**LABORATORIJSKA VEŽBA BR. 2**

# Principi low-level programiranja

## CILJ VEŽBE

- Upoznavanje sa programskim zahtevima kod programiranja Arduina
- Upoznavanje sa vrstom memorijskog prostora kod Aurdina
- Upoznavanje sa značajem veličine korišćenog memorijskog prostora
- Metode za efikasnije pisanje programskog koda
- Instaliranje i startovanje aplikacije na Arduino ploči
- Testiranje zauzeća memorijskog prostora na Arduino ploči
- Primena metoda za smanjivanje zauzeća memorijskog prostora

## POTREBNA OPREMA

- Računar sa instaliranim razvojnim softverom za Arduino
- Arduino UNO + Mega 3 komplet
- DHT i BH1750 senzore
- Multimetar
- Komplet alata za montažu

## TEORIJSKE OSNOVE

Low-level programiranje koje se primenjuje kod bežičnih senzorskih čvorova (BSČ) je sasvim drugačije od programiranja za uređaje opšte namene, kao što su računari ili mobilni telefoni(Android). Kako su kod BSČ resursi jako ograničeni, znatno veća pažnja se posvečuje što manjem zauzeću tih resursa kao što je Flash memorija(kompaktan programski kod), RAM memorija(manje promenljivih i integrisani sistemski programi sa aplikacijom), brzina procesora(smanjena potrošnja) kao i izboru odgovarajućeg režima rada pojedinih komponenti. Drugim rečima, veoma bitan naglasak se stavlja na optimiziaciju pojedinih delova programskog koda – manje vreme izvršavanja koda potrošiće i manje električne energije, što je osnovni cilj svake aplikacije u bežičnoj senzorskoj mreži.

### Mikrokontroleri

Mikrokontroleri koji se ugrađuju u BSČ (poput onih koji pokreću Arduino sistem) su dizajnirani za *embedded* (ugrađene) sisteme. Za razliku od računara opšte namene, ovakvi mikrokontroleri obično imaju procesor i dodatne komponente: serijski port, paraleleni port, A/D i D/A konvertore, tajmere, i td. Međutim, ove komponente imaju znatno osiromašene resurse koji su usko namenjeni za rešavanje unapred tačno definisanih zadataka koji moraju pouzdano i efikano da se izvršavaju i to uz minimalne troškove, pre svega utrošnje električne energije.

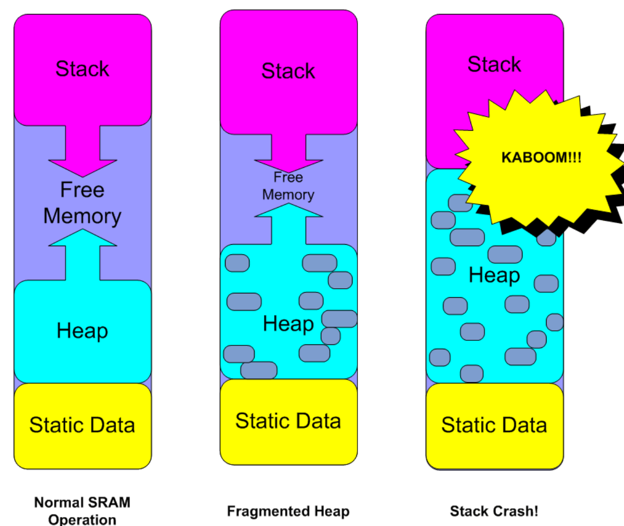
Najveća razlika između mikrokontrolera koji se koristiti na laboratorijskim vežbama (jednostavan 8-bitni ATmega328p ili složeniji CPU ATmega2560 - Arduino MEGA R3) i računara opšte namene (PC) je količina dostupne memorije. Arduino UNO ima samo 32KB flash memorije i 2KB SRAM-a što je

otprilike 2 000.000 puta manje od fizičke memorije kod PC-a (4 GB) I 32 000 000 puta manje u odnosu na sekundarnu memoriju (hard disk kapaciteta 1 TB). To je i glavni razlog što nekada može da se desi da Arduino bez ikakvih razloga "poludi" ili se resetuje sam po sebi, iako smo učitali programski kod koji je 100% ispravan. U takvim slučajevima, jedan od mogućih uzroka je nedostatak slobodnog RAM (*Random Access Memory*) memorije. Drugim rečima, mikrokontroler nema dovoljno slobodnog prostora u RAM memoriji kako bi smestio odgvarajući programski kod za izvršavanje traženog zadatka (vidi sliku br. 1).

