Proiect PATR

1. Descriere funcțională a aplicației

Această aplicație embedded controlează o trapă acționată de un servomotor, utilizând microcontroller-ul dsPIC33FJ128MC802 și sistemul de operare în timp real FreeRTOS. Poziția trapei este reglată în funcție de temperatura ambientală măsurată cu senzorul digital DS18B20 sau de o tensiune analogică aplicată extern. Comutarea între cele două moduri de lucru (automat/manual) este realizată prin comenzi UART.

Interacțiunea cu utilizatorul include:

- Afișaj LCD cu patru linii
- LED de stare (RB11)
- LED pentru mod de lucru (RB1)
- Buton de pornire/oprire (INT0)
- Interfață serială (UART) pentru comenzi și feedback

Starea aplicației este semnalizată prin LED-ul de pe pinul **RB11**:

- **LED aprins continuu** → aplicația este activă și rulează.
- **LED aprins intermitent** → aplicația este oprită.

2. Descriere funcțională a aplicației

- **Mod automat**
  Temperatura este citită la fiecare 2 secunde. Aceasta este mapată liniar în intervalul 20–30°C pentru a comanda poziția servomotorului prin semnal PWM. Formula utilizată este:

P1DC3 = temp_float * 625 - 12500;

Aceasta asigură un duty-cycle între 0% (la 20°C) și 100% (la 30°C), pentru o perioadă PWM de 10 ms (P1TPER = 6250).

- **Mod manual**
  Tensiunea aplicată pe RB3 este convertită folosind ADC-ul intern și afișată pe LCD. Dacă acest mod este activ, valoarea ADC este transformată proporțional într-un semnal PWM care comandă servomotorul.

- **Control UART**
  Sistemul răspunde la comenzi simple de la tastatură:
  - m → afișează modul curent
  - c → comută între modurile automat și manual
  - t → transmite temperatura curentă
  - h → afișează meniul de ajutor

3. Structura aplicației

3.1 Task-uri FreeRTOS implementate

Task	Funcție	Frecvență
`vAppStatusTask`	Controlează LED-ul de pe RB11: continuu = aplicație activă, intermitent = oprită.	200 ms
`vTempTask`	Citește temperatura de la DS18B20, controlează PWM și afișează temperatura.	2000 ms
`vAdcTask`	Citește tensiunea analogică de pe RB3, controlează PWM și afișează pe LCD.	100 ms
`vSerialMenuTask`	Gestionează comenzile UART, comută moduri, afișează status și comenzi.	100 ms

Structuri FreeRTOS utilizate:

- **Semafor semphr_Auto** – utilizat pentru a activa controlul PWM în mod automat. Acest semafor este verificat în interiorul task-ului `vTempTask` cu funcția `xSemaphoreTake(semphr_Auto, 0)`. Dacă semaforul este disponibil (a fost anterior "dat" din alt task, de ex. `vSerialMenuTask`), task-ul aplică valoarea de control derivată din temperatură direct asupra registrului `P1DC3`. În lipsa acestui semnal, controlul automat nu are efect, permițând astfel activarea exclusivă a modului automat la cererea utilizatorului.
- **Semafor semphr_Manual** – utilizat pentru a activa controlul PWM în mod manual. În `vAdcTask`, semaforul este accesat tot prin `xSemaphoreTake(semphr_Manual, 0)`. Doar dacă acesta este preluat cu succes, valoarea citită din ADC este convertită într-un semnal PWM proporțional și transmisă registrului `P1DC3`. Astfel se evită interferența între modurile de lucru.
- **Coada queue_temp** – este o coadă de tip `xQueueHandle` cu o singură poziție, de tip `float`, folosită pentru a transmite temperatura citită în `vTempTask` către `vSerialMenuTask`. Acest mecanism permite interogarea temperaturii curente prin UART la primirea comenzii `t`. `vTempTask` trimite temperatura în coadă folosind `xQueueSend(...)`, iar `vSerialMenuTask` o accesează prin `xQueueReceive(...)` pentru a o converti și transmite către terminalul serial.

