Sistem de detectare a obstacolelor cu semnalizare și control automat

Realizat de: Antoneac Ramona-Florina

1. Introducere

În cadrul proiectului „**Sistem de detectare a obstacolelor cu semnalizare și control automat**” se propune dezvoltarea unui sistem încorporat destinat vehiculelor, capabil să detecteze, în timpul deplasării, obstacolele aflate în traiectoria de rulare. Sistemul realizează atât semnalizarea prezenței obstacolelor, cât și controlul automat al sistemului de deplasare atunci când acestea se află la o distanță redusă față de vehicul, având ca obiectiv principal prevenirea coliziunilor.

Sistemul este implementat utilizând o placă de dezvoltare echipată cu un microcontroler din familia dsPIC, programat în mediul MPLAB IDE, și integrează senzori pentru detecția obstacolelor, elemente pentru semnalizarea acestora, precum și un motor, pentru evidențierea controlului automat. Funcționalitatea aplicației este gestionată prin intermediul unui sistem de operare în timp real, FreeRTOS.

2. Scopul și obiectivele proiectului

Scopul proiectului este realizarea unui sistem complet de detectare, semnalizare și control al obstacolelor.

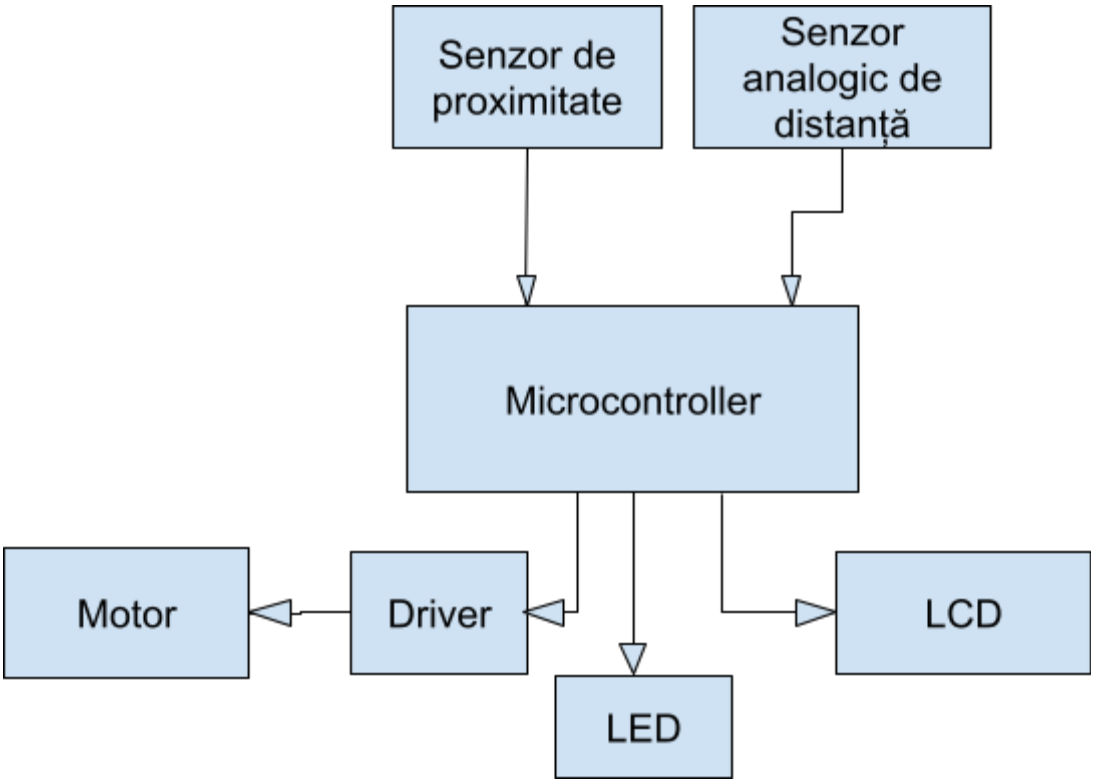
Obiectivele principale ale proiectului sunt:

- ➢ Detectarea prezenței unui obstacol folosind un senzor digital de proximitate
- ➢ Semnalizarea vizuală a obstacolului printr-un LED
- ➢ Afișarea unui mesaj relevant detecției prin intermediul unui LCD
- ➢ Achiziția distanței față de obstacol prin intermediul unui senzor analogic
- ➢ Conversia semnalului analogic utilizând modulul ADC al microcontrolerului
- ➢ Controlul vitezei unui motor electric prin generarea unui semnal PWM

3. Descrierea generală a sistemului

3.1 Diagramă bloc a sistemului

Diagrama bloc a sistemului evidențiază principalele componente hardware și modul în care acestea interacționează între ele. Microcontroller-ul dsPIC reprezintă elementul central, primind informații de la senzorii de intrare și comandând elementele de ieșire.



