Apstrakcija hardvera za STM32 mikrokontroler

Mihelčić, Branimir

Master's thesis / Diplomski rad

2022

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:200:414435

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

Sveučilišni studij

APSTRAKCIJA HARDVERA ZA STM32 MIKROKONTROLER

Diplomski rad

Branimir Mihelčić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 09.10.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Branimir Mihelčić
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-24ARK, 29.09.2020.
OIB studenta:	94395637521
Mentor:	Doc.dr.sc. Zdravko Krpić
Sumentor:	
Sumentor iz tvrtke:	Ivan Kvolik
Predsjednik Povjerenstva:	Doc.dr.sc. Tomislav Matić
Član Povjerenstva 1:	Doc.dr.sc. Zdravko Krpić
Član Povjerenstva 2:	Doc.dr.sc. Mirko Köhler
Naslov diplomskog rada:	Apstrakcija hardvera za STM32 mikrokontroler
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Dizajnirati i implementirati Hardware abstracion layer (HAL) koji će omogućiti jednostavnoj aplikaciji čitanje i pisanje s/na jedan digitalni port (koji ima 8 pinova: 4 ulaza, 4 izlaza), dva analogna ulaza i CAN sabirnice. Implementirati upravljački sloj za mikrokontroler STM32F1 koji će HAL koristiti za pristup resursima mikrokontrolera. Iznad HAL-a implementirati jednostavnu aplikaciju koja će čitati i pisati digitalne pinove te čitati stanja analognih pinova i ispisivati ih na CAN sabirnicu. Paralelno implementirati virtualne drivere koji će omogućiti rad HAL-a i aplikacije na windows operativnom sustavu. Drivere implementirati pomoću WinSocket2 tehnologije. Virtualno okruženje također treba sadržavati malu desktop aplikaciju koja omogućava postavljanje stanja virtualiziranih ulaza/izlaza i komunikacijske sabirnice. Tema rezervirana za: Branimir Mihelčić Sumentor iz tvrtke: Ivan Kvolik (Rimac Automobili)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)

Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	09.10.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 16.10.2020.

Ime i prezime studenta:	Branimir Mihelčić
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-24ARK, 29.09.2020.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Apstrakcija hardvera za STM32 mikrokontroler**

izrađen pod vodstvom mentora Doc.dr.sc. Zdravko Krpić

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 2

2. PREGLED POSTOJEĆIH RJEŠENJA 4

3. TEHNOLOGIJE KORIŠTENE PRI IZVEDBI APSTRAKCIJSKOG SLOJA 6

3.1. ARM arhitektura 6

3.2. Razlika između procesora (CPU) i mikrokontrolera (MCU) 8

3.3. STM32 10

3.4. Eclipse IDE 12

3.5. OpenOCD 16

3.6. ST-Link 19

3.7. wxWidgets 20

3.8. Poslužitelj – klijent model 23

3.8.1. OSI model 23

3.8.2. Socket 25

3.8.3. Winsock 25

3.8.4. Metode za posluživanje više klijenata 26

4. IZVEDBA SLOJA ZA APSTRAKCIJU MIKROKONTROLERA 28

4.1. BM_HAL 32

5.2. STM32 demonstracijska aplikacija 37

5.3. Windows aplikacija 40

5.4. Virtual MCU GUI 42

LITERATURA 45

SAŽETAK 47

ABSTRACT 48

ŽIVOTOPIS 49

PRILOZI 50

1. UVOD

Industrija danas svakim danom napreduje sve više i brže te je u takvim okolnostima vrlo važan faktor uspješnosti i vrijeme isporuke proizvoda klijentima. Kako se pri razvoju ugradbenih sustava mogu dogoditi mnogi nenadani ispadi sustava, posljedice mogu biti skupe za imovinu. Na primjeru tvrtke koja razvija električne automobile, može se vidjeti da projekti te tvrtke uključuju rad s energetskom elektronikom i upravljanje velikim električnim snagama. Vrlo lako se u tom slučaju može dogoditi da uslijed nepravilnog kontrolnog sustava, dođe do pojave previsokih napona i/ili struja na komponentama sustava (tranzistorima, otpornicima, transformatorima, diodama i dr.). Moderni sustavi izuzetno su složeni i gotovo je nemoguće prije samoga testiranja pokriti sve slučajeve koji mogu dovesti do greške u radu sustava, odnosno otkloniti sve ljudske pogreške prije no što su se manifestirale u samom radu sustava.

Kako bi se taj faktor ljudske pogreške smanjio ili po mogućnosti sveo na minimum, razvijaju se metode i alati za detaljnije testiranje sustava. Jedna od kategorija alata za testiranje su i simulatori sustava. Budući da ne utječu direktno na osjetljive dijelove sustava, nego ih zamjenjuju virtualnim okruženjem, simulatori su dobar način za smanjivanje troškova tijekom razvojnog ciklusa određenog proizvoda ili uređaja. Naravno, stvarno testiranje treba provoditi i provodi se na fizičkom uređaju, no virtualno okruženje dobar je početak pri detektiranju i otklanjanju jednog dijela grešaka. Stoga bi simuliranje sustava u sigurnim uvjetima bilo od vrlo velike vrijednosti, a implementaciju sa stvarnim komponentama bi bilo potrebno izvršiti tek nakon što simulacije pokažu da razvojni tim "ide" u dobrom smjeru.

U ovom radu razvijen je programski sloj „BM_HAL" u programskom jeziku C, koji omogućuje apstrakciju mikrokontrolera tako što pruža programeru skup funkcija koje se drugačije ponašaju ovisno o ciljnoj platformi. Navedeni sloj trenutno podržava rad s STM32F103 mikrokontrolerom te Windows operacijskim sustavom. Konačna aplikacija se tako bez ikakvih promjena može prevesti i pokrenuti ili na navedenom mikrokontroleru ili Windows OS-u. U okviru rada također su izrađene dvije prototipne pločice bazirane na STM32F103 mikrokontrolerima, kako bi se mogao demonstrirati rad aplikacija koje koriste BM_HAL sloj u fizičkom okruženju. Za potrebe demonstracije rada u virtualnom okruženju izrađena je aplikacija s grafičkim sučeljem te se pomoću nje mogu generirati vanjski signali koje „virtualni mikrokontroler" može interpretirati. Pokazalo se

kako je istu aplikaciju moguće, bez dodatnih napora, pokrenuti na dva različita sustava (mikrokontroler ili računalo s Windows OS-om), jer korištenjem BM_HAL sloja aplikacijski sloj može izgledati jednako u oba slučaja. Time se omogućava programerima razvoj programske podrške u aplikacijskom sloju, bez da je potrebno voditi računa o fizičkom sloju. Nakon razvoja programske podrške, aplikacija se može testirati na računalu s operacijskim sustavom MS Windows, gdje se ujedno može verificirati i validirati rad aplikacije, a zatim se aplikacija može testirati na fizičkom mikrokontroleru.

U drugom poglavlju rada opisano je trenutno stanje auto-industrije kada je riječ o apstrakcijskim sustavima. U trećem poglavlju dan je opis svih tehnologija i pojmova korištenih pri realizaciji ideje ovog rada. Neki od tih su STM32, OpenOCD, Eclipse i drugo za izvedbu apstrakcijskog sloja, kao i tehnologija Winsock za simulacijski dio u virtualnom okruženju. U poglavlju četiri, opisana je također i sama izvedba sustava i rad značajki koje su implementirane na stvarnom mikrokontroleru i u virtualnom okruženju na Windows operacijskom sustavu.

2. PREGLED POSTOJEĆIH RJEŠENJA

U [1], dana je specifikacija AUTOSAR (*Automotive Open System Architecture*) za apstrakciju sklopovlja. U navedenom radu predložena je arhitektura u tri sloja. Prvi sloj, apstrakcija elektroničke upravljačke jedinice, (engl. *Electronic Control Unit, ECU, Apstraction Layer*), čini osnovna programska podrška u koju spadaju razni elementi nužni za rad s fizičkim sklopovljem. To mogu biti upravljački programi, operacijski sustavi, komunikacijska sučelja i slično. Drugi sloj čini okruženje koje se odnosi na sami tijek rada (engl. *Runtime Environment, RTE*), gdje je opisana komunikacija između različitih ECU-a, te treći sloj označava aplikacijski sloj. AUTOSAR standard nastao je iz potrebe da se standardiziraju komponente za automobilsku industriju, kako bi proces razvoja i dostave bio brži. Brojne tvrtke koje razvijaju sustave za automobilsku industriju predlažu vlastita rješenja sukladna s AUTOSAR standardom. Neka od rješenja mogu se proučiti u [2], i [3]. U ovom radu sustav za apstrakciju baziran je po sličnom principu, međutim funkcionalnost se proširuje i na virtualno okruženje. Time se omogućava izvršavanje programske podrške pisane u aplikacijskom sloju i na Windows operacijskom sustavu, kao i na fizičkom mikrokontroleru.

U [4], tvrtka Infineon predstavlja nekoliko alata koji omogućavaju apstrakciju mikrokontrolera. Predstavljaju aplikaciju s grafičkim sučeljem (engl. *Graphical User Interface, GUI*) DAvE koja korisniku omogućava konfiguraciju korištenog mikrokontrolera i automatski generira dio programskog koda. Ovaj alat omogućava korisniku efikasnije podešavanje i inicijalizaciju kompleksnog sklopovlja. Također, u navedenom znanstvenom radu, opisuje se implementacija i verifikacija AUTOSAR upravljačkih programa kojima se odvajaju logičke cjeline od onih bližih fizičkom sklopovlju sve do aplikacijskog sloja.

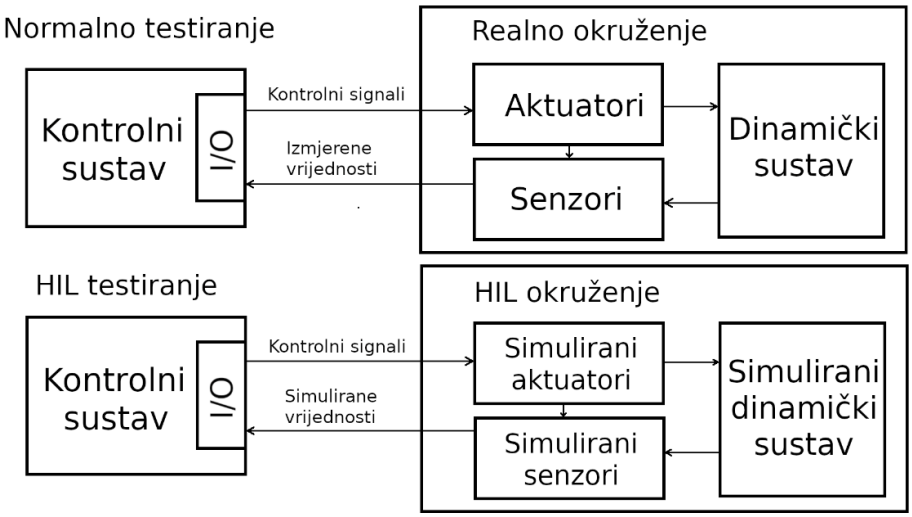
Sličan alat razvila je i tvrtka STMicroelectronics, naziva STM32CubeMX. Prema [5] to je također alat koji GUI aplikacijom omogućava korisniku lakšu konfiguraciju. Za razne značajke odabranog STM mikrokontrolera, korisniku se prikazuju sve moguće opcije koje bi se inače morale provjeravati u dokumentaciji i ručno inicijalizirati, ovim alatom taj dio programske podrške može se automatski generirati.

U [6] opisan je apstrakcijski sloj HAL razvijen od strane tvrtke ST Microelectronics. Ovaj sloj omogućava jednostavniju inicijalizaciju i razvijanje aplikacije na način da se zajedničkim funkcijama za različite mikrokontrolere može upravljati resursima mikrokontrolera. Budući da razne

linije mikrokontrolera imaju različite resurse, ne mogu se u svakom slučaju koristiti iste značajke. Kako bi se održala prenosivost programske podrške najčešće je potrebno definirati određeni simbol kojim se na jednostavan način mijenja konfiguracija programske podrške. To znači da se jednim simbolom mogu definirati sve memorijske adrese ovisne o korištenom mikrokontroleru, na način da se tokom izgradnje izvršne datoteke uzimaju u obzir različite datoteke zaglavlja. U ovom radu, za potrebe demonstracije, ali i dio programske podrške razvijanog apstrakcijskog sloja, korišten je navedeni apstrakcijski sloj tvrtke ST Microelectronics. Apstrakcijski sloj razvijan u ovom radu tako čini nadogradnju na postojeći sloj te kreiranje novog, višeg sloja s proširenim mogućnostima.

Također, u [7] opisan je sloj apstrakcije sklopovlja tvrtke Microchip. Opisana je zamišljena struktura, prednosti te nedostatci tog sloja. Glavna ideja je da se jednostavnim funkcijama mogu inicijalizirati određena sučelja mikrokontrolera, bez dodatnog podešavanja opcija koje ovise o korištenom mikrokontroleru. Pojednostavljeno, programski inženjer razvija programsku podršku u aplikacijskom sloju, a da ne mora previše „misliti“ o fizičkom sloju.

Još jedna tehnologija apstrakcije sklopovlja opisana je u [8]. U svrhu testiranja tehnoloških inovacija, razvijaju se i koncepti sklopovlja u petlji (engl. *Hardware in the loop, HIL*). U tom slučaju omogućava se izdvajanje pojedinih komponenti kompleksnog sustava (na primjer avionski pod-sustavi, automobilski, itd.) dok se preostali dijelovi simuliraju programski. Ovakav način testiranja smanjuje rizik od pogrešaka, ispada sustava i materijalne štete.



Slika 2.1. *Testiranje fizičkog sustava (gore) naspram HIL metode testiranja (dolje)*

3. TEHNOLOGIJE KORIŠTENE PRI IZVEDBI APSTRAKCIJSKOG SLOJA

O ovom poglavlju biti će opisane korištene tehnologije pri razvoju i testiranju apstrakcijskog sloja za STM32F103 mikrokontroler. To uključuje opis ARM arhitekture, opis fizičkih uređaja poput STM32 mikrokontrolera, ST-Link adaptera za programiranje i komunikaciju sa mikrokontrolerom prilikom otklanjanja pogrešaka, zatim opis programske podrške i alata za programiranje STM32 mikrokontrolera (OpenOCD, Eclipse IDE itd.) te opis poslužitelj-klijent modela, Winsock tehnologije i ostalog što je nužno za razvoj aplikacije s mogućnošću mrežne komunikacije preko TCP/IP stoga.

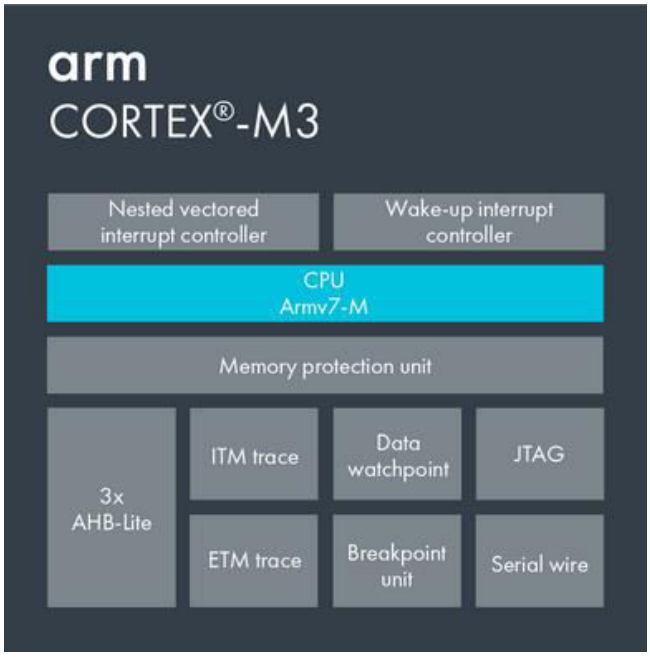
3.1. ARM arhitektura

Prema [9] ARM označava obitelj RISC (engl. *Reduced Instruction Set Computing*) procesorskih arhitektura. Tvrtka Arm Holdings ne proizvodi krajnje proizvode u obliku čipova, već samo razvijaju arhitekturu i kao takvu ju prodaju, tj. licenciraju ostalim proizvođačima koji prave fizičke čipove na temelju specifikacije te arhitekture. To uključuje, na primjer, specifikaciju procesorskih instrukcija, način na koji će se one izvršavati, organizacija i mapa memorije i ostalo što precizno definira kako će se određeni čip, baziran na toj arhitekturi, ponašati. Postoje mnoge revizije ARM arhitekture (ARMv6, ARMv6-M, ARMv7-M, ARMv7-A, itd.). Oznaka A odnosi se na aplikacijske svrhe (npr. pametni telefoni), oznaka R je linija za obradu podataka u stvarnom vremenu i oznaka M je za liniju ugradbenih mikrokontrolera (strojevi, upravljanje motorima i dr.). Također, kao što je to opisano u [10] definirane su i razne arhitekture jezgri čipova (Cortex-M0, Cortex-M1, Cortex-M3, Cortex-M4) koje su bazirane na prethodno navedenim revizijama. Proučavajući tako ARM procesore u početku može biti nejasno koji dio je točno licenciran od strane Arm Holdings-a, a koji su dijelovi specifični za proizvođača. Glavna razlika je što npr. ARMv7-M definira CPU (engl. *Central Processing Unit*), dakle centralni procesorski dio koji manipulira podacima, izvršava određene operacije nad njima, poput zbrajanja, oduzimanja i drugo te ih prosljeđuje nekim drugim obradnim jedinicama. Te dodatne obradne jedinice definira arhitektura jezgre (npr. Cortex-M3). Jezgra označava sve dodatne obradne jedinice zajedno sa procesorskom jezgrom (CPU). Dakle, specifikacija jezgre čipa, proširenje je specifikacije centralne procesorske jedinice i taj dio također definira ARM Holdings. U kasnijim poglavljima biti će opisan dodatan sloj, koji implementira neki

drugi proizvođač (STMicroelectronics, NXP, Texas Instruments, itd.), sloj izgrađen oko neke od navedenih ARM jezgri.

