Vežba 4 – Zadatak za vežbu

Zadatak 1. Kreirati direktorijum *Vezba4/Zadatak1* u vašem RSZDMK repozitorijumu na GitHub-u u okviru kog je potrebno sačuvati Eclipse projekat. Za početak, u okviru projekta, potrebno je kreirati biblioteku *logic_utils*, koju sačinjavaju dve datoteke, zaglavlje *logic_utils.h* i izvorna datoteka *logic_utils.c*. Biblioteka sadrži pomoćne funkcije koje implementiraju logičke operacije nad bitima neke promenljive. U nastavku zadatka, ove funkcije će biti korišćene prilikom realizacije drugih biblioteka.

Zaglavlje *logic_utils.h* sadrži deklaracije sledećih funkcija:

- `unsigned int SetBit(unsigned int reg, unsigned char bit_num);`
  - **Opis**: vrši postavljanje bita na vrednost 1, na poziciji *bit_num*, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: modifikovana vrednost promenljive *reg*
- `unsigned int ClearBit(unsigned int reg, unsigned char bit_num);`
  - **Opis**: vrši postavljanje bita na vrednost 0, na poziciji *bit_num*, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: modifikovana vrednost promenljive *reg*
- `unsigned int ToggleBit(unsigned int reg, unsigned char bit_num);`
  - **Opis**: vrši postavljanje bita na suprotnu vrednost od postojeće, na poziciji *bit_num*, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: modifikovana vrednost promenljive *reg*
- `unsigned int CheckBit(unsigned int reg, unsigned char bit_num);`
  - **Opis**: vrši proveru vrednosti bita, na poziciji *bit_num*, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: vrednost bita na datoj poziciji (0 ili 1)
- `unsigned char BitmaskSet(unsigned char reg, unsigned char mask);`
  - **Opis**: vrši postavljanje grupe bita na vrednost 1, specificiranih u promenljivoj `mask`, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: modifikovana vrednost promenljive *reg*
- `unsigned char BitmaskClear(unsigned char reg, unsigned char mask);`
  - **Opis**: vrši postavljanje grupe bita na vrednost 0, specificiranih u promenljivoj `mask`, u okviru 16-bitne promenljive *reg*
  - **Povratna vrednost**: modifikovana vrednost promenljive *reg*
- `unsigned int Not(unsigned int input);`
  - **Opis**: vrši bitsku negaciju 16-bitne promenljive `input`

  - **Povratna vrednost**: rezultat operacije
- `unsigned int And(unsigned int input1, unsigned int input2);`
  - **Opis**: vrši bitsku *i* operaciju nad 16-bitnim promenljivima `input1` i `input2`
  - **Povratna vrednost**: rezultat operacije
- `unsigned int Or(unsigned int input1, unsigned int input2);`
  - **Opis**: vrši bitsku *ili* operaciju nad 16-bitnim promenljivima `input1` i `input2`
  - **Povratna vrednost**: rezultat operacije
- `unsigned int Xor(unsigned int input1, unsigned int input2);`
  - **Opis**: vrši bitsku *eks-ili* operaciju nad 16-bitnim promenljivima `input1` i `input2`
  - **Povratna vrednost**: rezultat operacije
- `unsigned int ShiftLeft(unsigned int input, unsigned int num_of_shifts);`
  - **Opis**: vrši pomeranje (*eng. shift*) ulazne promenljive `input` u levo za `num_of_shifts` mesta.
  - **Povratna vrednost**: rezultat operacije
- `unsigned int ShiftRight(unsigned int input, unsigned int num_of_shifts);`
  - **Opis**: vrši pomeranje (*eng. shift*) ulazne promenljive `input` u desno za `num_of_shifts` mesta.
  - **Povratna vrednost**: rezultat operacije

U okviru izvorne datoteke *logic_utils.c* potrebno je implementirati sve navedene funkcije. Takođe, sav kod je potrebno dokumentovati koristeći *Doxygen*. Nakon toga, potrebno je napraviti biblioteku *adc_utils*, koju sačinjavaju dve datoteke, zaglavlje *adc_utils.h* i izvorna datoteka *adc_utils.c*. U okviru ove biblioteke, potrebno je implementirati osnovne funkcije za rad sa AD (analogno-digitalnim) konvertorom AVR mikrokontrolera. Kratak teorijski uvod o AD konverziji i registrima koji su nam neophodni za rad sa ADC modulom je dat u nastavku.