Elemente incluse în diagramă:

- ➢ Senzor de proximitate → intrare digitală
- ➢ Senzor analogic de distanță → intrare analogică
- ➢ Microcontroller dsPIC → unitate de procesare
- ➢ LCD și LED → elemente de afișare și semnalizare
- ➢ Driver motor + motor → elemente de acționare

3.2 Modul de funcționare

Funcționarea sistemului se desfășoară astfel:

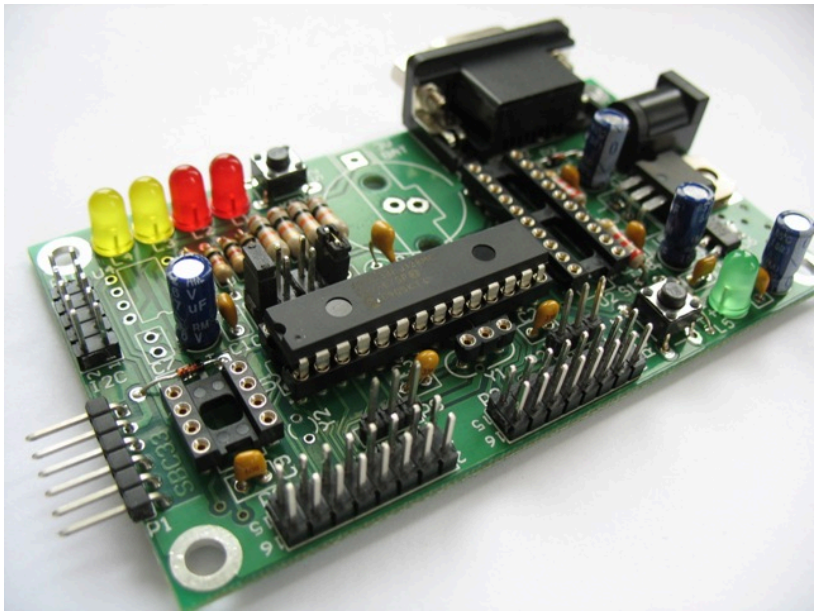
- ➔ Senzorul de proximitate detectează prezența unui obstacol și transmite un semnal digital către microcontroller.
- ➔ Microcontroller-ul afișează pe LCD un mesaj corespunzător existenței sau absenței obstacolului și comandă aprinderea sau stingerea unui LED.
- ➔ Senzorul analogic de distanță furnizează o tensiune proporțională cu distanța față de obstacol.

➔ Modulul ADC convertește tensiunea analogică într-o valoare numerică.
➔ Pe baza valorii obținute, se afișează un mesaj pe LCD, iar microcontroller-ul decide oprirea sau pornirea motorului și ajustează semnalul PWM corespunzător.

4. Componenta hardware

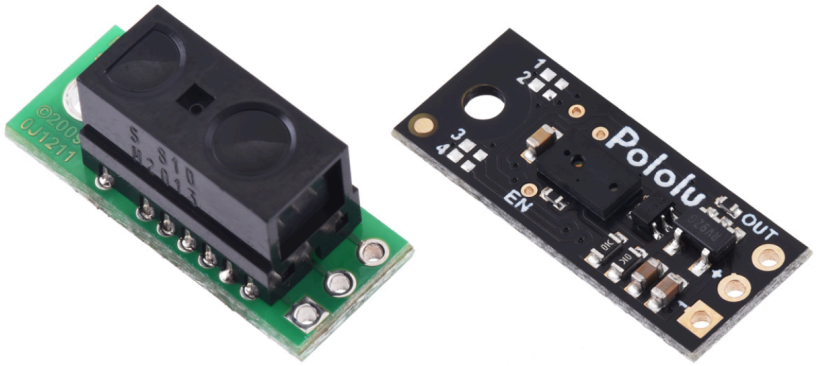
4.1 Microcontroller dsPIC

Microcontroller dsPIC este unitatea centrală de control a sistemului. Acesta este montat pe o placă de dezvoltare și coordonează funcționarea întregului sistem.