Na slici 3.1. dan je pregledan blokovski prikaz Cortex-M3 jezgre koja čini središnji dio STM32F103 mikrokontrolera, korištenog u ovom radu. Vidljivo je da je sam CPU zasebna cjelina zadužena za izvršavanje matematičkih operacija nad ulaznim podatcima čiji je način obrade definiran ARMv7-M specifikacijom. Oko centralne procesorske jedinice dodani su ostali gradivni blokovi koji omogućavaju razne značajke koje sam procesor ne bi mogao izvršavati. Neki od tih blokova su: ugradbeni vektorski upravljač prekidima (engl. *Nested vectored interrupt controller, NVIC*), komunikacijsko i testno sučelje JTAG (engl. *Joint Test Action Group*), točke prekida, jedinica za zaštitu memorije i ostalo.

JTAG modul je tako zadužen samo za praćenje određenih signale te omogućava komunikaciju, otklanjanje grešaka (engl. *debugging*), programiranje i drugo što naposljetku smanjuje opterećenost procesora.



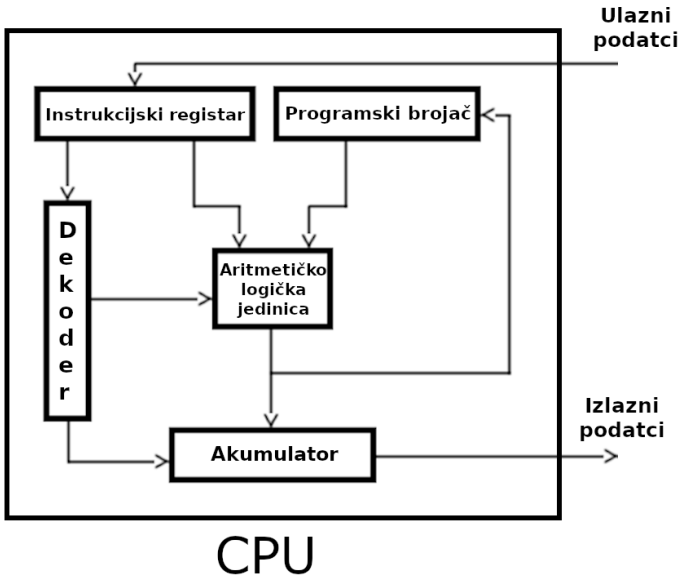
Slika 3.1. *ARM Cortex-M3 blok dijagram*[1]

[1] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m3

3.2. Razlika između procesora (CPU) i mikrokontrolera (MCU)

Na slici 3.2. može se vidjeti blokovski prikaz centralne procesorske jedinice. Prema [11] CPU je elektronička komponenta zadužena za izvršavanje operacija programa. Procesor nad ulaznim podatcima izvršava aritmetičke, logičke te ulazno/izlazne operacije određenim redoslijedom kako je to zapisano u programu. Izvedba CPU arhitekture ovisi od proizvođača do proizvođača pa tako i Arm Holdings licencira vlastitu dizajniranu arhitekturu. No svaki procesor je intrinzično sličan jedan drugome, odnosno sastoje se od sličnih blokova. Tako se unutar CPU-a može naći:

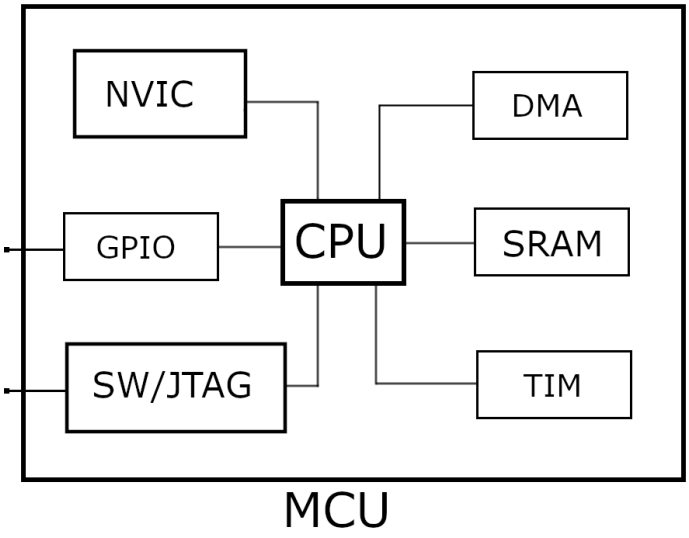
- ALU (Aritmetičko logička jedinica) – izvršava aritmetičke i logičke operacije
- Procesorski registri – sadrže trenutne podatke nužne za izvršavanje instrukcija (instrukcijski registar, programski brojač)
- Kontrolna jedinica – upravlja dohvaćanjem i izvršavanjem instrukcija iz memorije
- AGU (Generator adresa) – izračunava adrese za pristup procesora u memoriji



Slika 3.2. *Centralna procesorska jedinica*

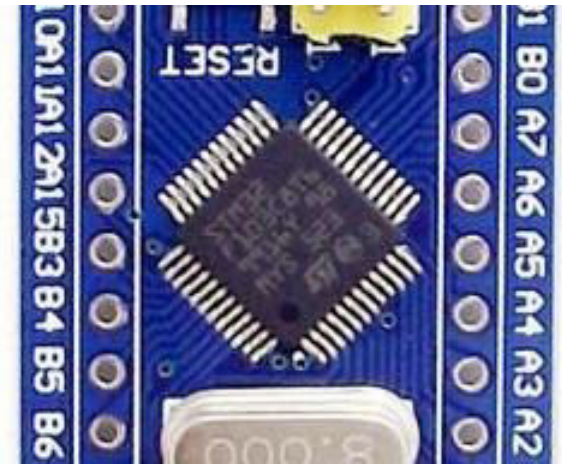
Ovaj dio (CPU) definiran je u slučaju ARM arhitekture nekom od revizija arhitekture, npr. ARMv7-M. Ta revizija stoga definira ponašanja procesora, set asemblerskih instrukcija koje taj procesor može izvršavati i način na koji se te instrukcije izvršavaju.

Nasuprot tomu, mikrokontroler je zaseban funkcionalan blok dizajniran za ugradbene aplikacije te samostalno može izvršavati određeni programski kod, budući da sadrži sve potrebne cjeline za to. Mikrokontroler građen je oko centralne procesorske jedinice, kojemu su dodani blokovi poput radne memorije (engl. *Random Access Memory, RAM*), modul za direktan pristup memoriji (engl. *Direct Memory Access, DMA*), NVIC, ulazi i izlazi opće namjene (engl. *General Purpose Input Output, GPIO*), vremenski brojač (engl. *Timer, TIM*), komunikacijsko sučelje za SW (engl. *Serial Wire*) protokol, JTAG i ostalo. Blokovski prikaz mikrokontrolera vidljiv je na slici 3.3., dok se fizički izgled "silicija" može vidjeti na slici 3.4.



Slika 3.3. *Općeniti blokovski prikaz mikrokontrolera*

Primjetljivo je da se obično na razvojnim pločama koji sadrže mikrokontroler ne može često pronaći mnogo komponenata, tek poneki otpornik ili kondenzator. To je iz razloga što su današnji mikrokontroleri toliko razvijeni, funkcionalnosti su im brojne te sadrže gotovo sve potrebno čineći vanjske čipove poput dodatnih operacijskih pojačala ili analogno-digitalnih pretvarača, suvišnima.



Slika 3.4. *STM32F103C8* mikrokontroler[2]

3.3. STM32

STM32 obitelj mikrokontrolera čine 32-bitni mikrokontroleri bazirani na ARM Cortex-M jezgri. Korisnicima pružaju mnoge mogućnosti ovisno o aplikaciji. Tako postoje linije iz te obitelji koje obilježavaju visoke performanse, obrada podataka u stvarnom vremenu, obrada digitalnih signala, rad u režimu niske potrošnje energije, povezivost i drugo [10], [12]. U ovom diplomskom radu korišten je STM32F103C8 mikrokontroler baziran na ARM Cortex-M3 jezgri koja radi na taktu 72 MHz. Sadrži također *Flash* (128 kB) memoriju i SRAM (20 kB) velike brzine. Osim toga ovaj mikrokontroler sadrži brojne dodatne periferijske blokove poput 12-bitnog analogno-digitalnog pretvarača (engl. *Analog to Digital Converter,* ADC), tri vremenska brojača (engl. *Timer*) opće namjene, jedan vremenski brojač za pulsno širinsku modulaciju (engl. *Pulse Width Modulation, PWM*) te dodatne standardne komunikacijske blokove kao što su I2C, SPI, USART, USB i CAN [13].

Za rad ove linije mikrokontrolera (STM32F103xx) visokih performansi potreban je napon od 2.0V do 3.6V. Postoje izvedbe deklarirane za rad od -40 °C do +85 °C ili naprednija verzija za okruženja od -40 °C do +105 °C.

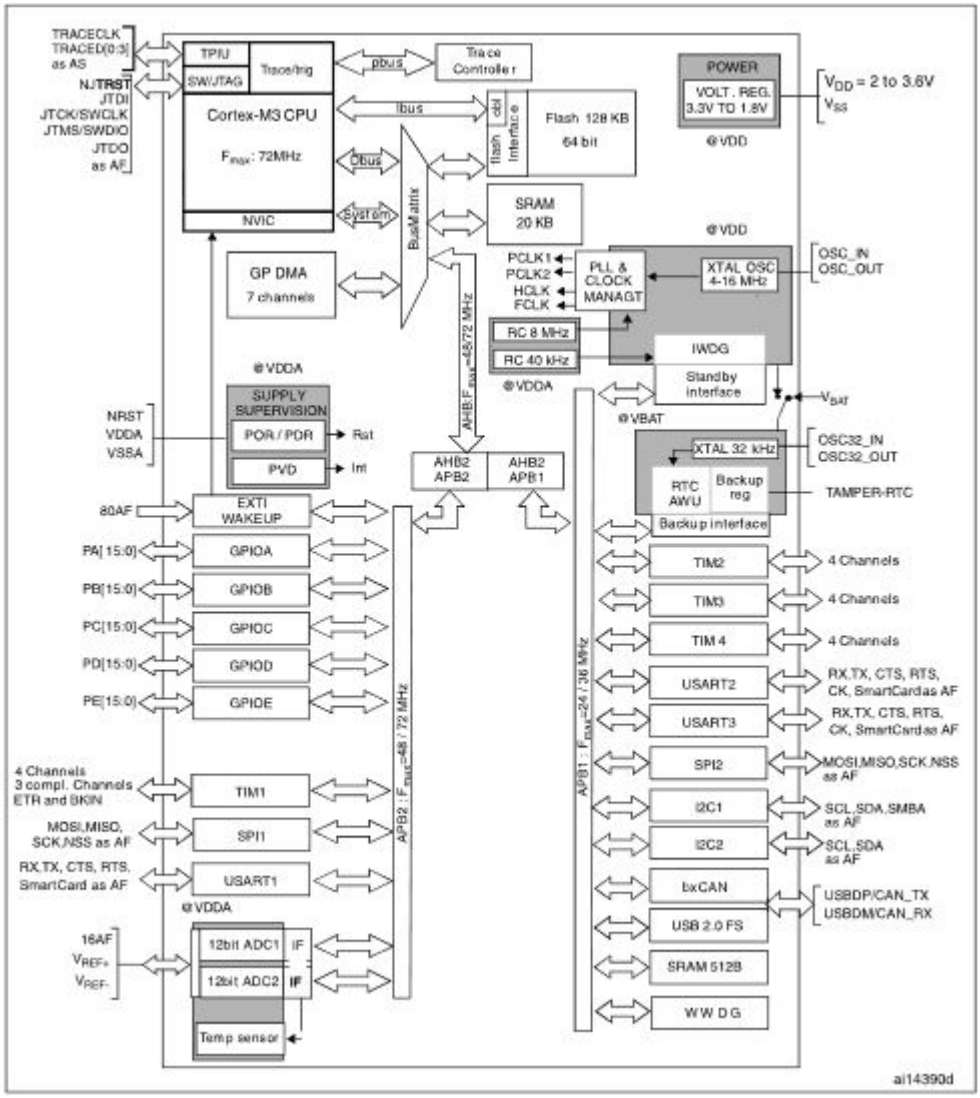
Što se tiče fizičke izvedbe postoje šest različitih pakiranja, od 36 do 100 pinova.

Na slici 3.4. prikazan je STM32F103C8 mikrokontroler za kojega je rađena apstrakcija te koji služi za demonstraciju ideje ovog rada. Unutar ovog fizičkog čipa nalaze se prethodno objašnjeni gradivni

[2] https://www.hackster.io/paramaggarwal/converting-an-stm32f103-board-to-a-black-magic-probe-e701d4

blokovi. Počevši od samog CPU-a baziranog na ARMv7-M arhitekturi, zatim dodavanjem blokova poput NVIC, SW/JTAG, AMBA-AHB (engl. *Advanced Microcontroller Bus Architecture - Advanced High-Performance Bus*) sučelje (opisuje podatkovni i instrukcijski tok) implementirana je Cortex-M3 jezgra i zatim dodavanjem blokova specifičnim za proizvođača nastane ovakav jedan mikrokontroler. U ovom slučaju tvrtka STMicroelectronics dodaje gradivne blokove oko Cortex-M3 (ili neke druge) jezgre i kreira linije mikrokontrolera ovisno o namjeni.

Na slici 3.5. prikazan je blokovski prikaz STM32F103 linije mikrokontrolera. U gornjem lijevom kutu vidljiva je Cortex-M3 jezgra te usko povezani joj blokovi poput NVIC ili SW/JTAG koje definira ARM Holdings tvrtka. Neke od ostalih blokova definiraju proizvođači čipova (u ovom slučaju ST Microelectronics) i to je na slici vidljivo pod blokovima periferije (GPIO, Timer, ADC, UART, SPI…). ST Microelectronics za navedene blokove također pruža i programsku podršku koji omogućavaju brži i jednostavniji razvoj aplikacija. O upravljačkim programima poput sloja za apstrakciju sklopovlja (engl. *Hardware Abstraction Layer, HAL*) biti će nešto više u narednim poglavljima.



