INFORMATIKA

Razvoj ugradbeih sustava s EFM8 mikrokontrolerom – 2. dio

Ivica Čulina[1]

Pisanje programa

Nastavimo s našim projektom. Upišite sljedeći kôd u text editor. Tekst formatiran kao ***bold italic*** je kôd programa koji upisujete.

Objašnjenja se nalaze nakom linija kôda te su odvojena crticama (----------). Pisana su u normalnom formatu i nisu dio kôda.

Ukoliko je tekst kôda u editoru, konzoli i debuggeru slabo čitljiv možete promijeniti postavke na *Window* → *Preferences*.... Potrebno je eksperimentirati dok ne dobijete željni efekt.

Globalne postavke mijenjate na > *General*, > *Editors*, > *Text Editors* te > *C/C++*, > *Editor*

Za promjenu formatiranja source codea odaberite > *C/C++*, > *Editor*, *Syntax Coloring* te izmjenite opcije pod *Assembly*.

Za promjenu formatiranja teksta na konzoli odaberite > *C/C++*, > *Build*, *Console*, te > *Run/Debug*, *Console*.

Za promjenu formatiranja debuggera odaberite > *C/C++*, >> *Debug*.

CSEG AT 0 ; start code at reset vector

CSEG nije strojna naredba za mikrokontroler nego interna direktiva asembleru/linkeru (eng. assembler directives, assembler control statements).

Direktiva je skraćenica za Code Segment.

Kaže linkeru da prilikom smještanja kôda u memoriju stavi kôd koji slijedi nakon direktive na memorijsku lokaciju 0 (decimalni zapis, odnosno 0x0000 heksadecimalno).

EFM8 mikrokontroler (odnosno 8051 procesor) nakon reseta (odnosno uključivanja napajanja) počinje dohvaćati kôd s memorijske lokacije 0.

----

start:

----

start je **label**. Label nije strojna naredba nego direktiva asembleru da trenutnoj adresi (lokaciji) u kôdu pridruži simbol (eng. symbol) naziva *start*. U ovome slučaju asembler će simbolu *start* pridružiti vrijednost 0.

[1] Autor je dipl. ing. elektrotehnike, stručni suradnik na Veleučilištu Velika Gorica;
e-pošta: ivica.culina@hotmail.com

Napomena. Start nije ključna riječ, mogli smo label nazvati bilo kako, primjerice *main* ili *begin*).

;clear all registers

Ovo je komentar (eng. comment) unutar kôda. Keil asembler komentarima smatra svaki dio teksta koji počinje sa znakom točka-zarez (**;**).

Asembler ignorira komentare tijekom generiranja strojnog kôda, te oni služe samo za nas programere kako bismo se lakše snalazili u kôdu.

MOV R0, #0

MOV R1, #0

CLR A

Sada kreću stvarne naredbe za mikrokontroler. Naredbe MOV Ri, #0 spremaju nulu u dati registar. Naredba CLR A, briše (resetira) akumulator (interni registar procesora) na 0.

MOV R0, #11 ; R0 = 11

MOV R1, #10 ; R1 = 10

Naredbe spremaju konstante (eng. literal, immediate) u date registre. Konstante smo naveli u decimalnom brojevnom zapisu. Asembler će za nas uraditi prijevod u binarni zapis koji računalo razumije.

MOV A, R0 ; A = R0

ADD A, R1 ; A = A + R1, odnosno A = R1 + R2

8051 je akumulatorska arhitektura što znači da se većina aritmetičkih operacija obavlja putem internog registra akumulatora (eng. accumulator). Kako bismo zbrojili sadržaj dva registra (R1 + R2) zbrajanje moramo obaviti preko akumulatora.

Prvo mičemo sadržaj registra R0 u akumulator. Potom izvršimo naredbu ADD A, R1. Naredba zbraja sadržaj akumulatora sa sadržajem unutar registra R1, te rezultat sprema natrag u akumulator.

Nakon izvršenja sadržaj u akumulatoru je 21 (decimalno).

