

SMARTKNOB: SISTEM HAPTIC DE CONTROL INTELIGENT

Candidat: Vlad PLĂVĂT

Coordonator științific: dr.ing. Valentin STÂNGACIU

Sesiunea: Iunie 2023-2024

CUPRINS

1 Introducere	7
1.1 Domenii abordate și obiective generale	7
1.2 Contextul de realizare	7
1.3 Specificații generale	8
1.4 Comparatie cu state of the art	9
1.5 Principalele rezultate obținute	11
1.6 Structura lucrării	11
2 Teorie generală	13
2.1 Raspberry Pi Pico	13
2.2 Microcontrolerul RP2040	15
2.2.1 ARM Cortex M0+	15
2.2.2 Memoriile sistemului	16
2.2.3 Perifericul USB	16
2.2.4 Perifericul SPI	16
2.2.5 Perifericul PWM	16
2.2.6 Perifericul PIO	17
2.3 Pico C SDK	18
2.4 Biblioteca TinyUSB	18
2.5 Interfața USB	18
2.6 Motorul BLDC	21
2.7 Modulația duratei impulsurilor (PWM)	23
2.8 Driverul de motor MX1616	24
2.9 Interfața SPI	25
2.10 Senzorul magnetic MT6701	26
2.11 Ecranul LCD GC9A01	27
2.12 Tensometre	28
2.13 Amplificatorul HX711	29
2.14 LED-ul WS2812	30
3 Arhitectura generală a sistemului	31
3.1 Arhitectura software a sistemului	31
3.2 Arhitectura hardware a sistemului	32
3.3 Arhitectura combinată a sistemului	33
4 Implementarea părții mecanice	34
4.1 Ansamblul motor	34
4.1.1 Construcția motorului	34
4.1.2 Selectia numărului de poli și dinti	37

4.1.3 Determinarea caracteristicilor înfășurărilor	38
4.2 Cadrul cu punctile tensometrice	41
4.3 Carcasa butonului	43
5 Implementarea hardware	44
5.1 Conectorul USB	44
5.2 Senzorul magnetic	45
5.3 Motorul	47
5.4 Ecranul LCD	51
5.5 Inelul cu LED-uri	52
5.6 Amplificatoarele și punctile tensometrice	54
6 Implementarea software	55
6.1 Biblioteca USB - tinyUSB	55
6.2 Biblioteca MT6701 - senzorul magnetic	58
6.3 Biblioteca Motor	59
6.3.1 Modul cu frecare	59
6.3.2 Modul cu viteză constantă	60
6.3.3 Aplicarea de vibrație	60
6.3.4 Setarea detentelor	60
6.4 Biblioteca CG9A01 - ecranul LCD	62
6.5 Biblioteca WS2812 - inelul cu LED-uri	67
6.6 Biblioteca HX711 - amplificatoarele	67
6.7 Meniul de setări	68
6.8 Programul principal	70
6.9 Crearea unei aplicații pentru interfațarea cu SmartKnob	71
7 Concluzii	73
7.1 Concluzii generale	73
7.2 Perspective de dezvoltare	73
7.3 Sinteza contribuțiilor	73
8 Anexa 1 - rapoarte HID	74
9 Bibliografie	77

LISTĂ DE FIGURI

1.1	SmartKnob	8
1.2	Tuya Zigbee (sursa: [2])	9
1.3	BHT-7000 Smart Knob (sursa: [3])	9
1.4	SmartKnob realizat de scottbez1 (sursa: scottbez1_smatrknob)	10
1.5	SmartKnob realizat de mine	10
2.1	Pinout-ul plăcii Raspberry Pi Pico (sursa: documentația Raspberry Pi Pico [4, p. 5])	13
2.2	Spatele plăcii Raspberry Pi Pico (sursa: documentația Raspberry Pi Pico [4, p. 4])	14
2.3	Bus fabric-ul microcontrolerului RP2040 (sursa: documentația RP2040 [5, p. 16])	15
2.4	Arhitectura perifericului PIO (sursa: documentația RP2040 [5, p. 311])	17
2.5	Tipuri de descriptori USB (sursa: USB in a nutshell [8])	19
2.6	Structura unui transfer USB (sursa: USB complete [10, p. 65])	20
2.7	Sectiunea unui motor simplu BLDC cu 3 dinti și 2 poli (sursa: Bavaria-direct [11])	21
2.8	Exemple de motoare cu $q < 0.25$ (sursa: Bavaria-direct [11])	22
2.9	Exemple de motoare cu $q > 0.5$ (sursa: Bavaria-direct [11])	22
2.10	Semnal PWM (sursa: OnionDOCS)	23
2.11	LED-uri controlate PWM cu duty cycle de 0.04, 0.2, 0.47 și 1 (alimentat direct) (sursa: PJRC)	23
2.12	Modul cu circuitul MX1616 (sursa: Art of Circuits)	24
2.13	Curentul prin motor în diferitele moduri (sursa: EEWeb)	24
2.14	Conexiune simplă a interfeței SPI (sursa: [14, p. 1])	25
2.15	Exemplu de transfer de date prin interfața SPI (sursa: [14, p. 1])	25
2.16	Ilustrare a modurilor interfeței SPI (diagramă generată cu https://wavedrom.com)	26
2.17	Senzorul magnetic MT6701 (sursa: [15, p. 1])	26
2.18	Ansamblul display GC9A01 (sursa: Makerfabs)	27
2.19	Funcționarea tensometrelor (sursa: GaugeHow)	28
2.20	Punte tensometrică (sursa: SparkFun)	28
2.21	Aplicație tipică a circuitului HX711 (sursa: [17, p. 1])	29
2.22	Un singur LED WS2812 (sursa: Ardushop)	30
2.23	Conecțarea a 3 LED-uri WS2812 în serie (sursa: [18, p. 4])	30
2.24	Codurile ce pot fi transmise către WS2812 (sursa: [18, p. 4])	30
2.25	Comunicarea cu mai multe LED-uri WS2812 (sursa: [18, p. 5])	30
3.1	Arhitectura software	a	sistemului
	(diagramă generată cu draw.io)	31
3.2	Arhitectura hardware	a	sistemului
	(diagramă generată cu draw.io)	32

3.3 Arhitectura combinată (diagramă generată cu draw.io)	a sistemului	33
4.1 Statorul	34	
4.2 Rotorul	34	
4.3 Statorul cu elementele de fixare (vedere de jos)	34	
4.4 Rotorul a fost introdus în stator	35	
4.5 Un magnet a fost atașat rotorului	35	
4.6 Axul pentru display și suportul acestuia	35	
4.7 Axul display-ului a fost introdus în motor	35	
4.8 Ansamblul văzut de sus fără display	35	
4.9 Ansamblul cu display	35	
4.10 Senzorul a fost lipit în partea de jos	35	
4.11 Secțiune transversală a motorului	36	
4.12 Orientarea și amplasarea înfășurărilor (sursa: [11])	37	
4.13 Curentul este maxim în înfășurări	38	
4.14 Cele 3 posibilități de realizare a înfășurărilor	39	
4.15 Determinarea curentilor în cele 3 cazuri	40	
4.16 Schema finală a înfășurărilor și o simplificare echivalentă (ce ține cont de inductanță) a motorului pentru calcule viitoare	40	
4.17 Îndoirea brațelor în unul din cazurile de înclinare	41	
4.18 Forma cadrului metalic	41	
4.19 Deformarea cadrului văzută de sus în diferite cazuri	41	
4.20 Măsurarea unui singur braț față de o referință	42	
4.21 Măsurarea diferenței între două brațe	42	
4.22 Partea superioară a carcasei	43	
4.23 Partea inferioară a carcasei și componentele electronice	43	
4.24 SmartKnob complet asamblat	43	
5.1 Conectorul USB	44	
5.2 Schema de conectare a conectorului USB	44	
5.3 Comunicarea SSI cu senzorul MT6701 (sursa: [15, p. 24])	45	
5.4 Conectarea senzorului MT6701	46	
5.5 Formele de undă măsurate pentru comunicarea cu MT6701	47	
5.6 Tensiunile la cele 3 borne ale motorului, dacă faza variază constant în timp (sursa: Vertiv)	47	
5.7 Comparație între cazul sincronizat (stânga) și cel nesincronizat (dreapta)	49	
5.8 Conectarea driverelor de motor și a motorului	50	
5.9 Schema modulului MX1616 cu dioda Zener adăugată (sursa: EasyEDA)	50	
5.10 Semnalele generate de perifericul SPI	51	
5.11 Inelul cu LED-uri	52	
5.12 Forma de undă măsurată, generată de Raspberry Pi	53	
5.13 Conectarea electrică a inelului cu LED-uri	53	

5.14 Conectarea electrică a amplificatoarelor HX711	54
6.1 Funcționarea bibliotecii MT6701	58
6.2 Configurație cu 3 detente echidistante	61
6.3 Randarea și transmisia nu sunt sincronizate. Unele elemente apar parțial, altele deloc	63
6.4 Sincronizarea elimină erorile de afișare, dar timpii de așteptare vor reduce numărul de cadre ce pot fi afișate pe secundă	63
6.5 Folosirea a două frame buffere	64
6.6 Structura unui frame buffer exemplificată pentru un display cu rezoluția de 8x8	64
6.7 Transferul unei linii (linia 7) către display	65
6.8 Meniul de setări	68
6.9 Setarea luminozității ecranului LCD. Momentan, luminozitatea este la o treime din maxim.	68
6.10 Structura programului principal	70
6.11 Screenshot al aplicației (surse pentru pictograme: cymbalfusion)	71

LISTĂ DE TABELE

1.1 Comparație între dispozitive	12
2.1 Exemplu de raport HID (sursa: USB Made Simple [9])	19
2.2 Tipuri de transfer USB (sursa: extras din USB complete [10, p. 62])	20
2.3 Valori ale factorului de înfășurare (sursa: [12])	21
2.4 Modurile de funcționare ale MX1616 (sursa: [13, p. 3])	24
4.1 Combinățiile favorabile de dinti și poli (sursa: [12])	37
5.1 Caracteristicile interfețelor senzorului MT6701	45
8.1 Descriptorul pentru modul joystick	74
8.2 Descriptorul pentru modul mouse	74
8.3 Descriptorul de intrare pentru modul SmartKnob	74
8.4 Descriptorul de ieșire pentru controlul inelului cu LED-uri	75
8.5 Descriptorul de ieșire pentru controlul motorului	75
8.6 Descriptorul de ieșire pentru controlul bibliotecii grafice	75
8.7 Structura unei comenzi pentru biblioteca grafică	76
8.8 Un descriptor pentru biblioteca grafică	76

LISTĂ DE FRAGMENTE

5.1	Codul mașinii PIO pentru senzorul MT6701	46
5.2	Codul mașinii PIO pentru ecranul LCD	51
5.3	Codul mașinii PIO pentru inelul cu LED-uri	52
5.4	Codul mașinii PIO pentru modulele cu HX711	54
6.1	Parte din codul pentru primirea rapoartelor	56
6.2	Parte din codul pentru trimiterea rapoartelor	57
6.3	Modificările aduse linker script-ului pentru a acomoda memoria frame bufferelor	65
6.4	Trimiterea unei liste pentru randarea imaginii cu un joystick în mijlocul ecranului	66
6.5	Structura ce reține setările și utilizarea acesteia	69
6.6	Codul C# pentru comunicarea cu SmartKnob	72

1. INTRODUCERE

1.1 DOMENII ABORDATE ȘI OBIECTIVE GENERALE

În cadrul acestei lucrări am urmărit construcția unui dispozitiv haptic de control. Acest dispozitiv l-am conceput ca fiind similar conceptual unui joystick, dar aducând multe funcții suplimentare. Alte puncte pe care am dorit să le ating sunt o compatibilitate ridicată cu sistemele de operare și realizarea practică a acestui sistem folosind piese disponibile în piață actuală a electronicelor. Dintre facilitățiile ce le-am dorit integra, cea mai importantă o consider cea de feedback haptic, anume capacitatea butonului de a aplica forță/cuplu.

Pentru a satisface aceste cerințe, am rezolvat probleme ce țin de mecanică (am construit motorul conform unor caracteristici speciale), de electronică (interconectarea diverselor module), dar și de software, atât embedded C pentru microcontroler, cât și C# pentru dezvoltarea unei mici aplicații de test pentru dispozitiv.

Din punct de vedere al funcționalității finale, am urmărit oferirea utilizatorului posibilitatea de a folosi SmartKnob în mai multe moduri. Modurile mai simple permit funcționarea dispozitivului în manieră similară cu un mouse, respectiv un joystick. Pentru a putea beneficia de toate funcționalitățile oferite se va folosi modul SmartKnob.

1.2 CONTEXTUL DE REALIZARE

Ideea acestui sistem a fost inspirată dintr-un proiect văzut pe internet, găsit pe GitHub [1]. Mi s-a părut interesantă ideea de feedback haptic configurabil într-un buton. Am mai întâlnit, bineînțeles, butoane cu mai multe poziții stabile, dar ideea de configurabilitate a acestora din software a prezentat un punct central de interes. Inițial am dorit replicarea proiectului, pentru a experimenta cu acest feedback haptic.

Totuși, proiectul original avea câteva elemente care m-au determinat să refac proiectul de la zero, după propria mea concepție. Proiectul original folosea un motor care nu mai era disponibil pe piață. Astfel am fost nevoit să construiesc propriul motor. Acest fapt a dus la schimbarea totală a părții mecanice. Microcontrolerul folosit în proiectul original nu avea conexiune USB hardware, deci am ales utilizarea unui alt microcontroler. Din cauza hardware-ului specific al microcontrolerelor și necesitatea de viteză ridicată a trebuit să refac complet și componenta software de pe microcontroler. Astfel, proiectul realizat de mine a fost regândit în întregime.

Realizarea practică a acestui sistem a necesitat accesul la o imprimantă 3D pentru construcția carcasei. De asemenea, au fost utile osciloscopul și analizorul logic pentru a putea observa eventualele probleme la nivelul electronic. Nu în ultimul rând, a fost necesar accesul la un laptop cu Pico SDK instalat pentru a putea programa placă Raspberry Pi Pico. Am lucrat la dispozitiv atât în laboratorul pus la dispoziție de facultate, cât și acasă, pe cont propriu, mai ales la partea de software.

1.3 SPECIFICATII GENERALE

Dispozitivul SmartKnob are următoarele funcționalități:

- conexiune USB
- măsurarea unghiului de rotație al butonului
- măsurarea înclinării pe cele două axe orizontale și a apăsării pe axa verticală
- generarea unui feedback haptic folosind un motor
- un ecran LCD în mijlocul butonului
- un inel cu 16 LED-uri în jurul butonului



Figura 1.1: SmartKnob

Din punct de vedere al utilizării, acesta are 3 moduri de funcționare:

- **modul mouse** - sistemul va fi detectat de orice sistem de operare ca fiind un mouse. Pe ecranul LCD va apărea imaginea unui mouse. Înclinarea pe axele orizontale vor deplasa cursorul. Apăsarea butonului corespunde apăsării butonului stâng al mouse-ului. Rotația butonului va avea funcția de derulare (scroll). Feedbackul haptic va simula rotația de derulare a unui mouse obișnuit. Inelul cu LED-uri va afișa 4 LED-uri aprinse care se vor roti o dată cu butonul.
- **modul joystick** - sistemul va fi detectat de orice sistem de operare ca fiind un joystick. Pe ecranul LCD va apărea imaginea unui joystick. Înclinarea pe axele orizontale vor corespunde axelor X și Y ale joystickului. Apăsarea butonului va determina o apăsare scurtă a butonului 2 al joystickului, iar ținerea apăsată a butonului va determina apăsarea continuă a butonului 1 al joystickului. Viteza de rotație a butonului corespunde axelor Z și a axelor de rotație X, Y și Z ale joystickului. Feedbackul haptic este dezactivat în acest mod. Inelul cu LED-uri va afișa 4 LED-uri aprinse care se vor roti o dată cu butonul.
- **modul SmartKnob** - sistemul va fi detectat ca fiind un dispozitiv HID special. O aplicație cu acces la driverele USB poate controla toate funcționalitățile butonului folosind rapoarte HID obișnuite. La intrarea în acest mod, va apărea pe display imaginea SmartKnob-ului, LED-urile vor fi aprinse în mod similar cu cele descrise mai sus, feedback-ul haptic este oprit. Funcționarea acestor module va fi modificată de aplicație.

Dispozitivul final are dimensiunile de 10cm x 10cm x 10cm. Conectorul USB este pe unul din pereții verticali ai carcasei, iar butonul pentru selecția modului se află pe partea de jos a carcasei. Ținerea apăsată a butonului în timpul conectării cablului USB duce la intrarea în meniul de setări. Apăsarea butonului în timpul funcționării în unul din modurile de mai sus duce la intrarea RP2040 în modul de programare, iar apăsarea butonului în modul de setări are funcționalitatea definită de meniul de setări.

1.4 COMPARAȚIE CU STATE OF THE ART

Voi realiza o comparație între proiectul meu, proiectul realizat de scottbez1 [1], Tuya Zigbee Smart Knob [2] și BHT-7000 Smart Knob [3].

TUYA ZIGBEE

Dispozitivul a fost gândit pentru a îndeplini sarcini simple de IoT, fiind conectat prin Zigbee. Acesta necesită o aplicație specializată și poate controla doar becurile Smart Tuya. Utilizabilitatea butonului este, deci, limitată. Pe de altă parte, energia utilizată este foarte redusă. Conform [2], bateria este de tip CR2032 (cu o capacitate de sub 1Wh), iar o baterie se descarcă în perioada de un an. Astfel ar putea înlocui într-o anumită măsură întrerupătoarele clasice.

Prin urmare, dispozitivul poate fi folosit doar pentru a controla dispozitive speciale, simple, și are o plajă limitată de input-uri (rotație și apăsare) și nici un fel de feedback (vizual sau fizic/haptic).



Figura 1.2: Tuya Zigbee (sursa: [2])

BHT-7000

Dispozitivul este destinat a fi încastrat în perete, înlocuind în totalitate un întrerupător clasic. În acest scop este alimentat direct de la rețea și poate controla direct(cu relee integrate) dispozitive electrice de putere: becuri, boilere, alte încălzitoare și are o protecție electrică mai robustă.

Comunicarea cu butonul se poate face numai prin Zigbee și este necesară o aplicație specială, din care se poate configura.

Consumul de energie nu este o problemă în acest caz, butonul fiind integrat în rețeaua electrică a casei. Comparativ cu sarcinile controlate, consumul este neglijabil 1W.

BHT-7000 poate fi folosit pentru a controla o gamă variată de dispozitive electrice, iar controlul poate fi mai complex, butonul fiind dotat și cu un display LCD. Utilizatorul va putea, deci, să își definească un meniu mai avansat dacă dorește.



Figura 1.3: BHT-7000 Smart Knob (sursa: [3])

SMARTKNOB CREAT DE SCOTTBEZ1

Proiectul a fost creat ca un hobby-Project și are ca scop integrarea feedback-ului haptic într-un dispozitiv similar unui buton. În plus, se urmărește un caracter DIY, proiectul fiind ușor de replicat de către oricine are acces la materialele necesare. (Piese 3D-printate și PCB-ul se pot comanda online).

Starea curentă a proiectului (iunie 2024) este un demo și nu are implementată nicio metodă de comunicare cu alte dispozitive. Pe lângă feedbackul haptic inovator, se poate detecta apăsarea fără a necesita un buton traditional, folosind puncte tensometrice.

Display-ul și inelul de LED-uri colorate, împreună cu elementele menționate mai sus permit crearea unor scheme de control complexe, simulând diverse tipuri de butoane rotative.



Figura 1.4: SmartKnob realizat de scottbez1 (sursa: [scottbez1_smatrknob](#))

SMARTKNOB CREAT DE MINE

Am intenționat să creez un buton similar cu cel al lui scottbez1, adică să păstrez caracteristicile principale. Totuși, dispozitivul meu va fi un periferic PC, nu un înlocuitor al intrerupătoarelor clasice.

În acest scop am ales comunicarea USB pentru a crește compatibilitatea cu sistemele moderne. Pentru a nu necesita instalarea de drivere, am optat ca dispozitivul să fie unul de tip HID (human interface device). Astfel dispozitivul va putea comunica cu aplicația cu care va fi folosit indiferent de sistemul de operare pe care aceasta rulează.

Caracterul DIY este, de asemenea, păstrat, SmartKnob fiind făcut din piese 3D printate și piese/submodule disponibile de comandat online.



Figura 1.5: SmartKnob realizat de mine

1.5 PRINCIPALELE REZULTATE OBȚINUTE

Am obținut un sistem de control cu feedback haptic ce are o compatibilitate ridicată cu sistemele de operare. Din punct de vedere fizic, dispozitivul este destul de voluminos, întrucât realizarea manuală are toleranțe mari. Totuși, dispozitivul este suficient de mic pentru a putea fi folosit ca orice alt dispozitiv de intrare pentru PC. Din punct de vedere electric, s-a reușit comunicarea cu toate dispozitivele periferice (senzorii), fără a avea o rată de erori măsurabilă. În final, software-ul creat este modular, permitând modificări relativ simple. Crearea unei aplicații pentru acest sistem a fost, de asemenea, facilitată de faptul că s-a folosit clasa HID pentru implementarea USB.

1.6 STRUCTURA LUCRĂRII

- Introducere - se prezintă aspecte generale despre sistem
- Teorie generală - se prezintă diversele elemente folosite
 - componente electronice folosite
 - interfețe folosite
 - biblioteci software folosite
 - descrierea microcontrolerului și a funcționalităților acestuia
- Arhitectura generală a sistemului
- Implementarea părții mecanice - se descriu
 - proiectarea și construcția motorului
 - cadrul cu punți tensometrice
 - carcasa exterioară
- Implementarea hardware - în acest capitol se prezintă interfațarea cu toate perifericele și conectarea acestora la microcontroler
 - se descriu interfețele de comunicație
 - se arată codul folosit de mașinile PIO, acolo unde este cazul
- Implementarea software
 - se prezintă utilizarea și modurile de funcționare ale diferitelor biblioteci/submodule
 - codul principal (main) este explicat
 - se exemplifică scrierea unei aplicații în C# pentru controlul sistemului
 - se enumera și analizează resursele folosite
- Concluzii
- Anexe - rapoarte HID

Tabelul 1.1: Comparație între dispozitive

Caracteristici	Tuya Zigbee	BHT-7000	Scottbez1	Proiectul meu
Alimentare	Baterie CR2032 3V DC	Alimentare de la rețea(230 VAC)	USB-C	USB-C
Comunicare	Zigbee, BLE	Zigbee + relee de control	-	USB HID
Microcontroler folosit	?	?	ESP32	RP2040 (Raspberry Pi Pico)
Detectarea rotatiei	X	X	X	X
Feedback haptic	-	-	X	X
Ecran	-	240x240 LCD	240x240 LCD	240x240 LCD
Illuminare suplimentară	-	-	16 LED-uri WS2812	16 LED-uri WS2812
Detectarea apăsării	X	X	X	X
Detectarea înclinării	-	-	-	X
Integrare cu dispozitive IoT	X	X	-	-
Versatilitate/configurabilitate	Necesită aplicația specială pentru dispozitiv	Necesită aplicația specială pentru dispozitiv	Software fix	Poate fi folosit de orice aplicație PC cu acces la dispozitive USB
Control direct al becurilor la 230VAC	-	X	-	-
Consum putere (în medie)	Foarte mic <1mW	~1W	?	1W fără feedback haptic, 5W cu feedback haptic activ

2. TEORIE GENERALÄ

2.1 RASPBERRY PI PICO

Raspberry Pi Pico este o placă de dezvoltare bazată pe microcontrolerul Raspberry Pi RP2040. Placa este dotată cu minimul de componente necesare funcționării și programării microcontrolerului RP2040. Prin urmare, placa oferă o flexibilitate mare în ceea ce privește prototiparea cu microcontrolerul RP2040. Mai mult, placa poate fi integrată ca modul SMD (suface-mount) într-un proiect mai mare.

Characteristicile esentiale a Raspberry Pi Pico sunt [4, p. 4]:

- microcontroler RP2040
 - 2MB de memorie flash (pentru program și alte date)
 - conexiune USB pentru alimentare, programare și date
 - conector DIP cu 40(2x20) de pini care expune 26 de pini de uz general (GPIO)
 - port pentru debug ARM SWD (Serial Wire Debug)
 - alimentare flexibilă - USB sau sursă externă; regulator de 3.3V integrat (sistemul fiind de 3.3V)

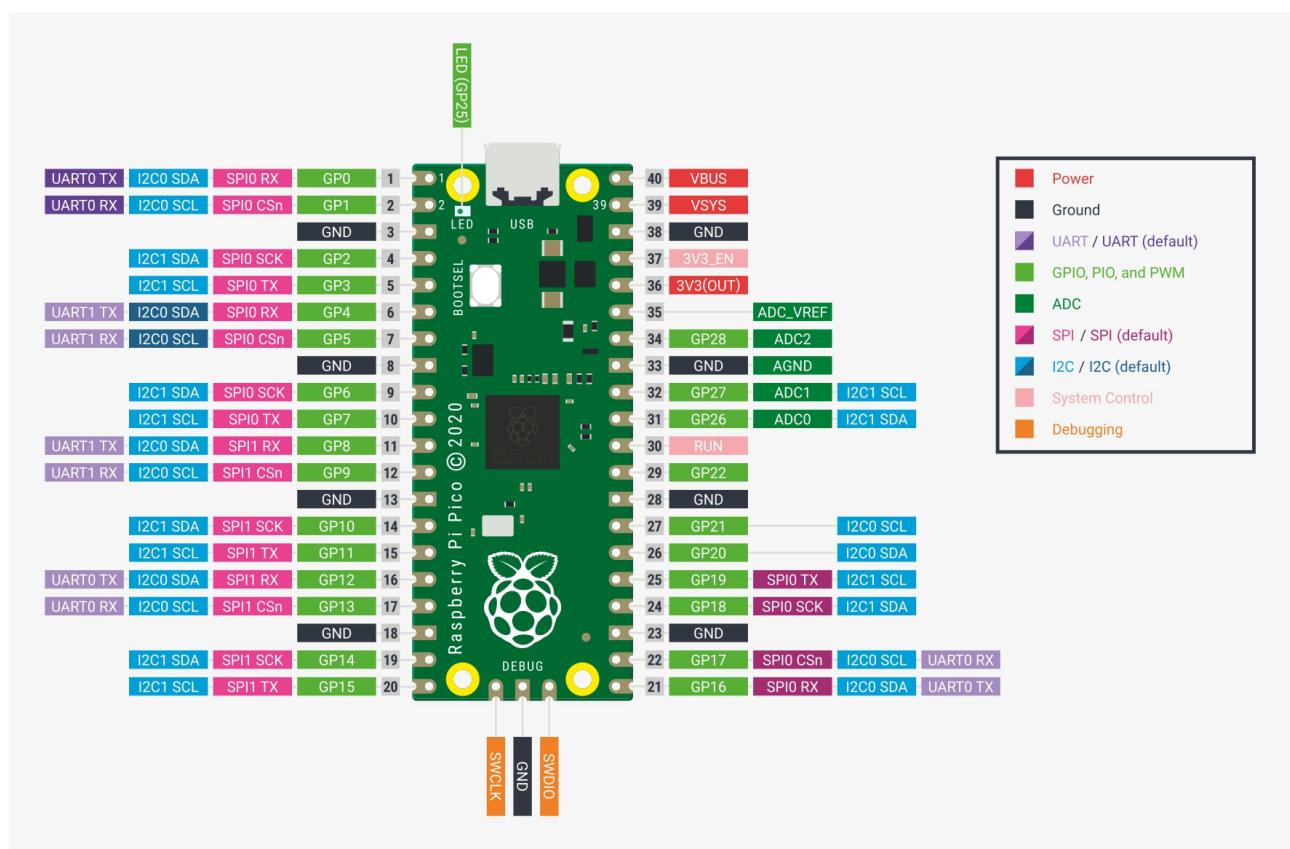


Figura 2.1: Pinout-ul placii Raspberry Pi Pico (sursa: documentatia Raspberry Pi Pico [4, p. 5])

Raspberry Pi Pico expune direct 26 din totalul de 30 de pini de uz general ai RP2040, anume GP0-GP22 și GP26-GP28. Ceilalți 4 pini sunt utilizati pentru diferite funcții specifice:

- **GPIO29** este folosit pentru măsurarea tensiunii de alimentare a plăcii (VSYS/3)
- **GPIO25** este conectat la LED-ul de pe placă
- **GPIO24** este folosit pentru detectarea prezenței tensiunii USB
- **GPIO23** controlează regulatorul de tensiune de pe placă

Butonul **BOOTSEL** este conectat la unul din pinii dedicati memoriei FLASH. Pentru programare, butonul BOOTSEL trebuie ținut apăsat în momentul conectării plăcii la un calculator. Raspberry Pi Pico va apărea ca un dispozitiv de stocare unde se va copia fișierul .uf2 ce conține programul compilat.