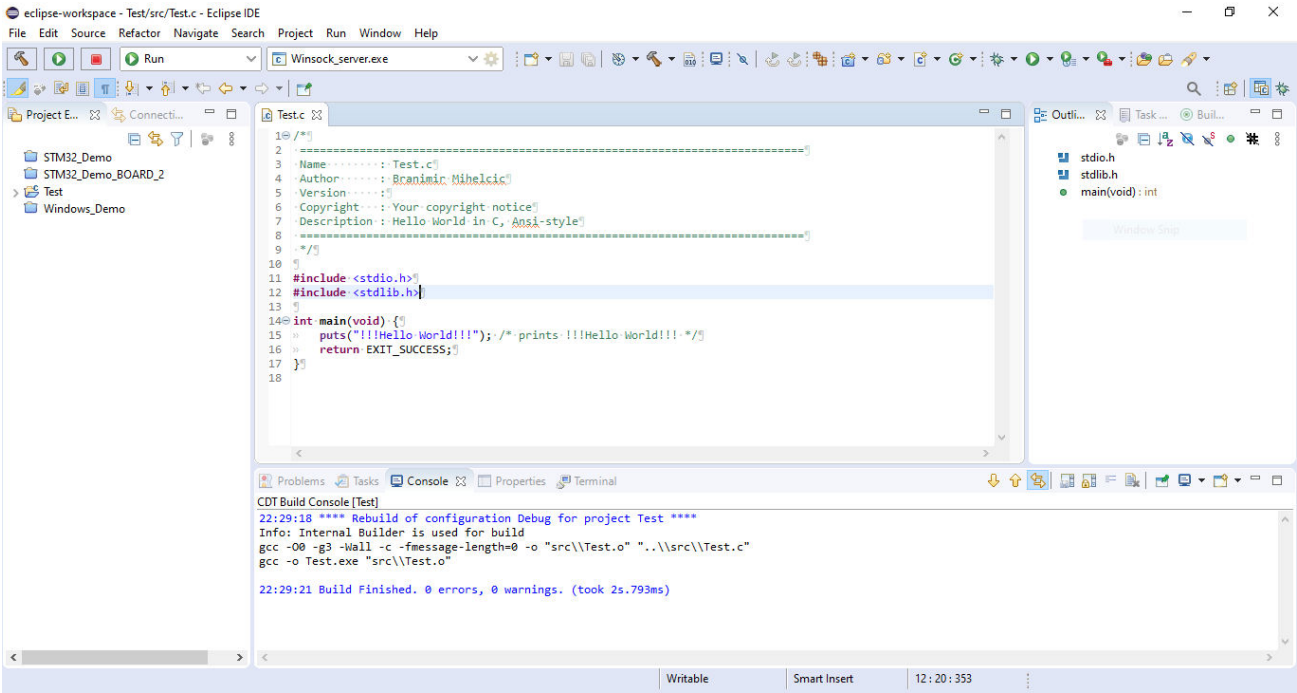
Slika 3.5. *Blokovski dijagram STM32F103x8 mikrokontrolera*[3]

3.4. Eclipse IDE

Eclipse integrirano razvojno okruženje (engl. *Integrated Development Environment, IDE*) jedan je od najpoznatijih programa korištenih u razvoju programske podrške. Osim osnovnog programa, postoje i razne izvedbe bazirane na Eclipse IDE-u. Tako na primjer tvrtke poput ST-a mogu prilagoditi Eclipse IDE program razvoju aplikacija za njihove mikrokontrolere te razviti IDE sa integriranim lancem alata (engl. *Toolchain*) za programiranje njihove linije mikrokontrolera. Tvrtka

[3] STM32F103x8, STM32F103xB datasheet *https://www.st.com › resource › datasheet › stm32f103c8*

ST razvila je tako alate STM32CubeIDE i Atollic TrueStudio, zasnovane na Eclipse IDE platformi. Sastavni dio programa čini radna površina koja omogućava uređenje koda. U isto vrijeme u programu je moguće prikazivati i mapu projekta sa svim pripadajućim datotekama te popis varijabli, funkcija, pretprocesorskih naredbi i ostalo, što uvelike olakšava kretanje po programskom kodu. Primjer takvog izgleda, odnosno "pogleda" (engl. *view*) programa moguće je vidjeti na slici 3.6. Osim pogleda za uređenje koda najznačajniji pri razvoju programske podrške je i pogled za otkirvanje i otklanjanje pogrešaka, tzv. *Debug view*.

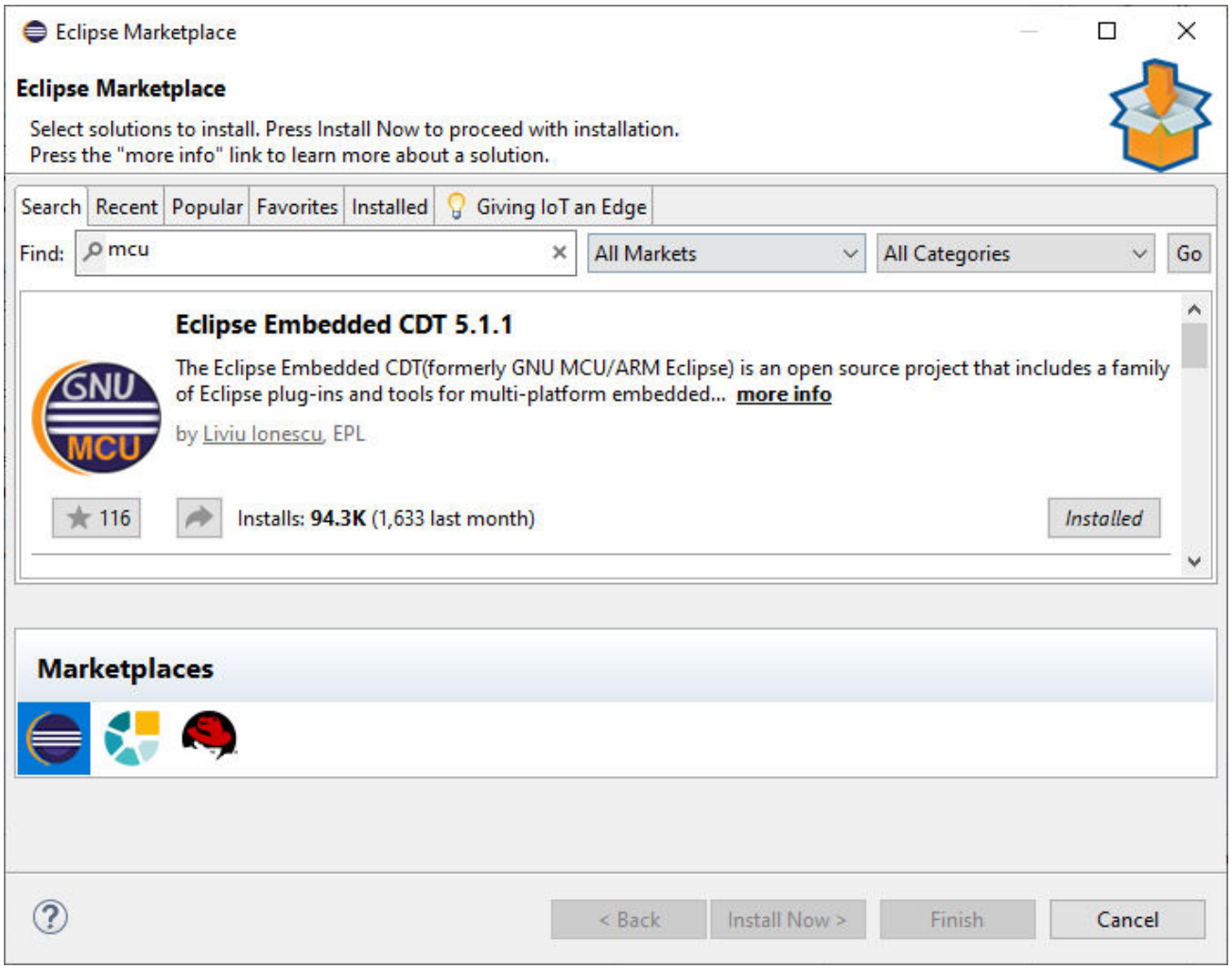


Slika 3.6. *Izgled Eclipse razvojnog okruženja*

Jedna izvanredna stavka koja određuje visoku mogućnost adaptacije Eclipse IDE raznim potrebama je mogućnost uključivanja programskih paketa. Budući da je Eclipse napisan uglavnom u Java programskom jeziku, svoju primarnu primjenu nalazi u razvoju Java aplikacija. Međutim kroz dodatne programske pakete i nastavke omogućeno je i programiranje u raznim drugim jezicima kao što su: Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Lasso, Lua, Perl, PHP, Prolog, Python, R, Ruby [14]. Eclipse platforma velik je projekt što potvrđuje i brojnost objavljivanja novih verzija. Od 2006. godine objavljuje se nova verzija jednom godišnje kroz istovremene objave (engl. *Simultaneous Release*) što uključuje Eclipse platformu i druge Eclipse projekte. Od 2018. Objavljivanje novih verzija izvršava se u kvartalima, četiri puta godišnje. U vrijeme rada na ovom diplomskom radu korištena je verzija 2020-03 (4.15.0), Eclipse

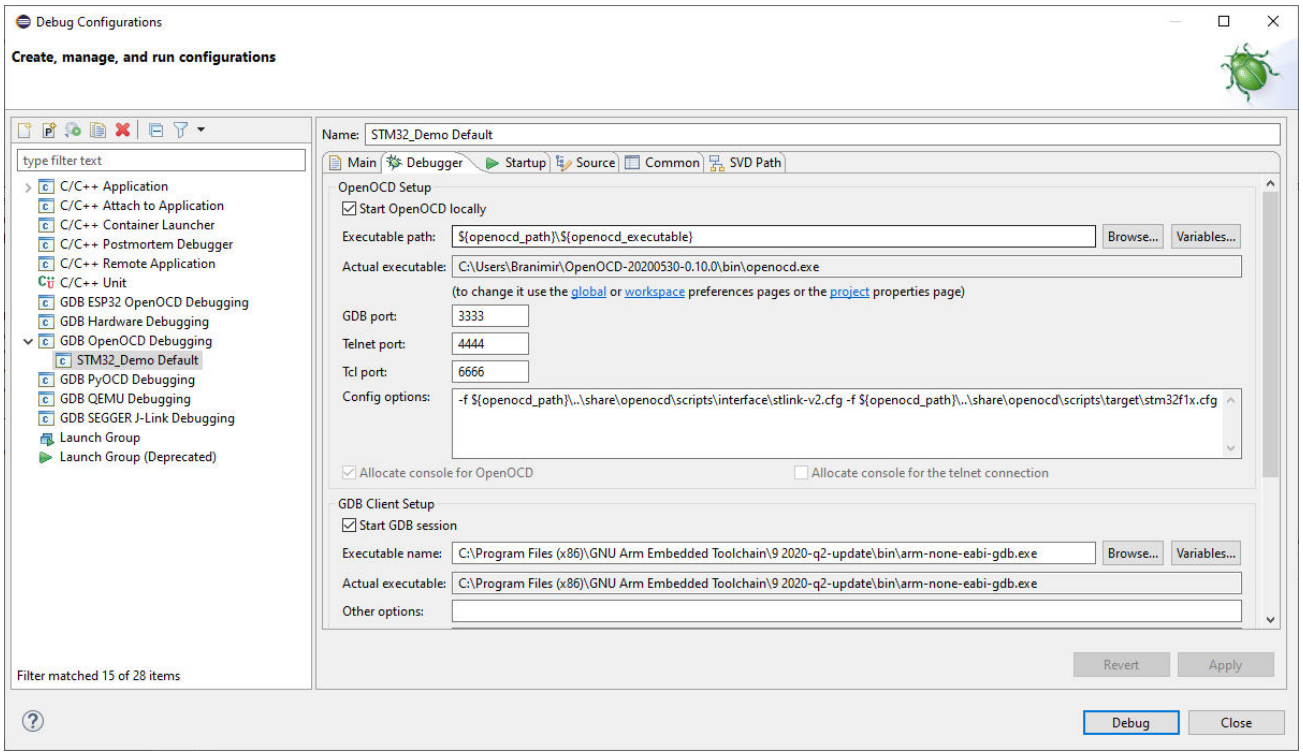
IDE za razvoj C/C++ programske podrške. Kako je navedeno Eclipse IDE omogućava uključivanje raznih nastavaka i programskih paketa. U svrhu olakšanog razvoja programske podrške, odnosno aplikacija za mikrokontrolere zajednica radi na vrlo korisnom paketu "Eclipse Embedded CDT" (prikaz paketa unutar *Eclipse* IDE-a nalazi se na slici 3.7.). Jednostavnom instalacijom u nekoliko klikova, omogućeno je integriranje alata potrebnih za razvoj programske podrške za mikrokontrolere. Neke od glavnih stavki ovog paketa su prema [15]:

- Izrada/izgradnja/upravljanje ugradbenim ARM/Aarch64/RISC-V aplikacijama bez potrebe za pisanjem vlastitog Makefile-a
- Pružanje gotovih predložaka za neke ARM Cortex-M procesore
- Pružanje mogućnosti traženja i otklanjanja grešaka putem JTAG/SWD
- Omogućavanje pogleda na registre i njihova izmjena prilikom *debug* sesije
- Podrška za mnoge 32-bitne i 64-bitne lance alata (engl. *toolchain*)



Slika 3.7. *Paket koji sadržava brojne alate potrebne za razvoj aplikacija za ugradbene sustave*

Eclipse IDE program može biti zahtjevan za početnike zbog svoje velike kompleksnosti i velikog broja postavki koje je moguće mijenjati i pomoću njih prilagođavati program za vlastitu aplikaciju. Kako je Eclipse Embedded CDT paketom omogućeno upravljanje aplikacija za mikrokontrolere, pa tako i STM32 linije, instalacijom paketa omogućava se i korištenje OpenOCD programske podrške. Na slici 3.8. prikazana je konfiguracija prilikom *debugiranja* OpenOCD programom. U Eclipse IDE, konfiguracija se podešava pod „*Run -> Debug Configuration -> GDB_OpenOCD_Debugging -> Debugger -> Config Options*“. Tu se predaju parametri OpenOCD programu pomoću zastavice “-f” praćene konfiguracijskom datotekom. Za rad s STM32F103C8 mikrokontrolerom potrebno je predati “stm32f1x.cfg” datoteku iz *target* mape te “stlink-v2.cfg” iz mape *interface* ukoliko se koristi ta verzija ST-Link-a.



Slika 3.8. *Postavljanje OpenOCD konfiguracije*

3.5. OpenOCD

OpenOCD (*Open On-Chip Debugger*) jedan je od mogućih načina programiranja mikrokontrolera pomoću računala. Točnije to je dodatna programska podrška koja omogućava komunikaciju s mikrokontrolerom, no najčešće je uz to potrebno i dodatno fizičko sklopovlje. Kako je navedeno u [16] OpenOCD je program koji omogućava *debugiranje*, programiranje i *boundary-scan* testiranje raznih mikrokontrolera. Obično je potrebno dodatno sklopovlje, odnosno, *debug adapter* koji je posrednik između programske podrške na *host* računalu i ciljnog (engl. *target*) uređaja/mikrokontrolera. OpenOCD nastao je kao diplomski rad [17] Dominica Ratha, 2005. godine i od tada, projekt je izrastao u veliki, aktivni projekt otvorenog koda, podržan od strane raznih programerskih zajednica i razvojnih inženjera diljem svijeta.

U praksi se trenutno mogu naći nekoliko adaptera koji podržavaju jedan ili više transportnih protokola. Postoje izvedbe u obliku odvojenih adaptera (poput ST-Linka), a ponekad ih proizvođači razvojnih ploča postavljaju direktno na ploču sa već spojenim podatkovnim linijama sa

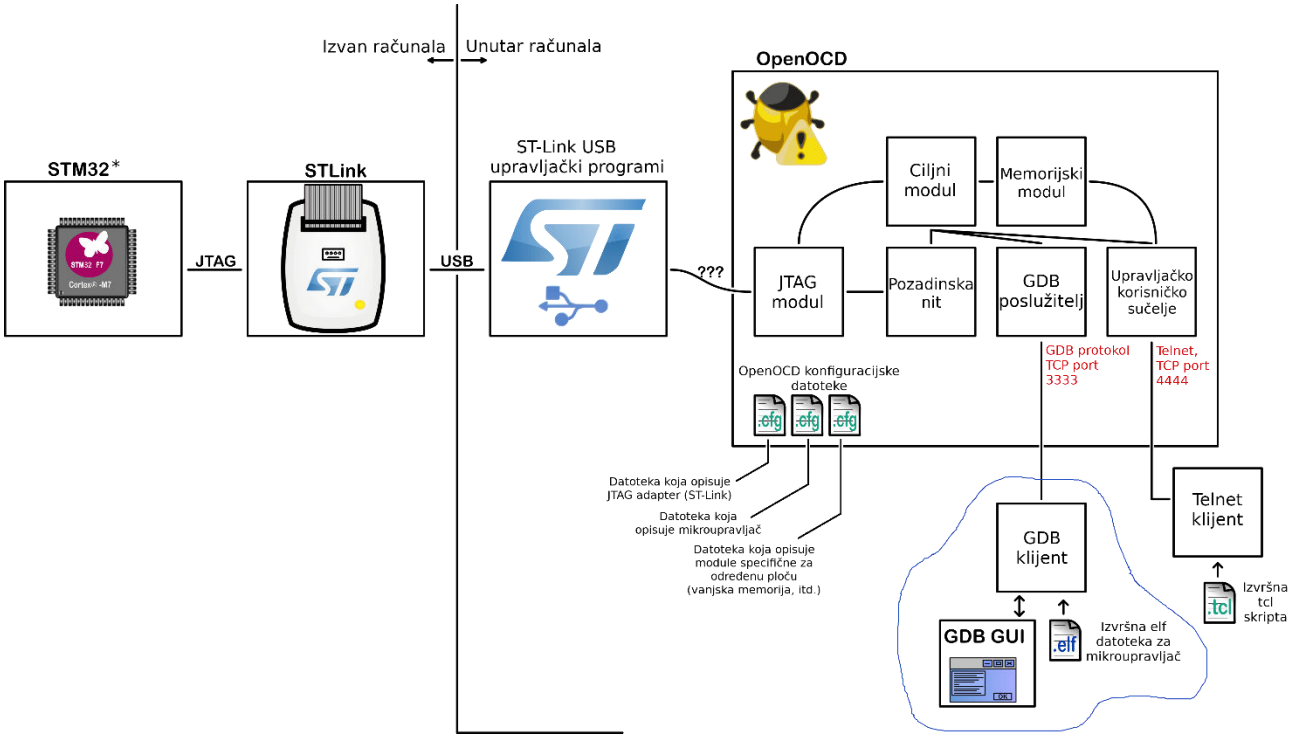
mikrokontrolerom, tako da se programiranje može na lagan način izvršiti, jednostavnim spajanjem USB kabela. Također spajanjem na takav način, mikrokontroleru se daje i napajanje preko USB-a, tako da se razvoj sustava ili testiranje može izvršiti bez dodatnih spajanja konektora.

