ICM Summer of Code Day 3: Prekidi, prekidni potprogrami i rutine

U ovoj vježbi upoznat ćemo se s prekidima (engl. *interrupt*): glavnim mehaniznom "istovremenog izvođenja" operacija na Arduinu. Prekidi funkcioniraju na način da se, po specificiranom događaju (istekom vremena, promjena na digitalnom ulazu i slično) poziva funkcija koju sami specificiramo. Da bi se funkcija pozvala, potrebno je **samo jednom** (hint*:* funkcija *setup*) specificirati na koji će događaj koja funkcija pozivati. Funkciju koju pozivamo **mora** imati navedeni prototip:

```
void proizvoljnoImeFunkcije();
```

što ujedno znači da funkcija ne može primiti nikakve argumente. Ukoliko želimo u takvu funkciju prenjeti kakav kontekst, potrebno je to obaviti globalnim varijablama (postoje i napredniji mehanizmi, ali je u svrhu radionica i Arduino SoC projekata nepotrebno).

Kako bi si bolje predočili **što** rade prekidi, pogledajmo slijedeći isječak kôda:

```
void loop(){
    digitalWrite(LEDICA, HIGH);
    delay(500);
    digitalWrite(LEDICA, LOW);
    delay(500);
}
```

Pomislili bi da jedino što program radi jest pali i gasi ledicu s periodom od oko sekunde. Međutim, ukoliko je u funkciji *setup* podešeno da se prekidna rutina pali na npr. pritisak gumba, čak i ako je naš glavni dio programa jednostavan, dodatna funkcionalnost se transparentno može ostvariti prekidima.

U ovoj vježbi fokus će biti na prekide inicirane ponašanjem na digitalnom ulazu. U slijedećim vježbama, obradit ćemo i prekide inicirane istekom vremena, periodične prekide. Bitno je napomenuti kako svaki mikrokontroler ima točno specificirane pinove na kojima može inicirati prekid. Razvojna pločica Arduino Uno i Arduino Nano (one koje su vam priložene u radionici) prekide podržavaju inicirane na digitalnim pinovima 2 i 3. Pretragom po internetskim resursima s ključnim riječima "`arduino attachInterrupt`" možete saznati više informacija, a i specifičnosti za druge Aruduino pločice.

Također, bitno je napomenuti kako neke operacije nije moguće koristiti unutar funkcija pozivanima interruptom. Tokom vježbe spomenut ćemo nekoliko takvih funkcija, ali za početak proučite nekoliko bitnih funkcija koje ćete koristiti tokom ove vježbe.

```
attachInterrupt(digitalPinToInterrupt(pin), imePrekidneFunkcije, mod)

// gdje mod može biti: LOW, CHANGE, RISING, FALLING
```

Postavlja Arduino u mod gdje, u trenutku akcije navedene agumentom *mod*, prekida program i poziva funkciju koja je specificirana argumentom.

Primjer: ukoliko želimo po pritisku gumba pozvati funkciju `upaliLED()` (koju sami moramo napisati), a gumb je spojen na pin 2, u funkciji `setup()` pozvat ćemo:

```
attachInterrupt(digitalPinToInterrupt(2), upaliLED, FALLING)
// moguće je mod podesiti i u LOW, ali ne preporuča se. Zašto?
```

Imajte na umu da gornji poziv pretpostavlja da je gumb spojen u modu `INPUT_PULLUP`. To je **svakako potrebno** definirati **prije** postavljanja prekida!

Primjetite da je funkcija `upaliLED` navedena bez zagrada, što nije nikakav previd - tako to mora biti. Za one koji žele znati zašto: `upaliLED()` je **poziv funkcije**, a `upaliLED` u programskom jeziku C predstavlja **programsku adresu** (npr. 0xFFA00) te funkcije, pod uvijetom da je ona implementirana (zapravo, da je negdje definirana ili deklarirana, odnosno da je specificirano njezino ponašanje). Nije moguće imati dvije prekidne funkcije pozvane istovremeno za isti *pin*. Za potrebe mijenjanja prekidnih postavki, potrebno je poništiti prethodno definirano ponašanje funkcijom:

```
detachInterrupt(digitalPinToInterrupt(pin))
```

Gdje je argument *pin* jednak onom kojem smo definirali ponašanje `attachInterrupt()` funkcijom.

Općenito, moguće je globalno zabraniti ili dopustiti prekide, za to postoje dvije funkcije koje ne zahtjevaju argumente:

1. `interrupts()`
2. `noInterrupts()`

Dobra je praksa kao prvu naredbu u prekidnom potprogramu zabraniti sve ostale prekide, a kao zadnju naredbu izvršiti naredbu dozvole ostalih prekida. Zašto?

