Implementarea unui Sistem de Control pentru o Periuță Electrică pe Platforma FRDM-KL25Z

Ema Maria Gâtan
Facultatea de Sisteme Informatice și Securitate Cibernetică
Academia Tehnică Militară „Ferdinand I"
București, România
Email: ema.gatan@mta.ro

David Sorin Stancu
Facultatea de Sisteme Informatice și Securitate Cibernetică
Academia Tehnică Militară „Ferdinand I"
București, România
Email: david.stancu@mta.ro

***Rezumat*—Această lucrare prezintă proiectarea și implementarea unui sistem de control pentru o periuță de dinți electrică, dezvoltat pe platforma de dezvoltare NXP FRDM-KL25Z. Sistemul propus integrează funcționalități avansate precum detectarea presiunii de periaj în timp real prin intermediul unui senzor rezistiv de forță (FSR), un cronometru cu ritmare pe cadrane dentare și multiple moduri de periaj implementate prin modularea semnalului PWM. Arhitectura sistemului este bazată pe un automat finit cu stări bine definite, asigurând o experiență de periaj optimizată și personalizată. Proiectul demonstrează integrarea eficientă între hardware și software într-o aplicație pentru sănătatea dentară.**

***Index Terms*—sistem embedded, FRDM-KL25Z, periuță electrică, senzor FSR, PWM, automat finit, ARM Cortex-M0+**

I. Introducere

A. Contextul General și Relevanța Proiectului

Sănătatea orală reprezintă o componentă esențială a bunăstării generale, iar tehnica de periaj corectă joacă un rol de bază în prevenirea afecțiunilor dentare. Conform recomandărilor Asociației Dentare Americane (ADA), periajul eficient presupune o durată de aproximativ două minute, cu presiune moderată și acoperirea uniformă a tuturor zonelor dentare [1]. Cu toate acestea, studiile arată că majoritatea persoanelor nu respectă aceste recomandări, fie din cauza lipsei unui feedback adecvat, fie din cauza obiceiurilor incorecte formate de-a lungul timpului.

Periuțele de dinți electrice moderne au evoluat semnificativ în ultimele decenii, integrând tehnologii precum senzorii de presiune, cronometrele inteligente și conectivitatea Bluetooth pentru monitorizare prin aplicații mobile [2]. Aceste dispozitive oferă utilizatorilor feedback în timp real și date statistice despre obiceiurile lor de periaj, contribuind la îmbunătățirea tehnicii și a sănătății orale pe termen lung.

B. Motivația Alegerii Proiectului

Alegerea acestui proiect este motivată de mai mulți factori. În primul rând, domeniul sistemelor embedded reprezintă o ramură în continuă expansiune a ingineriei software și hardware, cu aplicații directe în viața cotidiană. Dezvoltarea unei periuțe de dinți inteligente oferă oportunitatea de a explora practic concepte fundamentale precum achiziția de date analogice, controlul motoarelor prin PWM, gestionarea întreruperilor și implementarea automatelor finite.

În al doilea rând, platforma FRDM-KL25Z, bazată pe microcontrolerul ARM Cortex-M0+, reprezintă un mediu de învățare ideal datorită echilibrului între complexitate și accesibilitate. Aceasta oferă periferice variate și o documentație extinsă, permițând prototiparea rapidă și eficientă.

În al treilea rând, proiectul adresează o problemă reală de sănătate publică, demonstrând modul în care tehnologia embedded poate contribui la îmbunătățirea calității vieții utilizatorilor prin feedback inteligent și personalizat.

C. Scopul și Obiectivele Proiectului

Scopul principal al acestui proiect este crearea unui prototip funcțional al unei periuțe de dinți electrice, capabilă să aduca beneficii și să ghideze obiceiurile de periaj ale utilizatorului. Obiectivele specifice includ:

1) Implementarea unui sistem de detectare a presiunii de periaj utilizând un senzor rezistiv de forță (FSR), cu feedback vizual în timp real atunci când este aplicată o forță excesivă. Aceasta previne deteriorarea smaltului dentar și a gingiilor cauzată de periajul agresiv.
2) Dezvoltarea unui cronometru cu ritmare pe cadrane, care împarte ciclul de periaj de două minute în patru intervale de câte 30 de secunde, corespunzătoare celor patru cadrane dentare. La finalul fiecărui interval, sistemul emite un semnal haptic pentru a notifica utilizatorul să schimbe zona de periaj.
3) Implementarea a trei moduri de periaj distincte: Curățare Standard, Sensibil și Albire. Aceste moduri sunt realizate prin ajustarea parametrilor semnalului PWM trimis către motorul de vibrații, oferind o experiență de periaj adaptată nevoilor individuale.
4) Proiectarea unei interfețe cu utilizare intuitiva, bazată pe butoane tactile pentru control și indicatori LED pentru feedback vizual, completată de un display OLED pentru afișarea informațiilor detaliate despre sesiunea de periaj.

D. Lucrări Similare și Tehnologii Utilizate

În literatura de specialitate și pe piața comercială există numeroase exemple de periuțe de dinți inteligente. Producători precum Oral-B, Philips Sonicare și Colgate au dezvoltat dispozitive cu funcționalități avansate, incluzând conectivitate

Bluetooth, aplicații mobile pentru monitorizare și inteligență artificială pentru analiza tehnicii de periaj [3].

Din punct de vedere academic, proiecte similare au fost realizate pe diverse platforme de dezvoltare. Sisteme bazate pe Arduino au demonstrat fezabilitatea implementării senzorilor de presiune și a cronometrelor inteligente într-un format accesibil [4]. De asemenea, proiecte pe platforme ARM precum STM32 și NXP Kinetis au explorat capabilitățile avansate ale microcontrolerelor moderne în aplicații de sănătate personală.

