Blokovsko sučelje za programiranje Infineon XMC serije mikroupravljača

Kaučić, Luka

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:397965

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

BLOKOVSKO SUČELJE ZA PROGRAMIRANJE INFINEON XMC SERIJE MIKROUPRAVLJAČA

Diplomski rad

Luka Kaučić

Osijek, 2024.

Sadržaj

1 UVOD **1**

2 SUSTAVI VIZUALNOG PROGRAMIRANJA **3**

2.1 Scratch . . . 3

2.2 Blockly . . . 3

2.3 Modkit . . . 3

2.4 Node-RED . . . 4

2.5 Arduviz . . . 4

2.6 OpenPLC . . . 4

2.7 MIT App Inventor . . . 5

2.8 LabVIEW . . . 5

2.9 Code.org . . . 5

2.10 Snap! . . . 5

2.11 MakeBlock . . . 6

2.12 MATLAB Simulink . . . 6

3 KORIŠTENE TEHNOLOGIJE **8**

3.1 XMC mikroupravljači . . . 8

3.1.1 XMC 1400 . . . 8

3.1.2 XMC 4700 . . . 9

3.2 ModusToolbox razvojno okruženje . . . 10

3.2.1 Stvaranje projekta . . . 11

3.2.2 Postavljanje mogućnosti uređaja . . . 12

3.2.3 Izgradnja projekta i programiranje uređaja . . . 13

3.3 JavaScript . . . 13

3.4 JSON . . . 14

3.5 Blockly . . . 14

3.6 Python . . . 14

3.6.1 Flask biblioteka . . . 15

3.6.2 OS biblioteka . . . 15

3.7 Mocha . . . 15

3.8 Chai . . . 15

4 IMPLEMENTACIJA BLOKOVSKOG SUČELJA **17**

4.1 Razvoj vlastitih blokova 20

4.2 Razvoj vlastitog prevoditelja 26

4.3 Raspoznavanje priključenog uređaja 27

4.4 Izgradnja projekta i programiranje mikroupravljača 29

5 TESTIRANJE BLOKOVSKOG SUČELJA **31**

5.1 Unit testovi 31

5.2 End-To-End testovi 33

5.2.1 Trčeće svjetlo 34

5.2.2 Kontroler i RGB LE dioda 34

6 ZAKLJUČAK **37**

Literatura **40**

Sažetak **41**

Abstract **42**

Životopis **43**

Popis tablica **44**

Popis slika **45**

Popis programskih kodova **46**

1 UVOD

Korištenje ugradbenih računalnih sustava značajno je poraslo u popularnosti i dostupnosti u posljednjem desetljeću [1]. Ovaj veliki rast doveo je do toga da se izrada softvera za ugradbene sustave povjeri i manje iskusnim programerima [2]. Zbog toga je postalo nužno pronaći nove, jednostavnije i pristupačnije tehnike programiranja. Primjena već etabliranih alata koji se koriste u obuci početnika u programiranju, kao što su alati za vizualno programiranje, mogu se koristiti u ovoj situaciji [3], [4]. U ovom radu predstavljen je jednostavan i intuitivan koncept programiranju ugradbenih sustava, s posebnim naglaskom na koncept blokovskog vizualnog programiranja. Predloženi sustav napravljen je u svrhu jednostavnijeg programiranja Infineon XMC serije mikroupravljača..

Istraživanja su pokazala da vizualno programiranje značajno poboljšava iskustvo programiranja kako za početnike koji rade s mikroupravljačima. Autori u [5] koriste STM IDE, koji dijeli funkcionalne sličnosti s ModusToolbox alatom, službenim alatom za programiranje Infineonovih mikroupravljača te su istaknuli kako sustavi vizualnog programiranja pojednostavljuju proces programiranja putem jednog, jednostavnog sučelja koje objedinjuje uređivanje, prevođenje koda i otklanjanje pogrešaka. Ovaj pristup nudi intuitivnu početnu točku u ugradbenom programiranju za početnike, čineći složene koncepte programiranja dostupnijima, dok istovremeno pruža sveobuhvatne funkcije koje su iskusnim programerima potrebne za učinkovito izvođenje sofisticiranih projekata [6].

Ovaj rad usredotočen je na istraživanju i integraciji Blocklyja, Googleovog JavaScript okruženja otvorenog koda za izradu koncepta blokovskog programskog sustava te integraciju istog u razvojni proces za Infineonovu XMC seriju mikroupravljača. Fleksibilnost Blocklyja u stvaranju prilagođenih blokova i njegova prilagodljivost različitim programskim okruženjima čine ga idealnim kandidatom za ovu primjenu. Cilj je pokazati ovo kroz razvoj i testiranje specifičnih Blockly blokova za XMC seriju mikroupravljača.

Važnost ovog koncepta leži njegovim praktičnim primjenama, a i u njegovom doprinosu STEM obrazovanju. Prema istraživanju [7], studenti koji su koristili blokovsko vizualno programiranje postigli su viši stupanj vještina računalnog razmišljanja i pokazali veći interes za programiranje, što sugerira da ulaganje u ovakvu vrstu obrazovanja može imati pozitivan utjecaj na učenike zainteresirane za karijeru u ovom području.

Ostatak ovog rada strukturiran je na sljedeći način: pregled i usporedba relevantne literature na temu rada dani su u poglavlju 2. U 3. poglavlju dan je pregled korištenih tehnologija

(sklopovlja i programske podrške), dok je u četvrtom poglavlju opisana arhitektura i implementacija predloženog blokovskog sučelja. U 5. poglavlju opisane su metode odabrane za testiranje pojedinih komponenata sustava i cjelokupnog sustava, nakon čega je dan zaključak s osvrtom na cijeli rad.

2 SUSTAVI VIZUALNOG PROGRAMIRANJA

Vizualno programiranje posljednjih je godina postalo popularno kao pristup koji omogućuje korisnicima bez programskog iskustva da razvijaju složene softverske aplikacije kroz intuitivna, grafička sučelja. Takvi sustavi koriste blokove koji predstavljaju različite programske naredbe, omogućujući korisnicima da vizualno sastavljaju programe bez potrebe za pisanjem koda. U ovom poglavlju razmatramo nekoliko ključnih sustava vizualnog programiranja, s naglaskom na njihovu primjenu u obrazovanju i industriji.

2.1 Scratch

Jedan od najpoznatijih alata za vizualno programiranje je Scratch. Scratch, koji je razvio MIT Media Lab, usmjeren je na uvođenje djece i mladih u svijet programiranja. Korištenjem jednostavnog blokovskog sučelja, korisnici mogu kreirati programe spajanjem vizualnih blokova koji predstavljaju različite funkcije, kao što su petlje, uvjeti i događaji. Ovaj alat se široko koristi u obrazovnim institucijama diljem svijeta, te je prepoznat kao jedan od najuspješnijih alata za učenje programiranja, posebno zbog svoje jednostavnosti i pristupačnosti [8].

2.2 Blockly

Blockly je još jedan značajan alat, razvijen od strane Googlea, koji omogućuje stvaranje prilagođenih blokova i njihovu integraciju u različita programska okruženja. Blockly pruža fleksibilnost koja omogućuje njegovu primjenu u širokom rasponu aplikacija, uključujući razvoj softvera za ugradbene sustave. Njegova struktura omogućuje korisnicima da lako prilagode blokove specifičnim potrebama svojih projekata, čime se otvaraju mogućnosti za razvoj prilagođenih sustava koji su lako dostupni i razumljivi i početnicima i iskusnim programerima [3]. Blockly je osnova za nekoliko drugih alata, uključujući Modkit, Node-RED, i Arduviz.

2.3 Modkit

Modkit je alat temeljen na Blocklyju koji je posebno prilagođen za programiranje mikrokontrolera. Ovaj alat koristi blokove za predstavljanje funkcija specifičnih za mikrokontrolere, čineći proces programiranja pristupačnim i početnicima. Modkit se koristi u edukativnim

kontekstima, gdje omogućuje studentima i mladim inženjerima da lako uče o osnovama mikrokontrolerskog programiranja bez potrebe za savladavanjem složenih programskih jezika [9].

2.4 Node-RED

Node-RED je alat koji koristi vizualno programiranje za povezivanje hardverskih uređaja, API-ja i online usluga u jednostavne logičke protoke. Posebno je popularan u Internet of Things (IoT) zajednici, gdje se koristi za brzo prototipiranje IoT rješenja. Korisnici mogu jednostavno povlačiti i ispuštati funkcionalne blokove kako bi definirali tokove podataka između različitih uređaja i usluga, bez potrebe za dubokim tehničkim znanjem [10].

2.5 Arduviz

Pratomo i Perdana [11] nude rješenje za vizualno programiranje Arduino razvojnih ploča pod nazivom Arduviz. Arduviz je razvijen s ciljem da objedini prednosti postojećih alata poput Ardublock i miniBloq, koji omogućuju vizualno programiranje Arduino ploča, ali svaki ima svoje nedostatke. Arduviz koristi Blockly kao bazu zbog jednostavnosti i fleksibilnosti te Electron okruženje za Desktop aplikaciju. Prevođenje koda odvija se s pomoću Arduino Builder alata, dok se za prijenos na ploču koristi AVRDUDE uploader alat. Arduviz omogućuje korisnicima stvaranje programa povlačenjem i ispuštanjem blokova u grafičkom sučelju. Sustav je testiran na nekoliko jednostavnih primjera, uključujući Blink primjer i program s LDR senzorom. Blink primjer uključuje LE diodu koja treperi u određenim intervalima, dok LDR primjer upravlja LE diodom u ovisnosti o vrijednosti fotootpornika. Autori planiraju proširenje funkcionalnosti sustava na naprednije mogućnosti, kao što su I2C i SPI protokoli, ali ne navode točno na kojim serijama Arduino razvojnih ploča sustav radi.

2.6 OpenPLC

Još jedan primjer alata za vizualno programiranje je OpenPLC, koji se koristi za programiranje PLC (Programmable Logic Controller) sustava. OpenPLC omogućuje korisnicima da kreiraju složene automatizirane sustave s pomoću vizualnih blokova, čime se značajno pojednostavljuje razvoj industrijskih aplikacija. Ovaj alat je posebno koristan u industrijskim okruženjima gdje se programiranje PLC-a često koristi za kontrolu proizvodnih procesa i

automatizaciju [12].

2.7 MIT App Inventor

U području razvoja mobilnih aplikacija, App Inventor predstavlja jedan od najpopularnijih alata. Razvijen na MIT-u, App Inventor omogućuje korisnicima da kreiraju Android aplikacije koristeći jednostavno, blokovsko sučelje. Ovaj alat je posebno koristan u obrazovnim ustanovama jer omogućuje učenicima da brzo razvijaju aplikacije, čime se smanjuje krivulja učenja koja je obično povezana s razvojem mobilnih aplikacija. App Inventor također omogućuje korisnicima da odmah vide rezultate svog rada na stvarnim uređajima, što dodatno motivira učenike [13].

2.8 LabVIEW

LabVIEW je još jedan važan alat u području vizualnog programiranja, ali s naglaskom na znanstvene i inženjerske aplikacije. LabVIEW se koristi za obradu signala i kontrolu mjernih uređaja, a njegovo sučelje omogućuje korisnicima da kreiraju složene algoritme za obradu podataka bez potrebe za pisanjem koda. Ovaj alat je široko korišten u znanstvenim istraživanjima i industriji, gdje se koristi za analizu podataka i automatizaciju eksperimenata [14].

2.9 Code.org

Za obrazovanje mlađih generacija, Code.org nudi blokovske lekcije koje su posebno dizajnirane za podučavanje osnovnih koncepata programiranja. Ovaj alat omogućuje učenicima da kroz interaktivne lekcije uče programiranje koristeći blokove koji predstavljaju jednostavne naredbe. Code.org je posebno usmjeren na popularizaciju računalnog obrazovanja među djecom i mladima, te se koristi u školama širom svijeta [15].

2.10 Snap!

Snap! je alat koji se temelji na istim principima kao i Scratch, ali s naprednijim mogućnostima, uključujući podršku za korisnički definirane funkcije i rekurziju. Snap! je dizajniran s ciljem pružanja fleksibilnijeg okruženja za učenje programiranja, što ga čini pogodnim za

naprednije korisnike koji žele proširiti svoje znanje izvan osnovnih koncepata [16].

2.11 MakeBlock

Makeblock nudi platformu mBlock koja omogućava jednostavno blokovsko programiranje, posebno prilagođeno za edukaciju u STEM području. mBlock koristi vizualno programiranje gdje korisnici, s pomoću jednostavnog povlačenja i slaganja blokova, mogu kreirati kompleksne programe. Ova platforma podržava više uređaja, uključujući robote poput mBot-a i CyberPi-ja, te omogućava prelazak na Python kodiranje za naprednije korisnike. mBlock je besplatan za korištenje [17].

