2023

Αναφορά 5ης εργαστηριακής Άσκησης

Εργαστήριο μικροϋπολογιστών

ΕΜΜΑΝΟΥΗΛ ΞΕΝΟΣ (ΑΜ):03120850
ΠΑΠΑΔΟΠΟΥΛΟΣ ΣΠΥΡΙΔΩΝ (ΑΜ):03120033

Ζήτημα πρώτο

Στην άσκηση αυτή μας ζητήθηκε να πραγματοποιήσουμε ουσιαστικά την επικοινωνία του μικροεπεξεργαστή με το ολοκληρωμένο PCA9555. Για την διασφάλιση την σωστή επικοινωνία μεταξύ των δύο χρησιμοποιήθηκε το πρωτόκολλο TWI , το οποίο χρησιμοποιεί δύο κοινούς διαύλους για όλα τα ολοκληρωμένα με τα οποία συνδέεται ο επεξεργαστής μας, ώστε να διασφαλισθεί η ορθή επικοινωνία μεταξύ όλων των συσκευών . Μέσω του πρωτοκόλλου αυτού αποφεύγονται οι συγκρούσεις και συγχρονίζεται η επικοινωνία μεταξύ του master (η συσκευή που αρχίζει την μεταφορά, παράγει το σήμα ρολογιού και τερματίζει την μεταφορά) και του slave(η συσκευή που καλείται από τον master). Συνεπώς χρησιμοποιώντας μόνο δύο διαύλους, τον SCL που είναι για το ρολόι και τον SDA που είναι για την μεταφορά δεδομένων μπορούμε να επιτύχουμε επικοινωνία μεταξύ του μικροεπεξεργαστή μας και συσκευών ανάλογων σε αριθμό με το πλήθος των bit που υπάρχουν διαθέσιμα στον διάδρομο SDA για διευθυνσιοδότηση των εξωτερικών συσκευών. Περιληπτικά στο πρωτόκολλο αυτό όταν μία συσκευή θέλει να επικοινωνήσει με μία άλλη συγχρονίζεται με το ρολόι στέλνει ένα σήμα start και στην συνέχεια την διεύθυνση της συσκευής με την οποία θέλει να επικοινωνήσει. Αν αυτό το στάδιο είναι επιτυχές τότε η συσκευή δέκτης θα στείλει ένα ACK (acknowledgment) στον πομπό ότι είναι έτοιμη για να ξεκινήσει η επικοινωνία. Κάθε φορά που ένας πομπός θα στέλνει κάποια δεδομένα ο δέκτης θα πρέπει να επιβεβαιώνει ότι τα έλαβε στέλνοντας ένα σήμα ACK. Στο τέλος της επικοινωνίας θα σταλεί ένα σήμα που αντιστοιχεί στο STOP. Αυτή είναι περιληπτικά η βασική ιδέα του πρωτοκόλλου επικοινωνίας αυτού.

Στο πλαίσιο της υλοποίησης, μας δίνεται μία διαπροσωπεία(interface) (δηλαδή ένα σύνολο από συναρτήσεις) μέσω των οποίων μπορούμε να επικοινωνήσουμε με το ολοκληρωμένο PCA9555 χρησιμοποιώντας το TWI πρωτόκολλο επικοινωνίας. Συνεπώς, εμείς πρακτικά στο πλαίσιο της άσκησης μπορούμε απλώς να χρησιμοποιήσουμε την διαπροσωπεία αυτή για να λύσουμε τα ζητούμενα χωρίς να αναφερθούμε στο πρωτόκολλο, καθώς αυτό υλοποιείται από το σύνολο των συναρτήσεων που η διαπροσωπεία αυτή μας παρέχει. Ωστόσο θα κάνουμε μία σύντομη εξήγηση κάποιων συναρτήσεων που καλούμε στα ζητούμενα :

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

Η twi_init() αρχικοποιεί τον δίαυλο επικοινωνίας καθώς αρχικοποιεί της τιμές των flags που φαίνονται στον κώδικα, θέτοντας τον prescaler σε τιμή 1 και την συχνότητα του ρολογιού στα 100kHz.

```
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();
}
```

