

Universitatea Tehnică “Gheorghe Asachi” din Iași

Facultatea de Automatică și Calculatoare

Domeniul: Ingineria sistemelor

Specializarea: Automatică și Informatică Aplicată

Anul universitar 2022-2023

Proiect la disciplina

Programarea aplicațiilor de timp real

Studenți :

Botezatu Ioana Flavia - 1405A
Mateșescu Niki - 1405B

Profesor coordonator :

Ş.I.dr.ing. Cătălin Brăescu

Cuprins

1. Introducere.....	4
2. Implementare.....	5
3. Concluzii și direcții viitoare.....	13

1. Introducere - Cerințe și specificații - Tema proiectului

În cadrul proiectului s-a dorit proiectarea și realizarea unei aplicații încorporată bazată pe sistemul de operare FreeRTOS și microcontroller-ul dsPIC33FJ128MC802.

Aplicația trebuia să comande o trapă prin intermediul unui servomotor, pe baza informației de temperatură achiziționate prin intermediul unui senzor de temperatură digital DS18S20/DS18B20.

Aplicația ar fi trebuit să respecte următoarele detalii de implementare:

- Să fie pornită respectiv opriță prin intermediul butonului SW1 de pe plăcuță. Starea aplicației să fie indicată de un LED conectat la pinul RB0. LED-ul va fi aprins atunci când aplicația este pornită și va fi aprins intermitent când aplicația este opriță.

- Temperatura să fie achiziționată de senzorul DS18S20 sau DS18B20, deschiderea trapei fiind controlată prin servomotor. Temperaturii de 25°C îi va corespunde poziția centrală a servomotorului. Servomotorul s-ar deplasa între pozițiile extreme pentru limitele de temperatură (20°C-30°C). Senzorul de temperatură fiind conectat la pinul RB2 al Microcontroller-ului.

- Aplicația punea la dispoziție pe lângă modul automat de comandă a servomotorului și un mod manual de control prin intermediul unei tensiuni externe aplicate pe pinul RB3. Trecerea dintr-un mod de lucru în altul s-ar fi făcut prin intermediul meniului de comenzi implementat pe PC. În modul de lucru manual servomotorul va fi poziționat pe mijloc pentru o tensiune de 2V și se va deplasa în pozițiile extreme pentru valorile limită de 1V și 3V. Modul de lucru va fi indicat vizual prin intermediul unui LED conectat la pinul RB1. LED-ul va fi aprins în modul de lucru automat și stins în modul de lucru manual.

- Aplicația va permite comunicația prin interfață serială cu PC-ul. Placa cu dsPIC va receptiona comenzi sub forma unor caractere și va transmite drept răspunsuri de caractere.

- Pe un ecran LCD vor fi afișate informații privind temperatura, modul de lucru, tensiunea achiziționată de la pinul RB3 precum și ultima comandă primită.

- Aplicația va pune la dispoziția utilizatorului un meniu cu următoarele comenzi:

- interogare mod de lucru
- comutare mod de lucru automat/manual (modul de lucru predefinit este cel automat);
- interogare temperatură.

Conexiunile la portul B al microcontroller-ului vor fi făcute conform tabelului de mai jos:

RB 15	RB 14	RB 13	RB 12	RB 11	RB 10	RB 9	RB 8	RB 7	RB 6	RB 5	RB 4	RB 3	RB 2	RB 1	RB 0
L	L	L	L	P	P	RS	RS	L	L	L	L	AN	TEMP	L	L
C	C	C	C	W	W	232	232	C	C	C	C			E	E
D	D	D	D	M	M			D	D	D	D			D	D

MD – mod de lucru

ST – stare aplicație

T – senzor temperatură

U – tensiune externă

2. Implementare

În cadrul fiecărui laborator am integrat rând pe rând fiecare element hardware după cum urmează:

În prima săptămână am studiat lucrul cu întreruperi externe. În acest sens am introdus în aplicația noastră o întrerupere acționată de apăsarea butonului SW1 de pe plăcuță. Tratarea întreruperii era reprezentată prin modificarea stării LED-ului RB15.

```
void __attribute__((interrupt, no_auto_psv)) _INT0Interrupt(void)
{
    _RB15 = ~_RB15;
    _INT0IF = 0; // Resetam flagul corespunzator întreruperii
}
```

În săptămâna a doua am discutat despre generarea de semnale PWM. De asemenea am aflat că pentru servomotor avem nevoie de $T = 10 \div 20\text{ms}$, iar $d = 1 \div 2\text{ms}$ (Am ales $T = 10$ și $d = 1$). În cadrul proiectului a fost nevoie de calcularea prescalerului, deoarece $P1TPER > 32000$ (s-a calculat prescaler 1:16).

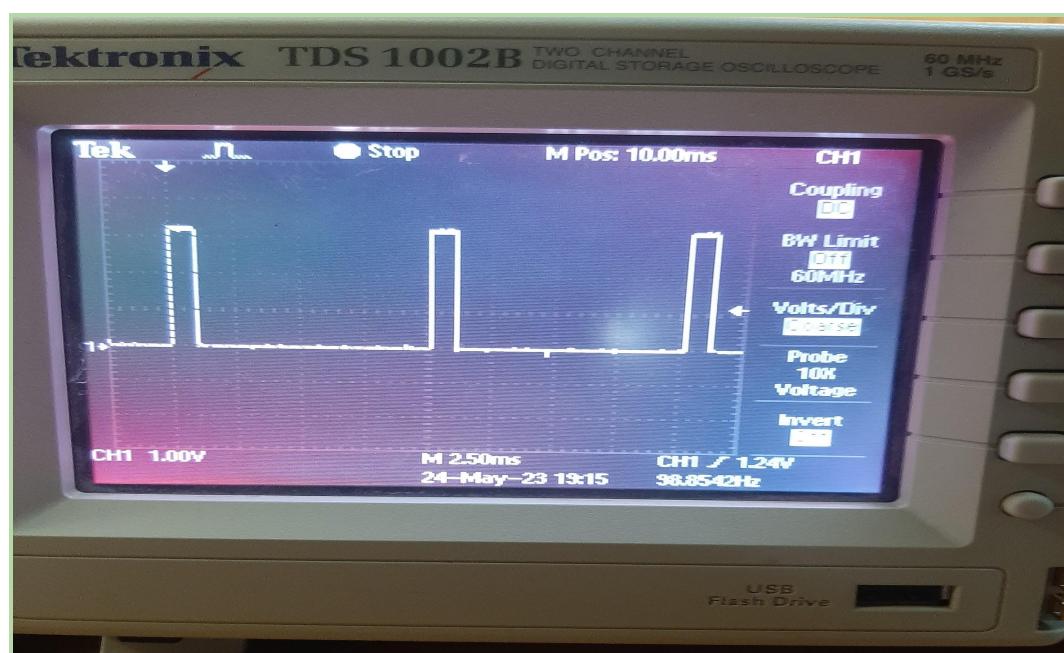
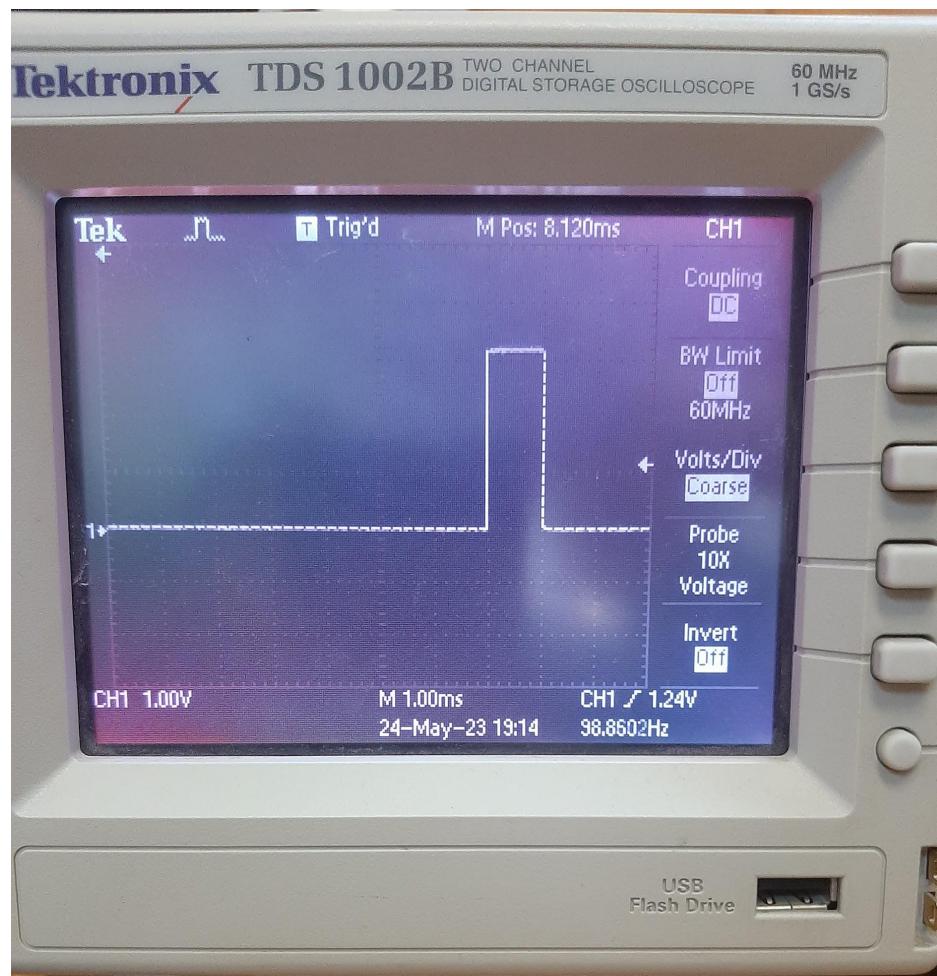
```
void init_PWM1()
{
    P1TCONbits.PTOPS = 0; // Timer base output scale
    P1TCONbits.PTMOD = 0; // Free running
    P1TMRbits.PTDIR = 0; // Numara in sus pana cand timerul = perioada
    P1TMRbits.PTMR = 0; // Baza de timp
    P1TCONbits.PTCKPS = 2; //prescaler 1:16
    P1DC3 = 5000;
    //avem T=10; d=1; momentan
    P1TPER = 25000;
    PWM1CON1bits.PMOD3 = 1; // Canalele PWM1H si PWM1L sunt independente
```

```
PWM1CON1bits.PEN3H = 1; // Pinul PWM1H setat pe iesire PWM rb10
PWM1CON1bits.PEN3L = 0; // Pinul PWM1L setat pe I/O general purpose rb11
```

```
PWM1CON2bits.UDIS = 1; // Disable Updates from duty cycle and period buffers
```

```
P1TCONbits.PTEN = 1; /* Enable the PWM Module */
}
```

Am verificat codul cu ajutorul osciloscopului. După cum se poate vedea, T și d au valori conform celor alese de noi.