2.12 MATLAB Simulink

MATLAB Simulink je platforma za blokovsko modeliranje i simulaciju dinamičkih sustava, široko korištena u industriji i akademiji za razvoj i testiranje složenih sustava. Simulink omogućava korisnicima da vizualno modeliraju, simuliraju i analiziraju sustave povlačenjem i slaganjem blokova koji predstavljaju različite matematičke funkcije, ulaze, izlaze i dinamiku sustava. Platforma je posebno korisna za inženjere i znanstvenike u područjima poput automatike, kontrolnih sustava, signalne obrade i modeliranja fizičkih sustava. Simulink se integrira s MATLAB-om, omogućavajući prelazak na pisanje koda za naprednije analize i prilagodbe modela. Ova platforma nije besplatna, ali je dostupna kroz različite licence [18]

Iako svi ovi alati imaju zajednički cilj olakšavanja programiranja korištenjem vizualnog sučelja, razlikuju se po svojoj složenosti, ciljanoj publici i područjima primjene. Scratch i Snap! su primarno usmjereni na edukaciju i uvod u programiranje, dok su Blockly i njegovi derivati, kao što su Modkit, Node-RED, i Arduviz, namijenjeni širem spektru korisnika, uključujući i one s tehničkim predznanjem. OpenPLC i LabVIEW su specijalizirani alati koji se koriste u industrijskim aplikacijama, dok App Inventor cilja na razvoj mobilnih aplikacija. S druge strane, Code.org je fokusiran na obrazovanje, s naglaskom na jednostavnost i pristupačnost za mlađe učenike. Usporedba navedenih sustava vizualnog programiranja dana je tablicom 2.1.

Tablica 2.1. Usporedba različitih sustava vizualnog programiranja

Naziv sustava	**Besplatno**	**Namjena**	**Link na stranicu**
Scratch	Da	Općenito (obrazovanje)	`https://scratch.mit.edu/`
Blockly	Da	Različite (ovisno o implementaciji)	`https://developers.google.com/blockly`
Modkit	Da	više mikroupravljača	`https://www.modkit.com/`
Node-RED	Da	Općenito (IoT, aplikacije)	`https://nodered.org/`
Arduviz	Da	Arduino	N/A
OpenPLC	Da	PLC	`http://www.openplcproject.com/`
App Inventor	Da	Android aplikacije	`http://appinventor.mit.edu/`
LabVIEW	Ne	Znanstvene i inženjerske aplikacije	`https://www.ni.com`
Code.org	Da	Općenito (obrazovanje)	`https://code.org/`
Snap!	Da	Općenito (obrazovanje)	`https://snap.berkeley.edu/`
MakeBlock	Da	mbot, CyberPi	`https://www.makeblock.com`
MATLAB Simulink	Ne	modeliranje sustava	`https://www.mathworks.com/products/simulink.html`

3 KORIŠTENE TEHNOLOGIJE

3.1 XMC mikroupravljači

Mikorkoupravljači su računala vrlo malih dimenzija na jednom integriranom krugu, a čine ih CPU, memorija i ulazi/izlazi. Inifneonova XCM serija nudi 32-bitne mikrokontrolere temeljene na RISC arhitekturi, a primjenjive u industriji, transportu i kućnoj uporabi. Navedeno je vidljivo i iz Infineonovog izvještaja iz 2020. godine [19] u kojem je navedeno da broj prodanih XMC mikrokontrolera prelazi sto milijuna. U nastavku je dana usporedba skupina unutar XMC serije te su opisani konkretni XMC mkroupravljači i njihove funkcionalnosti. Unutar XMC serije mikrokontrolera postoje tri skupine: XMC 1000 s ARM Cortex M0 procesorom, XMC 4000 s ARM Cortex M4 procesorom i XMC 7000 s ARM Cortex M7 procesorom. Usporedba navedenih skupina dana je tablicom 3.1.

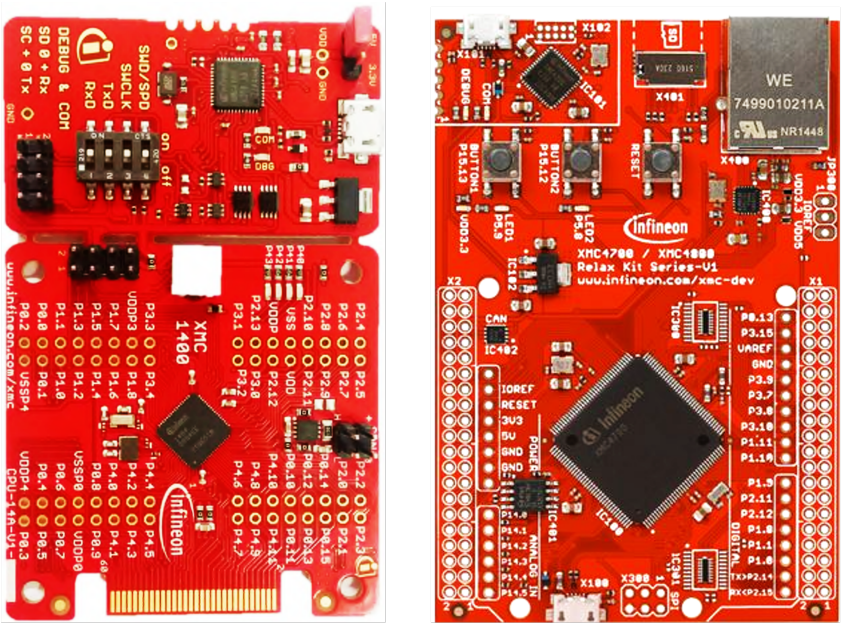
Tablica 3.1. Usporedba skupina XMC serije mikrokontrolera

XMC Skupina	**1000**	**4000**	**7000**
ARM Cortex procesor	M0	M4F	M7
Frekvencija procesora	do 48 MHz	do 144 MHz	do 350 MHz
Flash memorija	8-200 kB	64K-2 MB	1-8 MB
Broj pinova	16-64	48-196	100-272

3.1.1 XMC 1400

Infineon XMC 1400 serija mikroupravljača (Slike 3.1.-a) temelji se na ARM Cortex-M0 jezgrama i namijenjena je za aplikacije koje zahtijevaju nisku potrošnju energije i visoku efikasnost. Ovi mikroupravljači nude do 200 kB flash memorije i do 16 kB RAM-a.
Ključne karakteristike XMC 1400 serije:

- ARM Cortex-M0 jezgra: Radni takt do 48 MHz, omogućujući visoke performanse uz nisku potrošnju energije.



(a) (b)

Slika 3.1. XMC-1400 (a) i XMC-4700 (b) razvojni sustavi

- Bogati set periferija: Uključuje analogno-digitalne pretvarače (ADC), digitalno-analogne pretvarače (DAC), PWM module i serijske komunikacijske module (UART, SPI, I2C).
- Visoka integracija: Integrirani moduli za upravljanje energijom, sigurnosne značajke i komunikacijske protokole.
- Niska potrošnja energije: Optimiziran za aplikacije koje zahtijevaju dugotrajni rad na bateriji, s naprednim načinima rada za uštedu energije.
- Podrška za industrijske standarde: Kompatibilnost s industrijskim komunikacijskim standardima, čime se olakšava integracija u različite sustave.

3.1.2 XMC 4700

Infineon XMC 4700 serija mikroupravljača (Slika 3.1.-b) temelji se na ARM Cortex-M4 jezgrama i namijenjena je za zahtjevnije aplikacije koje zahtijevaju visoke performanse i napredne značajke. Ovi mikroupravljači nude do 2 MB flash memorije i do 352 kB RAM-a. Ključne karakteristike XMC 4700 serije:

- ARM Cortex-M4 jezgra: Radni takt do 144 MHz, s ugrađenom jedinicom za računske operacije s pokretnim zarezom (FPU), omogućujući visok stupanj obrade podataka.

- Napredni set periferija: Uključuje višekanalne ADC i DAC, PWM module visoke rezolucije, Ethernet MAC, CAN moduli i USB host/OTG podršku.
- Integrirane sigurnosne značajke: Uključuju sigurnosne module za kriptografske operacije i zaštitu memorije.
- Širok raspon primjena: Namijenjen za industrijske aplikacije, automatizaciju, upravljanje motorima i IoT uređaje
- Visoka fleksibilnost: Mogućnost konfiguracije za specifične potrebe aplikacije, što omogućuje optimizaciju performansi i potrošnje energije.

3.2 ModusToolbox razvojno okruženje

ModusToolbox je sveobuhvatno razvojno okruženje tvrtke Infineon Technologies, koje omogućuje jednostavan i efikasan razvoj aplikacija za njihove mikroupravljače, uključujući XMC seriju. ModusToolbox integrira različite alate i okvire u jedno sučelje, omogućujući programerima da brzo postave, razvijaju, testiraju i implementiraju svoje projekte [20]. Glavne komponente ModusToolboxa uključuju:

- *Project Creator*: Alat za brzo stvaranje projekata pomoću unaprijed definiranih predložaka. Omogućuje jednostavno postavljanje osnovnih parametara projekta i odabir ciljnih mikroupravljača.
- *Device Configurator*: Grafičko sučelje za konfiguraciju uređaja, omogućujući korisnicima da vizualno postave pinove, periferne module i druge konfiguracijske parametre. Ovaj alat automatski generira potrebne zaglavlja i inicijalizacijske datoteke na temelju odabranih postavki.
- *Middleware Libraries*: ModusToolbox uključuje bogat skup middleware knjižnica koje podržavaju razne komunikacijske protokole, upravljanje motorima i druge funkcionalnosti. Ove knjižnice pojednostavljuju integraciju složenih funkcionalnosti u aplikacije.
- Podržane razvojne okoline: ModusToolbox je kompatibilan s popularnim razvojnim okruženjima kao što su Eclipse i Visual Studio Code. Ovo omogućuje programerima da koriste alate s kojima su već upoznati, poboljšavajući produktivnost i smanjujući krivulju učenja.

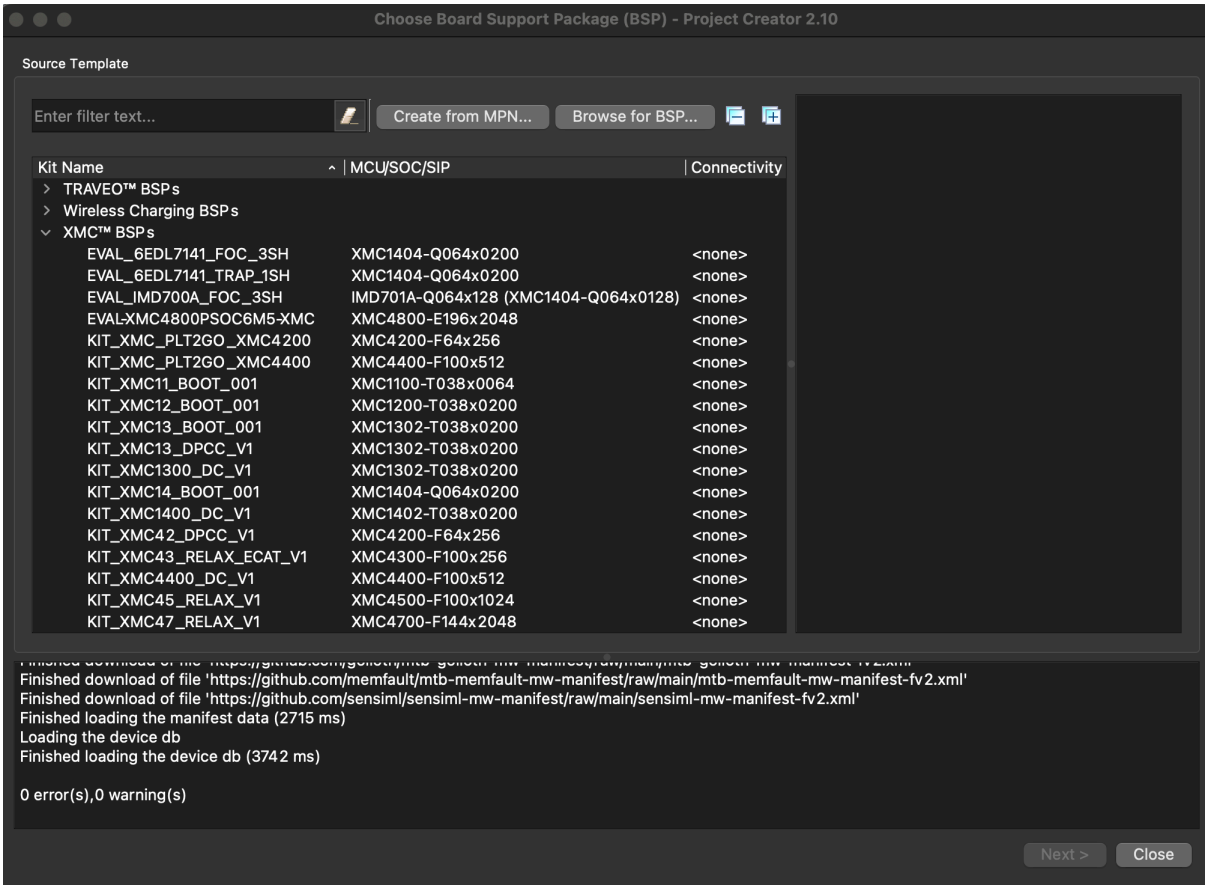
- *Command Line Interface* alati (CLI): Uz grafičke alate, ModusToolbox također nudi alate naredbenog retka koji omogućuju automatizaciju različitih zadataka kao što su izgradnja projekta, programiranje uređaja i upravljanje konfiguracijama.
- Alati za *debugging* i testiranje: ModusToolbox uključuje napredne alate za ispitivanje i debugiranje aplikacija. To uključuje mogućnosti za profiliranje performansi, praćenje izvršavanja koda i otkrivanje grešaka.