Pe lângă pinii de uz general, Raspberry Pi Pico mai expune pini de alimentare:

- **GND, ADC_GND** - masa sistemului și a convertorului analog-digital
- **VBUS** este conectat la pinul de alimentare al conectorului USB
- **VSYS** este alimentarea regulatorului de 3.3V; poate fi 1.8V-5.5V. VBUS este conectat printr-o diodă la VSYS.
- **3V3_EN** pornește/oprește regulatorul de 3.3V pentru microcontroler
- **3V3** ieșirea regulatorului conectată direct la alimentarea microcontrolerului

În plus, pinul **RUN** este pinul de resetare (activ pe 0) al microcontrolerului.

În cazul în care nu se dorește utilizarea conectorului USB, a butonului BOOTSEL sau a LED-ului de pe placă s-au pus la dispoziție 6 test points (pad-uri unde se pot lipi cabluri, sau care vor fi lipite la asamblarea SMD):

- **TP1** - masă
- **TP2** - semnal USB- (DM)
- **TP3** - semnal USB+ (DP)
- **TP4** - conectat direct la GP23 (controlul regulatorului)
- **TP5** - anodul LED-ului de pe placă
- **TP6** - conectat la butonul BOOTSEL

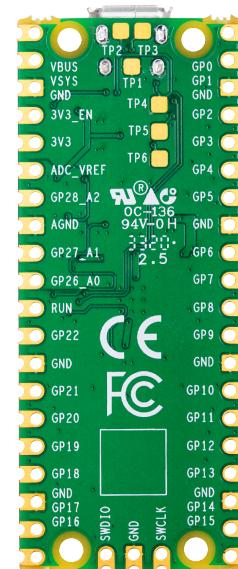


Figura 2.2: Spatele plăcii Raspberry Pi Pico (sursa: documentația Raspberry Pi Pico [4, p. 4])

2.2 MICROCONTROLERUL RP2040

Microcontrolerul RP2040 a fost dezvoltat de fundația Raspberry Pi și oferă o performanță ridicată la un cost scăzut. Acesta este dotat cu două procesoare ARM Cortex M0+, 264kB de SRAM on-chip, un set de periferice variate și un subsistem unic de intrare-iesire, numit PIO, dezvoltat special pentru acest microcontroler [5, p. 10].

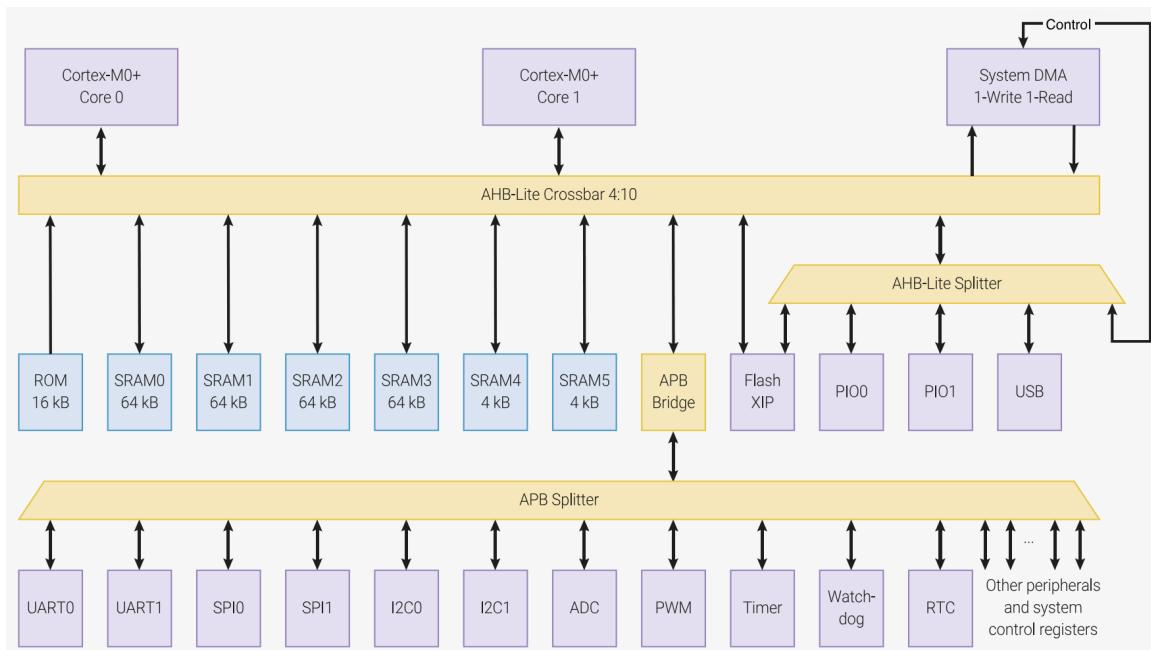


Figura 2.3: Bus fabric-ul microcontrolerului RP2040
(sursa: documentația RP2040 [5, p. 16])

Arhitectura AHB-Lite de tip crossbar permite celor 4 porturi upstream (2 procesoare și sistemul DMA, cu un port de citire și unul de scriere) un transfer paralel și rapid al datelor prin sistem. De exemplu, în același ciclu de tact este posibil ca DMA să facă o scriere, o citire și procesoarele pot, fiecare, să acceseze una din băncile de memorie SRAM sau alte periferice. Prin urmare este facilitată crearea unui sistem eficient care să beneficieze de procesare paralelă. În continuare vor fi prezentate câteva componente ale microcontrolerului RP2040 esențiale pentru această lucrare.

2.2.1 ARM CORTEX M0+

RP2040 are două procesoare ARM Cortex M0+ care folosesc exclusiv setul de instrucțiuni Thumb, deși sunt procesoare pe 32 de biți. Acestea pot funcționa la frecvențe de până la 133MHz. Un împărțitor hardware accelerează operația de împărțire, aceasta putând fi realizată în 8 cicluri de tact. Pentru comunicarea între cele două procesoare există câte un FIFO unidirecțional de 8 intrări a către 32 de biți fiecare, iar pentru o comunicare mai complexă între procesoare sunt disponibile 32 de spin-locks hardware. Pentru a accelera diverse tipuri de operații sunt disponibile câte 2 interpolatoare fiecărui procesor. Interpolatoarele pot fi folosit pentru calcule repetitive în cazul procesării de semnal.

2.2.2 MEMORIILE SISTEMULUI

- Pentru a stoca bootloader-ul, RP2040 are o memorie **ROM** de 16kB. Fiind ROM, aceasta nu poate fi suprascrisă din greșală. Pe lângă bootloader, aceasta mai stochează și diverse funcții utile precum operații în virgulă flotantă.
- Sistemul are memoria internă (**SRAM**) împărțită în mai multe bănci, tocmai pentru a permite accesul paralel. Această memorie este accesibilă într-un singur ciclu de tact. În Figura 2.3 se pot observa dimensiunile băncilor de memorie.
- Pentru a citi programul, subsistemul **XIP** ("execute in place") permite accesarea memoriei **FLASH** externe. De asemenea, XIP conține o memorie cache pentru instrucțiuni de 16kB, set-asociativă pe 2 căi. Această memorie îmbunătățește semnificativ performanța sistemului, întrucât accesul direct la memoria FLASH pentru fiecare instrucțiune ar fi foarte lent. XIP permite și citirea paralelă a datelor din memoria FLASH printr-un canal dedicat, astfel neinfluentând conținutul memoriei cache.

2.2.3 PERIFERICUL USB

RP2040 conține un controler USB conform cu standardul USB 2.0, care este capabil să funcționeze atât ca USB-host, cât și ca USB-device. Legătura cu exteriorul se face prin 2 pini, DP și DM, dedicati interfeței USB. Controlerul hardware se ocupă de partea low-level a protocolului, utilizatorului rămânându-i doar sarcina de configurare și preluare a datelor. Folosind intreruperi se poate eficientiza procesul. Controlerul este dotat cu 4kB de SRAM cu acces dual-port, folosit atât pentru configurare cât și pentru date. [5, p. 383]

2.2.4 PERIFERICUL SPI

RP2040 are două controlere SPI identice bazate pe ARM Primecell Synchronous Serial Port (SSP) (PL022) (Revision r1p4) [5, p. 504]. Acestea pot funcționa atât în mod master, cât și slave și suportă diverse implementări particulare ale protocolului SPI. Pentru un transfer eficient al datelor folosind SPI, un FIFO cu 8 intrări intră în componenta controlerului. Detalii despre funcționarea interfeței SPI se găsesc în capitolul *Interfața SPI*.

2.2.5 PERIFERICUL PWM

Perifericul PWM permite generarea sau măsurarea semnalelor cu modulație a duratei impulsurilor. Acesta are 8 subunități numite "slices", astfel încât se pot genera simultan 8 semnale independente [5, p. 525]. Un contor pe 16 biți și un divizor de clock fractionar de 8.4 biți permit generarea unei game largi de semnale precise. Detalii despre modulația duratei impulsurilor se găsesc în capitolul *Modulația duratei impulsurilor*.

2.2.6 PERIFERICUL PIO

PIO (programmable input/output) este un periferic special dezvoltat și are rolul unei interfețe hardware foarte versante. RP2040 conține două astfel de instance. Cu ajutorul perifericului PIO se pot implementa atât interfețe relativ simple, cum ar fi UART, SPI sau I2C, cât și interfețe complexe: USB, VGA, DVI [5, p. 311].

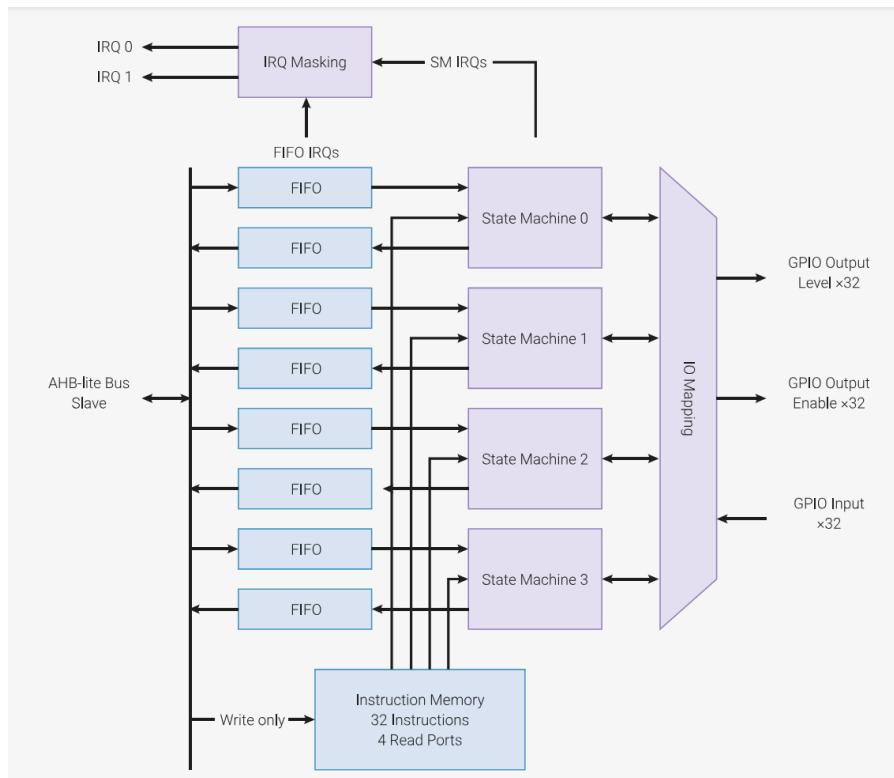


Figura 2.4: Arhitectura perifericului PIO (sursa: documentația RP2040 [5, p. 311])

PIO este programabil asemenei unui procesor. O memorie va reține până la 32 de instrucțiuni PIO, fiecare fiind pe 16 biți. Această memorie alimentează 4 subunități (numite în documentație mașini cu stări finite). Fiecare mașină are propriul program counter. Pentru comunicare, fiecare mașină are câte un FIFO bidirectional pentru citire și scriere. Întreruperi pot fi generate direct de către fiecare mașină. Toți pinii de intrare-iesire sunt disponibili pentru a fi folosiți de către perifericul PIO.

La fiecare ciclu de tact, o mașină PIO poate executa o instrucțiune. Aceste instrucțiuni simple permit prelucrări fine și precise a datelor de intrare-iesire, adică permit preluarea și depunerea datelor la nivel de grupuri de biți și controlarea pinilor la nivel individual. Astfel se pot transfera eficient date între procesor și mediul exterior. Determinismul, setul de instrucțiuni special și divizorul de tact fractionar 16.8 fac PIO o variantă ideală pentru implementarea unei multitudini de interfețe. Mai multe detalli și exemple se găsesc în documentația RP2040 [5, p. 311].

Pico C SDK are diverse facilități pentru a asambla cod pentru PIO și de a transfera codul PIO compilat în memoria de program.

2.3 PICO C SDK

Pentru a facilita programarea plăcii Raspberry Pi Pico, adică a microcontrolerului RP2040, fundația Raspberry Pi pune la dispozitie kitul de dezvoltare Pico C SDK. Acesta pune la dispozitie un API ce permite controlarea majorității funcționalităților microcontrolerului RP2040, inclusiv procesarea paralelă și sincronizarea între procesoare. Abstractizarea și ascunderea complexității acceselor la registre oferă începătorilor în lumea embedded o inițiere mai blandă. Pico C SDK este open-source, deci poate fi folosit și pentru a analiza detaliile de implementare a diverselor funcționalități. Pentru cei care vor să dezvolte un sistem mai complex, Pico C SDK elimină timpul necesar scrierii driverelor hardware (a funcțiilor low-level). Elementele puse la dispozitie de Pico C SDK se pot găsi pe site-ul dedicat [6].

2.4 BIBLIOTECA TINYUSB

TinyUSB este o bibliotecă USB dezvoltată pentru o multitudine de platforme. Pentru o siguranță a memoriei nu s-a folosit alocarea dinamică, iar pentru un control precis și eficient a funcționalităților din bibliotecă se folosesc intreruperi pentru introducerea evenimentelor de procesat într-o coadă. Prelucrarea propriu-zisă a evenimentelor se va face la momentul dorit de utilizator prin apelul unei singure funcții.

Biblioteca este în mare parte configurată prin diverse fișiere de tip header (.h). Astfel codul generat și utilizat de bibliotecă este strictul necesar. De exemplu, codul pentru USB host nu se va regăsi în fișierul compilat dacă se folosește doar funcționalitatea de USB device.

Folosind TinyUSB se poate implementa aproape orice fel de dispozitive USB, de la dispozitive HID (tastaturi, mouse) și de comunicații (CDC) la dispozitive de stocare în masă (MSC). Documentația completă a bibliotecii TinyUSB se găsește în repo-ul de GitHub [7].

2.5 INTERFAȚA USB

Interfața USB (Universal Serial Bus) a fost dezvoltată pentru a permite conectarea unei game variate de dispozitive la un calculator (host), totodată eliminând necesitatea multor cabluri și conectori incompatibili unul cu celălalt. Mă voi rezuma la funcționarea interfeței USB conform standardului 2.0 care este folosit în această lucrare. De asemenea nu voi prezenta interfața electrică, respectiv codarea de linie a informației, întrucât *perifericul USB folosit* are interfața electrică integrată împreună cu decodificările și verificarea erorilor. Folosind USB se pot conecta atât dispozitive ce necesită transfer regulat de date, cât și dispozitive ce transferă cantități mari de date sau chiar a unor dispozitive compuse.

În cadrul protocolului USB, orice dispozitiv trebuie să aibă un set de descriptori. Aceștia sunt seturi de informații despre diferite componente ale dispozitivului. Un descriptor de dispozitiv unic oferă informații generale despre dispozitiv, cum ar fi ID-ul producătorului, standardul USB implementat, numărul de configurații și numele dispozitivului. O configurație reprezintă un mod de funcționare al unui dispozitiv USB. În practică, multe dispozitive au o singură configurație. Descriptorul de configurație oferă informații despre interfețele dispozitivului USB.

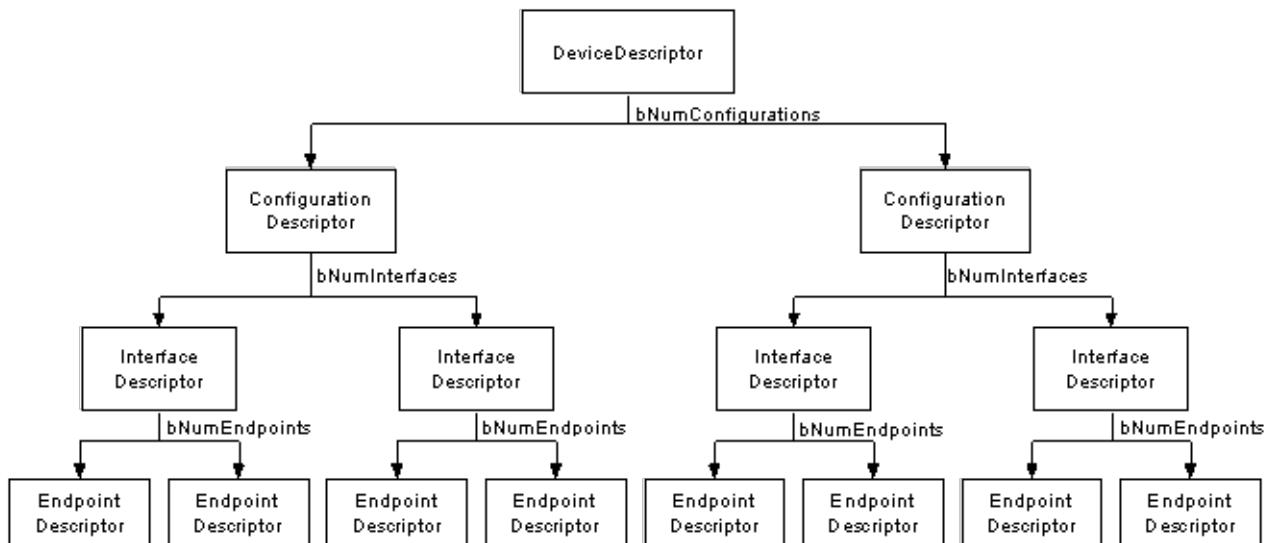


Figura 2.5: Tipuri de descriptori USB (sursa: USB in a nutshell [8])

O interfață poate fi asemănătoare cu un dispozitiv logic. De exemplu, un webcam USB poate avea o interfață pentru stream-ul video și o altă interfață pentru stream-ul audio. Practic se integrează dispozitivul "cameră" cu dispozitivul "microfon" într-un singur dispozitiv USB. Un descriptor de interfață conține informații despre clasa din care face parte și despre numărul de endpoint-uri. Un endpoint reprezintă o unitate adresabilă în cadrul protocolului USB căreia îi se trimit sau îi se cer date de la controlerul USB principal (host).

O clasă USB reprezintă o categorie de dispozitive având caracteristici similare și care îndeplinește funcții asemănătoare. Respectarea specificațiilor acestor clase la implementarea dispozitivelor asigură o compatibilitate a acestora cu driverele diferitelor sisteme de operare. Clasa HID (human interface device) impune existența unui descriptor HID, care oferă informații despre ce date vor fi furnizate de către dispozitiv. De exemplu, descriptorul HID al unui mouse USB va conține numărul de butoane și ce fel de roți pentru derulare are mouse-ul. Ulterior, datele propriu-zise vor fi transmise printr-un raport HID. Structura acestuia este, de asemenea, descrisă în descriptorul de raport. În tabelul următor se poate observa un exemplu de raport HID pentru un mouse simplu cu 5 butoane și o roată de derulare.

Tabelul 2.1: Exemplu de raport HID (sursa: USB Made Simple [9])

Usage	Bits
Button 1	1 Bit
Button 2	1 Bit
Button 3	1 Bit
Button 4	1 Bit
Button 5	1 Bit
Not Used	3 Bits
X	8 Bits
Y	8 Bits
Wheel	8 Bits

Din punct de vedere al datelor care se transmit prin cablurile de interconexiune, informația se transmite sub formă de:

- **transfer** - cea mai mare unitate de informație, alcătuită din una sau mai multe tranzacții; pot fi de tip IN (date de la dispozitiv la host), OUT (date de la host la dispozitiv) și SETUP/CONTROL (bidirectional, pentru configurarea dispozitivului).
- **tranzacție** - subunitate de informație în care datele utile (excluzând adrese, CRC, ...) circulă unidirectional; sunt alcătuite din pachete
- **pachet** - cel mai mic element al transmisiei de date; are în componență să mai multe câmpuri de informații pe un număr variabil de biți; un pachet este transmis pe cablu ca o secvență neîntreruptă

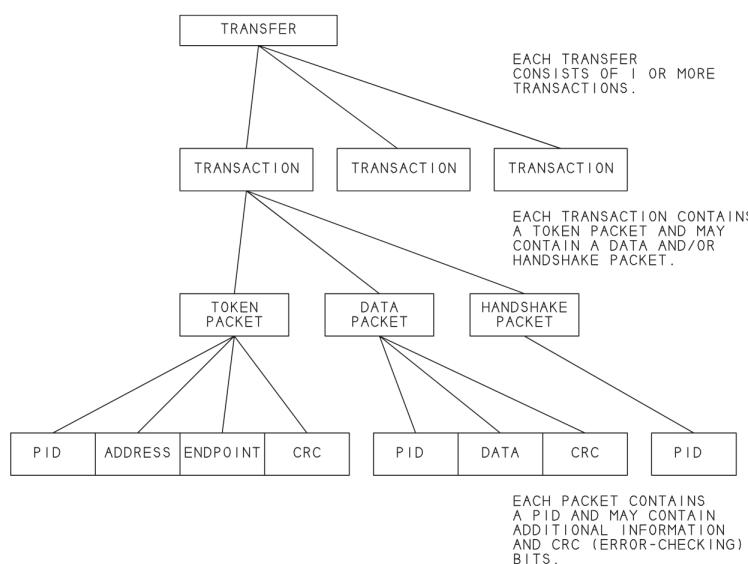


Figura 2.6: Structura unui transfer USB (sursa: USB complete [10, p. 65])

Un transfer USB poate fi de 4 tipuri (pe lângă tipurile descrise mai sus, date de direcția datelor), acestea fiind corelate cu tipurile endpoint-urilor și utilizate în diferite scopuri:

Tabelul 2.2: Tipuri de transfer USB (sursa: extras din USB complete [10, p. 62])

	control	bulk	interrupt	isochronous
utilizare	configurare	cantități mari de date	puține date care necesită transmisie imediată	flux continuu de date
lățime de bandă rezervată	10%-20%	nu	80%-90%	
detectie a erorilor	da	da	da	nu
rată de trimitere garantată	nu	nu	nu	da
latentă garantată	nu	nu	da	da

2.6 MOTORUL BLDC

Motorul fără perii de curent continuu (BLDC) este folosit în multe domenii de activitate: de la navomodelle-hobby la roboți industriali și autovehicule electrice. Avantajele acestui tip de motor sunt durabilitatea și eficiența acestuia. Cu un controler adecvat, motorul BLDC poate fi silentios.

Din punct de vedere structural, orice motor are un rotor (partea mobilă) și un stator (partea fixă). În cazul motoarelor BLDC, statorul este dotat cu bobine înfășurate pe un număr de dinți. Rotorul este alcătuit din magneti.

Configurația motorului din diagramă, dar și cel folosit în această lucrare este de tip outrunner, adică rotorul este în exterior iar statorul în interior.

Motoarele pe care le voi considera vor fi exclusiv cu 3 faze și înfășurări concentrate (doar o bobină/fază pe un dintă).

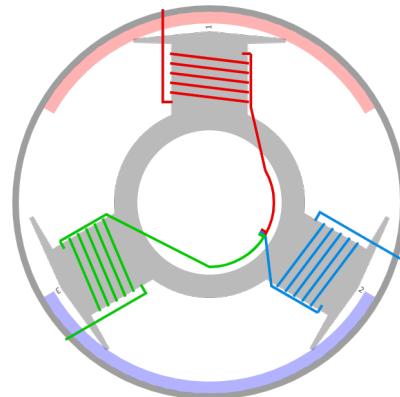


Figura 2.7: Secțiunea unui motor simplu BLDC cu 3 dinți și 2 poli (sursa: Bavaria-direct [11])

Pentru determinarea numărului ideal de dinți și poli, materialul din care să fie realizat statorul și direcțiile de înfășurare a bobinelor trebuie luati în considerare mai mulți factori.

Factorul de înfășurare, k_w , este un număr între 0 și 1 ce reprezintă fractiunea de curent armătură (bobinaj) care este folosită pentru a produce moment al forței [12]. Site-ul [12] prezintă valorile lui k_w pentru diferite combinații de dinți și poli. Din cauza faptului că se folosesc 3 faze, numărul de dinți trebuie să fie multiplu de 3, iar din cauza faptului că polii magnetici trebuie să alterneze, numărul de poli va fi mereu par.

Tabelul 2.3: Valori ale factorului de înfășurare (sursa: [12])

Ns	Nm											
	2	4	6	8	10	12	14	16	18	20	22	
3	0.866	0.866	0.000	-0.866	-0.866	0.000	0.866	0.866	0.000	-0.866	-0.866	
6	0.500	0.866	1.000	0.866	0.500	0.000	-0.500	-0.866	-1.000	-0.866	-0.500	
9	0.328	0.617	0.866	0.945	0.945	0.866	0.617	0.328	0.000	-0.328	-0.617	
12	0.250	0.500	0.683	0.866	0.933	1.000	0.933	0.866	0.683	0.500	0.250	
15	0.199	0.389	0.562	0.711	0.866	0.910	0.951	0.951	0.910	0.866	0.711	
18	0.167	0.328	0.500	0.617	0.735	0.866	0.902	0.945	1.000	0.945	0.902	
21	0.142	0.282	0.415	0.538	0.650	0.747	0.866	0.890	0.932	0.953	0.953	
24	0.125	0.250	0.366	0.500	0.583	0.683	0.760	0.866	0.885	0.933	0.949	
27	0.111	0.220	0.328	0.429	0.525	0.617	0.695	0.766	0.866	0.877	0.915	
30	0.100	0.199	0.296	0.389	0.500	0.562	0.640	0.711	0.774	0.866	0.874	

Factorul q oferă informații despre interacțiunea între înfășurări și magneti. Site-ul [12] prezintă valorile lui 1 pentru diferite combinații de dinti și poli.

Dacă $q < 0.25$, un dintă de stator va interacționa în orice moment cu mai mulți poli de sens opus. Astfel forțele aplicate asupra perechilor de poli opuși se reduc (nu neapărat complet). După cum se poate observa în diagramă, un număr mic de dinti și un număr mare de poli face ca $q < 0.25$.

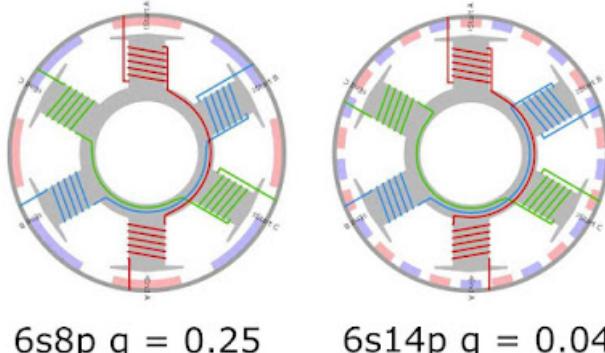


Figura 2.8: Exemple de motoare cu $q < 0.25$ (sursa: Bavaria-direct [11])

Dacă $q > 0.5$, un pol va fi influențat de mai mulți dinti (de mai multe faze) simultan. Astfel forțele generate de faze alăturate se pot reduce (nu neapărat complet). După cum se poate observa în diagramă, un număr mare de dinti și un număr mic de poli face ca $q > 0.5$.

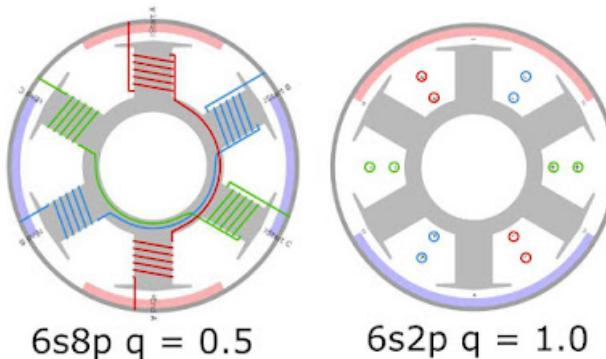


Figura 2.9: Exemple de motoare cu $q > 0.5$ (sursa: Bavaria-direct [11])

Trebuie eliminate și combinații în care numărul de dinti este egal cu numărul de poli, deoarece în acest caz polii se vor suprapune exact peste dinti și astfel nu se va mai putea genera moment al forței (forța este total normală, fără nicio componentă tangențială).