## Memorija Arduino mikrokontrolera

Memorija Arduina se sastoji od tri dela:

- Flash ili programska memorija
- SRAM
- EEPROM




Slika 1. Prikaz pada programa zbog nedostatka slobodnog memorijskog prostora

SRAM ili *Static Random Access Memory* je tip memorije u koju se mogu upisivati ili čitati podaci sve vreme dok se izvršava program. Podaci koji se upisuju/čitaju iz SRAM memorije imaju višestruku ulogu prilikom izvršavanja programa i to:

- **Statički Podaci** - Ovaj blok memorije SRAM-a je rezervisan prostor za sve globalne i statičke promenljive koje se koriste u programskom kodu koji se izvršava.
- **Heap** - koristi se za dinamički alocirane podatke.
- **Stack** - upotrebljava se za lokalne promenljive i za održavanje evidencije prekida (*interrupts*) i poziva funkcija. U njemu se pamte podaci koji su potrebni prilikom promene konteksta procesa koji treba da se izvrši prilikom prekida. Naime, potrebno je upamtiti trenutne vrednosti registara CPU prekinutog programa kako bi se po izvršenju procesa koji je zahtevao prekid moglo da se vrati na nastavak izvršenja prekinutog programa. *Stack* raste od vrha memorije pa na dole prema dnu (prema *Heap-u,* vidi sliku br.1). Svaki interupt, poziv funkcije i/ili lokalna promenljiva utiče na povećanje memorijskog prostora koji zauzima Stack-a. Po povratku iz interrupta ili poziva funkcije oslobodit će se deo memorije koji je taj interupt ili funkcija upotrebljavala.

U sledećoj tabeli prikazana je veličina memorijskog prostora kod različitih Arduino sistema.

| Arduino | Processor | Flash | SRAM | EEPROM |
|---|---|---|---|---|
| UNO, Uno Ethernet, Menta, Boarduino | Atmega328 | 32K | 2K | 1K |
| Leonardo, Micro, Flora, 32U4 Breakout, Teensy, Esplora | Atmega 32U4 | 32K | 2.5K | 1K |
| Mega, MegaADK | Atmega2560 | 256K | 8K | 4K |

Jedan od načina dijagnoze mogućeg problema kod korišćenja memorije je detekcija zauzeća memorije.