Stvaranje i programiranje aplikacije svodi se na sljedeće korake:

1. stvaranje projekta
2. postavljanje mogućnosti uređaja
3. pisanje koda
4. programiranje uređaja

3.2.1 Stvaranje projekta

Projekt je moguće stvoriti pomoću *Project-creator* alata (Slika 3.2.). Potrebno je odabrati razvojni sustav i mikroupravljač za koji se stvara projekt. Osim korištenjem grafičkog sučelja, projekt je moguće stvoriti i pokretanjem odgovarajuće naredbe u naredbenom retku. Pri tome je potrebno navesti razvojni sustav za koji se projekt stvara te predložak projekta koji se želi stvoriti (primjerice *hello-world* predložak)[21].



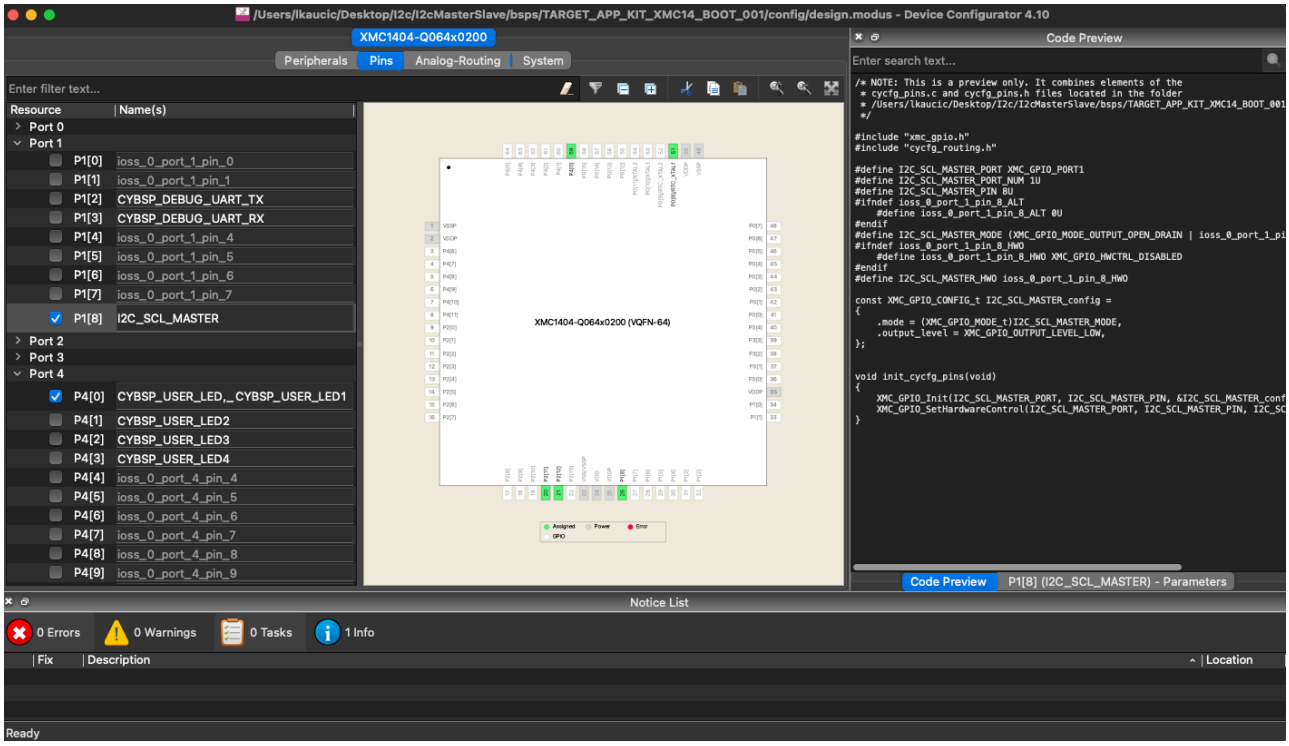
Slika 3.2. Grafičko sučelje Project-creator alata

3.2.2 Postavljanje mogućnosti uređaja

Nakon stvaranja projekta, a prije pisanja koda, potrebno je postaviti mogućnosti uređaja koje se u aplikaciji namjeravaju koristiti. Navedeno se može postići pomoću *device-configurator* grafičkog sučelja (Slika 3.3.). Neke od mogućnosti koje ovaj alat nudi su [22]:

- postavljanje pinova koji se koriste
- postavljanje kontrolera prekida
- postavljanje parametara CAN protokola

Ovisno o postavljenim parametrima i odabranim mogućnostima unutar *device-configurator* alata stvaraju se i popunjavaju odgovarajuće datoteke zaglavlja. Za razliku od stvaranja projekta, postavljanje dodatnih mogućnosti uređaja nije moguće preko jednostavne naredbe naredbenog retka. Alternativa grafičkom sučelju je samostalno stvaranje i popunjavanje datoteka zaglavlja.



Slika 3.3. Grafičko sučelje Device-configurator alata

3.2.3 Izgradnja projekta i programiranje uređaja

Izgradnja projekta (*engl. build*) nije nužna, zato što je sastavni dio procesa programiranja uređaja (svakako se izvodi prije programiranja uređaja). No, ukoliko se aplikacija razvija bez priključenog uređaja, moguće je zasebno izgraditi aplikaciju kako bi se provjerila točnost koda. Proces izgradnje projekta temelji se na alatu *GNU make* koji stvara izvršne datoteke prema *makefile* datoteci s odgovarajućim postavkama. Pokretanje procesa izgradnje ovisi o razvojnom okruženju koje se koristi, no moguće ga je pokrenuti i korištenjem *make build* naredbe unutar naredbenog retka. Nakon što je uređaj spojen i program spreman, uređaj je moguće programirati pomoću *make program* naredbe. Moguće je koristiti i naredbu *make qprogram* ukoliko se proces izgradnje želi preskočiti.

3.3 JavaScript

JavaScript je dinamički, interpretirani programski jezik korišten za razvoj interaktivnih web stranica i aplikacija. Omogućuje izradu interaktivnih elemenata na web stranicama te manipulaciju DOM-om, što omogućuje promjenu strukture i sadržaja web stranica u stvarnom vremenu. Podržava asinkrone operacije poput AJAX-a za dohvaćanje podataka s posluži-

telja, čime se poboljšava učinkovitost i responzivnost web aplikacija. JavaScript podržava različite paradigme programiranja, uključujući proceduralno, objektno-orijentirano i funkcionalno programiranje.

3.4 JSON

JSON (JavaScript Object Notation) je lagani format za razmjenu podataka koji je lako čitljiv za ljude i jednostavan za parsiranje i generiranje od strane računala. Koristi se za prijenos podataka između poslužitelja i web aplikacija kao tekstualni format. U Blocklyju, JSON se koristi za pohranu konfiguracija, struktura blokova i drugih podataka koji se dijele ili pohranjuju. JSON omogućuje učinkovitu serijalizaciju podataka koja olakšava njihovu razmjenu i pohranu.

3.5 Blockly

Blockly je okruženje otvorenog koda za vizualno programiranje koja omogućuje korisnicima kreiranje programa povlačenjem i ispuštanjem blokova. Izgrađena je korištenjem HTML-a, CSS-a i JavaScript-a. HTML se koristi za strukturiranje korisničkog sučelja, dok CSS omogućuje stiliziranje blokova i drugih elemenata. JavaScript upravlja logikom aplikacije, manipulacijom DOM-om i interaktivnošću blokova. Blockly omogućuje jednostavno učenje programiranja kroz grafičko sučelje koje automatski generira kod u različitim programskim jezicima kao što su JavaScript, Python, PHP, Lua i Dart. U obrazovnim okruženjima koristi se za poučavanje osnovnih programskih koncepata, dok u profesionalnim aplikacijama omogućuje brzi razvoj složenih sustava bez potrebe za detaljnim poznavanjem sintakse programskih jezika.

3.6 Python

Python je programski jezik koji se često koristi za pisanje skripti, automatizaciju procesa i razvoj poslužiteljske strane web aplikacija zbog svoje čitljivosti i jednostavnosti. U kontekstu ovog projekta, Python se koristi zajedno s Flaskom i OS bibliotekom. Integracija Pythona, Flaska i OS biblioteke omogućuje učinkovitu obradu podataka i interakciju s operacijskim sustavom, čime se osigurava dinamičko prilagođavanje i upravljanje različitim komponentama projekta.

3.6.1 Flask biblioteka

Flask je web framework za Python koji omogućuje jednostavno stvaranje web aplikacija i API-ja [23]. Flask omogućuje brzi razvoj aplikacija bez potrebe za složenom konfiguracijom ili opsežnim postavkama. Njegova fleksibilnost i modularnost omogućuju programerima da dodaju potrebne komponente i proširenja prema zahtjevima projekta. U ovom projektu Flask se koristi za kreiranje RESTful API-ja koji povezuje korisničko sučelje s backendom, omogućujući pokretanje skripti, provjeru priključenih uređaja te prijenos koda na mikroupravljače. Flask-CORS modul osigurava siguran prijenos podataka između različitih domena, dok Flask u kombinaciji s Python subprocess modulom omogućuje pokretanje vanjskih skripti, čime se automatizira proces programiranja mikroupravljača izravno putem web aplikacije.

3.6.2 OS biblioteka

OS biblioteka u Pythonu pruža funkcionalnosti za interakciju s operativnim sustavom. Koristi se za izvršavanje sistemskih naredbi, rad s datotekama i direktorijima te pristupanje informacijama o sustavu. U ovom projektu, OS biblioteka se koristi za prikupljanje informacija o USB uređajima spojenim na računalo i serijskim portovima, omogućujući skripti da prepozna i kategorizira povezane uređaje.

3.7 Mocha

Mocha je popularni JavaScript okvir za testiranje koji omogućava jednostavno pisanje i izvršavanje testova za web aplikacije. Pruža fleksibilno i intuitivno sučelje za definiranje testova, podržavajući različite stilove testiranja poput BDD (Behavior-Driven Development) i TDD (Test-Driven Development). Mocha omogućava asinkrono testiranje, čime se osigurava da se testovi mogu izvršavati u stvarnom vremenu bez blokiranja aplikacije. Uz podršku za različite reportere, poput *Mochawesome*, Mocha omogućava generiranje detaljnih izvještaja o testiranju, uključujući pregledne web izvještaje [24]

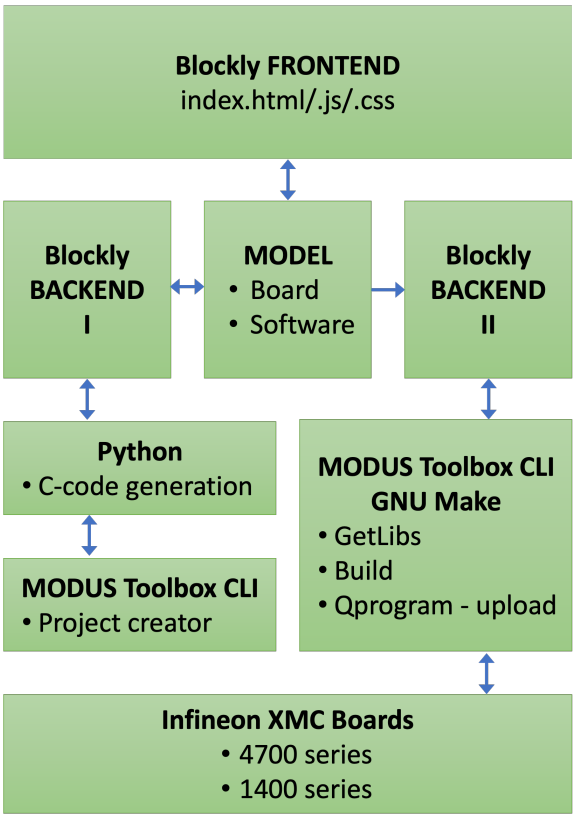
3.8 Chai

Chai je biblioteka za JavaScript koja se koristi zajedno s testnim okvirima poput Mocha-e za pisanje izražajnih i čitljivih testova. Chai podržava različite stilove evaluacije, uključujući "should", "expect" i "assert", što omogućuje fleksibilnost u pisanju testova prema prefe-

rencijama programera [25]. U kontekstu ovog rada, Chai se koristi za provjeru ispravnosti rezultata unutar Mocha testova, osiguravajući da blokovi ispravno generiraju kod i da su sve funkcionalnosti unutar Blockly sučelja ispravno implementirane. Chai dodatno olakšava čitanje i održavanje testova, omogućujući precizne i specifične asertacije koje doprinose pouzdanosti cjelokupnog testnog okvira.