De asemenea, este recomandabil ca statorul să prezinte simetrie (secțiuni repetitive de înfășurări). Astfel se pot reduce vibrații, forțele fiind echilibrate pe părți opuse ale rotorului.

În final, fenomenul de **cogging** denotă tendința motorului de a reveni într-o poziție în care polii se suprapun peste dinti datorită atracției între acestia. Cogging-ul se manifestă, deci, chiar și când motorul nu este alimentat. Cu alte cuvinte, va trebui să folosim energie (și calcule adiționale) pentru a da impresia că motorul are o rotație uniformă. Numărul de pași de cogging este dat de cel mai mic multiplu comun între numărul de dinti și cel de poli. Cu cât numărul de pași de cogging este mai mare, cu atât efectul se va simți mai puțin. Pentru a elmina acest efect se poate realiza un stator oblic sau se poate realiza statorul dintr-un material nonferomagnetic, nu fără o pierdere a momentului forței dezvoltat de motor.

2.7 MODULAȚIA DURATEI IMPULSURILOR (PWM)

Tehnica modulației duratei impulsurilor (PWM) este folosită pentru a reprezenta un semnal analogic folosind un semnal digital. Dacă T este perioada semnalului, o fracțiune D (duty cycle) din acest timp semnalul digital va avea valoarea 1 logic, adică nivelul nominal de tensiune U_0 , iar restul fracțiunii de timp semnalul va avea valoarea 0. Prin urmare, semnalul digital rezultat poate genera efecte similare unui semnal analogic de valoare $D * U_0$. Cu alte cuvinte se poate simula un semnal analogic cu valori în intervalul $[0, U_0]$. Totuși, efectele semnalului PWM trebuie înțelese în cazul particular folosit. Cu alte cuvinte, nu se poate înlocui orice semnal analogic cu unul PWM, dar nici invers.

În figura alăturată se poate observa un semnal PWM. Timpul în care semnalul este pozitiv este notat cu T_{on} , iar timpul în care acesta este zero este notat cu T_{off} . Relațiile de legătură între termeni sunt:

$$T = T_{on} + T_{off} \quad (2.1)$$

$$D = \frac{T_{on}}{T} \quad (2.2)$$

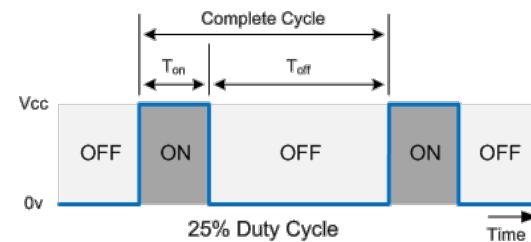


Figura 2.10: Semnal PWM (sursa: OnionDOCS)

Un exemplu des întâlnit de utilizare a PWM este controlul intensității luminoase al unui LED. Datorită faptului că intensitatea luminoasă generată de LED este direct proporțională cu intensitatea curentului care îl străbate, putem calcula intensitatea luminoasă medie a LED-ului pe o perioadă de timp egală cu perioada semnalului PWM. Consider I_0 curentul maxim prin LED, adică acel curent când semnalul PWM are valoarea 1 logic.

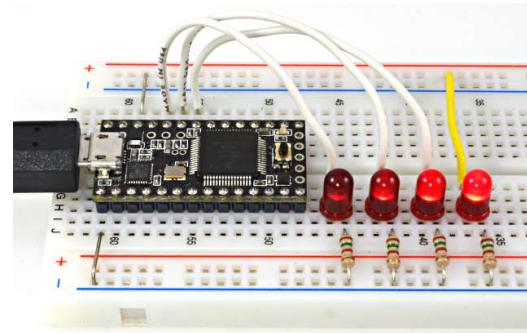


Figura 2.11: LED-uri controlate PWM cu duty cycle de 0.04, 0.2, 0.47 și 1 (alimentat direct) (sursa: PJRC)

$$I_{mediu} = \frac{\int_{T_0}^{T_0+T} I(t)dt}{T} = \frac{\int_{T_0}^{T_0+T_{on}} I_0 dt + \int_{T_0+T_{on}}^T 0 dt}{T} = \frac{I_0 * T_{on}}{T} = I_0 * D \quad (2.3)$$

Intensitatea luminoasă medie percepă va fi proporțională cu $I_0 * D$. Prin urmare putem controla intensitatea luminoasă variind parametrul D al semnalului. Această concluzie va fi valabilă pentru mai multe dispozitive controlate în curent, din care și solenoizii și unele tipuri de motoare de curent continuu.

2.8 DRIVERUL DE MOTOR MX1616

Controlul motoarelor implică posibilitatea de a comanda un curent de valori relativ mari. Acest lucru s-ar putea realiza cu câte un tranzistor per motor. Totuși, în cazul motoarelor mai complexe (BLDC, stepper) sau în cazul în care se dorește și controlul direcției rotației motorului este necesară utilizarea unor circuite speciale de control.

Circuitul MX1616 poate controla două motoare obișnuite (de curent continuu) folosind câte o punte H pentru fiecare. În plus, circuitul este prevăzut și cu diode de protecție (flyback), reducând astfel numărul de componente externe.

Fiecare motor poate fi comandat în unul din modurile următoare:

- forward - motorul se va roti într-un sens
- reverse - motorul se va roti în sens invers
- off - amândouă ieșirile sunt în impedanță ridicată
- brake - amândouă ieșirile sunt conectate la masă

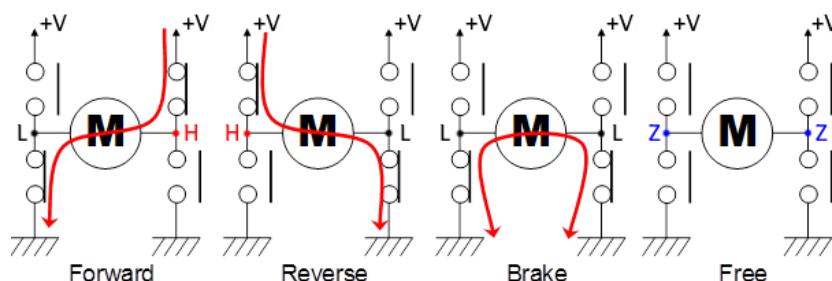


Figura 2.13: Curentul prin motor în diferitele moduri
(sursa: EEWeb)

Pentru a putea realiza aceste 4 combinații, fiecare canal pentru motor are două intrări (INAx și INBx). Funcționarea acestora se poate rezuma în următorul tabel. Mentionez că documentația originală am găsit-o doar în chineză, astfel încât am fost nevoie să refac tabelul cu unele câmpuri traduse.

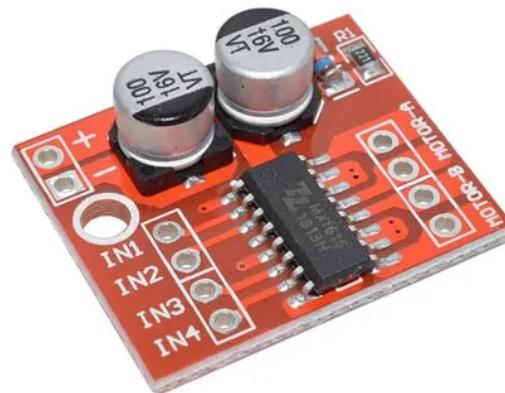


Figura 2.12: Modul cu circuitul MX1616 (sursa: Art of Circuits)

Tabelul 2.4: Modurile de funcționare ale MX1616 (sursa: [13, p. 3])

INAx	INBx	OUTAx	OUTBx	Function
L	L	Z	Z	off
H	L	H	L	forward
L	H	L	H	reverse
H	H	L	L	brake

2.9 INTERFAȚA SPI

Interfața SPI este o interfață serială, sincronă, care permite viteze de transfer relativ mari. Este de reținut faptul că interfața SPI nu are o specificație oficială, ceea ce poate fi considerat un avantaj, întrucât producătorii pot aduce mici modificări benefice diverselor componente dezvoltate. În același timp, această libertate poate duce la dificultăți de interconectare. [14, p. 1]

Interfața este în majoritatea cazurilor compusă din 4 linii unidirecționale, care pot purta diferite nume:

- \overline{CS} , \overline{CE} , \overline{SEL} , \overline{SS} - activează comunicația pe interfață
- CLK, SCK, SCL - semnalul de tact al interfeței
- MOSI, SDO, COPI - linia de date de la controler la periferic
- MISO, SDI, CIPO - linia de date de la periferic la controler

Când nu se transmit informații pe interfață, linia \overline{CS} este inactivă (1 logic), iar linia SCK este și ea în starea ei inactivă. Pentru a începe un transfer de date, linia \overline{CS} se pune pe 0 logic de către master/controler. Aceasta începe să genereze impulsuri de tact pe linia SCK. Cu fiecare ciclu de tact, controlerul va pune un bit de date pe linia MOSI, iar perifericul va pune un bit de date pe linia MISO. După ce toți bitii de date au fost transmiși, comunicația este încheiată de către master prin punerea liniei \overline{CS} pe 1 logic.

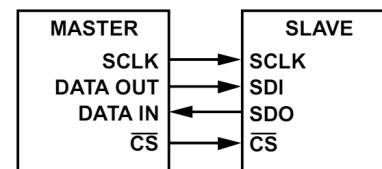


Figura 2.14: Conexiune simplă a interfeței SPI (sursa: [14, p. 1])

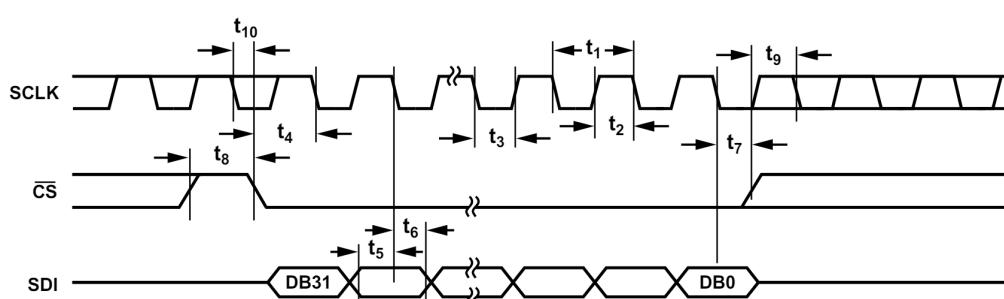


Figura 2.15: Exemplu de transfer de date prin interfață SPI (sursa: [14, p. 1])

Interfața SPI are mai multe moduri de funcționare, în funcție de starea inactivă a semnalului SCK și de frontul semnalului de tact pe care se citesc datele. Starea inactivă a SCK este dată de polaritatea semnalului, numită **CPOL**. În cazul în care **CPOL** este 0, atunci semnalul SCK este 0 logic în starea de repaus (inactivă) și invers. Faza semnalului, **CPHA**, alege frontul pe care se citesc datele. Atunci când **CPHA** este 0, primul front este cel de citire, iar cel de-al doilea este de schimbare a datelor și atunci când **CPHA** este 1, primul front este cel de schimbare, iar cel de-al doilea este de citire a datelor.

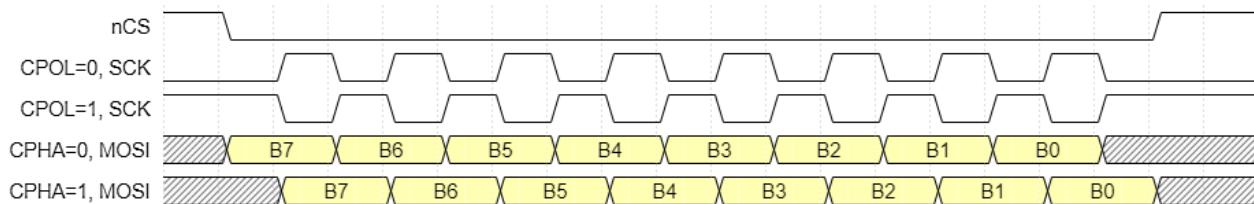


Figura 2.16: Ilustrare a modurilor interfeței SPI
(diagramă generată cu <https://wavedrom.com>)

Pe lângă modurile de funcționare, diverse dispozitive aduc diverse modificări interfeței SPI. Din această cauză compatibilitatea comunicării trebuie analizată de la caz la caz. Printre aceste modificări se numără:

- o singură linie bidirectională de date în loc de MOSI și MISO. Astfel comunicarea nu mai poate fi full-duplex.
- biți adiționali de așteptare introdusi în date (unul sau mai multe cicluri de tact nu produc date utile)
- absența semnalului de selecție, CS
- atribuirea mai multor funcții aceleiași linii. De exemplu, linia de date poate semnaliza și un anumit eveniment
- linii adiționale de comandă

2.10 SENZORUL MAGNETIC MT6701

Senzorul magnetic MT6701 se bazează pe tehnologia senzorilor Hall și are următoarele caracteristici :

- tensiuni de funcționare 3.3V-5.5V
- o multitudine de interfețe disponibile: I2C, SPI, ABZ, UVW, PWM și analogic
- diverse caracteristici, cum ar fi punctul de 0, programabile prin I2C
- întârzieri de 5us

Pentru a funcționa este nevoie ca un magnet diametric să fie atașat axului al cărui unghi vrem să îl măsurăm. Magnetul trebuie să ajungă deasupra circuitului integrat, paralel cu acesta, ca în figură.

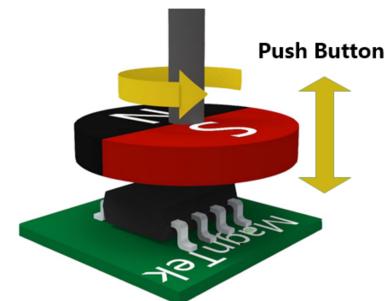


Figura 2.17: Senzorul magnetic MT6701 (sursa: [15, p. 1])

Dacă se folosește interfața SPI, pentru comunicație se vor folosi 3 linii: CSN (\overline{CS}), CLK (SCK) și DO (MISO). Nu există nicio linie de MOSI, întrucât datele circulă mereu unidirectional de la senzor la controler. Prin interfața SPI se poate citi direct unghiul ca număr pe 14 biți, dar și informații despre puterea câmpului magnetic și un câmp CRC. Toate aceste date sunt transmise ca un singur bloc continuu de 24 de biți.

2.11 ECRANUL LCD GC9A01

Un display LCD este un tip de display care se folosește de polarizarea luminii și de cristale lichide pentru a produce imagini. El facilitează transmiterea informației către utilizator în mod grafic. Pentru a elimina problema unghiului restrâns de vizionare a display-urilor LCD clasice s-a dezvoltat tehnologia IPS (in-plane switching), eliminând practic complet constrângerea de unghi de vizualizare. Display-ul LCD ales este de tip IPS. Termenul "GC9A01" se referă, de fapt, la circuitul integrat de control al ecranului propriu-zis, nu la întreg ansamblul display-ului (circuit integrat de control + ecran). Totuși, pe piață, numele circuitului de control se confundă cu cel al ansamblului întreg. Mai mult, displayul este, de regulă, cumpărat deja lipit pe un PCB, care mai poate avea diverse componente pentru a facilita conectarea display-ului. Prin urmare mă voi referi prin "GC9A01" la întreg ansamblul display.

Display-ul are următoarele caracteristici:

- rezoluție de 240x240px (cerc cu raza de 120px)
- culoare pe maxim 18 biți
- control al luminozității prin PWM
- interfață de comunicare SPI cu o linie de date și frecvență de până la 100MHz la scrierea datelor
- NU** prezintă nicio metodă de sincronizare verticală

Pe lângă pinii de comunicare SPI, mai este pus la dispoziție pinul D/C. Acesta selectează dacă datele transmise prin interfața SPI vor fi interpretate ca date sau comenzi.



Figura 2.18: Ansamblul display GC9A01 (sursa: Makerfabs)

În documentație [16], interfața de comunicație pusă la dispoziție este numită "4-line Serial Interface I".

Pentru a afișa imagini pe display, trebuie să se trimită comenzi către acesta pentru a scrie în memoria sa RAM. GC9A01 este dotat cu 172800B de RAM pentru a reține imaginea afișată. O comandă are un cod care trebuie transmis cu D/C=0 și 0 sau mai mulți parametri/date care trebuie transmise cu D/C=1.

O caracteristică importantă a comenzi de scriere a datelor de afișat, biții pot fi transmiși ca un flux continuu de date, fără a mai fi necesară trimiterea unei alte comenzi sau a modificării altor pini în afară de SCK/SCL și MOSI/SDA. Astfel viteza de transfer este limitată doar de frecvența maximă a interfeței SPI.

Pinii display-ului sunt:

- GND, VCC - alimentare
- SCL, SDA, DC, CS - comunicare
- RST - semnal de reset
- BLK - controlul iluminării de fundal

2.12 TENSOMETRE

Tensometrul este un dispozitiv ce are scopul de a măsura propria deformare. Acesta se atașează permanent (prin lipire) de un obiect de interes a cărui deformare se dorește, de fapt, a fi măsurată.

Funcționarea tensometrelor are la bază ecuația rezistivității electrice:

$$R = \rho * \frac{L}{S} \quad (2.4)$$

- ρ este rezistivitatea materialului din care este confectionat conductorul
- L este lungimea conductorului
- S este aria secțiunii conductorului

Tensometrul are în componență său un conductor lung care este dispus în formă de "S" cu părțile paralele cu deformarea care se dorește a fi măsurată.

Când tensometrul este întins (îmaginea B din figura alăturată), lungimea conductorului crește, iar suprafața secțiunii acestuia scade. Conform relației Equation 2.4, rezistența tensometrului va crește.

Când tensometrul este comprimat (îmaginea C din figura alăturată), lungimea conductorului scade, iar suprafața secțiunii acestuia crește. Conform relației Equation 2.4, rezistența tensometrului va scădea.

Orice conductor prezintă caracteristicile acestea, dar tensometrul are forma necesară pentru o măsurare eficientă a deformării. Chiar și așa, variația rezistenței tensometrului este destul de mică, anume în jur de 0.1%.

Din cauza variațiilor mici de rezistență se impune utilizarea unor tehnici pentru a amplifica semnalul util:

- se vor construi punți tensometrici din 2 sau 4 punți
- se vor folosi amplificatoare de semnal
- se vor folosi convertoare analog-digitale speciale

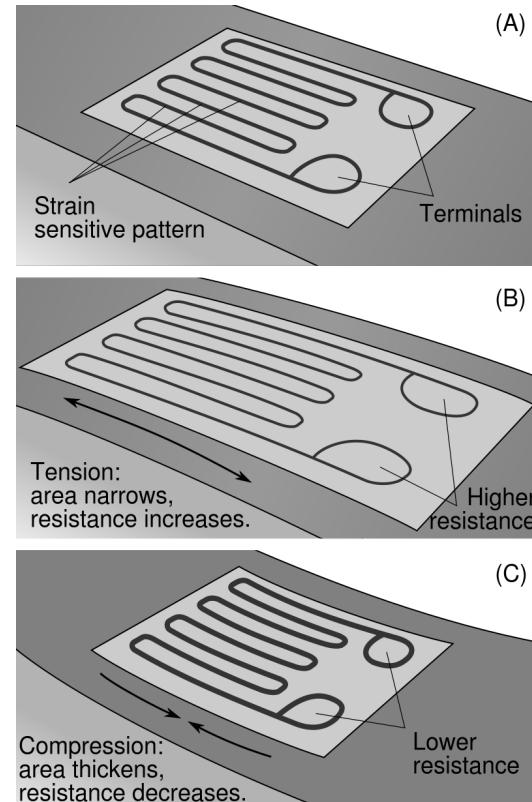


Figura 2.19: Funcționarea tensometrelor (sursa: GaugeHow)

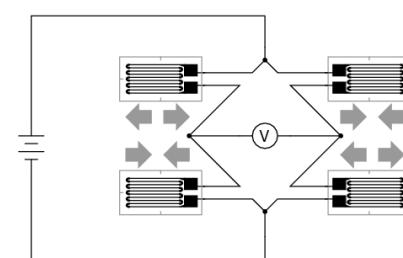


Figura 2.20: Punte tensometrică (sursa: SparkFun)

2.13 AMPLIFICATORUL HX711

Pentru a putea măsura semnalul generat de tensometre (punți tensometrice), trebuie să se folosească un amplificator și un convertor analog-digital de mare precizie. Aceste două componente se găsesc în circuitul integrat HX711. În plus, acesta mai conține și o sursă de tensiune stabilizată pentru ca puntile tensometrice folosite să nu fie influențate de perturbații de pe liniile de alimentare. HX711 a fost conceput special pentru utilizarea împreună cu tensometrele. [17, p. 1]

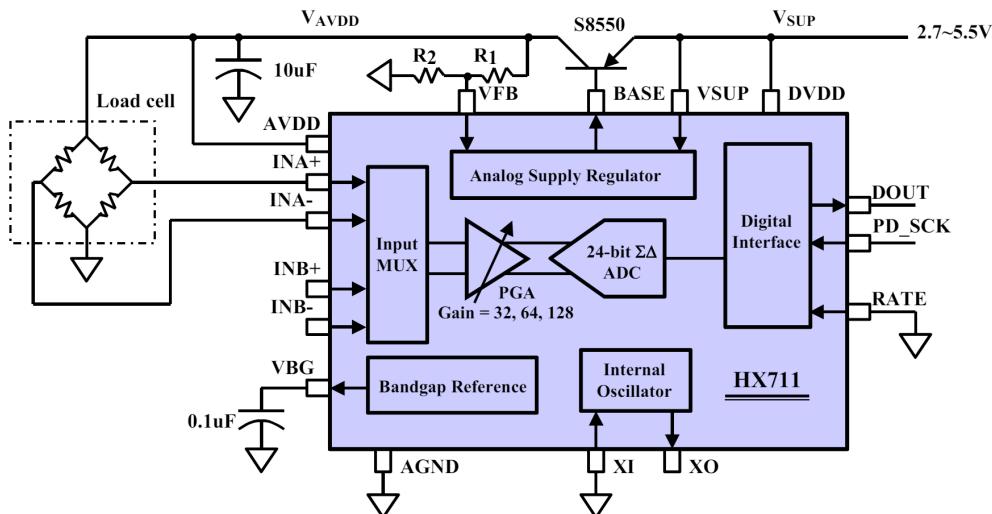


Figura 2.21: Aplicație tipică a circuitului HX711 (sursa: [17, p. 1])

Majoritatea plăcilor cu amplificator HX711 care se găsesc pe piață (incluzând cele folosite) implementează schema de mai sus.

Caracteristicile cele mai importante care fac HX711 să fie ideal pentru măsurarea semnalului generat sunt [17, p. 1]:

- intrare diferențială cu 3 amplificări disponibile
- rezoluție foarte mare, de 24 de biți
- amplificator, convertor și regulator integrate într-un singur circuit
- interfață simplă care necesită doar două linii
- necesită relativ puține componente externe

Comunicarea cu acest dispozitiv are două linii: una de tact (PD_SCK) și una de date (DOUT). Fiecare linie are mai multe funcții. Linia DOUT este atât ieșirea de date, cât și un indicator al finalizării noii conversiuni. Linia PD_SCK (tact) are și funcție de a-i transmite circuitului setarea de amplificare în funcție de numărul de impulsuri generate, dar și comanda de power-down, dacă este folosită. [17, p. 4-5]

2.14 LED-UL WS2812

WS2812 este un integrat care conține atât un LED tricolor (RGB), dar și un circuit de control. WS2812 a fost conceput pentru a putea înseria multe unități, astfel reducând numărul de linii necesar controlării mai multor LED-uri.

Comunicarea se desfășoară exclusiv digital. Fiecare circuit își controlează în curent propriul LED, ceea ce permite obținerea unei fidelități mari a culorilor.

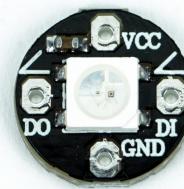


Figura 2.22: Un singur LED WS2812 (sursa: Ardushop)

Conecțarea se realizează în felul următor: de la controler, o linie de date va fi conectată la intrarea de date (**DI**) a primului LED, apoi ieșirea de date (**DO**) a primului LED va fi conectată la intrarea de date (**DI**) a celui de-al doilea LED și așa mai departe. Prin urmare, o singură linie este suficientă pentru a controla multe (până la 1204 [18, p. 1]) LED-uri.

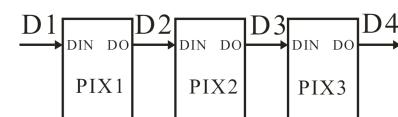


Figura 2.23: Conecțarea a 3 LED-uri WS2812 în serie (sursa: [18, p. 4])

Prin interfața serială (linia (**DI**)) se pot transmite 3 coduri: un bit de 0, un bit de 1, sau codul de resetare, (**RET**). Forma de undă necesară transmiterii acestora se observă în diagrama de mai jos. Pentru a transmite informația unui LED, prima dată se transmite codul de resetare, iar apoi 24 de biți de date, cel mai semnificativ bit primul (MSB). Din aceștia, primii 8 vor corespunde canalului verde, următorii 8 canalului roșu, iar ultimii 8 celui albastru.

Dacă sunt mai multe LED-uri de controlat, atunci datele (grupurile de 24 de biți) se transmit secvențial, după codul de resetare. LED-urile își vor transmite automat informațiile necesare de la unul la celălalt.

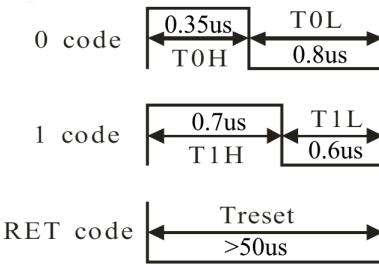


Figura 2.24: Codurile ce pot fi transmise către WS2812 (sursa: [18, p. 4])

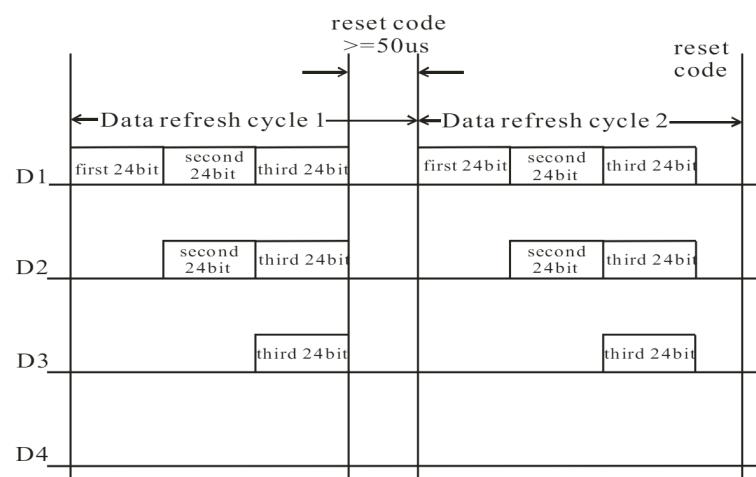


Figura 2.25: Comunicarea cu mai multe LED-uri WS2812 (sursa: [18, p. 5])

3. ARHITECTURA GENERALĂ A SISTEMULUI

3.1 ARHITECTURA SOFTWARE A SISTEMULUI

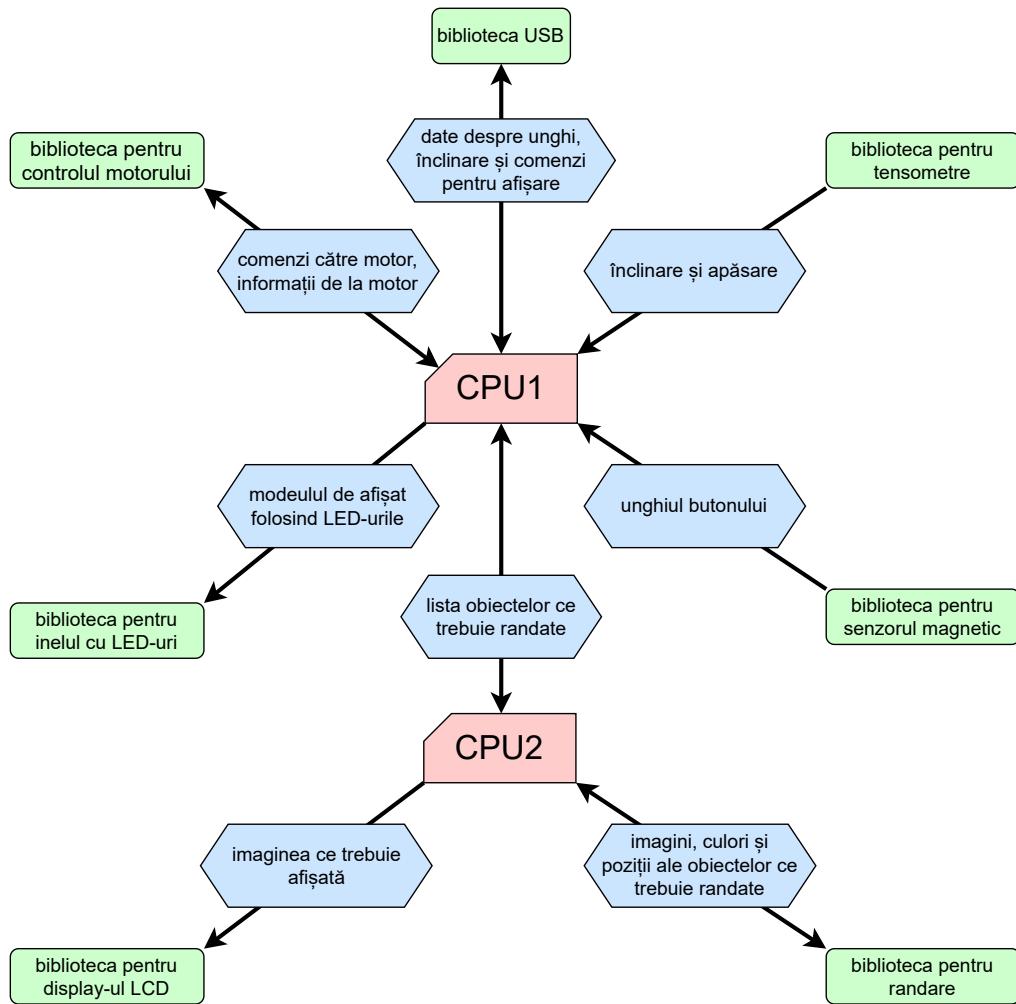


Figura 3.1: Arhitectura software a sistemului
(diagramă generată cu draw.io)

Din punct de vedere software, sistemul are mai multe componente independente. Pentru fiecare din acestea s-a creat o bibliotecă (reprezentate cu verde în figură) care comunică diverse date (reprezentate cu albastru în figură) procesoarelor (reprezentate cu roșu în figură).