4. Cod sursă complet

```
xSemaphoreHandle semphr_AppStatus, semphr_Manual, semphr_Auto; //
semafoare pentru controlul modurilor
xQueueHandle queue_temp; // coada pentru transmiterea temperaturii intre
task-uri

void vAppStatusTask(void *params)
{
    portTickType timer = 0; // timer pentru sincronizare la 200 ms
    for (;;)
    {
        _RB11 = ~(_RB0 | _RB11); // inverseaza starea LED-ului RB11 in
functie de RB0
        vTaskDelayUntil(&timer, 200); // delay fix pentru executie la
fiecare 200 ms
    }
}

void vTempTask(void *params)
{
    portTickType timer = 0;
    float temp_float = 0; // temperatura citita de la senzor
    char temp_char[10]; // buffer pentru afisare temperatura

    for (;;)
    {
        temp_float = ds1820_read(); // citire temperatura DS18B20
        xQueueSend(queue_temp, &temp_float, 0); // trimite temperatura in
coada

        if(xSemaphoreTake(semphr_Auto, 0) == pdTRUE) // daca suntem in mod
automat
        {
            P1DC3 = temp_float * 625 - 12500; // calculeaza valoarea PWM
corespunzatoare temperaturii
        }

        LCD_line(1); // selecteaza linia 1 pe LCD
        LCD_printf("TEMP:          "); // afiseaza prefix

        // conversia partii intregi a temperaturii din float in string
// (int)temp_float extrage partea intreaga (ex: 25.78 devine 25)
// _itoaQ15 transforma valoarea 25 in sirul de caractere "25"
```

```
        _itoaQ15((int)temp_float, temp_char); // ex: pentru 25.78 → "25"
        LCD_printf(temp_char);
        LCD_printf("."); // separator zecimal

        // prima cifra zecimala: (ex: 25.78 -> (25.78 - 25) * 10 = 7.8 →
(int)7.8 = 7)
// scoatem partea zecimala din float si o multiplicam cu 10 pentru a
obtine prima cifra
        _itoaQ15((int)((temp_float - (int)temp_float) * 10), temp_char);
        LCD_printf(temp_char);

        // a doua cifra zecimala: (ex: 25.78 * 100 = 2578; temp_float * 10
= 257.8 → (int) = 257 → 257 * 10 = 2570; 2578 - 2570 = 8)
// extrage ultima cifra din cele doua zecimale multiplicate cu 100
        _itoaQ15((int)((temp_float * 100) - ((int)(temp_float * 10) *
10)), temp_char);
        LCD_printf(temp_char);

        vTaskDelayUntil(&timer, 2000); // delay fix de 2 secunde
    }
}

void vAdcTask(void *params)
{
    portTickType timer = 0;

    int adc_raw = 0;
    float adc_float = 0;
    char adc_char[10];

    for (;;)
    {
        adc_raw = ADC1BUF0;

        if(xSemaphoreTake(semphr_Manual, 0) == pdTRUE)
        {
            P1DC3 = (((long)adc_raw * 6250) / 4095);
        }

        adc_float = (adc_raw * 3.3f) / 4095.0f;


        LCD_line(3);
        LCD_printf("VOUT:           ");
```

```
        _itoaQ15((int)adc_float, adc_char);
        LCD_printf(adc_char);
        LCD_printf(".");

        _itoaQ15((int)((adc_float - (int)adc_float) * 10), adc_char);
        LCD_printf(adc_char);

        _itoaQ15((int)((adc_float * 100) - ((int)(adc_float * 10) * 10)),
adc_char);
        LCD_printf(adc_char);

        vTaskDelayUntil(&timer, 100);
    }
}

volatile int mod_lucru = 1; // 1 = automat, 0 = manual
extern xQueueHandle queue_temp;



void vAdcTask(void *params)
{
    portTickType timer = 0; // timer pentru executie periodica