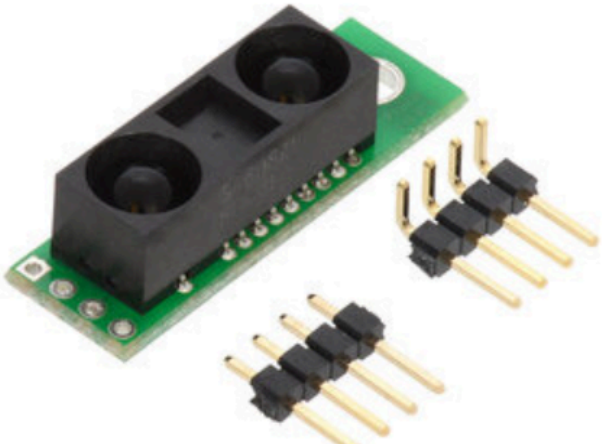


4.2 Componente hardware utilizate

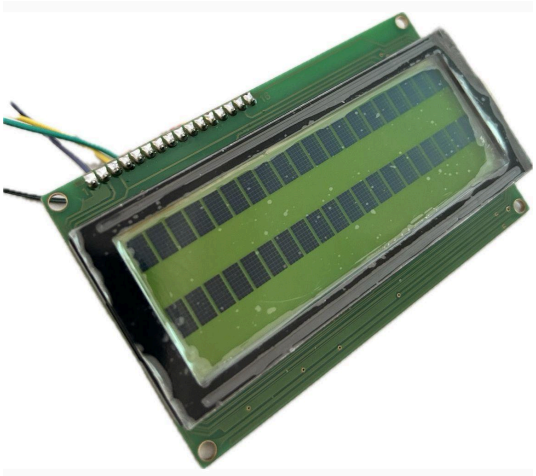
1. **Senzor de proximitate Pololu 1134** – detectare prezența obstacolelor



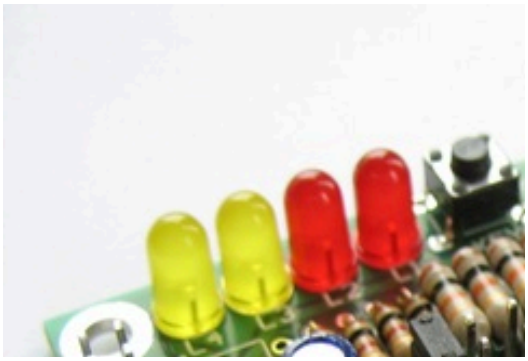
2. **Senzor analogic de distanță Pololu 2476** – furnizare informații privind distanța



3. **LCD** – afișare mesajelor de stare

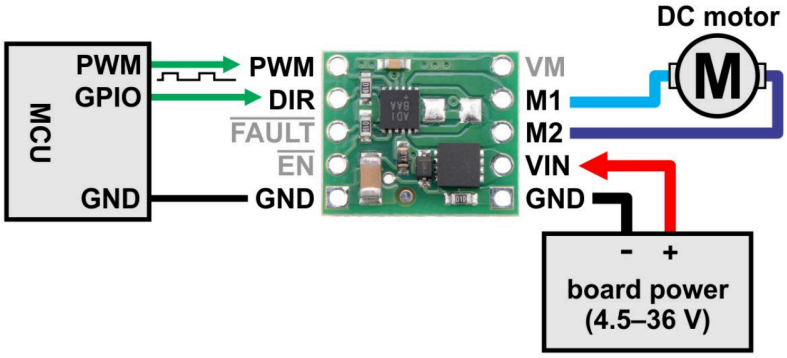


4. **LED** – semnalizare vizuală, conectați pe placă



5. **Motor electric DC**

6. **Driver motor Pololu 2961** – comandă motorul pe baza semnalului PWM



4.3 Alocarea pinilor și conexiunile hardware

Componentă	Pin dsPIC	Tip semnal	Alimentare
Senzor proximitate	RB6	Intrare digitală	3 V
Senzor analogic distanță	RB2	Intrare analogică	3 V
LCD	RB9–RB15	Ieșiri digitale	5 V
Driver motor	RB8	Ieșire digitală (PWM)	5 V
LED	RB7	Ieșire digitală	Conectat pe placa de dezvoltare

Motor	Conectat la driver	Element de execuție	Prin driver

5. Componenta software

5.1 Mediul de dezvoltare

Sistemul a fost dezvoltată folosind MPLAB IDE. Organizarea aplicației se bazează pe operare în timp real, prin intermediul FreeRTOS.

