Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

7η Σειρά Ασκήσεων

Μάθημα: Εργαστήριο Μικροϋπολογιστών

Εξάμηνο: 7ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Ζήτημα 7.1

Να υλοποιηθεί κώδικας για το μικροελεγκτή ATmega328PB, σε γλώσσα C για την επικοινωνία του μικροελεγκτή με τον αισθητήρα θερμοκρασίας DS18B20 στην κάρτα NtuAboard_G1. Το πρόγραμμα να περιλαμβάνει μόνο τις στοιχειώδεις λειτουργίες, λαμβάνοντας υπόψη ότι στον ακροδέκτη PD4 είναι συνδεδεμένος μόνο ένας αισθητήρας. Οι στοιχειώδεις αυτές λειτουργίες στηρίζονται στις εντολές που δέχεται ο αισθητήρας και παρουσιάζονται στο τεχνικό εγχειριδίου του DS18B20.

Η διαδικασία περιλαμβάνει τα εξής βήματα για την αλληλεπίδραση με τον αισθητήρα θερμοκρασίας (DS18B20) συνδεδεμένο στον ακροδέκτη PD4:

1. Αρχικοποίηση και έλεγχος σύνδεσης συσκευής με τη ρουτίνα one_wire_reset.
2. Χρήση της ρουτίνας one_wire_transmit_byte για αποστολή εντολής 0xCC, παρακάμπτοντας την επιλογή συσκευής λόγω μοναδικής σύνδεσης.
3. Αποστολή εντολής 0x44 για έναρξη μέτρησης θερμοκρασίας.
4. Χρήση one_wire_receive_bit για έλεγχο ολοκλήρωσης μέτρησης.
5. Νέα αρχικοποίηση συσκευής με one_wire_reset.
6. Αποστολή εντολής 0xCC.
7. Αποστολή εντολής 0xBE για ανάγνωση 16-bit θερμοκρασίας.
8. Αποθήκευση τιμής σε καταχωρητές r25:r24 με κλήση one_wire_receive_byte δύο φορές.
9. Επιστροφή τιμής 0x8000 αν δεν υπάρχει συσκευή στον ακροδέκτη PD4.

```
nclude <avr/io.h>
efine F_CPU   16000000UL
nclude <util/delay.h>
efine cbi(reg,bit) (reg &= ~(1 << bit))
efine sbi(reg,bit) (reg |= (1 << bit))


nt8_t one_wire_reset() {

  sbi(DDRD,PD4);

  cbi(PORTD,PD4);
  _delay_us(480);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(100);

  uint8_t temp = PIND;

  _delay_us(380);

  temp = (temp & 0x10);
  uint8_t res = 0x00;

  if (temp == 0x00)
      res = 0x01;

  return res;


id one_wire_transmit_bit(uint8_t bit) {

  sbi(DDRD,PD4);

  cbi(PORTD,PD4);
  _delay_us(2);
```

```
  if (bit == 0x01) sbi(PORTD,PD4);
  if (bit == 0x00) cbi(PORTD,PD4);

  _delay_us(58);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(1);
  return;


id one_wire_transmit_byte(uint8_t byte) {

  uint8_t bit;
  uint8_t i = 0x08;

  while(i != 0){
      bit = (byte & 0x01);
      one_wire_transmit_bit(bit);
      byte = (byte >> 1);
      i--;
  }

  return;


nt8_t one_wire_receive_bit() {
  uint8_t bit,temp;

  sbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(2);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(10);
```

```
  temp = (PIND & 0x10);
  bit = 0x00;
  if (temp == 0x10) bit = 0x01;
  _delay_us(49);

  return bit;


nt8_t one_wire_receive_byte() {
  uint8_t bit;
  uint8_t byte = 0x00;
  uint8_t i = 0x08;

  while(i != 0){
      bit = one_wire_receive_bit();
      byte = (byte >> 1);
      if (bit == 0x01) bit = 0x80;
      byte = (byte | bit);
      i--;
  }

  return byte;


t main(void)

  // Set DDRB and DDRD registers for output
  DDRB = 0xFF;
  DDRD = 0xFF;

  // Declare variables for temperature processing
  uint8_t temperatureLow, temperatureHigh, temperatureSign;
  uint16_t temperatureFinal, temperatureHigh16, temperatureFinalOutput;

  while (1)
  {
      // Check if device is connected
      if (!one_wire_reset()) {
```

```
            temperatureFinalOutput = 0x8000;
            continue;
        }

        // Send commands to request temperature reading
        one_wire_transmit_byte(0xCC);       // Send command 0xCC
        one_wire_transmit_byte(0x44);       // Send command 0x44

        // Wait for temperature conversion to complete
        while(one_wire_receive_bit() != 0x01);

        // Recheck if device is connected
        if (!one_wire_reset()) {
            temperatureFinalOutput = 0x8000;
            continue;
        }

        // Send commands to read temperature value
        one_wire_transmit_byte(0xCC);       // Send command 0xCC
        one_wire_transmit_byte(0xBE);       // Send command 0xBE

        // Receive temperature bytes
        temperatureLow = one_wire_receive_byte();
        temperatureHigh = one_wire_receive_byte();
        temperatureSign = temperatureHigh & 0xF8;
        temperatureHigh16 = temperatureHigh << 8;
        temperatureFinal = temperatureHigh16 + temperatureLow;

        // Check if temperature is negative or positive
        if (temperatureSign == 0xF8)
            temperatureFinalOutput = ~(temperatureFinal) + 1;   // Two's
mplement
        else
            temperatureFinalOutput = temperatureFinal;
    }
```