    int adc_raw = 0; // valoare ADC intre 0 si 4095
    float adc_float = 0; // valoare convertita in volti
    char adc_char[10]; // buffer pentru afisare

    for (;;)
    {
        adc_raw = ADC1BUF0; // citeste valoarea curenta din ADC

        if (xSemaphoreTake(semphr_Manual, 0) == pdTRUE) // daca suntem in
mod manual
        {
            // calculeaza duty-cycle proportional din intervalul 0...6250
            P1DC3 = (((long)adc_raw * 6250) / 4095);
        }

        // converteste valoarea ADC in tensiune reala:
        // 4095 → 3.3V, 0 → 0V, formula: (valoare_ADC * Vref) / 4095
        // unde Vref este tensiunea de referinta (3.3V in cazul de fata)
        adc_float = (adc_raw * 3.3f) / 4095.0f;

        LCD_line(3); // selecteaza linia 3 a LCD-ului
        LCD_printf("VOUT:            ");
```

```
        _itoaQ15((int)adc_float, adc_char); // partea intreaga
        LCD_printf(adc_char);
        LCD_printf(".");

        _itoaQ15((int)((adc_float - (int)adc_float) * 10), adc_char); //
prima cifra zecimala
        LCD_printf(adc_char);

        _itoaQ15((int)((adc_float * 100) - ((int)(adc_float * 10) * 10)),
adc_char); // a doua cifra zecimala
        LCD_printf(adc_char);

        vTaskDelayUntil(&timer, 100); // asteapta 100 ms
    }
}

// Task care primeste comenzi prin UART si afiseaza/gestioneaza starea
aplicatiei
void vSerialMenuTask(void *params) {
    portTickType timer = 0; // timer pentru executie periodica
    unsigned char rxChar; // caracter receptionat
    float temp_float = 0; // temperatura pentru afisare
    unsigned char temp_char[10]; // buffer pentru conversie

    const char *meniu =
        "\r\n--- Meniu Principal ---\r\n"
        "m - Interogare mod de lucru\r\n"
        "c - Comutare mod automat/manual\r\n"
        "t - Interogare temperatura\r\n"
        "h - Afisare ajutor\r\n"
        "-------------------------------------\r\n";

    vSerialPutString(NULL, meniu, comNO_BLOCK); // afiseaza meniul initial
    LCD_line(4);
    LCD_printf("Ultima comanda:     ");

    for (;;) {
        LCD_line(2);
        LCD_printf("MODE:          ");
        if (mod_lucru) {
            LCD_printf("  auto");
            xSemaphoreGive(semphr_Auto); // semafor pentru mod auto
            _RB1 = 0;
        } else {
```

```
            LCD_printf("manual");
            xSemaphoreGive(semphr_Manual); // semafor pentru mod manual
            _RB1 = 1;
        }

        xQueueReceive(queue_temp, &temp_float, 0); // citeste temperatura
din coada

        if (xSerialGetChar(NULL, &rxChar, 0)) {
            char buf_lcd[2] = { (char)rxChar, '\0' };
            LCD_line(4);
            LCD_printf("LASTCOMM:       ");
            LCD_printf(buf_lcd);

            switch (rxChar) {
                case 'm':
                    vSerialPutString(NULL, "Mod curent: ", comNO_BLOCK);
                    vSerialPutString(NULL, mod_lucru ? "auto\r\n" :
"manual\r\n", comNO_BLOCK);
                    break;
                case 'c':
                    mod_lucru = !mod_lucru;
                    vSerialPutString(NULL, "Mod comutat.\r\n",
comNO_BLOCK);
                    break;
