VJEŽBA 1: UPOZNAVANJE SA STM32F4 DISCOVERY RAZVOJNOM PLOČOM I INTEGRIRANIM RAZVOJNIM OKRUŽENJEM ATOLLIC TRUESTUDIO. PISANJE I POKRETANJE PROGRAMA. DEBUG.

I. Cilj vježbe

Upoznati se s karakteristikama i komponentama STM32F4 Discovery razvojne ploče. Upoznati se integriranim razvojnim okruženjem Atollic TrueStudio, sastavnim dijelovima projekta unutar ovog IDE-a te na koji se način programira i debugira program u mikroupravljaču. Upoznavanjem s arhitekturom ARM Cortex-M4 jezgre i asemblerskim jezikom.

II. Opis vježbe

Na laboratorijskim vježbama koristi se STM32F4 Discovery razvojna ploča kao hardverska komponenta koja sadrži mikroupravljač zasnovan na ARM Cortex-M4 dok se Atollic TrueStudio koristi kao pripadno integrirano razvojno okruženje.

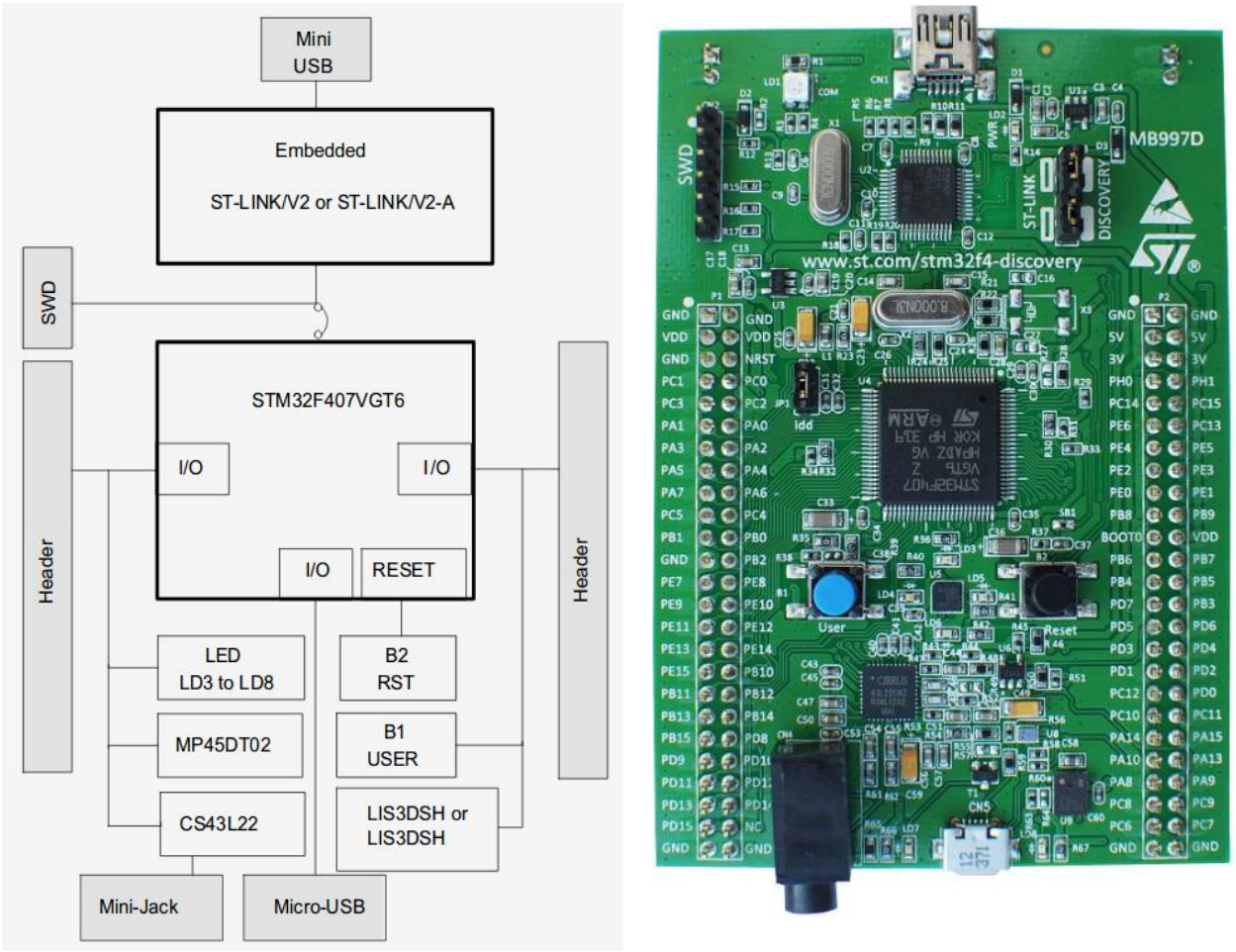
II.1. STM32F4 Discovery razvojna ploča

STM32F4 Discovery razvojna ploča sadrži STM32F407VG mikroupravljač visokih performansi koji ima ARM Cortex-M4 jezgru. Na ploči se osim mikroupravljača nalazi i dodatna periferija:

- programator/debugger ST-LINK/V2
- dva ST-MEMS digitalna akcelerometra
- digitalni mikrofon
- audio DAC
- svjetleće diode (LED)
- push buttons
- USB OTG micro-AB connector
- elektronika za napajanje, signalizaciju i zaštitu
- elektronika za generiranje takta
- pinovi mikroupravljača izvučeni u dvije letve 25x2

Budući da je na ploči ugrađen programator/debugger te su izvučeni pinovi mikopupravljača na dvije letve (engl. *header*), ploča omogućuje brzo prototipitanje i isprobavanje izgrađene programske podrške. Ploča se na PC spaja putem USB kabela koji ploči osigurava osnovno napajanje. Na slici 1.1. prikazana je STM32F4 Discovery razvojna ploča te pripadni hardverski blok dijagram.
Srce ove razvojne ploče je STM32F407VG ARM Cortex-M4 mikroupravljač. Detalji oko ovog mikroupravljača mogu se pronaći ovdje. Neke od važnijih karakteristika ovog mikroupravljača su:

- ARM® 32-bit Cortex®-M4 CPU takta 168 MHz
- FPU, MPU
- 1 MB Flash memorije
- 192 KB SRAM
- 1.8V do 3.6 V napajanje
- 3×12-bit 2.4 MSPS A/D konverter
- 2 x 12 bitni D/A konverter
- 17 brojača vremena (16 bitni i 32 bitni)
- SWD i JTAG debug sučelja
- 140 I/O portova
- 15 komunikacijskih sučelja (3 x I2C, 4 x UART, 3 x SPI, 2 x CAN, SDIO)
- DMA opće namjene
- generator nasumičnih brojeva