1. Flash memorija - Kompajler prilikom kompajliranja koda izračuna veličinu potrebne flash memorije tako da mi unapred možemo znati da li je kod prevelik za upload na Arduino sistem.
2. SRAM memorija - Korišćenje SRAM memorije je dinamičnije i zato je znatno teže saznati koliko nam je memorijskog prostora potrebno za izvršenje programskog koda. Jedna od mogućnosti je korišćenje funkcije ***free_ram()*** koje je prikazano kasnije u test programu. Korišćenje SRAM-a je promenljivo i ono će se dinamički menjati sa vremenom. Zato je potrebno da se ova funkcija, ***free_ram()***, poziva u različitim vremenskim trenucima i iz različitih delova programskog koda, kako bi videli kako se veličina SRAM memorije menja tokom izvršavanja programskog koda.

```
int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

U osnovi, funkcija ***free_ram ()*** daje informaciju o slobodnom prostoru između Stack-a i Heap-a. Upravo taj prostor zaista treba pratiti ako želimo da izbegnemo pad programa.

### Optimizacija memorije Arduino programa

Prilikom kompajliranja programskog koda IDE nam daje informaciju koliko je izvršni kod velik (npr. PlatformIO) tj. Koliko nam je memorijskog prostora potrebno za njegovo izvršavanje. Ako smo premašili raspoloživi memorijski prostor u našem Arduino sistemu jedno šta nam ostaje da uradimo je da zamenimo taj Arduino Sistema za neki sa većim resursima ili da primenimo neki od optimizacionih mehanizama za smanjivanje izvršnog koda, što je znatno prihvatljivije:

- **Uklonite nepotrebne (mrtve) delove koda** - može se dogoditi da je rezultirajući kod nastao kao kombinacija koda iz više izvora, pa se lako može dogoditi da se neki delovi koda uopšte ne koriste u izvršnoj verzije pa ih treba ukloniti. Ti delovi se odnose na nekorišćene biblioteke, nekoriščene funkcije, nekorišćene promenljive ili na nedostupne delovi koda.
- **Optimizacija SRAM-a** - Postoje razni načini na koji možemo smanjiti korišćenje SRAM memorije. Ovo su samo neki od njih.
  - *Eliminisanje nekorišćenih promenljivih* - može se ponekad dogoditi da se neke promenljive uopšte ne koriste pa ih jednostavno izbrišite.
  - *Smanjite nepotrebnu veličinu promenljivih* - ne koristite *float* kada je *int* dovoljan ili nije potrebno koristiti *int* kada je *byte* dovoljan. Pokušajte koristiti najmanji tip podataka koji će zadovoljiti potrebnu informaciju. Na slici broj 2 prikazano je zauzeće memorije za pojedine tipove podataka.

| Data Types | Size in Bytes | Can contain: |
|---|---|---|
| boolean | 1 | true (1) or false (0) |
| char | 1 | ASCII character or signed value between -128 and 127 |
| unsigned char, byte, uint8_t | 1 | ASCII character or unsigned value between 0 and 255 |
| int, short | 2 | signed value between -32,768 and 32,767 |
| unsigned int, word, uint16_t | 2 | unsigned value between 0 and 65,535 |
| long | 4 | signed value between -2,147,483,648 and 2,147,483,647 |
| unsigned long, uint32_t | 4 | unsigned value between 0 and 4,294,967,295 |
| float, double | 4 | floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.) |

Slika broj 2 – Memorijsko zauzeće za pojedine tipove podataka

- *Upotreba lokalnih varijabli* - globalne i statičke promenljive se prve učitaju u SRAM memoriju i one pomeraju Heap gore prema Stacku. Ukoliko je moguće, incijalizujte te lokalne promenljive unutar funkcija. Naime, prilikom izvršavanja funkcije alocira se deo Stack memorije. Unutar te memorije biće sadržani svi parametri koji su dodeljeni funkciji kao i sve lokalne promenjive koje su deklarisane u toj funkciji. Svi ovi podaci koji se koriste u funkciji će se, prilikom završetka rada funkcije, izbaciti iz dele memorije Stack-a koji je funkcija koristila, a samim tim i smanjiti zauzeti memorijski prostor koji je Steak zauzimao.
- *Upotreba PROGMEM za const podatke* - u mnogim slučajevima, velika količina RAM-a je preuzeta od strane statičke memorije kao rezultat korišćenja globalne promenljive (kao što su *Strings* ili *Int*). Kada unapred znate, da se vrlo verovatno promenljiva neće promeniti, ona se jednostavno može smestiti u tzv. PROGMEM (programsku memoriju). Kao jednostavan primer je upotreba **F()** makro-a koji kaže kompajleru da se String smesti u PROGMEM-u.

**Primer**: ako zamenimo:

*Serial.println("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-am sram. Lovely sram! Lovely sram! Lovely sram! Lovely sram! Lovely sram! Sram sram sram sram!");*

sa:

*Serial.println(F("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-am sram. Lovely sram! Lovely sram! Lovely sram! Lovely sram! Lovely sram! Sram sram sram sram!"));*

sačuvaćemo 180 byte-ova SRAM memorije.

- *Ne koristite rekurzivne funkcije* - na sledećem primeru možemo videti kako se nepažljivim korišćenjem rekurzivne funkcije može napuniti Stack memorija i time izazvati mogući pad programa.

## Zadatak 1

*Vaš zadatak je da korišćenjem PlatformIO-a testirate navedeni programski kod i iskomentarišete šta se dogodilo.*

```
// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
uint16_t a = 2;
```

```
#define DELAY 1000 // wait for a predefined delay (in milliseconds)

void powerUp();
void powerDown();
int freeRam();
void incVar();

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  Serial.println("Program begins here!");
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  powerUp();
}

void powerUp() {
  Serial.print("Free SRAM: ");
  Serial.print(freeRam());
  Serial.println(" bytes");
  incVar();
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(DELAY);
  powerDown();
};

void powerDown() {
  Serial.print("Free SRAM: ");
  Serial.print(freeRam());
  Serial.print(" bytes");
  incVar();
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(DELAY);
  powerUp();

};

