ICM SoC Day 4: EEPROM, ugrađene i vanjske biblioteke

Prva tema ove radionice je detaljnije upoznavanje s Arduino memorijskom arhitekturom. Arduino posjeduje 3 vrste memorije: 2 trajne i jednu izbrisivu (engl. *volatile*). *Flash* memorija je oblik trajne memorije na koju se prebacuje kompilirani programski (strojni) kod. Trajna memorija ne znači da je nije moguće promijeniti, što je očito jer programski kod ste na dosadašnjim radionicama mijenjali i do stotinu puta. Trajna memorija znači da se ona ne briše po resetiranju Arduina. Izbrisiva memorija je, kao i na "punokrvnim" računalima, RAM (engl. *Random Access Memory)* tj. memorija slučajnoga pristupa - ona u kojoj su pohranjene varijable, kontekst, stogovi i slično - sve ono što je potrebno da procesor nesmetano obavlja zadaće napisane u *Flash* memoriji. Arduino Uno i Nano posjeduju . Ona se puni po instrukcijama programskog koda iz *Flash* memorije, i to svaki puta iznova prilikom paljenja: ovu operaciju obavljaju sva moderna računala: od mobitela do superračunala. Razlog tomu je što je varijable vrijedno čuvati na brzom memorijskom uređaju: izbrisive memorije su često nekoliko redova veličine brži od ostalih oblika memorije. Arduino posjeduje i još jednu vrstu memorije: EEPROM (engl. *Electrically Erasable Programmable Read-Only Memory*) - jedina vrsta memorije koja je direktno dostupna korisniku, odnosno da je iz nje moguće čitati i pisati pomoću programskog koda.

Arduino Uno i Nano, te njihovi derivati - općenito mikrokontroleri zasnovani na mikrokontroleru Atmel ATmega328P imaju sljedeće memorijske specifikacije:

Flash: 32kB (od kojih 512B zauzima početni program - *bootloader*)
SRAM (RAM): 2kB
EEPROM: 1024B

Arduino IDE - razvojna okolina koju ste do sad koristili, procijenit će koliko ste koje memorije utrošili i koliko još imate slobodno. To ne mora značiti da ćete imati dovoljno memorije: ukoliko ste već "na rubu" mogućnosti, vrijeme je za prelazak na moćniju alternativu Arduina, koja se programira na isti način: primjerice NodeMCU, STM32Fx i slični. Te pločice cijenovno su zanemarivo skuplje, do 1000 puta moćnije dostupnim procesorskim vremenom (do 2GHz u odnosu na 16MHz, s mogućnošću izvršavanja više instrukcija po taktu) te sklopovskom podrškom za operacije nad realnim brojevima. Isto vrijedi i za memoriju, navedeni moderniji moduli imaju i do 100MB RAM memorije i do nekoliko GB *Flash* memorije.

Jedna od mana EEPROM-a je relativno kratak životni vijek u odnosu na druge dvije vrste memorija dostupne na Arduinu: oko 100 000 puta. *Flash* memorija, za usporedbu, može "izdržati" do 100 puta više, a RAM i do nekoliko milijuna **puta** više.

EEPROM:

Za potrebe korištenja EEPROM-a potrebno je uključiti programsku biblioteku EEPROM, na način da na početak svojeg programskog koda, u zaseban redak i izvan `setup` i `loop` funkcija napišete:

```
#include <EEPROM.h>
```

Nekoliko relevantnih funkcija postoje koje se tiču zapisa u EEPROM:

```
EEPROM.write(adresa, podatak);
```

zapisuje podatak u adresu. Adrese kreću od nule (0) i završavaju brojem količine EEPROM memorije minus 1. U slučaju Arudino Una 1023. Imajte na umu da funkcija zapisuje samo jedan B (byte) vašeg podatka - u slučaju *int*-a koji je na Arduinu 16b ili 32b, zapisat će se samo prvih 8 najmanje značanih bitova. To nam obično neće predstavljati problem ukoliko su nam brojevi manji od 128. Tipovi podataka koji **cijeli** stanu u jednu memorijsku ćeliju su: `byte, bool, uint8_t` (8 bitni integer koji može zapisati samo pozitivne brojeve i nulu, 0 do 255), `int8_t` (8 bitni integer koji može zapisati brojeve od -128 do 127), `char`. Kasnije ćemo se baviti i pretvorbom tipova podataka.

```
EEPROM.update(adresa, podatak);
```

Nešto sporija verzija funkcije `EEPROM.write`, koja provjerava je li podatak potrebno zapisati, odnosno postoji li već taj podatak u memoriji - ako postoji, neće napraviti ništa i preskočiti će naredbu zapisivanje - **brže** od `EEPROM.write`. Ukoliko je podatak potrebno ažurirati, izbrisati će postojeći podatak iz memorije te upisati novi. Ovo je nešto sporije jer je prethodno potrebno usporediti podatak s postojećim.

```
EEPROM.read(adresa);
```

Čita podatak s memorijske lokacije na zadanoj adresi. Obično ga koristimo na način:

```
int procitani_podatak;
procitani_podatak = (int) EEPROM.read(adresa);
```

Operacija `(int)` (zvana i operacijom *cast*) pretvara tip podatka iz onog upisanog u memoriju u podatak vrste kakvu očekujemo. To ne mora nužno biti `int` kao u navedenom primjeru, ali poželjno je da bude jednakog tipa kao i ciljna varijabla u koju pohranjujemo pročitanu vrijednost.