Având la dispoziție două procesoare, am împărțit sarcinile astfel încât puterea de procesare să fie utilizată optim. În acest scop grafica este deservită exclusiv de un singur procesor, iar restul funcțiilor sistemului de celălalt.

Procesorul **CPU1** are rolul și de a aloca resursele și de a controla **CPU2**.

3.2 ARHITECTURA HARDWARE A SISTEMULUI

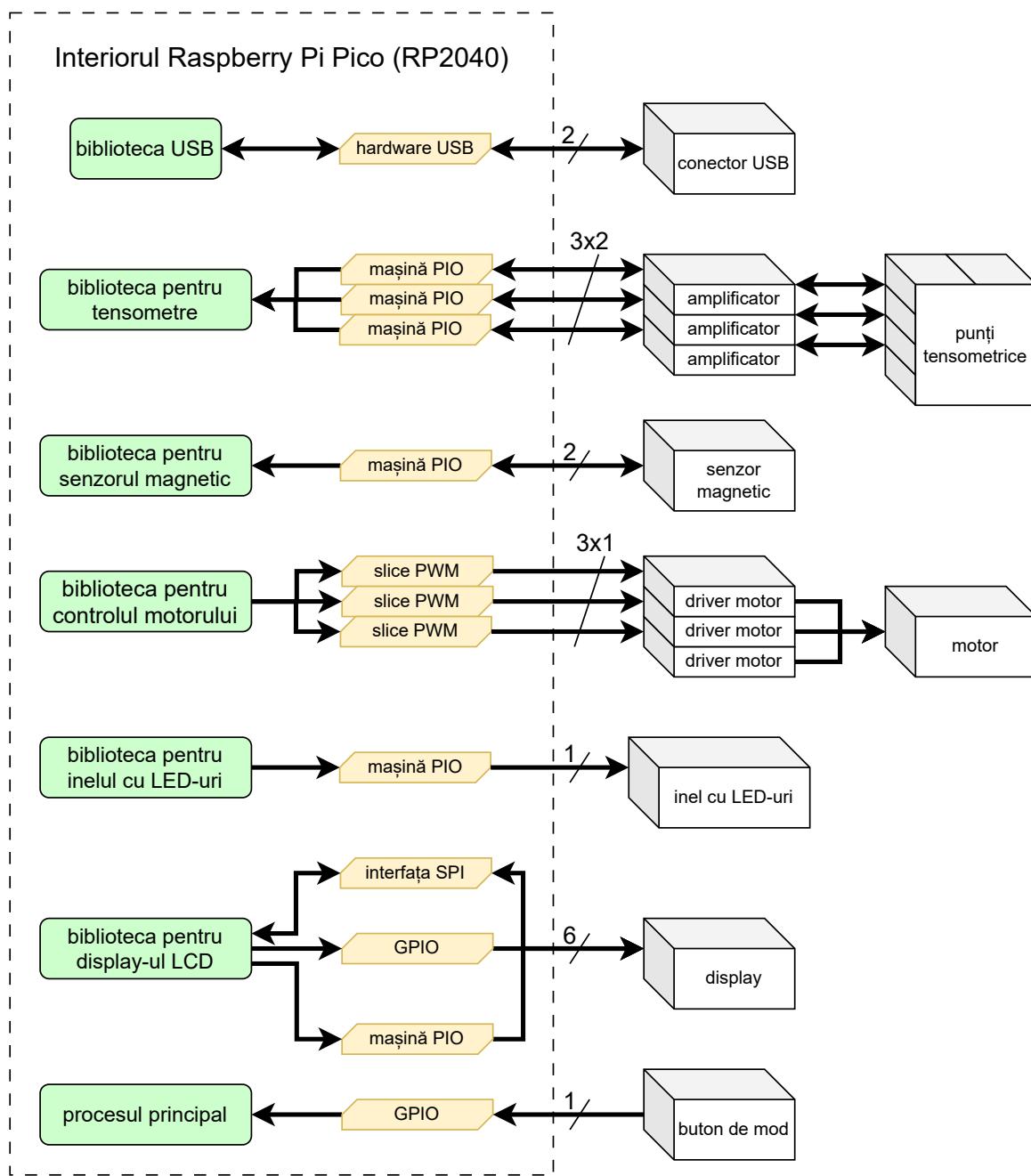


Figura 3.2: Arhitectura hardware a sistemului
(diagramă generată cu draw.io)

Din punct de vedere hardware, fiecare bibliotecă va folosi exclusiv anumite resurse din cadrul microcontrolerului RP2040 (reprezentate cu galben în figură) pentru a își îndeplini scopurile. Am menționat deasupra fiecărei conexiuni între RaspberryPi Pico și exteriorul de către pini a fost nevoie.

Se poate observa că de multe ori s-a ales utilizarea mașinilor PIO în detrimentul altor interfețe obișnuite (SPI) datorită vitezelor crescute ale acestora.

3.3 ARHITECTURA COMBINATĂ A SISTEMULUI

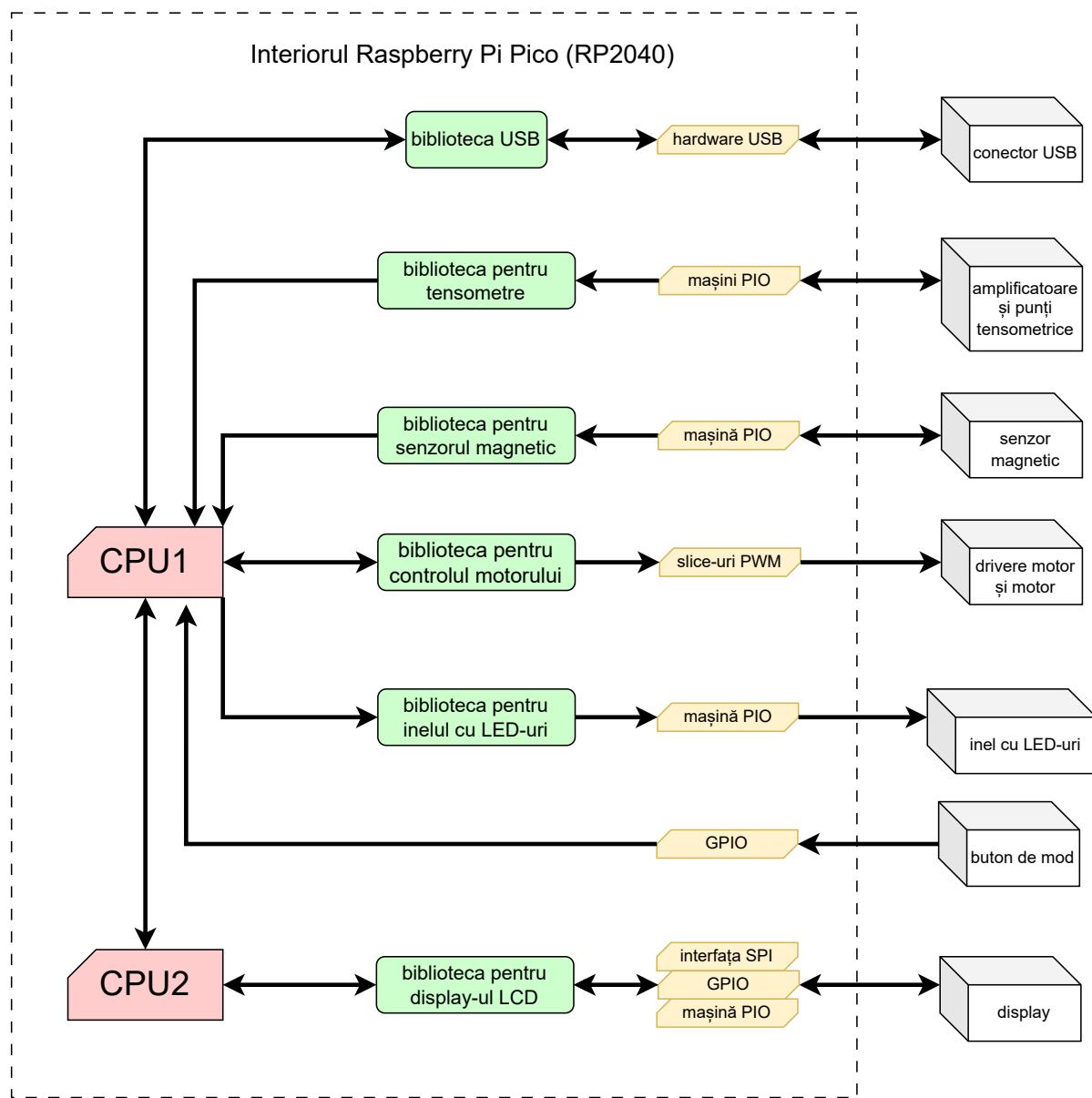


Figura 3.3: Arhitectura combinată a sistemului
(diagramă generată cu draw.io)

4. IMPLEMENTAREA PĂRTII MECANICE

4.1 ANSAMBLUL MOTOR

4.1.1 CONSTRUCTIA MOTORULUI

Motorul necesită anumite caracteristici speciale. Din acest motiv a trebuit să îl construiesc de la zero. Pentru a îl putea construi am luat în considerare următoarele:

- motorul trebuie să fie fixat în partea de jos de bază
- motorul trebuie să aibă rotorul în exterior (partea mobilă a butonului)
- motorul trebuie să permită măsurarea unghiului rotorului
- display-ul trebuie să fie fix și amplasat deasupra motorului
- display-ul necesită cabluri pentru alimentare și date
- motorul trebuie să aibă o mișcare lină, fără cogging

Pentru a rezolva primele două probleme, am ales structura outrunner pentru motor. Cu alte cuvinte, statorul va fi prins de cadru, iar rotorul va avea un ax central ce va intra în mijlocul statorului. În plus, doi rulmenți vor fi folosiți pentru a face legătura între stator și rotor.



Figura 4.1: Statorul



Figura 4.2: Rotorul

Pentru a putea avea display-ul fixat deasupra motorului, a trebuit să aleg un ax gol (în formă de tub) pentru rotor, astfel încât prin axul rotorului să poată trece un ax fix, de care să fie prins display-ul.

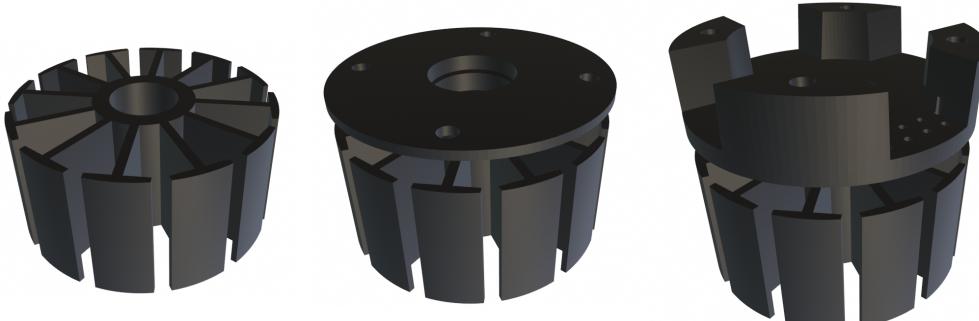


Figura 4.3: Statorul cu elementele de fixare (vedere de jos)

Elementele de fixare vor ajuta la prinderea statorului de bază folosind șuruburi.

După introducerea rotorului în stator se observă că axul rotorului depășește strictul necesar pentru a trece prin cel de-al doilea rulment (vezi Figura 4.4).

Pentru a putea măsura unghiul rotorului se va lipi un magnet de axul rotorului. Detalii despre orientarea polilor magnetului vor fi date în secțiunea *Implementarea măsurării unghiului* (vezi Figura 4.5).

Axul pe care va fi prins display-ul este lipit de un cadru și se poate observa în Figura 4.6

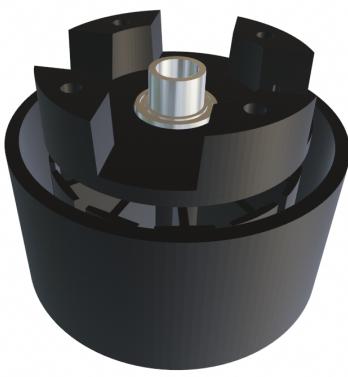


Figura 4.4: Rotorul a fost introdus în stator

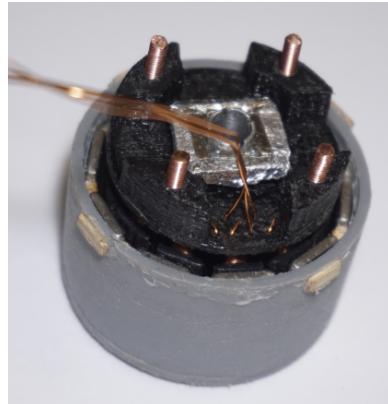


Figura 4.5:
Un magnet a fost
atașat rotorului

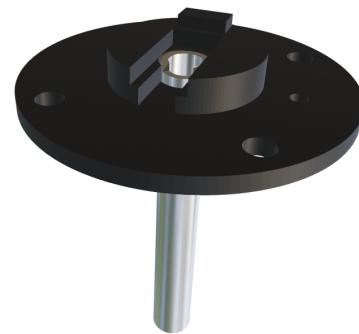


Figura 4.6:
Axul pentru display și suportul acestuia

Se va introduce axul display-ului prin axul rotorului și se va prinde de stator cu piulițe. Apoi se atașează display-ul LCD în partea de sus a motorului. În final se lipesc senzorul magnetic în partea de jos a motorului.

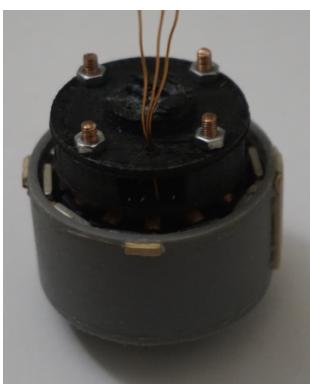


Figura 4.7:
Axul display-ului a fost
introdus în motor



Figura 4.8:
Ansamblul văzut
de sus fără display



Figura 4.9:
Ansamblul display
cu senzor

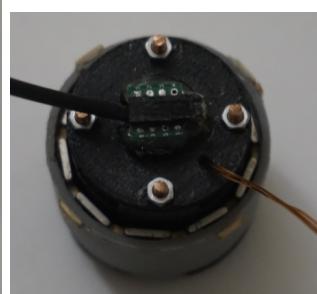


Figura 4.10:
Senzorul a fost lipit
în partea de jos

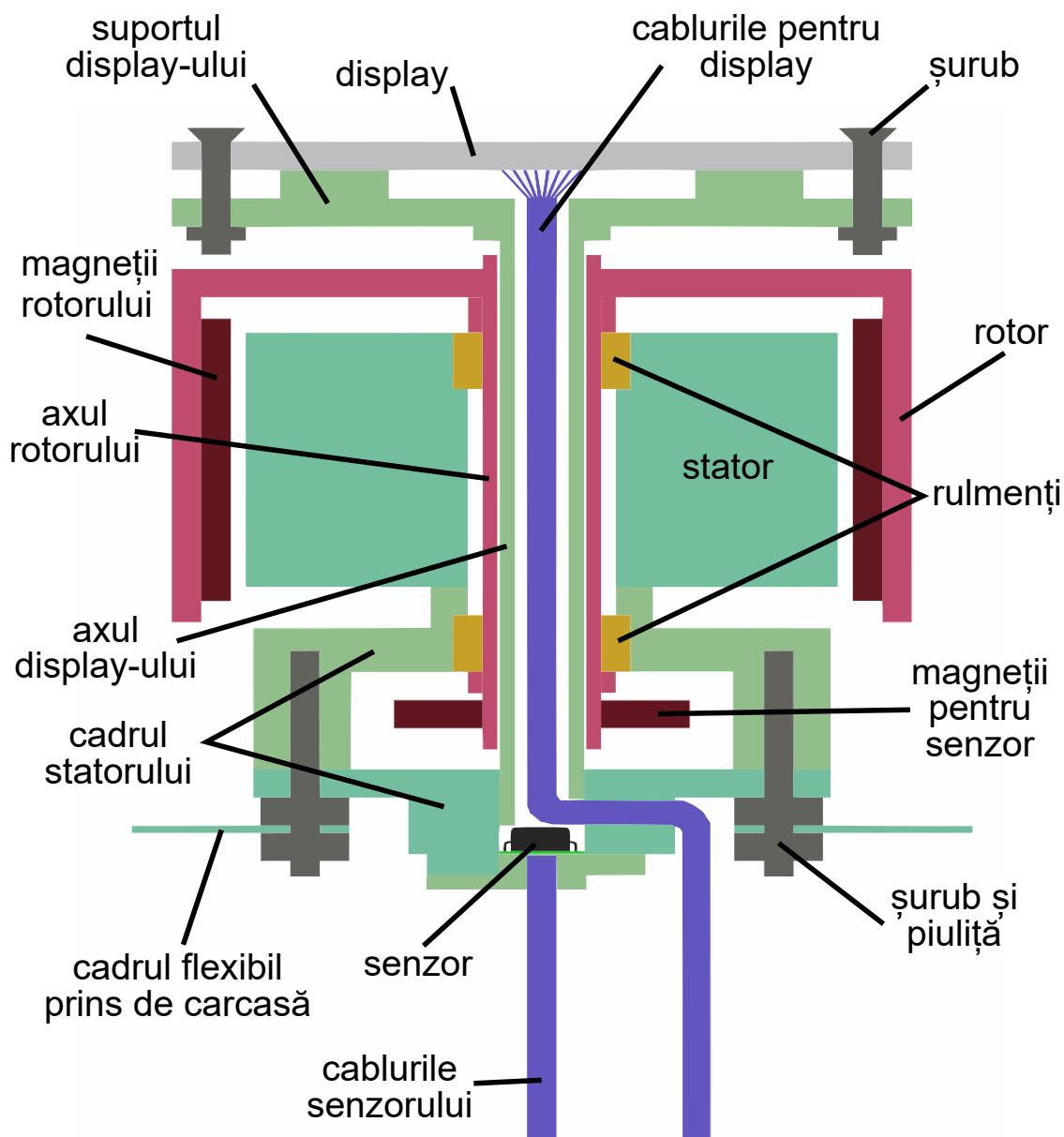


Figura 4.11: Secțiune transversală a motorului

În figură am reprezentat cu nuanțe de verde-albastru părțile care sunt fixe, iar cu nuanțe de roșu părțile mobile. Rulmenții sunt reprezentați cu galben.

Se poate observa că senzorul magnetic este amplasat central sub magnetii pentru măsurare. Totuși, între magneti și senzor vor trece cablurile display-ului.

4.1.2 SELECTIA NUMĂRULUI DE POLI ȘI DINTI

Pentru a avea un randament maxim, numărul de poli și cel de dinți trebuie ales cu atenție. Pentru o simplitate a realizării practice a înfășurărilor, am decis să nu folosesc înfășurări distribuite, ci concentrate.

După cum am descris în secțiunea teoretică *Motorul BLDC*, vom elmina diverse combinații nefavorabile de poli și dinți. Aceste calcule au fost realizate deja în [12], rezultând următorul tabel. Tabelul reprezintă factorul înfășurare pentru combinațiile care respectă regulile amintite. Prin urmare, o valoare mare este favorabilă.

Tabelul 4.1: Combinăriile favorabile de dinți și poli (sursa: [12])

Ns	Nm											
	2	4	6	8	10	12	14	16	18	20	22	
3	NoSym	NoSym	UnBal	q < 0.25	q < 0.25	UnBal	q < 0.25	q < 0.25	UnBal	q < 0.25	q < 0.25	
6	q > 0.5	0.866	UnBal	0.866	q < 0.25	UnBal	q < 0.25	q < 0.25	UnBal	q < 0.25	q < 0.25	
9	q > 0.5	q > 0.5	0.866	NoSym	NoSym	0.866	q < 0.25	q < 0.25	UnBal	q < 0.25	q < 0.25	
12	q > 0.5	q > 0.5	UnBal	0.866	0.933	UnBal	0.933	0.866	UnBal	q < 0.25	q < 0.25	
15	q > 0.5	q > 0.5	UnBal	q > 0.5	0.866	UnBal	NoSym	NoSym	UnBal	0.866	q < 0.25	
18	q > 0.5	q > 0.5	q > 0.5	q > 0.5	q > 0.5	0.866	0.902	0.945	UnBal	0.945	0.902	
21	q > 0.5	q > 0.5	UnBal	q > 0.5	q > 0.5	UnBal	0.866	NoSym	UnBal	NoSym	NoSym	
24	q > 0.5	q > 0.5	UnBal	q > 0.5	q > 0.5	UnBal	q > 0.5	0.866	UnBal	0.933	0.949	
27	q > 0.5	q > 0.5	q > 0.5	q > 0.5	q > 0.5	q > 0.5	q > 0.5	q > 0.5	0.866	NoSym	NoSym	
30	q > 0.5	q > 0.5	UnBal	q > 0.5	q > 0.5	UnBal	q > 0.5	q > 0.5	UnBal	0.866	0.874	

Am ales combinația de 12 dinți pe stator și 10 poli magnetici pe rotor. Următorul pas este determinarea felului în care trebuie realizate înfășurările. În acest scop am folosit un tool online [11].

O dată ce am realizat înfășurările conform figurii generate este necesară analizarea modului de interconectare a bobinajelor. Practic s-au realizat 3 înfășurări (câte una pentru fiecare fază). În acest moment putem simplifica din punct de vedere conceptual motorul înapoi la cel mai simplu motor BLDC: cel cu 3 dinți și 2 poli.

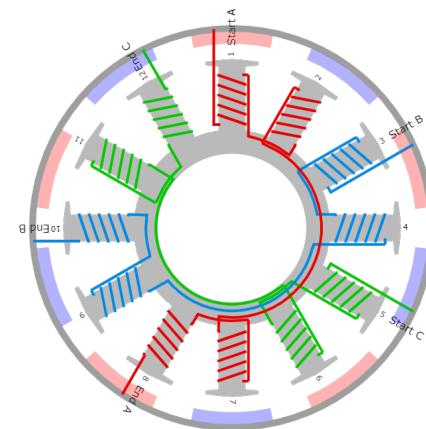


Figura 4.12: Orientarea și amplasarea înfășurărilor (sursa: [11])

Ce mai rămâne de făcut este determinarea numărului de spire pentru fiecare înfășurare și modul de interconectare al înfășurărilor (delta sau Y).

4.1.3 DETERMINAREA CARACTERISTICILOR ÎNFĂȘURĂRILOR

Atât diametrul firului emailat din care se realizează înfășurarea, cât și numărul de spire vor influența direct rezistența finală a bobinajului. Rezistența va influența curentul prin bobină. Curentul maxim este stabilit de:

- curentul maxim permis de interfața USB
- curentul maxim al driverului de motor
- capacitatea maximă de curent a firului emailat (pentru a preveni supraîncălzirea)

Conform standardului USB 3.0 (conectorul de tip C e conform cu standardul 3.0), curentul maxim admis este de 0.9A. Curentul maxim admis de drivere este de 1.6A, iar firul ales poate suporta continuu un curent de aproximativ 0.55A. Asta înseamnă că două bobine ce funcționează în paralel pot suporta 1.1A.

Acum trebuie analizat și felul în care voi actiona motorul. Mai concret, nu va fi mereu o singură bobină conectată între alimentare (5V) și masă. Pentru o precizie maximă a controlului am decis să alimentez motorul într-un fel sincron, echilibrat, care înseamnă un curent constant pentru o forță generată constantă și o simetrie în activarea bobinelor. Practic, tensiunile de alimentare a fiecărei faze este defazată cu $\frac{2\pi}{3}$ (120°) una față de celalătă. Mai multe detalii se găsesc în secțiunea *Controlul hardware al motorului*.

Curentul maxim prin motor ar fi cel când una din faze va fi conectată la 5V, iar celelalte două la GND. Rezistența internă a driverelor de motor este de aproximativ $R_d = 0.4\Omega$. Dacă se consideră rezistența unei înfășurări ca fiind R , atunci curentul maxim ar fi:

$$I_{total} = \frac{U}{R_d + \frac{R+R_d}{2}} = 2 * \frac{U}{R + 3R_d} \quad (4.1)$$

Se dorește alegerea unei valori ale rezistenței astfel încât să nu treacă peste limitele de mai sus, dar care să permită, dacă este cazul, curenți mai mari pentru scurte durate de timp.

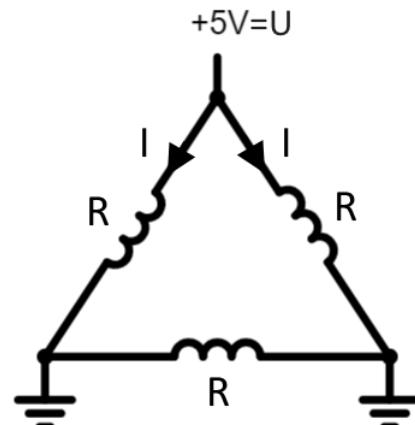


Figura 4.13: Curentul este maxim în înfășurări

Am ales în final ca $R = 6\Omega$. Astfel curentul maxim ar fi în jur de 1.38A. În funcționarea normală, curentul va fi limitat din software (motorul nu va fi mereu folosit la putere maximă). Neînțând cont de tipul conexiunii (stea sau delta), se poate afirma că rezistența fază-fază trebuie să fie de $R_{ff} = R_{stea} = R_{delta} = 4\Omega$.

Am avut la dispoziție fir emailat de un singur diametru, anume de 0.2mm. Din acesta am realizat o înfășurare completă a dinților unei faze. Astfel am obținut o rezistență de aproximativ 15Ω și un număr de $N_t = 240$ de spire pe fiecare dintă. Având în vedere că această valoare este foarte departe de cea dorită am încercat diverse moduri de realizare a înfășurărilor și de conectare a lor. Dacă rezistența unei înfășurări este R_0 , atunci în funcție de conectarea lor în manieră delta sau stea putem obține rezistență fază-fază:

$$R_{stea} = 2 * R_0, R_{delta} = \frac{2 * R_0}{3} \quad (4.2)$$

În acest caz $R_{stea} = 30\Omega$ și $R_{delta} = 10\Omega$. Niciuna din aceste variante nu este favorabilă. Se poate încerca reducerea numărului de spire și conectarea delta pentru a ajunge la valoarea dorită. $R_{delta} = 4\Omega$, deci $R_0 = 6\Omega$. Se aplică regula de 3 simplă și ajungem la numărul de înfășurări $N_0 = 96$. Aceasta ar fi prima variantă posibilă.

