



Liepājas Valsts tehnikums

## **Testēšanas rīks “TT/CU Skeneris”**

Programmatūras projekta tehniskā dokumentācija

Profesionālā kvalifikācija .....  
.....

Grupas nosaukums .....  
.....

Projekta izstrādātājs .....

/vārds, uzvārds, paraksts/

Eksāmena datums 2024. gada 21. jūnijā

Liepāja 2024

## Saturs

1.	Uzdevuma formulējums .....	6
2.	Programmatūras prasību specifikācija.....	8
2.1.	Produkta perspektīva .....	8
2.2.	Sistēmas funkcionālās prasības.....	8
2.2.1.	Rīka programmatūras funkcionālās prasības .....	8
2.2.2.	Konfigurācijas aplikācijas programmatūras funkcionālās prasības .....	23
2.3.	Sistēmas nefunkcionālās prasības .....	31
2.4.	Gala lietotāja raksturiezīmes.....	31
3.	Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums.....	32
3.1.	Izvēlēto risinājuma līdzekļu un valodu apraksts .....	32
3.1.1.	Izstrādes programmatūras.....	32
3.1.2.	Izstrādes darbstacijai .....	33
3.1.3.	Produkta komponentes un ierīces .....	34
3.1.4.	Programmatūras programmēšanas valodas un bibliotēkas .....	34
3.2.	Iespējamo risinājuma līdzekļu un valodu apraksts .....	35
4.	Sistēmas modelēšana un projektēšana.....	36
4.1.	Rīka sistēmas struktūras modelis .....	36
4.1.1.	Pievienota mikročipa noteikšana .....	37
4.1.2.	Rīka savienošana ar aplikāciju.....	37
4.1.3.	Rīka valodas maiņa .....	38
4.1.4.	Datu dzēšana no rīka .....	38
4.1.5.	Real-Time-Clock komponentes pārbaude .....	39
4.1.6.	EEPROM komponentes pārbaude .....	39
4.1.7.	ControlUnit releju pārbaude .....	40
4.1.8.	ControlUnit Wiegand karšu lasītāja pārbaude .....	40
4.1.9.	ControlUnit mikročipa pārbaudes rezultāts .....	41
4.1.10.	TTunit mikročipa pārbaudes rezultāts .....	41

4.1.11.	Pārbaudes atkārtošana .....	41
4.2.	Konfigurācijas aplikācijas sistēmas struktūras modelis .....	42
4.2.1.	Aplikācijas loga ielāde .....	42
4.2.2.	Vēstures rāmja ielāde .....	43
4.2.3.	Rīka datu izdzēšana.....	43
4.2.4.	Konfigurācijas rāmja ielāde.....	44
4.2.5.	Rīka saskarnes valodas maiņa.....	44
4.2.6.	Aplikācijas savienošana ar rīku .....	45
4.3.	Rīka programmatūras klašu diagramma.....	46
4.4.	Konfigurācijas aplikācijas programmatūras klašu diagramma.....	48
4.5.	Funkcionālais un dinamiskais sistēmas modelis.....	49
4.5.1.	Rīka programmatūras stāvokļu diagramma .....	49
4.5.2.	Konfigurācijas aplikācijas programmatūras stāvokļu diagramma .....	50
4.5.3.	Datu plūsmas diagramma .....	51
5.	Lietotāju ceļvedis .....	52
5.1	TTunit mikročipa pārbaudes sagataves process.....	52
5.2	ControlUnit mikročipa pārbaudes sagataves process.....	53
5.3	ControlUnit mikročipa Wiegand karšu lasītāja pārbaudes process .....	54
5.4	Rīka savienošanas process ar konfigurācijas aplikāciju .....	58
5.5	Rīka saskarnes pamācība .....	59
5.6	Konfigurācijas aplikācijas saskarnes pamācība .....	59
6.	Testēšanas dokumentācija .....	61
6.1.	Izvēlētās testēšanas metodes, rīku apraksts un pamatojums.....	61
6.1.1.	Melnās kastes testēšanas rīku apraksts un pamatojums .....	61
6.1.2.	Baltās kastes testēšanas rīku apraksts un pamatojums .....	62
6.2.	Testpiemēru kopa.....	63
6.2.1.	Rīka programmatūras testpiemēru kopa.....	63
6.2.2.	Konfigurācijas aplikācijas testpiemēru kopa .....	64

6.3.	Testēšanas žurnāls .....	65
6.3.1.	Rīka programmatūras testēšanas žurnāls .....	65
6.3.2.	Konfigurācijas aplikācijas testēšanas žurnāls .....	66
7.	Secinājumi .....	68
7.1.1.	Rezultāta novērtējums .....	68
7.1.2.	Uzdevumu sasniegšanas analīze .....	68
7.1.3.	Darba apjoms .....	69
8.	Lietoto terminu un saīsinājumu skaidrojumi .....	70
9.	Literatūras un informācijas avotu saraksts .....	71
	Pielikumi .....	72

## Ievads

Šis dokuments tiek uzrakstīts, ievērojot "Liepājas Valsts tehnikums" mācību programmas "Programmēšanas tehnīķis" kvalifikācijas eksāmenu prasības. Projekta mērķis ir izstrādāt augstas kvalitātes testēšanas rīku "TT/CU skeneris" programmatūru, kas atbilst šīs izglītības iestādes prasībām.

Galvenais aspekts, ko šis dokuments piedāvā, ir plaša programmatūras produkta analīze, ietverot perspektīvu, līdzekļus un modeļus, lai sasniegtu maksimālo efektivitāti un funkcionālo pilnveidi. Var uzskatīt, ka šis projekts ietverēs inovatīvus risinājumus, kas būs saskaņots ar profesijas nozares labākajām praksēm.

Izstrādātās programmatūras nepieciešamība izriet no vajadzībām izstrādātāja prakses vietas uzņēmumā "SIA InPass". Pēc pasūtītāja velmēm izstrādātājs ir apņēmies radīt divas programmatūras - rīka programmatūru un rīka konfigurācijas programmatūru, kas efektīvi pārbaudīs dažādus mikročipus, piemēram, TTunit un ControlUnit un tad mikročipu pārbaudes rezultātus apkops, konfigurēs rīka saskarnes valodu ar rīka konfigurācijas aplikāciju. Testējamie mikročipi ir būtiski izstrādātāja prakses vietā, veicot dažādu signālu pārraides un nodrošinot svarīgas funkcionalitātes, piemēram, releju ieslēgšanos un karšu nolasīšanu.

Rīka programmatūra tiks izstrādāta, lai nodrošinātu pārbaudi dažādām iekšējām mikročipa funkcionalitātēm un komponentēm, kas tiek izmantoti izstrādātāja prakses vietā. Papildus tam konfigurācijas aplikācija sniegs lietotājam iespēju apskatīt šo mikročipu pārbaudes rezultātus, kā arī pārmainīt produkta displeja valodu. Tā kā produkta gala lietotāji būs darbinieki no attiecīgā uzņēmuma, izstrādātājs pievērsīs īpašu uzmanību funkcionalitātes un ergonomikas izstrādei.

Rīks uz kura atradīsies mikročipu pārbaudes programmatūra sastāvēs no mikrokontrolliera, pogas, displeja un plastmasas ietvara, nodrošinot integrētu risinājumu ar elegantu un modernu dizainu. Rīka konfigurācijas aplikācija tiks līdzīgi vizuāli izstrādāta un ērti lietojama, gluži kā rīka programmatūra. Ergonomiskā un vienkāršā saskarne būs lietotājam draudzīga, un tā tiks precīzi aprakstīta angļu un latviešu valodā ar dažādiem nozares profesionālismiem, lai atbilstu starptautiskajiem standartiem un vienkāršotu lietošanas pieredzi darbiniekiem.

Rīka programmatūras saskarnes dizains tiks pielāgots, lai efektīvi mijiedarbotos ar pogu sistēmu, kas ļaus lietotājam intuitīvi progresēt caur testa posmiem un iegūt galīgos komponenšu darbības stāvokļa rezultātus. Izstrādātājs vēlas nodrošināt, ka, gan produkts, gan konfigurācijas aplikācija ne tikai izpilda savu pamatlīdzekļu, bet arī piedāvā lietotājiem ērtu un efektīvu darbību, atspoguļojot izstrādātāja apņemšanos sniegt augsta līmeņa risinājumu.

## 1. Uzdevuma formulējums

Šajā nodaļā tiek detalizēti aprakstīts programmatūras uzdevuma formulējums, gan rīkam (skatīt 1. attēlu), gan konfigurācijas aplikācijai, kas saistītas ar mikročipu pārbaudēm - TTunit vai ControlUnit. Produkts vēlāk seko precīzi definētiem procesiem, kas ietver dažādu funkcionalitātes pārbaudes veikšanu.

Sākot ar primāro uzdevumu, izveidot programmatūru ar mērķi noskaidrot TTunit vai ControlUnit mikročipa funkcionalitātes stāvokli, īstenojot dažādus funkcionalitātes testus. Produkta saskarne uz displeja tiek izstrādāta angļu un latviešu valodā, un pielāgota ar dažādiem profesionālismiņiem.



1.attēls. Riks.

Produkta testēšanas process tiek inicializēts, savienojot mikrokontrolieri ar TTunit vai ControlUnit mikročipu. Šajā posmā, izmantojot pogas un displeja interfeisu, produkts pakāpeniski progresē caur testu, identificējot mikročipu un veicot atbilstošus testus, kuru rezultāti tiek dokumentēti, saglabāti un pārbaudes beigās tiek izvadīti uz displeja.

Pievienojot produktu pie TTunit mikročipa (skatīt 2.attēlu), uz displeja tiek sniegtā signalizācija par veiksmīgu pievienošanos, un tiek izpildīti vairāki testi, tostarp Real-Time-Clock, EEPROM un zumera testēšana. Savukārt, savienojot produktu ar ControlUnit mikročipu (skatīt 3.attēlu), displejā tiek attēlots, ka ir pievienots ControlUnit mikročips, un tiek veikti vēl papildus pārbaudes ar vairākiem sarežģītiem procesiem, ietverot Real-Time-Clock, EEPROM, pogu releju, durvju releju, RFID karšu lasītāja un signalizācijas stāvokļa releja pārbaudi.



2.attēls. TTunit mikročips



3.attēls. ControlUnit mikročips

Pārbaudes rezultāti tiek skaidri atspoguļoti displejā, norādot, vai pārbaude ir veiksmīga vai neveiksmīga. Gadījumā, ja mikročips netiek pievienots produktam, tiek izvērstī dažādi scenāriji, lai informētu lietotāju par mikročipa neesošo pievienošanos vai par to, ka rīks tiek pievienots konfigurācijas aplikācijai. Arī gadījumā, ja viena no esošām mikročipa komponentēm, kas konfliktē ar mikročipa identificēšanu, ir daļēji bojāta, tad displejā tiek izvadīta iespējamā komponente, kas izraisa problēmu.

Produkts nodrošina, kad kāda no mikročipa testējamajām komponentēm nav darba kārtībā. Tad displejā tiek izvadīts paziņojums par neesošo komponentes darbību. Lietotājam tiek piedāvāta iespēja restartēt testu, izmantojot displeja interfeisu un pogu, ņaujot veikt papildu testēšanu pēc nepieciešamības un pārliecības.

Sekundārais uzvedums no izstrādājamā darba ir rīka konfigurācijas aplikācija, kas spēj apskatīt iepriekš veikto mikročipu pārbaudes rezultātus un mainīt produkta displeja saskarnes valodu. Savstarpējā komunikācija starp produktu un aplikāciju tiek izveidota, kad lietotājs ievada attiecošo seriālo porta numuru kurā rīks ir pievienots. Aplikācijā ir redzama mikročipu pārbaudes rezultātu izvade, kas tiek izvadīta ērtā un ergonomiskā veidā.

## **2. Programmatūras prasību specifikācija**

Šajā nodaļā tiek aprakstītas vairākas apakšnodaļas attiecoši rīka programmatūrai un konfigurācijas aplikācijas programmatūrai. Produkta perspektīva, kur tiek aprakstīta vispārējā produkta būtība, sistēmas funkcionalās un nefunkcionālās prasības nosaka visas produkta iespējamās funkcionalitātes, gala lietotāja raksturiezīmes iezīmē produkta palaišanas brīža gala lietotāja raksturiezīmes.

### **2.1. Produkta perspektīva**

Produkta primārā funkcija ir pārbaudīt dažādu mikročipu (TTunit vai ControlUnit) funkcionalitātes stāvokli. Veicot šo pārbaudi, produkts nodrošina galējo rezultātu atbilstošam mikročipam norādot to komponenšu stāvokli, kā arī specifiskos gadījumos (skatīt 15.attēlu) norādīs pievienoto ierīču stāvokli. Pārbaudes beigās produkts nodrošina lietotājam veikt vairākkārtēju pārbaudes iespējamību.

Sekundārā funkcija, jeb daļa no kopēja produkta ir rīka konfigurēšana ar konfigurācijas aplikāciju. Aplikācija nodrošina divas funkcionalitātes – iepriekšējo mikročipu pārbaudes rezultātu vizualizēšanu uz ekrāna, kā arī saglabāto rezultātu izdzēšanu un iespēju pārmainīt rīka saskarnes valodu.

### **2.2. Sistēmas funkcionalās prasības**

#### **2.2.1. Rīka programmatūras funkcionalās prasības**

##### **P.1. Galvenā mikročipu pārbaudes funkcija**

Mērķis: Uzsākt mikročipu pārbaudes procesu un iegūt rezultātus.

Ievaddati: Uzsākt rīku ar kādu no mikročipiem, skatīt procesu pēc 5. nodaļas.

Apstrāde:

1. Pārbauda kādas ir pievienotās komponentes pie mikročipa;
2. Izveido sākuma kadru ar jauna kadra funkciju (skatīt P.4.);
3. Pārbauda kāds mikročips ir pievienots pie rīka;
4. Uz ekrāna izvada mikročipa pievienotās komponentes heksadecimālās vērtības;
5. Attiecīgi pēc pievienotā mikročipa veic pārbaudes funkcijas (skatīt P.5 - P.12);
6. Pēc mikročipu pārbaudes funkciju izpildes informē, lai lietotājs nospiež fizisku pogu rīka labajā pusē(skatīt 4.attēlu).



**4.attēls. Fiziskas pogas nospiešana**

Izvaddati:

1. Uz displeja tiek izvadīts sākuma kadrs attiecīgi no pievienota mikročipa (skatīt 5. attēlu);



**5.attēls. Sākuma kadrs uz displeja**

2. Uz displeja tiek izvadīts jauns kadrs mikročipa komponenšu adreses (skatīt 6.attēlu);



**6.attēls. Mikročipa komponenšu adreses kadrs**

3. Uz displeja tiek izvadīti attiecošie pārbaudes funkciju kadri, atkarībā no pievienotā mikročipa, skatīt pārējās funkcionālās prasības;
4. Uz displeja tiek izvadīts informatīvais kadrss, kas liecina, lai lietotājs vēlreiz nospiež pogu mikročipa pārbaudes apstiprināšanas nolūkiem.

## P.2. Pirms testa EEPROM pārbaude

Mērķis: Pārbaudīt mikročipa EEPROM komponenti pirms galvenā testa sākšanas priekš precīzas EEPROM komponentes pārbaudes.

Ievaddati: Uzsākt rīku, skatīt procesu pēc 5. nodaļas.

Apstrāde:

1. Ieraksta vērtību “test” EEPROM komponentes atmiņā;
2. Pārbauda vai ierakstītā vērtība sakrīt ar vērtību “test” nolasot no EEPROM komponentes atmiņas;
3. Atgriež vērtību, kas liecinātu vai EEPROM komponente precīzi raksta un nolasa datus.

Izvaddati: Tieka atgriezta vērtība attiecīgi EEPROM komponentes rakstīšanas, lasīšanas funkcionalitātei.

### **P.3. Pirms testa rīka un ControlUnit mikrokontrollieru komunikācijas pārbaude**

Mērķis: Pārbaudīt ControlUnit un rīka komunikāciju pirms galvenā testa sākšanas priekš precīzas ControlUnit un rīka komunikācijas pārbaudes.

Ievaddati: Uzsākt rīku, skatīt procesu pēc 5. nodaļas

Apstrāde:

1. Pievieno četru baitu komandu – “get”, “board\_version” četru baitu komandu sarakstam (skatīt P.10);
2. Nosūta četru baitu komandu uz ControlUnit mikrokontrolliera (skatīt P.11.);
3. Nolasa atbildi no ControlUnit mikrokontrolliera ar UART palīdzību
4. Atgriež nolasīto vērtību no ControlUnit mikrokontrolliera.

Izvaddati: Tieki atgriezta vērtība attiecīgi ControlUnit mikrokontrolliera atbildei.

### **P.4. Jauna kadra izveide**

Mērķis: Izvadīt uz displeja jauna kadra izveidi

Ievaddati: Progresējot caur programmai funkcija tiek izsaukta specifiskos brīžos ar specifisku indeksu (skatīt 1.tabulu).

**1.tabula**

Jauna kadra ievaddati	
Mainīgā nosaukums	Vērtība
Page	Cipars vai ‘res’ vai ‘pc’

Apstrāde:

1. Funkcijai izsaucoties pārbauda kāds mikročips ir pievienots pie rīka un attiecoši izvada uz ekrāna mikročipa nosaukumu;
2. Izveido InPass logo un to uz displeja izvada;
3. No dota indeksa izvada uz displeja testa numuru vai rezultātu, ja ir dots cipars kā indekss, tad uz displeja izvada “x. test” vai, ja ir dots “res” kā indekss tad uz displeja izvada “Results”, taču ja “pc”, tad uz displeja labā apakšēja stūra tiek izvadīts “Computer”.

Izvaddati:

1. Ja specifiskais indekss ir cipars un ir pievienots TTunit, tad izvada jaunizveidoto kadru uz displeja (skatīt 7. attēlu);



**7.attēls. Jauns kadrs ar cipara indeksu un pievienotu TTunit**

2. Ja specifiskais indekss ir "res" un ir pievienots ControlUnit, tad uz displeja izvada jaunizveidoto kadru (skatīt 8.attēlu);



**8.attēls. Jauns kadrs ar rezultāta indeksu un pievienotu ControlUnit**

## P.5. Real-Time-Clock komponentes pārbaude

Mērķis: Laut lietotājam pārbaudīt Real-Time-Clock komponentes stāvokli.

Ievaddati: Izpildīt P.1 un nospiezt fizisku pogu rīka labajā malā (skatīt 4.attēlu)

Apstrāde:

1. Galvenā mikročipu pārbaudes funkcija palaiž Real-Time-Clock pārbaudes funkciju;
2. Funkcijas process ieraksta Real-Time-Clock komponentes noteiktu datumu;
3. Funkcijas process nolasa no Real-Time-Clock komponentes iepriekš ierakstīto datumu;
4. Funkcijas process pārbauda vai nolasītais Real-Time-Clock komponentes datums sakrīt ar Real-Time-Clock komponentes ierakstīto datumu iekš programmā.

Izvaddati:



**9.attēls. Real-Time-Clock pārbaudes kadrs**

Real-Time-Clock komponentes pārbaudes laikā uz displeja tiek izvadīts kadrs (skatīt 9. attēlu).

## P.6. EEPROM komponentes pārbaude

Mērķis: Laut lietotājam pārbaudīt EEPROM komponentes stāvokli.

Ievaddati: Pabeigt P.5. darbības procesu;

Apstrāde:

1. Galvenā mikročipu pārbaudes funkcija (skatīt P.1.) palaiž EEPROM pārbaudes funkciju.
2. Funkcijas process ieraksta iekš EEPROM komponentes nejaušus datus katrā EEPROM komponentes lapā.
3. Funkcijas process nolasa no EEPROM komponentes iepriekš ierakstītos datus.

4. Funkcijas process pārbauda vai nolasītie EEPROM komponentes dati sakrīt ar EEPROM komponentes ierakstītiem datiem iekš programmā.

Ievaddati:



**10.attēls. EEPROM pārbaudes kadrss**

EEPROM komponentes pārbaudes laikā uz displeja tiek izvadīts kadrs (skatīt 10. attēlu).

## P.7. Rīka un ControlUnit komunikācijas pārbaude

Mērķis: Laut lietotājam pārbaudīt rīka un ControlUnit komunikācijas funkcionalitātes stāvokli.

Ievaddati: Pabeigt P.6. darbības procesu;

Apstrāde:

1. Galvenā mikročipu pārbaudes funkcija palaiž rīka un ControlUnit komunikācijas pārbaudes funkciju;
2. Funkcijas process palaiž citu funkciju, kas pārbauda ControlUnit mikrokontrolliera komunikācijas funkcionalitāti (skatīt P.9.);
3. Iepriekš minētā funkcija izvada vērtību, kas nosaka rīka un ControlUnit komunikācijas stāvokli;

4. Pēc izvadītās vērtības rīks liecina vai rīka un ControlUnit komunikācija ir ejoša vai neeoša un šo stāvokli ievada sarakstā priekš mikročipa pārbaudes rezultāta noteikšanas;
5. Izveido jaunu kadru (skatīt P.4.) un uz displeja izvada ka notiek rīka un ControlUnit komunikācijas pārbaude.

Ievaddati:



**11.attēls. Rīka un ControlUnit komunikācijas pārbaudes kadrs**

Rīka un ControlUnit komunikācijas pārbaudes laikā uz displeja tiek izvadīts kadrs (skatīt 11. attēlu).

## P.8. ControlUnit releju komponenšu pārbaude

Mērķis: Laut lietotājam pārbaudīt ControlUnit releju komponenšu stāvokli.

Ievaddati: Pabeigt P.7. procesu un jābūt pievienotam ControlUnit mikročipam pie rīka.

Apstrāde:

1. Rīks nosūta specifisku četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu “Turnstile1\_a” releju un uz displeja izvada releja nosaukumu.
2. Pēc “Turnstile1\_a” releja aktivizēšanas rīks nosūta nākamo četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu “Turnstile1\_b” releju un uz displeja izvada releja nosaukumu.
3. Pēc “Turnstile1\_b” releja aktivizēšanas rīks nosūta nākamo četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu “Turnstile2\_a” releju un uz displeja izvada releja nosaukumu.

4. Pēc "Turnstile2\_a" releja aktivizēšanas rīks nosūta nākamo četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu "Turnstile2\_b" releju un uz displeja izvada releja nosaukumu.
5. Pēc "Turnstile2\_b" releja aktivizēšanas rīks nosūta nākamo četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu "Button1" releju un uz displeja izvada releja nosaukumu.
6. Pēc "Button1" releja aktivizēšanas rīks nosūta nākamo četru baitu komandu uz ControlUnit mikrokontrollieri, lai aktivizētu "Button2" releju un uz displeja izvada releja nosaukumu.

Izvaddati: Tieki aktivizētas visas sešu releju darbības un ik katu releja pārbaudes brīdi uz displeja tiek izvadīts releja nosaukums.

### P.9. Rīka un ControlUnit mikrokontrollieru komunikācijas pārbaudes funkcija

Mērķis: Laut lietotājam pārbaudīt rīka un ControlUnit savstarpējo mikrokontrollieru komunikāciju.

Ievaddati: Funkcija tiek palaista iekļaujoties P.7. procesā.