4 IMPLEMENTACIJA BLOKOVSKOG SUČELJA

Sučelje za programiranje mikroupravljača temelji se na Blocklyju. Blockly je okruženje koje je razvio Google za izradu pristupačnih programskih jezika temeljenih na blokovima [3]. Blockly je popularan prvenstveno zbog toga što je otvorenog koda, a samim time je i vrlo proširiv pa je tako baza za mnoge poznate platforme, kao što su Scratch, Micro:bit, Code.com i drugi. U ovom poglavlju opisana je prilagodba Blocklyja te izrada vlastitih blokova i prevoditelja, a s naglaskom na specifičnosti ciljane platforme (XMC serije mikroupravljača). Nakon toga opisan je i pozadinski dio, zadužen za komunikaciju između blokovskog sučelja i mikroupravljača. Arhitekturalni dizajn cijelog sustava prikazan je slikom 4.1.



Slika 4.1. Arhitektura osmišljenog blokovskog sučelja [26]

Sustav se sastoji od korisničkog sučelja razvijenog na Blocklyju te dvodjelnog pozadinske programske podrške: jedan dio zadužen je za generiranje C koda pomoću vlastitog prevoditelja te stvaranje i popunjavanje projektnih direktorija kodom, dok je drugi dio zadužen za komunikaciju s mikroupravljačem (prepoznavanje uređaja, izgradnja projekta te programiranje uređaja), a s Infineonovim *ModusToolbox* razvojnim okruženjem kao posrednikom. Struktura datoteka projekta dana je dijagramom na slici 4.2.. Projektna struktura je or-

ganizirana kako bi omogućila jasno razgraničenje funkcionalnosti i olakšala održavanje i proširivanje koda. Na vrhu hijerarhije nalazi se direktorij *src*, koji sadrži sve glavne komponente projekta. Unutar njega, datoteke kao što su *app.py*, *index.js*, *index.html*, *index.css*, s*erialization.js* i *toolbox.js* odgovorne su za sučelje i osnovne funkcionalnosti aplikacije.
Direktorij *Blocks* sadrži definicije Blockly blokova u *blocks.js* datoteci i vlastiti C prevoditelj u *Clang.js* datoteci. Blokovi definirani u *blocks.js* predstavljaju osnovne elemente vizualnog sučelja korisnika.
Direktorij *Toolboxes* uključuje konfiguracijske datoteke *xmc_14.js* i x*mc_47.js* koje definiraju specifične postavke i blokove koji se koriste za odgovarajući mikroupravljač.
Direktorij *PythonScripts* sadrži skripte koje omogućuju interakciju s fizičkim uređajima i automatizaciju različitih zadataka, kao što su provjera dostupnih uređaja u *CheckDevices.py* skripti, izgradnja projekta u *Build.py* skripti, preslagivanje sadržaja datoteka i datoteka zaglavlja u *ArrangeFiles.py* skripti i programiranje uređaja u *Upload.py* skripti.
Konačno, direktorij *MochaTests* rezerviran je za testiranje, koristeći Mocha i Chai okruženja za osiguranje kvalitete i ispravnosti blokova kroz testove.

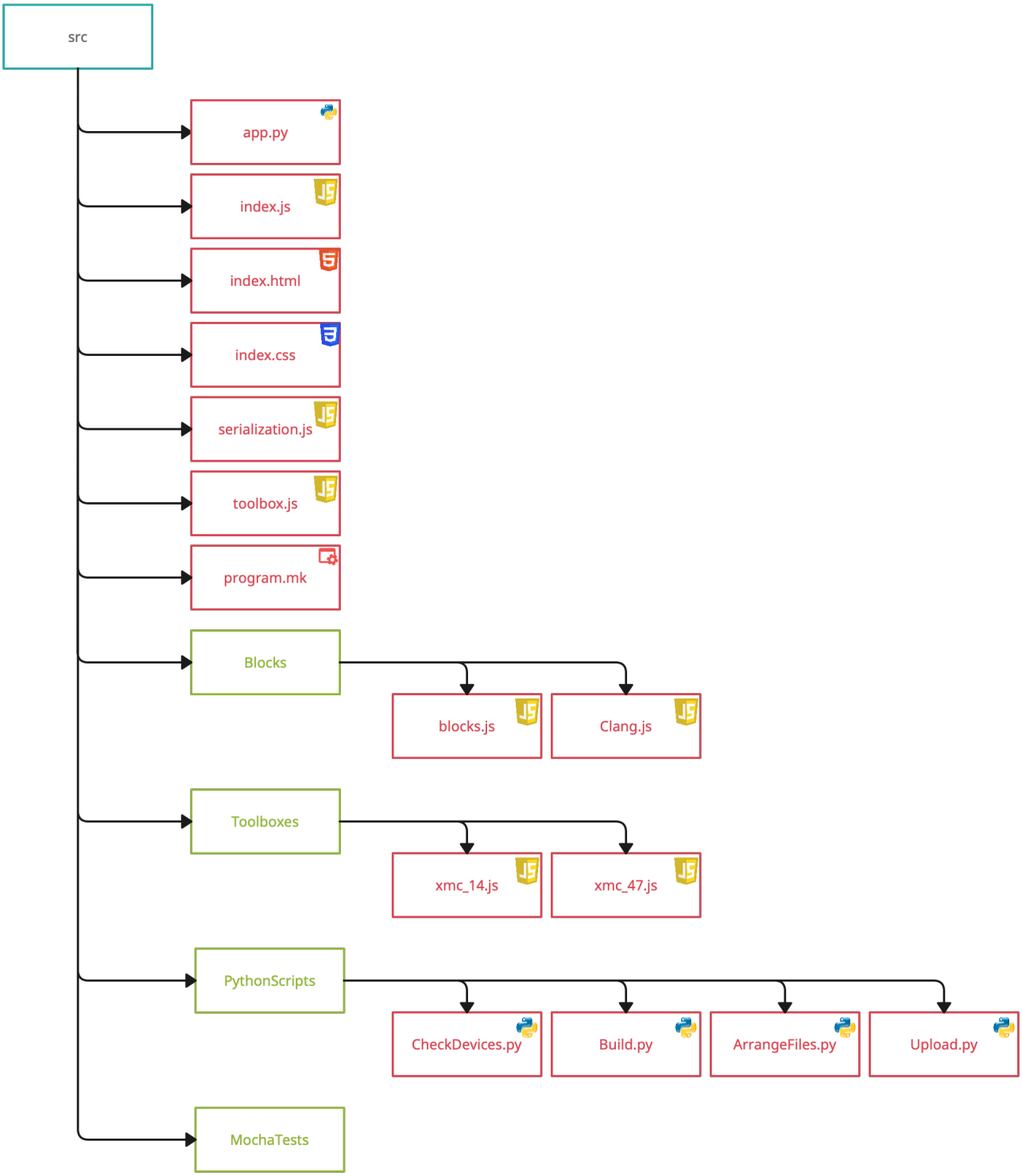
Nadalje, grafičko sučelje osmišljenog sustava vidljivo je na slici 4.3.. Sastoji se od 4 dijela:

A **Prikaz koda**. Nakon što je korisnik povukao blok na za to predviđeno mjesto, kod koji generira prevoditelj prikazuje se u dijelu A (osim koda konfiguracijskih blokova, što je objašnjeno u nastavku poglavlja)

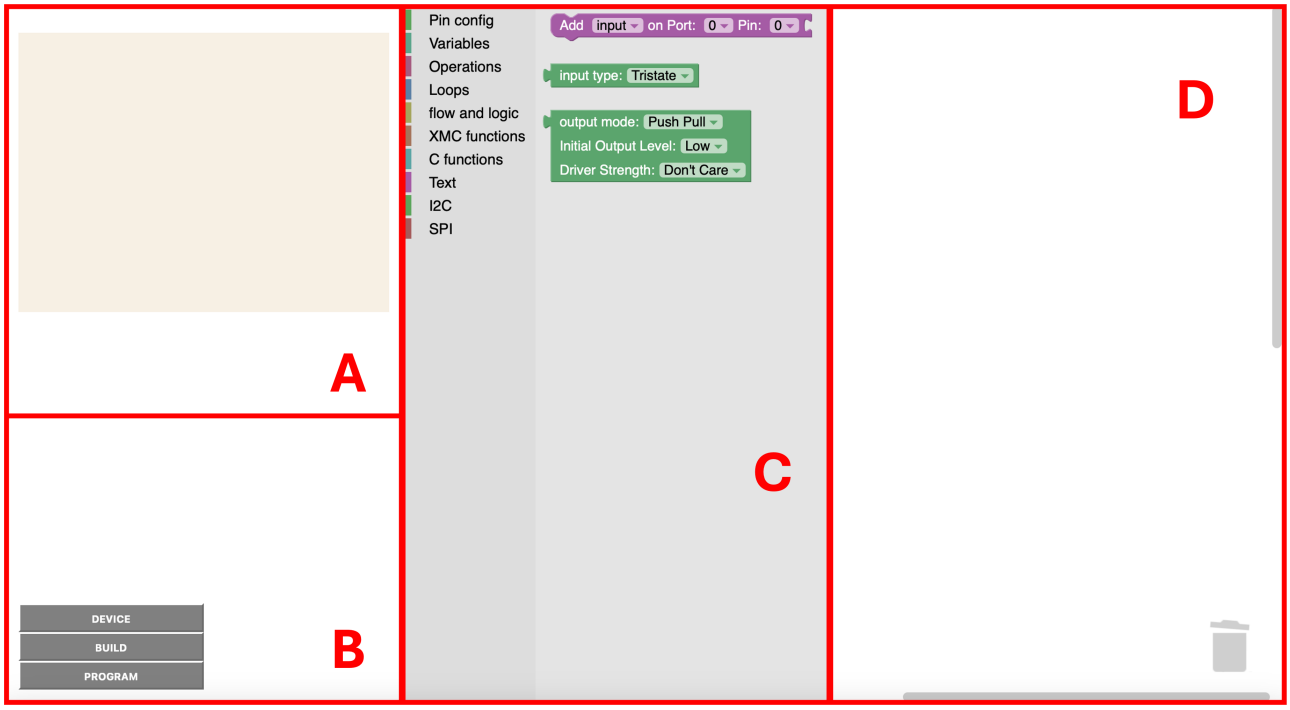
B **Dodatne funkcionalnosti**. Ovaj dio daje korisniku mogućnosti odabira uređaja, izgradnju projekta i programiranje uređaja. Prilikom pristiska gumba, u pozadini se pokreće odgovarajuća *Python* skripta.

C **Traka odabira blokova**. Dostupni blokovi ovise o odabranom uređaju. Korisnik najprije bira kategoriju blokova, a nakon odabira kategorije korisniku se ponude odgovarajući blokovi.

D **Radna površina**. Na radnu površinu korisnik vuče blokove, slaže ih i odabire opcije koje određeni blokovi nude. Prevoditelj blokova pokreće se pri svakoj promjeni radne površine te generira odgovarajući kod.