Putem încerca realizarea a două înfășurări și a le lega în paralel pentru a forma armătura de bobinaj a unei faze. Fiecare înfășurare va avea 120 de spire și va avea 7.5Ω . Conectându-le în paralel vom obține $R_0 = 3.75\Omega$. Conexiunea stea va oferi $R_{stea} = 7.5\Omega$ și $R_{delta} = 2.5\Omega$. Se poate alege conexiunea stea și reducerea numărului de spire de la 120 la $N_1 = 64$;

O ultimă variantă ar fi realizarea unei conexiuni serie-paralel între 3 înfășurări posibil de valori diferite. Se va realiza o înfășurare de N_s spire care se va insera cu conexiunea paralelă a două înfășurări, fiecare de N_p spire. Fie k o constantă de proporționalitate astfel încât $R = kN$ pentru orice înfășurare. Vom avea deci:

$$N_s + 2N_p = N_t, \quad R_s + \frac{R_p}{2} = R_0, \quad R_s = kN_s, \quad R_p = kN_p, \quad 15\Omega = k * 240 \quad (4.3)$$

Rezistențele posibile ale înfășurării rezultate, R_0 , vor fi astfel în intervalul $[3.75\Omega, 15\Omega]$. O conexiune stea ar permite valori ale rezistenței fază-fază de $[7.5\Omega, 30\Omega]$, ceea ce nu convine, iar o conexiune delta de $[2.5\Omega, 10\Omega]$ care convine. Dorim $R_{delta} = 4\Omega$, deci $R_0 = 6\Omega$. Vom rezolva sistemul de ecuații 4.3. Rezultă că $N_s = 48$ și $N_p = 96$.

Cele 3 cazuri de analizat pentru realizarea bobinajului sunt prin urmare:

- o înfășurare de 96 de spire (per fază), $R_0 = 6\Omega$, conexiune delta
- două înfășurări de 64 de spire ($R = 4\Omega$) legate în paralel (per fază), $R_0 = 2\Omega$, conexiune stea
- o conexiune serie-paralel (per fază) constând dintr-o înfășurare de 48 de spire ($R = 3\Omega$) în serie cu două în paralel de 96 de spire ($R = 6\Omega$), rezultă $R_0 = 6\Omega$, conexiune delta

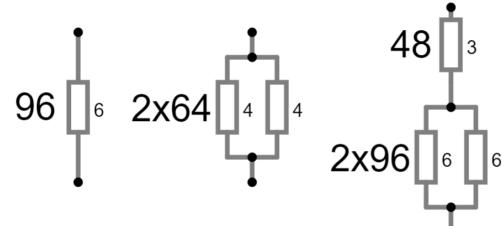


Figura 4.14: Cele 3 posibilități de realizare a înfășurărilor
Textul mare arată numărul de înfășurări, iar cel mic rezistențele înfășurărilor.

Acum trebuie ales între cele 3 variante. Rezistența fază-fază și, deci, curentul sunt stabilite prin calculele de mai sus. Se dorește ca motorul să dezvolte cuplu(forță) maxim. Se cunoaște că forța generată de o bobină într-un câmp magnetic este direct proporțională cu numărul de spire și curentul prin această, adică $F \propto N * I$. Se va calcula forța pentru fiecare caz. În acest scop putem considera că se aplică o tensiune fixă între 2 din cele 3 terminale ale motorului.

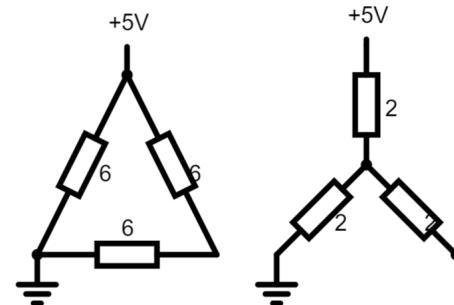


Figura 4.15: Determinarea curentilor în cele 3 cazuri

Pentru primul caz, prin una din înfășurări va trece $I_1 = \frac{U}{6}$, iar prin celelalte două $I_2 = \frac{I_1}{2}$. Forța totală va fi:

$$F \propto I_1 * 96 + 2 * I_2 * 96, \quad F \propto U * 32 \quad (4.4)$$

Pentru al doilea caz, prin una din înfășurări (fază) va trece $I = \frac{U}{2+2}$. Avem 2 înfășurări, fiecare formate din 2 sub-înfășurări de 64 de spire prin care va trece jumătate din curent. Forța totală va fi:

$$F \propto 2 * 2 * \frac{I}{2} * 64, \quad F \propto U * 32 \quad (4.5)$$

Pentru ultimul caz, prin una din înfășurări (fază) va trece $I = \frac{U}{6}$, iar prin celelalte două $I_2 = \frac{U}{6*2}$. Fiecare înfășurare are o sub-înfășurare de 48 de spire prin care trece întreg curentul, iar două sub-înfășurări de 96 de spire prin care va trece jumătate din curent. Forța totală va fi:

$$F \propto I * 48 + 2 * \frac{I}{2} * 96 + 2 * (\frac{I}{2} * 48 + 2 * \frac{I}{4} * 96), \quad F \propto U * 48 \quad (4.6)$$

Concluzia este că ultimul caz este cel mai favorabil. Prin urmare caracteristicile înfășurărilor sunt: o conexiune serie-paralel (per fază) constând dintr-o înfășurare de 48 de spire ($R = 3\Omega$) în serie cu două în paralel de 96 de spire ($R = 6\Omega$), rezultă $R_0 = 6\Omega$, conexiune delta, astfel încât rezistența fază-fază este $R_{ff} = 4\Omega$. Inductanța măsurată fază-fază este de $200\mu H$.

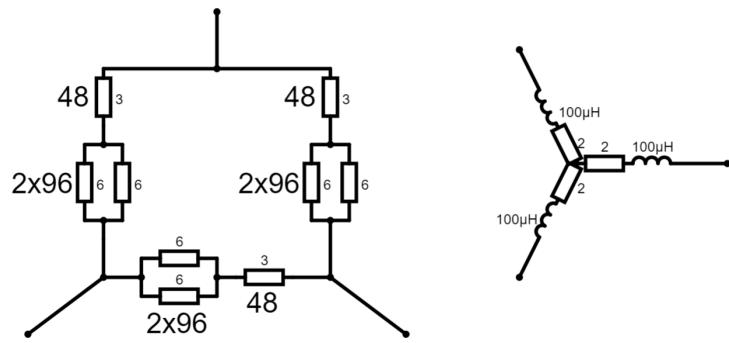


Figura 4.16: Schema finală a înfășurărilor și o simplificare echivalentă (ce ține cont de inductanță) a motorului pentru calcule viitoare

Ultima decizie de luat este selecția materialului și a formei statorului. Pentru a elibera orice posibilitate de cogging am decis să realizez statorul drept și din plastic.

4.2 CADRUL CU PUNTILE TENSOMETRICE

Pentru a putea măsura înclinarea butonului, trebuie ca motorul să fie prins de un cadru flexibil pe care să fie amplasate puncte tensometrice. Forma cadrului trebuie să:

- permită lipirea tensometrelor în punctele de îndoire
- permită montarea motorului
- fie prinsă de carcasă
- permită scoaterea cablurilor necesare de la motor, display și senzor magnetic

Pentru o stabilitate și rigiditate optimă am ales un cadru cu 4 "brațe". În centru se va prinde motorul folosind 4 piulițe, iar capetele fiecărui braț se vor fixa de carcasă folosind 4 șuruburi.

În funcție de înclinarea butonului, fiecare braț va fi îndoit în diferite feluri. În figura următoare se poate vedea cum unele părți ale cadrului devin convexe (marcate cu roșu), iar altele concave (marcate cu albastru)

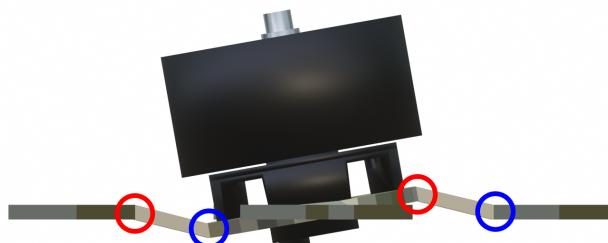


Figura 4.17: Îndoirea brațelor în unul din cazurile de înclinare

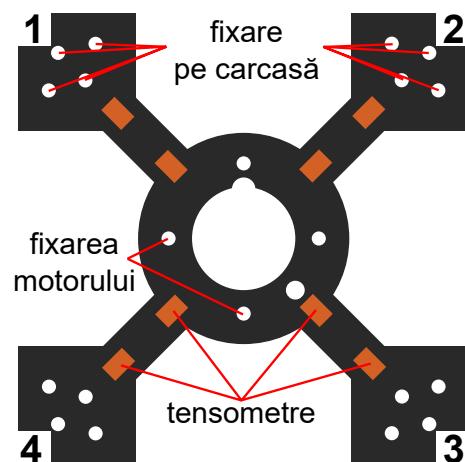


Figura 4.18: Forma cadrului metalic

Văzute de sus, cazurile de înclinare sunt următoarele:

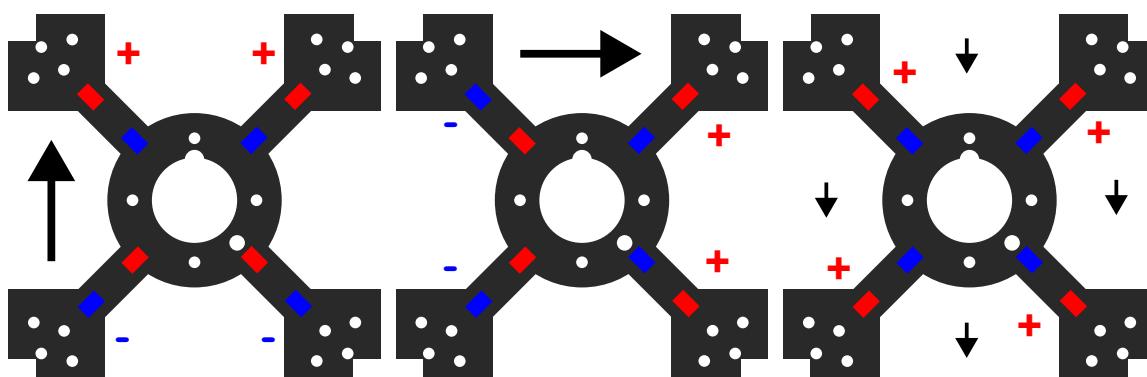


Figura 4.19: Deformarea cadrului văzută de sus în diferite cazuri

Se observă că mereu, pe un braț, deformarea celor două tensometre este opusă. Ca urmare, putem considera un braț ca fiind deformat în sens pozitiv sau negativ. O altă urmare este faptul că putem conecta câte două tensometre pentru a forma 4 jumătăți de puncte tensometrice.

Pentru a măsura deformările pe cele 3 axe, avem următoarele:

- Înclinarea pe axa X se obține prin diferența între brațele (2 și 3) sau (1 și 4)
- Înclinarea pe axa Y se obține prin diferența între brațele (2 și 1) sau (3 și 4)
- Apăsarea se obține prin suma oricărora brațe

Având la dispoziție 3 amplificatoare, avem următoarele opțiuni:

1. se măsoară 3 brațe individual (față de o referință formată din 2 rezistoare egale) și se calculează diferențele în software.
2. se folosește un amplificator pentru a măsura diferența între brațul 1 și 2, și altul pentru diferența între 2 și 3. Ultimul amplificator măsoară brațul 2 individual.
3. un amplificator măsoară diferența între brațul 1 și 2, altul diferența între 2 și 3 și ultimul diferența între 2 și 4. Puntea brațului 4 va fi conectată invers pentru a calcula, de fapt, suma între canalul 2 și 4 la ultimul amplificator.

În practică am încercat varianta 2 și 3. Este recomandabil să nu se folosească rezistoarele pentru referință, deoarece acestea nu sunt precise, iar valoarea lor variază cu temperatura, producând valori eronate.

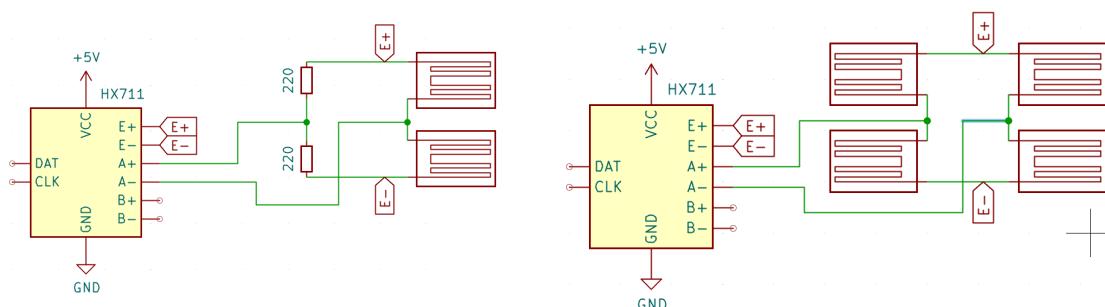


Figura 4.20: Măsurarea unui singur braț față de o referință

Figura 4.21: Măsurarea diferenței între două brațe

4.3 CARCASA BUTONULUI

Baza carcasei are o formă paralelipipedică. Înălțimea acesteia trebuie să fie suficient de mare pentru elementele electronice (Raspberry Pi Pico, drivere de motor, etc.).

În interiorul bazei se va fixa și inelul cu LED-uri. Deasupra inelului se lipeste hârtie laminată pentru a ajuta la difuzia luminii și a îmbunătăți aspectul vizual.

Peste rotorul motorului se aşază capacul butonului. Acesta are formă de cilindru gol pe dinăuntru și are o bucată de sticlă de ceas în partea de sus pentru a permite vizualizarea ecranului LCD. Dimensiunile finale sunt de 10cm x 10cm x 8cm.

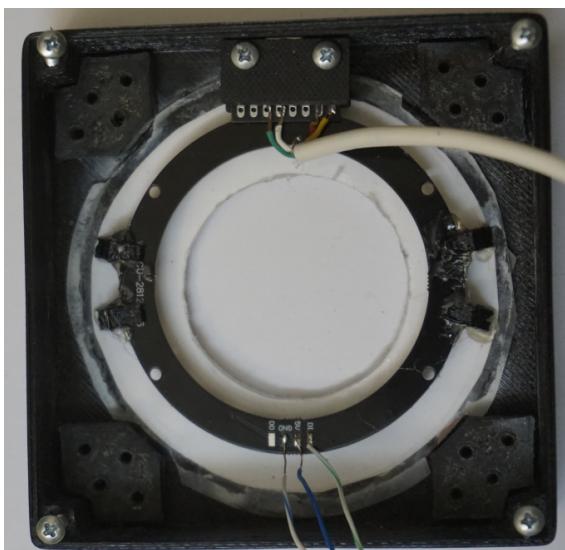


Figura 4.22: Partea superioară a carcasei

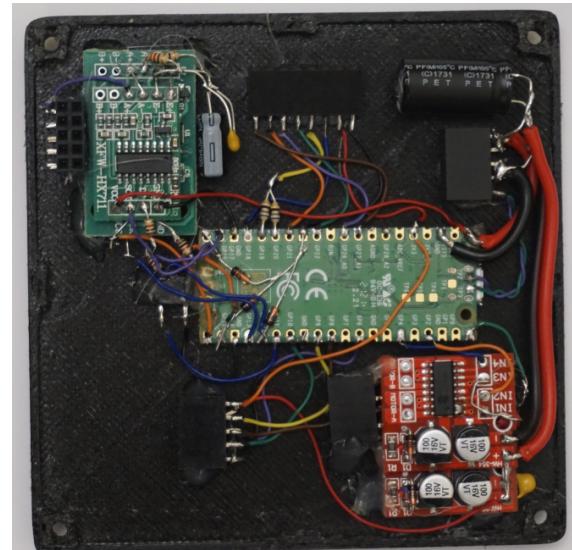


Figura 4.23: Partea inferioară a carcasei și componentele electronice



Figura 4.24: SmartKnob complet asamblat

5. IMPLEMENTAREA HARDWARE

În acest capitol voi prezenta atât conectarea electrică a dispozitivelor, cât și alegerea perifericelor folosite pentru comunicare. În cazul alegerii perifericului PIO, se prezintă și codul pentru acesta.

5.1 CONECTORUL USB

Conectorul USB asigură atât alimentarea, cât și comunicarea între SmartKnob și PC. S-a folosit un USB-C breakout board, de pe care s-au folosit pinii de alimentare: 5V și GND și cei de date: D+ și D-. Legăturile s-au făcut identic cu cele ale portului USB integrat pe placa Pico. Pinul 5V va fi conectat la VBUS, iar pentru D+ și D- se utilizează padurile marcate TP2 și TP3 de sub conectorul USB integrat.

Din punct de vedere electric, comunicarea USB se desfășoară pe o pereche diferențială. Totuși, datorită vitezei relativ reduse (12Mbps), nu este necesar un cablu special între Raspberry Pi și conector.

Curentul consumat de dispozitiv se va încadra în limitele impuse de standard. Pentru detalii se poate vedea Secțiunea 4.1.3. Motorul este cel mai mare consumator, iar celelalte dispozitive pot fi considerate neglijabile din punct de vedere al consumului.

Pentru a schimba modul de funcționare și pentru a intra în modul de programare s-a conectat un buton între unul din pinii de pe Raspberry (GPIO11) și GND.



Figura 5.1: Conectorul USB

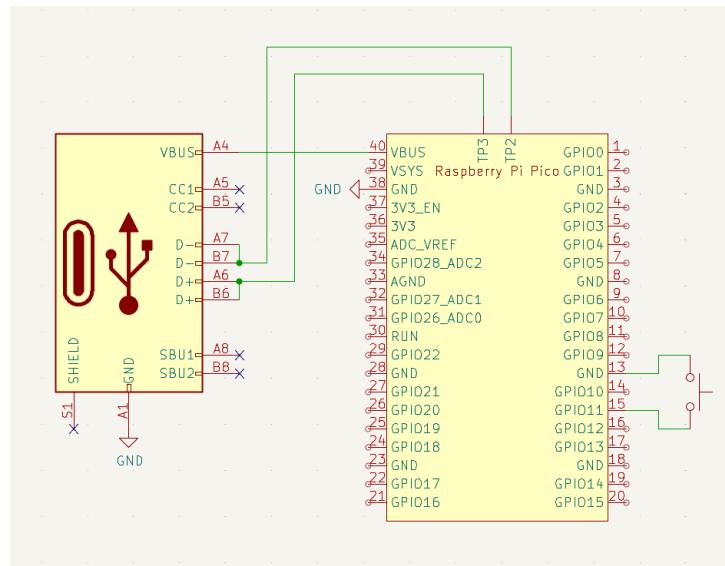


Figura 5.2: Schema de conectare a conectorului USB

RP2040 implementează un periferic USB dedicat care va fi folosit pentru a comunica prin USB. Considerând complexitatea protocolului și disponibilitatea *bibliotecii TinyUSB*, nu se pune problema unei alte metode hardware de comunicare USB.

5.2 SENZORUL MAGNETIC

Senzorul magnetic are o multitudine de interfețe disponibile comparate în următorul tabel.

Tabelul 5.1: Caracteristicile interfețelor senzorului MT6701

interfață	număr de linii	rezoluție imp/rotație	viteză de achiziție	hardware necesar
ABZ	2 sau 3	1024	virtual nelimitată	întreruperi
UVW	3	96	virtual nelimitată	întreruperi
analog	1	4096	500kHz	convertor analog-numeric
PWM	1	4096	1kHz	timer/slice PWM
I2C	2	16384	13kHz	periferic I2C
SSI/SPI	3	16384	1MHz	periferic SPI sau PIO

Pentru o precizie optimă am optat pentru o interfață digitală. În plus, am dorit ca viteza de achiziție să fie cât mai mare. Rezoluția trebuie să fie suficientă pentru a permite controlul fin al motorului. Dacă este posibil, se preferă o implicare minimă a procesorului în această achiziție.

Astfel am ales ultima variantă, anume interfața **SSI/SPI**. Această interfață este compatibilă cu interfața obișnuită SPI. O particularitate este aceea că folosește doar 3 linii (nu există date de la microcontroler către senzor).

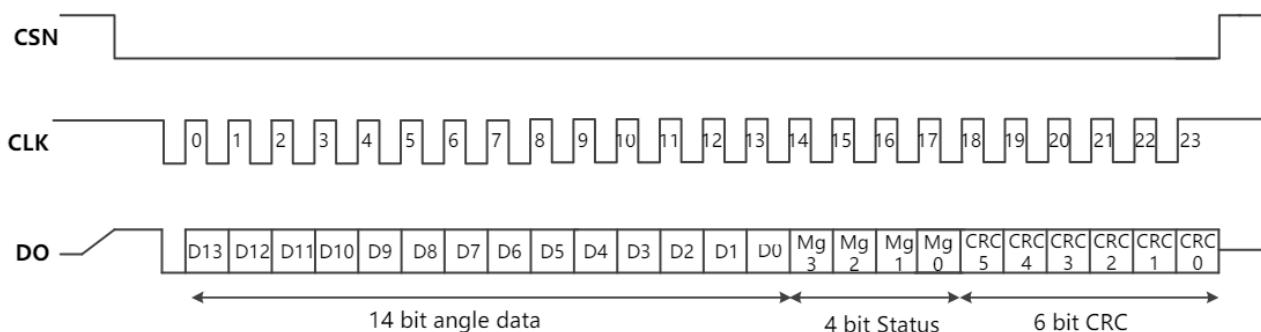


Figura 5.3: Comunicarea SSI cu senzorul MT6701 (sursa: [15, p. 24])

Prima idee ar fi să se utilizeze perifericul SPI al RP2040. Totuși avem o variantă mai bună. Se dorește un transfer de date care să implice procesorul cât mai puțin. Se pot folosi canale de DMA pentru transferul de date. Problema este că modulul SPI trebuie să trimită date (chiar dacă aceste date nu vor ajunge fizic niciunde) pentru a putea prelua date. Asta înseamnă că trebuie să avem canale DMA irosite pentru a transmite date inutile prin interfața SPI.

Astfel am ales utilizarea unei mașini PIO. Mașina PIO permite citirea fără transmisie. În plus, aceasta rulează în continuu, furnizând date către DMA. Dezavantajul este necesitatea scrierii unui program în limbajul de asamblare specific PIO. Mașina PIO poate utiliza orice pini. Am ales conectarea senzorului la pinii interfeței SPI, întrucât aceștia pot fi utilizati și de mașina PIO.

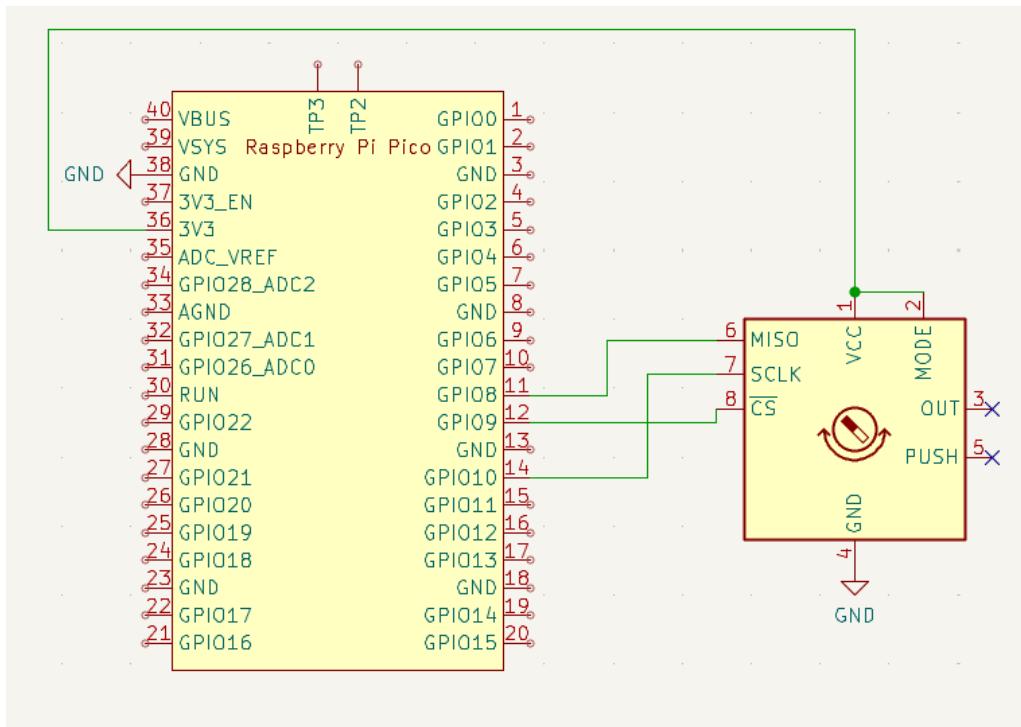


Figura 5.4: Conectarea senzorului MT6701

Acum rămâne de programat perifericul PIO. Codul este prezentat mai jos. Registrul y reține numărul de biți de citit minus 1, adică 13. CS este controlat de instrucțiunea *set*, iar CLK de *side*. Datele de intrare se citesc folosind instrucțiunea *in*. Datele din registrul de deplasare sunt transmise automat către FIFO-ul mașinii. Mașina va fi setată să ruleze la o sesime din frecvența sistemului, rezultând $20MHz$.

```
1 .wrap_target
2     set pins, 0 [2]      side 1 ;activez CS, CLK=1, delay conform
3         documentatiei
4     mov x, y            side 0 ;copiez numarul de biti-1, CLK=0
5     nop                 side 1 ;CLK=1
6 bitloop:
7     in pins, 1          side 0 ;citesc un bit in registrul de deplasare,
8         CLK=0
9     jmp x-- bitloop    side 1 ;repet de x ori, CLK=1
10    set pins, 1         side 1 ;in final dezactivez CS
11 .wrap
```

Fragmentul 5.1: Codul masinii PIO pentru senzorul MT6701

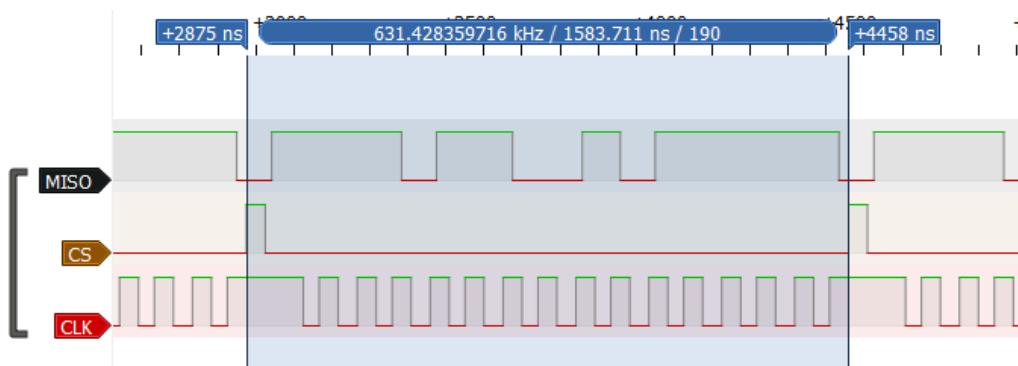


Figura 5.5: Formele de undă măsurate pentru comunicarea cu MT6701

Astfel am obținut o comunicare digitală cu o viteză de tact de $10MHz$. Viteza de achiziție (frecvența de citire a unghiului) de peste $600kHz$. Această viteză este satisfăcătoare în vederea controlului motorului.

5.3 MOTORUL

Motorul trebuie controlat lin și silentios. Pentru un control precis și lin, am ales o schemă de control similară cu cea a unui motor trifazat sincron, frecvența de control putând să ajungă la 0. Pentru a simula nivelele de tensiune analogice necesare controlului se folosesc modulația PWM, iar pentru a face întreg procesul silentios se alege o frecvență PWM peste frecvențele perceptibile de urechea umană.

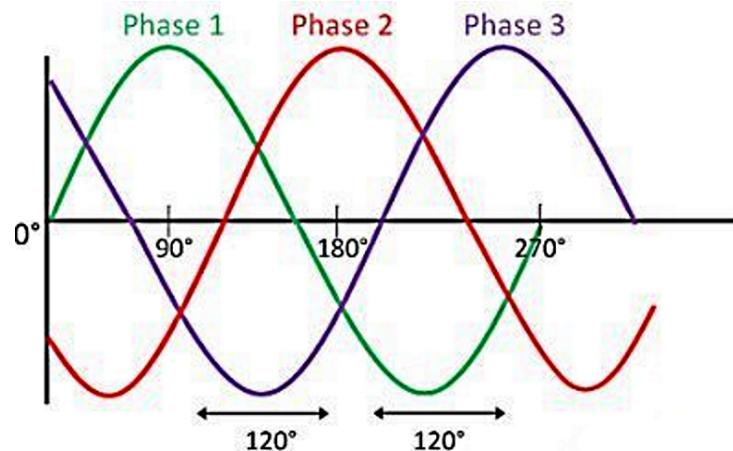


Figura 5.6: Tensiunile la cele 3 borne ale motorului, dacă faza variază constant în timp (sursa: Vertiv)

Putem considera că avem 3 canale de ieșire analogică pentru controlul motorului. Tensiunile vor respecta mereu următoarea regulă:

$$U_1 = 2.5 + 2.5 * \sin(\varphi), U_2 = 2.5 + 2.5 * \sin(\varphi + \frac{2\pi}{3}), U_3 = 2.5 + 2.5 * \sin(\varphi - \frac{2\pi}{3}) \quad (5.1)$$

Pentru a aplica forță (cuplu), va trebui să calculăm φ pe baza valorii curente ale unghiului. Dacă am lăsa φ constant în timp, motorul ar tinde către o poziție fixă. Mai exact am forma 5 cogging steps. Acest fapt se datorează faptului că rotorul are 5 perechi de poli. Practic, din 72° în 72° de grade se repetă comportamentul rotorului. La oricare din aceste poziții forța este 0 (rotorul a fost atras spre poziție). Dar și exact între două astfel de poziții forța va fi 0 (rotorul este atras egal de cele două poziții din apropiere). Deci, forța este maximă la un sfert din distanța între două poziții de cogging, adică la $\frac{72^\circ}{4} = 18^\circ$. Cu alte cuvinte, pentru

a aplica forța maximă trebuie să adunăm 18° la poziția curentă a rotorului.