5.2 Structura implementării

Aplicația este împărțită în 2 task-uri:

1. Task de detectare a obstacolului, realizat prin intermediul senzorului digital de proximitate, responsabil de semnalizarea prezenței acestuia prin afișarea unui mesaj pe LCD și activarea unui semnal luminos (LED).

2. Task de achiziție și procesare a distanței față de obstacol, utilizând senzorul analogic de distanță și modulul ADC al microcontrolerului, având rolul de a furniza informații privind poziția relativă a obstacolului pe LCD și de a realiza controlul automat al motorului, prin driver motor, în funcție de valoarea măsurată.

5.3 Achiziția și procesarea datelor digitale - *Anexa A*

Achiziția datelor digitale se realizează prin intermediul senzorului de proximitate, care furnizează un semnal logic către microcontroller, 1 reprezentând absență obstacolului, iar 0 prezența acestuia. În funcție de nivelul logic detectat, sistemul realizează semnalizarea corespunzătoare prin LED și LCD.

5.4 Achiziția datelor analogice (ADC) - *Anexa B*

Modulul ADC realizează o conversie pe 12 biți, permițând cuantizarea semnalului analogic în 4096 niveluri discrete. Conversiile sunt declanșate periodic cu ajutorul Timerului 3, iar rezultatele sunt prelucrate într-o rutină de întrerupere.

Senzorul analogic este conectat la pinul RB2, corespunzător canalului analogic AN4 al microcontrolerului.

Conversia analog-digital constă în transformarea tensiunii analogice furnizate de senzor într-o valoare numerică, proces realizat de modulul ADC al microcontrolerului. Având în vedere că modulul ADC realizează conversia pe 12 biți și utilizează o tensiune de referință de 3 V, domeniul de conversie este împărțit în 4096 niveluri discrete.

Pentru luarea deciziei privind apropierea obstacolului, a fost definit un prag de tensiune de 1,5 V. Valoarea digitală corespunzătoare acestui prag este calculată pe baza relației generale de conversie a ADC-ului, ținând cont de tensiunea de referință și de numărul de biți ai convertorului.

$$Valoare\_ADC\_prag = (V\_prag / V\_ref) \cdot (2\text{^}N - 1)$$

5.4 Controlul motorului prin PWM - *Anexa D*

Controlul motorului electric este realizat cu ajutorul modulului PWM2 al microcontrolerului dsPIC. Modulul utilizează un timer intern care numără crescător de la zero până la valoarea perioadei stabilite în registrul P2TPER, determinând frecvența semnalului PWM generat.

Factorul de umplere al semnalului PWM este controlat prin intermediul registrului P2DC1, care stabilește durata intervalului activ al semnalului într-o perioadă completă. În cadrul implementării, valorile registrelor P2TPER și P2DC1 au fost configurate astfel încât să permită atât oprirea motorului, prin setarea factorului de umplere la zero, cât și funcționarea acestuia la turație maximă, prin egalarea valorii P2DC1 cu perioada PWM.

Semnalul PWM este generat pe pinul RB8, corespunzător ieșirii PWM2H1, prin activarea funcției de ieșire PWM pentru acest pin. Pinul complementar PWM2L1 nu este utilizat, fiind configurat ca pin de control pentru LCD.

Modulul este inițializat cu un factor de umplere nenul, corespunzător unei stări de deplasare inițiale a vehiculului. Această abordare permite evaluarea funcționării sistemului de control automat în condiții de mers, evidențiind intervenția acestuia prin oprirea și repornirea motorului în funcție de distanța față de obstacole.

Activarea modulului PWM se realizează prin setarea bitului PTEN, iar actualizarea registrelor de perioadă și factor de umplere este permisă în timpul funcționării modulului, asigurând flexibilitatea controlului în timp real.

6. Limitări ale sistemului

Principala limitare observată constă în faptul că, în timpul testelor practice, motorul electric nu s-a rotit, deoarece nu a primit suficientă putere pentru a porni, chiar și utilizând driverul de motor. Această limitare este cauzată de constrângerile sursei de alimentare disponibile în timpul testării.