AJMP start ; skoči na početak

Ova naredba mijenja tok programa. Riječ je o naredbi skoka (absolute jump). Naredba kaže da ne dohvaća sljedeću naredbu nego da skoči na zadanu memorijsku lokaciju. Ovdje koristimo sposobnosti asemblera koji će automatski za nas, prilikom generiranja strojnog kôda, zamijeniti simbol *start* sa stvarnom memorijskom lokacijom (u ovom slučaju nula). Kôd radi jednako ako zamijenimo *start* s 0, odnosno napišemo AJMP 0 (provjerite!).

- - - - - - - - - - - -

END

- - - - - - - - - - - -

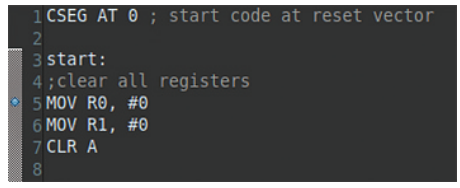
END je samo asemblerska direktiva koja znači da je source code našeg programa gotov. Asembler će ignorirati sav kôd napisan nakon END direktive.

- - - - - - - - - - - -

Prikaz kôda s numemriranim redcima (lines)

```
1.  CSEG AT 0 ; start code at reset vector
2.
3.  start:
4.  ;clear all registers
5.  MOV R0, #0
6.  MOV R1, #0
7.  CLR A
8.
9.  MOV R0, #11 ; R0 = 11
10. MOV R1, #10 ; R1 = 10
11.
12. MOV A, R0 ; A = R0
13. ADD A, R1 ; A = A + R1, odnosno A = R1 + R2
14. AJMP 0 ; skoči na početak
15.
16. END
```

Upišite kôd u tekst editor i dodajte **breakpoint** za debugger. Breakpoint dodajemo duplim klikom miša na broj koji označava liniju kôda. Pojavit će se mali plavi krug kraj linije kôda (označava breakpoint). Dodajte breakpoint na redak (line) 5.

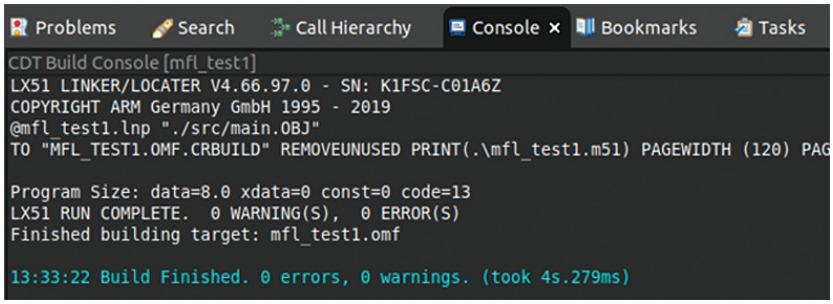


Slika 1. Breakpoint na redku 5 (line 5).

Sada je potrebno kompajlirati kôd (eng. **build**).

Pritisnite na tipkovni **Ctrl+b** (ili desni klik na mapu projekta u Project exploreru *Build Project*). Na konzoli (console) se prikazuje izlaz, odnosno rezultati kompajliranja.

Ukoliko ste kôd upisali točno rezultat će biti uspješan.

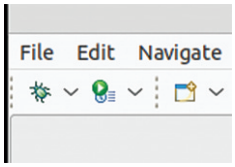


Slika 2. Konzola nakon uspješne izgradnje koda (build).

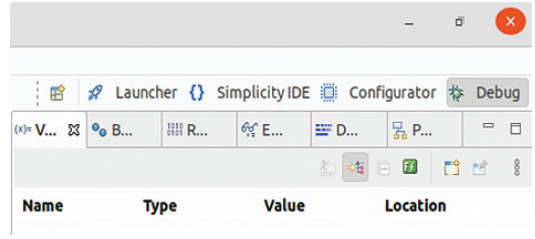
Debugging

Sada pristupimo debuggingu kako bismo vidjeli kako se kôd izvršava na mikrokontroleru.

Pritisnite ikonu “bube” – prva ikona s desne strane na alatnoj traci.