Din punct de vedere practic, se vor aproxima semnalele analogice prin trei semnale PWM trecute prin drivele de motor. Inductanța bobinelor motorului face ca intensitatea curentului prin acestea să fie netezită și să ajungă aproximativ la valoarea dorită și setată prin PWM.

Pentru a alege frecvența semnalului PWM avem restricția $F > 20kHz$ pentru a nu putea fi auzită. Pentru a avea o rezoluție satisfăcătoare a controlului motorului, am ales o valoare a counter-ului PWM de 1000. Asta înseamnă că vom avea 1000 de nivele de tensiune care vor putea fi simulate. Divizorul de tact pentru modulele PWM folosite este setat pe 4. Prin urmare, frecvența PWM este de $F_{PWM} = 31.25kHz$.

Formulele de calcul pentru factorii de umplere D_i sunt:

$$D_i = \frac{U_i}{U_{alim}} \approx \frac{U_i}{5V} \quad (5.2)$$

Se pune problema sincronizării canalelor PWM: un decalaj între canale poate elimina fenomenele de aliere. Vom analiza cazul sincronizat și cel nesincronizat. Fie $\varphi = \frac{pi}{2}$. Deci $U_1 = 5V, U_2 = 1.25V, U_3 = 1.25V$ sau $D_1 = 1, D_2 = 0.25, D_3 = 0.25$. Se va reprezenta o perioadă a semnalului.

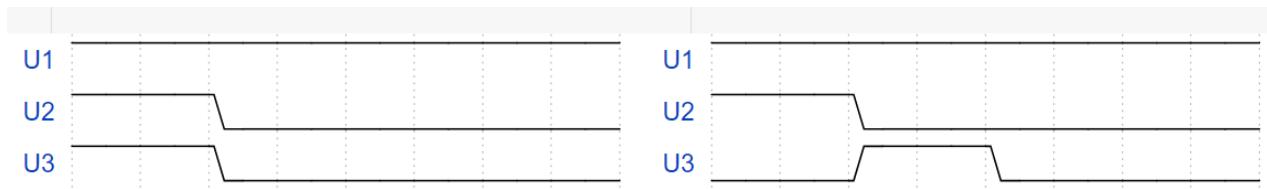


Figura 5.7: Comparație între cazul sincronizat (stânga) și cel nesincronizat (dreapta)

Din punct de vedere al forțelor cele două cazuri sunt identice. Pentru a demonstra voi enumera bobinele activate și durata de activare. În orice moment de timp vor fi active fie 0, fie 2 bobine:

- În cazul sincronizat: $0.75 * (B_{12} + B_{13})$
- În cazul nesincronizat: $0.25 * (B_{13} + B_{23}) + 0.25 * (B_{12} + B_{32}) + 0.5 * (B_{12} + B_{13}) = 0.75 * (B_{12} + B_{13})$

Totuși, din punct de vedere al curentului consumat, cele două cazuri nu sunt identice.

- În cazul sincronizat: $0.75 * (I_{12} + I_{13}) = 1.5 * I$
- În cazul nesincronizat: $0.25 * (I_{13} + I_{23}) + 0.25 * (I_{12} + I_{32}) + 0.5 * (I_{12} + I_{13}) = 2 * I$

Explicația este că, în cazul nesincronizat, vor exista doi timpi în care se consumă curent, dar forțele rezultante se anulează (cel puțin parțial). Altfel spus, se consumă curent pentru a aplica forțe a căror rezultantă este zero. Prin urmare, voi menține cele 3 canale sincronizate.

Pentru a genera semnalul PWM final (de 0-5V) trebuie folosite driverele de motor. Un modul cu driverul MX1616 are 2 canale, iar fiecare canal are două intrări și două ieșiri. Ar fi ideală folosirea unui canal MX1616 pentru două semnale PWM. Totuși, se observă că un canal nu poate genera toate 4 combinațiile de ieșiri (lipsește combinația HH la ieșire) conform cu Tabelul 2.4.

Din această cauză vom folosi un canal MX1616 pentru un canal PWM. Se va conecta intrarea **IN1/INA** la nivelul de tensiune superior. Astfel intrarea **IN2/INB** este inversată și amplificată la ieșire. Pe Raspberry toți pinii au astfel de capacitați, dar implementarea curentă necesită ca pinii să nu facă parte din PWM slices comune(nu pot fi consecutivi).

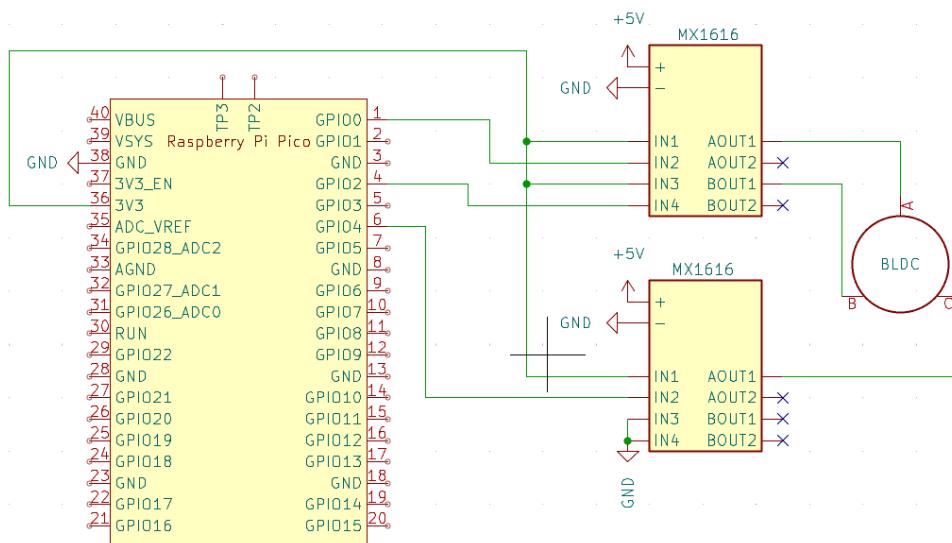


Figura 5.8: Conectarea driverelor de motor și a motorului

Pentru a reduce interferențele generate de motor, s-au adăugat un condensator electrolytic de 1mF și unul MLCC de 470nF în paralel cu alimentarea. Modulele MX1616 vor fi alimentate de la 5V. Pentru a putea funcționa în siguranță cu nivelul de tensiune de 3.3V al Raspberry, s-a adăugat o diodă Zener de 3.3V fiecărui modul pentru a coborî nivelele de tensiune de comandă la 3.3V .

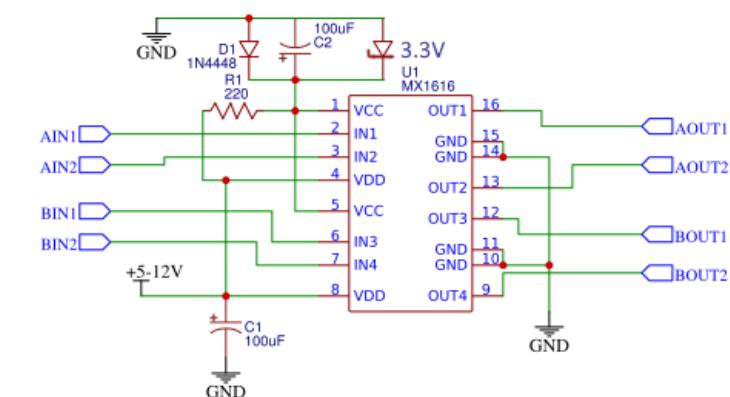


Figura 5.9: Schema modulului MX1616 cu dioda Zener adăugată (sursa: EasyEDA)

5.4 ECRANUL LCD

În cazul ecranului LCD putem vorbi despre 3 elemente importante ale comunicării/interfațării:

- controlul retroiluminării
- comunicarea de viteză mică (setări, configurare)
- comunicarea de mare viteză (imagină proprie-zisă)

Pentru a controla luminozitatea ecranului se folosește un slice PWM. Modulul cu display conține tranzistorul ce controlează direct LED-ul de fundal. Se poate folosi orice pin, însă toti pinii sunt capabili de PWM.

Pentru comunicarea simplă cu modulul se vor folosi doi pini GPIO pentru RST și DC. În rest, datele vor fi transmise folosind interfața SPI și bitbanging. Aici, interfața are particularitatea că există o singură linie pentru date bidirectionale. Pentru a preveni scurtcircuitul în cazul unei programări greșite s-au adăugat două rezistoare de valori mici, astfel încât să nu afecteze datele care circulă la frecvențe mari.

Pentru comunicarea de mare viteză nu am putut folosi perifericul SPI integrat în RP2040, deoarece acesta adaugă mereu un timp în care linia de selecție este inactivată, apoi activată din nou. Dacă transmit câte 16 biți o dată, se irosesc minim 1.5 perioade de tact. Această pierdere este de aproximativ 10%.

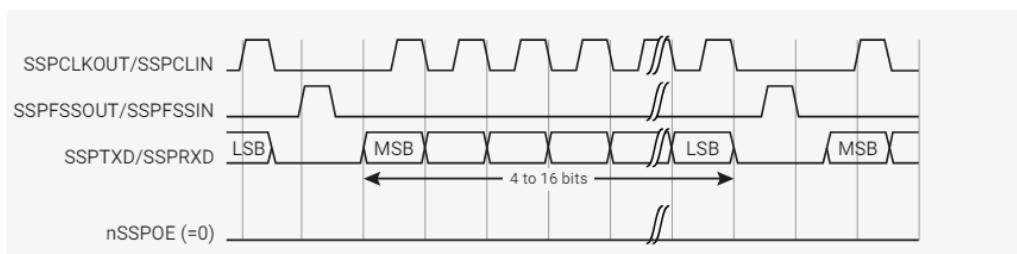


Figura 5.10: Semnalele generate de perifericul SPI

Din această cauză am ales ca transferul de date să se desfășoare utilizând o mașină PIO. Frecvența maximă a mașinii PIO este identică cu cea a perifericului SPI, dar nu se va mai insera dezactivarea semnalului de selecție. Se vor transmite comenzi folosind SPI bitbanging, iar când trebuie transmise datele imaginii, se va cupla mașina PIO. Tot ce trebuie să facă această mașină este să preia date și să le serializeze la viteză maximă.

```
1 .wrap_target
2     out PINS, 1      side 0;
3     nop               side 1;
4 .wrap
```

Fragmentul 5.2: Codul mașinii PIO pentru ecranul LCD

5.5 INELUL CU LED-URI

Inelul cu LED-uri este alcătuit din 16 LED-uri inteligente WS2812 conectate în serie. Inelul necesită alimentare la 5V pentru ca toate culorile să funcționeze corect. Pentru a comunica se va folosi un singur pin de date de la microcontroler către primul LED din inel. Deși nivelul de tensiune al pinului de date este de 5V, am observat că modulul funcționează corect și direct de la nivelul de 3.3V al Raspberry Pi Pico, fără a mai necesita translație de nivel logic. Acest fapt îmi permite conectarea directă a inelului la Raspberry Pi Pico.



Figura 5.11: Inelul cu LED-uri

Pentru a comunica va trebui să generez semnale speciale conform Secțiunea 2.14. În acest scop aș avea ca opțiuni folosirea bitbang-ului sau a unei mașini PIO. Pentru a crește eficiența comunicării am ales mașina PIO. Ideea comunicării este că va trebui să trimit vectorul ce conține culorile LED-urilor (în format binar), iar mașina PIO le va transforma în semnale corespunzătoare pentru controlul LED-urilor. Unicul pin de comunicare este controlat de instrucțiunea *side*.

Codul a fost preluat din *exemplile pentru Raspberry Pi Pico*, dar a fost modificat pentru a respecta mai precis timpii dați în documentația LED-urilor. Am constatat empiric faptul că LED-urile au o toleranță mare la variația duratelor impulsurilor.

```
1 .define public T0H 7
2 .define public DELTA 5
3 .define public T1L 11
4 .wrap_target
5 bitloop:
6     out x, 1      side 0 [T1L - 1] ; Side-set still takes place when
           instruction stalls
7     jmp !x do_zero side 1 [T0H - 1] ; Branch on the bit we shifted out.
           Positive pulse
8 do_one:
9     jmp bitloop   side 1 [DELTA - 1] ; Continue driving high, for a
           long pulse
10 do_zero:
11     nop          side 0 [DELTA - 1] ; Or drive low, for a short pulse
12 .wrap
```

Fragmențul 5.3: Codul mașinii PIO pentru inelul cu LED-uri

În primul rând, când nu sunt transmise date, codul rămâne în așteptare pe prima instrucțiune. Astfel pinul de comunicare va rămâne pe 0 logic, generându-se astfel comanda de resetare/initializare a comunicării. Apoi, atâtă timp cât avem date, câte un bit va fi codificat și prezentat la ieșire. Pentru a respecta timpii, mașina PIO va rula la $18.4MHz$. Fiecare bit are perioada de 23 de cicluri ai mașinii PIO (fie se execută instrucțiunile de la liniile (6, 7 și 9), fie (6, 7 și 11), rezultând într-un bitrate de $800kbps$, maximul recomandat de documentație.

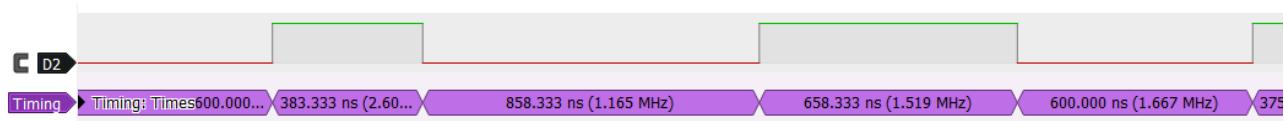


Figura 5.12: Forma de undă măsurată, generată de Raspberry Pi

Se observă că timpii sunt corecți, în limitele acceptabile ($\pm 150ns$ față de valorile menționate în Secțiunea 2.14). Pentru a ne asigura că și timpul de resetare este corect, software-ul va trebui să facă pauze de aproximativ 1ms între transmisii. 1ms este suficient timp pentru transmiterea întregului vector de culori și pentru timpul de resetare. Astfel putem obține o viteză de împrospătare de $1kHz$, care este mai mult ca suficient pentru inelul cu LED-uri.

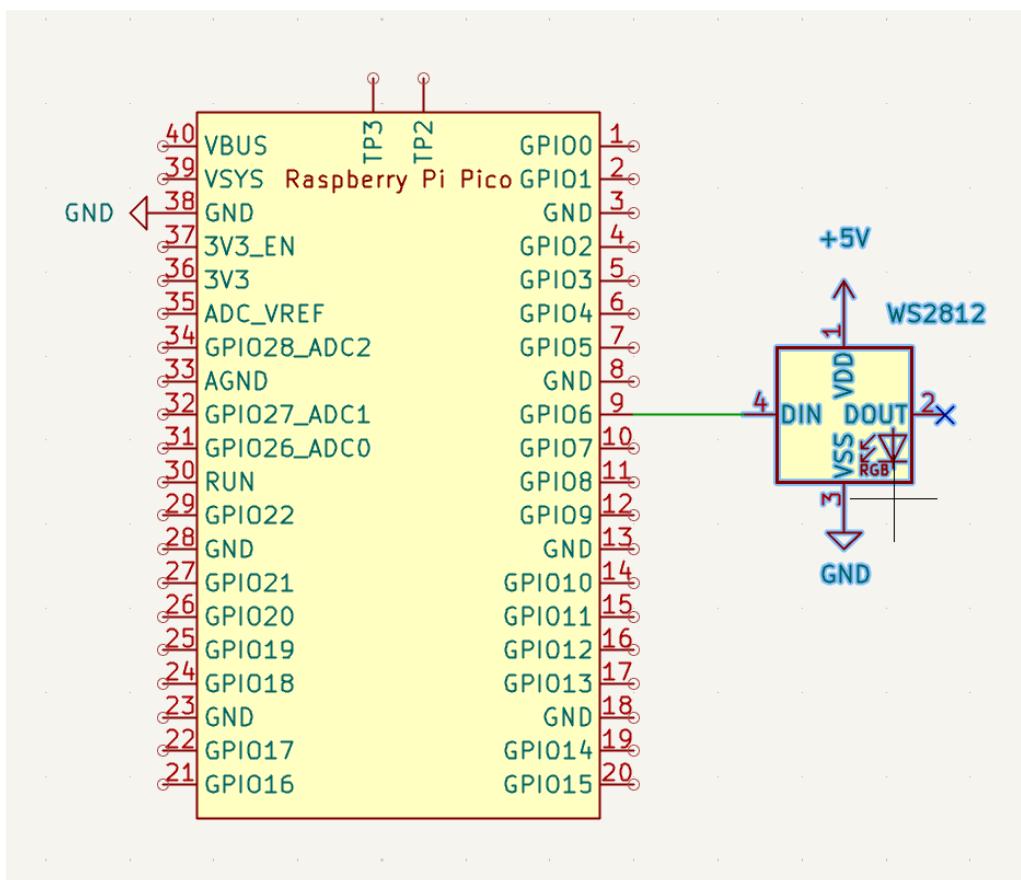


Figura 5.13: Conecțarea electrică a inelului cu LED-uri

5.6 AMPLIFICATOARELE ȘI PUNTILE TENSOMETRICE

Pentru a putea măsura 3 valori independente este nevoie de (cel puțin) 3 canale independente de măsurare. Viteza de achiziție poate fi $80Hz$ sau $10Hz$. Plăcuțele sunt setate prin hardware la frecvența de achiziție de $10Hz$. A trebuit să îñtrerup un trace pe PCB și să lipesc pinul RATE la alimentare.

Fiecare HX711 are două canale. Totuși, conform documentației [17, p. 3], comutarea între cele două canale durează $50ms$. Astfel s-ar limita frecvența de achiziție la $20Hz$, ceea ce nu este satisfăcător pentru o reacție rapidă. Prin urmare, am fost nevoit să folosesc 3 module cu HX711. Modulele necesită alimentarea la $5V$. Pentru a proteja RP2040, am folosit un rezistor și o diodă Zener între pinul de date și pinul corespunzător de pe RP2040.

Pentru a citi date de la module, se folosește o interfață SPI modificată. Pentru a putea comunica eficient am decis să folosesc 3 mașini PIO. Pinul de CLK este controlat de instrucțiunea *side*, iar pinul de date este folosit de instrucțiunea *wait* și citit de instrucțiunea *in*. Mașina va fi rulată la $4MHz$, rezultând într-o comunicare serială cu tactul de $2MHz$.

```

1 .define public NBITS 24
2
3 .wrap_target
4     wait 0 pin 0      side 0
5     set x, (NBITS-1)  side 1
6 bitloop:
7     in pins, 1        side 0
8     jmp x-- bitloop   side 1
9 .wrap

```

Fragmentul 5.4: Codul mașinii PIO pentru modulele cu HX711

La început se așteaptă ca pinul de date să revină la 0, ceea ce semnifică finalizarea conversiei. Apoi, se vor transmite 25 de cicluri de tact pentru a citi cei 24 de biți de date. La finalizarea citirii, datele sunt încărcate automat în FIFO-ul mașinii PIO. Cele 25 de cicluri de tact, mai exact numărul de cicluri peste 24, selectează și canalul folosit. În cazul de față se alege canalul A, cu o amplificare de 128.

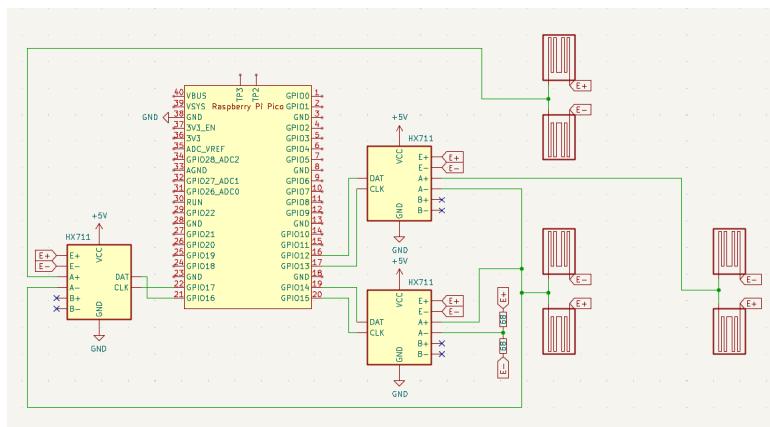


Figura 5.14: Conecțarea electrică a amplificatoarelor HX711

6. IMPLEMENTAREA SOFTWARE

În acest capitol voi prezenta implementarea bibliotecilor pentru diversele module, respectiv configurarea bibliotecii tinyUSB.

6.1 BIBLIOTECA USB - TINYUSB

Configurarea bibliotecii tinyUSB se face în mai mulți pași:

- alegerea clasei/claselor implementate de dispozitiv
- configurarea/generarea descriptorilor necesari
- modificarea funcțiilor de callback pentru a primi și a transmite date relevante

ALEGEREA CLASEI DISPOZITIVULUI

Clasele din care se poate alege sunt:

- HID - human interface device
- CDC - communication device class
- MSC - mass storage device class
- MIDI - pentru dispozitive MIDI
- VENDOR - pentru dispozitive cu configurație specială

Am ales clasa **HID** pentru că aceasta implică folosirea unor transfere/endpoint-uri de tip interrupere. Acestea au avantajul unei latențe (mici) garantate, un element important al dispozitivelor de interfață cu utilizatorul. În plus, clasa HID este ușor de folosit cu majoritatea sistemelor de operare, nemaifiind necesară instalarea unor drivere. Am mai folosit și clasa **CDC** pentru a simula un port serial prin care se vor transmite date de debug. Astfel, dispozitivul rezultat este un dispozitiv compus.

CREAREA DESCRIPTORILOR ȘI A RAPOARTELOR HID

Dispozitivul creat va putea funcționa în 3 moduri distincte: mouse, joystick și SmartKnob. Fiecare dintre acestea va necesita descriptori specifici. Descriptorul de dispozitiv este comun tuturor celor 3 moduri de funcționare. Restul descriptorilor au fost creați folosind macro-uri puse la dispozitiv de biblioteca tinyUSB. A trebuit să modific și funcțiile de returnare a descriptorilor, astfel încât să returneze descriptorul corespunzător modului de funcționare.

Pentru rapoartele HID ale modului mouse și joystick am păstrat rapoartele originale furnizate de biblioteca tinyUSB. Modul joystick simulează un joystick cu facilități enumerate în descriptor (3 axe cu rotație, butoane direcționale și 32 de butoane). Mouse-ul simulat are 8 butoane, deplasare pe axele X și Y, o roată de derulare verticală și una orizontală. Modul SmartKnob trimite și primește mereu rapoarte de 64 de bytes pentru simplificarea comunicării. 64 de bytes este maximul posibil pentru un raport HID. În tabel, coloana *Nume* reprezintă numele elementelor din structura de date (în C) pentru raport.

MODIFICAREA FUNCȚIILOR DE CALLBACK

Biblioteca tinyUSB se ocupă de majoritatea sarcinilor ce țin de comunicarea USB. Utilizatorului îi revine, deci, doar sarcina depunerii datelor corespunzătoare în rapoartele HID. În acest scop este pusă la dispoziție spre modificare funcția `static void send_hid_report(uint8_t report_id)`. Aceasta primește ca parametru ID-ul raportului, valoare neutilizată în implementarea curentă. Ea este apelată automat când este necesară trimitera unui raport și nu trebuie apelată în continuu de utilizator. Funcția trebuie să creeze un raport HID și să îl trimită folosind `tud_hid_report(report_id, &report, sizeof(report))`. În funcție de modul curent de funcționare se va crea, completa și trimită raportul corespunzător. Când se trimită un raport către dispozitiv, se apelează automat funcția `void tud_hid_set_report_cb(.....)` care primește printre parametri și un pointer către datele raportului. De aici utilizatorul decide cum se vor prelucra datele.

În cazul joystick-ului și al mouse-ului, rapoartele sunt completeate cu date chiar în funcția `send_hid_report`). Pentru modul SmartKnob, am creat un pointer la funcție (`usb_fill_descriptor_callback`) care trebuie setat în main și care trebuie să pointeze către o funcție care să completeze raportul. Astfel, funcția de completare a raportului în modul SmartKnob poate să fie o funcție de sine stătătoare.

Pentru primirea rapoartelor de ieșire am creat funcția `void handleUsbPacket(uint8_t *buffer, uint8_t buflen)`, care va cîti datele din raportul dat ca parametru și care va fi apelată din `void tud_hid_set_report_cb(.....)`. Ea va lua acțiunile necesare, anume va transmite datele relevante către biblioteca vizată în raport. Toate rapoartele de ieșire au pe prima poziție un câmp ce reprezintă submodulul care se dorește a fi controlat.

Explicarea structurii concrete a rapoartelor se găsește în **Anexa 1**

```
1 void handleUsbPacket(uint8_t *buffer, uint8_t buflen){  
2     if(buffer[0] == MESSAGE_LED){  
3         WS2812_set_ring_array(buffer+1, buffer[1+48]);  
4         ...  
5     }  
6     if(buffer[0] == MESSAGE_MOT){  
7         if(buffer[1])  
8             Motor_vibrate();  
9         if(buffer[2] == MOTOR_VEL)  
10            Motor_set_mode_constant_velocity();  
11         ...  
12     }  
13     ...  
14 }
```

Fragmențul 6.1: Parte din codul pentru primirea rapoartelor

```
1 static void send_hid_report(uint8_t report_id) {
2     if(usb_mode == USB_MOUSE){
3         int16_t Yval = Ytilt, Xval = Xtilt;
4         //prepare x and y values to send to PC
5         ...
6
7         static float prev_knob_angle;
8         float cr_angle = 360.01-knob_angle*360.0/1024/16;
9         float delta = cr_angle - prev_knob_angle;
10        //calculate scroll value to send
11        ...
12
13        hid_mouse_report_t report = {
14            .x    = Xval/10, .y = Yval/10, .wheel = -rx, .pan = 0,
15            .buttons = (Press>PressLimit2?MOUSE_BUTTON_LEFT:0)};
16        tud_hid_report(REPORT_ID_GAMEPAD, &report, sizeof(report));
17        return;
18    }
19    if(usb_mode == USB_SMART){
20        hid_smartknob_report_t report;
21        usb_fill_descriptor_callback(&report);
22        tud_hid_report(0, &report, sizeof(report));
23        return;
24    }
25    if(usb_mode == USB_JOYSTICK){
26        //calculate axes, rotations and buttons
27        ...
28        hid_gamepad_report_t report =
29        {
30            .x    = Xval, .y = Yval, .z = rx, .rz = rx, .rx = rx, .ry = rx,
31            .hat = 0, .buttons = (Press>PressLimit2?GAMEPAD_BUTTON_A:0) | (
32                btn2?GAMEPAD_BUTTON_1:0)
33        };
34        tud_hid_report(REPORT_ID_GAMEPAD, &report, sizeof(report));
35    }
36}
```

Fragmentul 6.2: Parte din codul pentru trimiterea rapoartelor

6.2 BIBLIOTECA MT6701 - SENZORUL MAGNETIC

Biblioteca pentru senzorul magnetic este gândită să ușureze cât mai mult munca procesorului. Această bibliotecă necesită inițializare prin apelul funcției `void MT6701_init(uint32_t* angle)`. Ca parametru i se dă adresa variabilei unde se dorește depunerea valorii citite a unghiului.

După inițializare, biblioteca va citi continuu date de la senzor și le va depune la adresa dată. Pentru a putea beneficia de un transfer continuu fără absolut niciun fel de intervenție a procesorului vor fi folosite două canale DMA, configurate să se pornească unul pe celălalt la finalizarea sa. Un singur canal DMA nu poate fi configurat să facă transferuri continue, ci poate realiza doar un număr dat de transferuri.