În săptămâna a 3-a am studiat despre realizarea de achiziții AD. Am folosit un potențiometru conectat la pinul RB3 pentru a verifica dacă registrul de control ADC1BUF0 își modifica valoarea, iar conversia A/D a funcționat.

```

void initAdc1(void)
{
    AD1CON1bits.AD12B = 1; // conversie AD pe 12 biti
    AD1CON1bits.FORM = 0; // rezultat conversie integer
    AD1CON1bits.SSRC = 2; // timerul 3 starteaza conversia
    AD1CON1bits.ASAM = 1; // incepe esantionarea noii valori imediat dupa
    terminarea // unei conversii

    AD1CON2bits.CSCNA = 1; // scaneaza intrarile pe CH0+ in timpul achizitiei A
    AD1CON2bits.CHPS = 0; // converteste doar CH0
    AD1CON2bits.SMPI = 0; // incrementeaza adresa DMA dupa terminarea fiecarei
    // conversii

    AD1CON3bits.ADRC = 0; // foloseste ceasul magistralei
    AD1CON3bits.ADCS = 63; // Timpul necesar unei conversii este de 19.2 us
    // Ceasul pentru conversia AD are formula
    Tad=Tcy*(adcs+1)
    // Tad=Tcy*(adcs+1)=(1/40)*64=1.6us

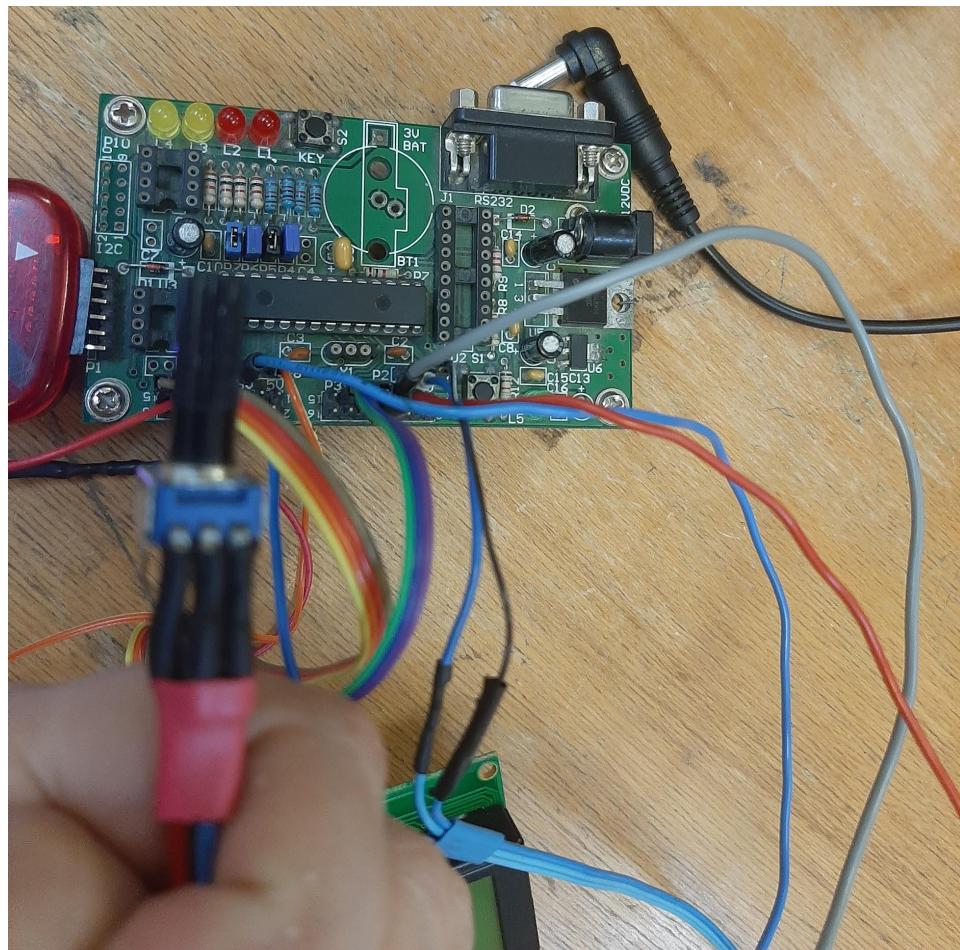
    // Se seteaza intrarile analogice
    AD1CSSLbits.CSS5 = 1; // Selectam intrarea analogica AN5(RB3) pentru a fi
    scanata

    // Scriem registrul de configurare al portului
    // Se va folosi doar registrul low al portului de configurare deoarece dsPIC33fJ128MC802
    // nu are implementati mai mult de 6 pini pentru ADC
    AD1PCFGL=0xFFFF; // Setam toti pinii portului ADC1 pe modul digital,
    // si activeaza citirea la intrarea portului
    AD1PCFGLbits.PCFG5 = 0;// Setam pinul AN5(RB3) pe intrare analogica,
    // ADC verifică voltajele pe acel pin (achiziție AD)
    IFS0bits.AD1IF = 0; // Reseteaza flag-ul intreruperii convertorului AD
    IPC3bits.AD1IP = 6; // Seteaza prioritatea intreruperii convertorului AD
    IEC0bits.AD1IE = 1; // Permite intreruperea convertorului AD

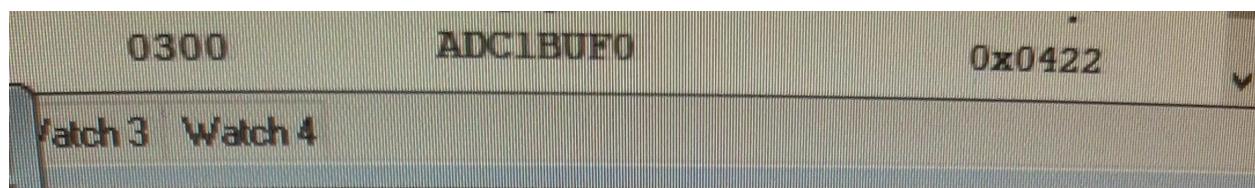
    AD1CON1bits.ADON = 1;
}