Ας αναλύσουμε τα βασικά μέρη του κώδικα:

1. Συνάρτηση **one_wire_reset()**:

- Αυτή η λειτουργία αρχικοποιεί και ελέγχει τη σύνδεση με τη συσκευή στον ακροδέκτη PD4.
- Ακολουθεί το πρωτόκολλο ενός καλωδίου για την προετοιμασία της συσκευής.
- Επιστρέφει 0x01 εάν εντοπιστεί μια συσκευή, 0x00 διαφορετικά.

2. Συνάρτηση **one_wire_transmit_bit()**:
- Στέλνει ένα μόνο bit στη συσκευή ενός καλωδίου στο PD4 χρησιμοποιώντας το One Wire πρωτόκολλο.

3. Συνάρτηση **one_wire_transmit_byte()**:
- Μεταδίδει ένα πλήρες byte στη συσκευή ενός καλωδίου καλώντας one_wire_transmit_bit για κάθε bit.

4. Συνάρτηση **one_wire_receive_bit()**:
- Λαμβάνει ένα μόνο bit από τη συσκευή ενός καλωδίου στο PD4 χρησιμοποιώντας το πρωτόκολλο ενός καλωδίου.

5. Συνάρτηση **one_wire_receive_byte()**:
- Λαμβάνει ένα πλήρες byte καλώντας το one_wire_receive_bit για κάθε bit.

6. **Main()** συνάρτηση:
- Ορίζει τους καταχωρητές DDRB και DDRD για έξοδο.
- Εισάγει έναν άπειρο βρόχο για συνεχή ανάγνωση των τιμών θερμοκρασίας.
- Ελέγχει εάν μια συσκευή είναι συνδεδεμένη χρησιμοποιώντας one_wire_reset(). Εάν όχι, ορίζει τη θερμοκρασίαFinalOutput σε 0x8000 και συνεχίζει.
- Στέλνει εντολές για να ζητήσει ένδειξη θερμοκρασίας και περιμένει να ολοκληρωθεί η μετατροπή.
- Ελέγχει ξανά τη σύνδεση της συσκευής και διαβάζει την τιμή θερμοκρασίας.
- Επεξεργάζεται τα δεδομένα θερμοκρασίας με βάση τις προδιαγραφές DS18B20.
- Η τελική τιμή θερμοκρασίας (temperatureFinalOutput) αποθηκεύεται ανάλογα με το αν είναι θετική ή αρνητική.