Apstrāde:

1. Rīka mikrokontrollieris pievieno komandu sarakstam (skatīt P.8) četru baitu komandu (skatīt 2. tabulu);

**2.tabula**

<b>Ievaddati no rīka uz ControlUnit</b>	
Metode	Komanda
"get"	"board_version"

2. Nosūta doto četru baitu komandu uz ControlUnit mikrokontrollieru ar ControlUnit mikrokontrolliera komandu nosūtīšanas funkciju (skatīt P.9);
3. Pārbauda vai tiek saņemta atbilde no ControlUnit mikrokontrolliera un atgriež vērtību attiecīgi vai ControlUnit mikrokontrollieris atgriež atbildi.

Izvaddati: Tieki atgriezta vērtība attiecīgi ControlUnit mikrokontrolliera atbildei.

### P.10. Četru baitu komandu saglabāšana sarakstā

Mērķis: Laut lietotājam pievienot četru baitu komandu sarakstā.

Ievaddati: Funkcija tiek palaista iekļaujoties P.3. vai P.7. vai P.9 procesā.

Apstrāde: Sadala četru baitu komandu ik pa četriem baitiem (skatīt 3. tabulu) un to pievieno sarakstam.

Četru baitu sadalīšana			
Sekvence	Tips	Atslēga	Vērtība
Komandu izpildes izkārtojums indekss	“SIGNAL”	“turnstile1_a”, “turnstile1_b”, “turnstile2_a”, “turnstile2_b”, “button1”, “button2”	Cipars
Komandu izpildes izkārtojums indekss	“GET”	“board_version”	Cipars

Izvaddati: Tieki pievienota četru baitu komanda sarakstam.

### P.11. ControlUnit mikrokontrolliera komandu saraksta nosūtīšana

Mērkis: Laut lietotājam palaist komandu no rīka uz ControlUnit.

Ievaddati: Funkcija tiek palaista iekļaujoties P.3. vai P.7. vai P.9 procesā.

Apstrāde: Caur UART nosūta saglabāto četru baitu komandu sarakstu uz ControlUnit mikrokontrolliera.

Izvaddati: Tieki nosūtīta komanda uz ControlUnit mikrokontrolliera un tas to izpilda.

### P.12. Karšu lasītāju funkcionalitātes pārbaude uz ControlUnit

Mērkis: Laut lietotājam pārbaudīt karšu lasītāja funkcionalitāti četrās ControlUnit mikročipa adresēs.

Ievaddati:

1. Pabeigt P.9. procesu un jābūt pievienotam ControlUnit mikročipam;
2. Pārbaudīt pirmo ControlUnit mikročipa adresi un noskenēt karti, skatīt 5. nodaļu;
3. Nospiest fizisku pogu rīka labajā pusē (skatīt 4.attēlu);
4. Pārbaudīt otro ControlUnit mikročipa adresi un noskenēt karti, skatīt 5. nodaļu;
5. Nospiest fizisku pogu rīka labajā pusē;
6. Pārbaudīt trešo ControlUnit mikročipa adresi un noskenēt karti, skatīt 5. nodaļu;
7. Nospiest fizisku pogu rīka labajā pusē;
8. Pārbaudīt ceturto ControlUnit mikročipa adresi un noskenēt karti, skatīt 5. nodaļu;
9. Nospiest fizisku pogu rīka labajā pusē.

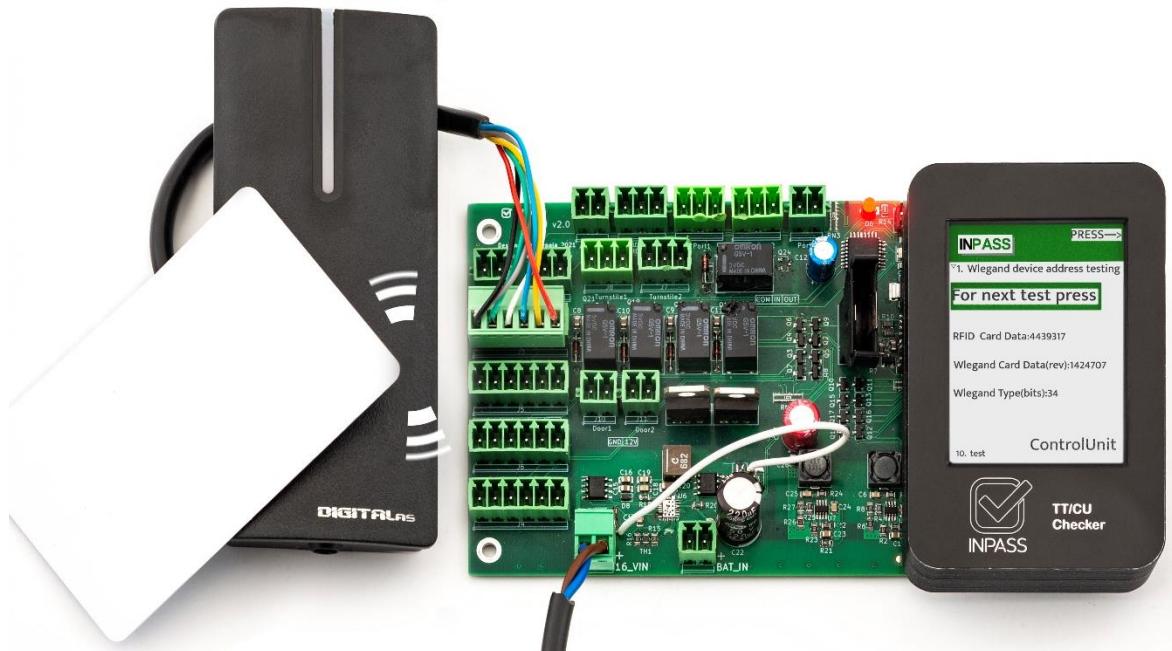
Apstrāde:

Šie četri punkti atkārtojas attiecīgi ControlUnit mikročipa četrām adresēm, jeb šie četri punkti atkārtosies četras reizes:



**12.attēls. Karšu lasītāja kadrss**

1. Uz rīka displeja tiek izvadīts karšu lasītāja kadrss (skatīt 12.attēlu);



**13.attēls. Karšu numura izvade uz displeja**

2. Pēc karšu skenēšanas uz displeja tiek izvadīts karšu numurs, apgriezts kartes numurs, kartes tips ar wiegand protokola palīdzību (skatīt 13. attēlu);
3. Pēc pogas nospiešanas tiek, atkarīgi vai karšu lasītājs ir nolasījis kartes informāciju, tiek saglabāts sarakstā karšu lasītāja adreses indekss, kas liecina vai karšu lasītājs nostrādājis.
4. Tieki izveidots nākošā kāršu lasītāja kadrss uz displeja.

Pēc ceturtās pogas nospiešanas reizes funkcija tiks pārtraukta.

Izvaddati: Uz displeja tiks izvadīta kartes informācija pēc katras karšu lasītāja adreses pārbaudes un tiks saglabāta karšu lasītāja funkcionalitātes stāvoklis sarakstā, attiecīgi vai karšu lasītājs nolasa informāciju.

### P.13. Testa rezultāta izvadīšana uz displeja

Mērkis: Laut lietotājam saprast mikročipa pārbaudes rezultātus.

Ievaddati:

1. Ja ir pievienots ControlUnit mikročips, tad pabeigt P.12. procesu;
2. Ja pievienots TTunit, tad pabeigt P.6. procesu;
3. Nospiest fizisku pogu rīka labajā malā (skatīt 4.attēlu).

Apstrāde:

1. Pārbauda kurš mikročips ir pievienots pie rīka;
2. Atkarīgi no pievienota mikročipa, ja ControlUnit, tad tiek aktivizēts signalizācijas stāvokļa relejs izmantojot P.8. un P.9. funkcijas, taču, ja TTunit, tad tiek aktivizēts zumers kurš atrodas uz TTunit mikročipa, lai liecinātu mikročipa pārbaudes rezultāta apstiprināšanu;
3. Tieki izveidots jauns kadrs (skatīt P.2.) ar ievaddatiem – “res”;
4. Tieki veikta pārbaude vai mikročipa pārbaude bijusi veiksmīga ar P.11. funkciju;



**14.attēls. Veiksmīga mikročipa pārbaudes rezultāta kadrs**

5. Ja mikročipa pārbaudes rezultāts ir veiksmīgs, tad uz displeja tiek izvadīts zaļā krāsā “The scanner test was successful!” un uz displeja tiek izvadīts visas mikročipa komponenšu pārbaudes rezultāti (skatīt 14.attēlu);



**15.attēls.** Neveiksmīga mikročipa pārbaudes rezultāta kadrs

6. Ja mikročipa pārbaudes rezultāts ir neveiksmīgs, tad uz displeja tiek izvadīts sarkanā krāsā “The scanner test was unsuccessful!” un uz displeja tiek izvadīts visas mikročipa komponenšu pārbaudes rezultāti (skatīt 15.attēlu);
7. Tieka dota iespēja lietotājam atkārtot mikročipa pārbaudi nospiežot fizisku pogu rīka labajā malā.

Izvaddati:

1. Lietotājs tiek informēts vai mikročipa pārbaudes rezultāts ir veiksmīgs vai neveiksmīgs;
2. Lietotājs tiek informēts kuras mikročipa komponentes darbojas korekti, kuras nekorekti;
3. Lietotājam dota iespēja atkārtot mikročipa pārbaudi.

#### **P.14. Mikročipa testa rezultāta pārbaudes funkcija**

Mērkis: Pārbaudīt vai testa process ir veiksmīgs vai neveiksmīgs.

Ievaddati: Progresēt cauri no P.1. līdz P.11. un iekš P.11 šī funkcija tiek palaista.

Apstrāde:

1. Pārbauda kāds mikročips ir pievienots pie rīka;
2. Pārbauda vai testa procesa laikā visi pievienoti dati sarakstā liecina ka katrs tests ir nosrādājis;
3. Izveido teksta struktūru, kura saglabā testa rezultātu failā.

Izvaddati: Funkcija atgriež vērtību, kas nosaka vai tests ir veiksmīgs vai neveiksmīgs un saglabā testa rezultātu failā.

## P.15. Mikročipa pārbaudes atkārtošanas sagatavošanas funkcija

Mērkis: Attīrīt visus datus par iepriekšējo mikročipa pārbaudi.

Ievaddati: Nospiest fizisku pogu rīka labajā malā (skatīt 4.attēlu) pēc P.11. izpildes.

Apstrāde: Attīra visas sarakstes un mainīgos, kas iepriekš glabāja datus par iepriekšējo mikročipa pārbaudi.

Izvaddati: Visas iepriekšējās mikročipa pārbaudes sarakstes un mainīgie tiek attīrīti.

## P.16. Komunikācija starp rīku un konfigurācijas aplikācijas funkcija

Mērkis: Veikt komunikāciju starp rīka programmatūru un konfigurācijas aplikāciju. Komunikācija sastāv no mikročipu pārbaudes rezultātu nosūtīšanas, dzēšanas un valodas maiņas apstiprināšana.

Ievaddati:

1. Pievienot rīku pie darbstacijas uz kuras ir uzinstalēta konfigurācijas aplikācija;
2. Nospiest fizisku pogu rīka labajā pusē;
3. Veiksmīgam funkcijas pielietojumam izpildīt P.20.

Apstrāde:

Funkcijai ir četri iespējami apstrādes scenāriji:

1. Konfigurācijas aplikācijai nosūtot caur seriālo komunikāciju “receive” rīks atpakaļ nosūta “transfer”, kas apstiprina savstarpējo komunikāciju un to, ka rīks veic mikročipu pārbaudes rezultātu datu plūsmu caur seriālo komunikāciju;
2. Konfigurācijas aplikācijai nosūtot “delete” rīks apstiprina un izpilda (skatīt 4.tabulu) datu dzēšanas procesu ar “deleted” nosūtīšanu atpakaļ caur seriālo komunikāciju;
3. Konfigurācijas aplikācijai nosūtot “lat” rīks apstiprina un izpilda (skatīt 4.tabulu) valodas maiņas procesu uz latviešu valodu ar “lat\_configured” nosūtīšanu atpakaļ;
4. Konfigurācijas aplikācijai nosūtot “eng” rīks apstiprina un izpilda (skatīt 4.tabulu) valodas maiņas procesu uz angļu valodu ar “eng\_configured” nosūtīšanu atpakaļ;

4.tabula

Datu maina attiecībā no sanemtiem datiem

Seriālās komunikācijas vērtība	Mainītie dati
“delete”	[“”, ””, ”start”]
“lat”	“conf-lat”
“eng”	“conf-eng”

Izvaddati:

1. Scenārijā, ja tiek saņemts “receive” no konfigurācijas aplikācijas, tad veiksmīgi tiek apspīrināta savstarpējā komunikācija un tiek nosūtīti esošie dati no rīka uz konfigurācijas aplikāciju;
2. Scenārijā, ja tiek saņemts “delete” no konfigurācijas aplikācijas, tad veiksmīgi tiek apspīrināta un izpildīta datu dzēšana no rīka;
3. Scenārijā, ja tiek saņemts “lat” no konfigurācijas aplikācijas, tad veiksmīgi tiek apspīrināta un izpildīta rīka valodas maiņa uz latviešu valodu;
4. Scenārijā, ja tiek saņemts “eng” no konfigurācijas aplikācijas, tad veiksmīgi tiek apspīrināta un izpildīta rīka valodas maiņa uz angļu valodu;

### **P.17. Rīka valodas maiņas funkcija**

Mērķis: Pārmainīt rīka saskarnes valodu attiecīgi faila datnei

Ievaddati: Uzsākt rīku ar kādu no mikročipiem vai pievienojot rīku pie darbstacijas, skatīt procesu 5. nodaļā.

Apstrāde: Nolasīt atbilstošo vērtību no “.json” faila un attiecīgi pārmaina mainīgo, kas atbildīgs par valodas izvadi (skatīt 5.tabulu).

**5.tabula**

<b>Valodas maiņas vērtību apstrāde</b>	
Vērtība no “.json” faila	Valodas maiņas mainīgā vērtība
“conf-lat”	“latvian”
“conf-eng”	“english”
None	“english”

Izvaddati: Veiksmīgi tiek atlasīta un nomainīta valoda rīka saskarnei.

## 2.2.2. Konfigurācijas aplikācijas programmatūras funkcionālās prasības

### P.18. Konfigurācijas aplikācijas loga izveidošanas funkcija

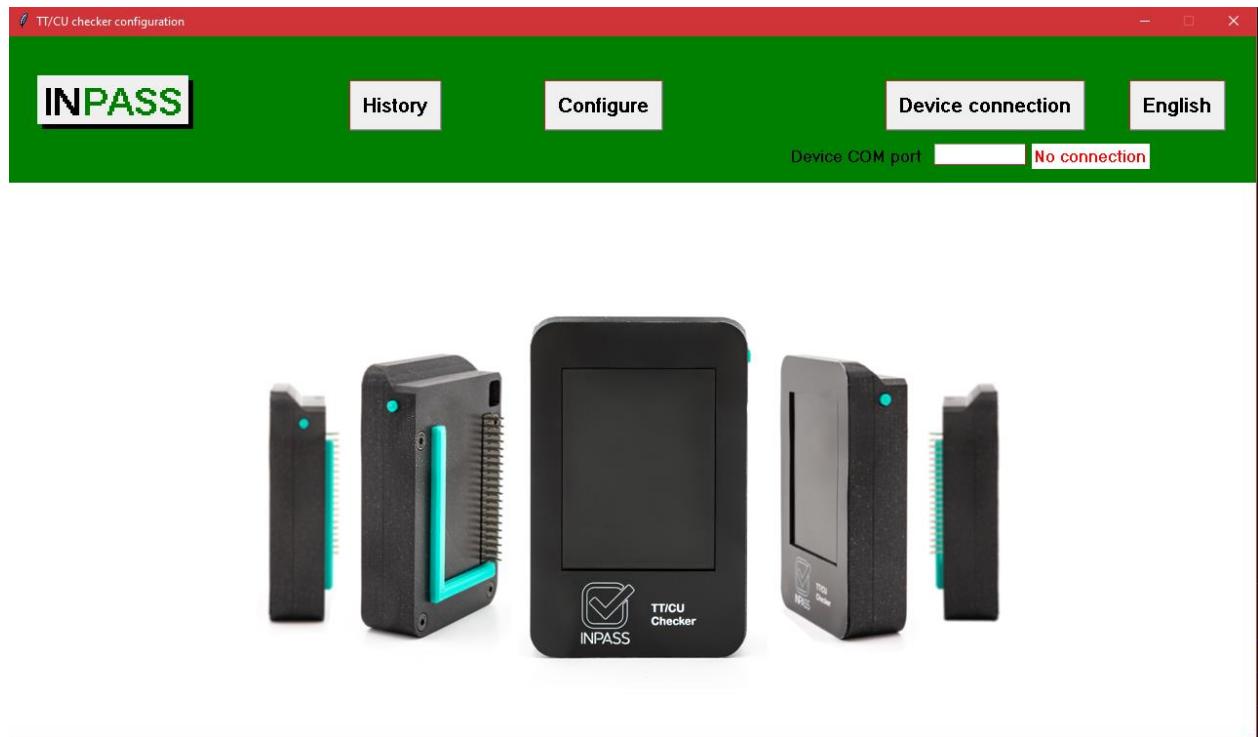
Mērķis: Izveidot konfigurācijas aplikācijas logu uz darbstacijas.

Ievaddati: Atvērt programmatūru “TT/CU checker configuration app.exe”.

Apstrāde:

1. Izveido logu uz ekrāna ar Tkinter bibliotēkas palīdzību;
2. Izveido navigācijas joslu uz izveidoto logu (skatīt P.19.);
3. Ja ar rīku nav iepriekš izveidots veiksmīgs savienojums, tad izvada tekstu, ka nav veiksmīgs savienojums, taču ja ir izveidots veiksmīgs savienojums, tad izvada pretējo.

Izvaddati: Tieki veiksmīgi izveidoti konfigurācijas aplikācijas logs (skatīt 16.attēlu).



16.attēls. Konfigurācijas aplikācijas logs

### P.19. Konfigurācijas aplikācijas navigācijas joslas izveidošanas funkcija

Mērķis: Izveidot konfigurācijas aplikācijas navigācijas joslu uz Tkinter loga.

Ievaddati: Atvērt programmatūru “TT/CU checker configuration app.exe”.

Apstrāde:

1. Izveido navigācijas joslas rāmi uz Tkinter loga;
2. Izvieto Tkinter pogas uz navigācijas joslas rāmja, piemēram, vēstures poga, konfigurācijas poga, ierīces savienojuma poga un valodas maiņas poga;
3. Izvieto ievades lauku, uz kura lietotājam būs iespējams ievadīt rīka savienoto seriālo portu, kā arī teksta lauku, kas norāda vai rīka komunikācija ir izveidota.

Izvaddati: Tieki veiksmīgi izveidoti konfigurācijas aplikācijas navigācijas logs uz Tkinter loga (skatīt 16.attēlu).

## P.20. Rīka un konfigurācijas aplikācijas komunikācijas savienojuma funkcija

Mērķis: Izveidot veiksmīgu seriālo komunikācijas savienojumu starp pievienoto rīku un konfigurācijas aplikāciju.

Ievaddati:

1. Izpildīt un progresēt cauri P.18;
2. Pievienot rīku pie darbstacijas, kur uzinstalēta konfigurācijas aplikācija un uzzināt rīka savienoto seriālo portu (skatīt 5.nodaļu);
3. Ievadīt zināmo seriālā porta ciparu vai nosaukumu, piemēram, ja seriālais ports ir “COM7”, tad ievadīt ievades laukā (skatīt 17.attēlu) “7” vai “COM7” vai “com7”;

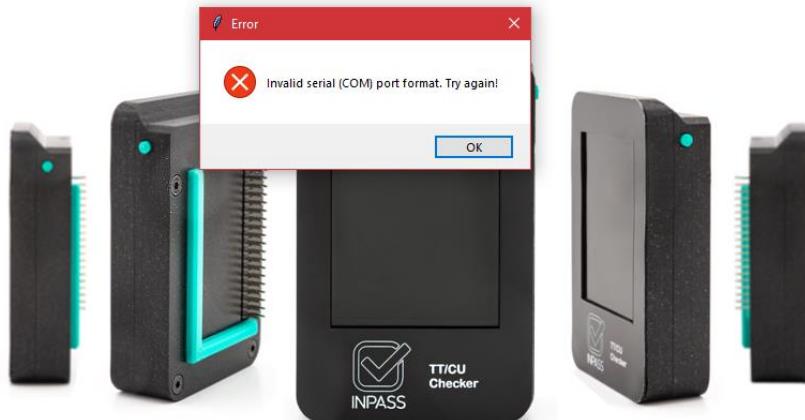


17.attēls. Ierīces savienojuma ievades lauks

4. Nospiest pogu “Device connection” vai “Ierīces savienojums” (atkārībā no izvēlētās valodas) ar kreiso peles klikšķi.

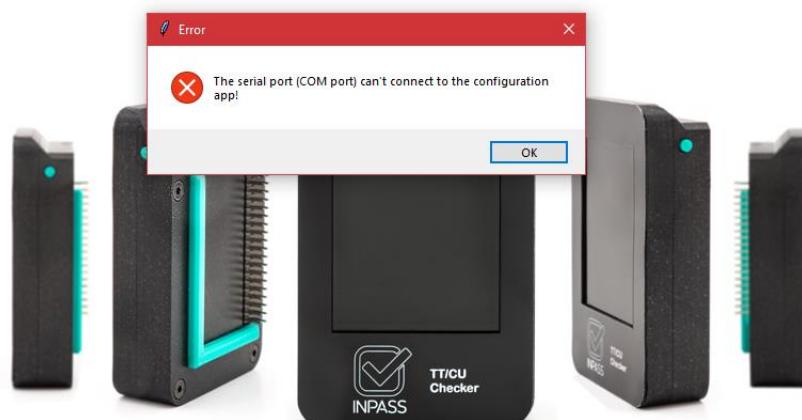
Apstrāde:

1. Tieki veikta seriālā porta vērtības validācijas pārbaude, ja validācija neveiksmīga, tad tieki izvadīts atsevišķs logs, kas ziņo lietotājam, ka formāts nav korekts (skatīt 18.attēlu);



**18.attēls. Nekorekta seriālā porta ievades logs**

2. Tieka nosūtīts teksts "receive" ar P.21. funkcijas palīdzību uz rīku, ja atbildē no rīka nav "b'transfer'", tad tiek izvadīts atsevišķs logs, kas ziņo, ka savienojuma process nav veiksmīgs (skatīt 19.attēlu) un apstādina funkcijas procesu;



**19.attēls. Neveiksmīga savienojuma procesa logs**

3. Ja atbildē no rīka ir "b'transfer'", tad tiek veikta seriālā lasīšana no rīka un ievietota sarakstē ar P.22. funkcijas palīdzību;
4. Tieka izvadīts teksts uz navigācijas joslas, kas ziņo, ka savienojums ir veiksmīgi izveidots (skatīt 20.attēlu).



20.attēls. Veiksmīga rīka savienojuma izveides teksts

Izvaddati: Tieki veiksmīgi izveidota seriālā komunikācija starp rīku un konfigurācijas aplikāciju (skatīt 20.attēlu).