Βλέπουμε ότι η PCA9555_0_write δέχεται σαν είσοδο τον καταχωρητή που επιθυμούμε να γράψουμε και την τιμή που θέλουμε να γράψουμε και καλεί αρχικά την twi_start_wait. Η twi_start_wait παίρνει σαν όρισμα την διεύθυνση του ολοκληρωμένου στο οποίο θέλουμε να γράψουμε και ξεκινάει την επικοινωνία μεταξύ του μικροεπεξεργαστή(master) και του ολοκληρωμένου PCA9555(slave).

```
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;
    }
}
```

Η έναρξη της επικοινωνίας και η επιβεβαίωση πως η σύνδεση μεταξύ μικροεπξεργαστή και ολοκληρωμένου στέφθηκε με επιτυχία πραγματοποιείται θέτοντας κάποια flags και μετά με polling ελέγχουμε αν ο slave θα τους αλλάξει τιμή. Τότε από τον δίαυλο των δεδομένων παίρνουμε το status που μας δείχνει αν η επικοινωνία είναι εφικτή τώρα ή όχι.

Με την ίδια ακριβώς λογική δουλεύει και το twi_write:

```
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;
}
```

Το twi_write παίρνει σαν είσοδο τα δεδομένα που θέλουμε να γράψουμε μέσω του καταχωρητή TWDR0 τα στέλνει στον δίαυλο SDA, θέτει τον TWCR0 που καθορίζει την έναρξη της εγγραφής και αναμένει στην συνέχεια ο ίδιος καταχωρητής να αλλάξει τιμή και να πάρει την κατάλληλη , ώστε να σημανθεί το πέρας της εγγραφής. Τέλος λαμβάνουμε το status της ενέργειας αυτής και με βάση την τιμή του δηλώνουμε επιτυχία στην εγγραφή ή αποτυχία

Τέλος, το twi_stop με την ίδια λογική θέτει τα κατάλληλα flags του καταχωρητή TWCR0 και στην συνέχεια εκτελεί polling για να βεβαιώσει ότι η επικοινωνία μέσω του διαύλου διακόπηκε. Πλέον ο δίαυλος είναι ελεύθερος να χρησιμοποιηθεί από κάποιον άλλο πιθανό χρήστη.

```
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));
}
```

Αυτές συνιστούν τις βασικές συναρτήσεις ώστε να λύσουμε το πρώτο ζήτημα, η λύση του οποίου παρατίθεται ακολούθως:

```
#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, 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;
}
```

```
int main(){
    // initialize the input/ouput
    uint8_t A,B,C,D,F0,F1;
    DDRB = 0x00;
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    while(1){
        //read all inputs and place them to the LSB of the uint8_t they are
assigned to
        A= ((PINB+1) & 1);
        B=((PINB+2) & 2) >>1;
        C=((PINB+4) & 4) >>2;
        D=((PINB+8) & 8) >>3;
        // calculate the result
        F0=((((A+1) & B) | ((B+1) & C & D))+1) &  1; // to find the complement
of the LSB just
                                                    // just  add 1. In the end
we keep only the last bit
        F1=((A & C) & (B | D))<<1;
        // writye the result to the desired output
        PCA9555_0_write(REG_OUTPUT_0, F0+F1);
    }
}
```

Ουσιαστικά χρησιμοποιώντας την δοθείσα διαπροσωπεία δεν έχουμε και τόσα πολλά πράγματα να υλοποιήσουμε στην main μας. Αρχικά θέτουμε ως είσοδο το PORTB, κάνουμε initialize την επικοινωνία του μικροεπεξεργαστή με τον δίαυλο και θέτουμε το EXT_PORT0 ως έξοδο. Στην συνέχεια εκτελούμε ένα συνεχή βρόχο , στον οποίο διαβάζουμε την είσοδο από το PORTB και κάθε bit το τοποθετούμε στο LSB μίας μεταβλητής, υπολογίζουμε την έξοδο και την γράφουμε ουσιαστικά στα IO0_0 και IO0_1 . Αυτή είναι η υλοποίηση μας για το ζήτημα πρώτο.