Είναι σημαντικό να σημειωθεί ότι ο κώδικας δεν χρησιμοποιεί τις μετρήσεις κάπου, απλά αποτελεί βάση για το Ζήτημα 7.2.

Στον συγκεκριμένο κώδικα, οι ορισμοί του «sbi» (set bit στον καταχωρητή) και του «cbi» (clear bit στον καταχωρητή) χρησιμοποιούνται για τον χειρισμό συγκεκριμένων bit στις τιμές του καταχωρητή. Αυτές οι μακροεντολές χρησιμοποιούνται για να αντικατοπτρίζουν στενά τις λειτουργίες που μπορούν να εκτελεστούν στη γλώσσα assembly.

1. **sbi (set bit στον καταχωρητή)** Μακροεντολή:
   - Ορισμός: `#define sbi(reg, bit) (reg |= (1 << bit))`

- Αυτή η μακροεντολή ορίζει το καθορισμένο bit στον δεδομένο καταχωρητή χρησιμοποιώντας μια λειτουργία bitwise OR με μια μάσκα δυαδικών ψηφίων που έχει μόνο το bit προορισμού ορισμένο σε 1.

2. **cbi (clear bit στον καταχωρητή)** Μακροεντολή:
   - Ορισμός: `#define cbi(reg, bit) (reg &= ~(1 << bit))`
   - Αυτή η μακροεντολή διαγράφει το καθορισμένο bit στον δεδομένο καταχωρητή χρησιμοποιώντας μια λειτουργία bitwise AND με μια μάσκα δυαδικών ψηφίων που έχει μόνο το bit-στόχο ορισμένο στο 0.

Αυτές οι μακροεντολές έχουν σχεδιαστεί για να μοιάζουν πολύ με τις αντίστοιχες εντολές assembly για τη ρύθμιση και την εκκαθάριση των bit σε έναν καταχωρητή. Ο σκοπός της χρήσης αυτών των μακροεντολών είναι να διατηρηθεί η δομή του κώδικα παρόμοια με τις λειτουργίες χαμηλού επιπέδου που εκτελούνται στη γλώσσα assembly.

Ζήτημα 7.2

Γράψτε πρόγραμμα σε C που με την χρήση της προηγούμενης ρουτίνας να απεικονίζει τη θερμοκρασία σε C στο LCD display σε δεκαδική τιμή τριών ψηφίων με το πρόσημο (-55C έως +125C). Επίσης στην περίπτωση που δεν υπάρχει συσκευή συνδεδεμένη να εμφανίζει το μήνυμα “NO Device”.

```
nclude <avr/io.h>
efine F_CPU   16000000UL
nclude <util/delay.h>

efine cbi(reg,bit) (reg &= ~(1 << bit))
efine sbi(reg,bit) (reg |= (1 << bit))

efine PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
efine TWI_READ 1 // reading from twi device
efine TWI_WRITE 0 // writing to twi device
efine SCL_CLOCK 100000L // twi clock in Hz
Fscl=Fcpu/(16+2*TWBR0_VALUE*PRESCALER_VALUE)
efine TWBR0_VALUE ((F_CPU/SCL_CLOCK)-16)/2

 PCA9555 REGISTERS
pedef 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 -------------------
efine TW_START 0x08
efine TW_REP_START 0x10

---------------- Master Transmitter ----------------------
efine TW_MT_SLA_ACK 0x18
efine TW_MT_SLA_NACK 0x20
efine TW_MT_DATA_ACK 0x28

---------------- Master Receiver ----------------
efine TW_MR_SLA_ACK 0x40
efine TW_MR_SLA_NACK 0x48
efine TW_MR_DATA_NACK 0x58

efine TW_STATUS_MASK 0b11111000
efine TW_STATUS (TWSR0 & TW_STATUS_MASK)

nt8_t buffer;

initialize TWI clock
id 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)