### P.21. Datu nosūtīšana un lasīšana caur seriālo komunikāciju

Mērķis: Nosūtīt attiecošos datus uz ievadīto seriālo portu caur seriālo komunikāciju.

Ievaddati: Veicot P.20. vai P.24 vai P.26 funkcijas.

Apstrāde:

1. Programmatūras funkcionalitāte paredz četrus iespējamus seriālās komunikācijas datu nosūtīšanas scenārijus – “receive”, “delete”, “eng”, “lat”;
2. Funkcija nolasa atbildi no rīka un to padod tālāk sekojošam kodam.

Izvaddati: Veiksmīgi tiek nosūtīti un nolasīti paredzētie dati starp rīku un konfigurācijas aplikāciju.

### P.22. Saņemto mikročipu pārbaudes rezultātu datu apstrādes funkcija

Mērķis: Apstrādāt no rīka saņemtos mikročipu pārbaudes rezultātu datus un ievietot tos vārdnīcā tālākai izmantošanai.

Ievaddati: Progresēt cauri P.20. funkcijai.

Apstrāde:

1. Saskaņīt un sadalīt datnes katra elementu skaitu atbilstoši teksta pirmajiem diviem simboliem un papildināt atbilstoši datnes elementa skaitam vārdnīcu (skatīt 7.tabulu);

7.tabula

#### Vārdnīcas papildināšana ar datnes elementiem

Datnes elementa pirmie divi simboli	Vārdnīcas papildināšana atbilstoši datnes elementiem
“tt”	{“ttunit”: indekss:{}}}
“cu”	{“controlunit”: indekss:{}}}

2. Sadalīt un papildināt vārdnīcu ar datnes katra elementa vērtībām (skatīt 8.tabulu).

8.tabula

#### Vārdnīcas papildināšana ar sadalītiem saņemtiem datiem

Datnes elementa vērtība	Vārdnīcas papildināšana atbilstoši datnes elementu vērtībām
“tt-true:ee-true:rtc-true”	{“ttunit”: indekss:{“tt”: “true”, “ee”: “true”, “rtc”: “true”}}
“cu-true:ee-true:rtc-true:com-true:w1-true:w2-true:w3-true:w4-true”	{“controlunit”: indekss: {“cu”: “true”, “ee”: “true”, “rtc”: “true”, “com”: “true”, “w1”: “true”, “w2”: “true”, “w3”: “true”, “w4”: “true”}}}

Izvaddati: Veiksmīgi tiek apstrādi saņemtie mikročipu pārbaudes rezultātu dati no rīka.

### P.23. Mikročipu pārbaudes rezultātu vēstures rāmja izveidošanas funkcija

Mērķis: Izveidot mikročipu pārbaudes rezultātu vēstures rāmi un atbilstoši izvadīt rīka nosūtītos datus uz Tkinter loga

Ievaddati: Nospiežot pogu ar kreiso peles klikšķi uz navigācijas joslas ar nosaukumu “Vēsture” vai “History” (nosaukums atbilstoši no izvēlētās valodas).

Apstrāde:

1. Noņem no Tkinter loga visus liekos rāmjus, kuri atrastos vēstures rāmja ietvaros;
2. Izveido TTunit un ControlUnit rāmjus, kuros atradīsies mikročipu pārbaudes rezultātu vizualizēšanas tabulas;
3. Ja ir veiksmīgi izveidots savienojums starp aplikāciju un rīku, tad tiek sadalīti saņemto rezultātu datu vārdnīca un atbilstoši mikročipiem izveidotas tabulas (skatīt 9. tabulu un 21.attēlu);

**9.tabula**

**Vārdnīcas vērtību sadalīšana tabulas izveidei**

Vārdnīcas mikročipa elementa vērtība	Vārdnīcas papildināšana atbilstoši datnes elementiem
“true”	"✓"
“false”	"X"
None	"_"

TT/CU checker configuration																													
INPASS		History		Configure		Device connection		English																					
Device COM port com7 Connected: COM7																													
TTunit test history																													
<table border="1"> <thead> <tr> <th>N.</th> <th>Status</th> <th>EEPROM</th> <th>RTC</th> <th> </th> <th> </th> <th> </th> <th> </th> <th> </th> <th> </th> </tr> </thead> <tbody> <tr> <td>1.</td> <td>✓</td> <td>✓</td> <td>✓</td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> <td> </td> </tr> </tbody> </table>									N.	Status	EEPROM	RTC							1.	✓	✓	✓							
N.	Status	EEPROM	RTC																										
1.	✓	✓	✓																										
ControlUnit test history																													
<table border="1"> <thead> <tr> <th>N.</th> <th>Status</th> <th>EEPROM</th> <th>RTC</th> <th>UART</th> <th>1. Wieg</th> <th>2. Wieg</th> <th>3. Wieg</th> <th>4. Wieg</th> <th> </th> </tr> </thead> <tbody> <tr> <td>1.</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>Delete device history</td> </tr> </tbody> </table>									N.	Status	EEPROM	RTC	UART	1. Wieg	2. Wieg	3. Wieg	4. Wieg		1.	✓	✓	✓	✓	✓	✓	✓	✓	Delete device history	
N.	Status	EEPROM	RTC	UART	1. Wieg	2. Wieg	3. Wieg	4. Wieg																					
1.	✓	✓	✓	✓	✓	✓	✓	✓	Delete device history																				

#### 21.attēls. Mikročipu pārbaudes rezultātu rāmis

4. Tieka ievietota poga uz vēstures rāmja “Delete device history” vai “Dzēst ierīces vēsturi” (nosaukums atbilstoši no izvēlētās valodas).

Izvaddati: Veiksmīgi tiek izveidots vēstures rāmis un izveidotas mikročipu pārbaudes rezultātu tabulas.

#### P.24. Mikročipu pārbaudes rezultātu dzēšana no ierīces pogas nospiešanas funkcija

Mērķis: Veiksmīgi izdzēst mikročipu pārbaudes rezultātu datus no rīka.

Ievaddati: Izpildīt P.23. funkcijas procesu un nospiest pogu “Delete device history” vai “Dzēst ierīces vēsturi” ar kreiso peles klikšķi (nosaukums atbilstoši no izvēlētās valodas).

Apstrāde:

1. Ja rīka seriālā komunikācija ir veiksmīgi izveidota, tad tiek nosūtīts teksts “delete” uz rīku ar P.21. funkcijas palīdzību;
2. Ja atbilde no rīka ir “ b‘deleted’ ”, tad tiek izvadīts atsevišķs logs, kas signalizē, ka rīka mikročipu pārbaudes rezultātu vēsture ir izdzēsta (skatīt 22.attēlu);

Device COM port com7

N.	Status	EEPROM	RTC
1.	✓	✓	✓

N.	Status	EEPROM	RTC	UART	1. Wieg	2. Wieg	3. Wieg
1.	✓	✓	✓	✓	✓	✓	✓

## 22.attēls. Rīka vēstures dzēšanas paziņojuma logs

3. Tieki izsaukta P.20. funkcija, lai nolasītu izdzēstos datus no rīka.

Izvaddati: Tieki veiksmīgi izdzēsta mikročipu pārbaudes rezultātu datus no rīka.

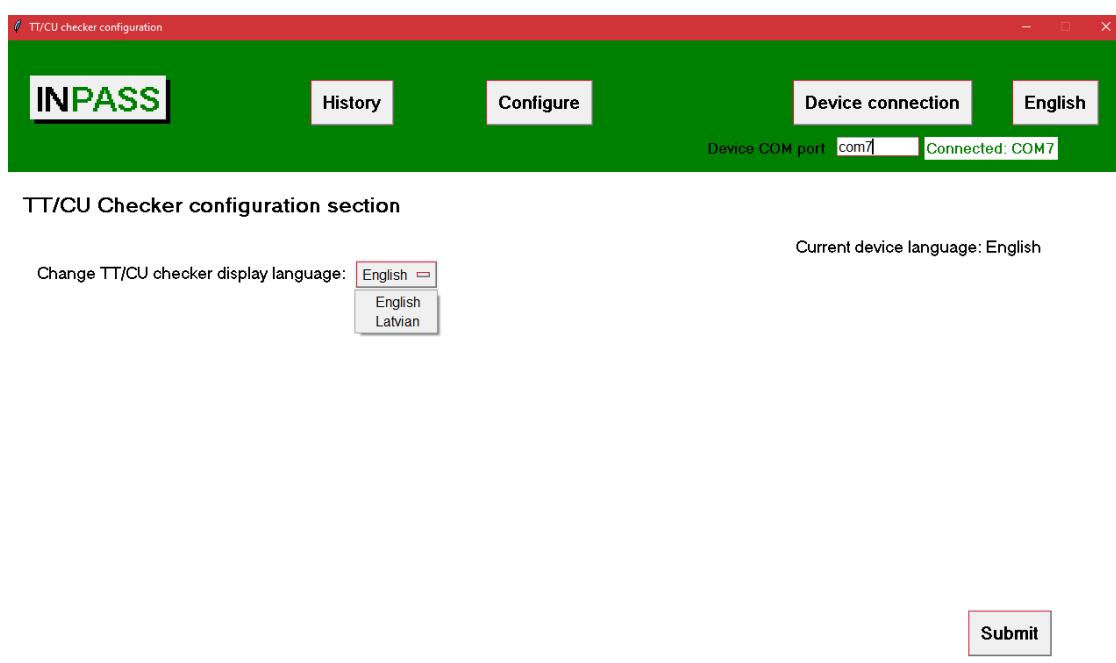
## P.25. Rīka konfigurācijas rāmja izveidošanas funkcija

Mērķis: Izveidot rīka konfigurācijas rāmi uz Tkinter loga

Ievaddati: Nospiežot pogu uz navigācijas joslas ar nosaukumu “Konfigurēt” vai “Configure” ar kreiso peles klikšķi (nosaukums atbilstoši no izvēlētās valodas).

Apstrāde:

1. Noņem no Tkinter loga visus liekos rāmjus, kuri atrastos konfigurācijas rāmja ietvaros;
2. Izveido konfigurācijas rāmi uz Tkinter loga;
3. Izvieto tekstu un nolaižamo logu uz konfigurācijas rāmja, kas dod iespēju lietotājam izvēlēties rīka saskarnes valodu (skatīt 23.attēlu);



## 23.attēls. Rīka konfigurācijas rāmis

- Izvieto pogu ar nosaukumu “Submit” vai “Apstiprināt” (nosaukums atbilstoši no izvēlētās valodas), kas ļauj lietotājam apstiprināt un nosūtīt uz rīka valodas maiņas nepieciešamību.

Izvaddati: Veiksmīgi tiek izveidots rīka konfigurācijas rāmis uz Tkinter loga.

### P.26. Rīka valodas maiņas apstiprināšanas pogas nospiešanas funkcija

Mērķis: Veiksmīgi mainīt rīka saskarnes valodu.

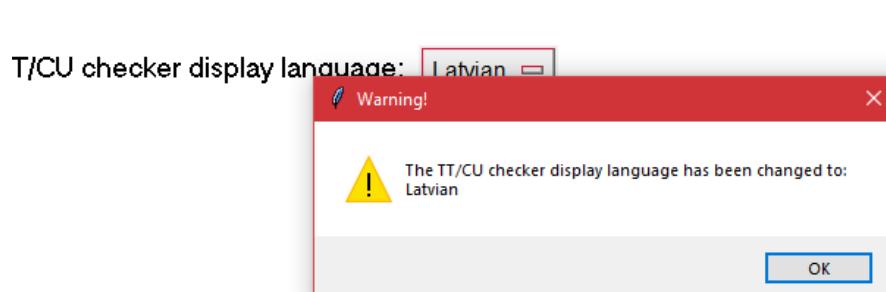
Ievaddati:

- Izpildīt P.25. funkcijas procesu;
- Izvēlēties vēlamo rīka saskarnes valodu nolaižamā logā – latviešu vai angļu;
- Nospiest pogu “Apstiprināt” vai “Submit” ar kreiso peles klikšķi (nosaukums atbilstoši no izvēlētās valodas).

Apstrāde:

- Ja rīka seriālā komunikācija ir veiksmīgi izveidota, tad tiek nosūtīts uz rīku, ar P.21. funkcijas palīdzību, teksts “lat” vai “eng” atkarībā no izvēlētās valodas nolaižamā logā;
- Ja atbilde no rīka ir “ b‘lat-configured’ ” vai “ b‘eng-configured’ ”, tad tiek izvadīts atsevišķs logs, kas signalizē, ka rīka mikročipu saskarnes valoda ir mainīta (skatīt 24.attēlu);

## checker configuration section



### 24.attēls. Rīka valodas maiņas pazīnojuma logs

- Tiek izsaukta P.20. funkcija, lai nolasītu jaunās valodas maiņas datus no rīka.

Izvaddati: Tieki veiksmīgi mainīta rīka saskarnes valoda.

### P.27. Aplikācijas saskarnes valodas maiņas funkcija

Mērķis: Pārmainīt rīka konfigurācijas aplikācijas saskarnes valodu.

Ievaddati: Nospiežot pogu “English” vai “Latviešu” ar kreiso peles klikšķi (atkarīgs no sākotnējo valodas izvēli).

Apstrāde:

- Ja iepriekšējā aplikācijas saskarnes valodas mainīgais ir “latvian”, tad saskarnes valodas mainīgais tiek pārmainīts uz “english”;
- Ja iepriekšējā aplikācijas saskarnes valodas mainīgais ir “english”, tad saskarnes valodas mainīgais tiek pārmainīts uz “latvian”.

Izvaddati:

- Ja iepriekšējā aplikācijas saskarnes valoda ir latviešu, tad saskarnes valoda tiek pārmainīta uz angļu valodu un nospiestā aplikācijas saskarnes valodas maiņas pogas nosaukums tiek pārmainīts uz “English”;
- Ja iepriekšējā aplikācijas saskarnes valoda ir angļu, tad saskarnes valoda tiek pārmainīta uz latviešu valodu un nospiestā aplikācijas saskarnes valodas maiņas pogas nosaukums tiek pārmainīts uz “Latviešu”;

## 2.3. Sistēmas nefunkcionālās prasības

- 1) Testēšanas rīka izstrādes procesa dokumentācijai ir jābūt noformētai un izstrādātai atbilstoši Latvijas Valsts standartam LVS 68:1996;
- 2) Saskaņei jābūt ērtai un ergonomiskai, lai ātrā laika posmā lietotājs spētu pārbaudīt mikročipu;
- 3) Testēšanas rīka displeja dominējošās krāsas – zaļa un balta;
- 4) Testēšanas rīka displeja izmantotā valoda – Angļu valoda ar dažādiem izmantotiem profesionālismiem;
- 5) Rīka programmatūrai jābūt izstrādātam Python programmēšanas valodā un rīkam jātiekt palaistam uz Micropython programmaparatūru;
- 6) Konfigurācijas aplikācijas programmatūra jātiekt palaistai, kā “.exe” failam.

## 2.4. Gala lietotāja raksturiezīmes

Testēšanas rīka programmatūras galvenā funkcija ir pārbaudīt mikročipus TTunit un ControlUnit. Tā kā mikročipus veido uzņēmums “SIA InPass”, tos nav iespējams iegādāties tirgū un vienīgie gala lietotāji būs uzņēmuma darbinieki. Pēc šī fakta, to pašu var teikt par konfigurācijas aplikāciju gala lietotāja atbilstību. Var secināt, ka gala lietotājs būs tehniskais speciālists ar vajadzību pārbaudīt mikročipus.

Gadījumā, ja pasūtītājs būs nolēmis, ka testēšanas rīks un konfigurācijas aplikācija tiek tirgota plašajā tirgū vai testēšanas rīks tiek papildināts ar vēl vairākām funkcionalitātēm, tad testēšanas rīka izstrādes procesa dokumentācija jāpapildina.

### **3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums**

Šajā nodaļā aprakstīti produkta izstrādes līdzekļi, dažādie izmantotie rīki, kā arī to izvēles pamatojumu.

#### **3.1. Izvēlēto risinājuma līdzekļu un valodu apraksts**

##### **3.1.1. Izstrādes programmatūras**

Izstrādes laikā tika izmantotas dažādas programmatūras. Katras programmatūras unikālās īpašības ļāva izstrādātājam ātri un efektīvi atrisināt dažādus izstrādes strupceļus. Šīs ir izmantotās izstrādes programmatūras:

**Thonny** - integrētā izstrādes vide (IDE), kas ir paredzēta galvenokārt Python programmatūrām. Šī IDE ir vienkārša, intuitīva un viegli lietojama, tādēļ lielākoties rīka programmatūras izstrādei tika izmantota šī izstrādes vide. Šeit ir programmatūras izvēles pamatojuma punkti:

- Vienkārša un tīra lietotāja saskarne, kas darba procesā izstrādātajam sniedza vieglu sadarbību ar rīku un darbstaciju.
- Iebūvēts interpretatora atbalsts ir liels iemesls kādēļ tika izmantota tieši šī programmatūra, tas ļāva izstrādātājam vieglu maiņu starp micropython programmaparatūrām.
- Klūdu atklāšana un labošana nodrošināja izstrādātājam efektīvi un ātri izprast izstrādes strupceļus, jo programmatūra sniedza atšifrētus rīka klūmes.
- Programmatūras atbalsts vairākām operētājsistēmām deva izstrādātājam ērtu programmatūras izmantošanu ar Windows un Linux operētājsistēmām, jo izstrādes laikā izstrādātājam strupceļu gadījumos bija nepieciešams izmantot vairākas operētājsistēmas.

**Visual Studio Code** - integrētā izstrādes vide (IDE), kas ir paredzēta dažādām programmēšanas valodām un programmatūras izstrādes vajadzībām. Šī IDE ir jaudīga, pielāgojama un viegli lietojama, tādēļ konfigurācijas aplikācijas programmatūrai tiek izmantota tieši šī izstrādes vide. Šeit ir programmatūras izvēles pamatojuma punkti:

- Jaudīga un pielāgojama lietotāja saskarne, kas darba procesā izstrādātajam sniedz efektīvu un ērti pielāgojamu darba vidi, nodrošinot dažādus paplašinājumus.
- Iebūvēts Git atbalsts ir liels iemesls kādēļ izstrādātājs izmantoja tieši šo programmatūru, jo tas ļauj izstrādātājam viegli pārvaldīt versiju kontroli.

- InteliSense funkcionalitāte izstrādātājam nodrošina ātrāku un precīzāku kodēšanu, piedāvājot automātiskos pabeigšanas ieteikumus un sintakses klūdu pārbaudi reällaikā.

**PuTTY** - bezmaksas un atvērtā koda termināla emulators un tālvadības pieteikums, kas galvenokārt tiek izmantots SSH, Telnet un seriālo savienojumu veidošanai un pārvaldīšanai ar citiem datoriem vai tīkla ierīcēm. Šeit ir programmatūras izvēles pamatojuma punkti:

- Seriālais savienojums ar rīku un darbstaciju izstrādātājam atviegloja informācijas izvadi no rīka testēšanas etapa.
- Portabilitāte un viegla instalācija starp rīka USB savienojuma un darbstacijas sniedza izstrādātājam ērti izveidot seriālo savienojumu bez liekiem strupceļiem.

**Arduino IDE** - integrētā izstrādes vide, kas ir paredzēta Arduino mikrokontrollieru programmēšanai un projektu izstrādei. Izstrādātāja pamatojums programmatūras izmantošanai bija strupceļa gadījumā izmantojot Arduino mikrokontrollieri to atrisināšanai.

**Git** - izplatīts versiju kontroles sistēma, kas izmantojama programmēšanas projektu vadīšanai un kodu sadarbībai starp dažādiem izstrādātājiem. Tā ir atvērtā koda un bezmaksas, un tās izmantošana ir kļuvusi par standarta praksi programmēšanas nozarē. Šeit ir programmatūras izvēles pamatojuma punkti:

- Distribuēta versiju kontrole izstrādātājam ļāva versionēt produkta izstrādes gaitu un strupceļa gadījumos atgriezt atpakaļ iepriekšējo produkta versiju.
- Atzaru un apvienošanas modelis izstrādātājam atviegloja produkta koda daļu apvienošanu ar dažādiem izstrādes zariem.
- Versiju viegla pārvaldīšana izstrādātājam sniedza vispārējo pārlūku starp iepriekšējo produkta versiju izmaiņām.

### 3.1.2. Izstrādes darbstacija

Izstrādes darbstacija nodrošina visu nepieciešamo produkta izstrādē. Šeit minētās ierīces ir izstrādātāja darbstacijas komplekts :

**Lenovo ThinkPad X230** – portatīvais dators, kuru izstrādātājs izmantoja gatavā produkta izstrādei. Komplektācijā arī portatīvā datora lādētājs. Ierīci nodrošināja prakses vieta projekta realizācijai.

**Logitech M90 datora pele** - ierīce, kas ļāva izstrādātājam daudz ērtāk un ergonomiskāk veikt programmatūras izstrādi. Ierīci nodrošināja prakses vieta.

**Arduino Nano ESP32** – mikrokontrolliera ierīce ar kuru ir iespējams veidot dažādus programmēšanas projektus. Izstrādātājs izmantoja šo ierīci, lai atrisinātu izstrādes ceļā izveidojušo strupceļu saistībā ar vairāku mikrokontrollieru savstarpējo sazināšanos.

**USB-A uz Micro-USB kabelis** – galvenā izstrādātāja aparatūra, kas izveido savienojumu starp portatīvo datoru un rīku. Kabelis arī darbojas kā savienotājs, nodrošinot strāvu rīkam un TTunit mikročipam.

### 3.1.3. Produkta komponentes un ierīces

Produkta komponentes un ierīces ir vispārējais aparatūras komplekts, kas izveido pašu rīku. Šis ir aparatūras komplekts :

**Rīka mikrokontrolliera ierīce** – izstrādātāja prakses vietas “SIA InPass” pašu izveidotā ierīce ar RP2040 mikrokontrolliera komponenti. Ierīcē tiek uzstādīta programmatūra, radot gatavu produktu (skatīt 23.attēlu).



23.attēls. Rīka mikrokontrolliera ierīce

**ILI9341 2.8 collu displejs** – komponente, kas nodrošina rīka saskarni un mikročipu pārbaudes informācijas izvadi. Komponentes priekšrocība ir tās slaidais izmērs un kvalitāte, kura vispiemērotāk atbilst produkta prasībām.

### 3.1.4. Programmatūras programmēšanas valodas un bibliotēkas

Precīzai programmatūras izpildei un funkcionalitātei liela būtība ir tās izmantotai programmēšanas valodai un bibliotēkām, kas nodrošina funkcionālu un ātru programmatūras izpildi. Šeit tiek minētas izmantotās izstrādes programmatūras valodas un bibliotēkas :

**Python** – programmēšanas valoda ar kuru izstrādātājs izgatavoja rīka programmatūru. Programmēšanas valodas izvēle ir pasūtītāja prasība, izrietot no šīs programmēšanas valodas izvēles ir nosacījums izmantot Python programmēšanas valodas balstīto Micropython programmaparatūru.

**Micropython** – programmaparatūra ar kuras palīdzību tiek palaista izstrādātā programmatūra. Pati programmaparatūra tiek uzstādīta uz rīka mikrokontrolliera. Programmaparatūra tiek lietota balstoties uz pasūtītāja prasībām.