- jedinica za izračunavanje CRC koda
- RTC



Sl. 1.1. STM32F4 Discovery razvojna ploča i pripadni hardverski blok dijagram.

II.2. Atollic TrueStudio

C/C++ prevoditelj, asembler i povezivač je niz *command line* alata za razvoj programske podrške za ugradbene računalne sustave i naziva se GNU toolchain. Uz ovaj niz alata obično se koriste dodatni alati za otklanjanje pogreški u programu kao što je GNU debugger (GDB) i odgovarajući tekstualni editori. Za proces prevođenja koristi se make alat koji čita makefile datatoteku u kojoj se nalaze upute kako izgraditi cijelu programsku podršku (pozivanje asemblera, prevoditelja i povezivača). Velika prednost navedenih alata je njihova otvorenost. Međutim, izrada cjelokupnog niza alata kako bi se pomoću njih razvijala programska podrška za određenu (ugradbenu) platformu predstavlja težak zadatak jer zahtijeva prevođenje, podešavanje, testiranje i održavanje navedenih alata.

Postoji čitav niz integriranih razvojnih okruženja (engl. *Integrated Development Environment* - IDE) koji su već podešeni za programiranje određenih hardverskih platformi i u kojima se postupak izrade programske podrške pomoću navedenog niza alata značajno pojednostavljuje, npr. automatskim upravljanjem makefile datotekom, dodatnim alatima prilikom debugiranja programa i sl.

Jedno takvo rješenje je Atollic TrueStudio koji je besplatni IDE za programiranje mikroupravljača serije STM32 (32 bitni ARM Cortex mikroupravljači). Karakteristike ovog alata:

- zasnovan na Eclipse frameworku
- dolazi s GCC toolchainom za ARM i x86
- debugging je zasnovan na GDB
- podržava programatore/debugere poput ST-LINKa
- napredni editor
- mogućnost analize memorije (RAM, FLASH), prikaza CPU registara i ostale periferije, prikaz pozivanja stoga, asemblerski prikaz i sl.
- ugrađena podrška za kontrolu verzije programske podrške (Subversion, Git i CVS)

Kao i u većini IDE-a, osnovni način izrade programske podrške je preko izrade projekta s odgovarajućim postavkama i u kojem se nalaze sve potrebne datoteke, a generalno se sastoji od aplikacijskog koda, konfiguracijskog i startup koda za mikroupravljača koje pruža proizvođač mikroupravljača, dodatnih biblioteka (CMSIS, device specific datoteke i driveri) i linker skripte (vidi sliku 1.2.)