Slika 4.2. Struktura projekta



Slika 4.3. Izgled osmišljenog blokovskog sučelja

4.1 Razvoj vlastitih blokova

Prilikom stvaranja vokabulara blokovskog programskog jezika, tri su moguća pristupa [6]:

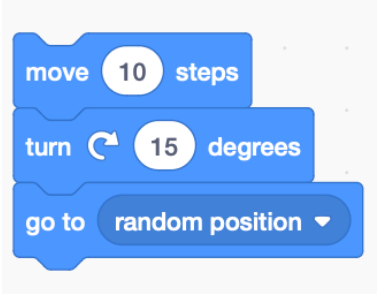
- **Ikone**: Ikone se oslanjaju na slike kao primarni vokabular blokova. Slikama se nastoji opisati funkcija pojedinog bloka. Ovakav pristup je dobar za izrazito mlade korisnike ili za postizanje sustava neovisnog o jeziku korisnika. Ikonografija omogućava intuitivno razumijevanje i jednostavno prepoznavanje funkcionalnosti bez potrebe za poznavanjem jezika. Primjer korištenja ikona kao vokabulara vidljiv je na slici 4.4.a.

- **Prirodni jezik**:Prirodni jezik deskriptivno opisuje funkciju pojedinog bloka te je specifičan jeziku korisnika. Često se koristi kombinacija prirodnog jezika i ikona, kao što je slučaj s platformom Scratch. [8]. Korištenje prirodnog jezika omogućava jasnije i detaljnije objašnjenje funkcionalnosti bloka, što može biti korisno za korisnike koji su novi u programiranju. Ovaj pristup olakšava razumijevanje složenih koncepata korištenjem poznatih izraza i metafora iz svakodnevnog života. Primjer korištenja prirodnog jezika kao vokabulara vidljiv je na slici 4.4.b

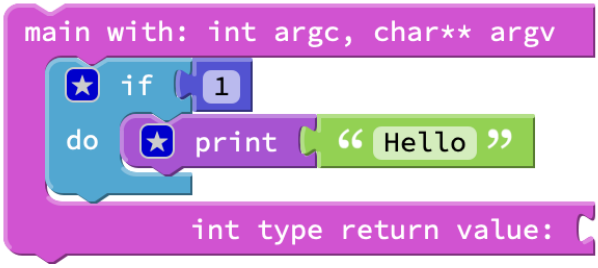
- **Računalni jezik**: Definiranje vokabulara blokova korištenjem računalnog jezika vrlo je slično samom programskom jeziku te je primjerenije skupini ljudi koja ima predz-

nanje o programiranju. Ovaj pristup omogućava veću preciznost i kontrolu nad programom, no može biti manje intuitivan za početnike. Primjer je Blockly, koji koristi sintaksu sličnu JavaScriptu za definiranje blokova, što pomaže korisnicima da lakše pređu na pisanje koda nakon što savladaju blokovsko programiranje. Primjer korištenja prirodnog jezika kao vokabulara vidljiv je na slici 4.4.c

(a) Primjer blokova s ikonama (Code-org [15]).



(b) Primjer blokova s prirodnim jezikom (Scratch [8]).

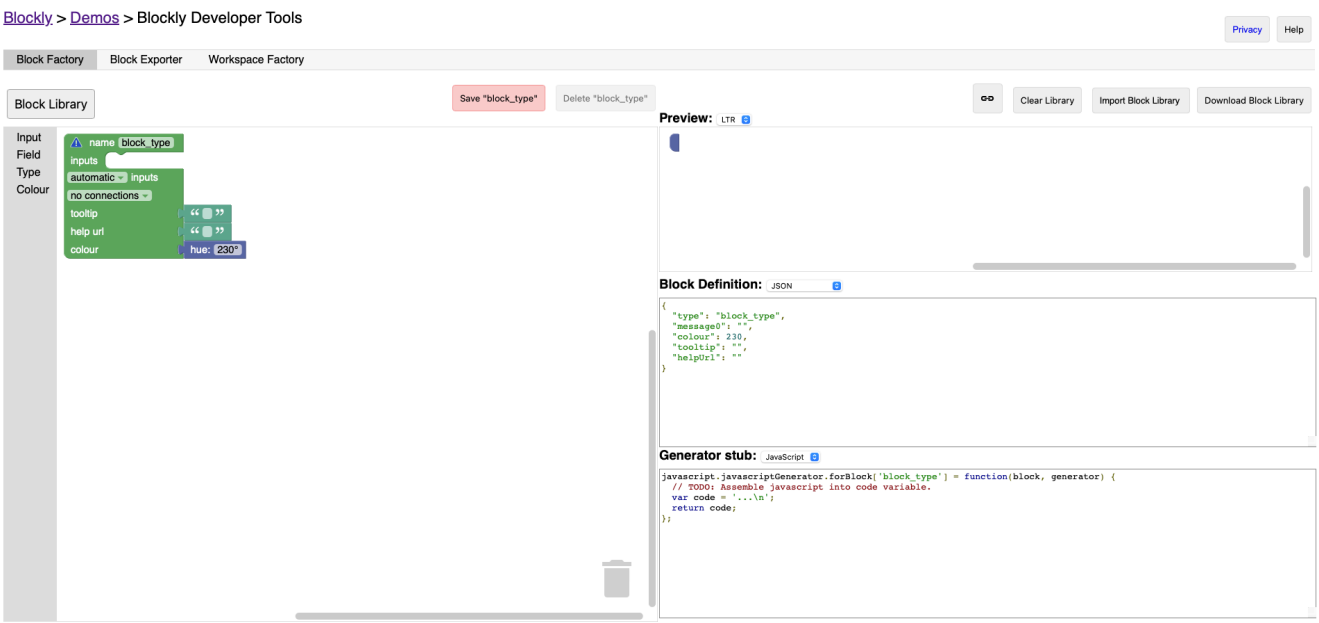


(c) Primjer blokova s računalnim jezikom (Cakecore [27]).

Slika 4.4. Različiti pristupi vokabularu blokovskog programiranja: prirodni jezik, ikone i računalni jezik.

Kombinacija različitih pristupa može pružiti najbolje od oba svijeta, omogućavajući korisnicima da se postupno uvode u složenije koncepte. Na primjer, platforme kao što su Scratch često kombiniraju prirodni jezik s ikonama i računalnim jezikom kako bi zadovoljile različite razine znanja korisnika. Korištenje animacija i interaktivnih elemenata unutar blokova može dodatno pojasniti njihovu funkciju. Na primjer, animirani blokovi mogu pokazati kako se podaci kreću kroz različite dijelove programa, čime se korisnicima pruža bolji uvid u dinamiku programiranja. Stvaranje blokova sastoji se od dva osnovna koraka [3]:

1. definiranje izgleda bloka

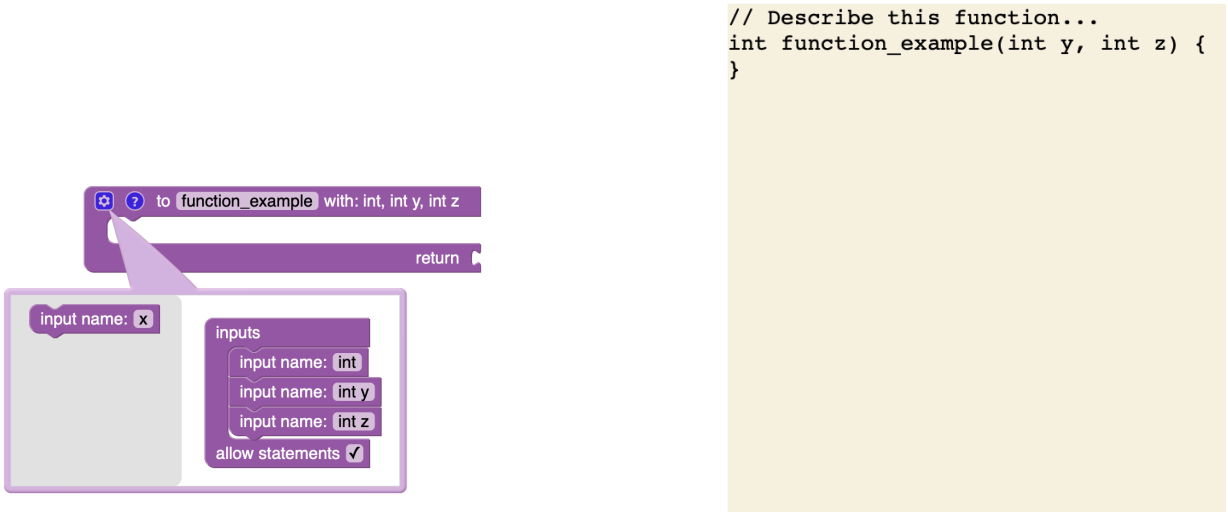


Slika 4.5. Izgled *Blockly Developer Tools* sučelja

2. stvaranje prevoditelja bloka u programski kod

Blockly nudi nekoliko načina za definiranje izgleda vlastitih blokova: korištenjem *Blockly Developer Tools* sučelja i pisanjem vlastitog koda u obliku JSON objekta ili korištenjem JavaScript API-ja. Najbrži način, koji je ujedno dovoljan za većinu jednostavnijih blokova koji ne zahtjevaju dodatnu, pozadinsku logiku je *Blockly Developer Tools* sučelje (Slika 4.5.). Prednost korištenja ovog sučelja je interaktivnost te automatsko stvaranje definicije bloka u JSON ili JavaScript obliku te automatsko stvaranje prevoditelja za blok koji se dizajnira. No, *Blockly* predefinirano daje prevoditelje za tek nekoliko programskih jezika (kako C nije u toj skupini, za potrebe ovog sustava potrebno je napraviti i vlastiti prevoditelj). Nedostatak *Blockly Developer Tools* sučelja je što ne nudi mogućnost stvaranja kompleksnijih blokova za koje je unutar definicije potrebno dati i logiku ponašanja bloka. Navedeno je moguće jedino ručnim pisanjem koda korištenjem JavaScript API-ja. Primjerice, blokove koji koriste funkcije proširenja (mutatori) i blokove koji ovise o događajima moguće je definirati isključivo korištenjem JavaScript API-ja. Primjer mutatora i generiranog koda dan je slikom 4.6.. Funkcije proširenja pokreću se kada je blok stvoren te predstavljaju korisnički definirano ponašanje. Konkretno, funkcija proširenja može izmijeniti izgled bloka naknadnim postavljanjem broja argumenata u bloku kada se registrira određeni događaj. Za potrebe programiranja XMC serije mikrokontrolera napravljene su dvije osnovne vrste blokova:

- konfiguracijski blokovi



(a) Mutator (b) Generirani kod

Slika 4.6. Mutator (lijevo) i za njega generirani kod (desno)

- standardni blokovi

Konfiguracijski blokovi zamjenjuju funkcionalnosti *Device Configurator* alata *ModusToolbox* razvojnog okruženja. Konfiguracijski blokovi primarno služe za postavljanje aktivnih pinova te prilagodbu istih (vrsta ulaza i izlaza). Uz navedeno moguće je i postavljanje prekida i vremenskih brojača. Postavljanjem konfiguracijskih blokova na radnu površinu aplikacije ne prikazuje se kod, već se isti sprema u memoriju te pri pokretanju procesa izgradnje projekta sprema u konfiguracijske datoteke popunjavanjem konfiguracijskih predložaka. Standardni blokovi služe kao zamjena za C kod logike programa te se prikazuje u za to predviđeno mjesto u aplikaciji. Prilikom pokretanja *build* procesa taj kod sprema se u *main.c* datoteku projekta.

Osim podjele na konfiguracijske i standardne blokove (što je podjela napravljena za potrebe ovog rada) prema Blockly dokumentaciji [3], blokovi mogu biti statični i dinamični. Kod statičnih blokova izgled bloka, ali i vrijednosti se ne mijenjaju. S druge strane, kod dinamičnih blokova moguće je promijeniti formu bloka (tada je riječ o mutatorima) ili je moguće promijeniti vrijednosti unutar bloka. Primjerice, za potrebe ovog rada napravljen je poseban dinamični blok za odabir i postavljanje pinova. Kako XMC1400 i XMC4700 nemaju jednak broj i raspored pinova, bitno je da se blok mijenja ovisno o:

a) priključenom uređaju

b) prethodno odabranoj opciji unutar bloka

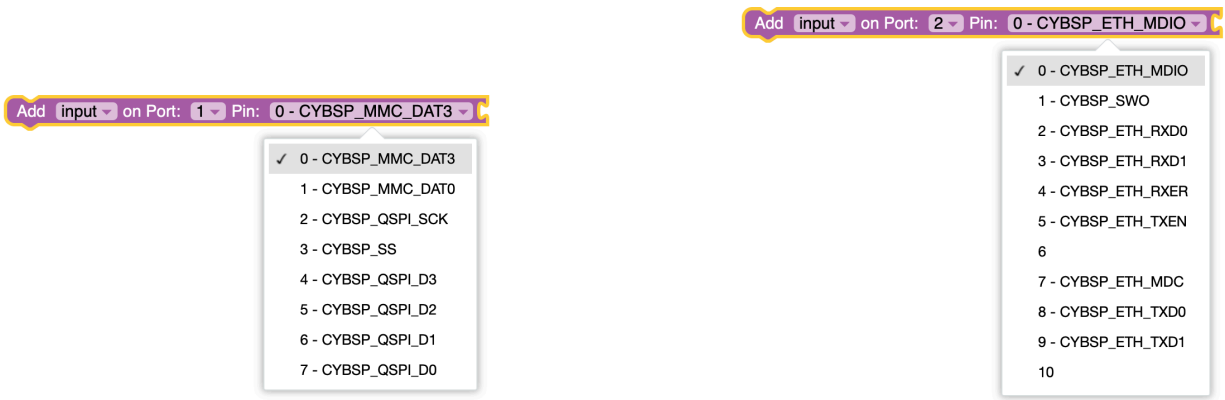
Uvjet b) podrazumijeva da se komponenta bloka mijenja svaki puta kada se promjeni i komponenta o kojoj ona zavisi. Konkretno, promjena *port* opcije *pin* polja. Navedeno se postiže korištenjem metoda *setOnChange* i *updatePins*, kako je prikazano programskim kodom 1.