**Wiegand** – protokola bibliotēka, kuru izmanto karšu lasītāja pārbaudes rīka darbības procesā. Bibliotēka tiek uzstādīta rīka mikrokontrolliera ierīcē un to nodrošina prakses vietas

izveidotā bibliotēka. Šī bibliotēka ir vispiemērotākā programmatūrai, jo tiek pārbaudītas “SIA InPass” izmantotie karšu lasītāji.

**EEPROM** – bibliotēka, kas ar tās funkcijām ātri un efektīvi ļauj izmantot EEPROM komponenti. Bibliotēkas bezmaksas piekļuves izmantošana un efektīvās iebūvētās funkcijas attaisno tās pielietojumu.

**ILI934xnew** – bibliotēka ar kuru tiek izveidota displeja saskarne. Bibliotēkas iebūvētās funkcijas izstrādātājs veiksmīgi pielietoja, izstrādājot saskarni, kura nodrošina ātru renderēšanu.

**Fonta izveides bibliotēkas** – bibliotēku kopums ar kuras palīdzību tiek renderēts teksts uz displeja un efektīvi tiek mainīti šie teksta fonta izmēri, krāsas, novietojums.

**Machine bibliotēka** – dažādu funkciju kopums, kas iebūvēts Micropython programmaparatūrā, ar kuras palīdzību programmatūra spēj izmantot digitālās tapas uz rīka mikrokontrolliera ierīces, radot seriālo komunikāciju starp citiem mikrokontrollieriem. Bibliotēkas efektivitāte padara to par vispiemērotāko rīka programmatūras konkrētajām vajadzībām.

**Tkinter bibliotēka** – dažādu funkciju kopums, kas iebūvēts Python standarta bibliotēkā, ar kuras palīdzību programmatūra spēj veidot grafiskās lietotāja saskarnes (GUI) uz dažādām platformām. Tkinter piedāvā plašu logrīku klāstu, jeb interfeisa elementus, ļaujot izstrādātājam viegli izveidot konfigurācijas aplikācijas programmatūru. Bibliotēkas vienkāršība padara to par vispiemērotāko konfigurācijas aplikācijas programmatūras konkrētajām vajadzībām.

### **3.2.Iespējamo risinājuma līdzekļu un valodu apraksts**

Programmatūru izstrāde ir plašs un daudzveidīgs process, kurā iespējams izmantot vairākus līdzekļus, komponentes, ierīces, programmēšanas valodas un bibliotēkas. Tomēr, projekta izstrādes iespējas ir ierobežotas no pasūtītāja noteiktajām prasībām. Potenciāli var tikt izmantotas citas efektīvākas bibliotēkas, kas uzlabotu programmatūras kvalitāti un veiktspēju. Iespējams, izmantot citu grafiskā interfeisa bibliotēku konfigurācijas aplikācijai. Vienlaikus, var arī uzstādīt jaunākās Micropython programmaparatūras versijas mikrokontrollierī, kas nodrošinātu ātrāku un efektīvāku veiktspēju, kā arī izmantot citus programmēšanas algoritmus, paņēmienus un datu struktūras, kas atļautu programmatūrai būt efektīvākai.

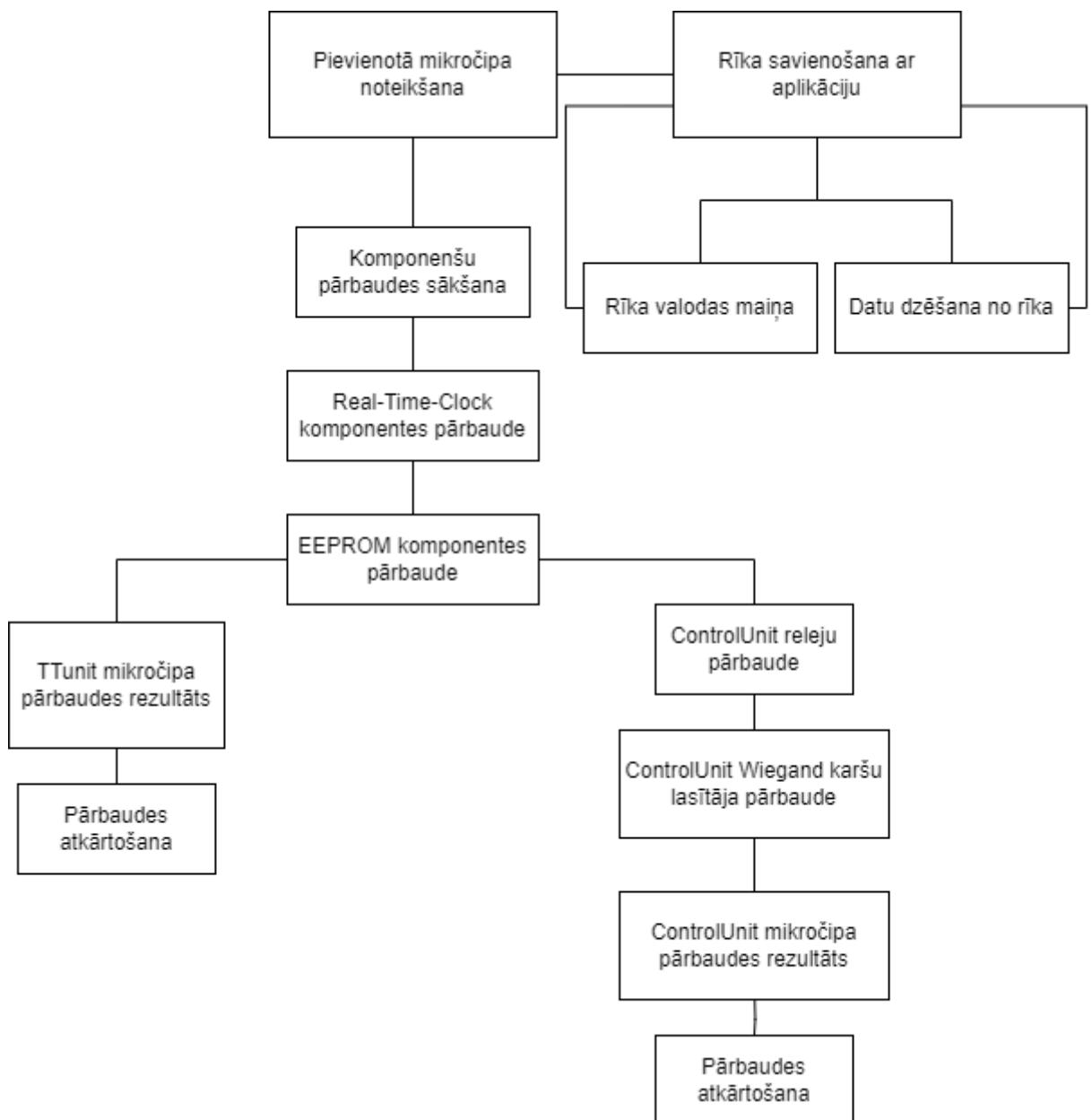
Pasūtītāja prasības neierobežo darbstaciju, tādēļ, izstrādes procesā izstrādātājs potenciāli varēja izmantot alternatīvas darbstacijas ierīces un izstrādes vides, piemēram, jaunākus un efektīvākus programmēšanas rīkus vai modernāku datoru aprīkojumu. Šāda izvēle būtu rezultējusi uz īsāku izstrādes laiku un ērtāku, efektīvāku izstrādes procesu.

## 4. Sistēmas modelēšana un projektēšana

Šajā nodaļā aprakstīta sistēmas modelēšana un projektēšana, precīzāk, sistēmas struktūras modeli, klašu diagrammas un aktivitāšu diagramma.

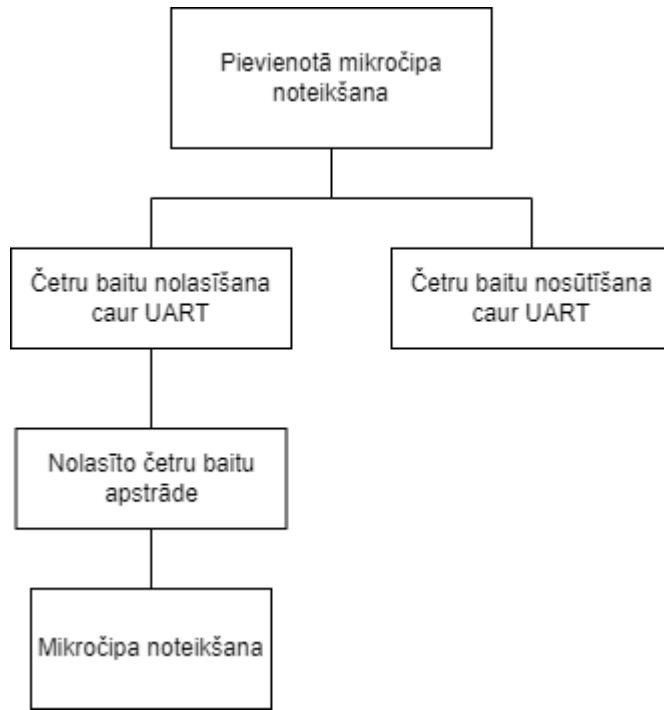
### 4.1. Rīka sistēmas struktūras modelis

Apakšnodaļā vizualizēts vispārējais sistēmas struktūras modelis un katrs redzamais bloka elements (skatīt 24. attēlu) tiek sīkāk sadalīts citos bloka elementos.



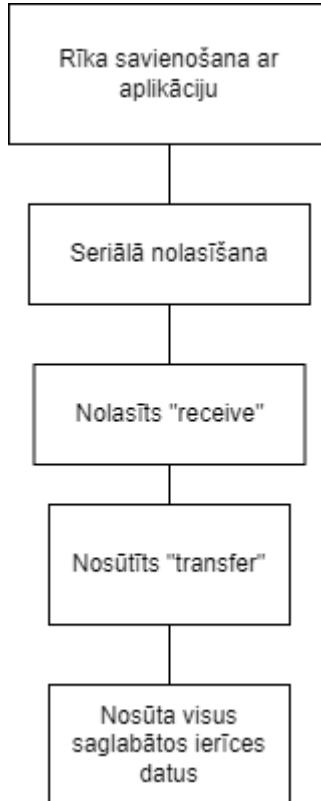
24. attēls. Rīka sistēmas struktūra

#### **4.1.1. Pievienota mikročipa noteikšana**



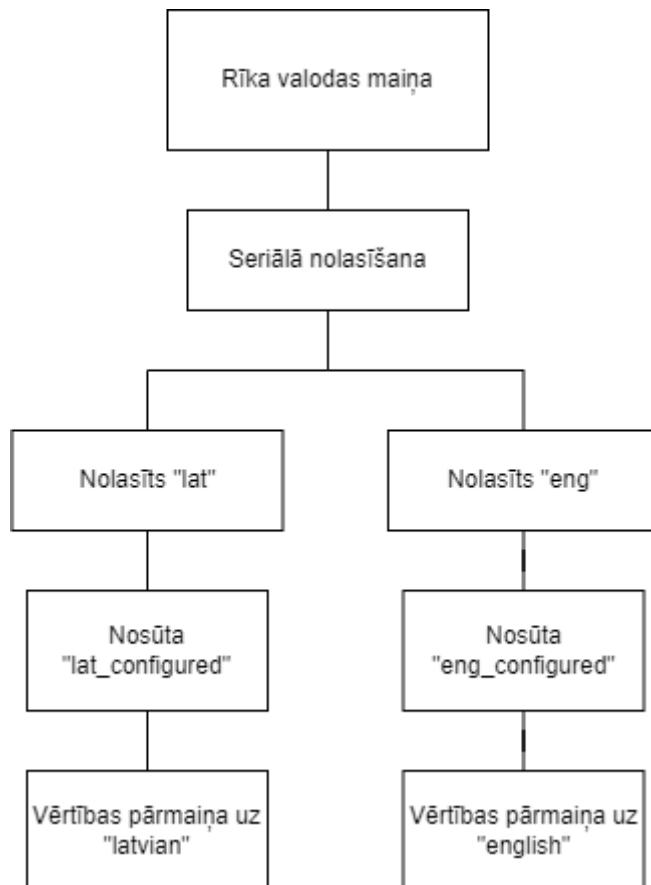
**25. attēls. Pievienota mikročipa noteikšana**

#### **4.1.2. Rīka savienošana ar aplikāciju**



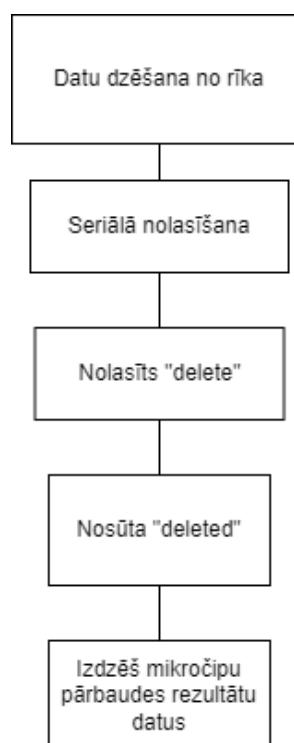
**26.attēls. Rīka savienošana ar aplikāciju**

#### **4.1.3. Rīka valodas maiņa**



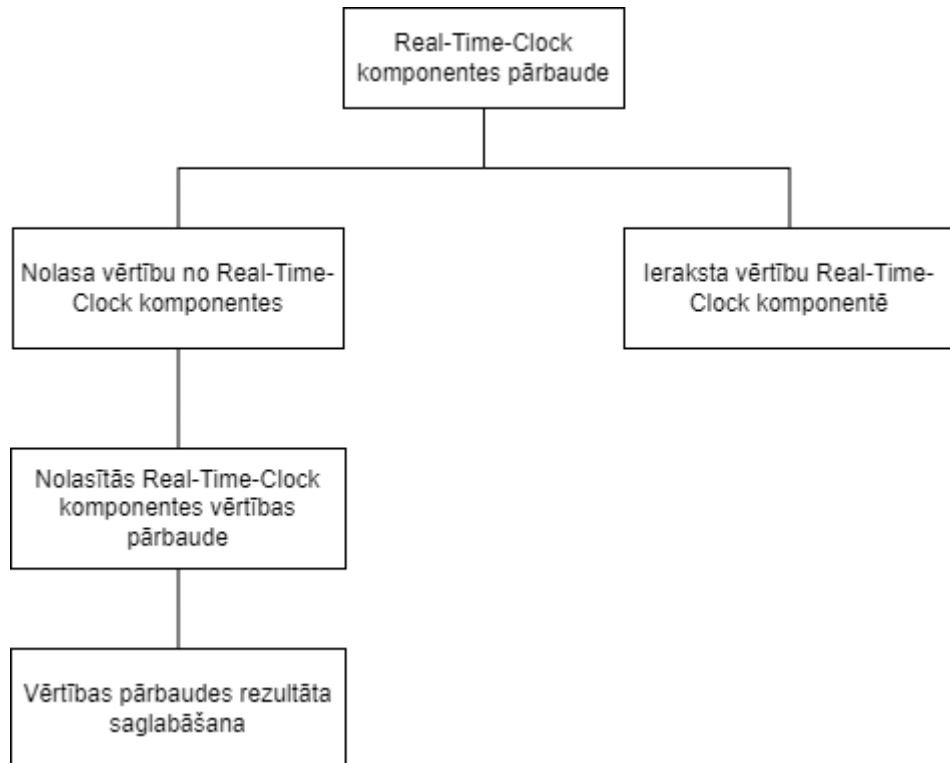
**27.attēls. Rīka valodas maiņas**

#### **4.1.4. Datu dzēšana no rīka**



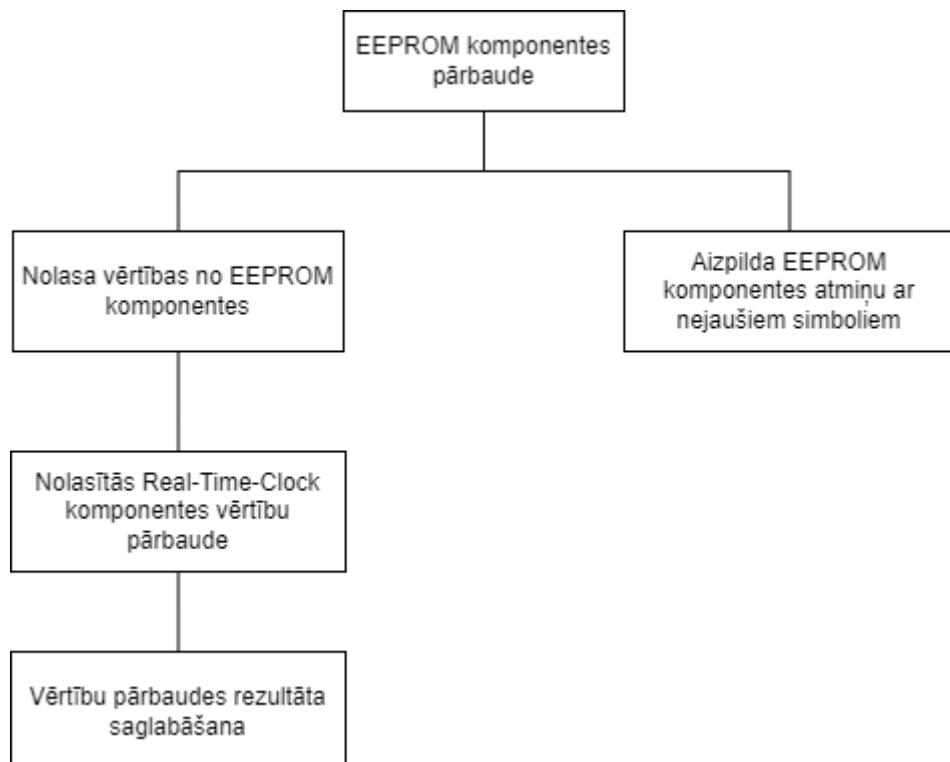
**28.attēls. Datu dzēšana no rīka**

#### 4.1.5. Real-Time-Clock komponentes pārbaude



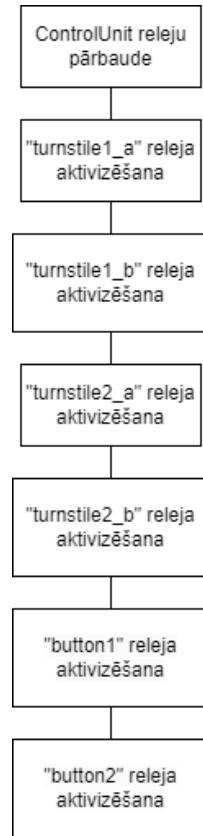
29.attēls. Real-Time-Clock komponentes pārbaude

#### 4.1.6. EEPROM komponentes pārbaude



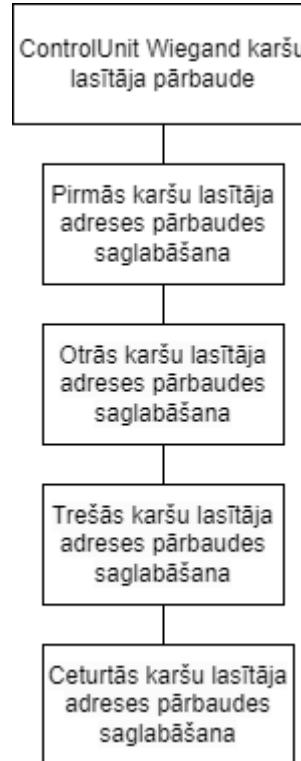
30.attēls. EEPROM komponentes pārbaude

#### **4.1.7. ControlUnit releju pārbaude**



**31. attēls. ControlUnit releju pārbaude**

#### **4.1.8. ControlUnit Wiegand karšu lasītāja pārbaude**



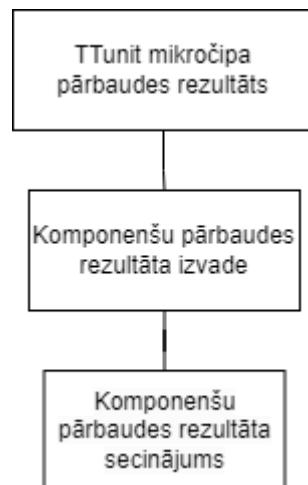
**32. attēls. ControlUnit Wiegand karšu lasītāja pārbaude**

#### **4.1.9. ControlUnit mikročipa pārbaudes rezultāts**



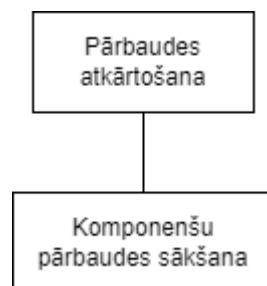
**33. attēls. ControlUnit mikročipa pārbaudes rezultāts**

#### **4.1.10. TTunit mikročipa pārbaudes rezultāts**



**34. attēls. TTunit mikročipa pārbaudes rezultāts**

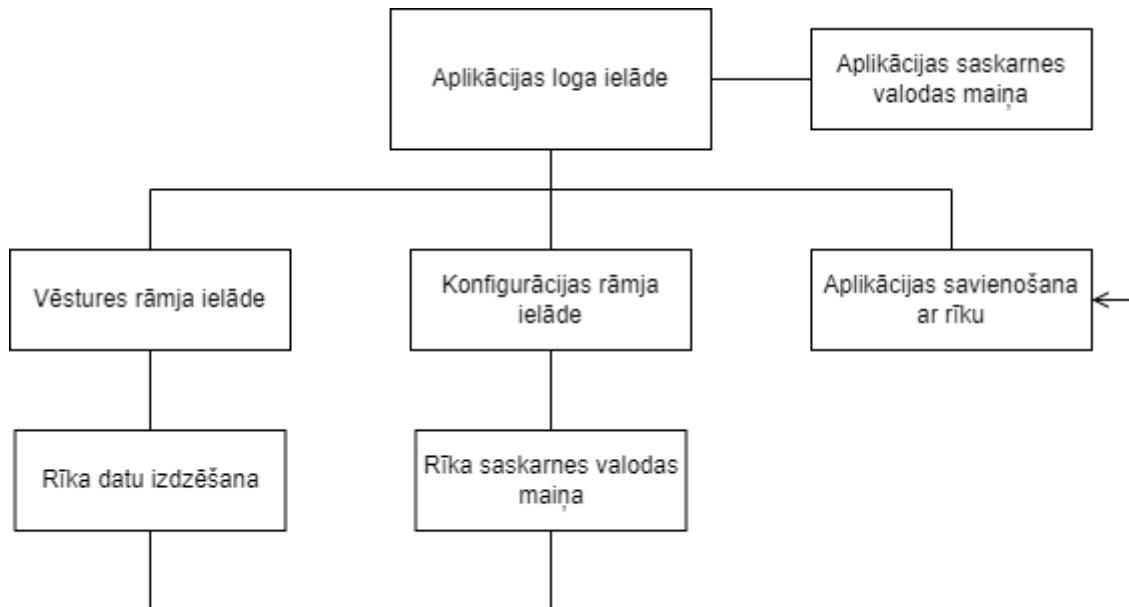
#### **4.1.11. Pārbaudes atkārtošana**



**35. attēls. Pārbaudes atkārtošana**

## **4.2.Konfigurācijas aplikācijas sistēmas struktūras modelis**

Apakšnodalā vizualizēts vispārējais sistēmas struktūras modelis un katrs redzamais bloka elements (skatīt 36. attēlu) tiek sīkāk sadalīts citos bloka elementos.