Funkcija `digitalPinToInterrupt()` u kontekstu radionice nije bitna, ona je prisutna iz povijesnih razloga na Arduino platformama, i koriste se gotovo isključivo u gore navedenim slučajevima.

Za sva pitanja obratite se voditelju radionice. Kada svladate gore navedeno gradivo (obično je dovoljno proučiti službene arduino stranice) prebacite se na zadatke. Nadalje, prije nego se bacite na zadatke, uputno je ponoviti i gradivo o *pullup* i *pulldown* spojevima.

Zadatak 1: bez korištenja vanjskog otpornika (kao *pullup* ili *pulldown*, radi jednostavnosti), napravite program čija je prekidna rutina slijedeća:

```
void paliLED(){
     digitalWrite(LED_BUILTIN, HIGH);
     while(1);
}

// u setup je potrebno dodati i slijedeću liniju koda:
// pinMode(LED_BUILTIN, OUTPUT);
```

Zadatak 2: umjesto trajne while petlje, u prekidnu rutinu dodajte funkciju *delay* s jednom sekundom. Ta funkcija, uz micros, millis, i bilo kakvom funkcionalnošću ispisa na seriju, su neke od funkcija koje nećeraditi u prekidnim rutinama.

Zadatak 3: pomoću jedne prekidne rutine, implementirajte program koji će na pritisak gumba paliti i gasiti LED diodu. Stanje LED-ice pohranjujte korištenjem globalne varijable. Pritom za definiciju te globalne varijable koristite ključnu riječ *volatile* i prikladno je iskoristite. Raspravite s voditeljem radionice o njezinom značenju.

Zadatak 4: isti program implementirajte pomoću dvije prekidne rutine.

Zadatak 5: probajte u prekidnoj rutini ispisati ispis na serijski monitor.

Zadatak 6: Implementirajte igru s jedne od prošlih radionica:

Napravi program koji će, nakon paljenja programa, po isteku slučajnog broja sekundi između 3 i 9, upaliti LED na pinu 13. Po paljenju LED-ice na pinu 13, dva korisnika moraju stisnuti gumb čim je prije moguće (svaki korisnik ima svoj gumb). Program ispisuje pobjednika u igri. Ukoliko jedan od igrača pritisne dugme prije paljenja LED-ice, ispisuje se da je korisnik gubitnik i igra završava.

Po potrebi modificirajte program na način prikladan korištenju pomoću prekidnih rutina.

Zadatak 7: Komunikacijom globalnim varijablama implementirajte brojač klikova u sekundi. Ispišite taj broj preko serije u glavnom programu.

Timeri

Kako je već prije navedeno, prekidi ne moraju biti pozivani isključivo vanjskim događajima: prekidi se mogu pozivati periodički uz definiran period: koncept Timera. Arduino Uno i Nano imaju 3 hardverska sklopa za generiranje prekida Timerima. Dostupni timeri su poimence: Timer0, Timer1 i Timer2. Imajte na umu da razne funkcije ili biblioteke već koriste dostupne timere (npr. AnalogWrite, PulseIn, PWM library, Servo i slično) te da Timeri ujedno interferiraju s fizičkim prekidima, pa je i u tom smislu potrebno voditi brigu i pomno istestirati program i proučiti o mogućnostima “interferencija” raznih mogućnosti Arduina koje koristite. U okviru ove radionice neće detaljno biti izloženi takvi slučajevi, ali je bitno da ih imate na umu ne bi li se kasnije mogli snaći u izradi Arduino projekta - uglavnom ćete se oslanjati na dostupne asistente i internet o mogućim problemima.

U okviru zadatka biti će korištern Timer1, za kojeg je potrebno skinuti vanjsku biblioteku: upitajte voditelja radionice kako to napraviti. Dvije funkcije će vam biti potrebne (potrebno ih je pozivati samo jednom od paljenja Arduina):

```
 Timer1.initialize(period_u_milisekundama);
 Timer1.attachInterrupt(imeFunkcije);
```

```
Te po potrebi:
```

```
Timer1.detachInterrupt();
```

Zadatak 7: Nadogradite 3. i 4. Zadatak na način na prekidne rutine zovete jednom u sekundi.

Zadatak 8: U glavnom programu (loop) ispisujte na Serial monitor vrijednost napona na potenciometru, svakih 100ms, dok periodički palite i gasite LED diodu kao iz zadatka 7. (nadogradite 7. zadatak...).

EEPROM:

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. 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.

Detalje kako programski raditi s EEPROM-om obraditi ćemo tek na sutrašnjoj radionici, a oni brzi mogu već sada proučavati kako funkcionira EEPROM, na linku:

https://www.arduino.cc/en/Reference/EEPROM

Zadatak 9: 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.

Zadatak 10: 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;
```