

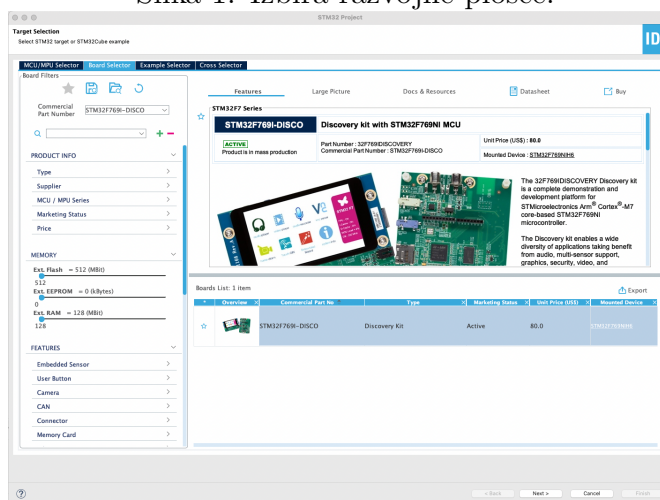
Spoznavanje z razvojnim okoljem in delo s splošno-namenskimi vhodom/izhodom

Na tokratni vaji se bomo spoznali z delom z razvojno ploščo STM32F769I-DISCO in razvojnim okoljem STM32CubeIDE. Naslednjih nekaj vaj bomo prižigali LED diode in brali ali je gumb pritisnjen. Na tokratnih vajah na malo bolj naiven način, ki ga bomo nato v naslednjih tednih nadgrajevali do najbolj optimalnega pristopa, uporabe knjižnice proizvajalca mikrokontrolerov.

Vzpostavitev in nalaganje prvega projekta

Za delo z razvojno ploščo bomo uporabljali STM32CubeIDE, ki si ga lahko brezplačno prenesete iz spletne strani podjetja ST ([povezava](#)). Po namestitvi zaženite program. Nov projekt za ploščo STM32F4 Discovery naredite tako, da v meniju izberete **File -> New -> STM32 Project**. Prikazal se vam bo meni za izbiranje ciljnega mikrokontrolerov. Izberite zavihek **Board Selector** in nato v desnem delu okna poiščite **STM32F769I-DISCO**, kot je prikazano na spodnji sliki.

Slika 1: Izbira razvojne plošče.

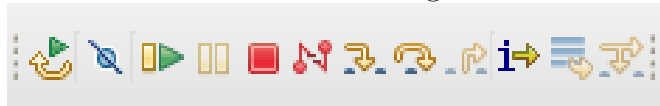


Nato kliknete **Next**, vpišite ime projekta, kliknete **Next** ter nato še **Finish**. Vse nastavitve pustite na prednastavljenih vrednostih. Na pojavno okno z vprašanjem **Initialize all peripherals with their default Mode?**

odgovorite z No. Na ostala vprašanja pojavnih (popup) oken odgovorite pritrdilno.

V oknu `Project Explorer` odprite datoteko `Core/Src/main.c`. Ta datoteka vsebuje `main` funkcijo našega programa. Nato priključite razvojno ploščo in izberite `Run -> Debug`. V pojavnem oknu izberite `STM32 MCU C++ Application`. Na tej točki se bo projekt prevedel. Na pojavno okno, ki vas bo vprašalo, če želite preklopiti v `Debug` perspektivo/način, odgovorite pritrdilno. V `Debug` načinu lahko program zaženete, ga ustavite, se postopoma pomikate čez program, spremljate vrednosti spremenljivk, itd.. Če želite samo zagnati vaš program kliknite na gumb `Resume` (tretji iz leve strani na sliki 2). Z gumbom `Terminate` (rdeč kvadrat na sliki 2) se vrnete nazaj v osnovno razvijalski pogled.

Slika 2: Kontrolni gumbi.



Vso kodo, ki jo boste pisali v `main.c` datoteko pišite v dele, ki so namenjeni uporabniški kodi. Ti deli so označeni z `BEGIN` in `END` komentarjem, kot je prikazano spodaj.

```
1  /* USER CODE BEGIN 1 */  
2  
3  /* USER CODE END 1 */
```

Na ta način se boste izognili temu, da vam IDE prepíše vašo kodo ko/če boste spreminjali nastavitve projekta. Prav tako ne odstranjajte klicev funkcij `HAL_Init()` in `SystemClock_Config()`, ki služita osnovnim nastavitvam razvojne plošče. Prav tako ne odstranjajte `while` zanke na koncu `main` funkcije, lahko pa poljubno spreminjate njeno vsebino. `Main` funkcija se namreč na mikrokontrolerju ne sme nikoli zaključiti, saj delovanje sistema po zaključku `main` funkcije ni definirano. Tu namreč nimamo operacijskega sistema, ki bi prevzel delovanje po koncu programa.

Naiven način dela s splošno-namenskim vhodom/izhodom

Branje iz naslova

Tisti, ki ste obiskovali predmet Arhitektura računalniških sistemov ste spoznali, da v zbirniku za HIP 32-bitni podatek na določenem naslov preberete z ukazom `lw r1, NASLOV (r0)`. Bolj splošna rešitev za poljuben naslov je podana spodaj. Poleg nje je zapisana tudi rešitev za isto operacijo v zbirniku za ARM procesorje, ki jo bolje poznate tisti, ki ste obiskovali predmet Računalniška arhitektura. V primeru, da nas zanima 16-biten ali 8-biten podatek uporabimo ukaza `lh` in `lb`.

```
1 //HIP
2 lhi r1, 0x40020C00
3 addui r1,r1, 0x40020C00
4 lw r2, 0 (r1)
5
6 //ARM
7 ldr r1,=0x40020C00
8 ldr r2, [r1]
```

V programskem jeziku C za branje iz specifičnega naslova potrebujemo kazalec. Temu kazalcu določimo naslov na katerega kaže, nato pa z dereferenciranjem (operatorjem `*`) preberemo vrednost na želenem naslovu. Kako bi to zapisali v primeru 32-bitne spremenljivke je prikazano spodaj.

```
1 uint32_t *p = (uint32_t *) 0x40020C00;
2 uint32_t vrednost = *p;
```

Če bi želeli brati 16-bitno ali 8-bitno vrednost, bi za to morali zgolj spremeniti tip kazalca v `uint16_t *` ali `uint8_t *`.