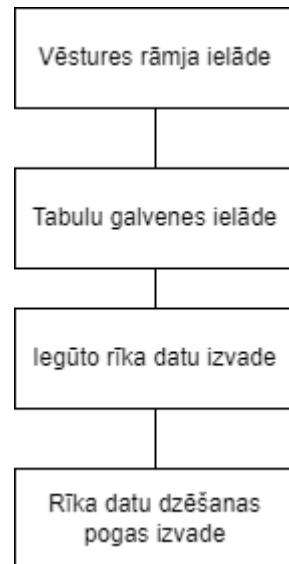
36.attēls. Konfigurācijas aplikācijas sistēmas struktūra

### **4.2.1. Aplikācijas loga ielāde**



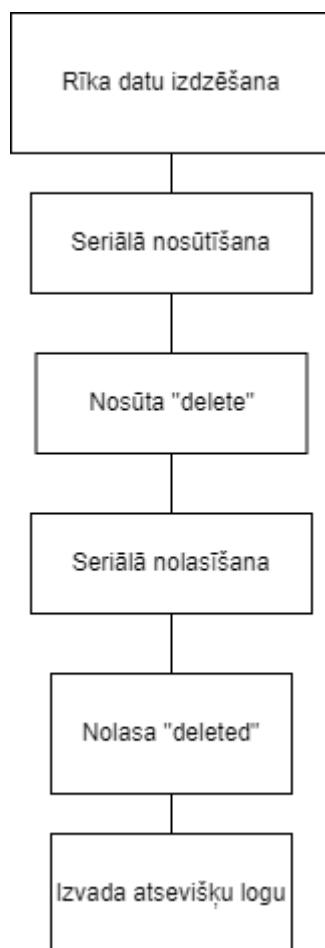
37.attēls. Aplikācijas loga ielāde

#### **4.2.2. Vēstures rāmja ielāde**



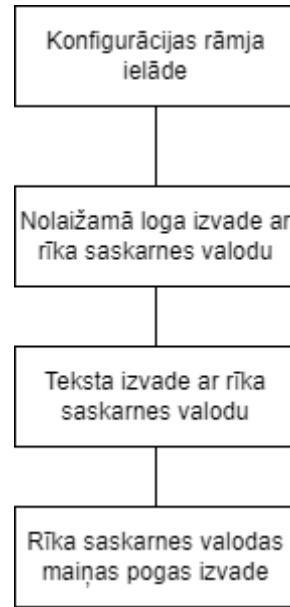
38.attēls. Vēstures rāmja ielāde

#### **4.2.3. Rīka datu izdzēšana**



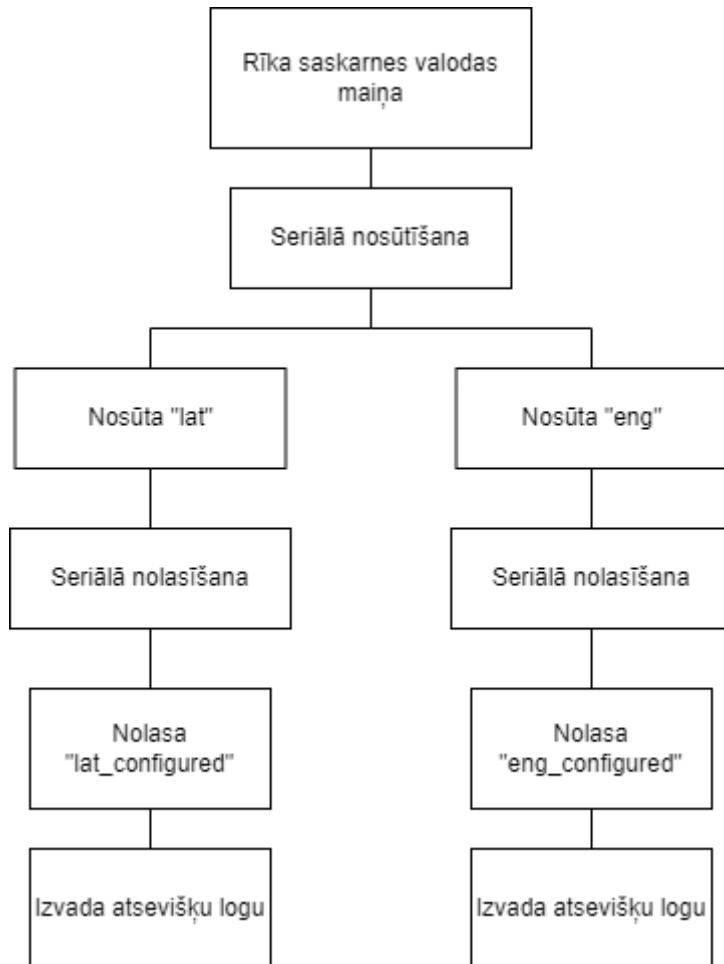
39.attēls. Rīka datu izdzēšana

#### **4.2.4. Konfigurācijas rāmja ielāde**



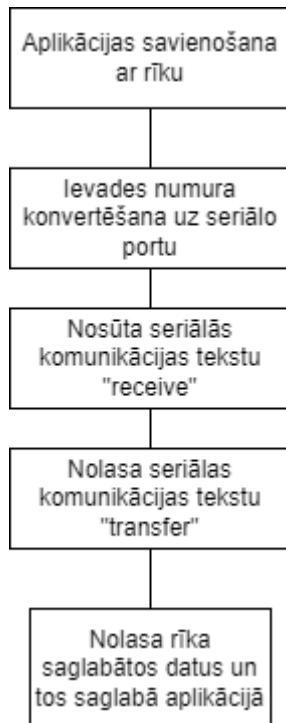
**40.attēls. Konfigurācijas rāmja ielāde**

#### **4.2.5. Rīka saskarnes valodas maiņa**



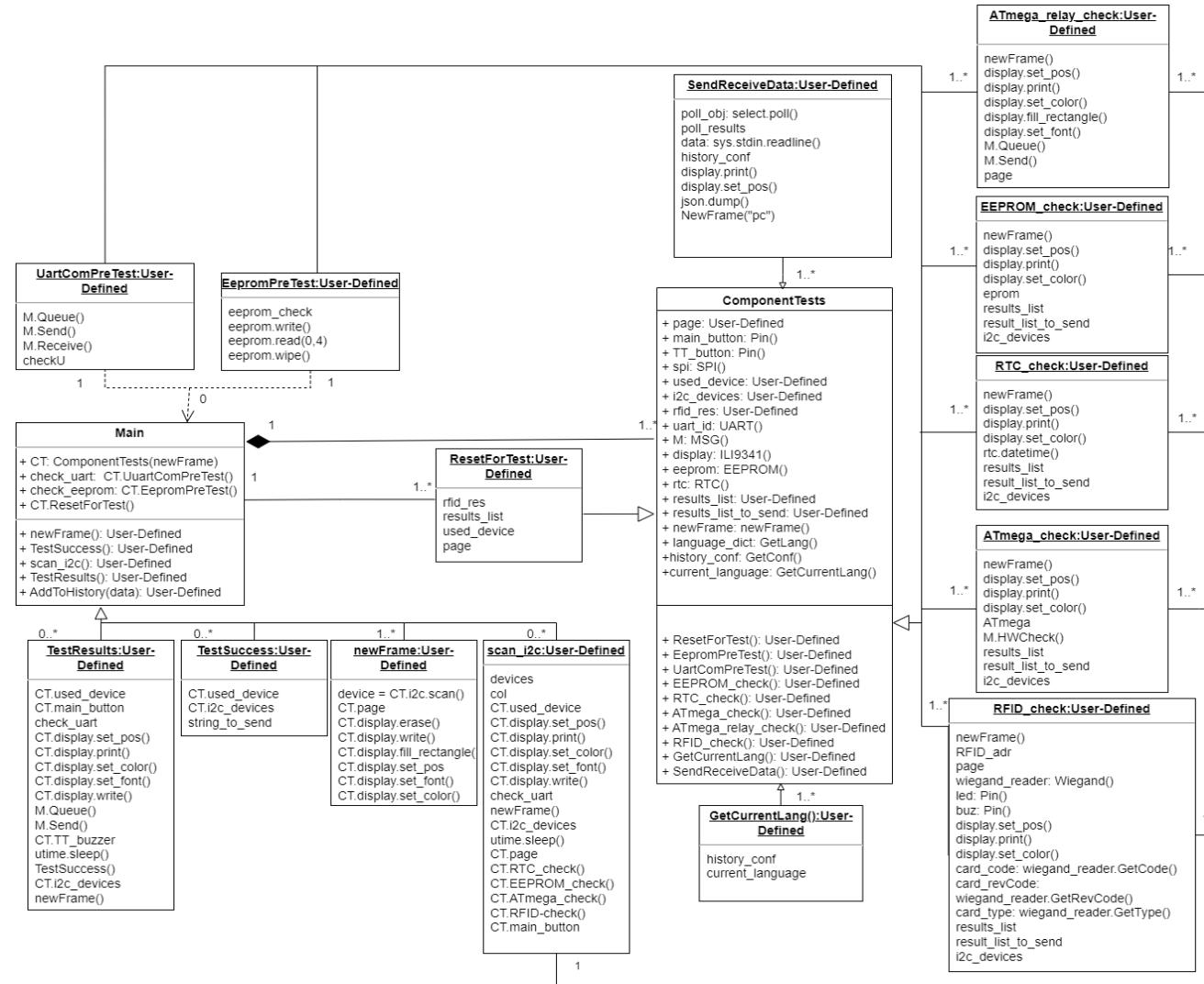
**41.attēls. Rīka saskarnes valodas maiņa**

#### **4.2.6. Aplikācijas savienošana ar rīku**

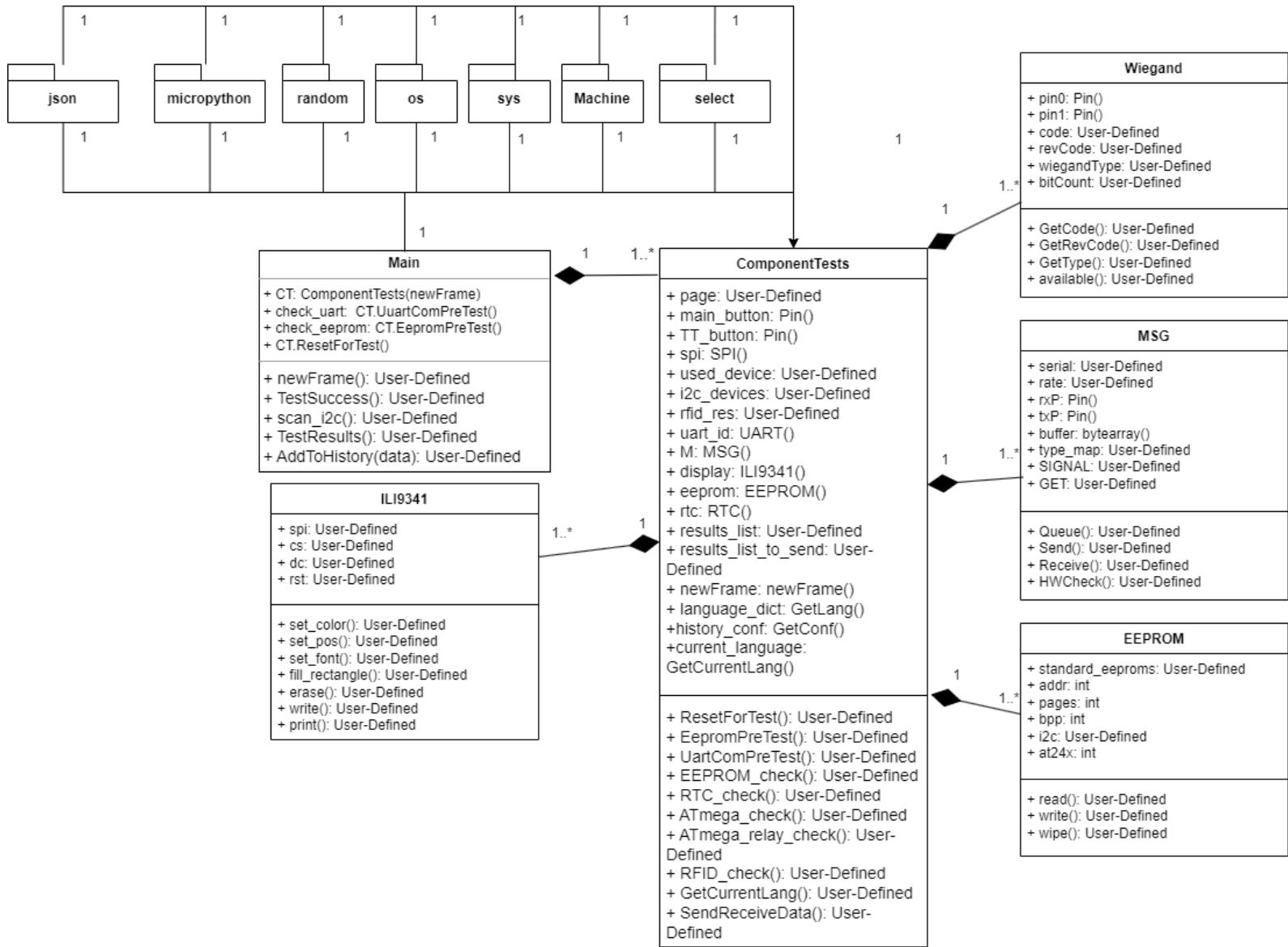


**42.attēls. Aplikācijas savienošana ar rīku**

### 4.3.Rīka programmatūras kļašu diagramma

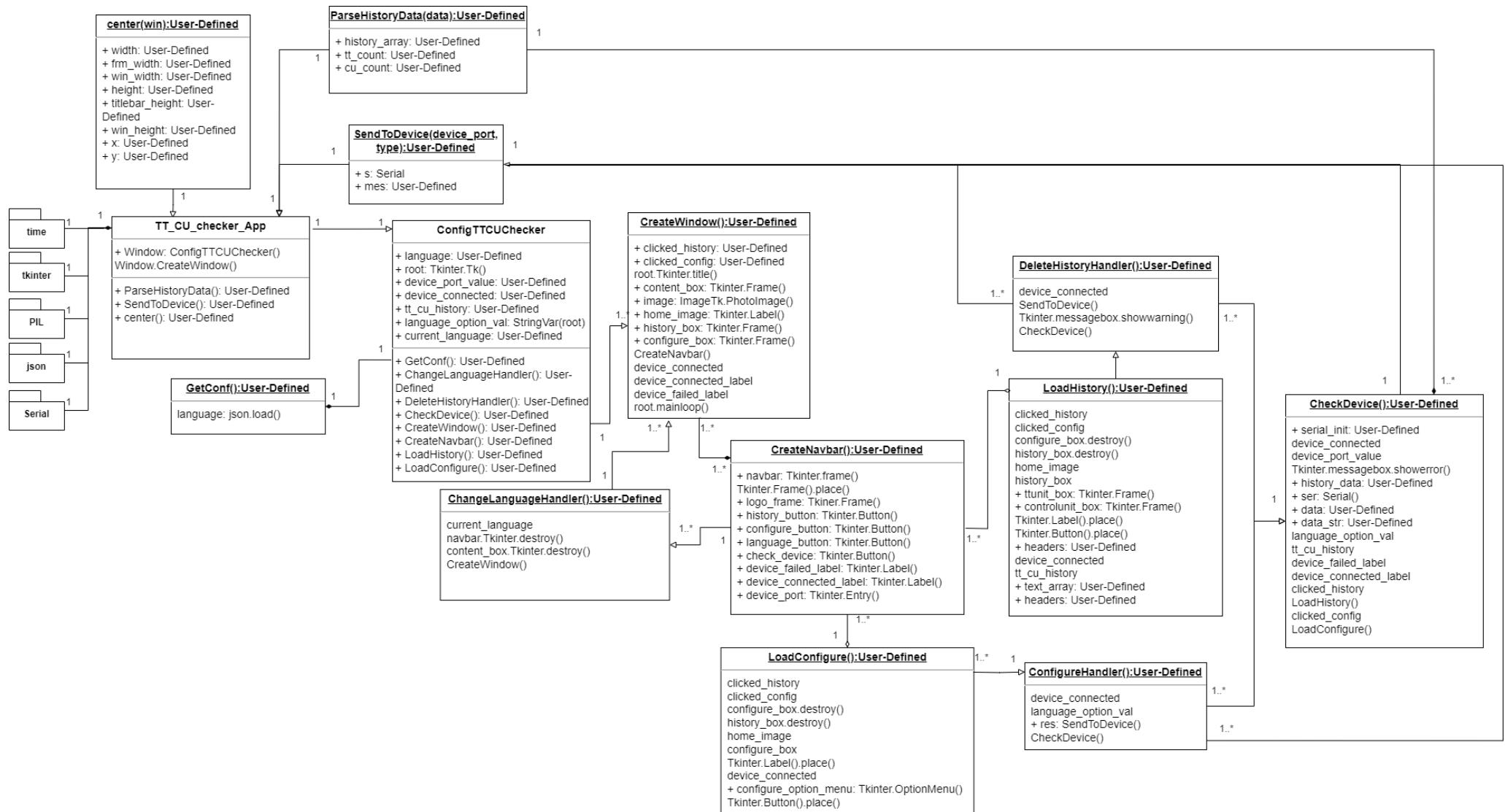


43. attēls. Rīka programmatūras kļašu diagramma



44. attēls. Bibliotēku un palīg-klašu kļaušu diagramma

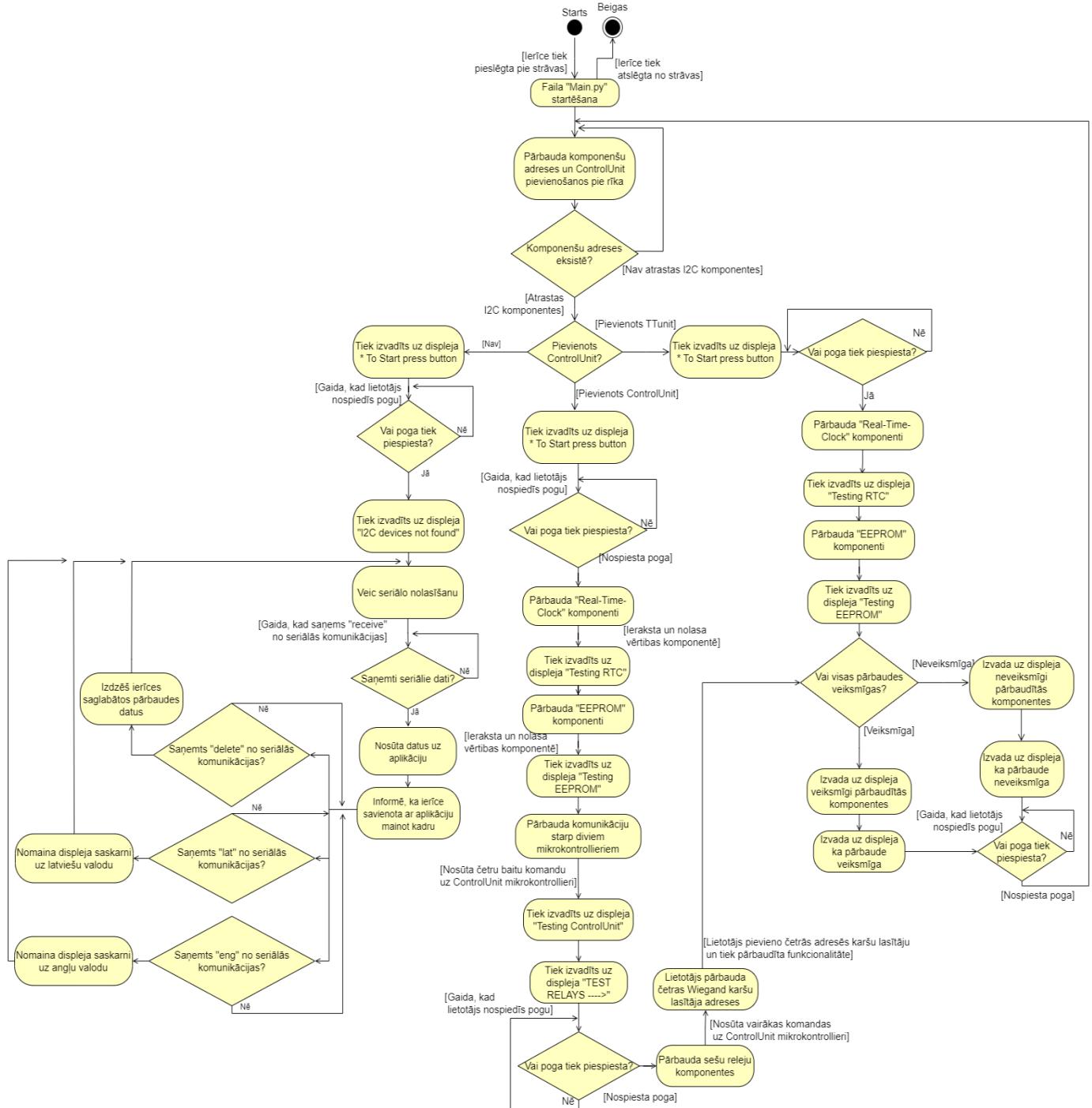
#### 4.4.Konfigurācijas aplikācijas programmatūras klašu diagramma



