

Αισθητήρας θερμοκρασίας DS1820 στην κάρτα ntuAboard_G1

Εργαστήριο Μικροϋπολογιστών

Κριθαρίδης Κωνσταντίνος, *el21045*

Μπαλάτος Δημήτριος, *el21170*

26 Νοεμβρίου 2024

1 Ζήτημα 7.1

Αρχικά μετατρέπουμε τον δωσμένο κώδικα από Assembly σε C. Στην συνέχεια ορίζουμε νέα βοηθητική συνάρτηση `read_temp(void)` που διαβάζει την τιμή του DS1820 και επιστρέφει αντίστοιχη `float` τιμή.

Για διευκόλυνση, ορίσαμε τύπο `bool` και μακροεντολές `SET(x, b)` , `CLEAR(x, b)` για την γρήγορη αλλαγή κάποιου bit σε αριθμό.

```
1  /*
2   * main.c
3   *
4   * Created: 11/22/2024 11:04:17 AM
5   * Author: User
6   */
7
8  #include <xc.h>
9
10 #define F_CPU 16000000UL
11 #include<avr/io.h>
12 #include<avr/interrupt.h>
13 #include<util/delay.h>
14
15 typedef uint8_t bool;
16 #define true 1
17 #define false 0
18
19 #define SET(x, b)    do { (x) |=  1 << (b); } while (0)
20 #define CLEAR(x, b) do { (x) &= ~(1 << (b)); } while (0)
21
22 /* TEMPLATE CODE FROM GIVEN ASSEMBLY */
23 /* {{{ */
24 bool one_wire_reset (void)
25 {
26     SET(DDRD, 4); // set output
27
28     CLEAR(PORTD, 4); // sent 0
```

```

29     _delay_us(480);
30
31     CLEAR(DDRD, 4); // set input
32     CLEAR(PORTD, 4); // disable pull-up
33     _delay_us(100);
34
35     int tmp = PIND; // read PORTD
36     _delay_us(380);
37     return (tmp & (1 << 4)) == 0;
38 }
39
40 uint8_t one_wire_receive_bit (void)
41 {
42     SET(DDRD, 4); // Set output
43     CLEAR(PORTD, 4);
44     _delay_us(2);
45
46     CLEAR(DDRD, 4); // set input
47     CLEAR(PORTD, 4); // disable pull-up
48     _delay_us(10);
49
50     uint8_t ret = PIND & 1;
51     _delay_us(49);
52     return ret;
53 }
54
55 void one_wire_transmit_bit (uint8_t in)
56 {
57     SET(DDRD, 4); // Set output
58     CLEAR(PORTD, 4);
59     _delay_us(2);
60
61     in &= 1; // keep only LSB
62     if (in) SET(PORTD, 4);
63     else    CLEAR(PORTD, 4);
64
65     _delay_us(58);
66     CLEAR(DDRD, 4); // set input
67     CLEAR(PORTD, 4); // disable pull-up
68     _delay_us(1);
69 }
70
71 uint8_t one_wire_receive_byte (void)
72 {
73     uint8_t ret = 0;
74     for (int i=0; i<8; ++i)
75         ret |= (one_wire_receive_bit() << i);
76     return ret;
77 }
78
79 void one_wire_transmit_byte (uint8_t in)

```

```

80 {
81     for (int i=0; i<8; ++i, in >>= 1)
82         one_wire_transmit_bit(in);
83 }
84 /* }}} */
85
86 float read_temp (void)
87 {
88     int16_t ret = 0;
89
90     if (one_wire_reset() != 0) return 0x8000;
91     one_wire_transmit_byte(0xCC);
92     one_wire_transmit_byte(0x44);
93     while (one_wire_receive_bit() != 1);
94
95     if (one_wire_reset() != 0) return 0x8000;
96     one_wire_transmit_byte(0xCC);
97     one_wire_transmit_byte(0xBE);
98
99     ret |= one_wire_receive_byte() << 8;
100    ret |= one_wire_receive_byte();
101    return (float)(ret) / 2;
102 }
103
104 int main(void)
105 {
106     return 0;
107 }
108

```

2 Ζήτημα 7.2

Στην πλακέτα μας είχαμε αισθητήρα DS18B20.