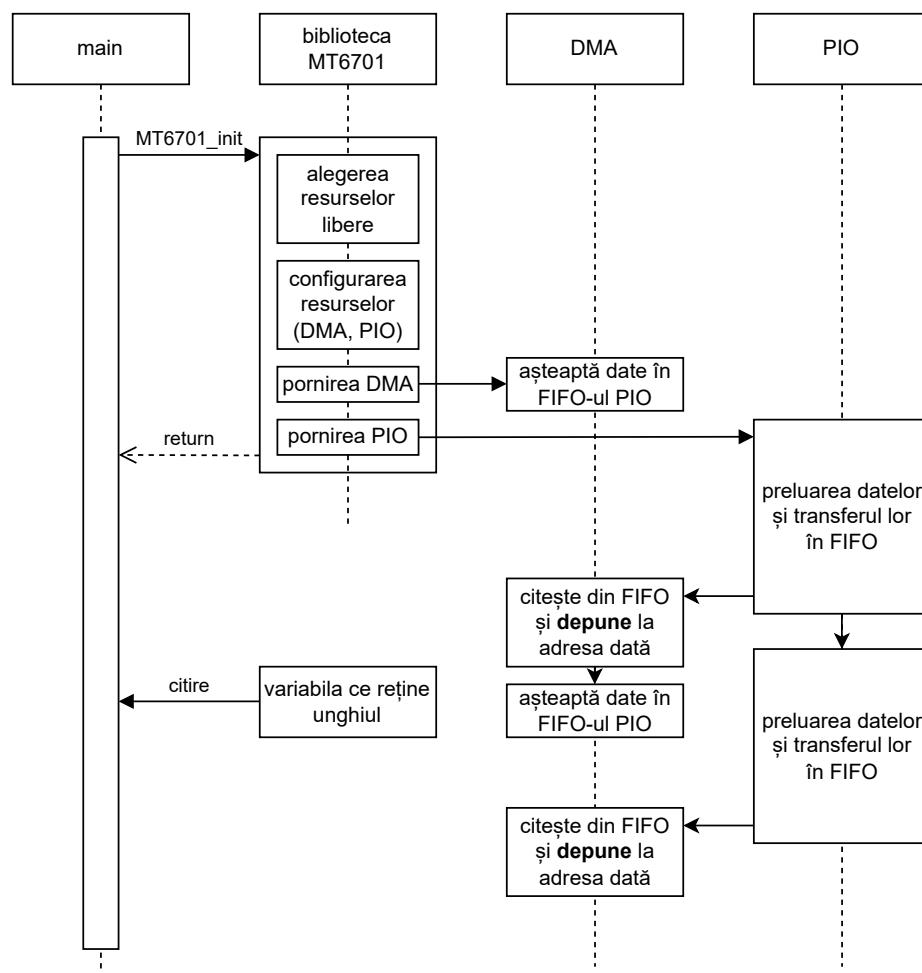


Figura 6.1: Funcționarea bibliotecii MT6701

Valoarea unghiului se poate citi în orice moment. Nu se pune problema consistenței datelor, întrucât datele constau dintr-o singură variabilă pe 32 de biți care este transferată în întregime de către DMA într-un singur ciclu de tact.

6.3 BIBLIOTECA MOTOR

Biblioteca pentru controlul motorului pune la dispoziție 3 tipuri de funcții:

- o funcție de inițializare, `void Motor_init(uint32_t *anglevar)`, căreia îi se va furniza adresa variabilei ce reține unghiul butonului
- o funcție de rulare a calculelor, `void Motor_task()`, care trebuie apelată cel puțin de 1000 de ori pe secundă pentru a menține un control fin
- mai multe funcții pentru a schimba modalitatea de control/feedback a butonului

Pentru a utiliza biblioteca este necesar un singur apel al funcției de inițializare. Apoi trebuie apelată în mod repetat funcția de calcul. Aceasta se poate apela oricără de des, deoarece conține un mecanism prin care calculele efective nu se fac mai rapid de $500\mu s$. Pentru a controla motorul putem fie modifica faza sinusoidelor generate, fie menține faza mereu la maxim în direcția dorită și modifica amplitudinea sinusoidelor. Funcționarea acestei funcții este următoarea:

- se calculează unghiul curent folosind formula $anglefloat = \frac{anglevar*360.0}{2^{14}}$, după care se calculează variația unghiului față de ultimul apel și viteza unghiulară a butonului
- în funcție de modul de funcționare, se vor realiza calculele corespunzătoare
- peste valoarea de comandă calculată se poate adăuga un semnal de vibrație, dacă este cazul
- se calculează valorile finale ale factorilor de umplere pentru semnalele PWM și se trimit către slice-urile PWM.

Considerând că funcția nu face o cantitate mare de calcule, am putut folosi virgula flotantă. Faptul că RP2040 nu are unitate de virgulă flotantă nu va prezenta un dezavantaj semnificativ. Între două apeluri ale funcției de calcul se pot apela funcțiile de setare a modului de funcționare.

În calculele următoare, ph reprezintă unghiul ce se adaugă la unghiul curent al motorului pentru a obține faza φ din formulele din Secțiunea 5.3. La finalul calculelor, se setează faza maximă în direcția dorită și se calculează puterea motorului (amplitudinea sinusoidelor generate) ca fiind raportul între faza calculată și cea maximă.

6.3.1 MODUL CU FRECARE

Modul cu frecare simulează un coeficient de frecare crescut sau scăzut. Cu alte cuvinte, față de rotirea liberă a butonului, rotirea este fie îngreunată, fie usurată. `void Motor_set_mode_friction(int8_t friction)` primește ca parametru intensitatea frecării. O valoare de 3 favorizează maxim rotirea butonului, iar o valoare de -50 o îngreunează maxim. Calculele pentru acest mod au la bază formula:

$$ph = angspeed * frictionvalue \quad (6.1)$$

În plus, la viteze mici ale rotorului am scăzut puterea motorului pentru a preveni vibrarea în loc după ce rotorul ajunge aproximativ în repaus.

6.3.2 MODUL CU VITEZĂ CONSTANTĂ

În acest mod se urmărește menținerea unei viteze constante de rotație a butonului. Viteza nu este configurabilă, ci este fixată la $\frac{540}{sec}$. Pentru a selecta acest mod se apelează `void Motor_set_mode_constant_velocity()`. Am decis ca viteza să fie fixă la această valoare, deoarece nu se poate menține orice viteză constantă cu o precizie satisfăcătoare. La viteze mici, imperfecțiunile realizării manuale a părții mecanice duc la variații semnificative ale vitezei de-a lungul rotației butonului. La viteze mari, motorul nu ar avea suficientă putere pentru a face o reglare precisă.

Se folosește un regulator PD pentru controlul vitezei. Experimental am determinat valorile pentru K_p și K_d , acestea fiind $K_p = -100$ și $K_d = 200$. Viteza unghiulară, *angspeed*, este măsurată în $\frac{\circ}{ms}$.

$$ph = K_p * (angspeed - targetspeed) + K_d * \frac{d \ ang speed}{dt} \quad (6.2)$$

6.3.3 APLICAREA DE VIBRAȚIE

Pentru a face butonul să vibreze se va apela funcția `void Motor_vibrate()`. Un apel va face butonul să vibreze timp de $50ms$ la o frecvență de $100Hz$.

Din punct de vedere al calculelor:

$$ph+ = 9 * \sin\left(\frac{time_{us}}{10000} * 2 * \pi\right); \quad (6.3)$$

6.3.4 SETAREA DETENTELOR

Detentele sunt anumite puncte către care motorul va tinde. Cu alte cuvinte, unghiul rotorului va tinde către una din pozițiile setate prin detente. Există două moduri principale de setare a detentelor:

- detente uniforme; `void Motor_set_mode_detents(int detents)` va forma detents detente uniform distanțate
- detente customize; `void Motor_set_mode_custom_detents(int nDetents, float *detents)` va seta detentele date ca parametru. Acestea trebuie să fie sortate crescător

În mod normal, când se apelează una din aceste funcții, se consideră că poziția actuală a rotorului este 0 (rotorul va tinde către detenta cea mai apropiată de 0). Pentru a modifica acest comportament, s-au pus la dispozitie două funcții similare cu cele de mai sus, dar care primesc ca parametru unghiul la care trebuie considerat ca fiind rotorul: `void Motor_set_mode_detents_offset(int detents, float offset)` și `void Motor_set_mode_custom_detents_offset(uint8_t nDetents, float *detents, float offset)`.

În plus, pentru a adăuga puncte finale (capete/limite de rotație) se poate apela funcția `void Motor_add_endstops_to_mode(float emin, float emax)`. În cazul detentelor customizate, nu mai este necesar apelul acestei funcții. Primul și ultimul element al vectorului dat se consideră puncte finale.

Pentru a calcula forța de aplicat, mai întâi se determină detenta către care se va aplica forța. Practic se caută detenta fată de care rotorul este cel mai apropiat în momentul curent. Pentru a simula acțiunea elastică a unui resort, forța va fi proporțională cu distanța rotorului față de detenta selectată. Forța va fi și invers proporțională cu distanța între două detente. Toate variabilele din formula de mai jos sunt date în grade.

$$ph = -\frac{(\text{angle_full_rot_offset} - \text{detent_positions}[\text{target}]) * 100}{\text{dist_between_detents}} \quad (6.4)$$

În exemplul următor s-au configurat 3 detente uniforme. Linile albastre reprezintă aceste detente. Liniile punctate verzi arată limitele de influență ale detentelor. Cu roșu este reprezentat rotorul și poziția sa. În cazul de față rotorul este în vecinătatea detentei de la 0 grade (verticală), deci forța se va aplica spre aceasta.

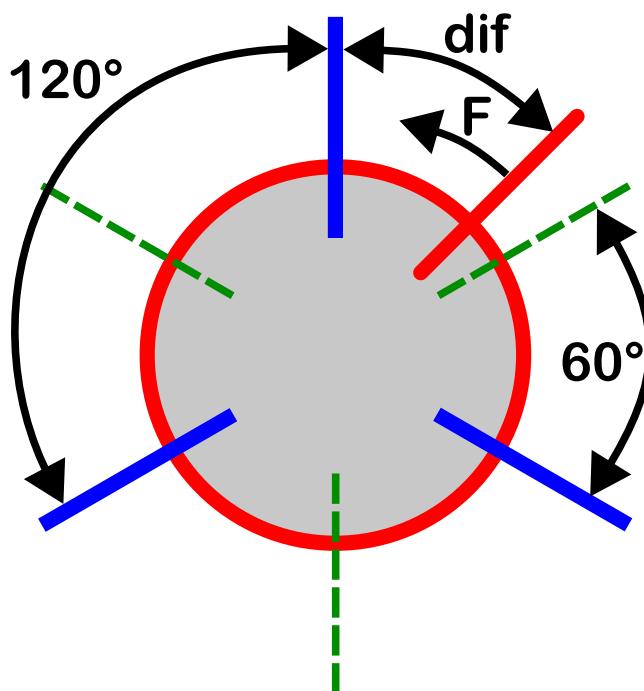


Figura 6.2: Configurație cu 3 detente echidistante

6.4 BIBLIOTECA CG9A01 - ECRANUL LCD

Biblioteca îndeplinește două funcții, anume cea de control al ecranului și cea de randare. Datorită faptului că randarea este un proces care necesită multe calcule, am decis ca biblioteca să fie rulată pe cel de-al doilea procesor al sistemului.

Partea de control a bibliotecii conține următoarele funcții:

- void GC9A01_Initial() conține rutina de inițializare a display-ului
- void LCD_SetPos(u32 Xstart, u32 Ystart, u32 Xend, u32t Yend) setează poziția curentă a indicatorului de memorie al displayului
- void write(uint8_t data) transmite parametrul dat prin SPI (DC neafectat)
- void Write_Cmd(uint8_t data) transmite comanda dată ca parametru (DC=0)
- void Write_Cmd_Data(uint8_t data) transmite data către display (DC=1)

Funcția GC9A01_Initial se va apela o singură dată la pornirea dispozitivului, iar LCD_SetPos (0, 0, W-1, H-1) trebuie apelată înainte de transmisia datelor de imagine către display. Parametrii reprezintă indecsii minimi și maximi pe axa orizontală și verticală între care se vor depozita datele. Mereu se va transfera întreaga imagine, deci indecsii vor fi de la 0 la ultimul pixel al ecranului pe fiecare axă. Codul funcțiilor de mai sus a fost preluat și modificat din exemplele de pe site-ul magazinului BuyDisplay.

Pentru a transmite datele de imagine s-ar putea folosi funcția void write(uint8_t data), dar o eficiență mult mai mare se poate obține folosind o mașină PIO, după cum am descris în Secțiunea 5.4. Mașina PIO necesită transferul a câte unui pixel o dată.

Urmează alegerea rezoluției, a numărului de biți per pixel, și a numărului de frame buffere:

- rezoluția maximă a ecranului este de 240x240 de pixeli. În software aș putea încerca reducerea acesteia la 120x120 (adică să dublez fiecare pixel). În practică aceasta s-a dovedit a fi neplăcută din punct de vedere vizual, mai ales că nu dispun de o modalitate de anti-aliasing. Prin urmare am ales ca rezoluția maximă să fie folosită și pentru randare (și pentru frame buffere)
- numărul de culori pe care display-ul îl pune la dispoziție prin interfața SPI este de 12, 16 sau 18. Pentru o simplă manipulare a datelor pixelilor am ales pixeli pe 16 biți. Astfel un pixel se va transmite într-un singur transfer pe 16 biți și poate fi reținut într-o singură variabilă pe 16 biți fără a irosi spațiu. Numărul de culori reprezentabile este astfel de 65K.
- numărul de frame buffere prezintă o alegere mai dificilă și va fi prezentată în cele ce urmează

Un frame buffer reprezintă zona de memorie în care se va face randarea și care va fi trimisă, apoi, către display. Prima idee ar fi folosirea unui singur frame buffer. Această alegere prezintă însă probleme. Dacă nu vom sincroniza finalizarea trimiterii bufferului cu începerea randării, se vor suprascrie zone din buffer în timp ce acestea sunt transmise, producând imagini eronate. Solutia ar fi ca în timp ce se desfășoară transferul bufferului, procesorul să nu înceapă randarea noului cadru. Asta înseamnă, deci, timp de procesor irosit.

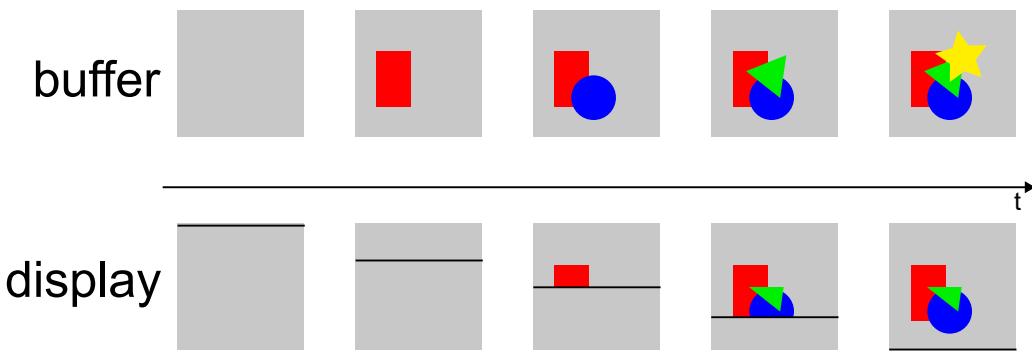


Figura 6.3: Randarea și transmisia nu sunt sincronizate. Unele elemente apar parțial, altele deloc

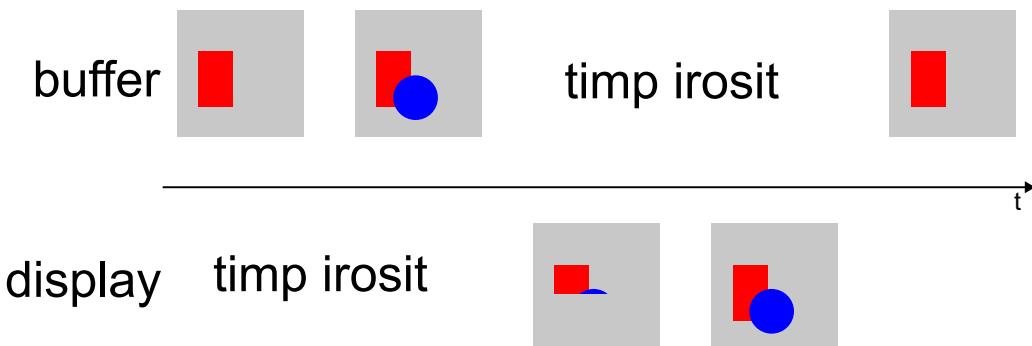


Figura 6.4: Sincronizarea elimină erorile de afișare, dar timpii de așteptare vor reduce numărul de cadre ce pot fi afișate pe secundă

Pentru a îmbunătăți situația se pot folosi două frame buffere. Acum se poate randa într-unul din ele, în timp ce se va transmite celălalt către display. Astfel nu vor mai exista timpi de așteptare nejustificate. Bineîntelese, în cazul în care noul cadru este randat înainte ca cel vechi să se fi transmis, procesorul va fi nevoie să aștepte finalizarea transmisiiei. Similar, dacă procesorul nu termină de randat cadrul nou până când este finalizat transferul celui vechi, transmisia noului cadru nu va putea începe încă. Dorindu-se o transmisie concomitentă cu randarea, este obligatorie folosirea DMA.

Totuși, folosirea a două frame buffere implică și o nouă problemă, anume consumul dublat de memorie. Un cadru are rezoluția de 240x240. Astfel un frame buffer trebuie să conțină 57600 de pixeli. Fiecare pixel are 16 biți (2 bytes), deci un frame bufer ar consuma 115200 bytes. Două frame buffere consumă 230400 bytes. Acestea ar încăpea marginal în cele 4 bânci de memorie SRAM ale RP2040.

Pentru a spori performanța calculului paralel am urmărit evitarea coliziunilor acceselor la memorie. În mod obișnuit, Raspberry Pi Pico folosește memoria SRAM ca striped memory. Asta înseamnă că adrese consecutive se află în bânci de memorie consecutive. Pentru aplicații obișnuite această tehnică este benefică. În cazul de față, striped memory ar permite plasarea frame bufferelor și a altor variabile frecvent accesate (de procesor sau DMA) în aceeași bancă de memorie, ceea ce ar duce la multe cicluri de întârziere, reducând performanța randării.

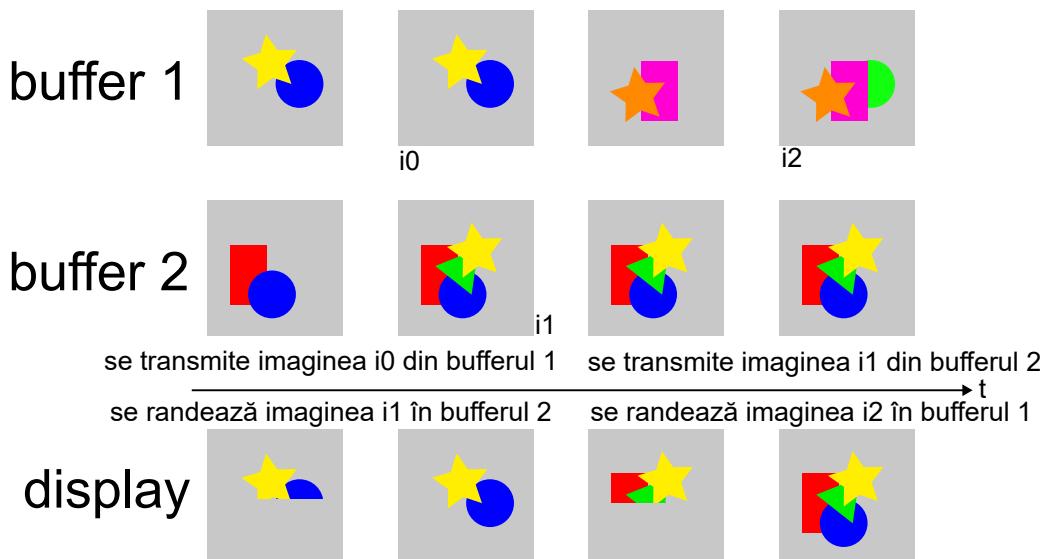


Figura 6.5: Folosirea a două frame buffere

Am urmărit atunci plasarea frame bufferelor în bănci de memorie separate de restul programului. Cele două frame buffere nu vor încăpea însă în doar 3 bănci de memorie ($3 \times 64kB = 192kB < 230kB$). Pentru a reduce consumul de memorie am folosit o proprietate importantă a afişajului, anume faptul că este rotund. Nu vom mai reține pixelii din zonele care nu există fizic. Cu alte cuvinte, vor mai fi necesari doar $\pi \times 120^2$ pixeli per frame buffer, adică în jur de 181kB pentru amândouă frame bufferele. Această reducere permite plasarea bufferelor în 3 bănci SRAM, rămânând și spațiu aditional.

Structura de date pentru memorarea unui frame buffer este un vector de 240 de pointeri. Fiecare pointer corespunde unei linii și va referi către un vector cu atâțea elemente, câți pixeli sunt pe linia respectivă. În plus, se mai folosește un vector de întregi, *cuts* [], pentru a reține numărul de pixeli de pe fiecare linie.

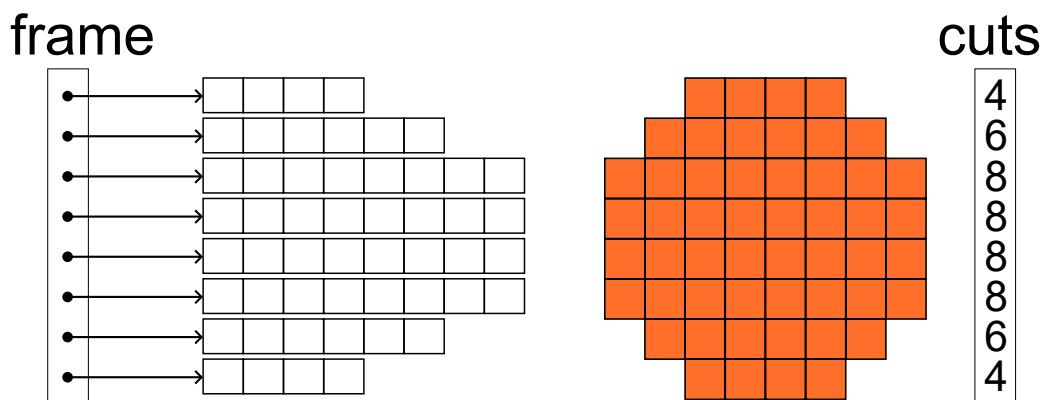


Figura 6.6: Structura unui frame buffer exemplificată pentru un display cu rezoluția de 8x8

```

1 MEMORY
2 {
3     FLASH(rx) : ORIGIN = 0x10000000, LENGTH = 2048k
4     RAM(rwx) : ORIGIN = 0x21000000, LENGTH = 64k
5     FRAME(rwx) : ORIGIN = 0x21010000, LENGTH = 192k
6     SCRATCH_X(rwx) : ORIGIN = 0x20040000, LENGTH = 4k
7     SCRATCH_Y(rwx) : ORIGIN = 0x20041000, LENGTH = 4k
8 }
9 ...
10 .frameAddress 0x21010000 :
11 {
12     KEEP(*(.frameAddress));
13 } > FRAME AT> FLASH

```

Fragmentul 6.3: Modificările aduse linker script-ului pentru a acomoda memoria frame bufferelor

Pentru a putea realiza cele mai sus prezentate am modificat linker script-ul, reconfigurând zonele de memorie RAM și adăugând nume de secțiuni pentru plasarea frame bufferelor în zonele corecte de memorie (vezi Fragmentul 6.3). Acum se pune problema transferului acestui buffer către display. Dacă înainte puteam folosi un canal DMA și trimite o matrice de 240x240 de pixeli, acum nu mai există acea matrice. O variantă ar fi să se transfere câte o linie, iar după fiecare linie procesorul să intervină și să seteze indexul memoriei displayului pentru a pregăti transferul următoarei linii. Putem însă aduce o îmbunătățire: către display vom transmite 240x240 de pixeli, dar pixelii care nu sunt vizibili vor conține, de fapt o dublură a pixelilor de pe liniile anterioare. Beneficiul este acela că procesorul nu va mai trebui să îi comunice displayului modificarea adreselor. Din moment ce dorim să transmitem dubluri va trebui să reconfigurăm canalul DMA după fiecare linie. Acest lucru este posibil fără intervenția procesorului, folosind un alt canal DMA care va configura canalul principal de transfer.

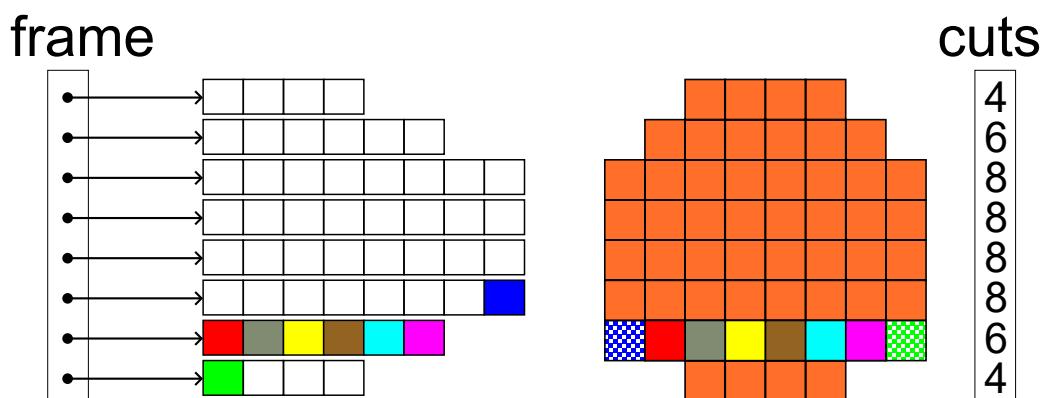


Figura 6.7: Transferul unei linii (linia 7) către display

Practic, vom începe transferul unei linii cu atâtia pixeli în urmă, câtii pixeli lipsesc din ecran. Similar, vom continua cu atâtia pixeli peste linia curentă din buffer, câtii pixeli lipsesc din ecran. În cazul concret arătat în figură, pixelul albastru face, de fapt, parte din linia 6. Totuși, el este transmis pentru a afișa linia 7. El va ajunge în memoria internă a display-ului, dar nu va apărea fizic niciunde pe ecran. La fel se va întâmpla și cu pixelul verde.

În final, către display se vor transmite 240x240 de pixeli a căte 16 biți fiecare, la frecvența maximă posibilă de 62.5MHz , rezultând într-o viteză de afișare de maxim 67 de cadre pe secundă.

Pentru randarea cadrelor s-au creat diverse funcții cu diversi parametrii, după caz:

- `drawRectangle(x, y, h, w, color)` - desenează un dreptunghi
- `drawRectGradientV`, `drawRectGradientH` - desenează dreptunghiuri cu culoare în gradient
- `drawImage` - desenează o imagine dată ca parametru
- alte funcții pentru desenarea rotită sau scalată a unei imagini
- `fillScreen` - umple ecranul cu o culoare
- `printLine` - printează un text pe ecran
- `lengthOf` - calculează lungimea în pixeli a unui text dat ca parametru

Pentru text am preluat font-ul din *M5Stack*.

Am încercat scrierea funcțiilor de randare în limbaj de asamblare pentru o eficiență sporită. Scriind o funcție simplă (cea de desenare a unui dreptunghi) în limbaj de asamblare nu am obținut un avantaj considerabil față de funcția scrisă în C (având totuși grijă să scriu cod C eficient). Acest experiment, numărul redus de registrii ai procesorului și dificultatea scrierii algoritmilor grafici în limbaj de asamblare m-a determinat să folosesc, în final, limbajul C pentru aceste funcții.