45.attēls. Konfigurācijas aplikācijas programmatūras klašu diagramma

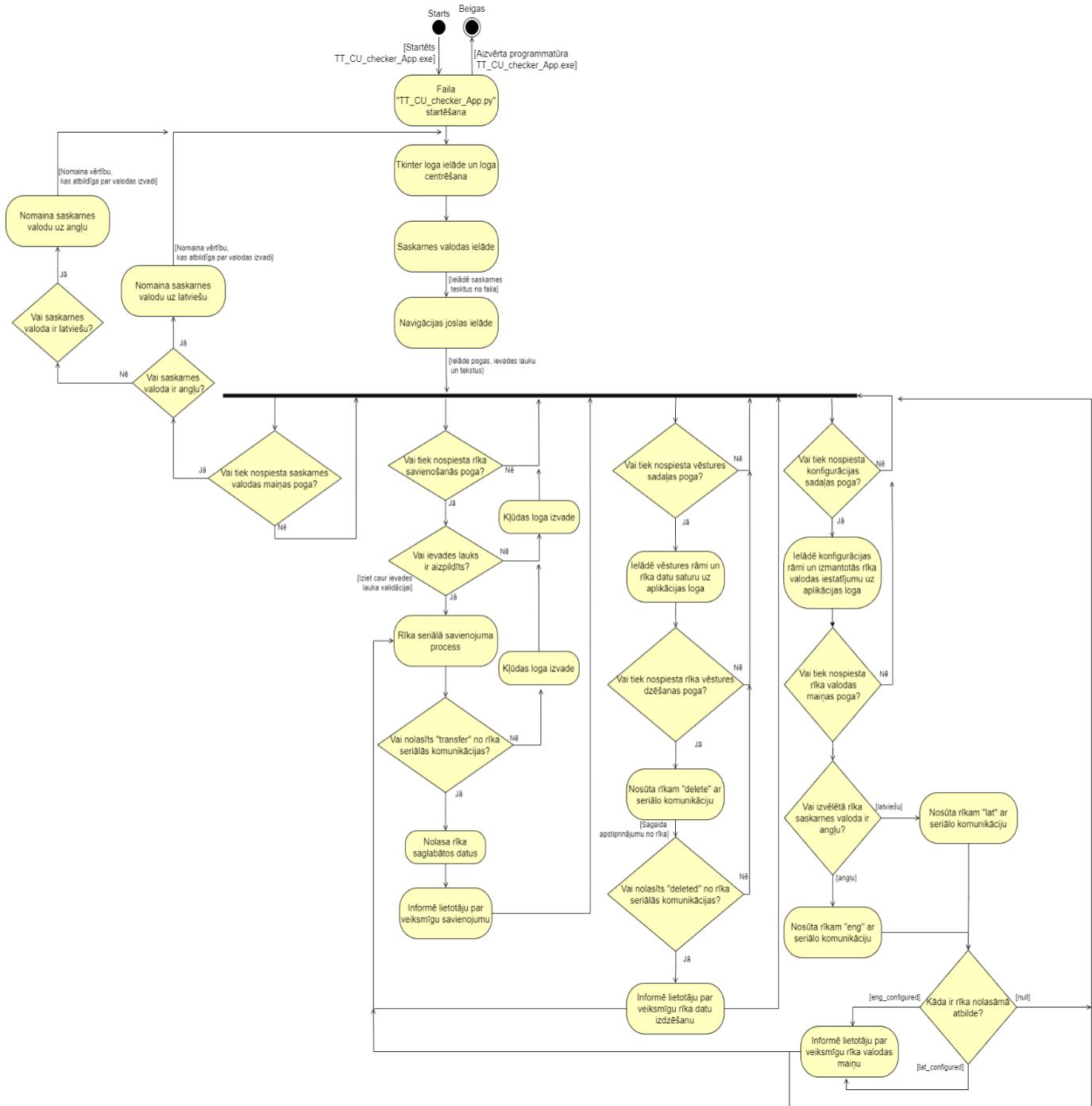
## 4.5. Funkcionālais un dinamiskais sistēmas modelis

### 4.5.1. Rīka programmatūras stāvokļu diagramma



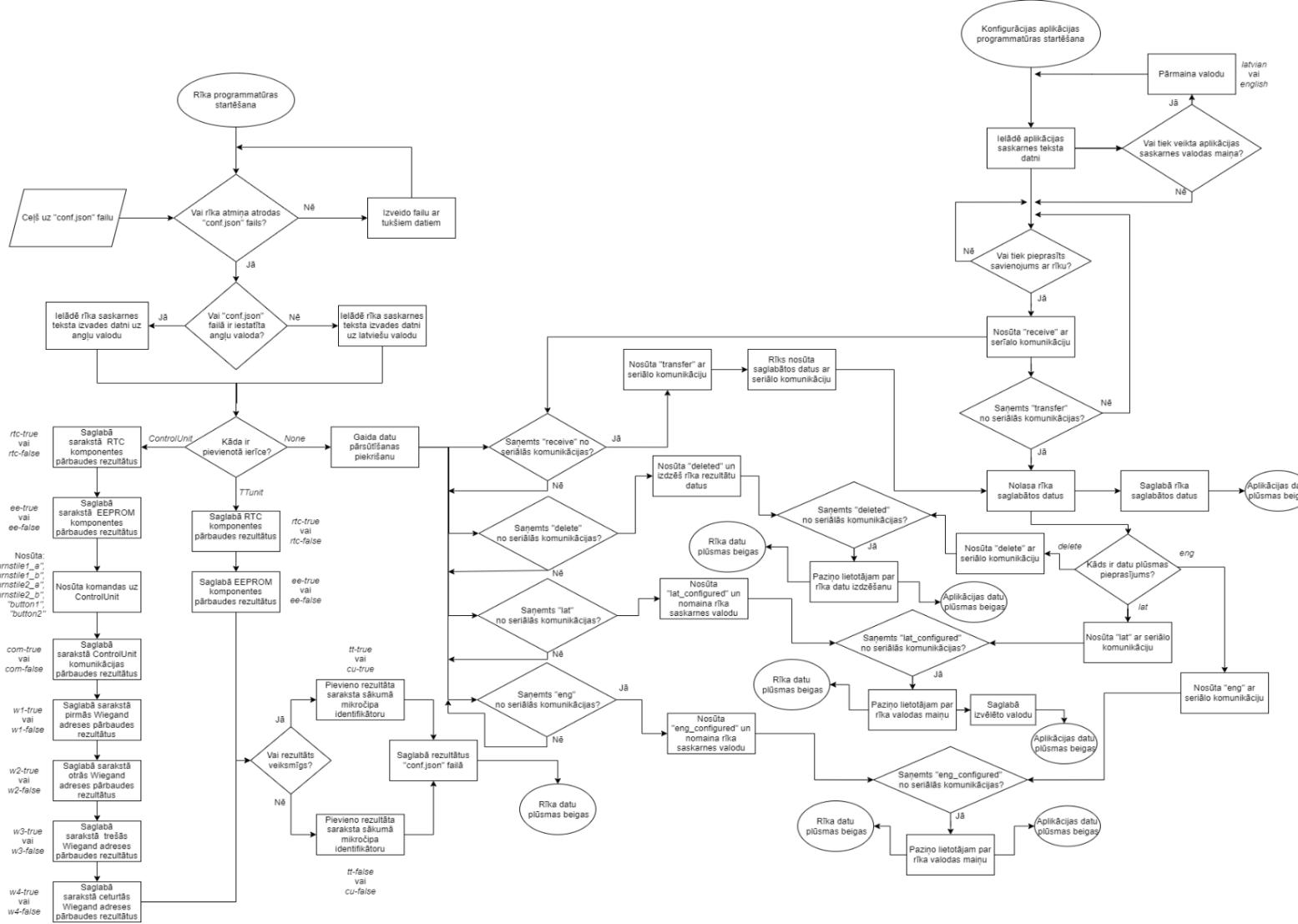
46. attēls. Rīka stāvokļu diagramma

#### 4.5.2. Konfigurācijas aplikācijas programmatūras stāvokļu diagramma



47. attēls. Konfigurācijas aplikācijas stāvokļu diagramma

### 4.5.3. Datu plūsmas diagramma



48. attēls. Datu plūsmas diagramma

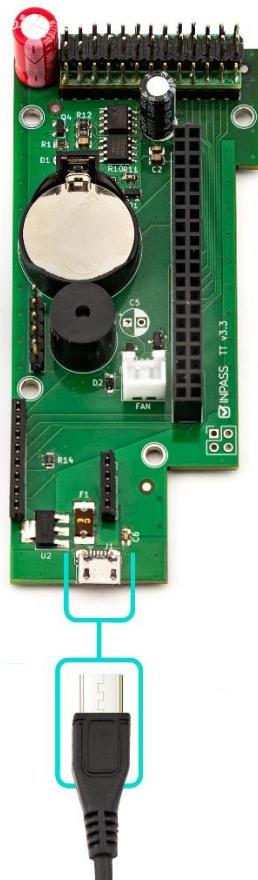
## 5. Lietotāju ceļvedis

Šajā nodaļā detalizēti aprakstīts ceļvedis, kurš lietotājam nodrošinās nepieciešamos norādījumus un procedūras, lai veiksmīgi izmantotu produktu un veiktu pareizu TTunit un ControlUnit mikročipu pārbaudi.

### 5.1 TTunit mikročipa pārbaudes sagataves process

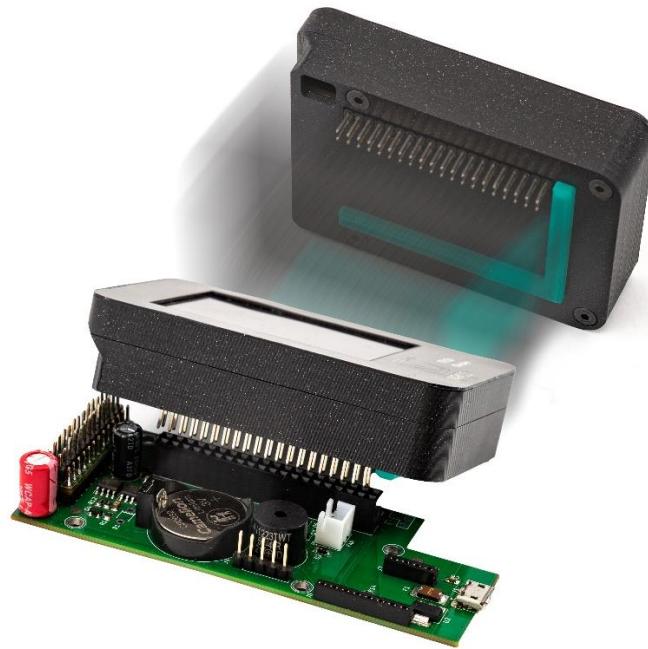
TTunit mikročipa pārbaudes sagataves process sastāv no sekojošiem soļiem:

1. Pieslēgt pārbaudei vēlamo TTunit mikročipu pie strāvas ar micro-USB vadu (skatīt 49. attēlu).



49. attēls. TTunit pieslēgšana pie strāvas

2. Pieslēgt rīku pie TTunit mikročipa 2x20 ligzdas (skatīt 50. attēlu).



**50. attēls. Rīka pieslēgšana pie TTunit**

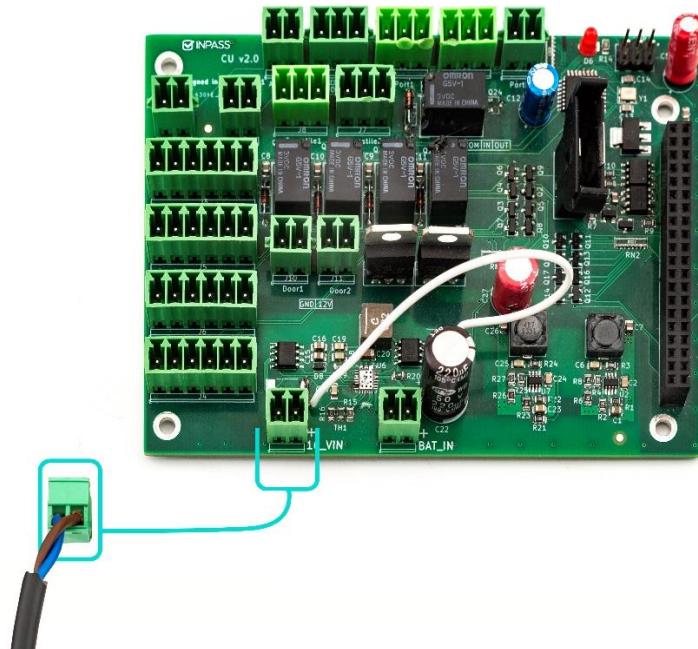
3. Gaidīt līdz rīka displejs ieslēdzas un tiek izvadīts sākuma kadrs (skatīt 5.attēlu).

Lietotājam veicot šos trīs soļus būs iespēja pārbaudīt pievienoto TTunit mikročipu un iegūt mikročipa funkcionalitātes stāvokli.

## 5.2 ControlUnit mikročipa pārbaudes sagataves process

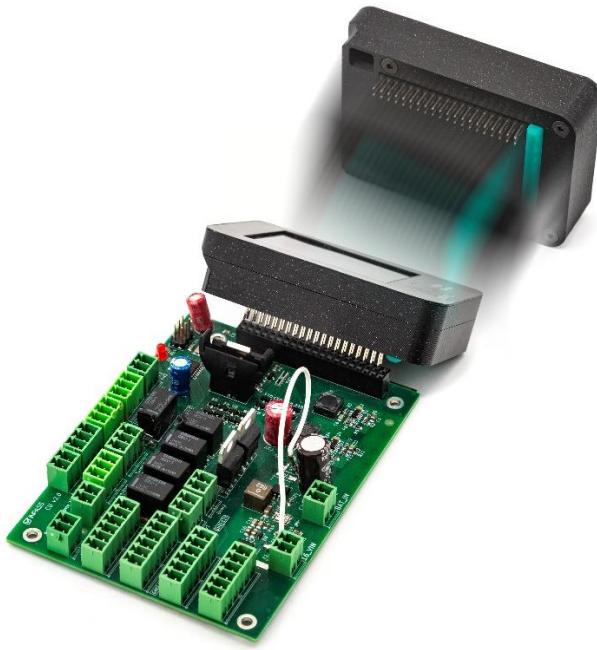
ControlUnit mikročipa pārbaudes sagataves process sastāv no sekojošiem soļiem:

- Pieslēgt pārbaudei vēlamo ControlUnit mikročipu pie strāvas ar 16V vadu pie 16\_VIN savienojuma ligzdas (skatīt 51. attēlu).



**51. attēls. ControlUnit pieslēgšana pie strāvas**

- Pievienot rīku pie ControlUnit mikročipa 2x20 ligzdas (skatīt 52. attēlu).



**52. attēls. Rīkā pievienošana pie ControlUnit**

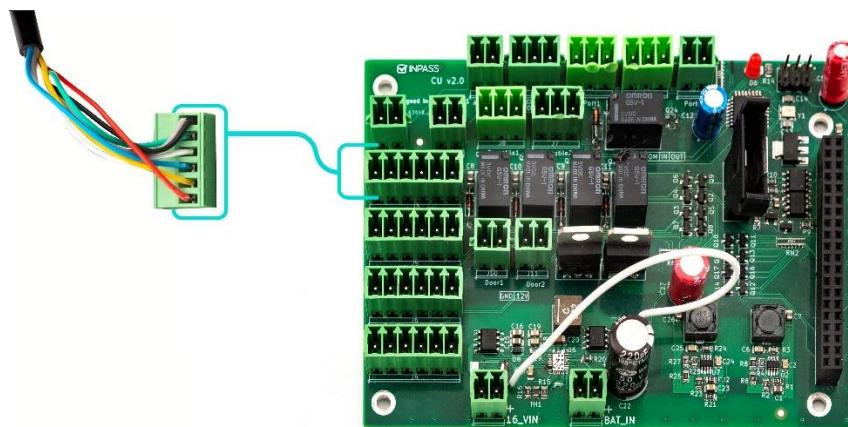
3. Gaidīt līdz displejs ieslēdzas un tiek izvadīts sākuma kadrs ar ControlUnit.

Veicot šos trīs soļus būs iespēja pārbaudīt pievienotā ControlUnit funkcionalitātes stāvokli un citu komponenšu stāvokli.

### 5.3 ControlUnit mikročipa Wiegand karšu lasītāja pārbaudes process

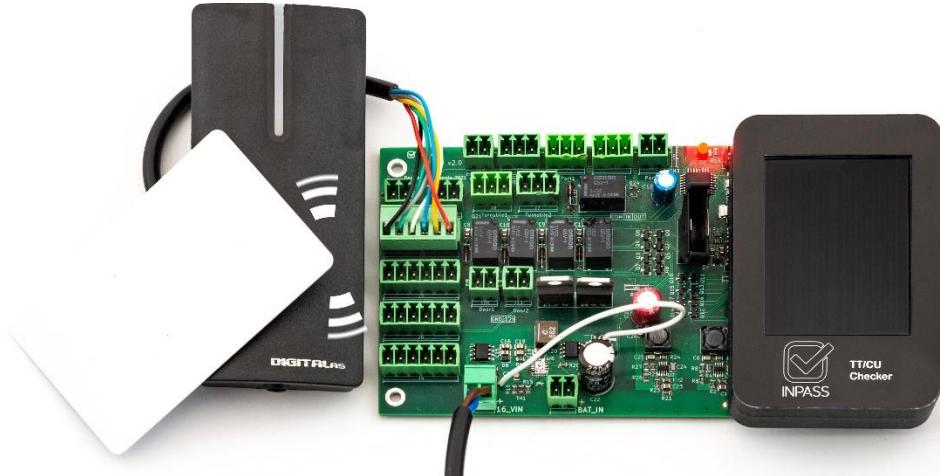
Pievienojot rīku pie ControlUnit un progresējot cauri mikročipa pārbaudei līdz desmitai pārbaudei, jeb “10. test”, lietotājam ir iespēja pārbaudīt četras ControlUnit Wiegand protokola karšu lasītāju adreses. Lietotājam jāveic karšu lasītāja pievienošana attiecīgi pēc izvadītās informācijas uz displeja (skatīt 12. attēlu). Lietotājam jāseko sekojošiem soļiem:

1. Gadījuma, ja uz displeja tiek izvadīts “1. Wiegand device address testing”, tad lietotājam jāpievieno Wiegand karšu lasītājs pie pirmās ControlUnit adreses (skatīt 53. attēlu).



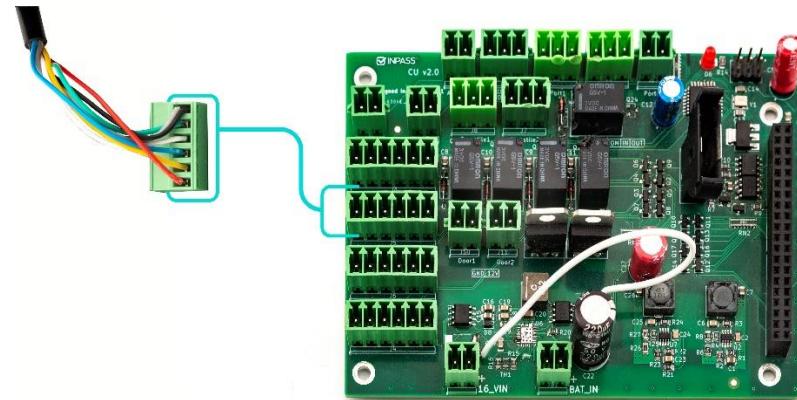
**53. attēls. Karšu lasītāja pievienošana pie pirmās ControlUnit adreses**

- Lietotājam pievienojot karšu lasītāju ir jānoskenē RFID kartiņa uzliekot to uz karšu lasītāja (skatīt 54. attēlu).



**54. attēls. RFID kartiņas noskenēšana uz pirmās adreses**

- Noskenējot kartiņu tiek izvadīts kartiņas numurs uz displeja (skatīt 13. attēlu).
- Gadījuma, ja uz displeja tiek izvadīts "2. Wiegand device address testing", tad lietotājam jāpievieno Wiegand karšu lasītājs pie otrās ControlUnit adreses (skatīt 55. attēlu).



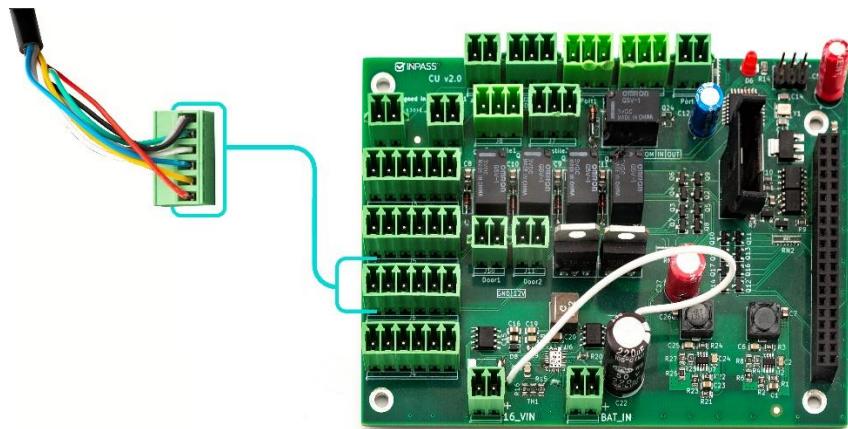
**56. attēls. Karšu lasītāja pievienošana pie otrās ControlUnit adreses**

- Lietotājam pievienojot karšu lasītāju ir jānoskenē RFID kartiņa uzliekot to uz karšu lasītāja (skatīt 57. attēlu).



**57. attēls. RFID kartiņas noskenēšana uz otrās adreses**

6. Noskenējot kartiņu tiek izvadīts kartiņas numurs uz displeja (skatīt 13. attēlu).
7. Gadījuma, ja uz displeja tiek izvadīts “3. Wiegand device address testing”, tad lietotājam jāpievieno Wiegand karšu lasītājs pie trešas ControlUnit adreses (skatīt 58. attēlu).



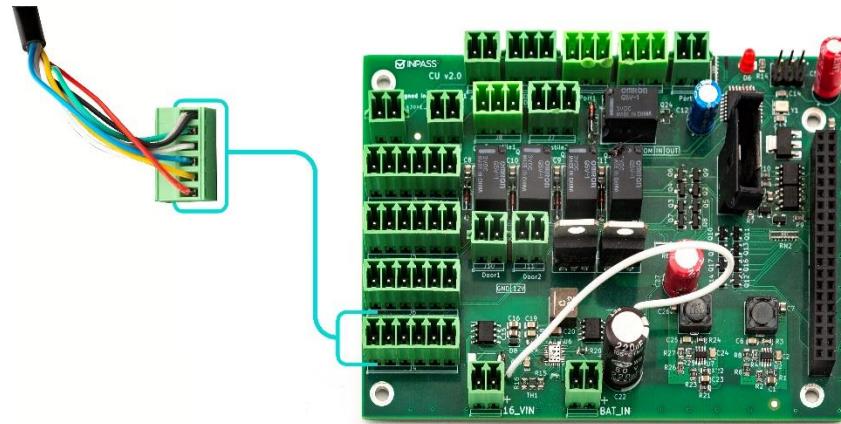
**58. attēls. Karšu lasītāja pievienošana pie trešas ControlUnit adreses**

8. Lietotājam pievienojot karšu lasītāju ir jānoskenē RFID kartiņa uzliekot to uz karšu lasītāja (skatīt 59. attēlu).



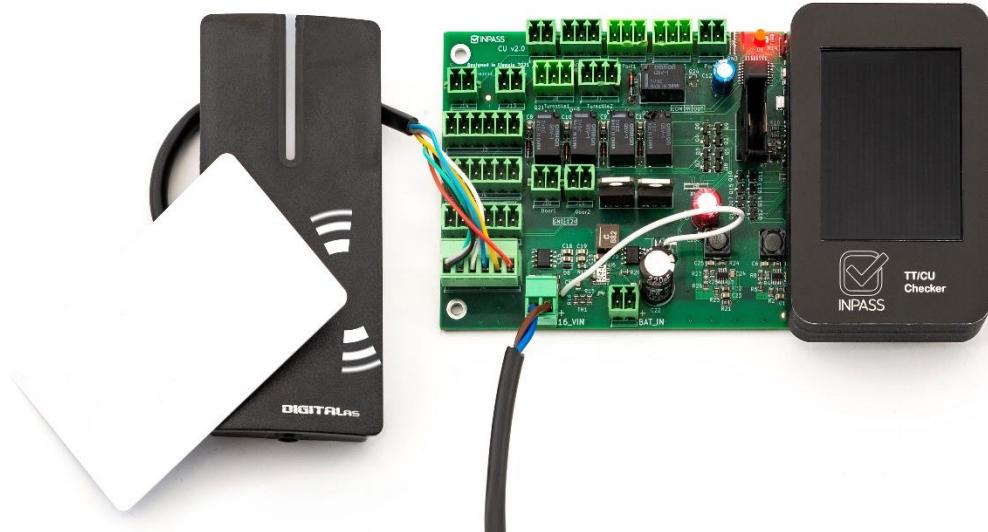
**59. attēls. RFID kartiņas noskenēšana uz trešās adreses**

9. Noskenējot kartiņu tiek izvadīts kartiņas numurs uz displeja (skatīt 13. attēlu).
10. Gadījuma, ja uz displeja tiek izvadīts “4. Wiegand device address testing”, tad lietotājam jāpievieno Wiegand karšu lasītājs pie ceturtās ControlUnit adreses (skatīt 60. attēlu).