Tehnologiile cheie utilizate în acest proiect includ: microcontrolerul NXP Kinetis KL25Z cu nucleu ARM Cortex-M0+ pe 32 de biți, senzorul rezistiv de forță pentru detectarea presiunii, modularea lățimii impulsului (PWM) pentru controlul motorului, conversia analog-digitală (ADC) pentru achiziția semnalului de la senzor și comunicația I2C pentru interfațarea cu display-ul OLED.

E. Structura Lucrării

Lucrarea este organizată după cum urmează: Capitolul II prezintă arhitectura sistemului și principiile de proiectare, detaliind componentele hardware și software ale soluției propuse. Capitolele următoare vor aborda implementarea detaliată, testarea și validarea sistemului, precum și concluziile și direcțiile de dezvoltare ulterioară.

II. Arhitectura Sistemului și Proiectare

A. Prezentare Generală a Arhitecturii

Arhitectura sistemului este concepută modular, cu separarea clară între subsistemele de intrare, procesare și ieșire. În centrul sistemului se află microcontrolerul NXP FRDM-KL25Z, care orchestrează toate operațiunile și asigură comunicarea între componente. Figura 1 ilustrează schema electrică de interconectare hardware a sistemului, evidențiind modul fizic în care senzorii, actuatorul și interfața utilizator sunt conectate la pinii (GPIO, ADC, I2C) plăcii de dezvoltare.

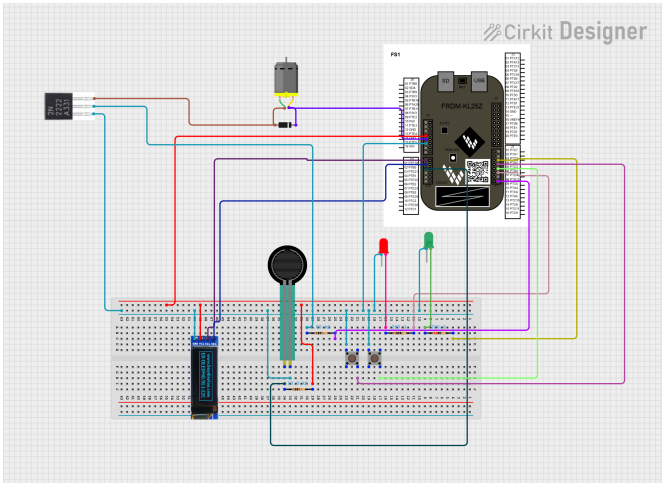

Figura 1. Schema electrică de interconectare hardware a sistemului

Sistemul poate fi descompus în trei subsisteme principale: subsistemul de intrare (senzori și butoane), unitatea centrală de procesare (microcontrolerul) și subsistemul de ieșire (actuatori și indicatori).

B. Platforma de Dezvoltare FRDM-KL25Z

Placa de dezvoltare FRDM-KL25Z [5] este construită în jurul microcontrolerului Kinetis KL25Z, echipat cu un nucleu ARM Cortex-M0+ pe 32 de biți, tactat la o frecvență maximă de 48 MHz. Această putere de procesare este suficientă pentru gestionarea simultană a senzorilor, temporizatoarelor, semnalelor PWM și interfeței cu utilizatorul.

Din punct de vedere al memoriei, dispozitivul oferă 128 KB de memorie Flash pentru stocarea programului și 16 KB de memorie SRAM pentru date și stivă. Aceste resurse sunt adecvate pentru aplicația propusă și permit eventuale extinderi ale funcționalității.

Alimentarea plăcii se realizează prin portul USB, care furnizează 5V. Un regulator de tensiune intern generează nivelul de 3.3V necesar pentru circuitele digitale și periferice. Este important de menționat că pinii GPIO operează la nivel logic de 3.3V, ceea ce impune selectarea atentă a componentelor externe pentru a asigura compatibilitatea electrică.

Perifericele relevante pentru acest proiect includ: modulul ADC pe 16 biți pentru achiziția semnalului de la senzorul FSR, modulele TPM (Timer/PWM Module) pentru generarea semnalelor PWM destinate motorului, interfața I2C pentru comunicarea cu display-ul OLED și porturile GPIO pentru conectarea butoanelor și LED-urilor.

C. Subsistemul de Intrare

1) Senzorul Rezistiv de Forță (FSR): Senzorul rezistiv de forță reprezintă componenta principală pentru detectarea presiunii de periaj. Principiul de funcționare al unui FSR [10] se bazează pe variația rezistenței în funcție de forța aplicată: pe măsură ce forța crește, rezistența senzorului scade. Această caracteristică îl face ideal pentru aplicații de detectare a presiunii în sisteme embedded.

Deoarece microcontrolerul măsoară tensiune și nu rezistență direct, senzorul FSR este integrat într-un circuit divizor de tensiune. Configurația aleasă utilizează un rezistor fix de 10 kΩ conectat între pinul ADC și masă, în serie cu senzorul FSR conectat la alimentarea de 3.3V. Tensiunea la punctul de joncțiune este dată de relația:

$$V_{out} = V_{cc} \cdot \frac{R_{fix}}{R_{FSR} + R_{fix}} \tag{1}$$

Astfel, pe măsură ce forța aplicată crește și R_{FSR} scade, tensiunea V_{out} crește proporțional, permițând microcontrolerului să cuantifice nivelul de presiune prin conversia analog-digitală.