Αντιγράφουμε τον παραπάνω κώδικα, με μία αλλαγή στην `read_temp(void)`: η συνάρτηση πλέον επιστρέφει σε `unsigned int` 16 bits την raw τιμή που επιστρέφει ο αισθητήρας, προκειμένου να διευκολυνθεί η εκτύπωση της θερμοκρασίας. Για την χρήση LCD οθόνης χρειάζεται να χρησιμοποιήσουμε port expander, άρα αντιγράφουμε και τον boilerplate κώδικα επικοινωνίας από παλιότερη άσκηση.

Δημιουργούμε νέα συνάρτηση για την εκτύπωση θερμοκρασίας `lcd_temp(val, precision)`: διαχωρίζουμε την raw τιμή σε ακέραιο και δεκαδικό μέρος, εκτυπώνουμε ολόκληρο το ακέραιο μέρος και μετά `precision` ψηφία του δεκαδικού. Στο τέλος εκτυπώνουμε το σύμβολο $^{\circ}C$.

Στην συνάρτηση `main`, διαβάζουμε συνέχεια τιμή θερμοκρασίας και, αν αυτή εκφράζει σφάλμα ανάγνωσης εκτυπώνουμε το ζητούμενο error message, αλλιώς εκτυπώνουμε την θερμοκρασία μέσω της συναρτήσης.

Για να επιταχύνουμε τον κώδικα εκτύπωσης, αντί να καθαρίζουμε την LCD μετακινούμε τον κέρσορα στην πρώτη στήλη της πρώτης γραμμής μέσω `lcd_command(instr)`.

```

1  /*
2   * main.c
3   *
4   * Created: 11/22/2024 11:05:01 AM
5   * Author: User
6   */
7
8  #include <xc.h>
9
10 #define F_CPU 16000000UL
11 #include<avr/io.h>
12 #include<avr/interrupt.h>
13 #include<util/delay.h>
14
15 typedef uint8_t bool;
16 #define true 1
17 #define false 0
18
19 /* LCD DISPLAY THROUGH PORT EXPANDER */
20 /* {{{ */
21 #define PCA9555_0_ADDRESS 0x40 //A0=A1=A2=0 by hardware
22 #define TWI_READ 1 // reading from twi device
23 #define TWI_WRITE 0 // writing to twi device
24 #define SCL_CLOCK 100000L // twi clock in Hz
25
26 //Fsc1=Fcpu/(16+2*TWBRO_VALUE*PRESCALER_VALUE)
27 #define TWBRO_VALUE ((F_CPU/SCL_CLOCK)-16)/2
28
29 #define NOP() do { __asm__ __volatile__ ( "nop "); } while (0)
30
31 // PCA9555 REGISTERS
32 typedef enum {
33     REG_INPUT_0 = 0,
34     REG_INPUT_1 = 1,
35     REG_OUTPUT_0 = 2,
36     REG_OUTPUT_1 = 3,
37     REG_POLARITY_INV_0 = 4,
38     REG_POLARITY_INV_1 = 5,
39     REG_CONFIGURATION_0 = 6,
40     REG_CONFIGURATION_1 = 7
41 } PCA9555_REGISTERS;
42
43 //----- Master Transmitter/Receiver -----
44 #define TW_START 0x08
45 #define TW_REP_START 0x10
46
47 //----- Master Transmitter -----
48 #define TW_MT_SLA_ACK 0x18
49 #define TW_MT_SLA_NACK 0x20
50 #define TW_MT_DATA_ACK 0x28
51