Ζήτημα δεύτερο

Στο ζήτημα δεύτερο θα χρειαστεί να χρησιμοποιήσουμε και την συνάρτηση PCA955_0_read για να διαβάσουμε το περιεχόμενο του IO1. Για αυτό τον λόγο θα την εξηγήσουμε περιληπτικά:

```
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;
}
```

Αρχικά η PCA9555_0_read παίρνει σαν Input τον καταχωρητή από τον οποίο θέλουμε να διαβάσουμε(δηλαδή ουσιαστικά την διεύθυνση του). Εκτελεί αρχικά την twi_start_wait με όρισμα το ολοκληρωμένο με το οποίο θέλουμε να επικοινωνήσουμε(δηλαδή την διεύθυνση αυτού)+ τον κωδικό ώστε να γράψουμε σε αυτό. Όπως εξηγήσαμε προηγουμένως η twi_start_wait χρησιμοποιείται για να δημιουργηθεί και επιβεβαιωθεί η σύνδεση μεταξύ του μικροεπεξεργαστή και μίας συσκευής/ολοκληρωμένου. Στην συνέχεια γράφουμε στον δίαυλο SDA μέσω της write την διεύθυνση του καταχωρητή στον οποίο θέλουμε εμείς ακριβώς να διαβάσουμε. Στην συνέχεια εκτελείται η twi_rep_start

```
unsigned char twi_rep_start(unsigned char address)
{
    return twi_start( address );
}
```

Η twi_rep_start απλώς στέλνει ξανά το σήμα start δηλώνοντας πως ο επεξεργαστής δεν θέλει να διακόψει την επικοινωνία μέσω του διαύλου και παίρνει σαν όρισμα την διεύθυνση του ολοκληρωμένου με το οποίο ο επεξεργαστής(σε αυτή την περίπτωση) θέλει να επικοινωνήσει. Εδώ η twi_rep_start χρησιμοποιείται για να αλλάξουμε την κατάσταση της επικοινωνίας , δηλαδή να μην γράφουμε πλέον στον reg αλλά να διαβάζουμε από αυτόν (ισοδυναμεί με αλλαγή στην κατάσταση του μικροεπεξεργαστή από master-transmitter σε master-receiver και του slave από slave-receiver σε slave transmitter)

Η twi_readNak() εκτελεί την ανάγνωση από τον καταχωρητή που έχει καθοριστεί από πιο πριν στην επικοινωνία μας και επιστρέφει το αποτέλεσμα της ανάγνωσης αυτής. Το Nak στο τέλος δηλώνει ότι μετά από την επικοινωνία αυτή δεν υπάρχουν άλλα δεδομένα να μεταφερθούν και η επικοινωνία πρέπει να διακοπεί, όπως γίνεται μετά με το twi_stop .

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

Για να κάνει αυτά που εξηγήσαμε η twi_readNak θέτει τα κατάλληλα flag του TCCR0 για να ξεκινήσει η επικοινωνία και εκτελεί polling για να ελέγξει αν άλλαξαν. Αν άλλαξαν τότε στην γραμμή SDA θα υπάρχουν τα περιεχόμενα του καταχωρητή που θέλαμε να διαβάσουμε και συνεπώς για αυτό η συνάρτηση επιστρέφει το TWDR0.

Πλέον έχουμε αναλύσει όλες τις συναρτήσεις και μπορούμε με ευκολία να εξηγήσουμε την υλοποίηση μας για το ζήτημα 2:

```
int main(){
    // initialize the input/ouput
    uint8_t input,output;
    DDRD = 0xFF;
    PORTD=0;
    twi_init();
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
    PCA9555_0_write(REG_CONFIGURATION_1, 0xF0); //Set IO1_4-IO1_7 as input and
IO1_0 as output
    while(1){
        PCA9555_0_write(REG_OUTPUT_1,0);
        input = PCA9555_0_read(REG_INPUT_1) & 0xF0; // read IO1_4 THROUGH
IO1_7
        output=(~(input>>4)) & 0x0F;            // we have to inverse the
input because we use inverse logic
                                                // due to pull up
        PCA9555_0_write(REG_OUTPUT_0,output); // Left shift 4 times to get the
input bits to the proper position
    }
}
```