Analogno-digitalna konverzija kod AVR mikrokontrolera

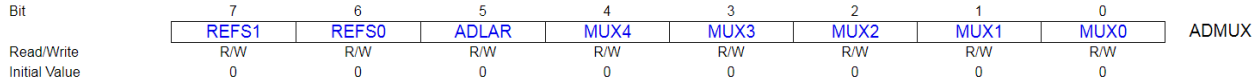
U praktičnim aplikacijama često se javlja potreba za očitavanjem različitih vrednosti (temperatura, pritisak, vlažnost, napon itd.) u analognom formatu. Međutim, obrada takvih podataka prilično neefikasna u pogledu brzine. Takođe, mikroprocesorski sistemi, ne mogu da obrađuju ovakvu vrstu podataka. Kako bi se ovo prevazišlo, koristi se postupak, poznat pod nazivom Analogno-digitalna konverzija. Analogno-digitalna konverzija predstavlja proces pretvaranja analognog signala u neku digitalnu vrednost (ceo broj). Za realizaciju ovog procesa koristi se modul unutar AVR mikrokontrolera koji se naziva AD (analogno-digitalni) konvertor. Vrednost analognog signala je moguće čitati sa jednog od 6 analognih ulaza koji se nalaze na Arduino UNO ploči. Što se samog ADC modula tiče, on poseduje 10-bitnu rezoluciju (na izlazu,

modul može da prikaže vrednosti iz intervala 0-1023) sa preciznošću od +/- 2 LSB. Napon na ulazu ADC modula može da uzme vrednost iz intervala 0-V_{cc}. Takođe, ADC modul poseduje tri različita moda rada: *free running*, *single* i *interrupt based conversion*. U okviru ovog zadatka, ADC modul će biti konfigurisan tako, da radi *single conversion* modu.
Registri neophodni za rad sa ADC modulom su:

- **ADMUX** – ADC Multiplexer Selection Register
- **ADCSRA** – ADC Control and Status Register A
- **ADC** – ADC Data Register

ADMUX registar

Struktura ADMUX registra data je na sledećoj slici.



Slika 1. ADMUX registar

Uloga pojedinačnih bita u okviru ovog registra je sledeća:

- Bit[7:6] – REFS1:0 (*Reference Selection Bits*) – ovi biti služe za izbor referentnog napona na osnovu kog ADC modul radi, u skladu sa sledećom tabelom.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC
1	0	Reserved
1	1	Internal 2.56V Voltage Reference

  U okviru ovog zadatka potrebno je koristiti napon napajanja kao referentni napon, odnosno vrednost V_{cc}.
- Bit[5] – ADLAR (*ADC Left Adjust Result*) – ukoliko je postavljen na 1, vrši se levo poravnjanje rezultata konverzije sa ciljem da sam rezultat bude 8-bitan. Prednost korišćenja ovakvog jeste veća brzina rada, dok je mana smanjena rezolucija. U okviru ovog zadatka, podrazumevati da je ovaj bit postavljen na 0.
- Bit[4:0] – MUX4:0 (*Analog Channel and Gain Selection Bits*) – pomoću ovih bita bira se jedan od ulaznih analognih kanala sa kojih je potrebno očitati vrednost, u skladu sa sledećom tabelom.

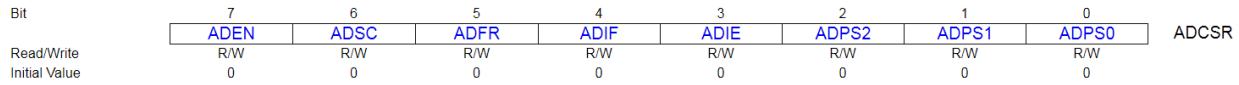
MUX4:0	Input
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3

00100	ADC4
00101	ADC5

Ostale vrednosti bita MUX4:0, za izradu ovog zadatka, nisu od interesa.