```

```

52 //----- Master Receiver -----
53 #define TW_MR_SLA_ACK 0x40
54 #define TW_MR_SLA_NACK 0x48
55 #define TW_MR_DATA_NACK 0x58
56
57 #define TW_STATUS_MASK 0b11111000
58 #define TW_STATUS (TWSRO & TW_STATUS_MASK)
59
60 //initialize TWI clock
61 void twi_init(void)
62 {
63     TWSRO = 0; // PRESCALER_VALUE=1
64     TWBRO = TWBRO_VALUE; // SCL_CLOCK 100KHz
65 }
66
67 // Read one byte from the twi device (request more data from device)
68 unsigned char twi_readAck(void)
69 {
70     TWCRO = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
71     while(!(TWCRO & (1<<TWINT)));
72     return TWDRO;
73 }
74
75 //Read one byte from the twi device, read is followed by a stop condition
76 unsigned char twi_readNak(void)
77 {
78     TWCRO = (1<<TWINT) | (1<<TWEN);
79     while(!(TWCRO & (1<<TWINT)));
80     return TWDRO;
81 }
82
83 // Issues a start condition and sends address and transfer direction.
84 // return 0 = device accessible, 1= failed to access device
85 unsigned char twi_start(unsigned char address)
86 {
87     uint8_t twi_status;
88
89     // send START condition
90     TWCRO = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
91
92     // wait until transmission completed
93     while(!(TWCRO & (1<<TWINT)));
94
95     // check value of TWI Status Register.
96     twi_status = TW_STATUS & 0xF8;
97     if ( (twi_status != TW_START) && (twi_status != TW_REP_START)) return 1;
98
99     // send device address
100     TWDRO = address;
101     TWCRO = (1<<TWINT) | (1<<TWEN);
102

```

```

103 // wait until transmission completed and ACK/NACK has been received
104 while(!(TWCRO & (1<<TWINT)));
105 // check value of TWI Status Register.
106 twi_status = TW_STATUS & 0xF8;
107 if ( (twi_status != TW_MT_SLA_ACK) && (twi_status != TW_MR_SLA_ACK) )
108 {
109     return 1;
110 }
111 return 0;
112 }
113
114 // Send start condition, address, transfer direction.
115 // Use ack polling to wait until device is ready
116 void twi_start_wait(unsigned char address)
117 {
118     uint8_t twi_status;
119     while ( 1 )
120     {
121         // send START condition
122         TWCRO = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
123
124         // wait until transmission completed
125         while(!(TWCRO & (1<<TWINT)));
126
127         // check value of TWI Status Register.
128         twi_status = TW_STATUS & 0xF8;
129         if ( (twi_status != TW_START) && (twi_status != TW_REP_START))
130             ↪ continue;
131
132         // send device address
133         TWDRO = address;
134         TWCRO = (1<<TWINT) | (1<<TWEN);
135
136         // wait until transmission completed
137         while(!(TWCRO & (1<<TWINT)));
138
139         // check value of TWI Status Register.
140         twi_status = TW_STATUS & 0xF8;
141         if ( (twi_status == TW_MT_SLA_NACK) || (twi_status
142             ↪ ==TW_MR_DATA_NACK) )
143         {
144             /* device busy, send stop condition to terminate write
145             ↪ operation */
146             TWCRO = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
147
148             // wait until stop condition is executed and bus released
149             while(TWCRO & (1<<TWSTO));
150             continue;
151         }
152         break;
153     }
154 }