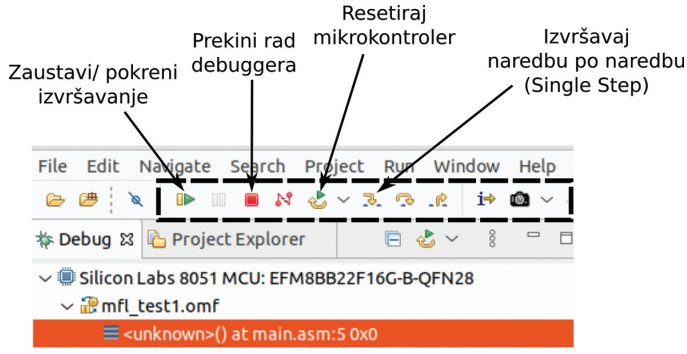
Slika 3. Ikonica za pokretanje debuggera.



Slika 4. Debug perspektiva.

IDE sada učitava naš strojni kôd u mikrokontroler (programira njegovu FLASH memoriju) i započinje debugging sesiju (eng. debbuging session). Debugging je moguć zbog ugrađenog sklopovlja unutar EFM8 mikrokontrolera.

Perspektiva se automatski mijenja na **Debug perspective**. Primjetite da je raspored ekrana i dostupnih alata različit.



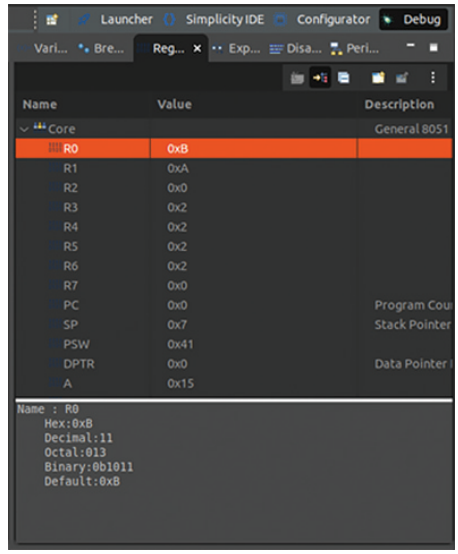
Slika 5. Debug sesija i ikonice alata debuggera.

IDE bi trebao automatski resetirati mikrokontroler te se zaustaviti na breakpointu na liniji 5. Ukoliko se to ne dogoditi napravite dupli klik na označenu stavku ($<$unknown$>$) na slici 5.

Na slici 5 također su prikazani osnovni alati za upravljanje debuggerom.

Pogledajmo sada desnu stranu ekrana. Kliknite mišem na tab **Registers**, te potom kliknite na **Core**.

Pojavit će se pregled internih registara mikrokontrolera sa njihovom trenutnom vrijednosti. Vrijednost je prikazana heksadecimalno, klikom na željeni registar prikazat će se vrijednost u decimalnom zapisu (slika 6).



Slika 6. Pregled internih registara za vrijeme izvršavanja koda.

Prođite kroz kôd naredbu po naredbu (single stepping) te pratite kako se stanja registara R0, R1 i A mijenjaju nakon izvršavanja pojedine naredbe.

Također pratimo vrijednost PC registra (Program Counter). To je registar koji sadrži adresu naredbe koju procesor dohvaća u datom taktu. Primijetimo kako registar počinje s vrijednosti 0 (dohvati naredbu na adresi 0x0) te se nakon izvršenja naredbe uvećava (broj za koji se uvećava ovisi o veličini pojedine naredbe, naredbe zauzimaju od 1 do 3 bytea).

Sada pogledajmo datoteke koje je kompajler stvorio.

Otvorite direktorij našeg projetka *mfl_test1/Keil 8051 v9.60.0 – Debug* te unutar njega uočite datoteke:

- **mfl_test1.m51**

Ova datoteka se zove **map file** i generira je linker. U njoj možemo vidjeti kako je naš program smješten u memoriji, te fizičke adrese koje je linker dodijelio simbolima u asemblerskom kôdu.

- **mfl_test1.hex**

Završni strojni kôd programa koji će se učitati u FLASH memoriju, prikazan u heksadecimalnom zapisu. Ovaj format zove se Intel hex format.

ISA – Koje naredbe procesor može izvršiti?