Sl. 1.2. Struktura projekta

III. Priprema za vježbu

1. Proučite dokument sa stranice STM32F4 Discovery razvojna ploča. Možete li odgovoriti na pitanja:
   a. Na koji pin mikroupravljača je spojena LED3?
   b. Na koji USB priključak je potrebno spojiti računalo ako se ploča želi programirati/debugirati?
   c. Koji mikroupravljač se nalazi na ovoj ploči?
   d. Na ploči se nalaze dva tipkala. Na koje pinove mikroupravljača su spojena ova tipkala i čemu služe?
   e. Kako se naziva programator/debugger koji je ugrađen na ovoj ploči? Na koji način programator/debugger komunicira s mikroupravljačem?

2. Pročitajte članak Embedded development using the GNU toolchain for ARM® processors. Od čega se sve sastoji GNU toolchain? Zašto je izrada vlastitog build sustava najčešće vrlo težak zadatak?

3. Zamislite jednostavan *hello world* program u C-u. Znate li koja je razlika između ova tri poziva gcc prevoditelja:

```
gcc -Wall -S hello.c -o hello.s
gcc -Wall -c hello.c -o hello.o
gcc -Wall hello.c -o hello
```

4. GCC prevoditelj ima niz dodatnih zastavica (engl. *flag*) kojima se specifira njegov rad. Koja zastavica se koristi za određivanje stupnja optimizacije? U slučaju ARM porta, što predstavljaju zastavice -mcpu i -mthumb? Koju vrijednost flaga -mcpu biste koristili za prevođenje programa namijenjenog mikroupravljaču na STM32F4 Discovery razvojnoj ploči?

5. Što predstavlja biblioteka newlib i zašto je važna?

6. Ponovite gradivo s AV ili ovdje vezano uz arhitekturu ARM Cortex-M4 jezgre.
   a. Koliko registara opće namjene ima ovakva arhitektura?
   b. Čemu služe registri R13, R14 i R15?
   c. Kako izgleda memorijski model ovakve jezgre? Na kojoj adresi počinje SRAM?

IV. Rad na vježbi

1. Pokrenite Atollic True Studio.

2. Potrebno je kreirati novi Projekt naziva *LV1* u kojem će se izraditi jednostavni program za STM32F4 Discovery razvojnu ploču. Odaberite *File-->New-->C project* te odaberite opciju *Embedded C Project.* Kako biste uspješno kreirali projekt potrebno je:
   a. dati odgovarajući naziv projektu (LV_1)
   b. odabrati raspoloživi hardware pod opcijom *Target* (točan naziv mikroupravljača pročitajte s raspoložive Discovery pločice)
   c. odabrati odgovarajući debug hardware (pročitajte s Discovery pločice)

   Ostale postavke nije potrebno mijenjati. Prisitkom na tipku *Finish* nastat će novi projekt naziva "LV_1".

3. Upoznajte se sa strukturom projekta. Od kojih se sve direktorija sastoji projekt? Čemu služe datoteke u pojedinom direktoriju? Pronađite `main` program. Pronađite linker skriptu. Na temelju skripte možete li reći veličinu RAM i flash memorije te na kojim adresnim područjima se nalaze?

4. Ako već projekt nije izgrađen, izgradite ga pritiskom na *Project-->Build Project.* Koja je razlika ove naredbe i naredbe *Project-->Build all ?*
   Informacije o tijeku izgradnje projekta prikazane su u *Console.* Možete li reći što predstavlja pojedini red u log-u koji je ovdje ispisan? Koji prevoditelj se koristi, s kojim opcijama i zašto? Koje su datoteke rezultat izgradnje projekta? Znate li što predstavlja *Print size information* i kratice *text data* i *bss*? Pogledajte *Build Analyzer* i memorijske regije. Pogledajte postavke projekta (*Project Properties*) kojima se definiraju različite opcije kod postupka prevođenja, spajanja i sl.