Od često korištenih funkcija dostupne su i `EEPROM.get()` i `EEPROM.put()` kojima se nećemo danas baviti, ali ih možete proučiti na internetu i pokušati riješiti zadatke i s njima.

Zadatak 1: preko Arduino Serial Monitora, pošaljite broj Arduinu u obliku *integera* te neka ga on spremi u EEPROM. Po resetiranju Arduina, neka vam Arduino, prije nego traži unos novoga broja, ispiše prethodno uspisani broj iz EEPROM-a. Broj pohranite na adresu 0. Koje sve brojeve možete pohraniti na ovaj način?

Zadatak 2: Implementirajte program koji generira pseudo-slučajne brojeve, i to na način da se brojevi **ne** ponavljaju po resetiranju Arduina. Po paljenju Arduina potrebno je ispisati jedan slučajan broj, a svakim pritisku gumba još jedan.

Jednostavna funkcija za generiranje slučajnih brojeva opisana je matematički u nastavku:

```
random_broj  = 7 * random_broj mod 11;
```

Zadatak 3: Unaprijedite prvi zadatak na način da korisnik unosi cijelu rečenicu (duljine do 1023 znakova) te da ju je Arduino u mogućnosti ponoviti nakon resetiranja.

Zadatak 4: (Bonus) Pokušajte riješiti 3. Zadatak upotrijebom funkcija `EEPROM.get()` i `EEPROM.put()`

U kontekstu opširnijih projekata, često je potrebno pohranjivati neke vrijednosti koje bi Arduino koristio kasnije, čak i nakon gašenja. Često su neke od tih vrijednosti postavke: npr ime *bluetooth* uređaja i njegovu šifru na koji se spajamo ukoliko želimo da ne moramo svaki puta preko računala specificirati te podatke. Zato je neophodno poznavati koncept memorije koja je dostupna korisniku.

Ugrađene i vanjske biblioteke

Ponekad koristimo već gotovo napisane funkcije koje su drugi ljudi napisali umjesto nas. Ugrubo, podijelit ćemo ih u 3 cjeline: one koje su dostupne bez uključivanja (npr. Funkcija `delay(), micros(), Serial.begin()`...), one koje su dostupne u standardnom Arduino skupu biblioteka ali zahtjevaju uključivanje (npr. `math, EEPROM, servo`), i one koje moramo skinuti s interneta (npr. `TimerOne` s prethodne radionice). Biblioteke uključujemo na način:

```
#include <EEPROM.h> // uključivanje EEPROM biblioteke
#include <math.h>   // uključivanje biblioteke s naprednim
                    //  matematičkim funkcijama
```

Biblioteke skinute s interneta obično uključujemo na sličan način:

```
#include "TimerOne.h"
```

Biblioteke s interneta je moguće skinuti na dva načina:

a) Ukoliko je biblioteka sluđbena ili potvrđena od strane Arduino zajednice, instalacija se vrši na način da se u Arduino IDE pokrene izbornik Sketch - Include Library - Manage Libraries… i u prozoru tražimo potrebnu biblioteku i pritisnemo install

b) Skinemo .zip datoteku s bibliotekom i instaliramo je na način Sketch - Include Library - Add .ZIP Library te odabirom datoteke s bibliotekom obavi se instalacija

Primjere korištenja naredbi iz instaliranih biblioteka obično je moguće naći u File - Examples - [ime biblioteke] - [ime primjera]. Te primjere moguće je i pokrenuti.

Zadatak 5: Pokrenite servo motor prema primjeru korištenja biblioteke. Za pomoć u spajanju servo motora obratite se voditelju radionice ili potražite na internetu.

Zadatak 6: Pomičite servo na kut zadan preko Serial Monitora.

Zadatak 7: Pomičite servo na kut proporcionalan očitanju napona s potenciometra. Podsjetite se korištenja funkcije *map*. Za provjeru, ispišite kut preko serije.

Po završetku zadataka, voditelj radionice će vam ustupiti neki od dostupnih vanjskih modula za koji ćete dobiti individualan zadatak: koristiti vanjsku biblioteku i implementirati neku funkcionalnost.