2) Butoane Tactile: Interfața cu utilizatorul include două butoane tactile: unul pentru pornirea și oprirea dispozitivului (Start/Stop, conectat la pinul PTA12) și unul pentru selectarea modului de periaj (Mod Periaj, conectat la pinul PTA4). Butoanele sunt conectate la pini GPIO configurați ca intrări cu rezistențe pull-up interne activate, asigurând un nivel logic stabil în starea de repaus.

Pentru eliminarea efectului de bouncing (oscilații mecanice ale contactelor), implementarea software include un mecanism de debouncing bazat pe întreruperi hardware și o mașină de stări software. Butoanele sunt configurate să genereze întreruperi pe frontul descrescător (falling edge), iar validarea apăsării se face printr-un automat cu trei stări (IDLE, DEBOUNCING, WAIT_RELEASE) cu un timp de filtrare de 80 ms.

D. Subsistemul de Ieșire

1) Motorul de Vibrații DC: Motorul de vibrații DC [9] reprezintă actuatorul principal al sistemului, generând atât acțiunea de curățare mecanică, cât și feedback-ul haptic pentru utilizator. Controlul motorului se realizează prin modularea lățimii impulsului (PWM), care permite ajustarea puterii medii livrate motorului fără pierderi semnificative de energie.

Diferitele moduri de periaj sunt implementate prin variația parametrilor PWM:

- **Modul Standard:** Factor de umplere de 60%, oferind o intensitate medie de vibrație adecvată pentru periajul zilnic obișnuit.
- **Modul Sensibil:** Factor de umplere de 40%, pentru utilizatorii cu gingii sensibile sau pentru zonele delicate.
- **Modul Albire:** Factor de umplere variabil între 30% și 90% .

2) Indicatori LED: Sistemul utilizează LED-uri pentru feedback vizual către utilizator. LED-ul RGB integrat pe placa FRDM-KL25Z este folosit pentru indicarea diferitelor stări ale sistemului:

- **LED Verde:** Indică funcționarea normală în modul de periaj activ.
- **LED Roșu:** Semnalizează avertismentul de presiune excesivă, alertând utilizatorul să reducă forța aplicată.

3) Display OLED: Pentru afișarea informațiilor detaliate despre sesiunea de periaj, sistemul include un display OLED monocrom. Comunicarea cu display-ul se realizează prin interfața I2C, minimizând numărul de pini necesari. Pe ecran sunt afișate: timpul total rămas din sesiunea de periaj, cadranul curent și modul de periaj selectat.

E. Arhitectura Software

1) Modelul de Stări: Software-ul sistemului este structurat ca un automat finit determinist (DFA - Deterministic Finite Automaton), cu stări bine definite și tranziții controlate de evenimente. Această abordare oferă predictibilitate, facilitează depanarea și permite extinderea ușoară a funcționalității.

Stările principale ale sistemului sunt:

STATE_IDLE: Starea de repaus, în care dispozitivul este alimentat dar inactiv, așteptând comanda de pornire de la utilizator. Consumul de energie este minimizat în această stare.

STATE_BRUSHING: Starea principală de funcționare, în care cronometrul este activ, motorul funcționează conform modului selectat și sistemul monitorizează continuu presiunea de periaj.

STATE_PRESSURE_WARNING: O sub-stare a STATE_BRUSHING, activată când senzorul FSR detectează depășirea pragului de presiune. LED-ul de avertizare se aprinde. Revenirea la STATE_BRUSHING se face automat când presiunea scade sub pragul de siguranță.

STATE_QUADRANT_SWITCH: O stare tranzitorie activată la fiecare 30 de secunde, semnalizând utilizatorului să treacă la următorul cadran dentar. Motorul execută un tipar haptic distinctiv (pauză scurtă sau puls dublu).

STATE_SESSION_COMPLETE: Starea finală, activată după expirarea celor două minute de periaj. Sistemul emite un semnal haptic distinctiv.

Figura 2 prezintă diagrama automatului finit care ilustrează tranzițiile între stări și condițiile care declanșează aceste tranziții. Punctele de decizie în formă de romb indică verificările efectuate în fiecare ciclu al buclei principale, cum ar fi verificarea presiunii și a timpului scurs.

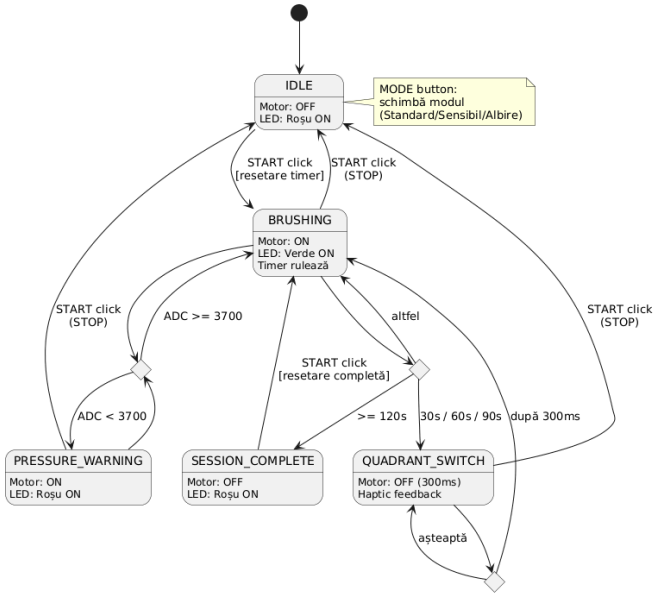

Figura 2. Diagrama automatului finit pentru controlul periuței electrice inteligente.