                case 't':
                    _itoaQ15((int)temp_float, temp_char);
                    vSerialPutString(NULL, (char *)temp_char,
comNO_BLOCK);
                    vSerialPutString(NULL, ".", comNO_BLOCK);
                    _itoaQ15((int)((temp_float - (int)temp_float) * 10),
temp_char);
                    vSerialPutString(NULL, (char *)temp_char,
comNO_BLOCK);
                    _itoaQ15((int)((temp_float * 100) - ((int)(temp_float
* 10) * 10)), temp_char);
                    vSerialPutString(NULL, (char *)temp_char,
comNO_BLOCK);
                    vSerialPutString(NULL, "\r\n", comNO_BLOCK);
                    break;
                case 'h':
                    vSerialPutString(NULL, meniu, comNO_BLOCK);
                    break;
                default:
```

```
                    vSerialPutString(NULL, "Comanda necunoscuta.\r\n",
comNO_BLOCK);
                    break;
                }
            }
            vTaskDelayUntil(&timer, 100); // asteapta 100 ms
        }
}



void __attribute__((interrupt, no_auto_psv)) _INT0Interrupt(void)
{
    _RB0 = ~_RB0;

    _INT0IF = 0;
}

int main( void )
{
    // Configure any hardware required for this demo.
    prvSetupHardware();

    TRISB = 0x0000;
    _TRISB7 = 1; // RB7 este setat ca intrare
    PORTB = 0xF000;

    _INT0IF = 0; // Resetem flagul coresp. intreruperii INT0
    _INT0IE = 1; // Se permite lucrul cu intreruperea INT0
    _INT0EP = 1; // Se stabileste pe ce front se genereaza INT0

    xTaskCreate(vAppStatusTask, (signed portCHAR *) "vAppStatusTask",
configMINIMAL_STACK_SIZE, NULL, 5, NULL);
    xTaskCreate(vTempTask, (signed portCHAR *) "vTempTask",
configMINIMAL_STACK_SIZE, NULL, 4, NULL);
    xTaskCreate(vAdcTask, (signed portCHAR *) "vAdcTask",
configMINIMAL_STACK_SIZE, NULL, 5, NULL);
    xTaskCreate(vSerialMenuTask, (signed portCHAR *) "vSerialMenuTask",
configMINIMAL_STACK_SIZE * 2, NULL, 6, NULL);

    vSemaphoreCreateBinary(semphr_AppStatus);
    vSemaphoreCreateBinary(semphr_Manual);
    vSemaphoreCreateBinary(semphr_Auto);

    queue_temp = xQueueCreate(1, sizeof(float));
```

```
    // Finally start the scheduler.
    vTaskStartScheduler();

    return 0;
}
/*-----------------------------------------------------------*/

void initPLL(void)
{
// Configure PLL prescaler, PLL postscaler, PLL divisor
    PLLFBD = 41;        // M = 43 FRC
    //PLLFBD = 30;      // M = 32 XT
    CLKDIVbits.PLLPOST=0;    // N1 = 2
    CLKDIVbits.PLLPRE=0; // N2 = 2

// Initiate Clock Switch to Internal FRC with PLL (NOSC = 0b001)
    __builtin_write_OSCCONH(0x01);  // FRC
    //__builtin_write_OSCCONH(0x03);     // XT
    __builtin_write_OSCCONL(0x01);

// Wait for Clock switch to occur
    while (OSCCONbits.COSC != 0b001);    // FRC
    //while (OSCCONbits.COSC != 0b011); // XT

// Wait for PLL to lock
    while(OSCCONbits.LOCK!=1) {};
}

void init_PWM1()
{
//lucram cu RB10 -> PWM1H3
 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
 /*
    Tcy=25ns;
    T=10ms;
    fu=20%;

  Perioada obtinuta trebuie sa fie mai mica decat 65536/2.
  P1TPER=T/Tcy=10ms/25ns=...=400000;
  cu prescaler de 64=>400000/64=  6250  <65536/2=P1TPER;
```