```

```

151 }
152
153 // Send one byte to twi device, Return 0 if write successful or 1 if write failed
154 unsigned char twi_write( unsigned char data )
155 {
156     // send data to the previously addressed device
157     TWDRO = data;
158     TWCRO = (1<<TWINT) | (1<<TWEN);
159
160     // wait until transmission completed
161     while(!(TWCRO & (1<<TWINT)));
162     if( (TW_STATUS & 0xF8) != TW_MT_DATA_ACK) return 1;
163     return 0;
164 }
165
166 // Send repeated start condition, address, transfer direction
167 //Return: 0 device accessible
168 // 1 failed to access device
169 unsigned char twi_rep_start(unsigned char address)
170 {
171     return twi_start( address );
172 }
173
174 // Terminates the data transfer and releases the twi bus
175 void twi_stop(void)
176 {
177     // send stop condition
178     TWCRO = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
179
180     // wait until stop condition is executed and bus released
181     while(TWCRO & (1<<TWSTO));
182 }
183
184 uint8_t LAST;
185
186 void PCA9555_0_write(PCA9555_REGISTERS reg, uint8_t value)
187 {
188     twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
189     twi_write(reg);
190     twi_write(value);
191     twi_stop();
192     LAST = value;
193     //if (reg != REG_CONFIGURATION_0) exit(0);
194 }
195
196 uint8_t PCA9555_0_read(PCA9555_REGISTERS reg)
197 {
198     uint8_t ret_val;
199     twi_start_wait(PCA9555_0_ADDRESS + TWI_WRITE);
200     twi_write(reg);
201     twi_rep_start(PCA9555_0_ADDRESS + TWI_READ);

```

```

202     ret_val = twi_readNak();
203     twi_stop();
204     return ret_val;
205 }
206
207 void flash ()
208 {
209     _delay_us(50);
210     uint8_t tmp = PCA9555_0_read(REG_INPUT_0);
211     PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 3));
212     _delay_us(50);
213     PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 3));
214 }
215
216 void write_2_nibbles(uint8_t data){
217     uint8_t temp = LAST & 0x0f;
218     uint8_t out = data & 0xf0 | temp;
219     PCA9555_0_write(REG_OUTPUT_0, out);
220     flash();
221
222     out = (data << 4) & 0xf0 | temp;
223     PCA9555_0_write(REG_OUTPUT_0, out);
224     flash();
225 }
226
227 void lcd_data (uint8_t data)
228 {
229     uint8_t tmp = LAST;
230     PCA9555_0_write(REG_OUTPUT_0, tmp | (1 << 2));
231     write_2_nibbles(data);
232     _delay_us(500);
233 }
234
235 void lcd_command (uint8_t instr)
236 {
237     uint8_t tmp = LAST;
238     PCA9555_0_write(REG_OUTPUT_0, tmp & ~(1 << 2));
239     write_2_nibbles(instr);
240     _delay_us(500);
241 }
242
243 void lcd_clear_display(){
244     lcd_command(0x01);
245     _delay_ms(200);
246 }
247
248 void lcd_init ()
249 {
250     _delay_ms(200);
251
252     uint8_t out = 0x30;

```



```

253     for (int i=0; i<3; ++i) {
254         PCA9555_0_write(REG_OUTPUT_0, out);
255         flash();
256         _delay_us(250);
257     }
258     PCA9555_0_write(REG_OUTPUT_0, 0x20);
259     flash();
260     _delay_us(250);
261
262     lcd_command(0x28);
263     lcd_command(0x0c);
264     lcd_clear_display();
265     lcd_command(0x06);
266 }
267
268 void lcd_number(uint16_t number){
269     uint8_t digits[18];
270     int i = 0;
271     for (; number; number >>= 1)
272         digits[i++] = number & 1;
273     for (i-=1; i>=0; --i)
274         lcd_data(digits[i] + '0');
275 }
276
277 void lcd_string (const char* str)
278 {
279     lcd_command(0x02);
280     for (; *str; str++)
281         lcd_data(*str);
282 }
283
284 void lcd_temp (int16_t val, int precision)
285 {
286     char tmp[17];
287     int idx = 0;
288     lcd_command(0x02);
289
290     if (val < 0) {
291         lcd_data('-');
292         val = -val;
293     }
294
295     int int_part = val >> precision;
296     float frac_part = ((float)(val) / (1 << precision)) - int_part;
297
298     if (int_part) {
299         for (; int_part; int_part /= 10)
300             tmp[idx++] = (int_part % 10) + '0';
301         for (idx -= 1; idx >= 0; --idx)
302             lcd_data(tmp[idx]);
303     } else {