Programski kod 1: Kod za definiciju bloka sa zavisnim padajućim izbornikom

```
1  //...
2  Blockly.Blocks['pins14'] = {
3    init: function() {
4      var thisBlock = this;
5      var optionsForName = default_1;
6      var dropdownName = new Blockly.FieldDropdown(optionsForName);
7
8      var input = this.appendValueInput("TYPE")
9        .appendField("Add ")
10       .appendField(new Blockly.FieldDropdown(input_options), "DIR")
11       .appendField("on Port: ")
12       .appendField(new Blockly.FieldDropdown(default_2, function(option) {
13         thisBlock.updatePins(option);
14       }), "NAME1")
15       .appendField("Pin: ")
16       .appendField(dropdownName, "NAME");
17
18     this.setPreviousStatement(true, null);
19     this.setNextStatement(true, null);
20     this.setColour(300);
21     this.setTooltip("config");
22     this.setHelpUrl("");
23
24     this.updatePins = function(port) {
25       var optionsForName;
26       switch (port) {
27         case "0":
28           optionsForName = options_0;
29           break;
30         case "1":
31           optionsForName = options_1;
32           break;
33         case "2":
```

```
34            optionsForName = options_2;
35            break;
36          case "3":
37            optionsForName = options_3;
38            break;
39          default:
40            optionsForName = options_default;
41            break;
42        }
43        dropdownName.menuGenerator_ = optionsForName;
44        dropdownName.setValue(optionsForName[0][1]);
45        // Set to first option in the new list
46      };
47
48      this.setOnChange(function(event) {
49        if (event.type == Blockly.Events.BLOCK_CHANGE
50        && event.blockId == this.id && event.name == 'NAME1') {
51          this.updatePins(event.newValue);
52        }
53      });
54    },
55  };
```

Zavisna ovisnost padajućeg izbornika iz programskog koda 1 vidljiva je i u blokovskom prikazu na slici 4.7..



(a) opcije pinova za port 1 (b) Opcije pinova za port 2

Slika 4.7. Dinamični blok sa zavisnim padajućim izbornicima

4.2 Razvoj vlastitog prevoditelja

Proces generiranja koda pretvara blokove u programski kod koji se može izvesti. Prvi dio generiranja koda odnosi se na jeziku specifično generiranje (prevoditelj jezika). Konkretno, odabire se prevoditelj zadužen za općenito formatiranje koda koje ne ovisi o bloku. Drugi dio prevođenja specifičan je bloku te se mora definirati za svaki blok koji se koristi (prevoditelj bloka). *Blockly* trenutno podržava prevoditelje za sljedeće jezike [3]:

- JavaScript ES5
- Python 3
- Lua 5.1
- Dart 2
- PHP 7

Kako C jezik nije podržan, potrebno je napraviti vlastiti prevoditelj: najprije prevoditelj jezika koji, nakon inicijalizacije, iterira po blokovima na radnoj površini te za svaki blok poziva odgovrajaući prevoditelj bloka. Prevoditelja bloka na primjeru dinamičkog bloka sa zavisnim padajućim izbornicima (Slike 4.7.) dan je programskim kodom 2.

Programski kod 2: Primjer prevoditelja bloka

```
1 clangGenerator.forBlock['pins47'] = function(block, generator) {
2   var port = block.getFieldValue('NAME1');
3   var pin = block.getFieldValue('NAME');
4   var direction = block.getFieldValue("DIR");
5   var mode = generator.valueToCode(block, 'variable', Order.ATOMIC);
6
7   var code = GetCustomConfigCode(port,pin,direction,mode);
8   return code[0] + code[1] + code[2];
9 };
```

Uvlačenje koda, redoslijed operatora i slično prilikom generiranja koda odrađuje generator jezika. Pretvorba blokova na radnoj površini u programski kod omogućava *workspaceToCode* metoda. Razdvajanje blokova, a time i koda na konfiguracijski i standardni nije predviđeno originalnom metodom pa se ista za potrebe ovog projekta morala ponovno definirati i prilagoditi.

Programski kod 3: Vlastita *workspaceToCode* metoda

```
clangGenerator.workspaceToCode = function(workspace = Blockly.Workspace){
  if (!workspace) {
    //podrska za blockly aplikaciju s švie radnih špovrina
    console.warn(
      'No workspace specified in workspaceToCode call.  Guessing.',
    );
    workspace = common.getMainWorkspace();
  }
  var code = '';
  var configCode = '';

  this.init(workspace);
  const unfilteredBlocks = workspace.getTopBlocks(true);

  const blocks =
    unfilteredBlocks.filter((block) => block.getTooltip() !== 'config');
  const configBlocks =
    unfilteredBlocks.filter((block) => block.getTooltip() === 'config');

  code = getCodeFromBlocks(blocks);
  configCode = getCodeFromBlocks(configBlocks);

  return [code,configCode];

}
```

Vidi se (programski kod 3) da izmijenjena metoda najprije dohvaća sve blokove na radnoj površini, razdvaja ih u dvije skupine te poziva *getCodeFromBlocks* metodu nad svakom skupinom. Potonja metoda dohvaća kod svakog bloka te linije koda slaže ovisno o pravilima definiranim vlastitim prevoditeljem.

4.3 Raspoznavanje priključenog uređaja

Proces raspoznavanja priključenih uređaja temelji se na korištenju skripti koje automatski prepoznaju i kategoriziraju uređaje povezane na računalo. Ovaj proces je ključan za dinamičko prilagođavanje dostupnih blokova za programiranje, čime se omogućuje specifična funkcionalnost ovisno o vrsti spojenog razvojnog sustava. Navedeno je i nužno, s obzirom da

dva razvojna sustava korištena u ovom radu (XMC 1400 i XMC4700) nemaju jednak broj i raspored pinova.

Prvi korak u procesu raspoznavanja uređaja uključuje prikupljanje informacija o trenutno priključenim USB uređajima i serijskim portovima. To se postiže korištenjem sistemskih naredbi iz programskog koda 4, čiji se rezultati pohranjuju u odgovarajuće datoteke (*devicesInfo.json* i *portsInfo.txt*). Ove naredbe omogućuju detaljan uvid u povezane uređaje i njihove karakteristike.

Programski kod 4: Prikupljanje podataka o USB uređajima i serijskim portovima

```
1 os.system(f'system_profiler SPUSBDataType -json > ./devicesInfo.json')
2 os.system(f'ls -l /dev/cu.* > ./portsInfo.json')
```

Nakon prikupljanja podataka, skripta učitava sadržaj JSON datoteke koja sadrži informacije o USB uređajima. Podaci se zatim analiziraju kako bi se izdvojila imena uređaja. Rekurzivna funkcija (Programski kod 5) pretražuje ugniježđene strukture podataka u JSON datoteci, tražeći specifične ključeve i vrijednosti koje predstavljaju nazive uređaja. Ovisno o serijskom broju prepoznatog uređaja, određuje se vrsta uređaja kao "XMC1400", "XMC4700" ili "drugi" uređaj. Ovisno o prepoznatom uređaju, glavni dio *Blockly* koda odabire listu blokova koji su korisniku dostupni. Definicije blokova koji se koriste za specifični uređaj nalaze se u *Toolboxes* direktoriju (slika 4.2.).

Programski kod 5: Rekurzivna funkcija za ekstrakciju imena uređaja iz JSON podataka

```
1  def extract_names(obj, parent_key=None):
2      names = []
3      if isinstance(obj, list):
4          for item in obj:
5              names.extend(extract_names(item, parent_key))
6      elif isinstance(obj, dict):
7          serial_num = obj.get("serial_num")
8          for key, value in obj.items():
9              if key == "_name" and parent_key == "_items":
10                 if(("Adapter" not in value) and ("Hub" not in value)):
11                     if(serial_num == "000591195364"):
12                         value="XMC1400"
13                         names.append(value)
14                     elif(serial_num == "000591199982"):
15                         value="XMC4700"
```

```
16                  names.append(value)
17              else:
18                  value="other"
19                  names.append(value)
20          names.extend(extract_names(value, key))
21      return names
```

4.4 Izgradnja projekta i programiranje mikroupravljača

Proces programiranja mikroupravljača predstavlja ključan korak u svakom projektu koji uključuje ugradbeno programiranje. Nakon što su svi blokovi postavljeni i konfigurirani unutar blokovskog sučelja, potrebno je generirati i pripremiti kod za ciljni mikroupravljač. Ovaj proces uključuje nekoliko faza koje se odvijaju pomoću skripte, ali zahtijevaju detaljno razumijevanje kako bi se osigurala ispravnost i učinkovitost krajnjeg sustava.
Nakon generiranja koda, slijedi faza izgradnje projekta. Ovaj korak koristi *GNU make* alat za kompilaciju koda, stvaranje izvršnih datoteka i pripremu projekta za programiranje mikroupravljača. Važno je napomenuti da se tijekom ove faze provjerava ispravnost koda, te se korisniku prijavljuju eventualne pogreške koje bi mogle ometati pravilno funkcioniranje sustava. Za izgradnju projekta zadužena je *build.py* skripta (Slika 4.2.). Klikom na *BUILD* dugme pokreće se programski kod 6 te kreće izgradnja projekta, kako je prikazano na slici 4.3.. Unutar skripte izgradnja se odvija u sljedećim koracima:

1. Pokretanje *arrangeFiles.py* skripte i stvarnje C datoteka (main.c i konfiguracijske datoteke zaglavlja) iz *app.py* glavne datoteke poslužiteljske strane.

2. kreiranje direktorija projekta lokalno na računalu

3. stvaranje praznog projekta *project-creator* CLI naredbom

4. kopiranje glavne C datoteke i konfiguracijskih datoteka u stvoreni projekt

5. ažuriranje potrebnih biblioteka *make getlibs* naredbom

6. kopiranje ispravne makefile datoteke u projekt (navedeno je potrebno zbog pogreške pronađenoj u ModusToolbox 3.1 verziji tijekom pisanja ovog rada)

7. pokretanje *make build* naredbe

Programski kod 6: Dodavanje funkcionalnosti gumba za izgradnju projekta i spremanja koda.

```
buildButton.addEventListener('click', function() {

    const [code,configurationCode] = clangGenerator.workspaceToCode(ws);

    fetch(`http://127.0.0.1:8000/runPythonScript`, {
        method: 'POST',
        headers: {
          'Content-type': 'application/json',
        },
        body: JSON.stringify({
        code: code,
        config_code: configurationCode,
        device: selected_device
        }),
    })
    .then(response => response.json())
    .then(data => {
        console.log(data); // Log the response from the server
    })
    .catch(error => {
        console.error('Error:', error);
    });

    //console.log(code);
    console.log(configurationCode);
});
```

Nakon izgradnje projekta, klikom na "PROGRAM" dugme pokreće se skripta za programiranje mikroupravljača. Najprije se provjerava izlaz "build.py" skripte, tj provjerava se je li izgradnja projekta uopće izvedena. Ukoliko je izvedena i povratna vrijednost je 0, pokreće se *make qprogram* naredba. U suprotnom se projekt prvo mora izgraditi pa se pokreće *make program* naredba.

5 TESTIRANJE BLOKOVSKOG SUČELJA

Nakon završetka razvoja blokovskog sučelja, ključno je provesti testiranje kako bi se osigurala ispravnost i pouzdanost sustava. U ovom poglavlju obrađuju se dvije temeljne metode testiranja: unit testovi i end-to-end testiranje cijelog sustava.

