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
//TWEA =0 -> No ACK
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;          // Isn't the & unnecessary
	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;
}
```

Έπειτα, θα γράψουμε διάφορες συναρτήσεις για να μπορούμε να δεχόμαστε input από το πληκτρολόγιο από τον χρήστη.

Η **scan_row**, διαβάζει μία γραμμή του πληκτρολογίου, γράφοντας στον καταχωρητή εξόδου του ολοκληρωμένου PCA9555 την τιμή 0 στην γραμμή που θέλουμε να διαβάσουμε και 1 στις υπόλοιπες 3 γραμμές του πληκτρολογίου. Έπειτα, διαβάζει από τον καταχωρητή εισόδου του ολοκληρωμένου, και κοιτώντας το high nibble του, καταλαβαίνει ποια πλήκτρα έχουν πατηθεί καθώς η αντίστοιχη στήλη έχει τιμή 0, και τέλος επιστρέφει ένα 8-bit αποτέλεσμα, το low nibble του οποίου περιέχει 1 στα πλήκτρα που έχουν πατηθεί και 0 στα υπόλοιπα.

Η **scan_keypad** καλεί 4 φορές την scan_row, μία για κάθε γραμμή και επιστρέφει ένα 16-bit αποτέλεσμα, που περιέχει 1 στα πλήκτρα που έχουν πατηθεί και 0 στα υπόλοιπα.

Η **scan_keypad_rising_edge** όταν κληθεί διαβάζει 2 φορές το πληκτρολόγιο με την scan_keypad, με διαφορά 15 ms, και επιστρέφει τα πλήκτρα που έχουν πατηθεί και τις 2 φορές.

Τέλος, η **keypad_to_ascii** καλεί την scan_keypad_rising_edge, βρίσκει ποιο πλήκτρο έχει πατηθεί από το πληκτρολόγιο, και επιστρέφει τον κατάλληλο ascii χαρακτήρα.

Ο κώδικας των παραπάνω συναρτήσεων βρίσκεται παρακάτω:

```
uint16_t scan_row(uint8_t row){
	uint16_t result = 0;

	uint8_t dummy = 0xFF;
	dummy &= ~(1 << (row-1));
	PCA9555_0_write(REG_OUTPUT_1,dummy);

	uint8_t input = PCA9555_0_read(REG_INPUT_1);
	if((input & 0x10) == 0x00) //1st element from row is pressed
	{
		result |= 0x01;
	}
	if((input & 0x20) == 0x00) // 2nd element from row is pressed
	{
		result |= 0x02;
	}
	if((input & 0x40) == 0x00) //3rd element from row  is pressed
	{
		result |= 0x04;
	}
	if((input & 0x80) == 0x00) //4th element from row is pressed
	{
		result |= 0x08;
	}
	return result;
}


uint16_t scan_keypad(){
	uint16_t result =0;
	result |= scan_row(1);		//Scan 1st row
	result |= (scan_row(2) << 4);		//Scan 2nd row
	result |= (scan_row(3) << 8);		//Scan 3rd row
	result |= (scan_row(4) << 12);		//Scan 4th row
	return result;				//Return total buttons pressed
}

uint16_t scan_keypad_rising_edge(){
	uint16_t pressed_keys_temp = 0, dummy = 0, result =0;;
	pressed_keys_temp = scan_keypad();
	_delay_ms(15);
	dummy = scan_keypad();

	//Here we take 2 measurements and we only keep the buttons which are pressed
in both (Correct??)
	pressed_keys_temp &= dummy;

	//pressed_keys = pressed_keys_temp;
```

```
	//Here we only update pressed_keys to only keep keys that are now pressed and
weren't before
	//result = (~(pressed_keys)) & pressed_keys_temp;
	result = pressed_keys_temp;
	return result;
}


uint8_t keypad_to_ascii() {
	uint16_t from_keys = scan_keypad_rising_edge();
	int counter = 0;
	while((from_keys & 0x0001) == 0 && counter != 16) {
		counter++;
		from_keys = from_keys >> 1;
	}
	if(counter == 16){
		return 0;
	}
	return ascii[counter];
}
```

Ζήτημα 6.1

Στο ζήτημα αυτό καλούμαστε να διαβάζουμε user input από το πληκτρολόγιο και να ανάβουμε συγκεκριμένα LEDs όταν πατιούνται συγκεκριμένα κουμπιά. Ο κώδικας μας βρίσκεται παρακάτω:

```
int main(){
	twi_init();
	PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output (to send
stuff to LEDs_
	PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set EXT_PORT1[7:4] as input and
[3:0] as output
	DDRB = 0xFF;                                //Set PORTB as output
	PORTB = 0x00;
	uint8_t reading;
	DDRD = 0xFF;
	PORTD = 0x00;

	while(1) {
		reading = keypad_to_ascii();
		if(reading == 'A') {
			PORTB = 0x01;
		}
		else if(reading == '8') {
			PORTB = 0x02;
		}
		else if(reading == '6') {
			PORTB = 0x04;
		}
		else if(reading == '*') {
```