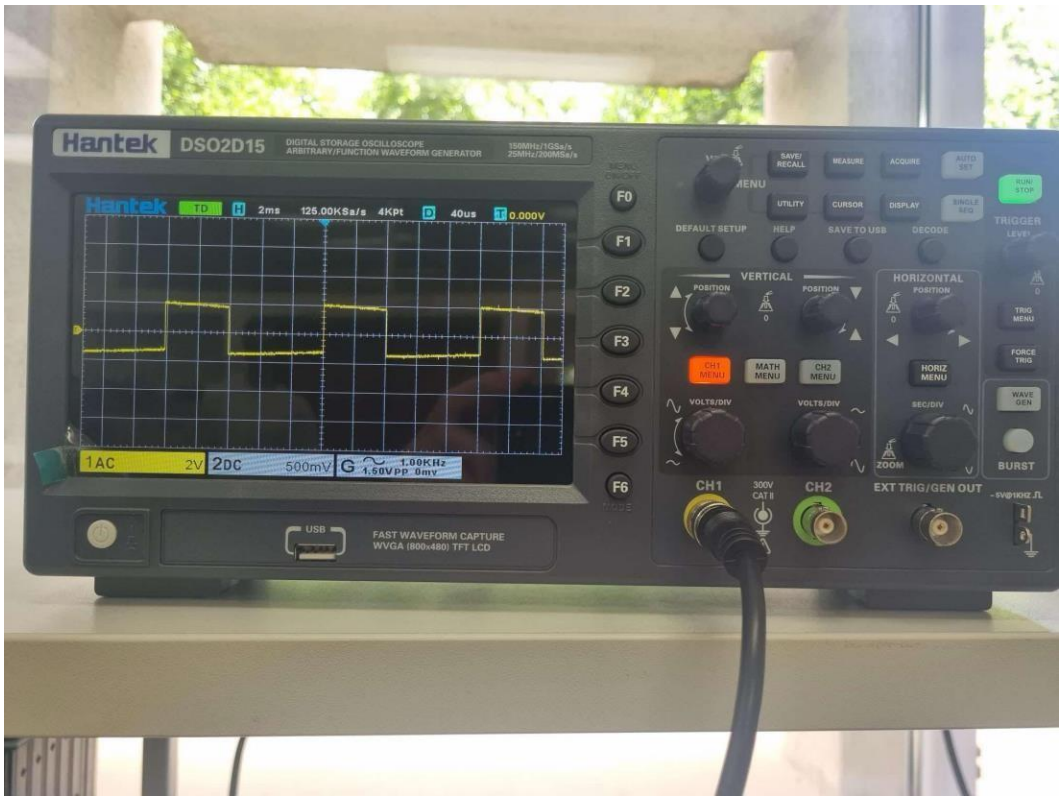
```
  fu=(d/T)*100 => d=(fu*T)/100 =...=2ms;
  P1DC1=(d/Tcy)*2=...=160000
  cu prescaler 64=> 160000/64=  2500  <65536/2=P1DC1*/

 P1DC3 = 2500;
 P1TPER= 6250;

 _PWM1MD = 0; //PWM1 module is enabled

 PWM1CON1bits.PMOD3 = 1; // Canalele PWM3H si PWM3L sunt independente
 PWM1CON1bits.PEN3H = 1; // Pinul PWM1H setat pe iesire PWM

 P1TCONbits.PTCKPS=0b11;

 PWM1CON2bits.UDIS = 0; // 1 da disable la update pentru semnale pwm

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

static void prvSetupHardware( void )
{
    ADPCFG = 0xFFFF;                //make ADC pins all digital - adaugat
    vParTestInitialise();
    initPLL();
    initTmr3();
    init_PWM1();
    initAdc1();
    _CN6PUE = 1;
    ONE_WIRE_DIR = 1;
    output_float();
    ONE_WIRE_PIN = 1;
    onewire_reset();
    LCD_init();
    xSerialPortInitMinimal( mainCOM_TEST_BAUD_RATE, comBUFFER_LEN );

}
```

5. Rezultate experimentale

În cele ce urmează sunt prezentate capturile oscilogramelor obținute pentru două valori ale tensiunii aplicate pe potențiometru: 0V și 3.3V.

Oscilogramă la 0V pe potențiometru

Imaginea de mai jos prezintă semnalul de ieşire PWM generat de microcontroller la aplicarea unei tensiuni de 0V pe potențiometru (mod manual activat).



Oscilogramă la 3V pe potențiometru

Imaginea de mai jos ilustrează semnalul PWM generat la aplicarea unei tensiuni de 3.3V pe potențiometru. Se observă modificarea factorului de umplere.