```
signed char twi_readAck(void)

  TWCR0 = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
  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

```
signed 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
  // 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
id 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))
ntinue;

      // 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_NA

      {
          /* device busy, send stop condition to terminate write operat

          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 writ
iled
signed 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

signed char twi_rep_start(unsigned char address)

  return twi_start( address );


 Terminates the data transfer and releases the twi bus
id 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));


signed char twi_readNak(void)

  TWCR0 = (1<<TWINT) | (1<<TWEN);
  while(!(TWCR0 & (1<<TWINT)));
```

```
  return TWDR0;


id 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();


nt8_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;


id write_2_nibbles(uint8_t input){

  uint8_t temp = input;
  uint8_t pin = buffer;
  pin &= 0x0F;
  input &=0xF0;
  input |= pin;
  buffer = input;
  buffer |= 0x08;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  buffer &= 0xF7;
  PCA9555_0_write(REG_OUTPUT_0, buffer);
```

```
  input = temp;
  input &= 0x0F;
  input = input << 4;
  input |= pin;
  buffer = input;
  buffer |= 0x08;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  buffer &= 0xF7;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  return;


id LCD_data(uint8_t x){

  buffer |= 0x04;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  write_2_nibbles(x);
  _delay_us(250);
  return;



id LCD_command(uint8_t x){

  buffer &= 0xFB;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  write_2_nibbles(x);
  _delay_us(250);
  return;



id LCD_clear_display(){

  uint8_t x = 0x01;
```

```
  LCD_command(x);
  _delay_ms(5);



id LCD_init(void){

  _delay_ms(200);

  buffer = 0x30;
  buffer |= 0x08;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  buffer &= 0xF7;
  PCA9555_0_write(REG_OUTPUT_0, buffer);
  _delay_us(250);

  buffer = 0x30;
  buffer |= 0x08;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  buffer &= 0xF7;
  PCA9555_0_write(REG_OUTPUT_0, buffer);
  _delay_us(250);

  buffer = 0x20;
  buffer |= 0x08;
  PCA9555_0_write(REG_OUTPUT_0, buffer);

  buffer &= 0xF7;
  PCA9555_0_write(REG_OUTPUT_0, buffer);
  _delay_us(250);

  LCD_command(0x28);
  LCD_command(0x0C);

  LCD_clear_display();
```

```
  LCD_command(0x06);



nt8_t one_wire_reset() {

  sbi(DDRD,PD4);

  cbi(PORTD,PD4);
  _delay_us(480);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(100);

  uint8_t temp = PIND;

  _delay_us(380);

  temp = (temp & 0x10);
  uint8_t res = 0x00;

  if (temp == 0x00)
      res = 0x01;

  return res;


id one_wire_transmit_bit(uint8_t bit) {

  sbi(DDRD,PD4);

  cbi(PORTD,PD4);
  _delay_us(2);

  if (bit == 0x01) sbi(PORTD,PD4);
  if (bit == 0x00) cbi(PORTD,PD4);
```

```
  _delay_us(58);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(1);
  return;


id one_wire_transmit_byte(uint8_t byte) {

  uint8_t bit;
  uint8_t i = 0x08;

  while(i != 0){
      bit = (byte & 0x01);
      one_wire_transmit_bit(bit);
      byte = (byte >> 1);
      i--;
  }

  return;


nt8_t one_wire_receive_bit() {
  uint8_t bit,temp;

  sbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(2);

  cbi(DDRD,PD4);
  cbi(PORTD,PD4);

  _delay_us(10);

  temp = (PIND & 0x10);
  bit = 0x00;
```

```
  if (temp == 0x10) bit = 0x01;
  _delay_us(49);

  return bit;


nt8_t one_wire_receive_byte() {
  uint8_t bit;
  uint8_t byte = 0x00;
  uint8_t i = 0x08;

  while(i != 0){
      bit = one_wire_receive_bit();
      byte = (byte >> 1);
      if (bit == 0x01) bit = 0x80;
      byte = (byte | bit);
      i--;
  }