int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}

void incVar() {
  a = a + 2;
  Serial.print("a + 2 = ");
  Serial.println(a);
};
```

## Zadatak 2

U nastavku je dat programski kod-skripta ***TempHumLight.ino*** koji je skinut sa GitHub sajta koji treba testirati u IDE okruženju. Ovaj program čita temperaturu i vlagu senzora (DHT11 ili DHT22) te nivo osvetljenja (pomoću senzora BH1750). Pre pokretanja/kompajliranja skripte potrebno je instalirati biblioteke za senzor DHT i BH1750. Da biste to uradili potrebno je prvo da otvorimo novi terminal u PlatformIO-u te upišemo sledeću naredbu: **platformio lib search DHT**. Nakon toga biće prikazan popis

svih biblioteka koje možete instalirati. Instalirajte biblioteku pod rednim brojem [19] tako što ćete otkucati:

**platformio lib -g install 19** (pod brojem 19 je Adafruit biblioteka za DHT senzor)

Sličnu stvar treba ponoviti i za senzor BH1750:

**platformio lib search BH1750**

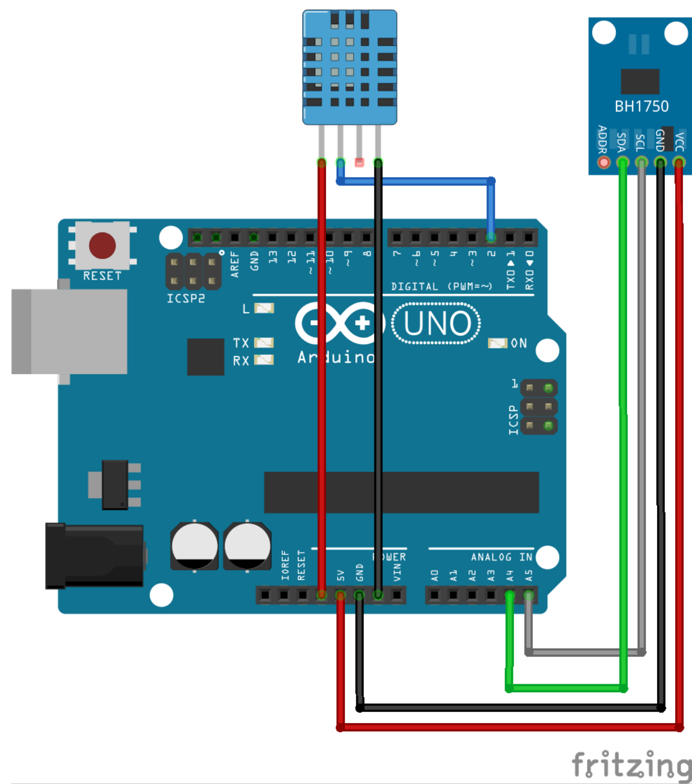
**platformio lib -g install 439** (redni broj biblioteke za senzor BH1750)

Nakon toga povežite DHT i BH1750 senzore kako je prikazano na slici broj 3 i testirajte navedeni kod. Vaš zadatak je da razumete programski kod i da pokušate da ga optimizirate na način da zauzme što manje SRAM-a korišćenjem gore navedenih uputstva.

```
/*
  BH1750 library usage example
  This example had some comments about advanced usage features.
  Connection:
    VCC -> 5V (3V3 on Arduino Due, Zero, MKR1000, etc)
    GND -> GND
    SCL -> SCL (A5 on Arduino Uno, Leonardo, etc or 21 on Mega and Due)
    SDA -> SDA (A4 on Arduino Uno, Leonardo, etc or 20 on Mega and Due)
    ADD -> GND or VCC (see below)
  ADD pin uses to set sensor I2C address. If it has voltage greater or equal to
  0.7VCC voltage (as example, you've connected it to VCC) - sensor address will be
  0x5C. In other case (if ADD voltage less than 0.7 * VCC) - sensor address will
  be 0x23 (by default).
*/
/*
 DHT11/22 library usage example
 Connect pin 1 (on the left) of the sensor to +5V or +3V3
 NOTE: If using a board with 3.3V logic like an Arduino Due connect pin 1
 to 3.3V instead of 5V!
 Connect pin 2 of the sensor to whatever your DHTPIN is
 Connect pin 4 (on the right) of the sensor to GROUND
 Connect a 10K resistor from pin 2 (data) to pin 1 (power) of the sensor
*/
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <Wire.h>
#include <BH1750.h>
#define DHTPIN 2     // what digital pin we're connected to
// Uncomment whatever type you're using!
//#define DHTTYPE DHT11   // DHT 11
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
//#define DHTTYPE DHT21   // DHT 21 (AM2301)
DHT dht(DHTPIN, DHTTYPE);
BH1750 lightMeter(0x23);
float hif;
float hic;
float h;
float t;
float f;
uint32_t lux;
```

```
void readTempHum();
void readLight();
int freeRam ();
void setup() {
  Serial.begin(9600);
  Serial.println("DHTxx test!");
  dht.begin();
  lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE);
  Serial.println(F("BH1750 Test"));
}
void loop() {
  readTempHum();
}
void readTempHum() {
  // Wait a few seconds between measurements.
  delay(2000);
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  f = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  // Compute heat index in Fahrenheit (the default)
  hif = dht.computeHeatIndex(f, h);
  // Compute heat index in Celsius (isFahreheit = false)
  hic = dht.computeHeatIndex(t, h, false);
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print(f);
  Serial.print(" *F\t");
  Serial.print("Heat index: ");
  Serial.print(hic);
  Serial.print(" *C ");
  Serial.print(hif);
  Serial.println(" *F");
  readLight();
}
void readLight() {
  lux = lightMeter.readLightLevel();
```

```
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  Serial.print("Free SRAM: ");
  Serial.print(freeRam());
  Serial.print(" bytes");
  readTempHum();
}
int freeRam ()
{
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```




Slika broj 3 Šema povezivanja senzora DHT i BH1750