```

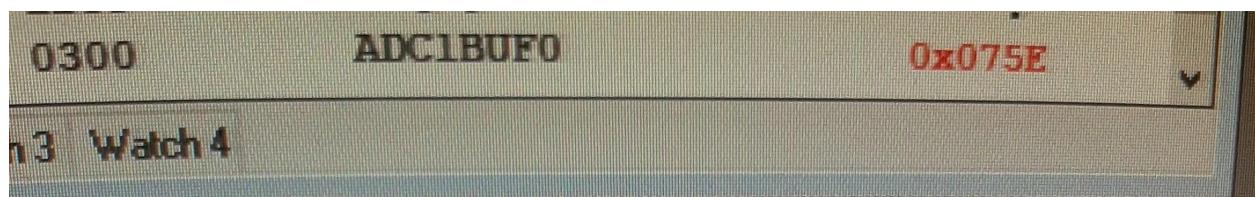
Am conectat un potențiometru la microcontroller-ul dsPIC33FJ128MC802.



Prima dată am obținut valoare de mai jos pe registrul ADC1BUFO :



După învârtirea potențiometrului se poate observa ca valoarea registrului ADC1BUFO s-a modificat :



În săptămâna a patra ne-am ocupat de afișarea informațiilor pe LCD alfanumeric.
Semnificația biților LCD-ului este următoarea :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
G N D	5V	con tra s t	control		Pini de date										BL	

Am ales RS - RB6, RW - RB5, E - RB4.

```
#define LCD_ON _RB7
#define LCD_RS _RB6
#define LCD_RW _RB5
#define LCD_E _RB4
#define LCD_BL _RB3
```