Comunicarea cu această bibliotecă reprezintă transferul unei liste cu elementele care trebuie desenate. În contextul RP2040, această listă va fi transmisă prin FIFO-ul interprocesor. Se începe transferul prin transmiterea unui macro `START_EDIT`. Apoi se transmit oricără grupuri de 9 words, fiecare constând dintr-un macro ce reprezintă funcția de randare ce se dorește a fi apelată urmată de parametrii acesteia. Dacă funcția are mai puțini parametrii, se completează cu 0. În sfârșit, se va trimite macro-ul `SUBMIT_LIST`. Din acest moment, biblioteca va randa cadrul următor folosind noua listă.

```
1 | push(START_EDIT);  
2 | push(FILL_SCREEN);push(0);  
3 | push(0);push(0);push(0);push(0);push(0);push(0);push(0);  
4 | push(DRAW_IMAGE);push(120);push(120);push(JOYSTICK_IMG);  
5 | push(0);push(0);push(0);push(0);push(1);  
6 | push(SUBMIT_LIST);
```

Fragmentul 6.4: Trimiterea unei liste pentru randarea imaginii cu un joystick în mijlocul ecranului

6.5 BIBLIOTECA WS2812 - INELUL CU LED-URI

Pentru a controla inelul cu LED-uri se pun la dispozitie următoarele funcții:

- `WS2812_init()` care initializează mașina PIO și canalul DMA
- `WS2812_set_ring_array(void *arr, bool rotating)` care setează vectorul de culori care să fie afișat folosind LED-urile. În plus, se poate specifica dacă aceste culori să urmărească butonul în rotația sa sau nu
- `WS2812_refresh(uint r)` - funcția de calculare și transmisie a culorilor către LED-uri

Vectorul transmis funcției `WS2812_set_ring_array` trebuie să aibă exact 48 de elemente pe 8 biți. Aceste elemente formează 16 grupuri (pixeli) de câte 3 (culori) și vor reprezenta, în ordine, canalele roșu, verde și albastru a fiecărui LED.

6.6 BIBLIOTECA HX711 - AMPLIFICATOARELE

Citirea valorilor de la punctile tensomterice este conceptual similară cu cea a senzorului magnetic. Apelul unei funcții de initializare, `void HX711_init()`, va determina pornirea a câte două canale DMA și a unei mașini PIO pentru fiecare amplificator HX711. Canalele DMA destinate unui amplificator sunt configurate să se pornească unul pe celălalt după terminarea fiecărui. Fiecare masină PIO va fi configurată să citească un amplificator diferit, iar fiecare pereche de canale DMA va fi configurată să preia date de la mașina PIO corespunzătoare și să le depună în variabile diferite, anume `int32_t Xdma, Ydma, Pdma;`.

După apelul acestei funcții de initializare s-ar putea folosi direct valorile citite. Acestea conțin, în cei mai puțin semnificativi 24 de biți, valoarea citită de la amplificator. Amplificatorul transmite un număr cu semn. Prin urmare, pentru a folosi variabilele citite în operații obisnuite va trebui aplicarea unei transformări: `((Xdma) << 8) >> 8`, propagând astfel bitul de semn în locul corespunzător.

S-a constatat în practică faptul că cele 3 canale nu corespund întocmai celor 3 axe de înclinare ale butonului. Pentru a rezolva această problemă se aplică valorilor citite o transformare afină (se scade un offset, după care se înmulțește cu o matrice). Pentru a efectua aceste calcule s-a pus la dispozitie funcția `void HX711_update()`. Valorile finale sunt reținute în variabilele `int32_t Xtilt, Ytilt, Press;`

$$\begin{bmatrix} Xtilt \\ Ytilt \\ Press \end{bmatrix} = \begin{bmatrix} txx & tyx & tpx \\ txy & tyy & tpy \\ txp & typ & tpp \end{bmatrix} * \left(\begin{bmatrix} Xdma \\ Ydma \\ Pdma \end{bmatrix} - \begin{bmatrix} Xoffset \\ Yoffset \\ Poffset \end{bmatrix} \right) \quad (6.5)$$

Variind termenii ecuației de mai sus se realizează, de fapt, o calibrare a punctilor tensometrice, rezultând astfel o măsurare mult mai precisă a înclinărilor și a apăsării.

6.7 MENIUL DE SETĂRI

Meniul de setări oferă utilizatorului posibilitatea de a configura aspecte ale funcționării dispozitivului. Totodată, din punct de vedere al programării, acest meniu servește drept exemplu de folosire a bibliotecilor descrise mai sus. Pentru a intra în acest meniu, se va ține apăsat butonul de pe partea inferioară a carcasei în timpul conectării acesteia la un sistem de calcul (PC). Meniul prezintă 6 setări/optiuni dispuse grafic într-un cerc. Pentru a naviga în meniu se folosește rotirea butonului, iar pentru a începe modificarea unei setări se apasă butonul. Setările sunt:

- **Mode** - setează modul de funcționare al dispozitivului. După apăsare, se prezintă utilizatorului o alegere între cele 3 moduri de funcționare (mouse, joystick, SmartKnob). Modurile sunt reprezentate prin pictograme. Selectia se face rotind butonul și apăsând la modul dorit.
- **Motor power** - setează puterea feedback-ului haptic, adică a motorului. La intrarea în meniu se poate roti de buton pentru a crește sau a scădea puterea. În mijlocul ecranului apare un cerc al cărui factor de umplere corespunde cu puterea setată. Confirmarea se face apăsând butonul.
- **Run** - salvează modificările și repornește dispozitivul. Dacă nu se mai ține apăsat butonul din partea inferioară a carcasei, dispozitivul va porni în modul selectat.
- **LED ring** - setează intensitatea luminoasă a inelului cu LED-uri. Funcționarea acestei setări este similară cu cea pentru puterea motorului.
- **LCD screen** - setează intensitatea luminoasă a ecranului LCD. Funcționarea acestei setări este similară cu cea pentru puterea motorului.
- **Calibrate** - permite calibrarea butonului. Se poate forța intrarea în acest meniu apăsând butonul de pe carcăsă după ce dispozitivul a intrat în meniul de setări. Se vor urmări apoi instrucțiunile de pe ecran.

La finalizarea modificării unei setări se realizează revenirea în meniul principal.



Figura 6.8: Meniul de setări



Figura 6.9:
Setarea luminozității ecranului LCD.
Momentan, luminozitatea este la o
treime din maxim.

Din programul principal, intrarea în meniul de setări se face printr-un singur apel de funcție, `void settings_menu()`, iar aceasta nu returnează niciodată. Ieșirea din meniu implică resetarea dispozitivului. Această funcție este, de fapt, o buclă în care se determină poziția utilizatorului în meniu, se trimit către bibliotecile de afișare și feedback comenzi adecvate și se trece, eventual, în altă stare a meniului. La selectarea opțiunii de ieșire din meniu, bucla este întreruptă. Apoi se apelează funcția de salvare a setărilor și se folosește watchdog-ul pentru a reporni microcontrolerul.

Pentru a citi și a salva setările am folosit memoria FLASH în care este stocat și programul. Am luat această decizie întrucât acesta este singurul mediu nonvolatil de stocare de care dispune placa Raspberry Pi Pico. Memoria FLASH este organizată în sectoare și pagini. Sectorul este unitatea minimă care se poate șterge și are dimensiunea de 4kB. Pagina este unitatea minimă care se poate scrie și are dimensiunea de 256B. Prin urmare, configurația setărilor va ocupa un sector întreg, fiind necesară ștergerea setărilor vechi, chiar dacă o pagină este suficientă pentru aceasta.

Pentru a aloca exact 256 de bytes indiferent de cât spațiu efectiv este folosit de variabilele am folosit o uniune. Astfel, dacă mai trebuie adăugate alte setări în structură, nu trebuie recalculat spațiul adițional necesar pentru a aduce dimensiunea structurii la 256 de bytes.

```
1 struct Settings{//should use 256 bytes (1 writable FLASH page)
2     union{
3         struct {
4             enum {MOUSE_MODE, JOYSTICK_MODE, SMART_MODE} mode;
5             int32_t LCD_max_brightness;
6             int32_t LED_max_brightness;
7             int32_t motor_power_max;
8             ...
9         };
10        uint8_t filler[256];
11    };
12};
13 const uint8_t *flash_target_contents = (const uint8_t *) (XIP_BASE +
14     SETTINGS_OFFSET); //settings address in FLASH
15 struct Settings settings; //editable settings copy in RAM
16 void read_settings(){
17     memcpy(settings.filler[i], flash_target_contents[i], 256); }
18 void save_settings(){...
19     flash_range_erase(SETTINGS_OFFSET, FLASH_SECTOR_SIZE);
20     flash_range_program(SETTINGS_OFFSET, (uint8_t*)&settings,
21     FLASH_PAGE_SIZE); }
```

Fragmentul 6.5: Structura ce reține setările și utilizarea acesteia

Structura este plasată la adresa corectă în FLASH folosind un simplu pointer către adresa dorită. Citirea din FLASH se face identic cu citirea din RAM datorită arhitecturii unificate a memoriei. Pentru ștergere și scriere am folosit funcții din biblioteca Pico SDK.

6.8 PROGRAMUL PRINCIPAL

Sarcinile programului principal sunt:

- citirea setărilor din memorie
- intrarea în meniul de setări, dacă este cazul
- inițializarea tuturor bibliotecilor pentru controlul perifericelor
- intrarea într-o buclă infinită, în care se vor apela funcțiile de actualizare/menținere a bibliotecilor
- depunerea și preluarea/procesarea datelor din descriptorii HID în modul SmartKnob

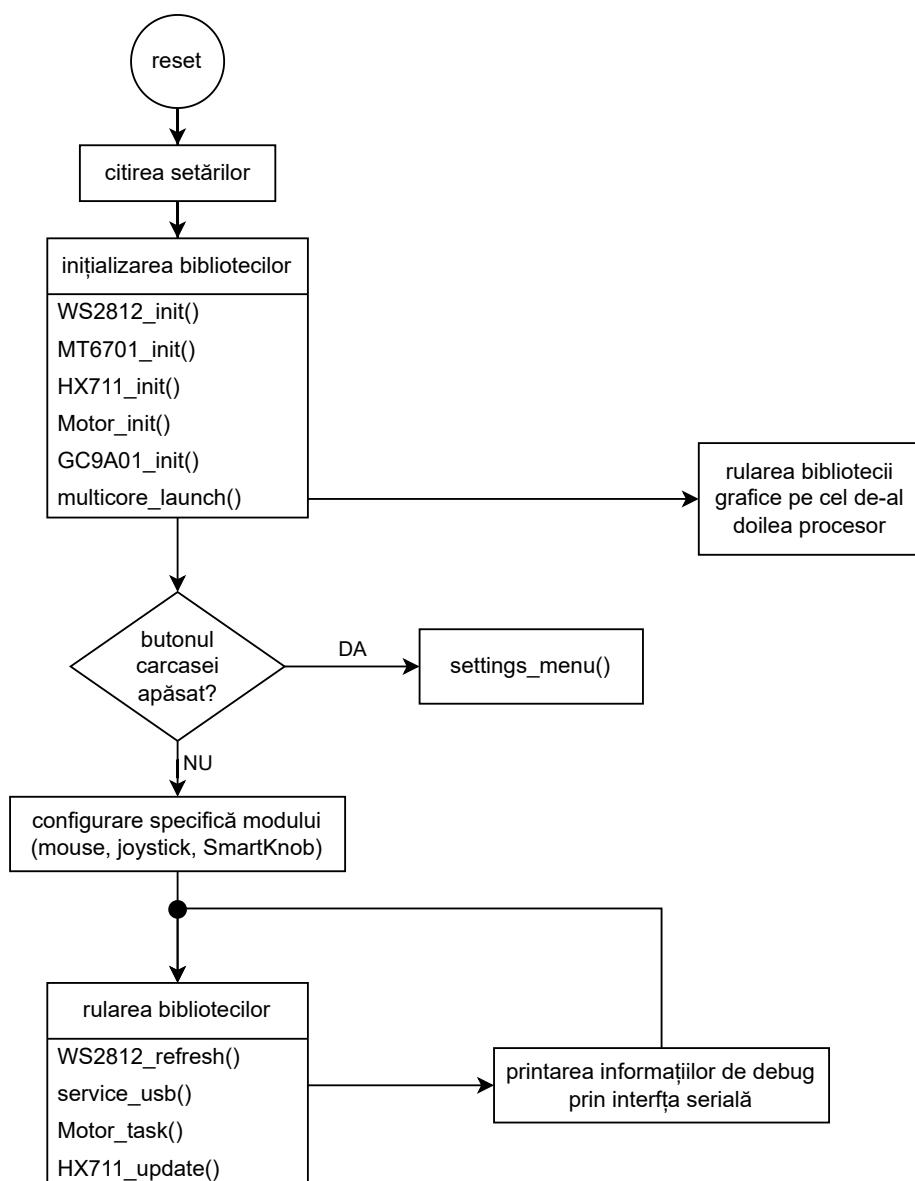


Figura 6.10: Structura programului principal

6.9 CREAREA UNEI APLICAȚII PENTRU INTERFAȚAREA CU SMARTKNOB

Pentru a putea valorifica toate funcționalitățile SmartKnob este necesară crearea unei aplicații. Datorită folosirii clasei HID în implementare, compatibilitatea este asigurată cu majoritatea sistemelor de operare. Se pot folosi o multitudine de biblioteci usb pentru comunicarea cu dispozitivul. În cazul în care sistemul de operare nu permite accesul direct (low-level) la usb se pot folosi biblioteci usb-HID. Acestea din urmă ar trebui să eliminate orice eventuală problemă de compatibilitate sau permișuni, dar poate avea performanțe mai scăzute. Toată comunicarea cu dispozitivul se desfășoară exclusiv folosind rapoarte HID.

Am decis să realizez aplicația de test în Unity, folosind limbajul de programare C# și biblioteca **HidLibrary** [19]. Elementele principale ale programului relevant (cel care trimite și primește rapoarte) sunt ilustrate în Fragmentul 6.6. La început se va căuta dispozitivul pe baza perechii VID-PID. Apoi se va începe o citire și o scriere a câte un raport, urmând ca rapoartele viitoare să fie transmise folosind funcții callback.

Câteva dintre funcționalitățile aplicației sunt:

- **Beat mode** - generarea unor ritmuri (beats). Înclinarea butonului în sus, jos, stânga sau dreapta determină redarea unui efect sonor diferit
- **Flanger mode** - simulează efectul de flanger. Rotirea butonului determină schimbarea unui parametru al efectului
- **Scratch mode** - butonul va începe să se rotească la viteză constantă și va simula un platân de scratching. Viteza de rotație a butonului controlează viteza de redare a unei melodii.



Figura 6.11: Screenshot al aplicației (surse pentru pictograme: cymbalfusion)

```
1 using HidLibrary;
2 public class Test_example_1 : MonoBehaviour
3 {
4     static HidDevice smartKnob = null;
5     static HidReport reportIn = null;
6     static Queue<HidReport> writeQueue = new Queue<HidReport>();
7     void Start()
8     {
9         foreach(HidDevice d in HidDevices.Enumerate())
10            if(d.Attributes.VendorId == 0xCAFE && d.Attributes.ProductId
11                == 0x4005)
12                smartKnob = d;
13        if(smartKnob == null) Debug.LogError("Device not found!");
14
15        smartKnob.OpenDevice();
16        if(!smartKnob.IsOpen) Debug.LogError("Device not open!");
17
18        smartKnob.ReadReport(OnReport);
19        HidReport rep = new HidReport(65);
20        for(int i=0;i<64;i++) rep.Data[i]=0;
21        smartKnob.WriteReport(rep, OnReportW);
22    }
23    private static void OnReport(HidReport report)
24    {
25        reportIn = report;
26        smartKnob.ReadReport(OnReport);
27    }
28    private static void OnReportW(bool success)
29    {
30        if(writeQueue.Count() == 0) return;
31        HidReport rep = writeQueue.Dequeue();
32        smartKnob.WriteReport(rep, OnReportW);
33    }
34    private static void SendReport(HidReport rep)
35    {
36        if(writeQueue.Count()==0)
37            smartKnob.WriteReport(rep, OnReportW);
38        else
39            writeQueue.Enqueue(rep);
40    }
41 }
```

Fragmentul 6.6: Codul C# pentru comunicarea cu SmartKnob

7. CONCLUZII

7.1 CONCLUZII GENERALE

Am reușit să creez un dispozitiv cu feedback haptic conform specificațiilor dorite. Forța dezvoltată de motor se poate simți, dar s-ar putea căuta o variantă pentru creșterea acesteia. Software-ul scris este modular și ușor de extins la nevoie. Sistemul dezvoltat este compatibil cu biblioteci obișnuite de comunicare USB.

Se poate realiza un motor cu statorul dintr-un material nemagnetic, dar puterea acestuia este mult diminuată și a trebuit compensată prin dimensiuni crescute. Fenomenul de cogging este eliminat complet. Puntele tensometrice trebuie lipite definitiv în poziția finală pentru a putea estima valorile generate de acestea. În plus, deformările plastice ale cadrului metalic afectează semnificativ măsurătorile.

Motorul, deși folosește curenti de ordinul sutelor de miliamperi, nu cauzează interferențe semnificative în sistem. Folosirea bibliotecilor tinyUSB și a Pico SDK a accelerat dezvoltarea sistemului. Folosind C# și biblioteca predefinită HidLibrary nu s-a obținut o performanță ideală din punct de vedere al ratei de transfer al rapoartelor HID.

7.2 PERSPECTIVE DE DEZVOLTARE

Din punct de vedere mecanic, s-ar putea analiza în continuare motorul. Se poate încerca un stator oblic din material feromagnetic. Astfel ar crește forța/cuplul și eficiența motorului, dar s-ar putea să apară probleme de cogging. Motorul ar putea fi astfel mai mic, iar astfel s-ar putea crea o carcăsa mai "subțire", mai plăcută din punct de vedere vizual. În plus, s-ar putea încerca diverse combinații ale materialului cadrului metalic și adezivului pentru lipirea tensometrelor. Se poate crea un PCB, reducând riscul dezlipirii unor fire și ajutând la reducerea dimensiunilor părții electronice.

Se pot adăuga biblioteci grafice diverse funcții de randare. Se poate adăuga un endpoint de tip bulk pentru a transfera imagini către SmartKnob. În plus, s-ar putea adăuga funcționalități IoT dispozitivului. Raspberry Pi Pico are o versiune ce are WiFi integrat, aceasta putând înlocui direct placă Raspberry Pi Pico folosită.

7.3 SINTEZA CONTRIBUȚIILOR

- crearea designului și printarea 3D carcasei dispozitivului
- proiectarea și construcția unui motor special fără perii
- proiectarea interconectării modulelor
- realizarea conexiunilor electrice între module fără a folosi un PCB
- crearea unor biblioteci software pentru interfațarea cu modulele
- scrierea unui soft pentru SmartKnob, permitând diverse moduri de funcționare
- alcătuirea unei aplicații demonstrative în Unity pentru a ilustra diverse utilizări ale dispozitivului

8. ANEXA 1 - RAPOARTE HID

Tabelul 8.1: Descriptorul pentru modul joystick

Nume	Offset	Tip	Descriere	Utilizare în proiect
x	0x0	int8	Axa X a joystickului	Înclinarea pe axa X
y	0x1	int8	Axa Y a joystickului	Înclinarea pe axa Y
z	0x2	int8	Axa Z a joystickului	Viteza de rotație a butonului
rz	0x3	int8	Rotatia pe axa Z	
rx	0x4	int8	Rotatia pe axa X	
ry	0x5	int8	Rotatia pe axa Y	
hat	0x6	uint8	Butoane direcționale	Neutilizat - permanent 0
buttons	0x7	uint32	Restul butoanelor	butonul 1 - corespunde apăsării butonului butonul 2 - corespunde unei scurte apăsări restul butoanelor - nefolosite

Tabelul 8.2: Descriptorul pentru modul mouse

Nume	Offset	Tip	Descriere	Utilizare în proiect
buttons	0x0	uint8	Butoane	apăsarea butonului corespunde apăsării butonului stânga al mouse-ului
x	0x1	int8	Deplasare pe axa X	Înclinarea pe axa X
y	0x2	int8	Deplasare pe axa Y	Înclinarea pe axa Y
wheel	0x3	int8	Derulare verticală	corespunde rotației butonului pentru a simula roata de derulare
pan	0x4	int8	Derulare orizontală	Neutilizat - permanent 0

Tabelul 8.3: Descriptorul de intrare pentru modul SmartKnob

Nume	Offset	Tip	Descriere
Xtilt	0x0	int32	Înclinarea pe axa X – rezultatul bibliotecii HX711
Ytilt	0x4	int32	Înclinarea pe axa Y – rezultatul bibliotecii HX711
Press	0x8	int32	Apăsarea butonului – rezultatul bibliotecii HX711
angle	0xC	uint32	Unghiul butonului – direct de la senzorul MT6701
speed	0x10	int32	Viteza unghiulară a butonului în 10^-3*grad/secundă

Tabelul 8.4: Descriptorul de ieșire pentru controlul inelului cu LED-uri

Nume	Offset	Tip	Descriere
type	0x0	uint8	Tipul raportului: pentru LED este 1
array	0x1	48 * uint8	Vectorul colorilor LED-urilor în ordine roșu-verde-albastru
rotating	0x49	uint8	Modelul va urmări rotația butonului dacă acest câmp nu e 0

Tabelul 8.5: Descriptorul de ieșire pentru controlul motorului

Nume	Offset	Tip	Descriere
type	0x0	uint8	Tipul raportului: pentru motor este 3
vibrate	0x1	uint8	Motorul va vibra pentru 50ms dacă acest câmp nu este 0
mode	0x2	uint8	Setează modul de funcționare: 1 - viteză constantă 2 - rotație liberă fără feedback 3 - rotație cu frecare 4 - detente uniforme 5 - detente customize următoarele câmpuri depind de câmpul de mod
n	0x3	(u)int8	modul 3: frecarea dorită
offset	0x4	float	modurile 4 și 5: număr de detente
detents	0x8	n * float	modul 5: pozițiile dorite ale detentelor

Tabelul 8.6: Descriptorul de ieșire pentru controlul bibliotecii grafice

Nume	Offset	Tip	Descriere
type	0x0	uint8	Tipul raportului: pentru LCD este 2
n	0x1	uint8	Numărul de elemente ale câmpului array. Fiecare element are 4 bytes.
flags	0x2	uint8	Un câmp de biți bitul 0 - primul raport din secvență bitul 1 - ultimul raport din secvență
neutilizat	0x3		
array	0x4	n * (int32, uint32, sau float)	Fiecare comandă conține indexul funcției dorite și parametrii acesteia. Dacă se dorește transmisia mai multor comenzi decât încap într-un raport, se transmit comenziile în rapoarte diferite. Se poate separa antetul de parametri și parametrii între ei în rapoarte succesive. Se va seta corespunzător câmpul flags.

Tabelul 8.7: Structura unei comenzi pentru biblioteca grafică

Nume	Offset relativ	Tip	Descriere
header	0x0	uint32	În cel mai semnificativ byte se găsește un câmp de biți. Fiecare bit corespunde prezenței în raport a unui parametru. Parametrii care nu sunt prezenti vor fi transmiși ca 0. În partea mai puțin semnificativă se găsește codul funcției de apelat.
parameters	0x4	m * (int32, uint32 sau float)	Pentru fiecare bit de 1 din câmpul de biți se va depune un parametru. Acesta este mereu pe 32 de biți și va fi transmis direct bibliotecii.

Tabelul 8.8: Un descriptor pentru biblioteca grafică

Offset	+3	+2	+1	+0	Explicație
0x0	0x0	0x03	0x07	0x02	se transmite un mesaj bibliotecii grafice se vor transmite 7 elemente raportul este primul și ultimul în secvență
0x4	0b00000001	0x0	0x0	0x0C	se trimit comanda FILL_SCREEN se transmite doar primul parametru
0x8	0x0	0xC0IA			se va folosi culoarea 0xC01A
0xC	0b00001111	0x0	0x0	0x09	se transmite comanda DRAW_CIRCLE se transmit primii 4 parametri
0x10	150 (decimal)				coordonata X a cercului de desenat
0x14	90 (decimal)				coordonata Y a cercului de desenat
0x18	10 (decimal)		0x0		raza cercului va fi de 10 pixeli formatul este virgulă fixă 16.16
0x1C	0x0		0xF0F0		cercul va avea culoarea 0xF0F0

9. BIBLIOGRAFIE

- [1] *Smartknob*, scottbez1, accesat iunie 2024. [Online]. Available: <https://github.com/scottbez1/smartknob>.
- [2] *Tuya zigbee smart knob*, accesat iunie 2024. [Online]. Available: https://torelectric.ro/index.php?id_product=290&rewrite=buton-scenariu-smart-knob-zigbee&controller=product.
- [3] *Bht-7000-gclzb 240v ac 3a smart knob*, accesat iunie 2024. [Online]. Available: <https://ro.buy2fix.store/products/bht-7000-gclzb-240v-ac-3a-smart-knob-thermostat-dry-junction-controller-with-zigbeeblack>.
- [4] *Raspberry pi pico datasheet*, Raspberry Pi Ltd, 2024. [Online]. Available: <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf>.
- [5] *Rp2040 datasheet*, Raspberry Pi Ltd, 2024. [Online]. Available: <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>.
- [6] *Pico c sdk*, Raspberry Pi Ltd, 2024. [Online]. Available: <https://www.raspberrypi.com/documentation/pico-sdk/>.
- [7] *Tinyusb*, hathach, 2024. [Online]. Available: <https://github.com/hathach/tinyusb>.
- [8] *Usb in a nutshell*, 2024. [Online]. Available: <https://www.beyondlogic.org/usbnutshell/usb1.shtml>.
- [9] *Usb made simple*, 2024. [Online]. Available: <https://www.usbmadesimple.co.uk/>.
- [10] J. Axelson, *USB complete*. 2005.
- [11] “Winding scheme calculator”, 2024. [Online]. Available: <https://www.bavaria-direct.co.za/scheme/calculator/>.
- [12] R. Parsons, “Selecting the best pole and slot combination for a bldc (pmsm) motor with concentrated windings”, 2019. [Online]. Available: <https://things-in-motion.blogspot.com/2019/01/selecting-best-pole-and-slot.html>.
- [13] *Dual brushed dc motor drive circuit mx1616h*, Sinotech Mixic Electronics, 2024. [Online]. Available: https://d229kd5ey79jzj.cloudfront.net/2656/H-Bridge-Dual_SMD_MX1616H.pdf.
- [14] M. f. A. D. Usach, “Spi interface”, [Online]. Available: <https://www.analog.com/media/en/technical-documentation/app-notes/an-1248.pdf>.
- [15] *Mt6701 magnetic sensor documentation*, MagnTek, 2024. [Online]. Available: https://uploadcdn.oneyac.com/attachments/files/brand_pdf/magntek/F3/CA/MT6701QT-STD.pdf.

- [16] *Gc9a01a tft lcd single chip driver*, Galaxycore, 2019. [Online]. Available: <https://www.buydisplay.com/download/ic/GC9A01A.pdf>.
- [17] *Hx711 24-bit analog-to-digital converter (adc) for weigh scales*, AVIA Semiconductor, 2024. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf.
- [18] *Ws2812 intelligent control led integrated light source*, Worldsemi, 2024. [Online]. Available: <https://cdn-shop.adafruit.com/datasheets/WS2812.pdf>.
- [19] *Hidlibrary*, accesat iunie 2024. [Online]. Available: <https://github.com/mikeobrien/HidLibrary>.

**DECLARAȚIE DE AUTENTICITATE A
LUCRĂRII DE FINALIZARE A STUDIILOR***

Subsemnatul PLĂVĂȚ VLAD

legitimat cu C.I. seria TZ nr. 554525

CNP 5011019350025

autorul lucrării SMARTKNOB: SISTEM HAPTIC DE CONTROL INTELIGENT

elaborată în vederea susținerii examenului de finalizare a studiilor de
LICENȚĂ organizat de către Facultatea
AUTOMATICĂ ȘI CALCULATOR din cadrul Universității
Politehnica Timișoara, sesiunea IONIE a anului universitar
2023-2024, coordonator DR.ING. VALENTIN STÂNGACIU, luând în
considerare art. 34 din *Regulamentul privind organizarea și desfășurarea examenelor de
licență/diplomă și disertație*, aprobat prin HS nr. 109/14.05.2020 și cunoscând faptul că în
cazul constatării ulterioare a unor declarații false, voi suporta sancțiunea administrativă
prevăzută de art. 146 din Legea nr. 1/2011 – legea educației naționale și anume anularea
diplomei de studii, declar pe proprie răspundere, că:

- această lucrare este rezultatul propriei activități intelectuale;
- lucrarea nu conține texte, date sau elemente de grafică din alte lucrări sau din alte surse fără ca acestea să nu fie citate, inclusiv situația în care sursa o reprezintă o altă lucrare/alte lucrări ale subsemnatului;
- sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor;
- această lucrare nu a mai fost prezentată în fața unei alte comisii de examen/prezentată public/publicată de licență/diplomă/disertație;
- În elaborarea lucrării am utilizat instrumente specifice inteligenței artificiale (IA) și anume _____ (denumirea) _____ (sursa), pe care le-am citat în conținutul lucrării/nu am utilizat instrumente specifice inteligenței artificiale (IA)¹.

Declar că sunt de acord ca lucrarea să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Timișoara,

Data

23.06.2024

Semnătura



*Declarația se completează de student, se semnează olograf de acesta și se inserează în lucrarea de finalizare a studiilor, la sfârșitul lucrării, ca parte integrantă.

¹ Se va păstra una dintre variante: 1 - s-a utilizat IA și se menționează sursa 2 – nu s-a utilizat IA