```

```

304         lcd_data('0');
305     }
306
307     lcd_data('.');
308     for (int i=0; i<precision; ++i) {
309         frac_part *= 10;
310         lcd_data((int)(frac_part) + '0');
311         frac_part -= (int)(frac_part);
312     }
313
314     lcd_data(0b11011111); lcd_data('C');
315 }
316 /* }}} */
317
318 #define SET(x, b)    do { (x) |=  1 << (b); } while (0)
319 #define CLEAR(x, b) do { (x) &= ~(1 << (b)); } while (0)
320
321 /* TEMPLATE CODE FROM GIVEN ASSEMBLY */
322 /* {{{ */
323 bool one_wire_reset (void)
324 {
325     SET(DDRD, 4); // set output
326
327     CLEAR(PORTD, 4); // sent 0
328     _delay_us(480);
329
330     CLEAR(DDRD, 4); // set input
331     CLEAR(PORTD, 4); // disable pull-up
332     _delay_us(100);
333
334     int tmp = PIND; // read PORTD
335     _delay_us(380);
336     return (tmp & (1 << 4)) == 0;
337 }
338
339 uint8_t one_wire_receive_bit (void)
340 {
341     SET(DDRD, 4); // Set output
342     CLEAR(PORTD, 4);
343     _delay_us(2);
344
345     CLEAR(DDRD, 4); // set input
346     CLEAR(PORTD, 4); // disable pull-up
347     _delay_us(10);
348
349     uint8_t ret = PIND & (1 << 4);
350     _delay_us(49);
351     return ret >> 4;
352 }
353
354 void one_wire_transmit_bit (uint8_t in)

```

```

355 {
356     SET(DDRD, 4); // Set output
357     CLEAR(PORTD, 4);
358     _delay_us(2);
359
360     in &= 1; // keep only LSB
361     if (in) SET(PORTD, 4);
362     else    CLEAR(PORTD, 4);
363
364     _delay_us(58);
365     CLEAR(DDRD, 4); // set input
366     CLEAR(PORTD, 4); // disable pull-up
367     _delay_us(1);
368 }
369
370 uint8_t one_wire_receive_byte (void)
371 {
372     uint8_t ret = 0;
373     for (int i=0; i<8; ++i)
374         ret |= (one_wire_receive_bit() << i);
375     return ret;
376 }
377
378 void one_wire_transmit_byte (uint8_t in)
379 {
380     for (int i=0; i<8; ++i, in >>= 1)
381         one_wire_transmit_bit(in);
382 }
383 /* }}} */
384
385 #define TEMP_ERR 0x8000
386
387 int16_t read_temp (void)
388 {
389     int16_t ret = 0;
390
391     if (one_wire_reset() != 1) return TEMP_ERR;
392     one_wire_transmit_byte(0xCC);
393     one_wire_transmit_byte(0x44);
394     // lcd_string("Waiting...");
395     while (one_wire_receive_bit() != 1);
396     // lcd_string("Finished!");
397
398     if (one_wire_reset() != 1) return TEMP_ERR;
399     one_wire_transmit_byte(0xCC);
400     one_wire_transmit_byte(0xBE);
401
402     ret |= one_wire_receive_byte();
403     ret |= one_wire_receive_byte() << 8;
404     return ret;
405 }

```

```

406
407 const char err_msg[] = "NO Device";
408
409
410 int main(void) {
411     twi_init();
412     PCA9555_0_write(REG_CONFIGURATION_0, 0x00); //Set EXT_PORT0 as output
413     lcd_init();
414     lcd_clear_display();
415
416     bool is_prev_err = false;
417     while (1) {
418         int16_t temp = read_temp();
419         if (temp != TEMP_ERR) {
420             is_prev_err = false;
421             lcd_temp(temp, 4);
422         } else if (!is_prev_err) {
423             is_prev_err = true;
424             lcd_string(err_msg);
425         }
426         _delay_ms(50);
427     }
428
429     return 0;
430 }
431

```