Funcționarea corectă a algoritmului de control a fost validată prin observarea semnalului PWM generat de microcontroller. Cu ajutorul osciloscopului se observă variația factorului de umplere al semnalului PWM în funcție de distanța față de obstacol, demonstrând astfel că partea software și logica de control sunt implementate corect.

7. Concluzii

Proiectul realizat demonstrează implementarea cu succes a unui sistem încorporat pentru detecția și controlul obstacolelor, bazat pe utilizarea unui microcontroller dsPIC. Obiectivele propuse au fost îndeplinite integral, funcționalitatea sistemului fiind validată prin semnalizarea vizuală și afișarea mesajelor informative, precum și prin intervenția automată asupra motorului, în funcție de datele furnizate de senzori.

8. Bibliografie

- Documentație Microchip – dsPIC
- Datasheet Pololu 1134
- Datasheet Pololu 2476
- Datasheet Pololu 2961
- Documentație FreeRTOS

9. Anexe

Anexa A – *Achiziţia şi procesarea datelor digitale*

```
void Task1(void *params) {              // task - senzor de proximitate


      for (;;){

            if (_RB6 == 1) {            // 1 - nu exista obstacol ; 0 -
exista obstacol

                  LCD_Goto(1, 10);      // mutare pointer LCD pe linia 1
coloana 12

                  LCD_printf("NU");     // mesaj obstacol nedetectat

                  _RB7 = 1;             // led stins

            }

            else {

                  LCD_Goto(2, 10);      // mutare pointer LCD pe linia 2
coloana 12

                  LCD_printf("DA");     // mesaj obstacol detectat

                  _RB7 = 0;             // led aprins

            }


            vTaskDelay(300);            // perioada refresh ecran

      }

}
```

Anexa B - *Achiziţia datelor analogice (ADC)*

```
#if defined(__dsPIC33F__)
#include "p33fxxxx.h"
#endif

#include "adcDrv1.h"

#define  SAMP_BUFF_SIZE                 10          //          Dimensiunea
buffer-ului in care se salveaza rezultatele conversiei
```

```
volatile unsigned int adc_peste_prag = 0;
volatile unsigned int adc_sub_prag  = 0;

#define prag_adc   2048                 //(1.5/3.0) в 4095  - prag 1.5 V


//INITIALIZARE ADC PENTRU SCANARE CANAL AN4(RB2)
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.CSS4 = 1;  // Selectam intrarea analogica AN4(RB2)
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.PCFG4  =  0;//  Setam  pinul  AN4(RB2)  pe  intrare
analogica,
```

```
                                        // ADC verifica voltajele pe
acel pin (achizitie 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;
}

// Timer-ul 3 este setat sa starteze conversia AD la fiecare 125
microsecunde (8Khz Rate).
void initTmr3()
{
        TMR3 = 0;
        PR3 = 4999;
        T3CONbits.TON = 1;   // Start Timer 3
}

int  ain4Buff[SAMP_BUFF_SIZE];
int  sampleCounter=0;

// rutina de tratare a intreruperii convertorului AD
void __attribute__((interrupt, no_auto_psv)) _ADC1Interrupt(void)
{

      int val = ADC1BUF0;

      ain4Buff[sampleCounter++]= val;
      if(sampleCounter==SAMP_BUFF_SIZE)
            sampleCounter=0;

      if (val > prag_adc)
        adc_peste_prag++;
    else
        adc_sub_prag++;

    IFS0bits.AD1IF = 0;           // Achita intreruperea convertorului AD
}
```

Anexa C - *Generare semnal PWM*

```
#if defined(__dsPIC33F__)
#include "p33fxxxx.h"
#endif

#include "pwm.h"

void init_PWM2()
{
 P2TCONbits.PTOPS = 0; // Timer base output scale
 P2TCONbits.PTMOD = 0; // Free running

 P2TMRbits.PTDIR = 0; // Numara in sus pana cand timerul = perioada
 P2TMRbits.PTMR = 0; // Baza de timp

 P2DC1 = 0x2710;

 P2TPER = 0x4E20;

 PWM2CON1bits.PMOD1 = 1; // Canalele PWM2H si PWM2L sunt independente

 PWM2CON1bits.PEN1H = 1; // Pinul PWM2H1 setat pe iesire PWM RB8
 PWM2CON1bits.PEN1L = 0; // Pinul PWM2L1 pe I/O general purpose RB9

  PWM2CON2bits.UDIS = 0  ; // Disable Updates from duty cycle and period
buffers
 P2TCONbits.PTEN = 1; /* Enable the PWM Module */
}
```