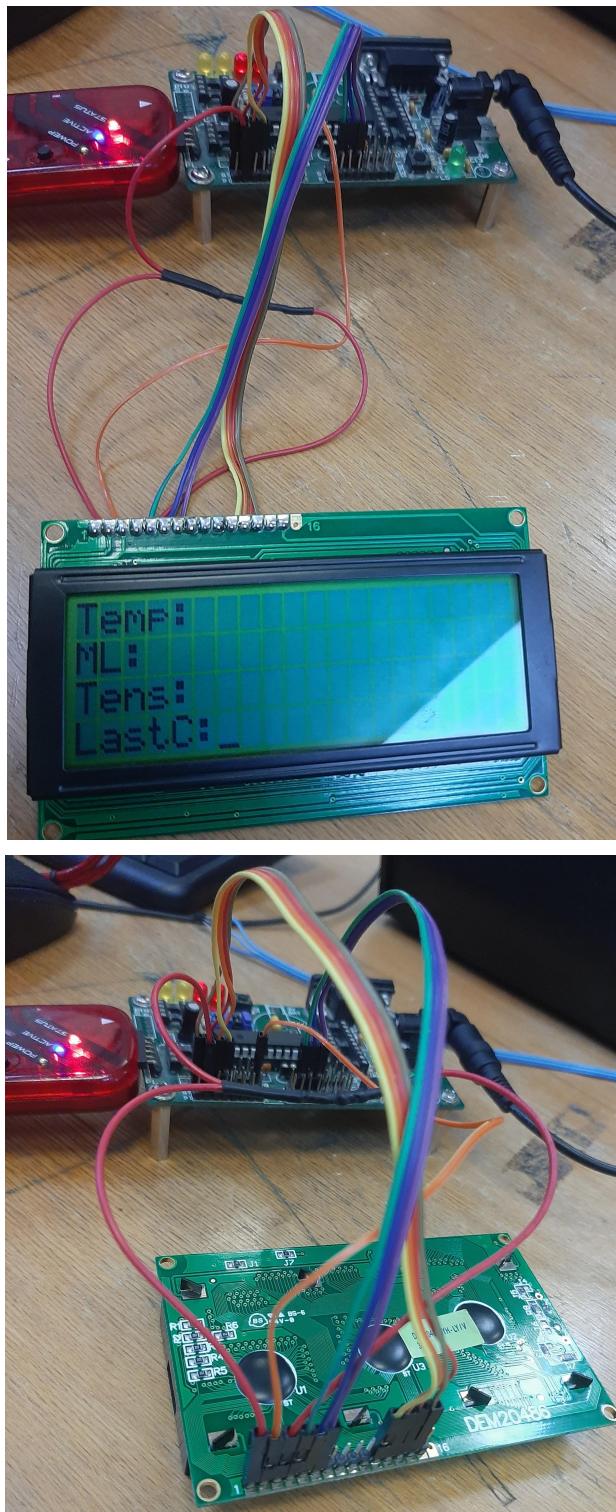
Am afișat pe LCD următoarele :

```
// informații privind temperatura,
LCD_line(1);
LCD_printf("Temp:");

// modul de lucru,
LCD_line(2);
LCD_printf("ML:");

// tensiunea achiziționată de la pinul RB3 precum
LCD_line(3);
LCD_printf("Tens:");

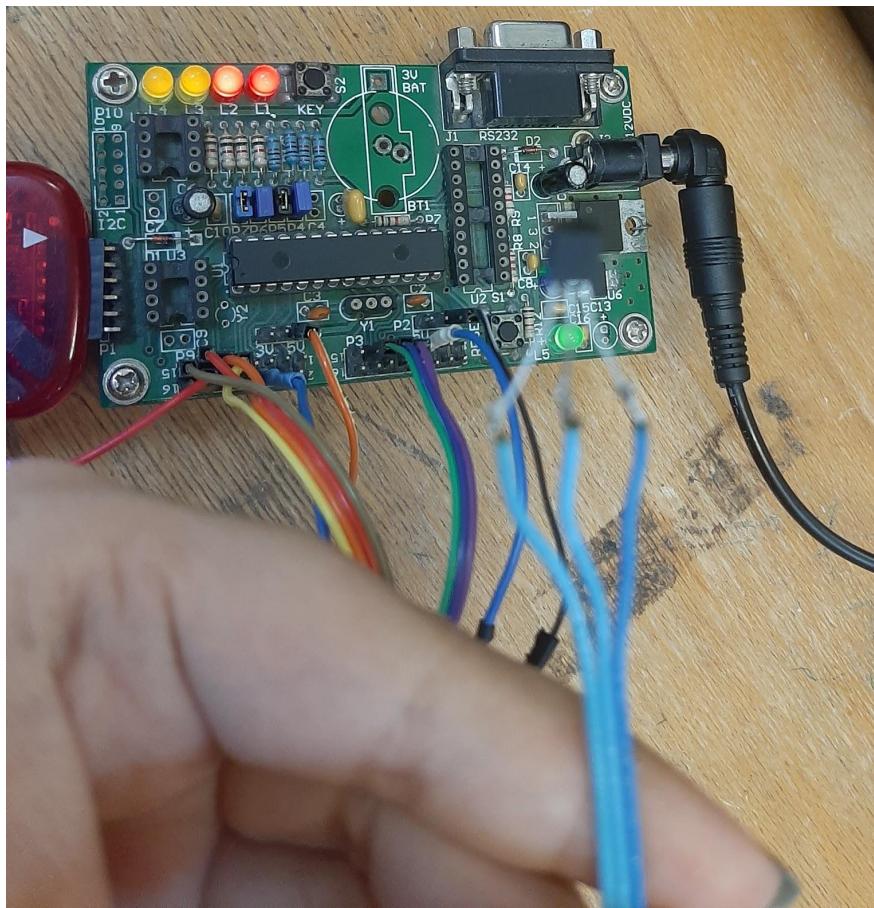
// și ultima comandă primită
LCD_line(4);
LCD_printf("LastC:");
```