Kako je navedeno, čest je slučaj da odabrani mikrokontroler podržava JTAG standard, jedan od najpopularnijih standarda kada je u pitanju programiranje mikrokontrolera. U slučaju kada razvojni inženjer želi prenijeti svoj programski kod sa računala na mikrokontroler, ovim standardom to je omogućeno. Najčešće se programski kod razvija u jednom od integriranih razvojnih okruženja. Kada je program napisan, potrebno je napraviti izvršnu datoteku. To je binarna datoteka koju mikrokontroler može intepretirati. Za generiranje izvršne datoteke potrebno je ispravno podesiti lanac alata (engl. *toolchain*) za korištenu arhitekturu. U ovom slučaju korišten je "ARM Cross GCC" *toolchain*. U tom *toolchain-u* dostupni su pretprocesor, prevoditelj, *asembler* te *linker*. Napisani programski kod prolazi kroz sve te alate kako bi se iz teksta izradio izvršni kod:

1. Pretprocesor – izlazna datoteka je i dalje C kod, zamjenjuju se svi dijelovi koje je programer nastojao skratiti pomoću #define, #include i ostalih naredbi koje počinju sa '#',
2. Prevoditelj – rezultat prevoditelja je asemblerski kod za danu arhitekturu,
3. *Asembler* – prevodi izlaz iz prevoditelja u strojni kod te se takva izlazna datoteka naziva "objektni kod",
4. *Linker* – budući da se često projekt sastoji od nekoliko izvornih (engl. *source*) datoteka potrebno je u zadnjem koraku sve objektne datoteke povezati u jednu cjelinu, tj. izvršnu datoteku sa svim funkcijama i varijablama koji su memorijski mapirani.

Ovaj postupak se najčešće automatizira ispravnim podešavanjem *toolchain*-a te pisanjem posebne datoteke za *make* alat ("*GNU make*" je poseban projekt koji omogućava automatizaciju prevođenja programskog koda). Takva datoteka najčešće se naziva *Makefile* te omogućuje programerima prevođenje programskog koda unutar IDE-a jednostavnim klikom na "*build"* tipku, nakon čega se dobiva izvršna binarna datoteka. Osim ručnog pisanja *Makefile* datoteke, moguće je podesiti IDE za automatsku generaciju iste. Kada se izvršna datoteka želi prenijeti na odabrani mikrokontroler, moguće je iskoristiti funkcionalnosti OpenOCD programske podrške. Pokretanjem OpenOCD-a, pokreću se dva poslužitelja (*telnet i gdb*). Programer se zatim može spojiti na jedan od ta dva poslužitelja (preko TCP veze) te slati naredbe vanjskom sklopovlju. Nakon što poslužitelj primi poslane naredbe proslijedit će ih do *debug* adaptera preko USB protokola. *Debug* adapter će primiti

te podatke i poslati ih ciljni mikrokontroler JTAG protokolom. Modul na mikrokontroleru koji prati JTAG signale, može interpretirati naredbe u takvom obliku te manipulirati unutarnjom memorijom i/ili slati povratne informacije. Na slici 3.9. prikazan je blok dijagram cijelog OpenOCD sustava.



Slika 3.9. *OpenOCD blok dijagram*[4]

Postoje također SWD (engl. *Serial Wire Debug*) adapteri, sa signalima koji mogu komunicirati sa ponekim novijim ARM jezgrama. SWD protokol podržava otklanjanje grešaka (*debugging*) kao i programiranje mikrokontrolera , no ne podržava *boundary scan* kao JTAG. *Boundary scan* koristi se za detekciju sklopovskih pogrešaka kao stanje vodova, naponskih razina, stanja pinova mikrokontrolera itd.

Bitno je još napomenuti da se OpenOCD programskoj podršci moraju pružiti konfiguracijske datoteke za *debug* adapter i *target* mikrokontroler, budući da su podržane mnoge arhitekture i mikrokontroleri. U ovom radu korišten je STM32F103C8 mikrokontroler te vanjski ST-Link-V2.

[4] https://stackoverflow.com/questions/38033130/how-to-use-the-gdb-gnu-debugger-and-openocd-for-microcontroller-debugging-fr

3.6. ST-Link

Aplikacije za ugradbene sustave najčešće se razvijaju na osobnim računalima iz razloga što imaju više resursa na raspolaganju nego izravno na korištenim platformama za ugradbeni sustav pa je i sam razvoj aplikacija brži i efikasniji. Međutim, direktna komunikacija *host* i *target* uređaja nije uvijek moguća, a to može uključivati bilo kakvu razmjenu podataka od traženja i otklanjanja pogrešaka do osnovnog prijenosa koda. Nakon što je procesom prevođenja dobivena izvršna datoteku za *target* uređaj, jedan od načina prijenosa te datoteke je pomoću ST-Link uređaja kojeg možemo vidjeti na slici 3.10. To je adapter koji povezuje računalo i mikrokontroler. Služi kao "prevoditelj" između ta dva sustava kako bi oni mogli ispravno komunicirati, što znači da generira odgovarajuće signale za oba sustava. Mikrokontroler može tako komunicirati s računalom putem protokola JTAG, kojeg računala obično ne podržavaju bez dodatne programske podrške i sklopovlja. Što se tiče strane mikrokontrolera, tvrtka ARM razvila je također tehnologiju/protokol SWJ-DP (engl. *Serial Wire/JTAG – Debug Port*) koji omogućava korištenje oba protokola (JTAG ili SWD) na istim fizičkim pinovima te se modul koji podržava navedenu tehnologiju nerijetko nalazi na STM32 mikrokontrolerima. Ovaj modul nalazi se i na korištenom STM32F103C8 mikrokontroleru te je za programiranje čipa potrebno povezati "SWDIO" i "SWCLK" linije mikrokontrolera i ST-Link-a. SWJ-DP nam je stoga najbitnija tehnologija kada je u pitanju programiranje, odnosno prijenos izvršne datoteke s računala na mikrokontroler.

Slika 3.10. *ST-Link uređaj*[5]

3.7. wxWidgets

Prema [18] wxWidgets je C++ biblioteka koja omogućava programerima izradu aplikacija za Windows, macOS, Linux i ostale platforme sa jedinstvenom bazom programskog koda. Postoje implementacije za neke popularne programske jezike kao što su Python, Perl, Ruby i mnogi drugi te suprotno nekim drugim bibliotekama za različite platforme, omogućava aplikacijama originalan izgled i dojam ovisno o korištenom operacijskom sustavu. To je postignuto korištenjem sučelja za programiranje aplikacija (engl. *application programming interface, API*) koje je domaće (engl. *native*) za pojedine platforme, umjesto da se grafičko sučelje emulira na neki način. Također wxWidget bibiloteka je kroz godine prošla mnoge testove, mnoge greške su ispravljene, projekt je sazrio i što je zanimljivo, ovo je besplatna biblioteka otvorenog koda.

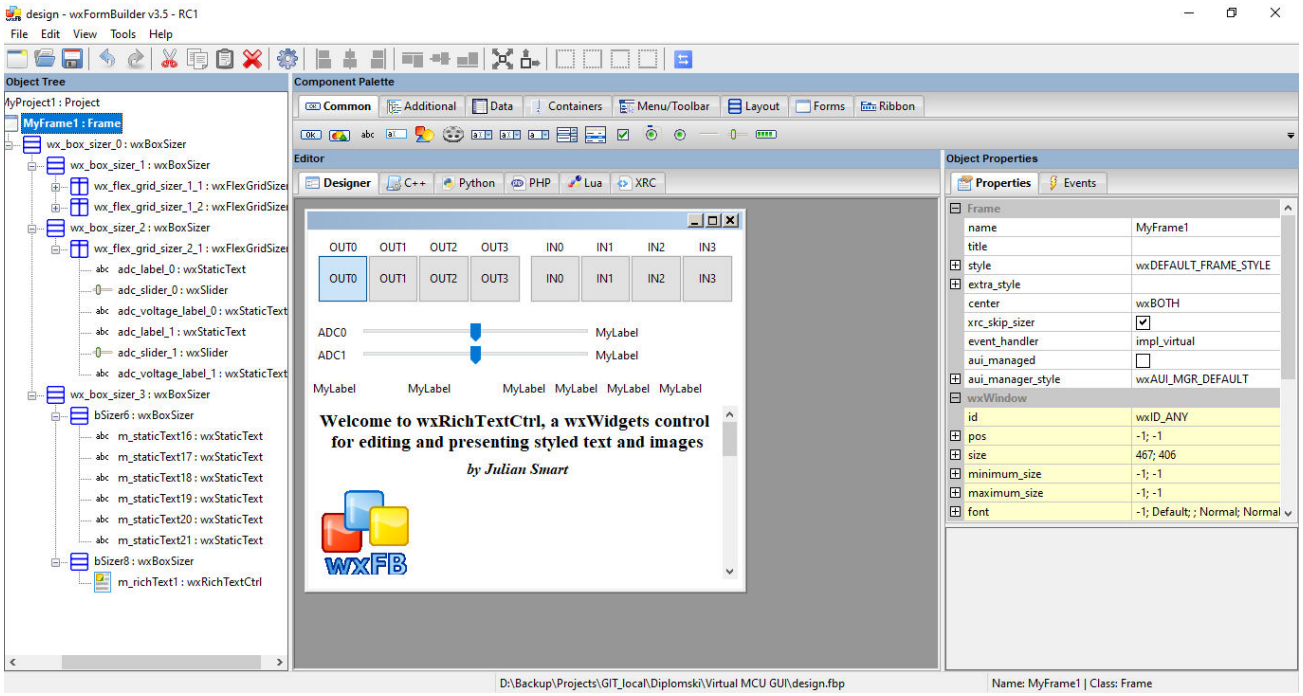
[5] https://www.st.com/en/development-tools/st-link-v2.html

Slika 3.11. *Izgled wxWidgets tipke na različitim platformama*[6]

Za programiranje wxWidgets bibliotekom potrebno je instalirati pripadajući wxWidgets paket sa svim potrebnim elementima. Upute za podešavanje sustava moguće je pronaći pod [18]. Također, kao i sa ostalim projektima otvorenog koda, moguće je i za ovaj preuzeti datoteke izvornog koda, izgraditi ih na svom sustavu te takve izvršne datoteke koristiti u projektima. Drugi izbor je preuzimanje već izgrađenih dinamičkih biblioteka. Korisnik ima slobodu izbora pri svom radu pa tako može sudjelovati i biti uključen u poboljšanje i daljnji razvoj cijelog projekta.

Kako bi se lakše mogao započeti rad sa wxWidgets bibliotekom korišten je alat "wxFormBuilder" koji omogućava dodavanje wxWidgets elemenata iz izbornika te generiranje osnovnog koda za inicijalizaciju tih dodanih elemenata. Alat je vrlo koristan jer je za svaki dodani element moguće također i mijenjati različite parametre poput veličine prozora, granice, razne zastavice i drugo, nakon čega se odmah u programu dinamičkim prilagodbama mogu uočiti promjene. To dodatno ubrzava proces razvoja, jer bi se inače svaka promjena morala isprobavati ručno, odnosno, nakon izgradnje aplikacije, svaki puta iznova pokretati kako bi rezultati promjena postali vidljivi.

[6] https://docs.wxwidgets.org/3.0/classwx_button.html



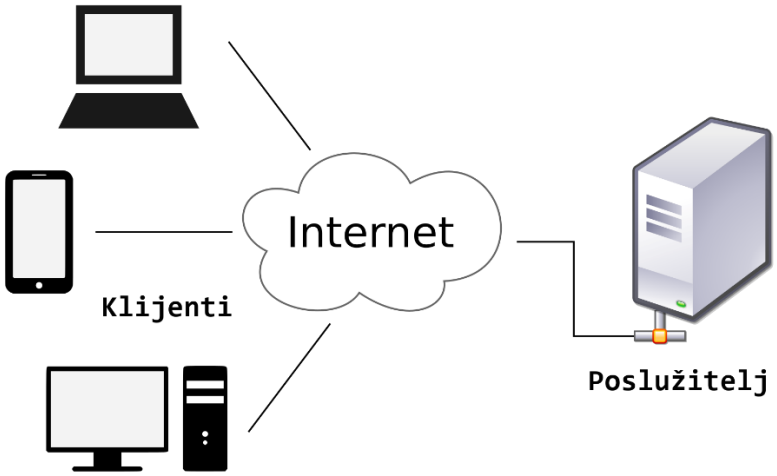
Slika 3.12. *wxFormBuilder program za izradu wxWidgets aplikacija*

Pri izradi aplikacije iz ovog rada korišteni su sljedeći wxWidgets elementi:

- wxBoxSizer
- wxFlexGridSizer
- wxToggleButton
- wxStaticText
- wxSlider
- wxMenuBar
- wxMenu
- wxMenuItem
- wxRichTextCtrl
- wxSocketClient

3.8. Poslužitelj – klijent model

Klijent-poslužitelj model distribuirana je aplikacijska struktura koja podjeljuje zadatke i rad između pružatelja resursa ili usluga, zvanih poslužitelj, i klijenata koji šalju zahtjeve za resursima i uslugama. Često klijenti i poslužitelji komuniciraju preko računalne mreže sa različitih uređaja, no isto tako, klijent i poslužitelj mogu biti unutar istoga sustava. U jednoj klasičnoj internetskoj mreži, poslužitelj tako može izvršavati nekoliko programa u isto vrijeme te odgovarati na zahtjeve klijenata i dijeliti svoje resurse. Klijenti u tom slučaju ne dijele svoje resurse, ali mogu zatražiti isporuku sadržaja ili servisa poslužitelja [19].



Slika 3.1. *Poslužitelj – klijent model*[7]

3.8.1. OSI model

OSI-model ili referentni model za otvoreno povezivanje sustava je najkorišteniji apstraktni opis arhitekture mreže. Opisuje komunikaciju sklopovlja, programa i protokola pri mrežnim komunikacijama. Koriste ga proizvođači pri projektiranju mreža, kao i stručnjaci pri proučavanju mreža. OSI model dijeli arhitekturu mreže u sedam logičkih razina, daje spisak funkcija, servisa i protokola koji funkcioniraju na svakoj razini [20].