5. STM32F4 Discovery razvojna ploča omogućava debugiranje mikroupravljača (postavljanje breakpointa, step into, step over, uvid u varijable i sl.). Isprobajte dva jednostavna programa koja su dana u tablici 1.1 a) i b).

   Tablica 1.1. Primjer programskog koda

   a)
   ```
   int main(void)
   {
         int i= 0;

         ++i;
         ++i;
         ++i;
   }
   ```

   b)
   ```
   int i= 0;

   int main(void)
   {
         ++i;
         ++i;
         ++i;
   }
   ```

   Izgradite projekt te pokrenite *Debug mode* pomoću *Run-->Debug* (nije potrebno podešavanje ponuđenih opcija). U novootvorenom prozoru debuging će se automatski zaustaviti kada mikroupravljač započne izvršavanje main funkcije. Pronađite sljedeće kratice/alate:

   - Zaustavljanje i pokretanje Debug moda, Step Into Step Over, Instruction Stepping mode
   - Prikaz instrukcija koje je prevoditelj generirao na temelju izvornog koda - *Disassembly*
   - Prikaz registara jezgre mikroupravljača (po potrebi mijenjajte format zapisa registra)
   - Prikaz memorije (podesite prikaz 1 row 1 column)
   - Prikaz lokalnih varijabli programa (podesite prikaz i lokacije pojedine varijable)

   Čemu služi pojedina kratica/alat? Za dani programski kod u Tab. 1.1. a) i b):
   a. Izvodite jednu po jednu instrukciju. Koju operaciju predstavljaju dobivene instrukcije? Provjerite stanja registara opće namjene. Što se događa s PC registrom i zašto?
   b. Na kojim adresama se nalaze adrese instrukcija? Pronađite ih u prikazu memorije. Koliko memorije zauzima svaka instrukcija? U kojem tipu memorije su pohranjene instrukcije?
   c. Koji tipovi adresiranja se koriste kod izvođenja ?
   d. Dodajte u *Memory* prozoru praćenje adrese na kojoj je pohranjena vrijednost varijable *i.* Unutar prozora *Variables* promijenite vrijednost varijable *i* u -559038737. Što primjećujete u memorijskom prozoru? Kako je pohranjena varijabla? Radi li se ovdje o little ili big endianu?
   e. Kako se očituje razlika između koda a) i b) s obzirom na asemblerski kod?
   f. Pronađite u postavkama projekta stupanj optimizacija prevoditelja te ga povećajte. Što se dogodilo s generiranim instrukcijama u odnosu na prevođenje bez optimizacije (-O0)?

6. Napišite jednostavnu funkciju 1.2. koja množi dani broj s 2 te ju pozovite iz glavnog programa. Pokrenite *Debug* mod te provjerite generirane instrukcije (koristite zastavicu -O0 prilikom prevođenja).
   a. Koja se instrukcija koristi za pozivanje funkcije? Što se događa odmah na početku funkcije i kako to objašnjavate? Na koji način se puni stog (prema gore ili prema dolje)?
   b. Koju instrukciju je prevoditelj generirao za množenje s brojem 2? Zašto?
   c. Pomoću koje instrukcije se izvođenje programa vraća nazada u glavni program?
   d. Koju instrukcije generira prevoditelj ako funkcija množi dani broj s 3?
   e. Koji registar se koristi za povratnu vrijednost funkcije?

Tablica 1.2. Primjer funkcije

```
int funkcija(int a)
{
	int retval;
	retval = a*2;

	return retval;
}
```

7. Modificirajte primjer funkcije iz zadatka 6. na način da liniju:

```
return retval;
```

zamijenite s:

```
return funkcija(retval/2);
```

Što se događa tijekom izvođenja programa?

8. Napišite jednostavni program kako je dano u tablici 1.3. Koje instrukcije su generirane za naredbu uvjetnog izvođenja `if else`? Koji registar se mijenja prilikom izvođenja instrukcije CMP i zašto? Isprobajte i ostale funkcije za kontrolu toka programa (`for, while`) i provjerite generirane instrukcije.

Tablica 1.3. Uvjetno izvođenje programa

```
int main(void)
{
	int i=0;

	if(i>3)
		i++;
	else
		i--;
}
```

V. Za one koji žele znati više

Programiranje pomoću gcc toolchaina omogućava kombiniranje asemblerskog i izvornog C koda u istoj datoteci na način da se asemblerski dijelovi koda umetnu u C kod (engl. *inline*) pomoću ključne riječi **`asm`**. Više detalja pročitajte ovdje.