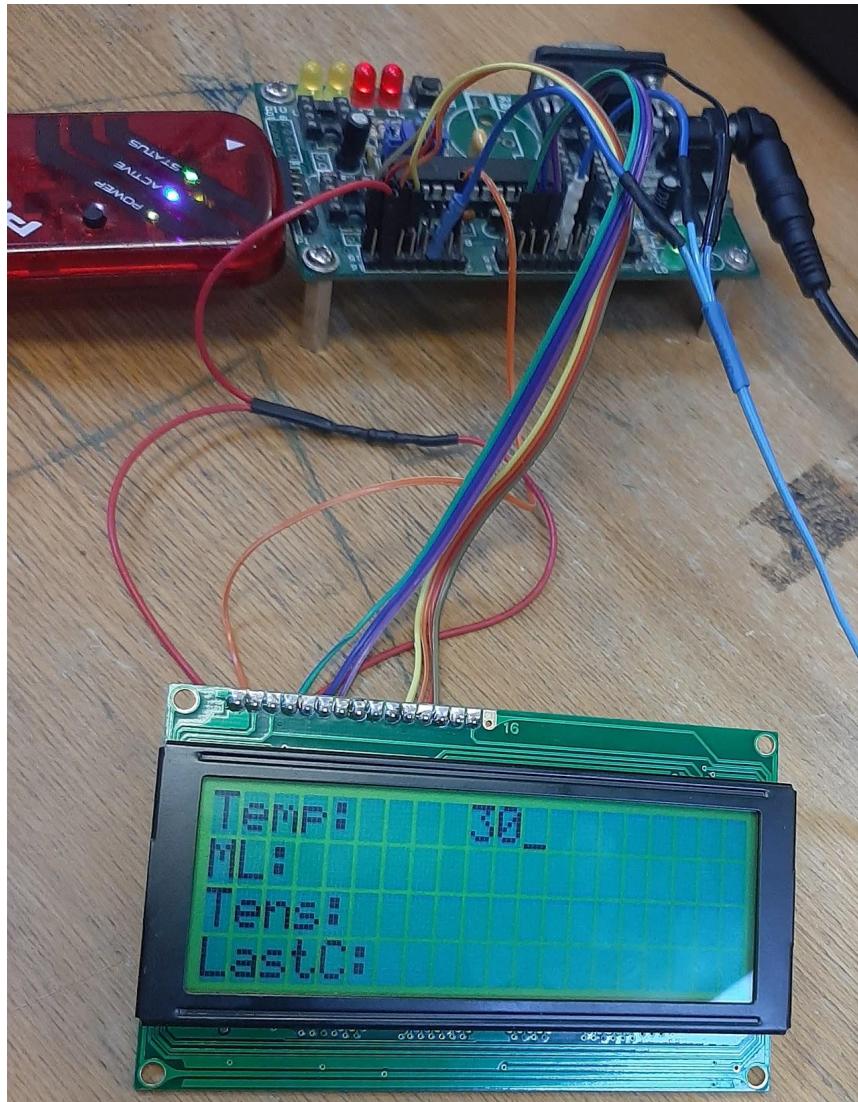


Pentru a testa codul, am afișat pe LCD diferite informații, precum temperatura achiziționată ulterior, în cadrul laboratorului următor.

În săptămâna următoare, a 5-a, am atins subiectul comunicației cu un senzor digital de temperatură DS18S20/DS18B20. Am conectat senzorul de temperatură la plăcuță, iar cu ajutorul unui breakpoint introdus la variabilă t, am putut vizualiza în watch cum aceasta se modifică odată cu schimbarea temperaturii. Ulterior am convertit valoarea numerică a variabilei 't' într-un sir de caractere și am memorat-o într-o nouă variabilă 'sir' pe care am afișat-o pe ecranul LCD.

```
void TaskTemp(void *params) {
    for(;;){
        t = ds1820_read();
        _itoaQ15(t,sir);
        vTaskDelay(4000);
        LCD_Goto(1,10);
        LCD_printf(sir);
    }
}
```





La ultimul laborator am discutat despre comunicația pe interfață serială RS232. Cu ajutorul unui driver descărcat (CP210x) și a software-ului CoolTerm, am transmis mesaje din MPLAB spre software.