[7] https://en.wikipedia.org/wiki/Clien-server_model

Tablica 5.1. *Slojevi OSI modela*[8]

Slojevi	**Prijenosna jedinica**	**Protokoli**
Aplikacija	Podatak	HTTP, FTP, Telnet, DNS, SSH, …
Prezentacija	Podatak	ALP, LPP, …
Sesija	Podatak	PAP, SCP, ASP, …
Transport	Segment/Datagram	TCP, UDP
Mreža	Paket	IP, ICMP, ARP
Veza	Okvir (engl. *Frame*)	HDLC, PPP, …
Fizički sloj	Bit	Token Ring, IEEE 802.11

Kada jedno računalo komunicira s drugim, poruka koju želi poslati prolazi kroz niz različitih slojeva, zaduženih za pojedine funkcionalnosti. U slučaju kada nekakva aplikacija na računalu želi poslati poruku, ona kreće od sloja aplikacije sve do donjeg fizičkog sloja. Fizički sloj označava tako fizičke signale (npr. napon), definira prijenos pojedinih bitova, odnosno poruka se u tom slučaju nalazi na komunikacijskom kanalu. Na drugoj strani poruka se prima obrnutim redoslijedom, od fizičkog nivoa sve do rekonstrukcije originalne poruke na aplikacijskoj razini. Poruka se kroz taj ciklus slanja i primanja razdvaja, odnosno poprima razne oblike. Pri slanju poruka se razdvaja u pojedine bajtove, bajtovi se formiraju u transportne segmente, segmenti se nakon pakiranja šalju od usmjerivača (engl. *router*) do usmjerivača u obliku paketa, a kada odredišna strana primi te pakete, redom ih raspakirava i dolazi do početnog podatka.

[8] https://hr.wikipedia.org/wiki/OSI_model

3.8.2. Socket

Kako bi dva računala mogla međusobno komunicirati preko mreže, potrebno je definirati komunikacijski kanal između njih. Postoje tako konekcijski orijentirani protokoli i beskonekcijski. Mrežni *socket* predstavlja programsku strukturu unutar mrežnog čvora računalne mreže koji služi kao kranja točka za slanje i primanje podataka preko mreže. Struktura i svojstva *socketa* definirana je korištenim API-em (drugim riječima, *socket* je programski definiran), no osnove funkcioniranja su iste u svim implementacijama. Također, *socketi* se kreiraju jedino tokom trajanja procesa neke aplikacije [21]. *Socket* je vrlo koristan koncept u interprocesnoj komunikaciji, što nije uvijek jednostavno za ostvariti. Kako bi dva procesa mogla međusobno razmjenjivati podatke, moraju podržavati *socket* sloj. Otvaranjem priključne točke, odnosno *socketa* na jednoj i drugoj strani otvara se i komunikacijski kanal putem kojega mogu razmjenjivati podatke. Pozitivna stvar kod *socketa* je i ta što nije bitno o kojim vrstama uređaja se radi, pa tako na jednom kraju može biti računalo, a na drugome hladnjak ili neki drugi kućanski aparat, sve dok taj uređaj podržava sloj *socket* tehnologije. Kako je navedeno, komunikacijski kanal može biti konekcijski ili beskonekcijski. U prvom slučaju, prilikom komunikacije, otvara se komunikacijski kanal, uspostavlja se veza i ta veza traje skroz tokom izmjene informacije, a nakon slanja podataka veza se zatvara. Konekcijski orijentirana komunikacija najčešće koristi TCP protokol. Kod beskonekcijske komunikacije, paketi s podatcima šalju se neovisno o uspostavljenoj vezi i na destinaciju mogu stići u nepravilnom redoslijedu. Pri tome se najčešće koristi UDP protokol. Ispravnu rekonstrukciju originalne poruke omogućavaju nam razne zastavice, oznake, identifikacijski brojevi i slično, kojima upravljaju niži slojevi.

3.8.3. Winsock

Prema [22] *Windows Sockets API* (WSA), skraćeno Winsock, računalna je specifikacija tehnologije koja definira pristup mrežnih aplikacija mrežnim uslugama, posebice TCP/IP stogu. *Windows Sockets API* pruža programerima set funkcija za izradu i razvoj poslužitelj/klijent aplikacija, odnosno omogućuje se programiranje u aplikacijskom sloju, dok se o nižim slojevima brine sama biblioteka. Winsock tehnologija je verzija *Berkeley socket* biblioteke prilagođena (engl. *Ported*) Windows operacijskom sustavu, tako da su neki argumenti funkcija ostali jedino radi kompatibilnosti. Npr. u funkciji *select(),* o kojoj će nešto više biti riječi kasnije, prvi argument koji

se predaje (cjelobrojni broj *nfds*) se ne koristi unutar same funkcije. Vrijedi također spomenuti da, iako su postojali određeni napori da se osigura prenosivost koda sa Windows OS-a na Unix i obrnuto, između ta dva OS-a postoje brojne fundamentalne različitosti u dizajnu, stoga danas aplikacije često neće raditi na oba sustava bez ikakvih promjena.

3.8.4. Metode za posluživanje više klijenata

U najjednostavnijim poslužiteljskim (engl. s*erver)* aplikacijama najčešće je podržano posluživanje samo jednog klijenta u nekom trenutku u vremenu. To bi značilo da poslužitelj, koji već pruža svoje usluge nekom klijentu, neće moći uspješno odgovoriti na zahtjeve novog klijenta. Njegovi resursi već su zauzeti. U tom slučaju posluživanje se vrši unutar procesa u jednoj glavnoj niti (engl. *thread*). Kako bi se većem broju klijenata omogućio istovremeni pristup resursima poslužitelja razvijene su određene metode za posluživanje.

Jedna od takvih metoda je i aplikacija s više niti. U tom slučaju se za svaki pristigli klijentski zahtjev pravi posebna nit unutar kojega se poslužitelj brine o posluživanju. Međutim, iako je za manji broj klijenata ova metoda izvediva pa čak i jednostavna, u slučaju kada poslužitelju pristupa puno veći broj klijenata nastaju problemi. Aplikaciju s mnogobrojnim nitima teško je razvijati, pratiti i naposljetku, otklanjati greške. Aplikacija se u tom slučaju često ponaša vrlo nepredvidivo, konstantno se prebacuje sa jednog dijela aplikacije na drugi te je programerima teško razvijati i otklanjati probleme u slučaju većeg i kompleksnijeg sustava. Zaključak je da je metoda s posluživanjem klijenata u posebnim nitima moguća za jednostavne aplikacije no teško ju je skalirati na veće sustave.

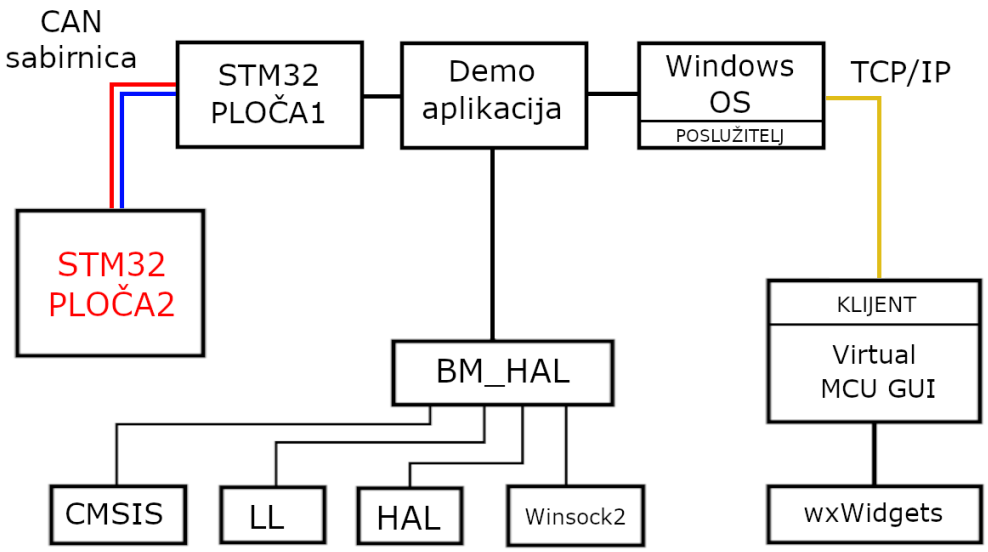
Druga metoda za posluživanje većeg broja klijenata je aplikacija koja koristi "*select()"* funkciju. Funkcija *select()* izvedena je iz BSD *socket* implementacije te implementirana unutar Winsock tehnologije. Ova metoda je atraktivna jer omogućava rad u jednoj glavnoj niti unutar procesa, ali u isto vrijeme praćenje svih pristiglih zahtjeva klijenata i mogućnost reakcije. Metoda pomoću *select()* funkcije djeluje kao programska implementacija prekida (engl. *interrupt*), odnosno tako se ponaša. Argumenti koji se predaju ovoj funkciji uključuju i listu *socket* deskriptora za slušanje te pisanje kao i varijablu koja označava vremenski period čekanja, tj. blokiranja. Lista *socketa* definira se *fdset* varijablom, odnosno strukturom koja sadrži listu *socketa* i veličinu te liste. Funkcija je blokirajuća

sve dok ne protekne vrijeme definirano zadnjim argumentom, ili beskonačno ako je taj argument jednak “NULL” pokazivaču. *Select()* tokom tog vremena nadgleda *socket*e definirane predanim parametrima te prilikom aktivacije pojedinog *socket*a vraća one *socket* deskriptore na kojima su detektirani zahtjevi. Aplikacija, odnosno poslužitelj može zatim poslužiti aktivirane *sockete*, a pri tome su implementirane i funkcije za rad s listama *socket* deskriptora:

- FD_ZERO(fd_set*) – brisanje liste,
- FD_SET(SOCKET, fd_set*) - upisuje *socket* u listu,
- FD_CLR(SOCKET, fd_set*) – briše *socket* iz liste,
- FD_ISSET(SOCKET, fd_set*) – provjerava stanje *socketa*.

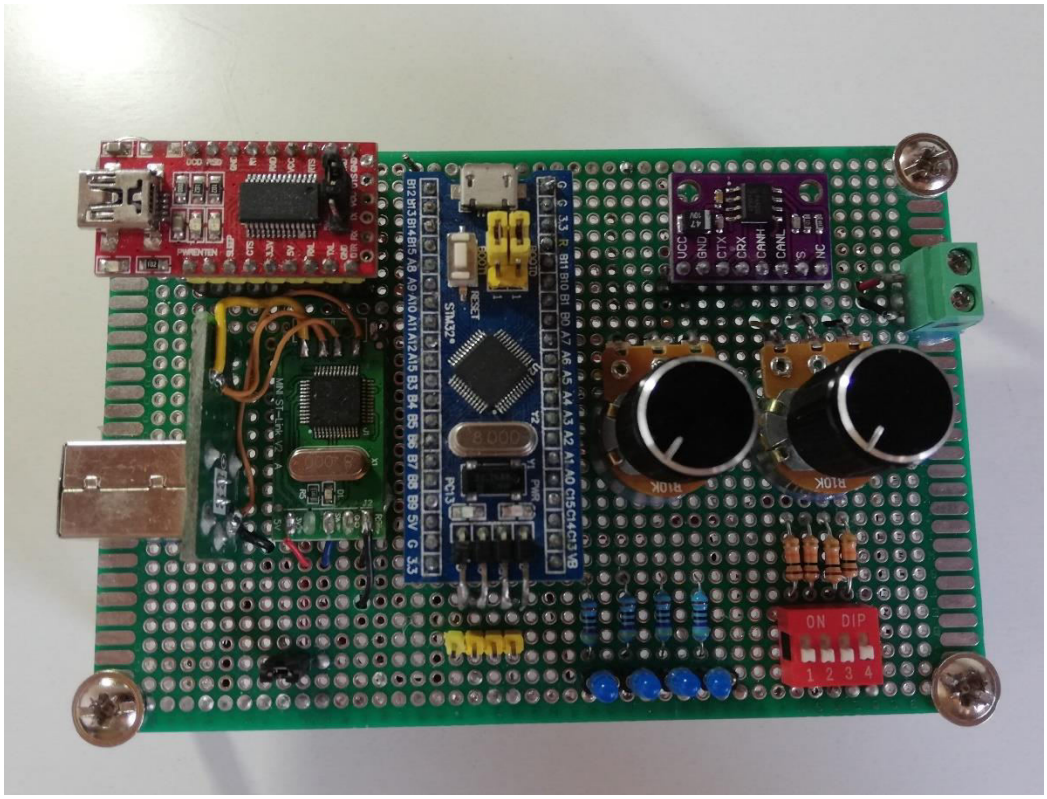
4. IZVEDBA SLOJA ZA APSTRAKCIJU MIKROKONTROLERA

Na slici 4.1 prikazan je cjelokupan sustav za demonstraciju apstrakcije mikrokontrolera, odnosno svi gradivni blokovi razvijeni ili korišteni kao predmet ovog rada. Središnji dio i predmet ovoga rada upravo je "BM_HAL" blok. „BM_HAL" blok predstavlja upravljačke programe (engl. *Drivers*) koji omogućavaju apstrakciju sklopovlja na način da koriste funkcionalnosti nižih slojeva. Tako se na taj blok vežu upravljački programi važni za STM32 mikrokontroler, a to su: CMSIS (engl. *Cortex Microcontroller Software Interface Standard*), LL (engl. *Low Level*) te HAL (engl. *Hardware Abstraction Layer*) upravljački programi. Bitno je napomenuti da je sav programski kod za STM32 mogao biti realiziran i bez "LL" upravljačkih programa, no nalazi se tu jednostavno zbog eksperimentiranja sa tim slojem te je korišten za jednostavnije funkcionalnosti poput funkcija koje se odnose na GPIO te ADC. HAL sloj nudi veću razinu apstrakcije i jednostavniji programski kod, no pomoću LL sloja kod je efikasniji i brži. Također osim blokova koji se odnose na STM32, na BM_HAL sloj se vežu i upravljački programi zaduženi za upravljanje aplikacije namijenjene za virtualni prostor, tj. aplikacije koja bi simulirala ponašanje mikrokontrolera na Windows operacijskom sustavu. Taj sloj naziva se Winsock2 (broj 2 u nazivu označava verziju 2 implementacije winsock tehnologije) sloj koji omogućava programiranje poslužitelj/klijent aplikacija u C programskom jeziku. U ovom slučaju Winsock2 tehnologija koristi se za realizaciju poslužiteljske aplikacije (Windows Demo App), koja zapravo predstavlja sami mikrokontroler. To znači da se vanjski događaji, poput ulaznog sučelja, analognog sučelja i slično fizičkog mikrokontrolera, mogu simulirati putem klijentskih zahtjeva koji postavljaju stanja ulaza/izlaza, CAN komunikacijske sabirnice itd. Takva jedna klijentska aplikacija prikazana je na slici 4.1. pod blokom "Virtual MCU GUI", gdje je također prikazano da je ta aplikacija realizirana pomoću wxWidgets tehnologije. Na slici 4.14. također je prikazan stvarni izgled klijentske aplikacije, koja omogućava komunikaciju sa poslužiteljem, odnosno virtualnim mikrokontrolerom i generira vanjske događaje.

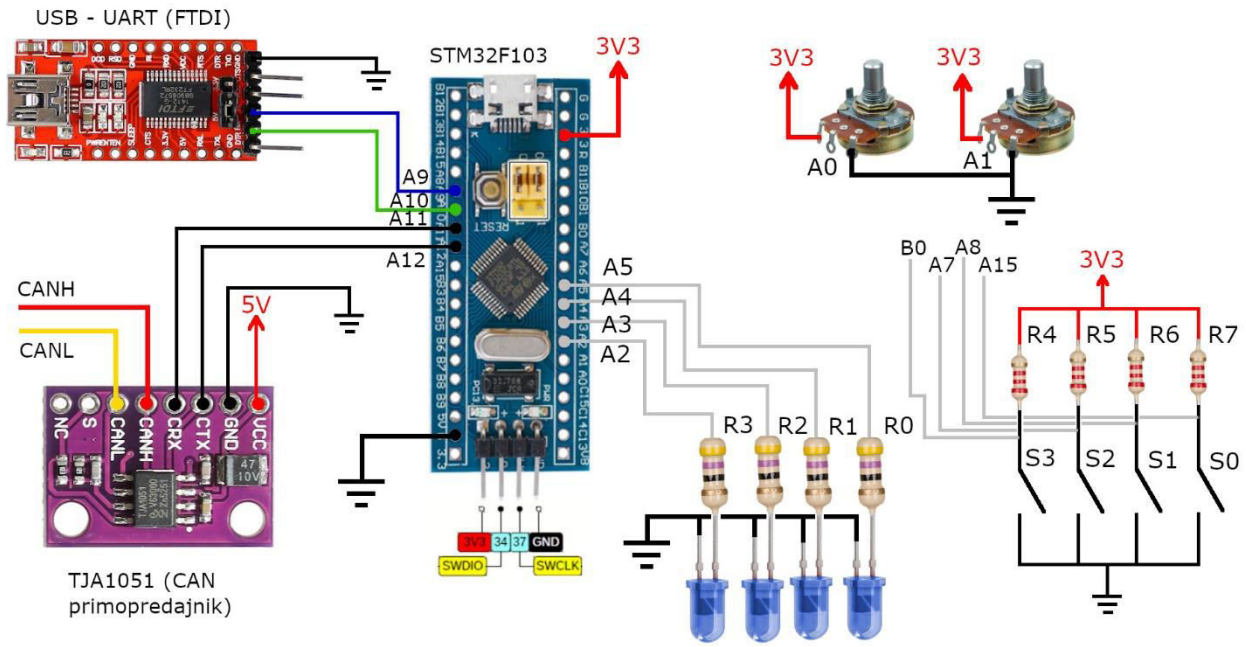


Slika 4.1. *Blokovski prikaz sustava za demonstraciju apstrakcijskog sloja*

STM32 ploča 1 odnosi se na glavnu ploču (vidjeti sliku 4.2.) isprogramiranu korištenjem BM_HAL upravljačkih programa. Služi za demonstraciju BM_HAL sloja, tj. svih implementiranih funkcionalnosti. Na njoj se nalaze mehanički prekidači povezani na ulazne digitalne pinove STM32 mikrokontrolera, LE-diode povezane na izlazne digitalne pinove, dva potenciometra povezana na dva kanala A/D pretvornika, CAN primopredajnik, UART modul koji omogućava serijsku komunikaciju sa računalom te ST-Link modul potreban za programiranje mikrokontrolera. Kasnije je ista demonstracijska aplikacija kao i za STM32 mikrokontroler pokrenuta i na Windows OS-u, gdje se zapravo pokazuje apstrakcija koju omogućava BM_HAL sloj. Drugim riječima, programer ima mogućnost razvijati programski kod BM_HAL slojem bez da razmišlja o fizičkom sklopovlju, a zatim tu aplikaciju može izvršiti na sklopovlju ili u virtualnom okruženju, odnosno na STM32 mikrokontroleru ili na Windows operacijskom sustavu. Ovime se postiže prenosivost programskog koda.