2) Diagrama de Flux a Datelor: Fluxul de date în sistem urmează un ciclu continuu de achiziție, procesare și răspuns. Semnalul analogic de la senzorul FSR este eșantionat periodic de către modulul ADC, valorile digitizate fiind apoi procesate pentru a determina nivelul de presiune. În paralel, temporizatoarele hardware generează întreruperi la intervale precise pentru actualizarea cronometrului și pentru generarea semnalului PWM.

Figura 3 prezintă diagrama de flux a datelor (DFD) de nivel 1, evidențiind procesele principale și fluxurile de informație între componentele sistemului. Datele circulă de la entitățile externe de intrare (butoane, senzor FSR, SysTick) prin modulele de achiziție și procesare, până la actuatorii de ieșire (motor, LED-uri, display OLED).

Bucla principală de control procesează evenimentele de la butoane (generate prin întreruperi hardware pe PORTA), actualizează afișajul OLED și gestionează tranzițiile între stări în funcție de evenimentele detectate. Arhitectura combină întreru-

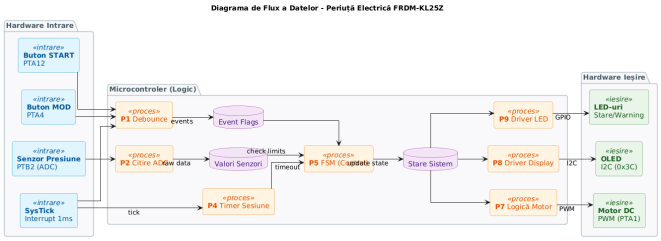

Figura 3. Diagrama de flux a datelor (DFD Nivel 1) pentru sistemul periuței electrice.

perile hardware (SysTick pentru timing, PORTA_IRQHandler pentru butoane) cu polling selectiv pentru senzorul de presiune, asigurând un răspuns prompt la acțiunile utilizatorului și un consum energetic eficient prin utilizarea instrucțiunii WFI între iterații.

3) Considerații de Timp Real: Sistemul impune cerințe de timp real soft pentru anumite operațiuni. Eșantionarea senzorului de presiune trebuie realizată la o rată suficient de mare pentru a detecta variațiile rapide ale forței aplicate, o frecvență de 100 Hz fiind adecvată pentru această aplicație. Generarea semnalului PWM necesită precizie temporală pentru a evita fluctuațiile perceptibile ale vibrațiilor motorului.

Temporizatorul de sesiune și cel pentru cadrane funcționează la rezoluție de secundă, fiind implementate folosind întreruperile periodice ale unui timer hardware configurat corespunzător.

III. Implementare

A. Mediul de Dezvoltare și Limbajul de Programare

Implementarea sistemului a fost realizată în limbajul C, alegere motivată de controlul direct asupra resurselor hardware și de eficiența execuției pe microcontrolere cu resurse limitate. Mediul de dezvoltare utilizat este MCUXpresso IDE, un IDE bazat pe Eclipse oferit de NXP Semiconductors, care include compilatorul GNU ARM Embedded Toolchain și suport nativ pentru depanare prin interfața OpenSDA integrată pe placa FRDM-KL25Z.

Codul sursă este organizat într-un singur fișier `main.c`, abordare tipică pentru proiectele embedded de dimensiuni mici, care simplifică gestionarea dependențelor și compilarea. Structura codului urmează o organizare logică pe secțiuni: definițiile hardware și constantele, variabilele globale, funcțiile driver pentru periferice, implementarea automatului finit și bucla principală.

B. Protocoale de Comunicare

1) Comunicația I2C cu Display-ul OLED: Display-ul OLED SSD1306 [8] comunică cu microcontrolerul prin protocolul I2C (Inter-Integrated Circuit), configurat în modul master cu adresa dispozitivului slave 0x3C. Implementarea include mecanisme de recuperare în caz de blocare a magistralei I2C, prin resetarea automată a perifericului când este detectată o stare de ocupat prelungită.

Tabelul I prezintă parametrii de configurare ai comunicației I2C.

Tabela I
CONFIGURAREA COMUNICAȚIEI I2C

Parametru	Valoare
Frecvență ceas	400 kHz
Adresă OLED	0x3C (7-bit)
Pin SDA	PTB1
Pin SCL	PTB0
Timeout operație	2000 iterații

2) Conversia Analog-Digitală: Modulul ADC este configurat pentru conversie pe 12 biți cu mediere hardware pe 4 eșantioane, reducând zgomotul și îmbunătățind precizia măsurătorilor. La inițializare, se execută o procedură de autocalibrare pentru compensarea erorilor sistematice.

C. Structuri de Date și Variabile Globale

Starea sistemului este menținută prin variabile globale, organizate în categorii funcționale:

- **Timing:** `system_millis` (contor global), `session_seconds` (durata sesiunii)
- **Evenimente:** `evt_start_click`, `evt_mode_click` (flag-uri setate în întreruperi)
- **Stare FSM:** `current_state`, `current_mode`, `current_quadrant`
- **Senzori:** `pressure_adc` (ultima valoare citită de la FSR)
- **Display:** `oled_need_update_*` (flag-uri pentru actualizare selectivă)

D. Implementarea Funcțiilor Cheie

1) Debouncing prin Întreruperi: Mecanismul de debouncing este implementat printr-o arhitectură hibridă: întreruperile hardware detectează fronturi descrescătoare pe pinii butoanelor, iar o mașină de stări software validează apăsările. Această abordare reduce consumul de energie (nu mai este nevoie de polling continuu) și oferă un răspuns mai precis. Fiecare buton are asociată o structură care menține starea curentă (IDLE, DEBOUNCING, WAIT_RELEASE), timestamp-ul începerii debounce-ului și un flag de eveniment.