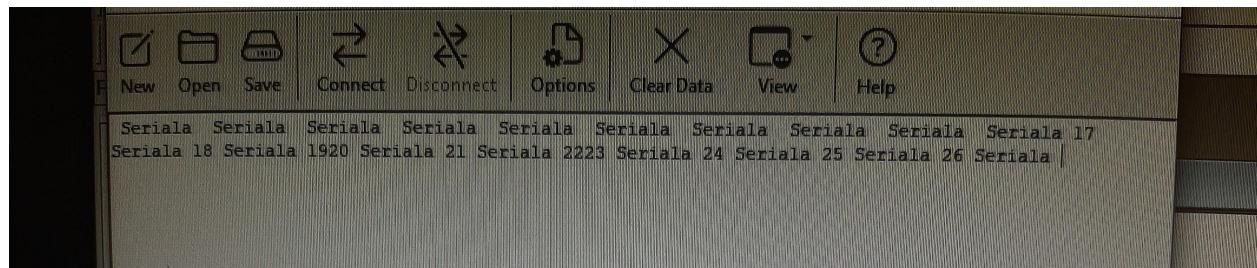
```
void TaskSeriala2(void *params) {
    static int i=0;
    static char sir1[3];
    signed char cByteRxed;

    for (;;)
    {
        i++;
        _itoaQ15(i,sir1);
        vSerialPutString( NULL, sir1, comNO_BLOCK );
```

```

/* Block on the queue that contains received bytes until a byte is
available. */
if( xSerialGetChar( NULL, &cByteRxed, comRX_BLOCK_TIME ) )
{
}
}

```



3. Concluzii și direcții viitoare

În săptămânile ce au urmat, am continuat lucrul la proiect, astfel am reușit să integrăm toate elementele enumerate mai sus în proiectul nostru, fără să ne apară alte erori.

Deși nu am reușit să implementăm complet funcționalitățile dorite, am reușit să integrăm cu succes componente hardware și să obținem o bază solidă pentru dezvoltarea ulterioară.

Mulțumim pentru îndrumare!