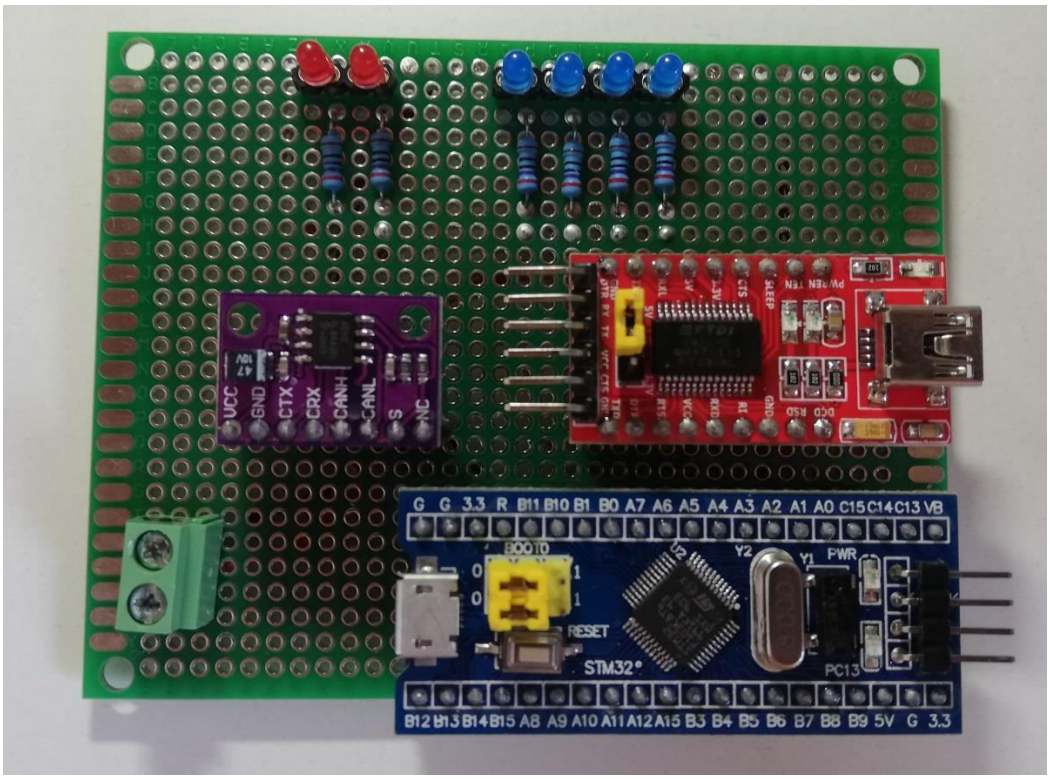
Slika 4.2. *STM32 ploča 1*



Slika 4.3. Spojna shema sustava ploče 1[9]

Ploča 2 (slika 4.4.) svojevrsna je dopunska ploča napravljena za potrebe demonstracije u slučaju fizičkog okruženja. Na njoj se također nalaze LE-diode za vizualizaciju stanja na glavnoj ploči, te se postavljaju na logičku jedinicu ili nulu, ovisno o primljenim CAN porukama. Tu su dakle još i CAN primopredajnik kao i UART modul za serijsku komunikaciju. Četiri plave LE-diode svojevrsna je kopija LE-dioda sa ploče 1, dok su pored njih također postavljene i dvije crvene LE-diode na kojima se vrši vizualizacija stanja potenciometara sa ploče 1. Aplikacija isprogramirana za ploču 2 ne koristi BM_HAL sloj, već direktno HAL sloj, budući da je sve potrebno demonstrirano na ploči 1, a osim toga za paljenje crvenih LE-dioda koristi se i funkcionalnost vremenskog brojača (engl. *Timer*) koji trenutno nije podržan BM_HAL slojem.

[9] Slike su preuzete sa sljedećih web stranica:
https://predictabledesigns.com/introduction-stm32-blue-pill-stm32duino/
https://store.roboticsbd.com/robotics-parts/658-ftdi-usb-to-ttl-serial-converter-adapter-ft232rl-in-bangladesh.html
https://pmdway.com/products/tja1051-high-speed-low-voltage-can-transceiver-board
https://visualled.com/en/glossary/led-dip/
https://en.wikipedia.org/wiki/Resistor
https://en.wikipedia.org/wiki/Potentiometer

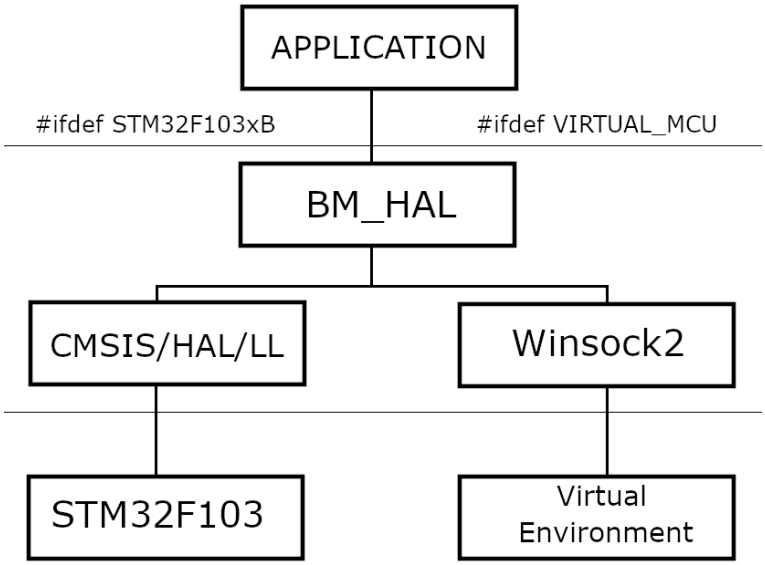


Slika 4.4. *STM32 ploča 2*

4.1. BM_HAL

Ideja BM_HAL sloja je da bude samostalan sloj upravljačkih programa pomoću kojega programer ima mogućnost razvoja i rada u aplikacijskom sloju neovisno o kojem se fizičkom uređaju, odnosno sklopovlju radi. Trenutno je podržan rad sa STM32F103C8 mikrokontrolerom te virtualnom sustavu Windows OS-a. Slikovito bi arhitektura izgledala kao na slici 4.5. Također djelomičan kod, uglavnom za inicijalizaciju periferije u slučaju STM32 mikrokontrolera, generiran je pomoću alata "CubeMX" kojega je razvila tvrtka STMicroelectronics. Trenutno je BM_HAL slojem podržan rad nekoliko tipova perifernih uređaja:

- Digitalni ulaz,
- Digitalni izlaz,
- Analogni ulaz,
- UART,
- CAN.



Slika 4.5. *Slojeviti prikaz ideje BM_HAL upravljačkih programa*

Na slici 4.5. prikazano je i na koji način se postiže apstrakcija. Programer je jedino dužan definirati za koji sustav planira prevesti aplikaciju i to pomoću pretprocesorskih naredbi. Primjerice, #define STM32F103xB ako je aplikacija namijenjena izvršavanju na STM32F103 mikrokontroleru ili #define VIRTUAL_MCU ukoliko je aplikacija namijenjena izvršavanju na Windows OS-u, u virtualnom okruženju. U ovom radu korišten je popularni STM32F103C8 mikrokontroler te je odgovor na pitanje zašto se onda definira simbol s nastavkom „xB“ umjesto „C8“ moguće pronaći u CMSIS upravljačkim programima. Na slici 4.6. prikazan je dio „stm32f1xx.h“ zaglavne datoteke iz CMSIS sloja. Vidljivo je da ukoliko se definira simbol „STM32F103xB“ koristit će se apstrakcija za STM32F103C8 mikrokontroler, ali i za ostale srodne mikrokontrolere iz te porodice poput STM32F103R8, STM32F103T8, itd.

```
stm32f1xx.h
60 #if !defined (STM32F100xB) && !defined (STM32F100xE) && !defined (STM32F101x6) && \
61     !defined (STM32F101xB) && !defined (STM32F101xE) && !defined (STM32F101xG) && !defined (STM32F10
62     !defined (STM32F103xB) && !defined (STM32F103xE) && !defined (STM32F103xG) && !defined (STM32F10
63   /* #define STM32F100xB  */   /*!< STM32F100C4, STM32F100R4, STM32F100C6, STM32F100R6, STM32F100C8,
64   /* #define STM32F100xE */    /*!< STM32F100RC, STM32F100VC, STM32F100ZC, STM32F100RD, STM32F100VD,
65   /* #define STM32F101x6  */   /*!< STM32F101C4, STM32F101R4, STM32F101T4, STM32F101C6, STM32F101R6
66   /* #define STM32F101xB  */   /*!< STM32F101C8, STM32F101R8, STM32F101T8, STM32F101V8, STM32F101CB,
67   /* #define STM32F101xE */    /*!< STM32F101RC, STM32F101VC, STM32F101ZC, STM32F101RD, STM32F101VD,
68   /* #define STM32F101xG  */   /*!< STM32F101RF, STM32F101VF, STM32F101ZF, STM32F101RG, STM32F101VG
69   /* #define STM32F102x6 */    /*!< STM32F102C4, STM32F102R4, STM32F102C6 and STM32F102R6 */
70   /* #define STM32F102xB  */   /*!< STM32F102C8, STM32F102R8, STM32F102CB and STM32F102RB */
71   /* #define STM32F103x6  */   /*!< STM32F103C4, STM32F103R4, STM32F103T4, STM32F103C6, STM32F103R6
72   /* #define STM32F103xB  */   /*!< STM32F103C8, STM32F103R8, STM32F103T8, STM32F103V8, STM32F103CB,
73   /* #define STM32F103xE */    /*!< STM32F103RC, STM32F103VC, STM32F103ZC, STM32F103RD, STM32F103VD,
74   /* #define STM32F103xG  */   /*!< STM32F103RF, STM32F103VF, STM32F103ZF, STM32F103RG, STM32F103VG
75   /* #define STM32F105xC */    /*!< STM32F105R8, STM32F105V8, STM32F105RB, STM32F105VB, STM32F105RC
76   /* #define STM32F107xC  */   /*!< STM32F107RB, STM32F107VB, STM32F107RC and STM32F107VC */
77 #endif
```

Slika 4.6. *definiranje odredišnog uređaja*

Nekoliko linija ispod u istoj datoteci zaglavlja vidljivo je i samo uključivanje različitih datoteka ovisno o prethodno definiranom simbolu (slika 4.7.). Naredba „#include „stm32f103xb.h““ uključuje datoteku koja definira sve adrese i registre periferija unutar STM32F103C8 mikrokontrolera, koji su dakako svi memorijski mapirani te je definiranje simbola „STM32F103xB“ stoga neizostavno u slučaju korištenja HAL i/ili LL upravljačkih programa pri programiranju STM32 mikrokontrolera. Treba još napomenuti i kako se navedeni simbol, za uspješno prevođenje aplikacije može, ili otkomentirati u datoteci prikazanoj na slici 4.6., ili definirati u nekoj drugoj datoteci na vidljivom mjestu ili se može definirati u „*Makefile*“ datoteci.

```
stm32f1xx.h
112 #if defined(STM32F100xB)
113   #include "stm32f100xb.h"
114 #elif defined(STM32F100xE)
115   #include "stm32f100xe.h"
116 #elif defined(STM32F101x6)
117   #include "stm32f101x6.h"
118 #elif defined(STM32F101xB)
119   #include "stm32f101xb.h"
120 #elif defined(STM32F101xE)
121   #include "stm32f101xe.h"
122 #elif defined(STM32F101xG)
123   #include "stm32f101xg.h"
124 #elif defined(STM32F102x6)
125   #include "stm32f102x6.h"
126 #elif defined(STM32F102xB)
127   #include "stm32f102xb.h"
128 #elif defined(STM32F103x6)
129   #include "stm32f103x6.h"
130 #elif defined(STM32F103xB)
131   #include "stm32f103xb.h"
132 #elif defined(STM32F103xE)
133   #include "stm32f103xe.h"
134 #elif defined(STM32F103xG)
135   #include "stm32f103xg.h"
136 #elif defined(STM32F105xC)
137   #include "stm32f105xc.h"
```

Slika 4.7. *Uključivanje datoteke koja definira memorijsku mapu STM32F103C8 periferije*

Budući da se navedeni simbol već koristi u CMSIS upravljačkom sloju, isti je korišten i za razlučivanje funkcioniranja BM_HAL upravljačkih programa. Ako se pogleda izvorni kod BM_HAL sloja, vidljivo je da će se funkcije drugačije ponašati ovisno o definiranom simbolu. U slučaju STM32 mikrokontrolera funkcije će tako direktno manipulirati registrima STM32 mikrokontrolera koristeći neke od STM-ovih upravljačkih programa poput HAL i LL, dok će definiranjem simbola „VIRTUAL_MCU“ funkcije zapravo činiti proširenje aplikacijskog sloja. Ako se promotri primjer „GPIO“ upravljačkog programa na slici 4.8., moguće je na konkretnom primjeru to i vidjeti.

```
bm_hal_gpio.c
139 void BM_HAL_GPIO_digitalWrite(GPIO_TypeDef *GPIOx, uint32_t pin_mask,
140         BM_HAL_GPIO_pin_state_t pin_state)
141 {
142 #ifdef STM32F103xB
143     if (pin_state != BM_GPIO_LOW)
144     {
145         LL_GPIO_SetOutputPin(GPIOx, pin_mask);
146     }
147     else
148     {
149         LL_GPIO_ResetOutputPin(GPIOx, pin_mask);
150     }
151 #elif defined(VIRTUAL_MCU)
152     if (BM_GPIO_LOW == outputPort[pin_mask] && BM_GPIO_HIGH == pin_state)
153     {
154         outputPort[pin_mask] = BM_GPIO_HIGH;
155     }
156     else if(BM_GPIO_HIGH == outputPort[pin_mask] && BM_GPIO_LOW == pin_state)
157     {
158         outputPort[pin_mask] = BM_GPIO_LOW;
159     }
160
161 #endif
162 }
```

Slika 4.8. *BM_HAL_GPIO_digitalWrite() funkcija*

Ako je funkcija namijenjena za izvođenje na stvarnom mikrokontroleru te je definiran simbol „STM32F103xB“ funkcija za pisanje digitalnog porta „BM_HAL_GPIO_digitalWrite(…)“ koristit će niži „LL“ sloj za upravljanje GPIO portom te će na mapiranoj adresi podesiti određeni registar, odnosno određeni bit, na vrijednost 0 ili 1. Posljedica navedenog biti će i vidljiva promjena stanja izlaza na tom pinu mikrokontrolera, koju je moguće i vizualizirati pomoću, primjerice, LE-diode. Međutim, ukoliko je aplikacija namijenjena izvršavanju u virtualnom okruženju, tj. na Windows operacijskom sustavu te je analogno tome definiran simbol „VIRTUAL_MCU“, upravljački program iz BM_HAL sloja će zapravo postavljati određene varijable, u ovom slučaju elemente iz polja cjelobrojnih brojeva „outputPort“ koji simuliraju stanja pinova. Navedeno polje je također inicijalizirano kao globalna varijabla, tako da se može reći da BM_HAL sloj u tom slučaju čini proširenje aplikacijskog sloja, a ne *firmware*-a koji manipulira registrima kao u slučaju STM32 mikrokontrolera.