2) Controlul PWM al Motorului: Semnalul PWM pentru motor este generat de modulul TPM2, canal 0, configurat pentru o frecvență de 1 kHz (MOD = 1000). Funcția `Set_Motor_Speed(uint8_t percent)` calculează valoarea registrului CnV pentru factorul de umplere dorit:

$$CnV = \frac{percent \times MOD}{100} \tag{2}$$

3) Modul Albire - Variație Dinamică: Modul Albire implementează o variație continuă a vitezei motorului între 30% și 90%, cu un pas de 2% la fiecare 50 ms. Direcția de variație se inversează la atingerea limitelor, creând un efect de „respirație" al vibrațiilor.

E. Gestionarea Display-ului OLED

Driver-ul pentru SSD1306 [8] implementează funcții de inițializare, ștergere ecran și afișare text. Font-ul utilizat este unul personalizat de 5x8 pixeli, optimizat pentru afișarea verticală specifică orientării display-ului. Actualizările sunt efectuate selectiv, doar pentru liniile modificate, reducând traficul I2C și consumul de energie.

F. Fragmente de Cod Semnificative

1) Definirea Stărilor FSM: Automatul finit este implementat folosind o enumerare pentru stări și o funcție centralizată de tranziție. Această abordare asigură consistența tranzițiilor și simplifică depanarea.

```
typedef enum {
    STATE_IDLE,               // Repaus - motor oprit
    STATE_BRUSHING,           // Periaj activ
    STATE_PRESSURE_WARNING,   // Avertizare presiune
    STATE_QUADRANT_SWITCH,    // Tranzitie cadran
    STATE_SESSION_COMPLETE    // Sesiune terminata
} SystemState;

volatile SystemState current_state = STATE_IDLE;
```

Listing 1. Enumerarea stărilor automatului finit

2) Gestionarea Butoanelor prin Întreruperi: Butoanele sunt gestionate printr-o combinație de întreruperi hardware și o mașină de stări pentru debouncing. Structura `Button` encapsulează starea fiecărui buton.

```
typedef enum {
    BTN_IDLE,           // Asteptam apasare
    BTN_DEBOUNCING,     // In curs de debounce
    BTN_WAIT_RELEASE    // Asteptam eliberarea
} ButtonState;

typedef struct {
    volatile ButtonState state;
    volatile uint32_t debounce_start;
    volatile uint8_t event_flag;
    uint8_t pin;
} Button;

volatile Button btn_start = {BTN_IDLE, 0, 0, BTN_START_PIN
    };
volatile Button btn_mode = {BTN_IDLE, 0, 0, BTN_MODE_PIN};
```

Listing 2. Structuri pentru gestionarea butoanelor

Handler-ul de întrerupere pentru PORTA detectează apăsarea și inițiază procesul de debouncing.

```
void PORTA_IRQHandler(void) {
    uint32_t isfr = PORTA->ISFR;

    if (isfr & (1 << BTN_START_PIN)) {
        if (btn_start.state == BTN_IDLE) {
            // Dezactiveaza IRQ, incepe debounce
            Button_Disable_IRQ(BTN_START_PIN);
            btn_start.state = BTN_DEBOUNCING;
            btn_start.debounce_start = system_millis;
        }
        PORTA->ISFR = (1 << BTN_START_PIN);  // Clear flag
    }

    if (isfr & (1 << BTN_MODE_PIN)) {
        if (btn_mode.state == BTN_IDLE) {
            Button_Disable_IRQ(BTN_MODE_PIN);
            btn_mode.state = BTN_DEBOUNCING;
            btn_mode.debounce_start = system_millis;
        }
        PORTA->ISFR = (1 << BTN_MODE_PIN);
    }
}
```

Listing 3. Handler-ul de întrerupere pentru butoane

Procesarea debounce-ului se face în bucla principală, verificând dacă starea pinului este stabilă după timpul de filtrare.

```
void Button_Debounce_Process(volatile Button* btn) {
    switch (btn->state) {
        case BTN_IDLE:
            break;  // Asteptam intreruperea

        case BTN_DEBOUNCING:
            if (system_millis - btn->debounce_start >=
                DEBOUNCE_TIME_MS) {
                if (Button_Read_Pin(btn->pin) == 0) {
                    // Buton confirmat apasat
                    btn->event_flag = 1;
                    if (btn->pin == BTN_START_PIN)
                        evt_start_click = 1;
                    else if (btn->pin == BTN_MODE_PIN)
                        evt_mode_click = 1;
                    btn->state = BTN_WAIT_RELEASE;
                } else {
                    // Zgomot, revenim la IDLE
                    btn->state = BTN_IDLE;
                    Button_Enable_IRQ(btn->pin);
                }
            }
            break;

        case BTN_WAIT_RELEASE:
            if (Button_Read_Pin(btn->pin) == 1) {
                btn->state = BTN_IDLE;
                Button_Enable_IRQ(btn->pin);
            }
            break;
    }
}
```

Listing 4. Mașina de stări pentru debouncing

3) Configurarea Întreruperilor pentru Butoane: Inițializarea hardware configurează butoanele cu întreruperi pe front descrescător și activează handler-ul în NVIC.

```
// Configurare butoane: GPIO + Pull-up + Falling edge IRQ
PORTA->PCR[BTN_START_PIN] = PORT_PCR_MUX(1) |
                            PORT_PCR_PE_MASK |
                            PORT_PCR_PS_MASK |
                            PORT_PCR_IRQC(0x0A);  //
                                Falling edge

PORTA->PCR[BTN_MODE_PIN] = PORT_PCR_MUX(1) |
                           PORT_PCR_PE_MASK |
                           PORT_PCR_PS_MASK |
                           PORT_PCR_IRQC(0x0A);

// Curata flag-urile si activeaza NVIC
PORTA->ISFR = (1 << BTN_START_PIN) | (1 << BTN_MODE_PIN);
NVIC_SetPriority(PORTA_IRQn, 2);
NVIC_ClearPendingIRQ(PORTA_IRQn);
NVIC_EnableIRQ(PORTA_IRQn);
```