ISA – Instruction Set Arhitecture – je skup strojnih naredbi koje dati procesor (mikrokontroler) može izvršiti.

ISA za 8051 mikrokontroler se nalazi na stranici 83, *Reference Manual*.

U prvoj koloni tablice data je asemblerska mnemonika naredbe, a u drugoj koloni se nalazi kratki opis. Preporučljivo je isprintati cijelu tablicu te je uvijek imati na stolu kao referencu prilikom programiranja.

Detaljniji opis svake pojedine naredbe možete pronaći na stranicama Keil toolchaina – `https://www.keil.com/support/man/docs/is51/is51_instructions.htm`.

Rad s vanjskim jedinicama (I/O)

Sada ćemo pogledati program koji stvarno nešto radi. U tome i leži prava privlačnost ugradbenog programiranja – zbilja vidimo kako naš kod nešto radi u vanjskom svijetu – pali lampice, motor, pumpu, ...

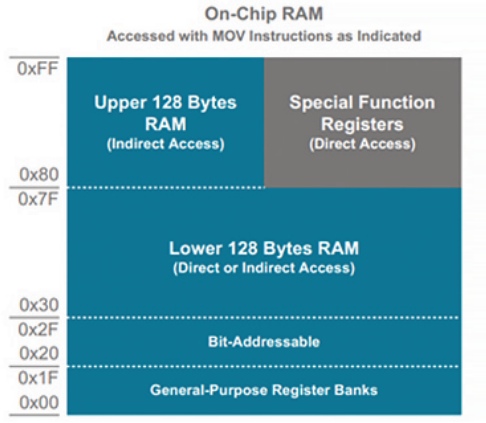
Ovdje ćemo napraviti za početak jednostavan program koji na pritisak dugmeta (push button) pali LED lampice.

Iskorištavamo mogućnosti razvojne pločice na kojoj se nalaze postavljeno dugme i LEDice.

Za razliku od PC računala gdje se većina vanjskih uređaja (I/O devices) nalazi u obliku kartica koje se spajaju na matičnu ploču, kod mikrokontrolera su svi uređaji integrirani na istom čipu kao i procesor.

Rad s vanjskim uređajima se svede na konfiguraciju registara vanjske jedinice – što je jednako pisanju bilo koje varijable u memoriji.

Takav pristup se zove *memory mapped I/O* – registri vanjskih jedinica su dio memorijske mape procesora, te je pisanje/ čitanje registara vanjske jedinice identično pisanju/ čcitanju varijable u memoriji.

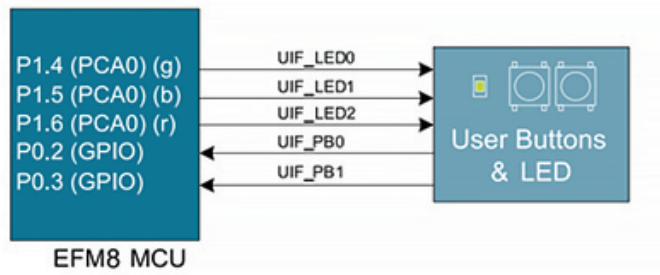


Slika 7. Memorijska mapa.

Na stranici 24 *Reference Manual* se nalazi opis memorije. Vidimo da su u gornjih 128 bita RAMa mapirani registri vanjskih jedinica (peripherals). Registrima moramo pristupiti naredbama za direktni pristup memoriji (primjerice MOV A, direct). Pristupanje istim adresama indirektno putem pokazivača (primjerice naredba MOV A, @Ri) uzrokuje pristup RAM memoriji. Dio registara vanjskih jednica je također bit-adresabilan, što znači da se svaki bit registra može nezavisno manipulirati pomoću naredbi iz grupe *Boolean manipulation.* Više informacija potražite u *Reference Manual-u*.

Nas ovdje zanima vanjska jedinica koja omogućava pristup vanjskim pinovima uređaja – takva jedinica se zove General purpose I/O ili skraćeno GPIO. Jedinice su obično podijeljene na *portove*, gdje je svaki port skup nekoliko pinova.