Jedna od mana BM_HAL sloja i prostor za napredak je u relativno nefleksibilnoj inicijalizaciji sustava. To znači da je dodano sklopovlje poput LE-dioda, potenciometara i ostalog predefinirano na određene pinove mikrokontrolera te za sad ne postoji mogućnost promjene i drugačija definicija što

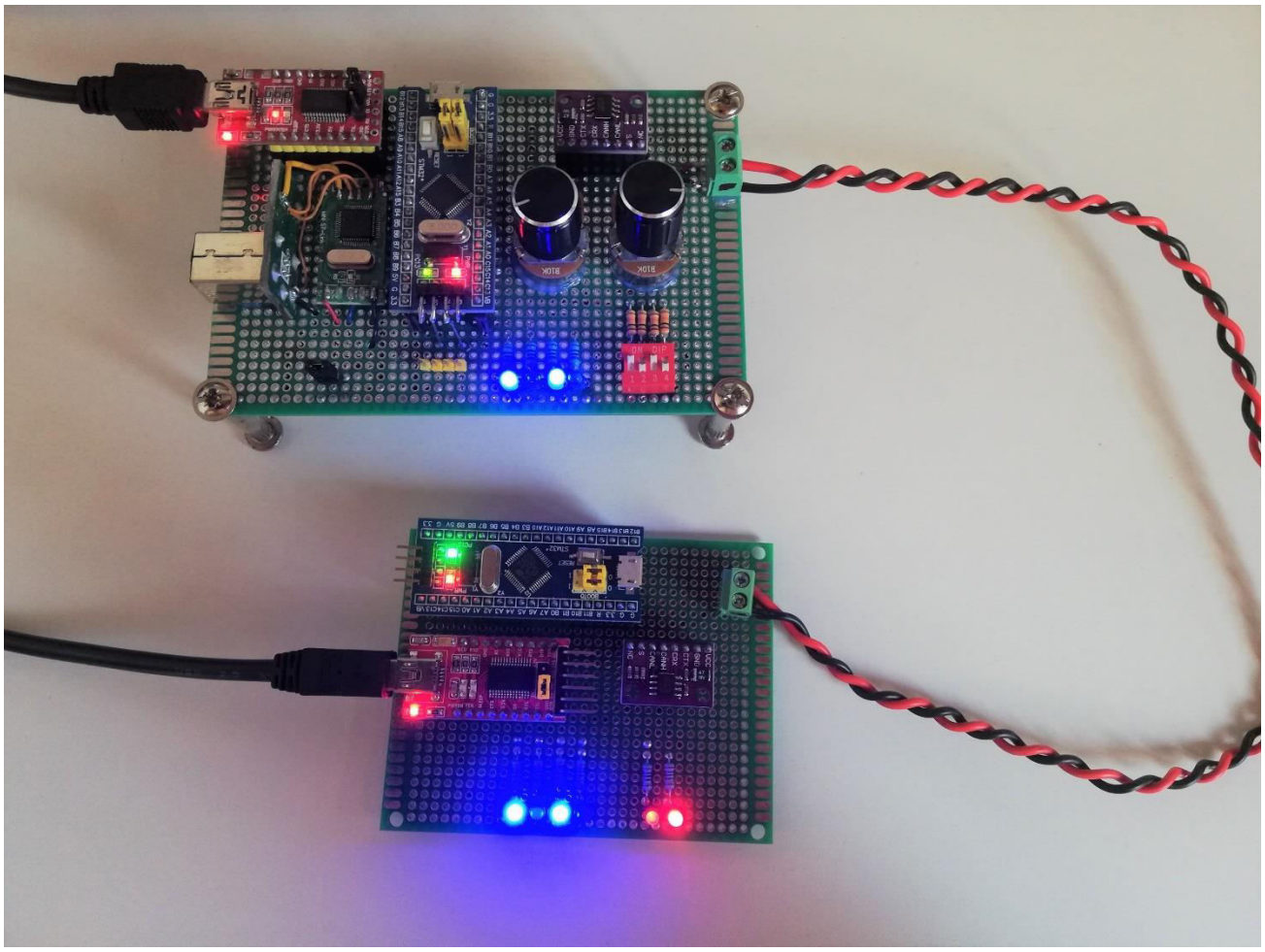
se tiče inicijalizacije, osim promjene izvornog koda. Drugim riječima nisu implementirane funkcije BM_HAL sloja koje mogu definirati ulaze i izlaze prema želji korisnika putem nekih konfiguracijskih struktura i slično.

5.2. STM32 demonstracijska aplikacija

Za potrebe demonstracije rada BM_HAL sloja isprogramirana je odgovarajuća aplikacija[10] s ciljem da se prikažu sve funkcionalnosti tog sloja, a trenutno podržane su:

- čitanje i pisanje digitalnog porta (4 ulaza, 4 izlaza),
- čitanje analognog porta (2 ulaza),
- pisanje preko UART protokola,
- čitanje i pisanje na CAN sabirnicu.

[10] Demonstracijsku aplikaciju moguće je pronaći na github platformi:
https://github.com/bmihelcic/Diplomski/blob/master/STM32_Demo/Src/main.c



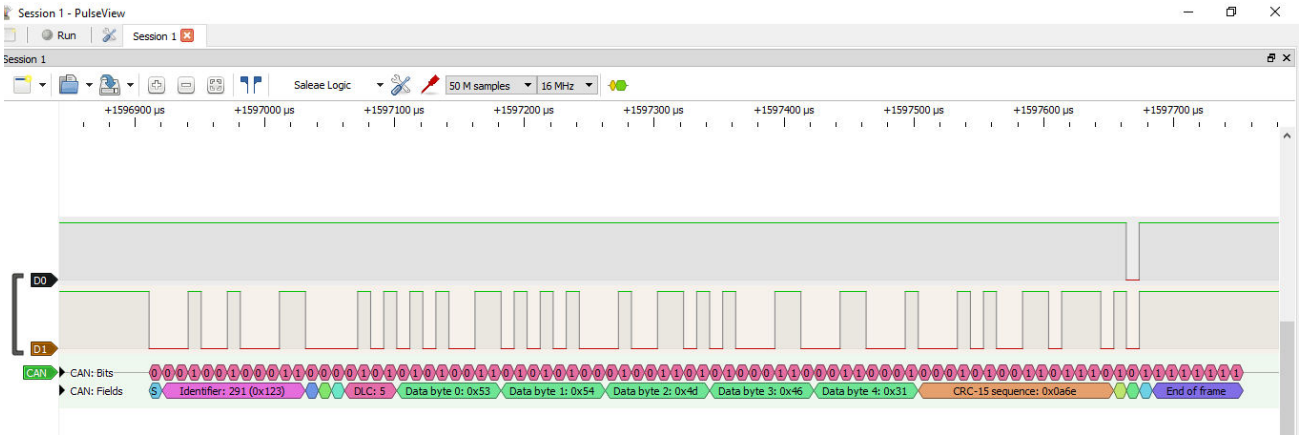
Slika 4.9. *Demonstracija rada STM32 mikrokontrolera*

S ciljem da se pokaže funkcionalnost upravljačkih programa za digitalni ulaz/izlaz, aplikacija redom čita svaki digitalni ulaz te se u ovisnosti o stanju pojedinog ulaza odgovarajući izlaz postavlja na 0 ili 1. Ulaz 0 tako odgovara izlazu 0 i tako redom (0-3). Sklopovlje digitalnog ulaza je izvedeno na način da su svi ulazi uobičajeno na naponu od 3,3V, a kada je detektiran napon na 0V smatra se da je ulaz "uključen", odnosno na logičkoj jedinici. Aplikacija će tada uključiti odgovarajuću LE-diodu, tako da je funkcionalnost pisanja digitalnog izlaza moguće vizualizirati.

Što se tiče analognog ulaza, na ploči 1 nije moguće direktno vidjeti stanje tog analognog kanala, međutim pročitane vrijednosti šalju se preko CAN sabirnice u obliku poruka te se vizualizacija odvija na drugoj strani. Ploča 2 primit će te poruke, iščitati vrijednosti te kontrolirati dvije crvene LE-diode prema primljenim vrijednostima. Time je zapravo pokazan ispravan način rada i čitanja analognog ulaza, ali i pisanja na CAN sabirnicu. U ovom slučaju upravlja se faktorom vođenja dvaju

PWM (engl. *Pulse Width Modulation*) signala. Jedan od problema koji je bilo potrebno riješiti je i taj što se CAN protokolom šalju poruke u obliku pojedinih bajtova, dok je rezolucija ADC modula kod STM32 mikrokontrolera 12 bita, a to je više od jednog bajta. Pročitanih 12 bita potrebno je tako bilo razdvojiti na dva dijela. Prvih 8 bitova spremljeno je u jedan bajt poruke, dok je preostala 4 bita spremljeno u drugi bajt poruke. Na prijemnoj strani, zatim se obrnutom operacijom pomicanja bitova rekonstruirala izvorna vrijednost te shodno tomu postavlja se postotak faktora vođenja što na posljetku rezultira različitim intenzitetima svjetline LE-dioda.

Osim slanja analognih vrijednosti, U jednoj CAN poruci šalje se također i varijabla koja označava stanje digitalnih ulaza, a svaki ulaz zamišljen je kao 1 bit. Vrijednost ulaza zapravo je na taj način jedan 4-bitan broj. Na slici 4.10. moguće je vidjeti kako izgleda jedan cijeli CAN okvir. Osim CAN identifikacijskog broja, vide se i pet podatkovnih bajtova (zelena boja). Bajt 0 nosi informaciju o stanju ulaza, dok preostali bajtovi (po dva) označavaju stanja dva analogna ulaza.



Slika 4.10. *Izgled jednog CAN okvira*

Isto tako, u aplikaciji je isprogramirano da se podatci iz CAN poruke, pošalju i na serijsku vezu do računala UART protokolom. To je moguće vidjeti na slici 4.11. Na slici 4.12. također je prikazan ispis od strane STM32 mikrokontrolera na ploči 2. Na taj način moguće je potvrditi ispravan primitak CAN poruka i samih vrijednosti varijabli.

```
COM4
Send
INPUT: 10  ADC0: 88  ADC1: 3311

CAN TX MSG: 0x125 : 0xa 0x58 0x0 0xf0 0xc
INPUT: 10  ADC0: 88  ADC1: 3312

CAN TX MSG: 0x125 : 0xa 0x59 0x0 0xf1 0xc
INPUT: 10  ADC0: 89  ADC1: 3313

CAN TX MSG: 0x125 : 0xa 0x58 0x0 0xf0 0xc
INPUT: 10  ADC0: 88  ADC1: 3312

CAN TX MSG: 0x125 : 0xa 0x59 0x0 0xef 0xc
INPUT: 10  ADC0: 89  ADC1: 3311

CAN TX MSG: 0x125 : 0xa 0x58 0x0 0xf0 0xc
INPUT: 10  ADC0: 88  ADC1: 3312
Autoscroll  Show timestamp  Newline  115200 baud  Clear output
```

Slika 4.11. *Ispis serijskog porta pri slanju Ploče 1*

```
COM5
Send
CAN MSG RECEIVED: 0x125 : 0xa 0x58 0x0 0xf0 0xc
LED: 10 ADC0: 88 ADC1: 3312

CAN MSG RECEIVED: 0x125 : 0xa 0x59 0x0 0xf1 0xc
LED: 10 ADC0: 89 ADC1: 3313

CAN MSG RECEIVED: 0x125 : 0xa 0x58 0x0 0xf0 0xc
LED: 10 ADC0: 88 ADC1: 3312

CAN MSG RECEIVED: 0x125 : 0xa 0x59 0x0 0xef 0xc
LED: 10 ADC0: 89 ADC1: 3311

CAN MSG RECEIVED: 0x125 : 0xa 0x58 0x0 0xf0 0xc
LED: 10 ADC0: 88 ADC1: 3312
Autoscroll  Show timestamp  Newline  115200 baud  Clear output
```

Slika 4.12. *Ispis serijskog porta Ploče 2*

5.3. Windows aplikacija

Kao što je rečeno u prethodnim poglavljima, aplikacija isprogramirana BM_HAL slojem može se izvoditi i na Windows operacijskom sustavu. To je postignuto pomoću funkcionalnosti Winsock tehnologije. U datoteci “bm_hal.c” pri inicijalizaciji, otvara se TCP “*socket”* na portu 8080. Taj prvi nastali *socket* služi za slušanje zahtjeva na portu 8080 na koji se klijenti mogu spojiti te ako je taj

socket aktivan znači da je poslužitelju pristigao novi zahtjev za uspostavom veze, tj. novi klijent. Pri inicijalizaciji također se kreiraju i dvije niti (engl. *Thread*), od kojih je jedna za slušanje zahtjeva i pozivanje odgovarajućih funkcija za odrađivanje potrebnih radnji, poput čitanja primljenih poruka ili bilježenja nove veze. Druga nit odnosi se na slanje poruka i unutar nje se simulira slanje CAN poruke. U ovom slučaju radi se o TCP porukama odnosno poruke se šalju u obliku ASCII znakova. Klijentska aplikacija te znakove prima, parsira i ponaša se u skladu sa vrijednostima varijabli. Tako se poslužitelj aplikacija u ovom slučaju može zamisliti kao STM32 na ploči 1, dok se klijentska aplikacija (Virtual MCU GUI) ponaša kao STM32 na ploči 2. Na slici 4.13. vidljiv je ispis tijekom rada Windows poslužiteljske aplikacije. Vrijednosti zapisane u varijablama koje predstavljaju digitalni izlaz te analogni ulaz, šalju se klijentskoj aplikaciji „Virtual MCU GUI“, koja će biti opisana u sljedećem poglavlju. Također, u ispisu su vidljiva dva pristupa poslužitelju. Poslužitelj bilježi *socket* deskriptor svih povezanih klijenata, tako da može posluživati i više klijenata u isto vrijeme.

```
D:\Backup\Projects\GIT_local\Diplomski\Executables\Windows_Demo.exe
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
Host disconnected
New connection, new socket descriptor is: 348, ip is: 127.0.0.1, port: 57145
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
< sending: 0x125: 0x1 0x98 0x9 0xcd 0x4
> received: SET ADC0 2047
> received: SET ADC1 2047
> received: SET INPUT0 HIGH
> received: SET INPUT1 HIGH
> received: SET INPUT2 HIGH
> received: SET INPUT3 HIGH
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
> received: SET INPUT1 LOW
< sending: 0x125: 0x0 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
Host disconnected
New connection, new socket descriptor is: 352, ip is: 127.0.0.1, port: 57146
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
> received: SET INPUT1 LOW
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
< sending: 0x125: 0x2 0xff 0x7 0xff 0x7
```

