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

Αναφορά

5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ ΓΙΑ ΤΟ ΜΑΘΗΜΑ "Εργαστήριο Μικροϋπολογιστών"

Χρήση εξωτερικών Θυρών Επέκτασης στον AVR

Ομάδα 1

Ξανθόπουλος Παναγιώτης (03119084)

Παπαναστάσης Αθανάσιος (03113197)

Ζήτημα 5.1

Στην άσκηση αυτή χρησιμοποιούμε το ολοκληρωμένο επέκτασης θυρών PCA9555. Συνδέουμε με καλώδια τους ακροδέκτες IO0_0 και IO0_1 του κονέκτορα P18 με τους ακροδέκτες LED_PD0 και LED_PD1 του κονέκτορα J18 αντίστοιχα, όπως αναφέρεται στην εκφώνηση.

Στο κύριο πρόγραμμα, θέτουμε την PORTB ως είσοδο ώστε να μπορούμε να βλέπουμε ποια κουμπιά έχουν πατηθεί. Καλούμε την twi_init() ώστε να αρχικοποιήσει το Port Expander στην κατάλληλη ταχύτητα ρολογιού και θέτουμε την θύρα 0 του PCA9555 ως έξοδο.

Στη συνέχεια, σε έναν βρόχο while(1), υπολογίζουμε κάθε φορά το F0 και το F1. Αυτά αποθηκεύονται στο LSB 2 global μεταβλητών F0 και F1. Κάνουμε αριστερή ολίσθηση στο F1, το προσθέτουμε με το F0 και με την συνάρτηση PCA9555_0_write, στέλνουμε το αποτέλεσμα στην θύρα 0. Επειδή έχουμε συνδέσει τα 2 LSB της με τα 2 LED, το αποτέλεσμα φαίνεται στα LED.

(Σημείωση: Όταν πατάμε κάποιο κουμπί, θεωρούμε ότι η αντίστοιχη μεταβλητή γίνεται 1. Αντιστρέφουμε το PINB κάθε φορά ώστε να ακυρώσουμε την αντίστροφη λογική των κουμπιών.)

```
#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;
}

static volatile uint8_t A;
static volatile uint8_t B;
static volatile uint8_t C;
static volatile uint8_t D;
static volatile uint8_t F0;
static volatile uint8_t F1;

// if a button is pressed, it is represented by logical 1
// for the outputs, logical 1 is represented by LED

void computeF0(void) {
   A = (~PINB) & 0x01;
   B = ((~PINB) & 0x02) >> 1;
   C = ((~PINB) & 0x04) >> 2;
   D = ((~PINB) & 0x08) >> 3;
   uint8_t notA = ~A;
   uint8_t notB = ~B;
   uint8_t help1 = notA & B;
   uint8_t help2 = notB & C & D;
   F0 = ~(help1 | help2);
}
```

```
void computeF1(void) {
   // A,B,C,D have already been read because computeF1 is always called after computeF0
   // also, if the buttons change between the calls, we don't want inconsistent values between F0 and
F1

   uint8_t help1 = A & C;
   uint8_t help2 = B | D;
   F1 = help1 & help2;
}

int main(void) {
   DDRB = 0x00;
   twi_init();
   PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
   while(1)
   {
      computeF0();
      computeF1();
      F0 &= 0x01;
      F1 &= 0x01;
      F1 = F1 << 1;
      uint8_t F = F0 | F1;
      PCA9555_0_write(REG_OUTPUT_0, F);
//       _delay_ms(100);
   }
}
```

Ζήτημα 5.2

Στην άσκηση αυτή, συνδέουμε με καλώδια τους ακροδέκτες IO0_0 έως IO0_3 του κονέκτορα P18 με τους ακροδέκτες LED_PD0 έως LED_PD3 του κονέκτορα J18 αντίστοιχα, όπως αναφέρεται στην εκφώνηση.

Στο κυρίως πρόγραμμα, αρχικά καλούμε την συνάρτηση αρχικοποίησης twi_init(). Στη συνέχεια, ορίζουμε τη θύρα επέκτασης 0 ως έξοδο, και από τη θύρα 1 τους ακροδέκτες IO1_0 έως IO1_3 ως εξόδους, ενώ τους ακροδέκτες IO1_4 έως IO1_7 ως εισόδους.

Σε έναν βρόχο while(1), αρχικά στέλνουμε 0 στην θύρα 1 του PCA. Αυτό σημαίνει ότι οι ακροδέκτες που ορίστηκαν ως έξοδοι, θα βγάζουν λογικό 0. Όταν πατηθεί κάποιο κουμπί, ο αντίστοιχος ακροδέκτης εκ των IO1_4 έως IO1_7 θα γίνει 0. Οπότε, διαβάζουμε την θύρα 1. Αν έχει πατηθεί το κουμπί «*», θα λάβουμε την τιμή 1110 XXXX (τα X αντιστοιχούν στους ακροδέκτες εξόδου). Κάνουμε OR με το 0x0F ώστε τα X να γίνουν 1, παίρνουμε το συμπλήρωμα (γιατί τα LED δε λειτουργούν με αντίστροφη λογική), κάνουμε 4 shift right (ώστε τα 4 MSB να έρθουν στα 4 LSB) και δίνουμε το αποτέλεσμα στην έξοδο. Επειδή έχουμε συνδέσει τα καλώδια, το αποτέλεσμα φαίνεται στα LED_PD0 έως LED_PD3.

```
#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);
```

```
      // 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(void) {
   twi_init();
   PCA9555_0_write(REG_CONFIGURATION_0, 0x00);
   PCA9555_0_write(REG_CONFIGURATION_1, 0xF0);
//    PCA9555_0_write(REG_OUTPUT_0, 0x00);
   while(1) {
      PCA9555_0_write(REG_OUTPUT_1, 0x00);
      uint8_t temp = (0x0F | PCA9555_0_read(REG_INPUT_1));
      temp = ~temp;
      temp = (temp >> 4);
      PCA9555_0_write(REG_OUTPUT_0, temp);
   }
}
```