**60. attēls. Karšu lasītāja pievienošana pie ceturtās ControlUnit adreses**

11. Lietotājam pievienojot karšu lasītāju ir jānoskenē RFID kartiņa uzliekot to uz karšu lasītāja (skatīt 61. attēlu).



**62. attēls. RFID kartiņas noskenēšana uz ceturtās adreses**

12. Noskenējot kartiņu tiek izvadīts kartiņas numurs uz displeja (skatīt 13. attēlu).

## **5.4 Rīka savienošanas process ar konfigurācijas aplikāciju**

Veiksmīgam rīka savienojumam ar konfigurācijas aplikāciju ir jāveic specifisks process, gan operētājsistēmā, gan fiziski ar rīku:

1. Pievienot rīka ierīci ar micro-USB vadu (skatīt 63.attēlu);



**63.attēls. Rīka savienošana ar micro-USB vadu**

2. Pievienot USB vadu pie darbstacijas, kur ir uzinstalēta rīka konfigurācijas aplikācija;
3. Windows operētājsistēmā atrast atbilstošo seriālo porta numuru, kurā rīks tiek pievienots. Seriālā porta numura noskaidrošanas process:
  1. Atvērt Ierīču pārvaldnieku;
  2. Sarakstā jāatrod Ports (COM & LPT);
  3. Uzklikšķinot uz saraksti jāatrod ierīce ar nosaukumu “USB Serial Device” un jānoskaidro seriāla porta numurs, piemēram, COM7.
4. Atvērt rīka konfigurācijas aplikāciju uz Windows operētājsistēmas;
5. Tālāk sekot P.16. un P.20. funkciju procesiem.

## **5.5 Rīka saskarnes pamācība**

Saskarne ir būtisks aspeks rīka programmatūras lietojamībā, kur uzsvars tiek likts uz fiziskās pogas izmantošanu dažādos darbības scenārijos. Lietotājiem tiek sniegti precīzi norādījumi par to, kā un kad izmantot fizisko pogu, lai nodrošinātu ne tikai pareizu programmatūras darbību, bet arī optimālu lietošanas pieredzi. Šie norādījumi ietver gan praktiskus piemērus, gan sekojošo procesu aprakstus, kas palīdz lietotājiem saprast nākamo soli un veikt attiecīgās darbības bez problēmām vai neskaidrībām.

Scenārijos, kad uz displeja labajā augšējā stūrī tiek parādīts signāls “PRESS--->”, “BUZZ--->” vai “RESTART-->”, (skatīt 5. attēlu), lietotājam ir jāveic fiziskās pogas nospiešana (skatīt 4. attēlu), lai virzītos uz priekšu un turpinātu programmatūras darbību. Šāda saskarne starp lietotāju un programmatūru nodrošina nepieciešamo līdzsvaru starp digitālo un fizisko vidi.

Attiecīgajā scenārijā, kad tiek veikta ControlUnit releju pārbaude, uz displeja izvadās signāls "Listen for relay!" Šis norādījums informē lietotāju par nepieciešamību klausīties releju aktivizēšanās skaņu. Lietotājs, dzirdot šo skaņu, var novērtēt releja darbību un secināt, vai attiecīgais relejs ir darbspējīgs vai nē. Šāda pieeja lietotājam nodrošina kontroli veicot nepieciešamos pasākumus manuāli, aicinot lietotāju aktīvi iesaistīties darbību veikšanā. Rezultātā problēmu risināšana notiek efektīvi un lietošanas pieredze pielāgota lietotāja vajadzībām.

## **5.6 Konfigurācijas aplikācijas saskarnes pamācība**

Aplikācijas saskarne ir izstrādāta tā, lai tā būtu intuitīva un viegli saprotama, nodrošinot, ka ikviens lietotājs varētu to veiksmīgi izmantot. Saskaņa sastāv no vairākām būtiskām komponentēm, tostarp dažādām pogām, nolaižamiem logiem un teksta ievades laukiem, kas nodrošina pilnīgu piekļuvi visām aplikācijas funkcijām, izmantojot gan peli, gan tastatūru.

Aplikācijas pogas ir pieejamas ar peles kreiso klikšķi, un uz tām redzamie teksti skaidri norāda to funkciju, tādējādi lietotājs viegli saprot, kāda darbība tiks veikta pēc pogas nospiešanas. Šī vizuālā un funkcionālā skaidrība ievērojami atvieglo aplikācijas lietošanu.

Nolaižamie logi arī pieejami izmantojot peles kreiso klikšķi, un to struktūra ir veidota tā, lai būtu intuitīva un viegli izprotama. Lietotāji var vienkārši izvēlēties divas nepieciešamās opcijas no nolaižamā saraksa, kas padara datu ievadīšanu lietotājam draudzīgu.

Teksta ievades lauki nodrošina datu ievadīšanas iespēju, jo tos var izmantot gan ar peles kreiso klikšķi, gan ar tastatūru. Šie lauki ļauj lietotājiem ātri un ērti ievadīt nepieciešamo seriālo portu, tādējādi, uzlabojot lietošanas pieredzi un produktivitāti.

Kopumā aplikācijas saskarne ir veidota, ņemot vērā lietotāju ērtības un vieglu pieejamību, nodrošinot, ka pat tie, kuriem ir tikai pamata datora zināšanas, spēs to efektīvi izmantot bez grūtībām. Šāda pieeja uzlabo aplikācijas vispārējo pieejamību un lietojamību.

## **6. Testēšanas dokumentācija**

Šajā nodaļā detalizēti aprakstīts un vizualizēts programmatūras testēšanas process, iekļaujot izvēlētās metodes un rīkus, kā arī demonstrēti testpiemēri.

### **6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums**

Programmatūras testēšanas posmā izstrādātājs ir izvēlējies divas nozīmīgas testēšanas metodes - melnās kastes testēšanu un baltās kastes testēšanu. Šīs divas metodes nodrošina pietiekamu un rūpīgu pārbaudi, kas aptver gan programmatūras funkcionalitāti no lietotāja perspektīvas (melnās kastes testēšana), gan arī tās iekšējo struktūru un loģiku (baltās kastes testēšana). Šāda vispusīgā pieeja palīdz nodrošināt augstu kvalitāti un uzticamību visā programmatūras testēšanas procesā.

#### **6.1.1. Melnās kastes testēšanas rīku apraksts un pamatojums**

Lai veiktu augstas kvalitātes programmatūras melnās kastes testus, izstrādātājam ir jāizmanto un jānodrošina vairāki rīki:

- Vairākums programmatūras funkcionalitātes balstās uz fizisku mikročipu pārbaudi. Izstrādātājam jānodrošina visaptveroša pārbaude, aptverot visus iespējamos scenārijus. Piemēram, nepieciešams pārbaudīt vai EEPROM komponente darbojas pareizi uz TTunit mikročipa, vai arī jāpārbauda, vai mikrokontrollieru pareizi mijiedarbojas starp produktu un ControlUnit. Šo rīku noderīgums slēpjelas tajā, ka programmatūras lietošanas scenāriji var būt ļoti dažādi. Programmatūras uzticamai testēšanai, nepieciešams iegūt pilnīgu un detalizētu priekšstatu par visiem iespējamajiem scenārijiem. Šāda pieeja ļauj izstrādātājam veiksmīgi identificēt un novērst iespējamās problēmas pirms tās nonāk lietotāju rokās.
- Produkta konfigurācijas aplikācija ir papildus sistēmas elements, kas balstās uz dažādiem ievades laukiem un fizisku komunikāciju starp produktu un darbstaciju. Lai nodrošinātu augstu kvalitāti un uzticamus rezultātus, izstrādātājam ir jāveic rūpīga ievades lauku manipulācija. Tas ietver ne tikai ievades lauku vērtību dažādo variantu testēšanu, bet arī komunikācijas traucējumu simulācijas, lai novērtētu sistēmas izturību un atgūstīšanas spējas neparedzētos apstākļos. Izstrādātājs uzņemas atbildību par šo procesu plānošanu, izpildi un dokumentēšanu, lai nodrošinātu pilnīgu un precīzu produktu konfigurācijas aplikācijas darbību un savietojamību ar dažādām darba vides situācijām.

Problēmsituāciju laicīga identifikācija un to novēršana jau izstrādes stadijā, nodrošinot augstu programmatūras kvalitāti un lietotāja apmierinātību.

Šīs melnās kastes testēšanas metodes ir būtiskas, lai novērstu potenciālas klūdas un nodrošinātu labāko gala lietotāja pieredzi, ar iespēju uz manuālām manipulācijām, pēc paša lietotāja ieskatiem.

#### **6.1.2. Baltās kastes testēšanas rīku apraksts un pamatojums**

Lai veiktu augstas kvalitātes programmatūras baltās kastes testus, izstrādātājam ir jāizmanto un jānodrošina vairākas testēšanas metodes, kas sniedz padziļinātu ieskatu programmēšanas kodā un nodrošina pārliecību par programmatūras darbību:

- Izvades validācijas testēšanas metodes ļauj izstrādātājam veiksmīgi veikt baltās kastes testus, pārbaudot programmas izvades rezultātus un to atbilstību uzstādītajiem kritērijiem. Piemēram, nepieciešams pārbaudīt, vai konkrēta programmas izdrukātā vērtība atbilst paredzētajam rezultātam noteiktā situācijā. Šī pieeja nodrošina uzticamu programmatūras funkcionalitātes pārbaudi dažādos darbības scenārijos un apstākļos, nodrošinot to precizitāti un atbilstību specifikācijām.
- Cikla testēšanas metode ir būtiska izstrādājamās programmatūras testēšanas procesā, jo lielākā daļa programmatūras balstās uz atkārtotiem cikliem un pastāvīgu darbību. Cikla testēšana sniedz pārliecību par programmatūras stabilitāti un pareizu darbību, izmantojot gan programmatūras, gan iekšējos ciklus. Tā palīdz identificēt un novērst iespējamās klūdas, kas var rasties bezgalīgā darbības ciklā.

Šīs baltās kastes testēšanas metodes ir būtiskas, lai nodrošinātu pilnīgu un precīzu ieskatu programmēšanas kodā un garantētu augstu programmatūras kvalitāti, atbilstību specifikācijām un izcilu lietotāju pieredzi.

## 6.2. Testpiemēru kopa

### 6.2.1. Rīka programmatūras testpiemēru kopa

10.tabula

#### Rīka programmatūras testpiemēru kopa

Testpiemēra ID	Testpiemēra nosaukums	Testpiemēra izpildes nosacījumi	Testpiemēra apraksts	Testpiemēra izpildes soli	Testpiemēra ievades dati	Testpiemēra sagaidāmais rezultāts
Black Box						
TP.R.01	Rīka programmatūras startēšana	Pievieno rīku pie strāvas	Rīka programmatūras "main.py" startēšana	Pievieno rīku pie strāvas ar ControlUnit vai TTunit vai darbstacijas	Pievieno rīku pie strāvas ar ControlUnit vai TTunit vai darbstacijas	Startēta rīka programmatūra, jeb fails "main.py"
TP.R.02	Mikročipa pārbaudes funkcijas startēšana	Pievieno rīku pie TTunit vai ControlUnit	Tiek palaista programmatūras mikročipa pārbaudes funkcija.	Pievieno rīku pie TTunit vai ControlUnit	Rīkam ieslēdzoties nospiest pogu labā augšējā sānā.	Pēc rīka ieslēgšanās un pogas nospiešanas tiek palaista programmatūras mikročipa pārbaudes funkcija.
TP.R.03	Ejošas EEPROM komponentes pārbaude ar programmatūru	Pievienot rīku pie TTunit ar ejošu EEPROM komponenti	Pievienojot rīku pie TTunit ar ejošu EEPROM komponenti un uzsākot mikročipa pārbaudi, var uzzināt vai rīks pārbauda, ka EEPROM komponente ir funkcionāla	Pievienot rīku pie TTunit ar ejošu EEPROM komponenti un uzsākt TTunit mikročipa pārbaudi	1) Pievienot rīku pie TTunit ar ejošu EEPROM komponenti 2) Uzsākt TTunit mikročipa pārbaudi ar pogu labā augšējā sānā.	Pārbaudes beigās uz rīka displeja jāizvada informāciju, ka EEPROM komponente ir funkcionējoša.
TP.R.04	Neejošas EEPROM komponentes pārbaude ar programmatūru	Pievienot rīku pie TTunit ar neeošu EEPROM komponenti	Pievienojot rīku pie TTunit ar neeošu EEPROM komponenti un uzsākot mikročipa pārbaudi, var uzzināt vai rīks pārbauda, ka EEPROM komponente ir nefunkcionāla	Pievienot rīku pie TTunit ar neeošu EEPROM komponenti un uzsākt TTunit mikročipa pārbaudi	1) Pievienot rīku pie TTunit ar ejošu EEPROM komponenti 2) Uzsākt TTunit mikročipa pārbaudi ar pogu labā augšējā sānā.	Pārbaudes beigās uz rīka displeja jāizvada informāciju, ka EEPROM komponente ir nefunkcionējoša.
TP.R.05	Ejošas Real-Time-Clock komponentes pārbaude ar programmatūru	Pievienot rīku pie TTunit ar ejošu Real-Time-Clock komponenti	Pievienojot rīku pie TTunit ar ejošu Real-Time-Clock komponenti un uzsākot mikročipa pārbaudi, var uzzināt vai rīks pārbauda, ka Real-Time-Clock komponente ir funkcionāla	Pievienot rīku pie TTunit ar ejošu Real-Time-Clock komponenti un uzsākt TTunit mikročipa pārbaudi	1) Pievienot rīku pie TTunit ar ejošu Real-Time-Clock komponenti 2) Uzsākt TTunit mikročipa pārbaudi ar pogu labā augšējā sānā.	Pārbaudes beigās uz rīka displeja jāizvada informāciju, ka Real-Time-Clock komponente ir funkcionējoša.
TP.R.06	Neejošas Real-Time-Clock komponentes pārbaude ar programmatūru	Pievienot rīku pie TTunit ar neeošu Real-Time-Clock komponenti	Pievienojot rīku pie TTunit ar neeošu Real-Time-Clock komponenti un uzsākot mikročipa pārbaudi, var uzzināt vai rīks pārbauda, ka Real-Time-Clock komponente ir nefunkcionāla	Pievienot rīku pie TTunit ar neeošu Real-Time-Clock komponenti un uzsākt TTunit mikročipa pārbaudi	1) Pievienot rīku pie TTunit ar ejošu Real-Time-Clock komponenti 2) Uzsākt TTunit mikročipa pārbaudi ar pogu labā augšējā sānā.	Pārbaudes beigās uz rīka displeja jāizvada informāciju, ka Real-Time-Clock komponente ir nefunkcionējoša.
TP.R.07	Zummera komponentes pārbaude ar rīka programmatūru	Pievieno rīku pie TTunit ar savienotu zummera komponenti	Programmatūra aktivizē zummera komponentes funkcionalitāti ar skaņu	Pievieno rīku pie TTunit ar uzstādītu zummera komponenti Uzsākt mikročipa pārbaudes procesu līdz zummera pārbaudei Nospiest rīka pogu labā augšējā sānā Klausīties zummera komponentes skaņu	1) Pievieno rīku pie TTunit ar uzstādītu zummera komponenti 2) Uzsākt mikročipa pārbaudes procesu līdz zummera pārbaudei 3) Nospiest rīka pogu labā augšējā sānā 4) Klausīties zummera komponentes skaņu	Pēc rīka pogas nospiešanas zummera komponenteti ir pāris reizes jānodūc
TP.R.08	Veiksmiga mikročipa pārbaudes rezultāta katra izvade	1) Pievienot rīku pie TTunit mikročipa ar ejošu EEPROM un Real-Time-Clock 2) Uzsākt pārbaudi un to pabeigt līdz beigām	Pārbaudot pilnīgi ejošu TTunit mikročipu ir jāpārliecinās vai to komponentu funkcionalitāti korekti izvadīs uz displeja	1) Pievienot rīku pie TTunit mikročipa ar ejošu EEPROM un Real-Time-Clock 2) Uzsākt mikročipa pārbaudes procesu ar pogu 3) Procesēt cauri visiem komponenšu pārbaudījumiem	1) Pievienot rīku pie TTunit mikročipa ar ejošu EEPROM un Real-Time-Clock 2) Uzsākt mikročipa pārbaudes procesu ar pogu 3) Procesēt cauri visiem komponenšu pārbaudījumiem	Komponenšu pārbaudes beigās uz rīka displeja izvadas informācija, ka mikročipa komponentes ir ejošas un informē, ka pārbaude ir veiksmiga
TP.R.09	Neveiksmiga mikročipa pārbaudes rezultāta katra izvade	1) Pievienot rīku pie TTunit mikročipa ar neeošu EEPROM un Real-Time-Clock 2) Uzsākt pārbaudi un to pabeigt līdz beigām	Pārbaudot neeošu TTunit mikročipu ir jāpārliecinās vai to komponenšu funkcionalitāti korekti izvadīs uz displeja	1) Pievienot rīku pie TTunit mikročipa ar neeošu EEPROM un Real-Time-Clock 2) Uzsākt mikročipa pārbaudes procesu ar pogu 3) Procesēt cauri visiem komponenšu pārbaudījumiem	1) Pievienot rīku pie TTunit mikročipa ar neeošu EEPROM un Real-Time-Clock 2) Uzsākt mikročipa pārbaudes procesu ar pogu 3) Procesēt cauri visiem komponenšu pārbaudījumiem	Komponenšu pārbaudes beigās uz rīka displeja izvadas informācija, ka mikročipa komponentes ir neeošas un informē, ka pārbaude ir neveiksmiga

## 6.2.2. Konfigurācijas aplikācijas testpiemēru kopa

11.tabula

Konfigurācijas aplikācijas testpiemēru kopa						
Testpiemēra ID	Testpiemēra nosaukums	Testpiemēra izpildes nosacījumi	Testpiemēra apraksts	Testpiemēra izpildes soli	Testpiemēra ievades dati	Testpiemēra sagaidāmais rezultāts
Black Box						
TP KA.01	Konfigurācijas aplikācijas programmatūras startēšana ar ".exe" failu	Konfigurācijas aplikācijai jābūt uzinstalētai uz Windows operētāsistēmas darbstacijas	Konfigurācijas aplikācijas programmatūrās atvēršana, jeb palaišana ar ".exe" failu	Atvērt uzinstalēto aplikāciju - "TT/CU checker configuration app.exe" uz darbstaciju	Ar kreiso peles klikšķi uzklīksināt uz aplikāciju "TT/CU checker configuration app.exe"	Startēta konfigurācijas aplikācijas programmatūra, jeb fails "TT/CU checker configuration app.exe"
TP KA.02	Konfigurācijas aplikācijas valodas maiņa uz latviešu valodu	Atvērt programmatūru "TT/CU checker configuration app.exe"	Konfigurācijas aplikācijas navigācijas joslā atradas valodas maijas pogā ar kuru iespējams nomainīt saskarnes valodu uz latviešu valodu	Atvērt aplikāciju - "TT/CU checker configuration app.exe" un pārliecināties vai šībrīža uzklīksināt uz pogu "English"	Ar kreiso peles klikšķi uzklīksināt uz pogu "English"	Pēc pogas nospiešanas aplikācijas saskarnes valoda pārmainas uz latviešu valodu
TP KA.03	Konfigurācijas aplikācijas valodas maiņa uz angļu valodu	Atvērt programmatūru "TT/CU checker configuration app.exe"	Konfigurācijas aplikācijas navigācijas joslā atradas valodas maijas pogā ar kuru iespējams nomainīt saskarnes valodu uz angļu valodu	Atvērt aplikāciju - "TT/CU checker configuration app.exe" un pārliecināties vai šībrīža uzklīksināt uz pogu "Latviešu"	Ar kreiso peles klikšķi uzklīksināt uz pogu "Latviešu"	Pēc pogas nospiešanas aplikācijas saskarnes valoda pārmainas uz angļu valodu
TP KA.04	Konfigurācijas aplikācijas seriāla savienošanās ar rīku ar pareizu seriāla porta numuru	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Sagatavot rīku priekš brīva savienojuma ar aplikāciju	Seriāla savienojuma izveide ar konfigurācijas aplikāciju un rīku, ievadot pareizu rīka seriāla porta numuru	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Pievienot rīku pie strāvas un sagatavot rīku priekš brīva savienojuma ar aplikāciju 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	1) Ievadit aktuālo rīka seriāla porta numuru, piemēram, "com7" uz konfigurācijas aplikācijas navigācijas joslas ievades laukā 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	Tiek izveidots seriālis savienojums starp rīku un konfigurācijas aplikāciju, tiek iegūti rīka saglabātie dati un uz navigācijas joslas tiek izvadīts teksts "Connected: COM7"
TP KA.05	Konfigurācijas aplikācijas seriāla savienošanās ar rīku nekorektā formātā	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Sagatavot rīku priekš brīva savienojuma ar aplikāciju	Seriāla savienojuma izveide ar konfigurācijas aplikāciju un rīku, ievadot nekorekta formāta seriāla porta numuru	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Pievienot rīku pie strāvas un sagatavot rīku priekš brīva savienojuma ar aplikāciju 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	1) Ievadit nekorekta formāta seriāla porta numuru, piemēram, "cmd7" uz konfigurācijas aplikācijas navigācijas joslas ievades laukā 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	1) Seriālais savienojums starp rīku un konfigurācijas aplikāciju netiek izveidots 2) Netiek iegūti rīka saglabātie dati 3) Uz darbstacijas monitora izvadas atsevišķs logs ar informāciju "Invalid serial (COM) port format. Try again!"
TP KA.06	Konfigurācijas aplikācijas seriāla savienošanās ar rīku ar nepareizu seriāla porta numuru	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Sagatavot rīku priekš brīva savienojuma ar aplikāciju	Seriāla savienojuma izveide ar konfigurācijas aplikāciju un rīku, ievadot nepatīstību rīka seriāla porta numuru	1) Atvērt programmatūru "TT/CU checker configuration app.exe" 2) Pievienot rīku pie strāvas un sagatavot rīku priekš brīva savienojuma ar aplikāciju 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	1) Ievadit neatbilstošu seriāla porta numuru, piemēram, "com5" uz konfigurācijas aplikācijas navigācijas joslas ievades laukā 2) Nospiest pogu "Device connection" ar kreiso peles klikšķi	1) Serialais savienojums starp rīku un konfigurācijas aplikāciju netiek izveidots 2) Netiek iegūti rīka saglabātie dati 3) Uz darbstacijas monitora izvadas atsevišķs logs ar informāciju "The serial port (COM port) can't connect to the configuration app!"
TP KA.07	Konfigurācijas aplikācijas vēstures rāmja ielādešana bez iegūtiem rīka datiem	Atvērt programmatūru "TT/CU checker configuration app.exe" un neveikt iepriekšēju savienojumu ar rīku	Vēstures rāmja un tā saturā ielāde bez iegūtiem rīka mikročipu pārbaudes rezultātu datiem uz konfigurācijas aplikācijas loga	Atvērt aplikāciju - "TT/CU checker configuration app.exe" un pārliecināties vai rīks nav savienots ar konfigurācijas aplikāciju	Ar kreiso peles klikšķi uzklīksināt uz pogu "History"	Vēstures rāmim ir jāielādējas uz aplikācijas loga bez mikročipu pārbaudes rezultātu datiem, rāmja saturā tik ielādējas pogā "Delete device history" un tabulu galvenes
TP KA.08	Konfigurācijas aplikācijas vēstures rāmja ielādešana ar iegūtiem rīka datiem	Atvērt programmatūru "TT/CU checker configuration app.exe" un veikt iepriekšēju savienojumu ar rīku	Vēstures rāmja un tā saturā ielāde ar iegūtiem rīka mikročipu pārbaudes rezultātu datiem uz konfigurācijas aplikācijas loga	Atvērt aplikāciju - "TT/CU checker configuration app.exe" un pārliecināties vai rīks ir savienots ar konfigurācijas aplikāciju	Ar kreiso peles klikšķi uzklīksināt uz pogu "History"	Vēstures rāmim ir jāielādējas uz aplikācijas loga ar mikročipu pārbaudes rezultātu datiem, rāmja saturā ielādējas pogā "Delete device history" un tabulu galvenes ar rezultātu datiem
TP KA.09	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana ar konfigurācijas aplikāciju bez rīka savienojuma	Atvērt programmatūru "TT/CU checker configuration app.exe" un neveikt iepriekšēju savienojumu ar rīku	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana bez iepriekšējas rīka savienošanas ar konfigurācijas aplikāciju	1) Atvērt aplikāciju - "TT/CU checker configuration app.exe" 2) Atvērt vēstures rāmji 3) Pārliecināties vai rīks nav savienots ar konfigurācijas aplikāciju	Ar kreiso peles klikšķi uzklīksināt uz pogu "Delete device history"	Pēc pogas nospiešanas nekam nav jānotiek
TP KA.10	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana ar konfigurācijas aplikāciju izveidojot rīka savienojumu	Atvērt programmatūru "TT/CU checker configuration app.exe" un veikt iepriekšēju savienojumu ar rīku	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana ar iepriekšējas rīka savienošanas ar konfigurācijas aplikāciju	1) Atvērt aplikāciju - "TT/CU checker configuration app.exe" 2) Atvērt vēstures rāmji 3) Pārliecināties vai rīks ir savienots ar konfigurācijas aplikāciju	Ar kreiso peles klikšķi uzklīksināt uz pogu "Delete device history"	1) Pēc pogas nospiešanas tiek izvadīts atsevišķs logs ar informāciju "TT/CU checker testing history is now deleted from the device and is not recoverable!" 2) Pēc pogas nospiešanas uz rīka displejā tiek izvadīta informācija "Izdzēsta ierices vēsture"