Anexa D - *Prelucrare date de la senzor analogic și control motor*

```
void Task2(void *params) {              //  task2  -  senzor  analogic  de
distanta


      unsigned int over, under;

      for (;;){


            taskENTER_CRITICAL();             // protejarea datelor

        over  = adc_peste_prag;               // valoare peste prag

        under = adc_sub_prag;                 // valoare sub prag

        adc_peste_prag = 0;                   //      resetare      pentru
contorizare

        adc_sub_prag = 0;

        taskEXIT_CRITICAL();


        if (over > under){                    //   verificare   pozitionare
fata de pragul stabilit

                LCD_Goto(3, 1);

            LCD_printf("aproape");      // valoare peste prag

                P2DC1 = 0;              // motor oprit

           }

        else{

                LCD_Goto(4, 1);

            LCD_printf("departe");      // valoare sub prag

                P2DC1 = P2TPER;               //motor pornit

           }


           vTaskDelay(300);                        //   perioada   refresh
ecran
```

```
	}

}
```

Anexa E - *Configurări program*

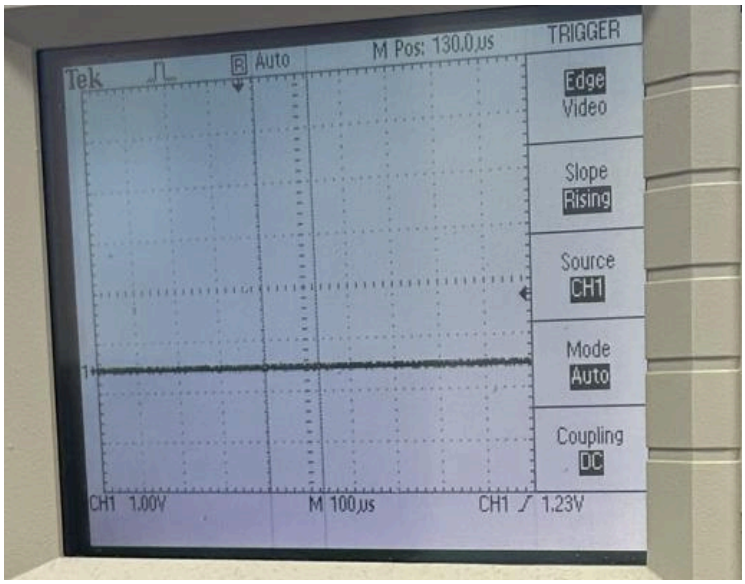
```
static void prvSetupHardware( void )
{
        RCONbits.SWDTEN=0;
        initPLL();
	ADPCFG = 0xFFFF;                  //  make  ADC  pins  all  digital  -
adaugat
	vParTestInitialise();

	PORTB = 0X0000;
	TRISB = 0x0000;                   // toti pinii: OUT
	_TRISB6 = 1;                      //   RB6   -   IN   (senzor   digital
proximitate)
	_TRISB2 = 1;                      //   RB2   -   IN   (senzor   analogic
distanta)
// Peripheral Initialisation
//    initPLL();

	initAdc1();                  // Initialize ADC
	initTmr3();                       // Initialise TIMER 3

	LCD_init();

	LCD_line(1);
	LCD_printf("Obstacol:");
	init_PWM2();                      // initializare PXW2

}
```

Anexa F - *Funcția principală a aplicației*

```
int main( void )

{

      /* Configure any hardware required for this demo. */

      prvSetupHardware();



      xTaskCreate(Task1,       (signed      portCHAR       *)       "Ts1",
configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 1, NULL);

      xTaskCreate(Task2,       (signed      portCHAR       *)       "Ts2",
configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY + 1, NULL);



      /* Finally start the scheduler. */

      vTaskStartScheduler();



      return 0;

}
```

Anexa G - *Observarea semnalului PWM generat*

➢ *Motorul oprit în apropierea obstacolului*



➢ *Motorul pornit la distanță de obstacol*