Παραθέτουμε μόνο την main καθώς μόνο αυτή αλλάζει σε σχέση με το ζήτημα 1. Για το πρόβλημα μας θέτουμε ως έξοδο το PORTD και το αρχικοποιούμε στο 0. Στην συνέχεια αρχικοποιούμε τον δίαυλο επικοινωνίας και θέτουμε το EXT_PORT0 ως έξοδο, το IO1_0 ως είσοδο και τα IO1_4-IO1_7 ως εξόδους. Με βάση το σχήμα που δίνεται παρατηρούμε ότι όταν πατάμε ένα από τα πλήκτρα της τελευταίας σειράς του πληκτρολογίου βραχυκυκλώνουμε ανάλογα το πλήκτρο το αντίστοιχο από τα IO1_4-IO1_7 με το IO1_0. Συνεπώς παρατηρούμε ότι αν εμείς θέτουμε μία τιμή στο IO1_0 (αφού είναι έξοδος) μπορούμε να παίρνουμε την ίδια τιμή στην έξοδο(IO1_4-IO1_7) όταν το κατάλληλο πλήκτρο πατιέται. Η default τιμή των IO1 είναι 1 λόγω των pull-up resistors για αυτό θα γράφουμε το

0 στο IO1_0 και θα αναζητούμε αυτό στις εξόδους. Το τελικό αποτέλεσμα θα το γράφουμε στο IO0. Με βάση την λογική αυτή στο πρόγραμμα μας εκτελούμε έναν συνεχή βρόχο, όπου αρχικά γράφουμε στο IO1_0 το 0. Στην συνέχεια διαβάζουμε το IO1 και κρατάμε τα 4 MSB καθώς αυτά αντιστοιχούν στα IO1_4-IO1_7 . Εκτελούμε 4 shift δεξιά αυτού που διαβάσαμε καθώς με βάση τα ζητούμενα στην εκφώνηση θλεουμε να αντιστοιχίσουμε το PD0 στο IO1_4, το PD1 στο IO1_5,το PD2 στο IO1_6 και το PD3 στο IO1_7. Επιπλέον αντιστρέφουμε το input καθώς εμείς θέλουμε να ανάβει το led όταν πατάμε ένα πλήκτρο(δηλαδή να έχουμε έξοδο 1), αλλά όπως εξηγήσαμε παραπάνω όταν πατάμε ένα πλήκτρο έχουμε είσοδο 0. Τέλος, γράφουμε την έξοδο στο IO0 δηλαδή στην τρέχουσα συνδεσμολογία στο EXT_PORT0 . Αυτή ήταν η υλοποίηση μας για το ζήτημα 2