Listing 5. Configurarea întreruperilor pentru butoane

4) Controlul PWM al Motorului: Funcția `Set_Motor` calculează valoarea registrului CnV pentru factorul de umplere dorit, limitând intrarea la 100%.

```
void Set_Motor(uint8_t speed) {
    if(speed > 100) speed = 100;
    // MOD = 1000, deci CnV = speed * 10
    TPM2->CONTROLS[0].CnV = (speed * 1000) / 100;
}
```

Listing 6. Funcția de control PWM pentru motor

5) Citirea Senzorului de Presiune: Funcția `Read_ADC` inițiază o conversie pe canalul specificat și așteaptă finalizarea prin polling pe flag-ul COCO (Conversion Complete).

```
uint16_t Read_ADC(uint8_t channel) {
    ADC0->SC1[0] = ADC_SC1_ADCH(channel);
    while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK));
    return ADC0->R[0];
}

void Update_Pressure_Warning(void) {
    adc_pressure = Read_ADC(FSR_ADC_CHANNEL);
    uint8_t new_warning = (adc_pressure <
         PRESSURE_THRESHOLD);

    if (new_warning != pressure_warning) {
        pressure_warning = new_warning;
        oled_need_update_pressure = 1;
    }

    // Control LED-uri in functie de presiune
    if (pressure_warning) {
        PTD->PSOR = (1 << LED_RED_PIN);    // RED ON
        PTA->PCOR = (1 << LED_GREEN_PIN); // GREEN OFF
    } else {
        PTD->PCOR = (1 << LED_RED_PIN);    // RED OFF
    }
}
```

Listing 7. Citirea ADC pentru senzorul FSR

6) Algoritmul Sweep pentru Modul Albire: Modul Albire implementează o variație continuă a vitezei între limitele definite, cu inversarea direcției la atingerea extremelor.

```
#define ALBIRE_MIN_SPEED 30
#define ALBIRE_MAX_SPEED 90
#define ALBIRE_SWEEP_STEP 2
#define ALBIRE_SWEEP_INTERVAL 50  // ms

void Update_Albire_Sweep(void) {
    if (!FSM_Is_Motor_Active() || mod_curent != 2)
        return;

    if (system_millis - last_albire_update <
         ALBIRE_SWEEP_INTERVAL)
        return;
    last_albire_update = system_millis;

    albire_speed += albire_direction * ALBIRE_SWEEP_STEP;

    // Inversare directie la limite
    if (albire_speed >= ALBIRE_MAX_SPEED) {
        albire_speed = ALBIRE_MAX_SPEED;
        albire_direction = -1;
    } else if (albire_speed <= ALBIRE_MIN_SPEED) {
        albire_speed = ALBIRE_MIN_SPEED;
        albire_direction = 1;
    }

    Set_Motor(albire_speed);
}
```

Listing 8. Implementarea variației de viteză pentru modul Albire

7) Funcția de Tranziție FSM: Funcția `FSM_Enter_State` gestionează toate tranzițiile între stări, executând acțiunile specifice la intrarea în fiecare stare.

```
void FSM_Enter_State(SystemState new_state) {
    previous_state = current_state;
    current_state = new_state;

    switch(new_state) {
        case STATE_IDLE:
            Set_Motor(0);
            Display_Time_Status("PAUZA");
            break;

        case STATE_BRUSHING:
            if (previous_state == STATE_IDLE) {
                timer_periaj = 0;
                cadran = 1;
                timer_start_millis = system_millis;
            }
            Set_Motor(Get_Current_Speed());
            break;

        case STATE_QUADRANT_SWITCH:
            haptic_active = 1;
            haptic_end = system_millis + 300;
            Set_Motor(0);  // Pauza pentru feedback
            cadran++;
            Display_Cadran();
            break;

        case STATE_SESSION_COMPLETE:
            Set_Motor(0);
            Display_Time_Status("GATA!");
            break;
    }
}
```

Listing 9. Funcția centralizată de tranziție între stări

8) Bucla Principală: Bucla principală implementează un pattern de polling cu economie de energie, folosind instrucțiunea WFI între iterații.

```
int main(void) {
    Init_Hardware();  // Configureaza si PORTA IRQ
    Init_OLED();
    Init_ADC();

    while(1) {
        // Procesare debounce butoane (state machine)
        Button_Debounce_Process(&btn_start);
        Button_Debounce_Process(&btn_mode);

        // Verificare presiune periodic
        if (system_millis - last_pressure_check >= 200) {
            last_pressure_check = system_millis;
            Update_Pressure_Warning();
        }

        // FSM Dispatch
        switch(current_state) {
            case STATE_IDLE:
                FSM_State_Idle();
                break;
            case STATE_BRUSHING:
                FSM_State_Brushing();
                break;
            // ... alte stari
        }

        __WFI();  // Sleep pana la intrerupere
    }
}
```

Listing 10. Structura buclei principale

G. Analiza Liniilor Critice din Codul Sursă

Această secțiune detaliază trei linii de cod esențiale din `main.c`, fără de care sistemul nu ar funcționa corect.