## 6.3. Testēšanas žurnāls

### 6.3.1. Rīka programmatūras testēšanas žurnāls

12.tabula

Rīka programmatūras testpiemēru žurnāls

Testēšanas ID	Datums	Testpiemēra ID	Testpiemēra nosaukums	Testēja	Statuss	Kļūdas ziņojums	Kļūdas ziņojuma Nr.
<b>Black Box</b>							
TZ.R.01	28/5/2024	TP.R.01	Rīka programmatūras startēšana	Renārs Puķis	Veiksmīgs		
TZ.R.02	28/5/2024	TP.R.02	Mikročipa pārbaudes funkcijas startēšana	Renārs Puķis	Veiksmīgs		
TZ.R.03	28/5/2024	TP.R.03	Ejošas EEPROM komponentes pārbaude ar programmatūru	Renārs Puķis	Veiksmīgs		
TZ.R.04	28/5/2024	TP.R.04	Neejošas EEPROM komponentes pārbaude ar programmatūru	Renārs Puķis	Veiksmīgs		
TZ.R.05	28/5/2024	TP.R.05	Ejošas Real-Time-Clock komponentes pārbaude ar programmatūru	Renārs Puķis	Veiksmīgs		
TZ.R.06	28/5/2024	TP.R.06	Neejošas Real-Time-Clock komponentes pārbaude ar programmatūru	Renārs Puķis	Veiksmīgs		
TZ.R.07	28/5/2024	TP.R.07	Zummera komponentes pārbaude ar rīka programmatūru	Renārs Puķis	Veiksmīgs		
TZ.R.08	28/5/2024	TP.R.08	Veiksmīga mikročipa pārbaudes rezultāta kadra izvade	Renārs Puķis	Veiksmīgs		
TZ.R.09	28/5/2024	TP.R.09	Neveiksmīga mikročipa pārbaudes rezultāta kadra izvade	Renārs Puķis	Veiksmīgs		

### 6.3.2. Konfigurācijas aplikācijas testēšanas žurnāls

13.tabula

Konfigurācijas aplikācijas testpiemēru žurnāls							
Testēšanas ID	Datums	Testpiemēra ID	Testpiemēra nosaukums	Testēja	Statuss	Kļūdas ziņojums	Kļūdas ziņojuma Nr.
<b>Black Box</b>							
TZ.KA.01	28/5/2024	TP.KA.01	Konfigurācijas aplikācijas programmatūras startēšana ar ".exe" failu	Renārs Puķis	Veiksmīgs		
TZ.KA.02	28/5/2024	TP.KA.02	Konfigurācijas aplikācijas valodas maiņa uz latviešu valodu	Renārs Puķis	Veiksmīgs		
TZ.KA.03	28/5/2024	TP.KA.03	Konfigurācijas aplikācijas valodas maiņa uz angļu valodu	Renārs Puķis	Veiksmīgs		
TZ.KA.04	28/5/2024	TP.KA.04	Konfigurācijas aplikācijas seriālā savienošanās ar rīku ar pareizu seriālā porta numuru	Renārs Puķis	Veiksmīgs		
TZ.KA.05	28/5/2024	TP.KA.05	Konfigurācijas aplikācijas seriālā savienošanās ar rīku nekorektā formātā	Renārs Puķis	Veiksmīgs		
TZ.KA.06	28/5/2024	TP.KA.06	Konfigurācijas aplikācijas seriālā savienošanās ar rīku ar nepareizu seriālā porta numuru	Renārs Puķis	Veiksmīgs		
TZ.KA.07	28/5/2024	TP.KA.07	Konfigurācijas aplikācijas vēstures rāmja ielādešana bez iegūtiem rīka datiem	Renārs Puķis	Veiksmīgs		
TZ.KA.08	28/5/2024	TP.KA.08	Konfigurācijas aplikācijas vēstures rāmja ielādešana ar iegūtiem rīka datiem	Renārs Puķis	Veiksmīgs		

TZ.KA.09	28/5/2024	TP.KA.09	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana ar konfigurācijas aplikāciju bez rīka savienojuma	Renārs Puķis	Veiksmīgs
TZ.KA.10	28/5/2024	TP.KA.10	Rīka mikročipu pārbaudes rezultātu saglabāto datu izdzēšana ar konfigurācijas aplikāciju izveidojot rīka savienojumu	Renārs Puķis	Veiksmīgs

## **7. Secinājumi**

Šajā nodaļā aprakstīti paša izstrādātāja secinājumi par izstrādāto programmatūru, darba apjomu, problēmām un izaicinājumiem.

### **7.1.1. Rezultāta novērtējums**

Uz darba rezultātu varu atskatīties divejādi – no pasūtītāja un no personīgas perspektīvas. Protams, ka rīks un tā programmatūra pirmkārt bija darbs, atbildīgs darbs. Nozīmīgi bija izstrādāt iespējamī labu un kvalitatīvu gala produktu pasūtītājam. Sekundāri izmantoju sev šo kā vērtīgāko mācīšanās un pieredzes uzkrāšanas iespēju līdz šim.

Esmu gandarīts par iespēju iepazīties ar jaunām programmēšanas jomām un izmēģināt teorijā apgūto arī praksē. Paralēli programmatūrai tiku iepazīties arī ar elektronikas izstrādi – man diezgan jaunu praksi. Šī pieredze devusi man labāku ieskatu savas nākotnes plāniem, potenciālām studiju un karjeras iespējām. Jāatzīst arī, ka padziļināts darbs un izpēte vienā konkrētā projektā ir bijusi vērtīgākā manas izglītības pieredze, un esmu pateicīgs par iespēju veltīt tik daudz laika šim.

Process nav bijis tikai solis uz priekšu skolas un karjeras trepēs, bet gan pamats manai profesionālajai identitātei un nākotnes izaicinājumiem. Tas ir apliecinājums tam, ka man ir potenciāls sasniegt savus mērķus un sapņus, kā profesionālam programmēšanas tehnikim.

### **7.1.2. Uzdevumu sasniegšanas analīze**

Projekts tika izstrādāts lai piepildītu pasūtītāja, “SIA InPass” pasūtījumu. Nonācu kontaktā ar “SIA InPass”, jo izvēlējos šo firmu kā savu prakses vietu. Izvēle man šķita pašsaprotama : viņi ir vietējie līderi savā jomā, ir pretimnākoši jauniem talantiem un tādēļ, arī saskatīju labas izaugsmes iespējas sev. Pieņemt šo projektu bija izaicinājums. Kā ceturtā kursa studentam, kas nav strādājis šāda tipa uzņēmumā, tas šķita nedaudz biedējoši, bet apziņa, ka man apkārt ir uzticami mentori procesu ļoti atviegloja.

Man svarīgi bija demonstrēt pēdējo četru gadu laikā skolā apgūto un praktizēto. Īpaši noderēja pirmajā kursā izpētītie Arduino Uno mikrokontrolieri. Tiem ir līdzīgs princips kā manis programmētajam rīkam, kas ļāva labāk izprast to darbību.

Darba process nebija iepriekš uzstādīti, noteikti punkti kuriem sekot. Nedarbojos pēc stingra plāna, bet gan izmantojot Spējo programmatūras izstrādes metodi (Agile software development). Šī metode darba procesu vada cikliski, atbildot uz sešiem atslēgpunktiem, un tās rezultātā iespējams radīt veiksmīgāku gala produktu. Šī metode ļauj viegli pielāgoties arī gadījumos kur pasūtītājs maina savas velmes un prasības.

Izstrādes procesā lielākie izaicinājumi bija saistīti ar seriālo komunikāciju starp citiem mikročipiem un darbstaciju, pareizo programmatūras palīg-klašu izvēli un rīka saskarnes izkārtojumu. Šie aspekti tika rūpīgi pārdomāti un optimizēti, lai nodrošinātu ierīces un tās konfigurācijas aplikācijas efektīvu darbību un lietošanas ērtību gala lietotājam.

Tomēr, darba gaitā saskaros arī ar dažādiem mazākiem šķēršļiem un izaicinājumiem:

- Izpētīt un izprast pārbaudāmo mikročipu komponenšu funkcionēšanos ar programmaparatūrām, tā skaitā noskaidrot visus iespējamos komponenšu funkcionēšanos scenārijus pārbaudot mikročipus.
- Izprast mikrokontrollieru sadarbību ar citām ierīcēm, komponentēm un dažādu protokolu struktūrām, lai nodrošinātu veiksmīgu programmatūras izpildi.

Pats varu secināt, ka izstrādājot programmatūru, nēmu vēra visas prasības ko pasūtītājs izvirzījis un tās izpildīju, lai nodrošinātu atbilstību pasūtītāja zīmola identitātei. Man arī nozīmīgi bija veikt pasūtījumu tā, lai tas atbilstu manis uzstādītiem, personīgajiem standartiem. Izstrādātās programmatūras, kuras pasūtītājs šobrīd aktīvi ikdienā izmanto uzņēmumā “SIA InPass”, apstiprina, ka programmatūru gala rezultāts ir veiksmīgs un efektīvi integrēts uzņēmuma operatīvajos procesos.

### 7.1.3. Darba apjoms

Tā kā programmatūras funkcionālais process ir diezgan sarežģīts un visaptverošs, tā darba apjoms, manā uztverē, ir plašs. Darbu kavēja un apgrūtināja dažādi ārēji apstākļi, materiālu trūkums, kā arī citas atbildības, bet neskatoties uz tām gala rezultāts ir kaut kas ar ko gan pats, gan prakses vieta, “SIA InPass”, var lepoties. Darbs noteikti bija apjomīgs, un vislabāk to varu atspoguļot ar reāliem skaitļiem:

- Python failu skaits ir 14 un “.JSON” failu skaits ir 3.
- Funkciju un metožu skaits ir aptuveni 52.
- Aptuvenais kopējais komandrindu un “.JSON” datu skaits ir 4000 (neieskaitot tukšumus vai komentārus).
- Kopējais komandrindu un “.JSON” datu skaits rīka programmatūrai ir 3429 (neieskaitot tukšumus vai komentārus).
- Kopējais komandrindu un “.JSON” datu skaits konfigurācijas aplikācijas programmatūrai ir 403 (neieskaitot tukšumus vai komentārus).
- Patēriņtais laiks programmatūru un dokumentācijas izstrādei ir bijuši aptuveni 2 pilni mēneši.

## 8. Lietoto terminu un saīsinājumu skaidrojumi

14.tabula

**Terminu un saīsinājumu skaidrojumi**

<b>Termins vai saīsinājums</b>	<b>Skaidrojums</b>
TT/CU skeneris	Izstrādātā produkta nosaukums.
Mikročips	Pusvadītāju materiāla plāksne ar pievienotām komponentēm .
TTunit	Mikročips ko veido un izmanto uzņēmums “SIA InPass” laika uzskaitei.
ControlUnit	Mikročips ko veido un izmanto uzņēmums “SIA InPass” dažādu mehatronisku elementu darbībām.
Mikrokontrollieris	Vadības ierīce jeb komponente, kas ietver mikroprocesoru.
Real-Time-Clock	Digitāla pulkstena komponente ar primāro funkciju, lai precīzi sekotu laikam pat tad, ja strāvas padeve ir izslēgta.
RTC	Saīsinājums terminam Real-Time-Clock.
EEPROM	Rakstāma un lasāma atmiņa, kuras saturu var izdzēst un pārprogrammēt, izmantojot impulsa spriegumu.
EEPROM lapa	Atmiņas bloks EEPROM komponentē, ko izmanto neatkarīgām lasīšanas un rakstīšanas darbībām.
Zumers	Elektriskā signāla komponente, kas rada dūkstošu skaņu.
RFID	Tehnoloģija, kas izmanto radioviļņus, lai bezvadu režīmā identificētu objektus.
RFID karšu lasītājs	Ierīce, kas nolasa RFID radioviļņus un izvada datus uz procesora.
UART	Saīsinājums terminam Universālais asinhronais uztvērējs-raidītājs. Tas ir mikrokontrollieru sakaru protokols, ko izmanto seriālai saziņai starp ierīcēm.
ControlUnit adreses	Ligzdas uz ControlUnit mikročipa, kas izvada datus no citām ierīcēm uz mikrokontrolliera.
Wiegand	Sakaru protokols, ko parasti izmanto piekļuves kontroles sistēmās, lai pārsūtītu datus starp karšu lasītājiem un vadības paneļiem.
IDE	Saīsinājums terminam Integrētā izstrādes vide.
SSH	Kriptogrāfijas tīkla protokols, kas nodrošina drošu saziņu neaizsargātā tīklā, un ļauj lietotājiem droši piekļūt un pārvaldīt attālās sistēmas un ierīces.
Micro-USB	Standarta savienotājs, ko plaši izmanto mobilo ierīču savienošanai uzlādes nolūkos.
2x20 ligzda	Ligzdas uz ControlUnit un TTunit mikročipa, kas sniedz strāvu citiem mikročipiemiem, kā arī datu plūsmu starp mikrokontrollieriem.
16_VIN	Galvenā strāvas ligzda uz ControlUnit mikročipa.
Tkinter	Python bibliotēka ar kuras palīdzību var izveidot grafiskos lietotāja interfeisa logus.

## **9. Literatūras un informācijas avotu saraksts**

Nodaļā tiek uzskaitīta izstrādes laikā izmantotā literatūra un informācija.

- Micropython programmaparatūras dokumentācija.  
Avots: <https://docs.micropython.org/en/latest/>
- Tkinter grafiskā lietotāja interfeisa dokumentācija.  
Avots: <https://docs.python.org/3/library/tk.html>
- Wiegand protokola izmantošanas dokumentācija  
Avots: <https://raspberrypi.stackexchange.com/questions/100156/how-to-read-wiegand-serial-data-rx-tx>
- UART komunikācijas izmantošanas dokumentācija  
Avots: <https://docs.micropython.org/en/latest/library/machine.UART.html>

## **Pielikumi**

## 1.pielikums

```
# Bezgaliga cikla funkcija, kas veic datu plūsmu ar darbstaciju.
def SendReceiveData(self):

    # Veic datu plūsmas apstiprinājuma uzgaidi.
    poll_obj = select.select()
    poll_obj.register(sys.stdin, select.POLLIN)

    # Bezgalīgs cikls, programmatūra netiek virzīta vairs uz citu sazarojumu.
    while True:
        while True:

            # Ja datu plūsmas apstiprinājums tiek saņemts.
            poll_results = poll_obj.poll(1)
            if poll_results:

                # Nolasa apstiprinājuma ziņu no darbstacijas.
                data = str(sys.stdin.readline().strip())

                if data == "receive":

                    # Nosūta apstiprinājuma atbildi uz darbstaciju.
                    print("transfer")
                    sys.stdout.write("transfer\r")

                    # Nosūta mikročipu pārbaudes vēsturi uz darbstaciju.
                    for hist in self.history_conf["history"]:
                        print(hist, "\n")
                        utime.sleep(0.01)

                    # Nosūta izmantojamo saskarnes valodu uz darbstaciju.
                    print(self.history_conf["language"], "\n")

                    # Nosūta ziņu, ka datu plūsma ir pabeigta
                    print("END\n")

                    # Izveido jaunu displeja kadru, kas liecina, ka rīks savienots ar darbstaciju.
                    self.NewFrame("pc")

                    self.display.set_pos(10, 60)
                    self.display.print(self.language_dict[self.current_language][0]["pc_connected_label"])

                    break
```

## 2.Pielikums

```
# Funkcija, kas veic pirms pārbaudi par pievienotā mikročipa EEPROM komponenti.
def EepromPreTest(self):
    try:
        for i in range(10):
            self.eeprom.write(0, "test")
            eepromVal=self.eeprom.read(0,4)
    except:
        eeprom_check = False
        print(eeprom_check)
    else:
        eeprom_check = True
        print(eeprom_check)

    try:
        self.eeprom.wipe()
    except:
        eeprom_check = False

    return eeprom_check

# Funkcija, kas veic pirms pārbaudi par pievienotā ControlUnit mikročipa komunikācijas stāvokli.
def UartComPreTest(self):
    try:
        self.M.Queue("GET","board_version")
        self.M.Send(True)
        self.M.Receive(True)

        checkU=self.uart_id.read()

    except Exception:
        checkU = None

    return checkU
```

### 3.Pielikums

```
# Real-Time-Clock komponentes pārbaudes funkcija.
def RTC_check(self):

    # Izveido jaunu kadru, kas liecina, ka tiek pārbaudita komponente.
    self.NewFrame(self.page)

    self.display.set_pos(10, 60)
    self.display.set_color(color565(0, 0, 0), color565(255, 255, 255))
    self.display.print(self.language_dict[self.current_language][0]["testing_rtc"])

    # Saglabā šī briža saglabāto laiku no Real-Time-Clock komponentes.
    oldRtc=self rtc.datetime()

    # Ieraksta jaunu datumu uz Real-Time-Clock komponentes.
    try:
        self.rtc.datetime((2020, 1, 21, 2, 10, 32, 36, 0))

        self.results_list.append(self.language_dict[self.current_language][0]["rtc_write_ok"])
        self.results_list_to_send.append("rtc-true:")

    # Ja Real-Time-Clock komponentes rakstišana nestrādā.
    except:
        self.results_list.append(self.language_dict[self.current_language][0]["rtc_write_er"])
        self.results_list_to_send.append("rtc-false:")
        self.i2c_devices[1][2]='False'

    # Nolasa vai tikko ierakstītais datums sakrit ar gadu un minūtēm.
    try:
        if self.rtc.datetime()[0] == 2020 and self.rtc.datetime()[4] == 10:

            self.results_list.append(self.language_dict[self.current_language][0]["rtc_read_ok"])

            if "rtc-false:" not in self.results_list_to_send and "rtc-true" not in self.results_list_to_send:
                self.results_list_to_send.append("rtc-true:")

    # Ja Real-Time-Clock komponentes lasīšana nestrādā.
    except:
        self.results_list.append(self.language_dict[self.current_language][0]["rtc_read_er"])

        if "rtc-false:" not in self.results_list_to_send:

            if "rtc-true" in self.results_list_to_send:
                self.results_list_to_send[1] = "rtc-false:"
            else:
                self.results_list_to_send.append("rtc-false:")

        self.i2c_devices[1][2]='False'

    # Ievieto atpakaļ originālo laiku uz Real-Time-Clock komponentes
    self.rtc.datetime(oldRtc)
```

#### 4.pielikums

```
# Funkcija, kas ielādē aplikācijas saskarnes vārdnīcu.
def GetConf(self):
    with open(path.abspath(path.join(path.dirname(__file__), 'conf.json')), encoding='utf-8') as f:
        self.language = json.load(f)

# Funkcija, kas pārmaina aplikācijas saskarnes valodu un to ielādē aplikācijas logā.
def ChangeLanguageHandler(self):

    if self.current_language == "english":
        self.current_language = "latvian"

    elif self.current_language == "latvian":
        self.current_language = "english"

    self.navbar.destroy()
    self.content_box.destroy()
    self.CreateWindow()

# Funkcija, kas veic rīka pārbaudes rezultātu izdzēšanu.
def DeleteHistoryHandler(self):
    if self.device_connected:
        if SendToDevice(self.device_port_value, "delete") == b'deleted':

            # Izvāda atsevišķu logu, kas signalizē par datu izdzēšanu.
            messagebox.showwarning(self.language[self.current_language][0]["deleted_history_title"], self.language[self.current_language][0]["deleted_history_message"])

            # No jauna iegūst ierices datus.
            self.CheckDevice()

# Funkcija, kas veic rīka valodas saskarnes maiņu.
def ConfigureHandler(self):
    if self.device_connected:
        if self.language_option_val.get() == "Latvian" or self.language_option_val.get() == "Latviešu":
            res = SendToDevice(self.device_port_value, "lat")

        elif self.language_option_val.get() == "English" or self.language_option_val.get() == "Angļu":
            res = SendToDevice(self.device_port_value, "eng")

        if res == b'lat_configured' or res == b'eng_configured':
            messagebox.showwarning("Warning!", self.language[self.current_language][0]["changed_language_message"]+self.language_option_val.get())
            self.CheckDevice()
```

**5.pielikums**

Izstrādāto programmatūru pirmkods pieejams versionēšanas platformā GitHub, skatīt:

<https://github.com/pulkitisRenars/TT CU tool>