```
                PORTB = 0x08;
            }

            else
            {
                PORTB = 0x00;
            }
        }

}
```

Ζήτημα 6.2

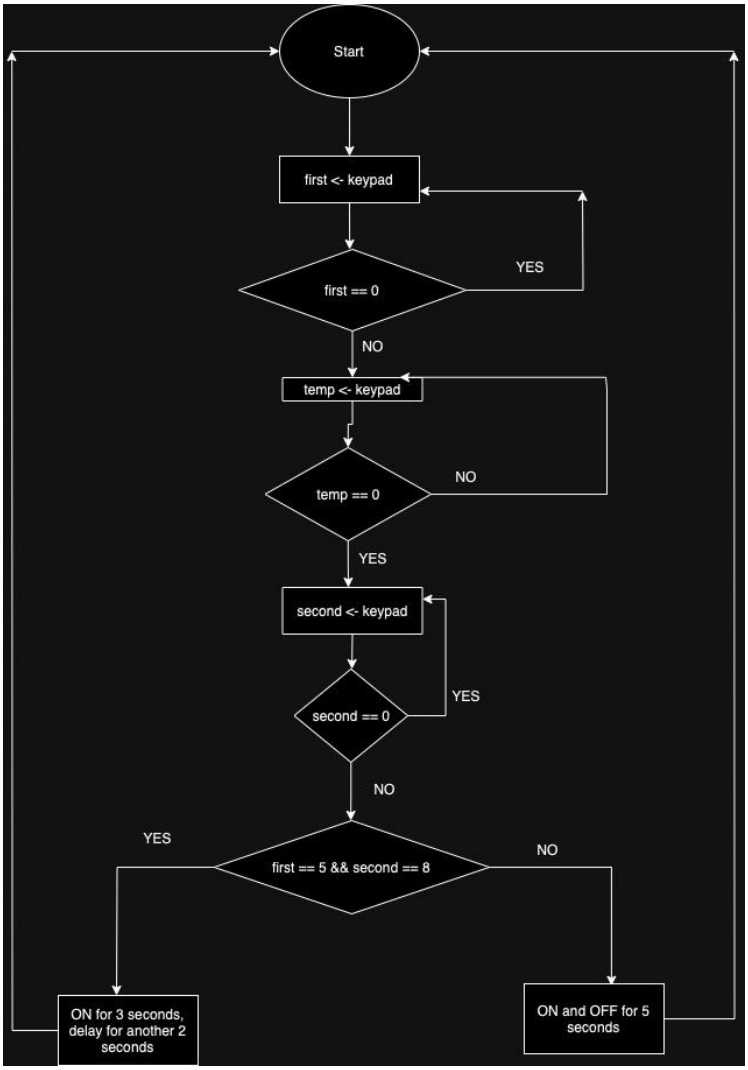
Στο ζήτημα αυτό καλούμαστε να γράψουμε κώδικα ο οποίος θα απεικονίζει στην οθόνη το τελευταίο πλήκτρο που έχει πατηθεί από τον χρήστη,

```
int main() {
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //PORT0 as output (just to send stuff
to LCD screen)
       PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //PORT1[7:3] = input, [3:0] =
output
    lcd_init();
    lcd_clear_display();
    uint8_t reading;
    char no_keys1[] = "PRESS SOME KEYS", no_keys2[] = "ALREADY";
    for(int i=0; i<strlen(no_keys1); i++)
    {
        lcd_data(no_keys1[i]);
    }
    lcd_change_line();
    for(int i=0; i<strlen(no_keys2); i++)
    {
        lcd_data(no_keys2[i]);
    }

    while(1)
    {
        reading = keypad_to_ascii();
        if(reading != 0)
        {
            lcd_clear_display();
            lcd_data(reading);
        }
    }

}
```

Ζήτημα 6.3

Στο ζήτημα αυτό μας ζητείται να γράψουμε πρόγραμμα συνεχόμενης λειτουργίας, το οποίο θα διαβάζει user input 2 πλήκτρα από τον χρήστη. Αν τα 2 πλήκτρα αναπαριστούν σωστά τον αριθμό της ομάδας μας (58), ανάβουν πρώτα τα 6 PB LEDs για 3 δευτερόλεπτα, και μετά περνάνε άλλα 2 δευτερόλεπτα που το πρόγραμμα δεν δέχεται input. Αν αντίθετα δεν δοθεί ο αριθμός 58, αναβοσβήνουν τα PB LEDs για 5 δευτερόλεπτα.

Το διάγραμμα ροής του προγράμματος μας είναι παρακάτω:



Ο κώδικας είναι ο παρακάτω:

```
int main() {
	twi_init();
	PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //PORT1[7:3] = input, [3:0] =
output
	DDRB = 0xFF;                                    //Set PORTB as output;
	PORTB = 0x00;
	uint8_t first, second;

	while(1) {

		//Maybe we need some delays between the whiles to avoid de-bouncing??

		//Wait for the first key to be pressed and when it is, store it in
variable first
		while((first = keypad_to_ascii()) == 0);

		//Loop up until the user un-presses the first key
		while((keypad_to_ascii()) != 0);

		//Wait for the second key to be pressed and when it is, store it in
variable second
		while((second = keypad_to_ascii()) == 0);

		//Now we have the values of the pressed keys and we can continue
		if(first == '5' && second == '8')
		{
			PORTB = 0xFF;
			_delay_ms(3000);
			PORTB = 0x00;
			_delay_ms(2000);
		}
		else
		{
			for(int i=0; i<5; i++)
			{
				PORTB = 0xFF;
				_delay_ms(500);
				PORTB = 0x00;
				_delay_ms(500);
			}
		}
	}
}
```