  return byte;


id displayNoDeviceMessage() {
  LCD_clear_display();
  _delay_ms(2);
  LCD_data('N');
  LCD_data('O');
  LCD_data(' ');
  LCD_data('D');
  LCD_data('E');
  LCD_data('V');
  LCD_data('I');
  LCD_data('C');
  LCD_data('E');
  _delay_ms(500);



t main(){
```

```
twi_init();

LCD_init(); // Initialize the LCD
DDRB = 0xFF; // Set all pins of PORTB as output
DDRD = 0xFF; // Set all pins of PORTD as output

 // Declare variables for temperature processing
uint8_t temperatureLow, temperatureHigh, temperatureSign, temperature
uint16_t temperatureFinal, temperatureHigh16, temperatureFinalOutput;

 while (1)
{
     // Check if device is connected
    if (!one_wire_reset()) {
        displayNoDeviceMessage();
        continue;
    }

    // Send commands to request temperature reading
    one_wire_transmit_byte(0xCC);       // Send command 0xCC
    one_wire_transmit_byte(0x44);       // Send command 0x44

    // Wait for temperature conversion to complete
    while (one_wire_receive_bit() != 0x01);

    // Recheck if device is connected
    if (!one_wire_reset()) {
        displayNoDeviceMessage();
        continue;
    }

    // Send commands to read temperature value
    one_wire_transmit_byte(0xCC);       // Send command 0xCC
    one_wire_transmit_byte(0xBE);       // Send command 0xBE


    // Receive temperature bytes
    temperatureLow = one_wire_receive_byte();
```

```
        temperatureHigh = one_wire_receive_byte();
        temperatureSign = temperatureHigh & 0xF8;
        temperatureHigh16 = temperatureHigh << 8;
        temperatureFinal = temperatureHigh16 + temperatureLow;
        //temperatureDec = temperatureLow & 0x0F;

        // Check if temperature is negative or positive
        if (temperatureSign == 0xF8)
            temperatureFinalOutput = ~(temperatureFinal) + 1;    // Two's
mpliment
        else
            temperatureFinalOutput = temperatureFinal;

        /* DECIMAL PART CALCULATE */

        temperatureDec = temperatureFinalOutput & 0x0F;

        int decDigit1 = 0;
        int decDigit2 = 0;
        int decDigit3 = 0;
        int decDigit4 = 0;
        int decSum = 0;
        uint8_t bitOne;

        bitOne = temperatureDec & 0x08;
        if (bitOne == 8)
            decSum += 5000;

        bitOne = temperatureDec & 0x04;
        if (bitOne == 4)
            decSum += 2500;

        bitOne = temperatureDec & 0x02;
        if (bitOne == 2)
            decSum += 1250;

        bitOne = temperatureDec & 0x01;
        if (bitOne == 1)
            decSum += 625;
```

```
while (decSum >= 1000) {
    decDigit1++;
    decSum -= 1000;
}

while (decSum >= 100) {
    decDigit2++;
    decSum -= 100;
}

while (decSum >= 10) {
    decDigit3++;
    decSum -= 10;
}

while (decSum >= 1) {
    decDigit4++;
    decSum -= 1;
}

/* INTEGER PART CALCULATE */

temperatureFinalOutput = temperatureFinalOutput >> 4;

int intDigit100 = 0;
int intDigit10 = 0;
int intDigit1 = 0;

while (temperatureFinalOutput >= 100) {
    intDigit100++;
    temperatureFinalOutput -= 100;
}

while (temperatureFinalOutput >= 10) {
    intDigit10++;
    temperatureFinalOutput -= 10;
}
```

```
    while (temperatureFinalOutput >= 1) {
        intDigit1++;
        temperatureFinalOutput -= 1;
    }

    /* PRINT ON LCD */

    intDigit100 |= 0x30;
    intDigit10 |= 0x30;
    intDigit1 |= 0x30;
    decDigit1 |= 0x30;
    decDigit2 |= 0x30;
    decDigit3 |= 0x30;
    decDigit4 |= 0x30;

    LCD_command(0x01);
    _delay_ms(2);

    if (temperatureSign == 0xF8)
        LCD_data('-');
    else
        LCD_data('+');

    if (intDigit100 != 0x30)
        LCD_data(intDigit100);
    if (intDigit10 != 0x30)
        LCD_data(intDigit10);

    LCD_data(intDigit1);
    LCD_data('.');
    LCD_data(decDigit1);
    LCD_data(decDigit2);
    LCD_data(decDigit3);
    LCD_data(decDigit4);
    LCD_data('\xdf');
    LCD_data('C');
    _delay_ms(500);

}
```