Slika 4.13. *Ispis Windows demo aplikacije*

5.4. Virtual MCU GUI

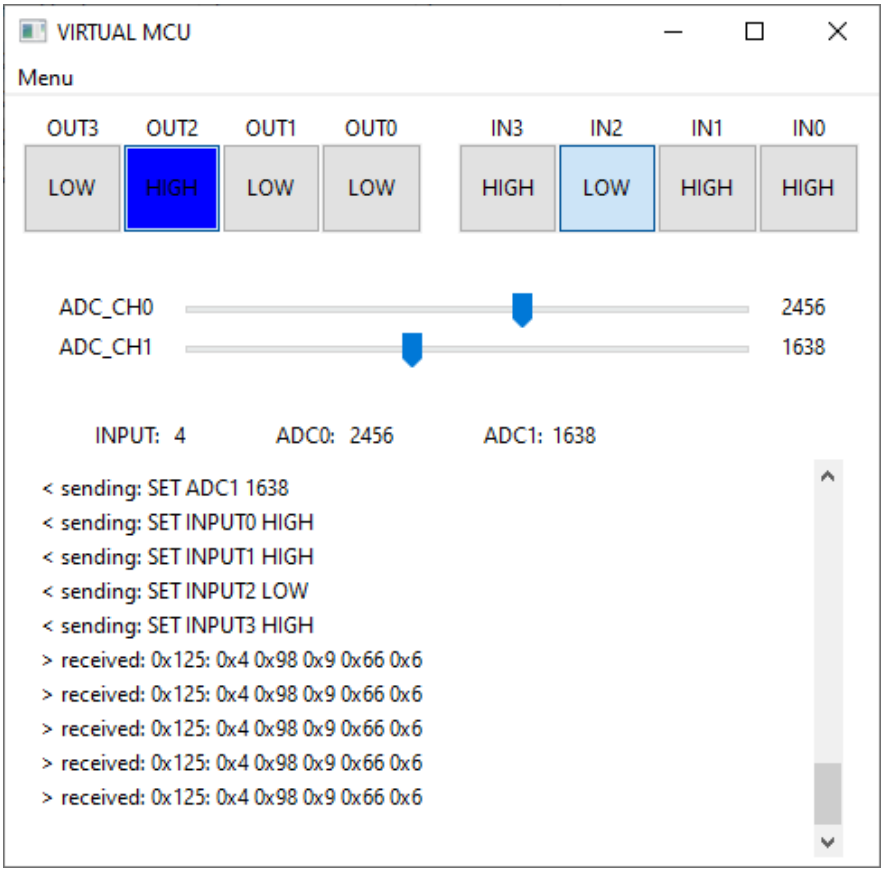
Budući da u virtualnom okruženju ne postoji fizičko sklopovlje koje se može vanjskim utjecajima i događajima mijenjati, napravljena je jedna GUI (engl. *Graphical User Interface*) aplikacija isprogramirana korištenjem wxWidgets tehnologije (C++ programski jezik). Takva aplikacija omogućava manipuliranje određenim varijablama te služi poput simulacije okruženja i vanjskih signala kao što je to slučaj u HIL metodi testiranja. Na slici 4.14. vidimo kako izgleda jedna takva aplikacija te što ona sadržava. U gornjem dijelu nalaze se tipke koje simuliraju GPIO port na mikrokontroleru. Dodano je 8 tipki, 4 koje predstavljaju izlazno sučelje i 4 koje predstavljaju ulazno sučelje. Izlazne tipke predstavljaju zapravo LE-diode koje su u stvarnosti povezane na izlazne pinove STM32 mikrokontrolera te one promjenom stanja mijenjaju boju kao i u stvarnosti. Klikom korisnika na te tipke, stanje se neće promijeniti, jer ni u stvarnosti se ne može izvana direktno promijeniti stanja izlaza, no ti izlazi mijenjat će svoja stanja u ovisnosti o primljenim porukama poslanim od strane poslužiteljske aplikacije. Te poruke simuliraju poruke na CAN sabirnici u stvarnom okruženju. Na slici 4.14. vidljiv je i primjer jedne takve poruke "> received: 0x125: 0x4 0x98 0x9 0x66 0x6". Ta poruka označava slanje CAN ID-a na prvom mjestu u poruci, zatim 5 bajta poruke koji odgovaraju vrijednostima varijabli sa poslužitelja.

U srednjem dijelu GUI aplikacije nalaze se također dva kliznika (engl. *slider*) koji predstavljaju zapravo potenciometar u stvarnom okruženju te čitanje ADC kanala. Vrijednosti je moguće postavljati od 0 do 4095 koji odgovaraju 12-bitnoj razlučivosti ADC modula na STM32 mikrokontroleru.

Ispod kliznika nalaze se tri tekstualne oznake koje se mijenjaju prema primljenim porukama sa poslužitelja.

Dakle, korisnik ima mogućnost promjene stanja četiri ulazne tipke te dva kliznika, nakon čega se (ako je aplikacija povezana sa poslužiteljem) šalju poruke koje mijenjaju varijable na poslužitelju, a poslužitelj zatim povratnim porukama, manipulira bojom četiri izlazne tipke i vrijednostima pored oznaka "INPUT:", "ADC0:" te "ADC1:". Sve izlazne i ulazne poruke ispisuju se u jednom prozoru te je iste moguće vidjeti na slici 4.14. Neke od poruka koje se šalju poslužitelju tako imaju oblik "SET ADC1 1638" što zapravo simulira okretanje potenciometra, ili na primjer "SET INPUT0 HIGH" što bi u stvarnom okruženju bilo jednako postavljaju ulaznog pina na +3,3V.

Također je bitno napomenuti da se navedene poruke šalju poslužitelju preko TCP/IP veze i to u obliku ASCII znakova. Da bi se TCP veza uopće mogla povezati sa poslužiteljem, izrađena je tipka “Connect” koja se prikazuje klikom na tipku “Menu”. Pored nje nalaze se također i tipke “Refresh” te “Close connection” koje su omogućene tek nakon uspješnog spajanja veze sa poslužiteljem. Tipka “Refresh” služi kako bi se sinkronizirala stanja varijabli na poslužitelju sa varijablama kod klijenta i to na način da klijent od jednom pošalje sve naredbe koje postavljaju određene vrijednosti poslužitelju. To je moguće i vidjeti na slici 4.14. u prvih pet redova ispisa (poruke koje počinju sa “< sending..”). Ova funkcionalnost je omogućena jer na poslužitelju mogu ostati zastarjele vrijednosti varijabli koje je podešavao neki drugi klijent. Na taj način nije potrebno ručno postavljati sve varijable, odnosno slati pojedinačne zahtjeve. Tipka “Close connection” radi upravo kao što i sami naziv govori. Klikom na tu tipku postojeća veza između klijenta i poslužitelja, koja je cijelo vrijeme otvorena tokom rada aplikacije, biti će zatvorena.



Slika 4.14. *Klijent aplikacija (VIRTUAL MCU GUI)*

ZAKLJUČAK

Budući da je pravovremena dostava kvalitetnih proizvoda danas jedan od najbitnijih faktora u bilo kojoj industriji potrebno je osmisliti proces koji će uspješno unaprijediti sve moguće aspekte djelovanja. Tom procesu uvelike pomažu alati za efikasno i ekonomično testiranje, stoga su apstrakcijski sustavi jedan od najpopularnijih opcija u fazi testiranja. Pomoću HIL metode testiranja, moguće je određen podsustav testirati u izoliranom okruženju, gdje se određeni signali simuliraju programski te se na taj način verificira rad promatranog podsustava. Može se tako testirati samo jedan ECU iz bilo kojeg automobilskog podsustava (upravljanje kočionim sustavom, motorom, zračnim jastucima, klimom, …), a preostale signale iz okoline simulirati (npr. CAN poruke drugih ECU-ova).

Po sličnom tom principu razvijan je i apstrakcijski sloj u ovom radu, naziva „BM_HAL“. Ovaj sloj kombinacija je apstrakcijskog sloja „HAL“ koju je razvila tvrtka STMicroelectronics te tehnologije Winsock. Winsock tehnologijom omogućeno je proširenje HAL apstrakcijskog sloja za STM32F103 mikrokontroler i na virtualno okruženje. Pokazalo se kako je istu aplikaciju, razvijenu pomoću BM_HAL sloja, bez bilo kakvih promjena moguće izvršavati na dva različita sustava. Jednom kada razvojni programer napiše željenu aplikaciju, moguće je uz minimalne promjene postavki pri prevođenju izvornog koda u izvršnu datoteku, pokrenuti istu aplikaciju na fizičkom mikrokontroleru ili u virtualnom okruženju.

Ovakva vrsta simulacije mikrokontrolera, može poslužiti za obuku programera o podržanom mikrokontroleru i pripadajućim upravljačkim programima odnosno API-u (engl. *Application programming interface*), ali isto tako i za provjeru ispravnosti razvijenog aplikacijskog sloja. Slični simulacijski sustavi (Atmel Studio Simulator, Proteus, Virtronics itd.) postoje i za neke druge platforme, međutim niti jedan od programa ne koristi tehnologiju poput Winsock, odnosno poslužitelj/klijent aplikacije što otvara vrata mnogim mogućnostima i daljnji razvoj aplikacija za testiranje, prikupljanje podataka i slično koje bi se mogle povezati sa glavnom aplikacijom. Povezivanje s takvom aplikacijom u virtualnom okruženju može se također ostvariti i udaljenim putem, preko interneta. Samo testiranje može se tako izvršavati i sa više različitih lokacija, što može biti praktično ako dva tima djeluju na različitim lokacijama.

Trenutno razvijeni BM_HAL sloj ima prostora za napredak u vidu fleksibilnije konfiguracije mikrokontrolera, proširenja funkcionalnosti i na ostale značajke mikrokontrolera te unaprjeđenje funkcionalnosti u virtualnom okruženju.

LITERATURA

[1] AUTOSAR menadžment izdavanja dokumentacije, AUTOSAR specifikacija apstrakcije ulaza/izlaza sklopovlja, verzija 4.3.0, 30.11.2016., [online], dostupno na: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_IOHardwareAbstraction.pdf [5.10.2020.]

[2] FUJITSU, sloj apstrakcije mikrokontrolera – MCAL prema AUTOSAR 2.1 verziji, 2008., [online], dostupno na: https://www.fujitsu.com/downloads/EDG/binary/pdf/find/26-3e/1.pdf [26.9.2020.]

[3] ST, AUTOSAR-MCAL sloj, [online], dostupno na: https://www.st.com/en/embedded-software/spc5-autosar-mcal.html [26.9.2020.]

[4] T. Wenzel, R. Zalman, D. Nugraha, J. Venkataraman, „Implementing Automotive Microcontroller Abstraction Layer (MCAL) on 32-bit Architectures“, SAE Technical Paper 2006-01-1554, 2006.

[5] STMicroelectronics, STM32CubeMX, [online], dostupno na: https://www.st.com/en/development-tools/stm32cubemx.html [26.9.2020.]

[6] STMicroelectronics, Description of STM32F1 HAL and low-layer drivers, UM1850, Veljača 2020., [online], dostupno na: https://www.st.com/content/ccc/resource/technical/document/user_manual/72/52/cc/53/05/e3/4c/98/DM00154093.pdf/files/DM00154093.pdf/jcr:content/translations/en.DM00154093.pdf. [26.9.2020.]

[7] Microchip, Hardware Abstraction Layer, Rev 1.0, 2017., [online], dostupno na: http://ww1.microchip.com/downloads/en/DeviceDoc/hardware-abstraction-layer.pdf [26.9.2020.]

[8] G. Ellis, Control System Design Guide (Fourth Edition), Butterworth-Heinemann, Ujedinjeno Kraljevstvo, 2012.

[9] Wikipedia, ARM arhitektura, [online], dostupno na: https://en.wikipedia.org/wiki/ARM_architecture [17.8.2020.]

[10] C. Noviello, Mastering STM32, Leanpub, 2018., [online], dostupno na: https://leanpub.com/mastering-stm32 [10.8.2020.]

[11] Wikipedia, Centralna procesorska jedinica, [online], dostupno na: https://en.wikipedia.org/wiki/Central_processing_unit [26.9.2020.]

[12] STMicroelectronics, STM32 – 32 bitni mikrokontroleri, [online], dostupno na: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html [26.9.2020.]

[13] STMicroelectronics, STM32F103C8 dokument tehničke specifikacije, [online], dostupno na: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf [26.9.2020.]

[14] Wikipedia, Eclipse program, [online], dostupno na: https://en.wikipedia.org/wiki/Eclipse_(software) [17.8.2020.]

[15] Eclipse, Eclipse C/C++ razvojni alati, [online], dostupno na: https://projects.eclipse.org/projects/iot.embed-cdt [17.8.2020.]

[16] OpenOCD, [online], dostupno na: http://openocd.org/doc/html/About.html [17.8.2020.]

[17] D. Rath, Diploma Thesis, Open On Chip Debugger (OpenOCD), Sveučilište primijenjenih znanosti, Odjel za računalne znanosti, Augsburg, 2005.

[18] WxWidgets, [online], dostupno na: https://www.wxwidgets.org/ [17.8.2020.]

[19] Wikipedia, Klijent – poslužitelj model, [online], dostupno na: https://en.wikipedia.org/wiki/Client%E2%80%93server_model [17.8.2020.]

[20] Wikipedia, OSI model, [online], dostupno na: https://hr.wikipedia.org/wiki/OSI_model [17.8.2020.]

[21] Wikipedia, Mrežni *socket*, [online], dostupno na: https://en.wikipedia.org/wiki/Network_socket [17.8.2020.]

[22] Wikipedia, Winsock tehnologija, [online], dostupno na: https://en.wikipedia.org/wiki/Winsock [17.8.2020.]

SAŽETAK

U ovom radu je prikazana je arhitektura sloja za apstrakciju mikrokontrolera koja omogućava neovisnost o sklopovlju. Postojeći sustavi za apstrakciju nemaju toliku fleksibilnost kada je riječ o korištenom sklopovlju. Najčešći je slučaj da se pod „slojem za apstrakciju mikrokontrolera“ smatra generiranje dijela programske podrške koji pomaže pri bržoj konfiguraciji, no i dalje služi za razvoj programske podrške za određeni mikrokontroler. Slojem BM_HAL za apstrakciju mikrokontrolera omogućava se također izvršavanje aplikacije i u virtualnom okruženju. Prilikom razvoja u aplikacijskom sloju, programer ima mogućnost izbora na kojem sustavu će se izvršavati izvršna datoteka. Pomoću jedne pretprocesorske naredbe odabire se o kojem je sustavu riječ. Trenutno je omogućen rad sa STM32F103C8 mikrokontrolerom te radom u virtualnom okruženju Windows operacijskog sustava. Razvojni inženjer pomoću zajedničkog API-a, naziva BM_HAL, ima mogućnost pisati programski kod bez razmišljanja o fizičkom sloju te kada želi testirati aplikaciju, dovoljno je samo definirati ciljnu platformu. Nakon prevođenja, aplikacija se bez ikakvih dodatnih izmjena može izvršavati na prethodno definiranom željenom sustavu. Osim programske podrške i upravljačkih programa, fizički su napravljena dva sustava sa STM32F103C8 mikrokontrolerima, potrebna za demonstraciju rada apstrakcijskog sloja. Sustavi su spojeni preko uvrnute parice preko koje se odvija komunikacija CAN protokolom. Za potrebe demonstracije rada BM_HAL sloja za apstrakciju, razvijena je i jedna aplikacija sa grafičkim sučeljem putem koje je omogućeno postavljanje virtualnih ulaza i izlaza.

Ključne riječi: HAL, STM32, Winsock, apstrakcija, Windows, mikrokontroler

ABSTRACT

This paper presents the architecture of a microcontroller abstraction layer that allows hardware independence. Existing abstraction systems do not have as much flexibility when it comes to the hardware used. The most common case is that the "microcontroller abstraction layer" is considered to generate a piece of software that helps in faster configuration, but still serves to develop software for a particular microcontroller. The BM_HAL layer, developed for microcontroller abstraction, also allows application execution in a virtual environment. When developing in the application layer, the programmer has the ability to choose on which system the executable file will be executed. One preprocessor command selects which system it is. It is currently possible to work with the STM32F103C8 microcontroller and work in the virtual environment of the Windows operating system. A development engineer using a common API, called BM_HAL, has the ability to write program code without thinking about the physical layer, and when he wants to test an application, all he has to do is define the target platform. After compiling, the application can be run on a predefined desired system without any additional changes. In addition to software and drivers, two boards with STM32F103C8 microcontroller, required to demonstrate the operation of the abstraction layer, were made. The boards are connected via a twisted pair through which communication via CAN protocol takes place. For the purpose of demonstrating the operation of the BM_HAL abstraction layer, an application with graphical user interface has been developed, which enables the setting of virtual inputs and outputs.

Keywords: HAL, STM32, Winsock, abstraction, Windows, microcontroller

ŽIVOTOPIS

Branimir Mihelčić rođen je 24. svibnja 1996. godine u Slavonskom Brodu. Osnovnu školu završava u OŠ „Ivan Meštrović“ Vrpolje. Po završetku osnovne škole upisuje Gimnaziju „Matija Mesić“ u Slavonskom Brodu, program Prirodoslovno-matematička gimnazija koji uspješno završava 2015. godine. Iste godine upisuje preddiplomski sveučilišni studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku te se nakon prve godine opredjeljuje za smjer Komunikacije i informatika. Stjecanjem titule prvostupnika 2018. godine, nastavlja svoje akademsko obrazovanje na istom fakultetu na diplomskom studiju, smjer Automobilsko računarstvo i komunikacije. Tijekom diplomskog studija, također pridružuje se osječkom timu tvrtke Rimac Automobili d.o.o. gdje je u trenutku pisanja ovog diplomskog rada proveo već 10 mjeseci.

PRILOZI

Sav programski kod isprogramiran u ovom radu nalazi se na GitHub platformi

https://github.com/bmihelcic/Diplomski