Ζήτημα τρίτο

Στο τελευταία ζήτημα ζητήθηκε να συνδεθεί η LCD display 2x16 στην θύρα επέκτασης 0 του ολοκληρωμένοι PCA9555 και να υλοποιηθεί πρόγραμμα σε C που θα απεικονίζει στην οθόνη το όνομα και το επίθετό μας (εμείς απεικονίζουμε τα ονόματά μας). Ο κώδικας που υλοποιεί τα παραπάνω ακολουθεί:

```
#define F_CPU 16000000UL
#include<avr/io.h>
#include<avr/interrupt.h>
#include<util/delay.h>
#define PCA9555_0_ADDRESS 0x40
#define TWI_READ 1
#define TWI_WRITE 0
#define SCL_CLOCK 100000L
//A0=A1=A2=0 by hardware
// reading from twi device
// writing to twi device
// 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)));

    // 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);
void lcd_clear_display();
void lcd_command(uint8_t com);
void lcd_data(unsigned char data);
void lcd_init();
void lcd_string(char *str);

void write_2_nibbles(uint8_t c){

    uint8_t temp= c;
    PCA9555_0_write(REG_OUTPUT_0, (PCA9555_0_read(REG_INPUT_0) & 0x0f) + (temp
& 0xf0)); //LCD Data High Bytes
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    asm("nop");
    asm("nop");
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) & (~0x08));

    c=(c<<4)|(c>>4);
    PCA9555_0_write(REG_OUTPUT_0, (PCA9555_0_read(REG_INPUT_0) & 0x0f) + (c &
0xf0)); //LCD Data Low Bytes

    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    asm("nop");
    asm("nop");
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) & (~0x08));

}

void lcd_clear_display(){
    lcd_command(0x01);
    _delay_ms(5);
}

void lcd_command(uint8_t com){
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) & (~0x04));
//LCD_RS=0 => Instruction
    write_2_nibbles(com);
    _delay_us(250);
```

```
}

void lcd_data(uint8_t  data){
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x04);
//LCD_RS=1 => Data
    write_2_nibbles(data);
    _delay_us(250);
}

void lcd_init(){
    _delay_ms(200);
    int i=0;
    while(i<3){      //command to switch to 8 bit mode
        PCA9555_0_write(REG_OUTPUT_0, 0x030);
        PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
        asm("nop");
        asm("nop");
        PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) & (~0x08));
        _delay_us(250);
        ++i;
    }

    PCA9555_0_write(REG_OUTPUT_0, 0x20); //command to switch to 4 bit mode
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) | 0x08);
    _delay_us(2);
    PCA9555_0_write(REG_OUTPUT_0, PCA9555_0_read(REG_INPUT_0) & (~0x08));

    _delay_us(250);

    lcd_command(0x28); //5*8 dots, 2 lines
    lcd_command(0x0c); //display on, cursor off

    lcd_clear_display();

}

void lcd_string(char *str){
    int i;
    for(i=0; str[i]!=0; i++) lcd_data(str[i]);
}




int main()
{

    twi_init();
```

```
    PCA9555_0_write(REG_CONFIGURATION_0, 0x00);

    lcd_init();

    lcd_clear_display();
    char name[]="Spyros Manos";
    lcd_string(name);
}
```

Στην άσκηση αυτή δεν μπορούμε να χρησιμοποιήσουμε αυτούσιες τις συναρτήσεις που χρησιμοποιούσαμε στις προηγούμενες ασκήσεις για την αρχικοποίηση και την επικοινωνία με την οθόνη γιατί αυτές οι συναρτήσεις χρησιμοποιούσαν την PORTD του ATMega328PB ενώ εδώ επιθυμούμε η οθόνη να είναι χρησιμοποιεί την θύρα επέκτασης 0 του PCA9555. Έτσι τροποποιήσαμε κατάλληλα τις συναρτήσεις write_2_nibbles(uint8_t c), lcd_command(uint8_t com), lcd_data(uint8_t data) και lcd_init() ώστε τα παραπάνω να γίνονται μέσω της External Port 0. Οι τροποποιήσεις που έγιναν δεν είναι τίποτα παραπάνω από το να αντικαταστήσουμε τις αναγνώσεις των PIND και το γράψιμο της PORTD με PCA9555_0_read(REG_INPUT_0) και PCA9555_0_write(REG_OUTPUT_0,…) αντίστοιχα (στο δεύτερο όρισμα της PCA9555_0_write βάζουμε ό,τι πληροφορία θέλουμε να γράψουμε).

Αφού λοιπόν έχουμε φτιάξει τον τρόπο που επικοινωνούμε με το LCD Display μπορούμε να προχωρήσουμε στην main. Σε αυτή αρχικοποιούμε το TWI, θέτουμε το External Port 0 ως έξοδο ώστε να μπορούμε να στέλνουμε τα δεδομένα στην LCD Display και αρχικοποιούμε την οθόνη μέσω της συνάρτησης lcd_init(). Έπειτα καθαρίζουμε την LCD Display και αρχικοποιούμε ένα string ως “Spyros Manos” που είναι τα μικρά μας ονόματα. Στην συνέχεια μέσω της συνάρτησης lcd_string, βάζοντάς της ως όρισμα το string που αρχικοποιήσαμε με τα ονόματά μας, εκτυπώνονται τα ονόματά μας στην LCD Display.