ADCSR registar

Struktura ADCSR registra data je na sledećoj slici.



Slika 2. ADCSR registar

Uloga pojedinačnih bita u okviru ovog registra je sledeća:

- Bit[7] – ADEN (*ADC Enable*) – predstavlja bit dozvole za ADC modul. Ukoliko nije postavljen na "1", nijednu ADC operaciju nije moguće izvršiti.
- Bit[6] – ADSC (*ADC Start Conversion*) – predstavlja start bit za početak konverzije. Kako bi se proces konverzije započeo potrebno je postaviti ovaj bit na "1". Kada je konverzija uspešno završena, ovaj bit se *automatski* ponovo postavlja na "0". Stoga, za narednu konverziju, ponovo je potrebno njegovo postavljanje na vrednost "1".
- Bit[5] – ADFR (*ADC Free Running Select*) – kada je postavljen na "1", aktivira se *free running* mod rada ADC modula, u okviru kog modul neprekidno vrši konverziju vrednosti sa ulaza. Kao što je već rečeno, ovaj mod rada *neće* biti korišćen u okviru ovog zadatka.
- Bit[4] – ADIF (*ADC Interrupt Flag*) – kada je konverzija uspešno izvršena, ovaj bit se automatski postavlja na vrednost "1". Prema tome, na osnovu njega je moguće vršiti proveru da li je konverzija završena ili ne.
- Bit[3] – ADIE (*ADC Interrupt Enable*) – kada je ovaj bit postavljen na vrednost "1"', omogućen je *interrupt based* mod rada ADC modula. Takođe, kao što je već napomenuto, ovaj mod rada *neće* biti potreban u okviru ovog zadatka.
- Bit[2:0] – ADPS2:0 (*ADC Prescaler Select Bits*) – ovi biti služe sa podešavanje preskalera (faktora skaliranja taktnog signala na kom ADC modul radi), u skladu sa sledećom tabelom.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32

1	1	0	64
1	1	1	128

Objašnjenje preskalera: ADC modul izvršava semplovanje signala sa ulaza u određenim vremenskim intervalima. U opštem slučaju ADC modul radi na frekvenciji između 50kHz i 200kHz. Budući da je frekvencija na kojoj radi sistem značajno veća (16MHz), potrebno je izvršiti skaliranje frekvencije. Na primer za faktor skaliranja 64 dobija se sledeća vrednost frekvencije ADC modula:

$$F_{ADC} = \frac{F_{MCU}}{64} = \frac{16M}{64} Hz = 250kHz$$

Pitanje koje se postavlja jeste, koji faktor skaliranja izabrati? Odgovor je – u zavisnosti od date situacije, budući da je uvek potrebno tražiti kompromis između frekvencije (brzine) i preciznosti konverzije. Prema tome, veća frekvencija znači manju preciznost i obrnuto. Kako u okviru ovog zadatka nije potrebno postići visoku preciznost, prilikom testiranja potrebno je izabrati najveći mogući faktor skaliranja (= 128).

ADC registar

Predstavlja 16-bitni registar u okviru kog se smešta 10-bitni rezultat konverzije. Zapravo, rezultat se smešta u registre ADCL – donjih 8-bita i ADCH – gornjih 2-bita (u slučaju da je ADLAR bit u okviru ADMUX registar postavljen na "0"), odnosno obrnuto (u slučaju da je ADLAR bit u okviru ADMUX registar postavljen na "1"), ali, kako bi se olakšalo korišćenje rezultata od strane korisnika, interno su ove vrednosti formatirane i smeštene u 16-bitni ADC registar (pri čemu se podrazumeva da je ADLAR bit postavljen na "0").