Η συνάρτηση main() συνδυάζει τις λειτουργίες TWI, PCA9555 και LCD με τον υπάρχοντα κώδικα αισθητήρα θερμοκρασίας DS18B20. Αρχικοποιεί το TWI, ρυθμίζει τις ακίδες GPIO για LCD και άλλες εξόδους και εισάγει έναν βρόχο για να διαβάσει τη θερμοκρασία και να την εμφανίσει στην οθόνη LCD μέσω του Port Expander. Εάν ο αισθητήρας DS18B20 δεν εντοπιστεί, εμφανίζει ένα μήνυμα στην οθόνη LCD.

Αφού προσδιορίσει το πρόσημο της θερμοκρασίας, το πρόγραμμα προχωρά στην αποθήκευση της τελικής τιμής θερμοκρασίας σε μια μεταβλητή. Στη συνέχεια, αναλύει τη θερμοκρασία στα ακέραια και δεκαδικά μέρη της. Η διαδικασία εξαγωγής του δεκαδικού μέρους περιλαμβάνει τον υπολογισμό έως και τεσσάρων δεκαδικών ψηφίων, ανάλογα με τη θέση κάθε ψηφίου στο δεκαδικό μέρος. Η θέση του κάθε δεκαδικού ψηφίου φαίνεται στον παρακάτω πίνακα:

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
LS BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	BIT 15	**BIT 14**	**BIT 13**	**BIT 12**	**BIT 11**	**BIT 10**	**BIT 9**	**BIT 8**
MS BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

Για να επιτευχθεί αυτό, το πρόγραμμα δημιουργεί ένα άθροισμα και για τη θέση κάθε ψηφίου στο δεκαδικό μέρος, προστίθεται ένας αντίστοιχος αριθμός σε αυτό το άθροισμα. Οι αριθμοί που προστίθενται σε κάθε θέση πολλαπλασιάζονται επί 10000 για ευκολία στον υπολογισμό, διευκολύνοντας την καταμέτρηση χιλιάδων, εκατοντάδων, δεκάδων και μονάδων. Αυτή η προσέγγιση επιτρέπει την αποθήκευση κάθε αξίας θέσης στην αντίστοιχη μεταβλητή της.

Ομοίως, το ακέραιο μέρος υπολογίζεται μετά την εκτέλεση μιας λειτουργίας μετατόπισης δεξιά κατά τέσσερις θέσεις. Στη συνέχεια, το πρόγραμμα μετράει τις εκατοντάδες, τις δεκάδες και τις μονάδες στο ακέραιο μέρος, αποθηκεύοντάς τις στις αντίστοιχες μεταβλητές.

Συνοπτικά, αυτή η σχολαστική διαδικασία διασφαλίζει την ακριβή αναπαράσταση και τον χειρισμό τόσο των ακέραιων όσο και των δεκαδικών συνιστωσών της θερμοκρασίας. Η χρήση του πολλαπλασιασμού με το 10000 απλοποιεί τον χειρισμό των δεκαδικών ψηφίων, συμβάλλοντας σε έναν πιο συστηματικό και οργανωμένο υπολογισμό των τιμών θερμοκρασίας.