1) Dezactivarea Watchdog Timer-ului: **Linia 759:** `SIM->COPC = 0;`

Această instrucțiune dezactivează *Computer Operating Properly Controller* (watchdog timer). Pe KL25Z, watchdog-ul este activat implicit după reset. Dacă nu este dezactivat sau "hrănit" periodic, acesta generează un reset hardware după expirarea timeout-ului (1024 ms). Prin scrierea valorii 0, se dezactivează complet watchdog-ul.

Fără această linie, microcontrolerul s-ar reseta continuu, făcând imposibilă execuția normală a programului.

2) Configurarea Bazei de Timp SysTick: **Linia 270:** `SysTick_Config(SystemCoreClock / 1000);`

Funcția configurează timer-ul SysTick să genereze întreruperi la fiecare **1 ms**. Cu `SystemCoreClock = 48 MHz` și divizorul 1000, rezultă 48000 cicluri între întreruperi.

Această configurare stabilește "pulsul" sistemului. Variabila `system_millis`, incrementată în `SysTick_Handler()`, servește ca bază de timp pentru: debouncing butoane (80 ms), polling presiune (200 ms), cronometru sesiune (1000 ms), sweep Albire (50 ms) și feedback haptic (300 ms).

Fără această linie, toate funcțiile dependente de timp ar fi compromise.

3) Instrucțiunea Wait For Interrupt: **Linia 827:** `__WFI();`

Instrucțiunea `__WFI()` pune procesorul în modul sleep până la următoarea întrerupere. Beneficiile includ: reducerea consumului de energie (crucial pentru alimentare pe baterie), reducerea emisiilor EMI și sincronizarea cu întreruperile.

4) Interacțiunea dintre Liniile Critice: Cele trei linii formează un ansamblu coerent: (1) `SIM->COPC = 0` asigură rularea fără resetări; (2) `SysTick_Config(...)` creează baza de timp; (3) `__WFI()` optimizează consumul între întreruperi. Împreună, definesc modelul event-driven cu economie de energie al sistemului.

IV. Testare și Validare

Această secțiune prezintă comportamentele așteptate ale sistemului, derivate din cerințele funcționale și non-funcționale definite în etapa de proiectare.

A. Cerințe Funcționale

Tabelul II sintetizează cerințele funcționale ale sistemului și comportamentul așteptat pentru fiecare.

Tabela II
Cerințe funcționale și comportament așteptat

Cerință	Comportament Așteptat
Pornire sesiune	La apăsarea butonului START, motorul pornește și cronometrul începe numărătoarea de la 120s
Oprire sesiune	La apăsarea butonului START în timpul periajului, sesiunea se oprește și sistemul revine în IDLE
Schimbare mod	Butonul MOD ciclează între Standard, Sensibil și Albire; modul curent este afișat pe OLED
Timer cadrane	La 30s, 60s și 90s, motorul execută feedback haptic (pauză 300ms) și cadranul afișat se incrementează
Finalizare sesiune	La 120s, motorul se oprește, LED-ul verde se stinge și OLED afișează mesaj de finalizare
Detectare presiune	Când ADC < 3700, LED-ul roșu se aprinde și OLED afișează avertisment
Afișare OLED	Display-ul arată în timp real: modul, timpul rămas și cadranul curent

B. Cerințe Non-Funcționale

- **Timp de răspuns:** Sistemul trebuie să reacționeze la apăsarea butoanelor în maximum 100 ms, asigurând o experiență de utilizare fluidă.
- **Precizie temporală:** Cronometrul sesiunii trebuie să fie precis la nivel de secundă, fără acumulare de erori pe durata celor 2 minute.
- **Stabilitate:** Sistemul trebuie să funcționeze continuu fără blocări sau comportament imprevizibil pe durata unei sesiuni complete de periaj.
- **Feedback vizual:** LED-urile și display-ul OLED trebuie să reflecte corect starea curentă a sistemului în orice moment.
- **Consum energetic:** În starea IDLE, sistemul trebuie să minimizeze consumul prin utilizarea instrucțiunii WFI (Wait For Interrupt).

C. Scenarii de Utilizare

Scenariu 1 - Sesiune completă de periaj: Utilizatorul pornește dispozitivul, selectează modul dorit, apoi apasă START. Periajul decurge normal timp de 2 minute, cu notificări la schimbarea cadranelor. La final, sistemul semnalizează finalizarea.

Scenariu 2 - Presiune excesivă: În timpul periajului, utilizatorul aplică presiune prea mare. LED-ul roșu se aprinde instant și pe OLED apare avertismentul. Când presiunea scade, indicatorii revin la normal.

Scenariu 3 - Oprire prematură: Utilizatorul decide să oprească sesiunea înainte de finalizare. La apăsarea START, motorul se oprește imediat și sistemul revine în IDLE.

D. Prototipul Final

Figura 4 prezintă starea finală a prototipului, cu toate componentele integrate și funcționale: placa FRDM-KL25Z, display-ul OLED, senzorul de presiune FSR, motorul de vibrații și butoanele de control.

V. Concluzii și Perspective

A. Rezultate Obținute

Proiectul a demonstrat cu succes fezabilitatea implementării unui sistem de control pentru periuță electrică pe platforma FRDM-KL25Z. Toate obiectivele propuse au fost atinse: detectarea presiunii funcționează corect și oferă feedback în timp real, cronometrul cu cadrane ghidează utilizatorul prin sesiunea de periaj, iar cele trei moduri de periaj oferă flexibilitate în funcție de preferințele individuale.

Arhitectura bazată pe automat finit s-a dovedit robustă și ușor de extins, facilitând adăugarea de noi funcționalități fără modificări structurale majore. Separarea clară între driverele hardware și logica de aplicație permite portarea codului pe alte platforme cu efort minim.