Napomena: *Za vežbu, pokušati "ručno" konstruisati (formatirati) rezultat konverzije na osnovu vrednosti ADCL i ADCH registara (podrazumeva se da je ADLR bit postavljen na vrednost "0") u okviru funkcije za očitavanje rezultata konverzije.*

Prema tome, prilikom inicijalizacije ADC modula potrebno je izvršiti sledeći niz koraka:

1. izabrati vrednost referentnog napona
2. izabrati faktor skaliranja
3. izabrati *single conversion* mod rada ADC modula
4. postaviti bit dozvole

Prilikom očitavanja vrednosti sa izlaza ADC modula potrebno je izvršiti sledeći niz koraka:

1. postaviti analogni kanal za čitanje
2. izvršiti pokretanje konverzije
3. sačekati dok se konverzija ne izvrši
4. očitati vrednost rezultata

Na osnovu svega navedenog, potrebno je implementirati funkcije za rad sa AD konvertorom. Sve logičke operacije koje se koriste u definicijama ovih funkcija, potrebno je realizovati korišćenjem gotovih funkcija iz biblioteke *logic_utils*. Zaglavlje *adc_utils.h* sadrži deklaracije sledećih funkcija:

- `void InitADC(unsigned char reference, unsigned char division_factor);`
  - **Opis**: vrši inicijalizaciju AD konvertora na osnovu prosleđenih parametara (referentni napon i faktor skaliranja).
  - Povratna vrednost: nema povratnu vrednost
- `unsigned int ReadADC(unsigned char channel);`
  - **Opis**: vrši inicijalizaciju AD konvertora na osnovu prosleđenih parametara (referentni napon i faktor skaliranja).
  - **Povratna vrednost**: očitana vrednost sa izlaza AD konvertora
- `void SetVref(unsigned char reference);`
  - **Opis**: vrši postavljanje referentnog napona (u okviru ADMUX registra) na osnovu parametra `reference`.
  - **Povratna vrednost**: nema povratnu vrednost
- `void SetPrescaler(unsigned char division_factor);`
  - **Opis**: vrši postavljanje faktora skaliranja (u okviru ADCSRA registra) na osnovu parametra `division_factor`.
  - **Povratna vrednost**: nema povratnu vrednost
- `void SetEnable(unsigned char enable);`
  - **Opis**: vrši postavljanje dozvole rada AD konvertora (u okviru ADCSRA registra) na osnovu parametra `enable`.
  - **Povratna vrednost**: nema povratnu vrednost
- `void SetChannel(unsigned char channel);`
  - **Opis**: vrši izbor analognog ulaznog kanala AD konvertora (u okviru ADMUX registra) na osnovu parametra `channel`.
  - **Povratna vrednost**: nema povratnu vrednost
- `void RunConversion();`
  - **Opis**: implementira izvršavanje AD konverzije, odnosno njeno pokretanje i čekanje na njen završetak.
  - **Povratna vrednost**: nema povratnu vrednost

U okviru izvorne datoteke *adc_utils.c* potrebno je implementirati sve navedene funkcije. Ponovo, sav kod je potrebno dokumentovati koristeći *Doxygen*.

Nakon realizacije ove biblioteke, potrebno je realizovati novu biblioteku koja implementira *generator pseudoslučajnih brojeva* pomoću *Linear-Feedback Shift* registara (skraćeno *LFSR*) u dve varijante. Stoga, sledeći korak podrazumeva pravljenje dve grane na Git-u: *branch_many_to_one* i *branch_one_to_many*. Na grani *branch_many_to_one* potrebno je

napraviti biblioteku *rand_mto*, koja se sastoji od dve datoteke, zaglavlja *rand_mto.h* i izvorne datoteka *rand_mto.c*. Ova biblioteka implementira generator pseudoslučajnih brojeva upotrebom 16-bitnog *many-to-one LFSR*-a. Slično, na grani *branch_one_to_many* potrebno je napraviti biblioteku *rand_otm*, koja se, takođe, sastoji od dve datoteke, zaglavlja *rand_otm.h* i izvorne datoteka *rand_otm.c*. Ova biblioteka implementira generator pseudoslučajnih brojeva upotrebom 16-bitnog *one-to-many LFSR*-a.
Teorijsko objašnjenje obe varijante registara je dato u nastavku.