Pisanje na naslov

Prav tako ste pri predmetih ARS in RA izvedeli, da v zbirniku za HIP vrednost na določen naslov zapišemo z ukazom `sw r1, 0x400 (r0)`, oziroma da v zbirniku za ARM to dosežemo z ukazom `str r2, [r1]`.

```
1 //zelimo zapisati 0x55448000 na naslov 0x40020C18
2 //HIP
3 lhi r1, 0x40020C18
4 addui r1,r1, 0x0C18
5 lhi r2, 0x55448000
6 addui r2,r2, 0x8000
7 sw r2,0(r1)
8
9 //ARM
10 ldr r1,=0x40020C18
11 ldr r2,=0x55448000
12 str r2,[r1]
```

Za rešitev v C-ju zopet potrebujemo kazalec, ki mu določimo naslov. Za nastavljanje vrednosti na naslovu pa zopet uporabimo operator *, ki dereferencira kazalec. Za zapisovanje 16-bitnih ali 8-bitnih vrednosti moramo zopet zgolj spremeniti tip kazalca.

```
1 uint32_t *p = (uint32_t *) 0x40020C18;
2 *p = 0x55448000;
```

Rezervirana beseda volatile

Rezervirano besedo `volatile` uporabimo podobno kot `unsigned`: `volatile int i = 5;`. S tem prevajalniku sporočimo, da naj ukazov s spremenljivko `i` ne optimizira. `Volatile` spremenljivke se uporabljajo kadar vemo, da se lahko vrednost spremenljivke nenadoma spremeni ali preprosto ne želimo optimizacije dela s spremenljivko. Do nenadnih sprememb spremenljivke lahko pride, če:

- Je spremenljivka pomnilniško preslikana, na primer predstavlja stanje gumba.
- Do spremenljivke dostopamo iz prekinitveno servisnih programov (več o tem kaj to sploh je boste spoznali čez nekaj tednov).
- Do spremenljivke dostopa več niti.

Izklop optimizacije za posamezno spremenljivko pa potrebujemo kadar želimo narediti preprosto zakasnitev.

```
1     int i = 5000000;
2     // prevajalnik bi to zanko odstranil
3     while (i--) {
4     }
5     i = 1000000;
6     // tudi to zanko bi prevajalnik odstranil
7     while (1) {
8         i--;
9     }
10    volatile int j = 5000000;
11    // ta zanka bi izvedla vseh 5000000 iteracij
12    while (j--) {
13    }
```

Naloga

Iz učilnice naložite izhodišče za main.c datoteko vašega projekta. V datoteki realizirajte sledeče funkcije:

- `button_port_clock_on()`, ki 32-bitnemu podatku (registru) na naslovu `0x40023830` nastavi bit na poziciji 0 na 1.
- `led_port_clock_on()`, ki 32-bitnemu podatku (registru) na naslovu `0x40023830` nastavi bita na poziciji 0 in 9 na 1.
- `button_init()`, ki 32-bitnima podatkom na naslovih `0x40020000` in `0x4002000C` pobriše (nastavi na 0) bita 0 in 1 .
- `led_init()`, ki:
 - pobriše (nastavi na 0) bita 27 in 31 ter postavi (nastavi na 1) bita 26 in 30 na naslovu `0x40022400`,
 - pobriše bite 26, 27, 30 in 31 na naslovih `0x40022408` in `0x4002240C` ter bita 13 in 15 na naslovu `0x40022404`,
 - pobriše bit 24 in postavi bit 25 na naslovu `0x40020000`,
 - pobriše bita 24 in 25 na naslovih `0x40020008` in `0x4002000C`,
 - pobriše bit 12 na naslovu `0x40020004`.

- `led_on(uint8_t i)`, kjer je `i` vrednost od 0 do 2. V primeru, da je `i=0`, 16-bitnem registru na naslovu `0x40022418` postavi bit 15 na 1. Če je `i=1`, 16-bitnem registru na naslovu `0x40022418` postavi bit 13 na 1. Če je `i=2`, 16-bitnem registru na naslovu `0x40020018` postavi bit 12 na 1.
- `led_off(uint8_t i)`, kjer je `i` vrednost od 0 do 2. V primeru, da je `i=0`, 16-bitnem registru na naslovu `0x4002241A` postavi bit 15 na 1. Če je `i=1`, 16-bitnem registru na naslovu `0x4002241A` postavi bit 13 na 1. Če je `i=2`, 16-bitnem registru na naslovu `0x4002001A` postavi bit 12 na 1.
- `read_button()`, ki vrne 1, če je bit 0 16-bitnega registra na naslovu `0x40020010` 1, sicer vrne 0.

Pri realizaciji funkcij si lahko pomagata s funkcijami, ki ste jih spisali pri osvežitvi programskega jezika C.

Z realiziranimi funkcijami napišite program, ki bo ob pritisku na gumb prižge vse 3 LED diode. Ko gumb izpustite, naj se LED diode ugasnejo.