Unit testovi predstavljaju osnovu svakog postupka testiranja softvera, jer omogućuju provjeru ispravnosti pojedinačnih dijelova koda u izolaciji. Prema istraživanjima, korištenje unit testova smanjuje broj grešaka u kasnijim fazama razvoja te omogućuje brže otkrivanje problema [28][29]. U okviru ovog rada, unit testovi su primijenjeni na ključne komponente sustava, uključujući generiranje koda iz blokova, konfiguracijske blokove i Python skripte za komunikaciju s mikroupravljačem.

Nakon što su pojedinačne komponente testirane, slijedi funkcionalno testiranje cijelog sustava (end-to-end testovi) kroz dva primjera koji prikazuju stvarne primjene sučelja. Ovaj pristup omogućuje procjenu sustava u uvjetima koji simuliraju stvarno korištenje, čime se potvrđuje da su sve komponente ispravno integrirane i da sustav kao cjelina funkcionira prema očekivanjima [30].

Cilj ovih testova je osigurati da sustav ispunjava sve funkcionalne zahtjeve, da je stabilan, te da korisnicima omogućuje jednostavno i efikasno programiranje mikroupravljača putem blokovskog sučelja.

5.1 Unit testovi

Blockly koristi unit testove kao ključni alat za održavanje kvalitete koda i osiguravanje ispravnosti funkcionalnosti unutar svoje jezgre. Na službenim stranicama za razvojne programere [3], Google daje smjernice o tome kako napisati i organizirati unit testove. Ovi testovi koriste Mocha okruženje, u kombinaciji s Chai okruženjem, za provjeru očekivanih rezultata unutar Blockly sustava. Testiranje se obavlja automatski, čime se omogućuje brzo otkrivanje grešaka i osigurava stabilnost koda kroz različite verzije.

Blocklyjeva dokumentacija preporučuje pisanje testova za svaki novi komad funkcionalnosti koji se dodaje u sustav, kako bi se osiguralo da promjene ne uvode regresije ili neželjene greške. Testovi se mogu izvršavati lokalno na računalu programera ili putem CI/CD sustava za kontinuiranu integraciju, što dodatno pomaže u održavanju visoke kvalitete projekta.

Unit testovi za blokove u Blocklyju obično uključuju provjeru ispravnosti generiranja koda (a

time zajednički provjeravaju funkcionalnost blokova, ali i prevoditelja) i reakcija na različite promjene unutar radnog prostora (primjerice, pri testiranju dinamičkih blokova). Programeri pišu testove koji simuliraju stvarne situacije korištenja blokova, provjeravajući kako se blokovi ponašaju kada su povezani ili izmijenjeni.

Jedan od ključnih aspekata pisanja unit testova za blokove jest osiguranje da svi dijelovi bloka funkcioniraju u skladu s očekivanjima kada su povezani s drugim blokovima, te da ne uzrokuju regresije u ostatku sustava. Testovi također provjeravaju kako blokovi reagiraju na različite ulaze te ažuriraju li se ispravno kada dođe do promjena u radnom prostoru.

Primjer Unit testa za blok dan je programskim kodom 7. Riječ je o unit testu za testiranje dinamičkog bloka sa zavisnim padajućim izbornicima danog programskim kodom 1 i slikom 4.7.. Za navedeni blok ispituju se dva slučaja:

1. Prvi test provjerava ispravnost početnih vrijednosti polja unutar bloka "pins14". Konkretno, provjerava se:
   - je li vrijednost polja smjera inicijalno ispravno postavljena na 'INPUT'.
   - je li vrijednost polja port ispravno postavljena na '0'.
   - je li vrijednost polja pin ispravno postavljena na '0'.
2. Drugi test provjerava ažuriraju li se opcije u padajućem izborniku za "pin" ispravno kada se promijeni odabrani "port". Test mijenja port na vrijednost '1' i zatim poziva metodu updatePins, koja bi trebala ažurirati opcije u padajućem izborniku pin u skladu s novim odabirom porta. Nakon ažuriranja, test provjerava sadrži li padajući izbornik očekivane opcije za odabir pina te je li broj opcija točno 9, što odgovara konfiguraciji za port '1'.

Programski kod 7: Primjer Unit testa.

```
1  import { assert } from 'chai';
2  import Blockly from 'blockly';
3
4  describe('pins14 Block Tests', function() {
5    let workspace;
6
7    beforeEach(function() {
8      workspace = new Blockly.Workspace();
```

```
9    });
10
11   afterEach(function() {
12     workspace.dispose();
13   });
14
15 // TEST 1
16   it('should have the correct initial values', function() {
17     const block = workspace.newBlock('pins14');
18     assert.equal(block.getFieldValue('DIR'), 'INPUT');
19     assert.equal(block.getFieldValue('NAME1'), '0');
20     assert.equal(block.getFieldValue('NAME'), '0');
21   });
22
23
24 // TEST 2
25   it('should update pin options based on port', function() {
26     const block = workspace.newBlock('pins14');
27     block.setFieldValue('1', 'NAME1');  // Change port to 1
28     block.updatePins('1');  // Call updatePins method
29     assert.equal(block.getField('NAME').menuGenerator_.length, 9);
30     assert.deepEqual(block.getField('NAME').menuGenerator_,
31     [["0", "0"], ["1", "1"],
32     ["2 - CYBSP_DEBUG_UART_TX", "CYBSP_DEBUG_UART_TX"],
33     ["3 - CYBSP_DEBUG_UART_RX", "CYBSP_DEBUG_UART_RX"],
34     ["4", "4"], ["5", "5"], ["6", "6"],
35     ["7", "7"], ["8", "8"]]);
36   });
37
38 });
```

5.2 End-To-End testovi

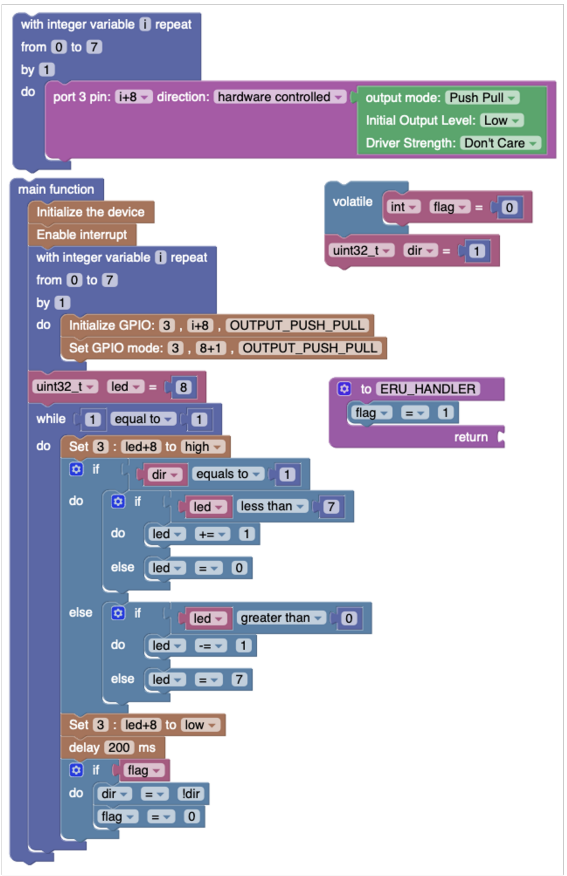
End-to-end testovi ključni su za provjeru funkcionalnosti cjelokupnog sustava, od korisničkog sučelja razvijenog pomoću Blocklyja do komunikacije s mikroupravljačem putem ModusToolbox okruženja. Ovi testovi simuliraju stvarne korisničke scenarije kako bi se osigurala pravilna integracija i rad svih komponenti sustava.

5.2.1 Trčeće svjetlo

Prvi testni scenarij uključuje programiranje jednostavne aplikacije za kontrolu LE dioda na mikroupravljaču. Korištenjem blokova, korisnik kreira program koji omogućuje paljenje i gašenje niza LE dioda u određenim vremenskim intervalima. Test započinje s odabirom odgovarajućih blokova za kontrolu GPIO pinova na mikroupravljaču. Nakon generiranja C koda, provjerava se ispravnost sintakse koda i njegova kompatibilnost s mikroupravljačem. Test završava prijenosom koda na mikroupravljač te vizualnom provjerom. Uspješan test potvrđuje da su svi koraci, od kreiranja programa do njegove implementacije na mikroupravljaču, pravilno izvedeni.

5.2.2 Kontroler i RGB LE dioda

Drugi testni scenarij odnosi se na kontrolu intenziteta pojedinih komponenti RGB LED diode pomoću joysticka. Korisnik kreira program koristeći blokove za povezivanje joysticka s odgovarajućim GPIO pinovima mikroupravljača, što omogućuje kontrolu crvene, zelene i plave komponente LED diode. Nakon generiranja C koda, provodi se sintaksna provjera, a zatim se kod prenosi na mikroupravljač. Test završava provjerom ispravnosti rada LED diode u stvarnom vremenu; pomicanjem joysticka mijenjaju se intenziteti boja na RGB LED diodi. Uspješan test potvrđuje sposobnost sustava da ispravno interpretira signale s joysticka i upravlja LED diodom prema očekivanjima.

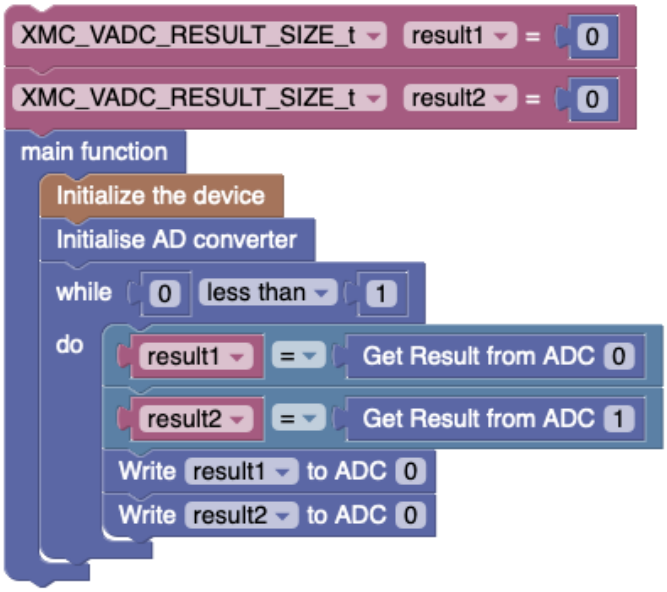


(a) Blokovsko rješenje

```
volatile int flag = 0;
uint32_t dir = 1;
void ERU_HANDLER(void) { flag = 1; }
int main(void) {
        cy_rslt_t result = cybsp_init();
        if (result != CY_RSLT_SUCCESS) { CY_ASSERT(0);}
        NVIC_SetPriority(INTERRUPT_PRIORITY_NODE_ID,
                         INTERRUPT_EVENT_PRIORITY);
        NVIC_EnableIRQ(INTERRUPT_PRIORITY_NODE_ID);
        for(int i = 0; i <=7; i+=1){
                XMC_GPIO_Init(3,i+8,
                        XMC_GPIO_MODE_OUTPUT_PUSH_PULL);
                XMC_GPIO_SetMode(XMC_GPIO_PORT3, 8+1,
                        OUTPUT_PUSH_PULL);
        }
        uint32_t led = 8;
        while (1==1){
        XMC_GPIO_SetOutputLevel(XMC_GPIO_PORT3, led+8,
                                XMC_GPIO_OUTPUT_LEVEL_HIGH );
                if (dir == 1) { if (led < 7) { led += 1; } else { led = 0; }
                } else { if (led > 0) { led -= 1; } else { led = 7; } }
                XMC_GPIO_SetOutputLevel(XMC_GPIO_PORT3, led+8,
                        XMC_GPIO_OUTPUT_LEVEL_LOW );
                XMC_Delay(500);
                if (flag) { dir = !dir; flag = 0; }
        }
        return 0;
}
```

(b) Generirani kod

Slika 5.1. Primjer 1: Trčeće svjetlo



(a) Blokovsko rješenje

```
XMC_VADC_RESULT_SIZE_t result1 = 0;
XMC_VADC_RESULT_SIZE_t result2 = 0;
int main(void){
    cy_rslt_t result = cybsp_init();
    if (result != CY_RSLT_SUCCESS){CY_ASSERT(0);}
        XMC_DAC_CH_Init(XMC_DAC0, dac_0_ch_0_NUM, &dac_0_ch_0_config);
        XMC_DAC_CH_Init(XMC_DAC0, dac_0_ch_1_NUM, &dac_0_ch_1_config);
        XMC_DAC_CH_StartSingleValueMode(XMC_DAC0, dac_0_ch_0_NUM);
        XMC_DAC_CH_StartSingleValueMode(XMC_DAC0, dac_0_ch_1_NUM);
    while (0<1){
        result1 = XMC_VADC_GROUP_GetResult(vadc_0_group_0_HW,0);
        result2 = XMC_VADC_GROUP_GetResult(vadc_0_group_0_HW,1);
        XMC_DAC_CH_Write(XMC_DAC0, dac_0_ch_0_NUM, result1);
        XMC_DAC_CH_Write(XMC_DAC0, dac_0_ch_1_NUM, result2);
    }
    return 0;
}
```

(b) Generirani kod

Slika 5.2. Primjer 2: Kontroler i RGB LED dioda

6 ZAKLJUČAK

U ovom radu predstavljeno je blokovsko sučelje za programiranje Infineon XMC mikroupravljača, temeljeno na Blocklyju. Njegova jednostavnost i intuitivno korisničko sučelje omogućuju manje iskusnim korisnicima da brzo savladaju osnove programiranja mikroupravljača. Ipak, iako je Blockly vrlo moćan i fleksibilan alat, njegova implementacija za XMC mikroupravljače pokazala je određene izazove.