Linear-Feedback Shift Registri

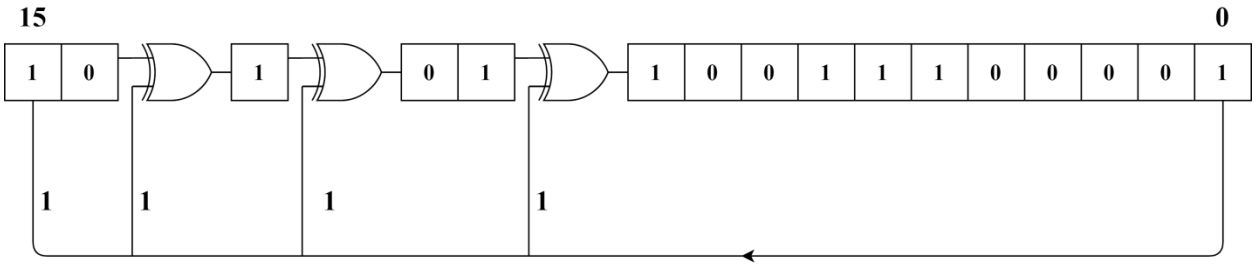
Linear-feedback shift registri (skraćeno LFSR) predstavljaju posebnu klasu pomeračkih registara, čije naredno stanje (ili samo jedan bit) je linearna funkcija prethodnog stanja. Najčešće korišćena operacija za realizaciju ovakvih funkcija je *eks-ili* operacija. Stoga, LFSR je najčešće pomerački registar čije naredno stanje ili samo jedan bit (u zavisnosti od toga da li posmatramo *one to many* LFSR ili *many to one* LFSR) je generisano korišćenjem isključivo *eks-ili* kola.
Inicijalno stanje LFSR-a se naziva *seed*. Budući da je svako naredno stanje određeno prethodnim, registar sadrži konačan skup mogućih stanja, što znači da u krajnjem slučaju može da se vrati na početak (i ponovo počne prolazak kroz ista stanja). Broj stanja između dva ponovljenja stanja (početne i krajnje tačke) se naziva *period* LFSR-a.
Međutim, uz dobar odabir funkcije za generisanje narednog stanja, moguće je ostvariti prilično velike periode i time generisati stanja u naizgled slučajnom redosledu. Iz tog razloga se LFSR često koriste kao generatori pseudoslučajnih brojeva u situacijama u kojima nam generisanje slučajnih brojeva visoke entropije nije od presudnog značaja, budući da je njegova struktura prilično jednostavna i brzina generisanja narednog slučajnog broja izrazito velika, uzimajući u obzir da koristi jednostavnu *eks-ili* operaciju.

Razlikujemo dve vrste LFS registara:

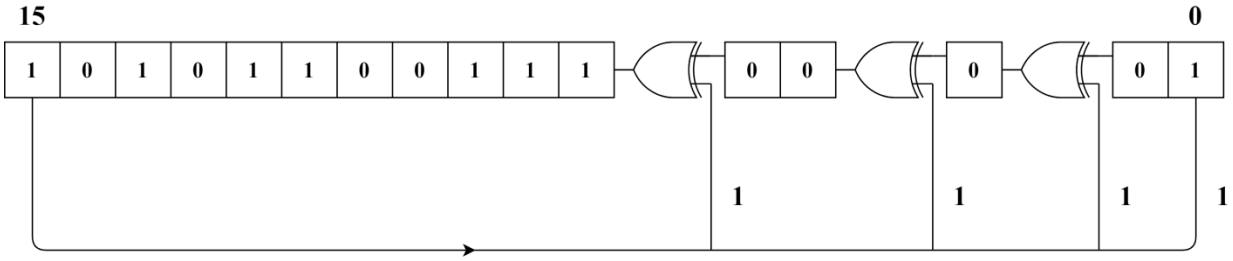
1. *One-to-many* – kod kojih se na osnovu jednog bita trenutnog stanja LFSR-a generiše više različitih bita u okviru narednog stanja LFSR-a. Primer poznatog *one-to-many* registra jeste LFSR u konfiguraciji Galoa.
2. *Many-to-one* – kod kojih se na osnovu više različitih bita u okviru trenutnog stanja LFSR-a generiše jedan bit narednog stanja. Primer *one-to-many* registra je Fibonačijev LFSR.