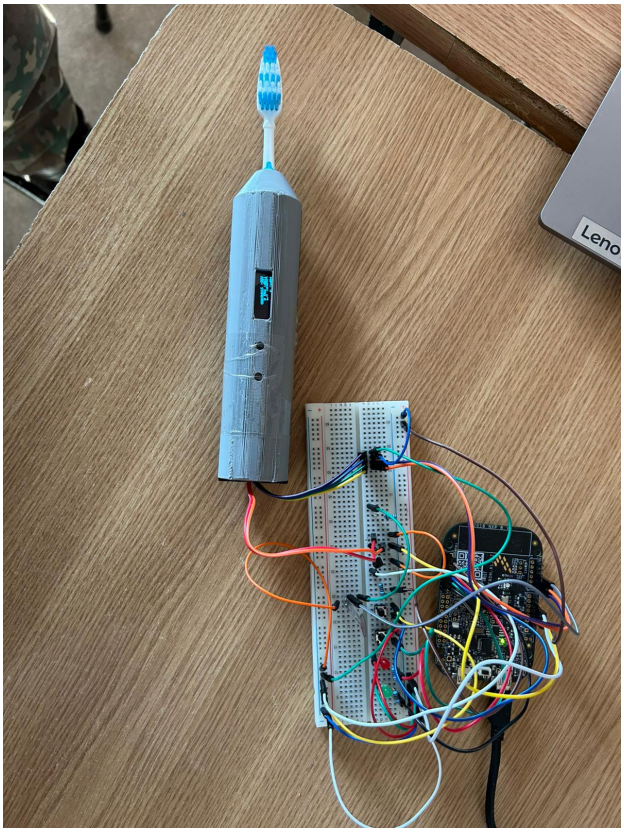

Figura 4. Prototipul final al sistemului de control pentru periuța electrică.

B. Dificultăți Întâmpinate

Principalele provocări tehnice au inclus:

- **Calibrarea senzorului FSR:** Determinarea pragului optim de presiune a necesitat testare empirică, valorile variind în funcție de montajul fizic și de caracteristicile individuale ale senzorului.
- **Comunicația I2C:** Display-ul OLED a prezentat ocazional blocări ale magistralei I2C, necesitând implementarea unui mecanism de timeout și reset automat.
- **Debouncing:** Butoanele mecanice au generat inițial evenimente multiple, problema fiind rezolvată prin implementarea filtrării software cu prag de 50 ms.

C. Analiză Critică

Arhitectura aleasă, cu un singur fișier sursă și variabile globale, este adecvată pentru complexitatea actuală a proiectului, dar ar deveni greu de gestionat pentru extinderi semnificative. O structurare pe module separate (drivers, FSM, UI) ar îmbunătăți mentenabilitatea.

Platforma FRDM-KL25Z s-a dovedit potrivită pentru prototipare, oferind periferice suficiente și documentație bogată. Totuși, pentru un produs comercial, ar fi necesară migrarea către un microcontroler cu consum mai redus și formă mai compactă.

D. Direcții de Dezvoltare Ulterioară

Proiectul poate fi extins în mai multe direcții:

- **Conectivitate Bluetooth:** Adăugarea unui modul BLE pentru sincronizarea datelor cu o aplicație mobilă, permițând monitorizarea obiceiurilor de periaj pe termen lung.
- **Stocare date:** Implementarea unui jurnal al sesiunilor în memoria Flash, cu statistici despre durata medie, presiunea aplicată și modurile utilizate.
- **Moduri suplimentare:** Adăugarea de programe de periaj personalizate, cu secvențe PWM complexe pentru diferite tehnici de curățare.
- **Optimizare consum:** Implementarea modurilor de low-power ale microcontrolerului pentru extinderea autonomiei în cazul alimentării de la baterie.
- **Feedback audio:** Integrarea unui buzzer pentru notificări sonore, complementând feedback-ul haptic și vizual existent.

BIBLIOGRAFIE

[1] American Dental Association, "Brushing Your Teeth," *ADA Oral Health Topics*, 2023. [Online]. Disponibil: https://www.ada.org/resources/research/science-and-research-institute/oral-health-topics/brushing-your-teeth

[2] M. Yaacob et al., "Powered versus manual toothbrushing for oral health," *Cochrane Database of Systematic Reviews*, no. 6, 2014.

[3] S. Lee și J. Kim, "Smart Toothbrush Systems: A Review of Technologies and Applications," *IEEE Access*, vol. 10, pp. 45678–45692, 2022.

[4] R. Patel, "DIY Smart Toothbrush with Arduino," *Instructables*, 2020. [Online]. Disponibil: https://www.instructables.com

[5] NXP Semiconductors, "FRDM-KL25Z User's Guide," Rev. 4, 2023.

[6] ARM Limited, "Cortex-M0+ Technical Reference Manual," ARM DDI 0484C, 2012.

[7] Interlink Electronics, "FSR Integration Guide," Application Note, 2019.

[8] Solomon Systech, "SSD1306: 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller," Datasheet, Rev 1.5, Apr. 2008. [Online]. Disponibil: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[9] Components101, "Toy DC Motor (130 Size) - Pinout, Specifications and Datasheet," [Online]. Disponibil: https://components101.com/sites/default/files/component_datasheet/Toy%20DC%20motor%20Datasheet.pdf

[10] Generic Manufacturer, "RFP602 Thin Film Pressure Sensor Specifications," Datasheet, [Online]. Disponibil: https://img.ozdisan.com/ETicaret_Dosya/951867_157120.pdf