XMC mikroupravljači su složeni i zahtijevaju visoku razinu apstrakcije kako bi njihova funkcionalnost bila predstavljena kroz jednostavne blokove u Blocklyju. Ovaj rad je uspješno postigao osnovnu razinu apstrakcije, no postoji još prostora za poboljšanje. Daljnja apstrakcija kompleksnosti XMC mikroupravljača mogla bi dodatno pojednostaviti rad s ovim alatima, što bi omogućilo širu primjenu i olakšalo proces programiranja za korisnike bez tehničkog predznanja.

Jedan od glavnih nedostataka trenutnog rješenja je ovisnost o ModusToolbox razvoju, koji mora biti instaliran kako bi sučelje moglo funkcionirati. Ova ovisnost smanjuje prenosivost sustava i komplicira proces postavljanja radnog okruženja. Buduće nadogradnje sustava trebale bi se fokusirati na uklanjanje ove ovisnosti, što bi omogućilo potpuno samostalno programiranje mikroupravljača direktno kroz Blockly sučelje. Takav razvojni smjer zahtijevao bi dodatno istraživanje i razvoj, ali bi značajno unaprijedio cjelokupni sustav.

Osim toga, potrebno je pronaći bolju metodu za stvaranje konfiguracijskih datoteka zaglavlja potrebnih za postavljanje pinova te rad s prekidima i vremenskim brojačima. Također, sučelje bi moglo biti prošireno podrškom za naprednije funkcionalnosti poput SPI, I2C protokola i drugih komunikacijskih standarda. Ove funkcionalnosti su ključne za kompleksnije projekte i njihovo uvođenje u sustav bi dodatno proširilo mogućnosti korisnika i povećalo primjenjivost sučelja u stvarnim aplikacijama.

Prednosti Blocklyja kod ugradbenog programiranja, posebno u obrazovnom kontekstu i među korisnicima koji tek ulaze u svijet programiranja postoje. Međutim, izazovi poput onih opisanih u ovom radu ukazuju na potrebu za kontinuiranim razvojem i prilagodbom alata kako bi mogli odgovoriti na sve veće zahtjeve i složenost modernih ugradbenih sustava.

U konačnici, rad na ovom projektu pokazao je da je moguće približiti složene tehničke sustave korisnicima kroz intuitivna sučelja, ali i da je za postizanje potpune funkcionalnosti i prilagodljivosti potrebno nastaviti istraživati nove metode i alate koji će dodatno unaprijediti rad u ovom području.

Literatura

[1] K. Iniewski, *Embedded Systems: Hardware, Design and Implementation*. Wiley, 2012. adresa: https://books.google.hr/books?id=IQfAgWJ3ZH0C.

[2] J. Devine, J. Finney, P. de Halleux, M. Moskal, T. Ball i S. Hodges, „Makecode and codal: Intuitive and efficient embedded systems programming for education", *Journal of Systems Architecture*, sv. 98, str. 468–483, 2019. adresa: https://www.sciencedirect.com/science/article/pii/S1383762118306088.

[3] Google, *Blockly Developer Documentation*, https://developers.google.com/blockly, Pristupljeno: 28.03.2024.

[4] K. Fontichiaro i C. Van Lent, *Coding at the Grocery Store* (21st Century Skills Innovation Library: Makers as Innovators Junior). Cherry Lake Publishing, 2020. adresa: https://books.google.hr/books?id=61K6DwAAQBAJ.

[5] S. Canet, F. Rekik, S. Dhouib i M. Coppola, „Studio4Education: Model Driven Graphical Programming of IoT applications for Education", *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, str. 1–6, 2022. adresa: https://api.semanticscholar.org/CorpusID:254458390.

[6] E. Pasternak, R. Fenichel i A. N. Marshall, „Tips for creating a block language with blockly", *2017 IEEE Blocks and Beyond Workshop (B&B)*, IEEE, listopad 2017. adresa: http://dx.doi.org/10.1109/BLOCKS.2017.8120404.

[7] D. Sun i dr., „Block-based versus text-based programming: a comparison of learners' programming behaviors, computational thinking skills and attitudes toward programming", *Educational Technology Research and Development*, 2024.

[8] Scratch, *Scratch - Imagine, Program, Share*, https://scratch.mit.edu/, Pristupljeno. 29.03.2024.

[9] Arduinos For Schools, *Modkit Micro*, adresa: https://www.modkit.com/, 2017.

[10] Node-RED, *Node-RED: Low-code programming for event-driven applications*, adresa: https://nodered.org/, 2020.

[11] A. B. Pratomo i R. S. Perdana, „Arduviz, a visual programming IDE for arduino", *2017 International Conference on Data and Software Engineering (ICoDSE)*, IEEE, studeni 2017. adresa: http://dx.doi.org/10.1109/icodse.2017.8285871.

[12] J. Moreno i J. Skoglund, „OpenPLC: An Open Source Alternative to Automation", *2015 IEEE International Symposium on Industrial Electronics (ISIE)*, IEEE, 2015., str. 20–25.

[13] D. Wolber, „App Inventor and real-world motivation", *Proceedings of the 42nd ACM technical symposium on Computer science education*, ACM, 2011., str. 601–606.

[14] National Instruments, *LabVIEW*, adresa: https://www.ni.com/en-us/shop/labview.html, 2014.

[15] Code.org, *Code.org*, `https://code.org`, Pristupljeno. 08.08.2024.

[16] B. Harvey i J. Mönig, „Bringing "No Ceiling" to Scratch: Can one language serve kids and computer scientists?", *Proceedings of the 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2012., str. 120–123.

[17] Makeblock Co., Ltd., *Global STEM Education Company*, `https://www.makeblock.com`, Pristupljeno: 21/08/2024, 2024.

[18] MathWorks, *Simulink - Simulation and Model-Based Design*, `https://www.mathworks.com/products/simulink.html`, Pristupljeno: 21/08/2024, 2024.

[19] Infineon, *XMC™ and AURIX™ – industrial microcontrollers portfolio*, `https://www.infineon.com/cms/en/product/microcontroller/`, Pristupljeno: 25.05.2024.

[20] Infineon Technologies AG, *ModusToolbox™ Tools Package User Guide*, v01.00, Munich, Germany, 2024. adresa: `https://www.infineon.com/ModusToolboxUserguide`.

[21] Infineon Technologies AG, *ModusToolbox™ Project Creator User Guide*, v01.00, Munich, Germany, 2024. adresa: `https://www.infineon.com/ModusToolboxProjectCreator`.

[22] Infineon Technologies AG, *ModusToolbox™ Device Configurator User Guide*, Munich, Germany, 2024. adresa: `https://www.infineon.com/ModusToolboxDeviceConfig`.

[23] Pallets Projects, *Flask - Web Development, One Drop at a Time*, Pristupljeno: 07/08/2024, 2024. adresa: `https://flask.palletsprojects.com/en/3.0.x/`.

[24] Mocha.js, *Mocha - the fun, simple, flexible JavaScript test framework*, Pristupljeno: 07/08/2024, 2024. adresa: `https://mochajs.org`.

[25] Chai.js, *Chai - BDD / TDD Assertion Library for Node.js and the Browser*, Pristupljeno: 07/08/2024, 2024. adresa: `https://www.chaijs.com`.

[26] L. Vulić, L. Kaućić, I. Aleksi i T. Matić, „Visual Programming Concept for Infineon XMC Series Microcontrollers”, *Zbornik radova MIPRO 2024 konferencije*, Opatija, Hrvatska, 2024.

[27] C. A. Kuklewicz, *cake-core: Main core of the CAKE Language - Block Based Programming for Infineon XMC MCUs*, Pristupljeno: 08.08.2024., 2023. adresa: `https://github.com/cra16/cake-core`.

[28] K. Beck, *Test Driven Development: By Example*. Addison-Wesley, 2003.

[29] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2019.

[30] G. J. Myers, C. Sandler i T. Badgett, *The Art of Software Testing*. John Wiley & Sons, 2011.

Sažetak

U ovom diplomskom radu istražena je mogućnost razvoja blokovskog sučelja za programiranje Infineon XMC serije mikroupravljača, koristeći Blockly kao osnovu za vizualno programiranje. Razvijen je sustav koji omogućuje korisnicima jednostavno kreiranje programa putem grafičkog sučelja, čime se smanjuje složenost kodiranja i omogućuje šira primjena mikroupravljača u industrijskim i edukacijskim okruženjima. Posebna pažnja posvećena je izradi vlastitih blokova i prevoditelja za generiranje C koda, kao i integraciji s ModusToolbox razvojnim okruženjem. Testiranje sustava provedeno je kroz nekoliko praktičnih primjera, pri čemu su identificirane mogućnosti za daljnje unaprjeđenje i optimizaciju sustava.

Ključne riječi: Blockly, vizualno programiranje, Infineon, XMC mikroupravljači, ModusToolbox, generiranje koda, blokovsko sučelje

Abstract

This thesis explores the development of a block-based interface for programming Infineon XMC series microcontrollers using Blockly as the foundation for visual programming. A system was developed to enable users to easily create programs through a graphical interface, thereby reducing the complexity of coding and broadening the application of microcontrollers in industrial and educational settings. Special attention was given to the creation of custom blocks and a translator for generating C code, as well as the integration with the ModusToolbox development environment. The system was tested through several practical examples, identifying opportunities for further enhancement and optimization.

Keywords: Blockly, visual programming, Infineon, XMC microcontrollers, ModusToolbox, code generation, block-based interface

Životopis

Luka Kaučić rođen je 2001. godine u Osijeku. Nakon završene opće gimnazije u Donjem Miholjcu upisuje Prijediplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.

Tijekom studiranja aktivno volontira u nekoliko studentskih udruga. Više puta je sudjelovao na sajmovima inovacija u Iloku i Ivanić-Gradu te ostvario izvrsne rezultate, a 2023. godine Fakultet mu dodjeljuje Priznanje za izvannastavne aktivnosti.

Najprije odrađuje QA stručnu praksu u agenciji FIVE, gdje je kratko i radio, a potom *Mainframe* stručnu praksu u tvrtki CROZ, gdje je radio godinu dana.

Suautor je dva konferencijska rada, a 2024. dobio je i Rektorovu nagradu za seminarski rad na temu kvantnog računarstva.

(Luka Kaučić)

Popis tablica

2.1. Usporedba različitih sustava vizualnog programiranja 7
3.1. Usporedba skupina XMC serije mikrokontrolera 8

Popis slika

3.1. XMC-1400 (a) i XMC-4700 (b) razvojni sustavi 9
3.2. Grafičko sučelje Project-creator alata 12
3.3. Grafičko sučelje Device-configurator alata 13
4.1. Arhitektura osmišljenog blokovskog sučelja [26] 17
4.2. Struktura projekta 19
4.3. Izgled osmišljenog blokovskog sučelja 20
4.4. Različiti pristupi vokabularu blokovskog programiranja: prirodni jezik, ikone i računalni jezik. 21
4.5. Izgled *Blockly Developer Tools* sučelja 22
4.6. Mutator (lijevo) i za njega generirani kod (desno) 23
4.7. Dinamični blok sa zavisnim padajućim izbornicima 25
5.1. Primjer 1: Trčeće svjetlo 35
5.2. Primjer 2: Kontroler i RGB LED dioda 36

Popis programskih kodova

1 Kod za definiciju bloka sa zavisnim padajućim izbornikom 24

2 Primjer prevoditelja bloka 26

3 Vlastita *workspaceToCode* metoda 27

4 Prikupljanje podataka o USB uređajima i serijskim portovima 28

5 Rekurzivna funkcija za ekstrakciju imena uređaja iz JSON podataka 28

6 Dodavanje funkcionalnosti gumba za izgradnju projekta i spremanja koda. . 30

7 Primjer Unit testa. 32