Nas zanima kako su dugmi i LEDice spojeni na razvojnoj pločici. Pogledajmo na dokumentaciji (*Kit's User Guide)* na stranici 7.



Slika 8. Spoj dugmeta i LEDica na portove mikokontrolera.

Vidimo da su LEDice spojene na port P1 (pinovi 4, 5, 6), te da su dugmi spojeni na port P0 (pinovi 2 i 3).

U programu ćemo implementirati jednostavnu funkciju: na pritisak dugma PB0 upali plavu LED diodu (LED1). Kada je gumb pritisnut LEDica je upaljena, kada otpustimo gumb LEDica se gasi.

Program počinje s inicijalizacijom vanjske GPIO jedinice. Inicijalizacija se sastoji od pisanja potrebnih vrijednosti u registre vanjke jedinice. Kako bismo znali koje vrijednosti treba upisati u koje registre moramo proučiti danu tehničku dokumentaciju (stranica 91 *Reference Manual*). Prilikom pisanja koda ne moramo se služiti adresama registara – dovoljno je napisati ime registra. Imenu registra je pridružena odgovarajuća adresa unutar datoteke *SI_EFM8BB2_Defs.inc* koju nam sporučuje proizvođač.

Unutar inicijalizacijskog koda potrebno je namjestiti P0.2 pin kao ulazni (PB0) te P1.5 pin kao izlazni (plava LEDica). Ovdje nemamo dovoljno mjesta za detaljni opis inicijalizacije. Zainteresirani čitatelj mora sam proučiti dokumentaciju mikrokontrolera.

Nakon inicijalizacije vanjske jedinice počinje kôd "glavnog programa" koji izvršava sljedeću logiku: provjeri da li je gumb pritisnut, ako je promijeni izlaznu vrijednost pina na koji je spojena plava LED dioda.

Pisano pseudo kôdom:

loop (forever)

Da li je gumb pritisnut?

Ako je izmjeni vrijednost pina LEDice

Program

Ispod slijedi kôd s komentarima. Upišite ga u editor našeg projekta i testirajte!

Kada držimo dugme pritisnuto LEDica je upaljena, kada otpustimo gumb LEDica se gasi.

```
;-----------------------
; Definicije
;-----------------------
; naredba za dodavanje datoteke sa definicijama adresa registara
            $INCLUDE (SI_EFM8BB2_Defs.inc)
; datoteku možete pronaći pod datotekom Includes unutar mape projekta
; definicije pinova za LED i PB
; registrima su bit adresabilni
; ovdje definiramo mnemoniku za bit radi bolje čitljivosti programa
; služimo se direktivom EQUđ
            LED_B   EQU         P1.5; plava LEDica
            PB_0    EQU         P0.2; gumb 0
            CSEG AT 0     ; reset
;-----------------------
; inicijalizacija vanjske jedinice
;-----------------------
            ; Disable WDT
            MOV         WDTCN, #0x0DE
            MOV         WDTCN, #0x0AD
            ; Configure the crossbar and I/O
            ORL         P1SKIP, #0111$0000b ; skip the LED pins
            ORL         P2SKIP, #0000$1000b ; skip the DISP_EN pin
            ORL         P0SKIP, #0000$1100b ; skip the buttons
            ORL         P2MDOUT, #0000$1000b ; configure DISP_EN pin as push_pull
            CLR         P2.3                ; give disp control to board
            ORL         P0, #0000$1100b ; set buttons as inputs
            ORL         XBR2, #0100$0000b ; enable the crossbar
;-----------------------
; glavni program
;-----------------------
loop:
; testiraj vrijednost bit-a (pina gdje je spojen PB0)
; ako je bit u registru 0 znači da je dugme pritisnuto
            SETB        LED_B ; ugasi ledicu (postavi bit na 1)
            JB          PB_0, loop ; ako je bit 1 vrati se na loop label
             ; gumb je pritisnut
            CLR         LED_B ; upali LEDicu
            SJMP        loop ; opet sve ponovi (beskonačna petlja)
END
```

Napomena. Isprobajte logiku programa koristeći single stepping funkcionalnost debuggera. Pokušajte izmijeniti ili nadopuniti program.