One-to-many LFSR

Na sledećim slikama prikazan je 16-bitni LFSR u konfiguraciji Galoa u desnoj i levoj orijentaciji.



Slika 3. LFSR u konfiguraciji Galoa u desnoj orijentaciji

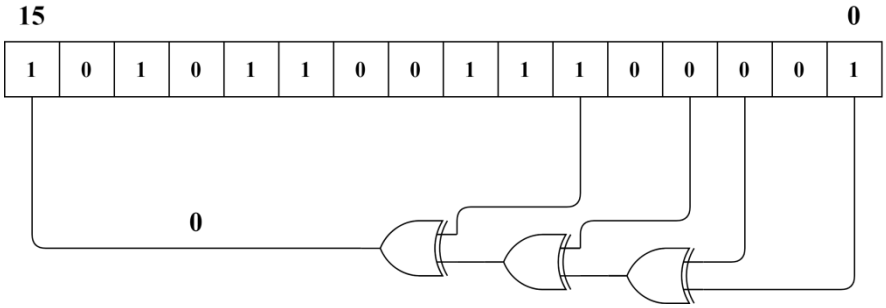


Slika 4. LFSR u konfiguraciji Galoa u levoj orijentaciji

Kao što se može primetiti, jedan bit (LSB ili MSB u zavisnosti od orijentacije) trenutnog stanja registra učestvuje u generisanju 15, 13, 12 i 10 bita narednog stanja registra u desnoj orijentaciji, odnosno 5,3,2 i 0 bita narednog stanja registra u levoj orijentaciji. Maskimalan period ponavljanja ovako konstruisanog registra je 65535 (vrednosti pseudoslučajnih brojeva). Za stanje na slici 3. (0xACE1) naredno generisano stanje imaće vrednost 0xE270.
U okviru implementacije ovog LFSR-a, potrebno je omogućiti izbor orijentacije (ovaj zahtev naveden je i u specifikaciji funkcija koje je potrebno implementirati, niže u tekstu).

Many-to-one LFSR

Na sledećoj slici prikazan je 16-bitni Fibonačijev LFSR.



Slika 5. Fibonačijev LFSR

Kao što se sa slike može videti jedina operacija koja se koristi je *eks-ili* operacija, gde vrednosti 0, 2, 3 i 5 bita trenutnog stanja učestvuju u generisanju vrednosti 15 bita narednog stanja. Naredno stanje se formira na osnovu generisanog bita i pomerene vrednosti prethodnog stanja.

Maskimalan period ponavljanja ovako konstruisanog registra je, takođe, 65535 (vrednosti pseudoslučajnih brojeva). Za stanje na slici 5. (0xACE1) naredno generisano stanje imaće vrednost 0x5670.
Na osnovu ovog objašnjenja, potrebno je implementirati funkcije za rad sa generatorom pseudoslučajnih brojeva, u oba slučaja. Sve logičke operacije koje se koriste u definicijama ovih funkcija, potrebno je realizovati korišćenjem gotovih funkcija iz biblioteke *logic_utils*.
Zaglavlje *rand_mto.h*, na grani *branch_many_to_one*, sadrži deklaracije sledećih funkcija:

- `void InitRand(unsigned int seed);`
  - **Opis**: vrši inicijalizaciju početnog stanja generatora na osnovu parametra `seed`.
  - **Povratna vrednost**: nema povratnu vrednost
- `unsigned int Rand();`
  - **Opis**: vrši generisanje 16-bitnog pseduoslučajnog broja.
  - **Povratna vrednost**: generisana 16-bitna vrednost
- `unsigned int RandRange(unsigned int min, unsigned int max);`
  - **Opis**: vrši generisanje 16-bitnog pseduoslučajnog broja iz intervala `min-max`.
  - **Povratna vrednost**: generisana 16-bitna vrednost
- `unsigned char GenerateNextBit(unsigned int current_state);`
  - **Opis**: vrši generisanje naredne vrednost MSB, na osnovu trenutnog stanja registra.
  - **Povratna vrednost**: generisana vrednost MSB
- `unsigned int UpdateState(unsigned int state, unsigned char bit);`
  - **Opis**: vrši generisanje novog stanja registra na osnovu trenutnog stanja registra i nove vrednosti MSB.
  - **Povratna vrednost**: generisano novo 16-bitno stanje registra

U okviru izvorne datoteke *rand_mto.c* potrebno je implementirati sve navedene funkcije.
Takođe, sav kod je potrebno dokumentovati koristeći *Doxygen*.
Zaglavlje *rand_otm.h*, na grani *branch_one_to_many*, sadrži deklaracije sledećih funkcija:

- `void InitRand(unsigned int seed);`
  - **Opis**: vrši inicijalizaciju početnog stanja generatora na osnovu parametra `seed`.
  - **Povratna vrednost**: nema povratnu vrednost
- `void SetDirection(unsigned char dir);`
  - **Opis**: vrši postavljanje orijentacije – leve ili desne (na ulazu se očekuje 0 ili 1 za desnu, odnosno levu orijentaciju, respektivno)
  - **Povratna vrednost**: nema povratnu vrednost
- `unsigned int Rand();`
  - **Opis**: vrši generisanje 16-bitnog pseduoslučajnog broja.
  - **Povratna vrednost**: generisana 16-bitna vrednost
- `unsigned int RandRange(unsigned int min, unsigned int max);`
  - **Opis**: vrši generisanje 16-bitnog pseduoslučajnog broja iz intervala `min-max`.

  - **Povratna vrednost**: generisana 16-bitna vrednost
- `unsigned char GetLSB(unsigned int state);`
  - **Opis**: vrši izdvajanje vrednosti LSB trenutnog stanja registra.
  - **Povratna vrednost**: izdvojena vrednost LSB
- `unsigned char GetMSB(unsigned int state);`
  - **Opis**: vrši izdvajanje vrednosti MSB trenutnog stanja registra.
  - **Povratna vrednost**: izdvojena vrednost MSB
- `unsigned int ShiftAndToggle(unsigned int state, unsigned char bit);`
  - **Opis**: vrši generisanje novog stanja na osnovu trenutnog stanja registra i nove vrednosti LSB(u slučaju desne orijentacije) i MSB(u slučaju leve orijentacije).
  - **Povratna vrednost**: nema povratnu vrednost

U okviru izvorne datoteke *rand_otm.c* potrebno je implementirati sve navedene funkcije. Slično, sav kod je potrebno dokumentovati koristeći *Doxygen*.

Konačno, u okviru *main.c* datoteke (*na obe grane*), potrebno je testirati rad AD konvertora i realizovanih generatora pseudoslučajnih brojeva. Prvo, potrebno je očitati vrednost sa proizvoljnog analognog ulaza i dobijenu vrednost iskoristiti kao inicijalnu vrednost (eng. *seed*) za generator pseudoslučajnih brojeva. Nakon toga, potrebno je generisati broj iz intervala **2-8**. Dobijena vrednost predstavlja broj ponavljanja treptanja diode. Vremenski interval tokom kog dioda svetli, odnosno ne svetli, je *600ms* (*tj. poluperioda treptanja iznosi 300ms*). Za realizaciju treptanja diode, koristiti gotove funkcije iz biblioteke ***pin***. Po potrebi, koristiti i gotove funkcije iz biblioteke ***pulsing***. Za realizaciju kašnjenja, koristiti gotove funkcije iz biblioteke ***timer_0***. Nakon odgovarajućeg broja ponavljanja, potrebno je napraviti pauzu od *2s*, nakon čega se postupak ponavlja.