Poglavlje 5

Organizacija projekata sa više izvornih datoteka

5.1 Uvod

U svim dosadašnjim primerima, razvijane su jednostavnije aplikacije koje su realizovane pisanjem izvornog koda koji je smešten u jednu izvornu (*.c*) datoteku. U slučaju složenijih aplikacija, programski kod se deli u više logičkih celina koje pripadaju različitim izvornim datotekama. Tipičan primer programskih logičkih celina su biblioteke sa drajverskim rutinama za različite tipove periferijskih uređaja. Na ovaj način postiže se bolja preglednost koda, kao i mogućnost ponovnog korišćenja jednom napisanog koda u više različitih projekata (eng. *code reuse*). Izvorni kod napisan u programskom jeziku C smešta se u dva tipa datoteka:

- ***.h* datoteke** sadrže deklaracije (prototipove) funkcija, makro konstante, makro funkcije, konstante, definicije tipova podataka korišćenih prilikom implementacije funkcija i sl.
- ***.c* datoteke** sadrže definicije (implementacije) funkcija deklarisanih u okviru .h datoteke, kao i varijabli korišćenih u okviru implementacije.

U nastavku je prikazan primer jednostavnog, dobro organizovanog, projekta koji sadrži ukupno pet izvornih datoteka:

- *main.c* (sadrži glavni program, odnosno funkciju *main*)
- *addition.h*
- *addition.c*
- *subtraction.h*
- *subtraction.c*

Sadržaj ovih datoteka je prikazan na listinzima koji slede u nastavku.

```
#include <stdint.h>
#include "addition.h"
#include "subtraction.h"

void main()
{
    int16_t a;
    a  = add(1, subtract(4, 2));

    while(1)
    ;
}
```

Listing 5.1: Datoteka main.c

```
int16_t add(int16_t a, int16_t b);
```

Listing 5.2: Datoteka addition.h

```
#include "addition.h"

int16_t add(int16_t a, int16_t b)
{
    return (a + b);
}
```

Listing 5.3: Datoteka addition.c

```
int16_t subtract(int16_t a, int16_t b);
```

Listing 5.4: Datoteka subtraction.h

```
#include "subtraction.h"

int16_t subtract(int16_t a, int16_t b)
{
    return (a - b);
}
```

Listing 5.5: Datoteka subtraction.c

Kao što se može uočiti sa prethodnih listinga, čitav projekat je dekomponovan na tri logičke celine: deo koji implementira sabiranje dve vrednosti, deo koji implementira oduzimanje dve vrednosti i glavni deo koji izvršava odgovarajuću složenu operaciju.

U nastavku je objašnjen postupak dekompozicije projekta na zasebne logičke celine i koncept pisanja biblioteka, na primeru programa koji se izvršava na mikrokontrolerskoj platformi.

5.2 Organizacija projekta i koncept pisanja biblioteka

Razvojni put projekta koji se sastoji od više biblioteka biće ilustrovan na primeru programske podrške upravljanju pinovima i tajmer/brojač modulom 0. Na kraju ovog poglavlja je dat kod programa koji implementira naizmenično treperenje diode različitim brzinama.

Kod je napisan unutar jedne datoteke, gde su definisane sve makro konstante, promenljive i funkcije koje su upotrebljene prilikom realizacije programa. Kako se broj makro konstanti, promenljivih, funkcija i komentara koji ih opisuju povećava, preglednost i ogranizacija koda se smanjuje. U cilju poboljšanja organizacije koda potrebno je sve elemente grupisati u logičke celine koje će biti organizovane unutar posebnih datoteka. Primer procesa disperzije elemenata koda u različite datoteke je prikazan u nastavku.

Datoteka se sastoji od sledećih funkcija:

- pinPulsing
- pinPulse
- pinSetValue
- pinInit
- timer0Init
- timer0Millis
- timer0DelayMs
- ISR(TIMER0_COMPA_vect)
- calculateHalfPeriod

i makro konstanti:

- `HIGH`
- `LOW`
- `OUTPUT`
- `INPUT`
- `PORT_B`
- `PORT_C`
- `PORT_D`
- `DIODE_PIN`
- `FAST`
- `SLOW`
- `FAST_REPS`
- `SLOW_REPS`

Na početku, potrebno je primetiti osnovne, nezavisne elemente od kojih se sastoji program. To su, pre svega, elementi koji služe za inicijalizaciju i osnovne manipulacije nad komponentama koje se koriste. Na osnovu ovoga, možemo pretpostaviti da su nam potrebne dve biblioteke, za upravljanje pinovima i tajmerom 0.

Biblioteka za upravljanje pinovima je implementirana unutar dve datoteke, a to su zaglavlje *pin.h* i izvorna datoteka *pin.c*. Za inicijalizaciju pinova, koristi se funkcija *pinInit()*, dok se za osnovnu manipulaciju proizvoljnim pinom koristi funkcija *pinSetValue()*. Stoga, ove dve funkcije je potrebno realizovati unutar biblioteke **pin**, gde je takođe potrebno definisati odgovarajuće vrednosti upotrebom makro konstanti. Makro konstante koje se koriste prilikom realizacije i upotrebe ovih funkcija su `HIGH`, `LOW`, `OUTPUT`, `INPUT`, `PORT_B`, `PORT_C` i `PORT_D`.

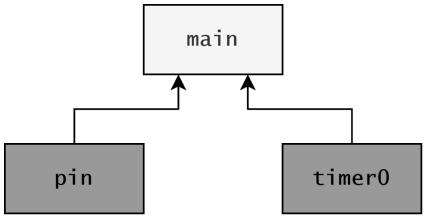
Sa druge strane, prilikom implementacije biblioteke za manipulaciju tajmer/brojačkim modulom 0, **timer0**, potrebno je kreirati zaglavlje *timer0.h* i izvornu datoteku *timer0.c*. Ova biblioteka će, kao svoj sadržaj, posedovati funkciju za inicijalizaciju modula – *timer0Init()*, funkciju koja vodi računa o broju milisekundi, proteklom od početka merenja vremena – *timer0Millis()*, funkciju za implementaciju pauze – *timer0DelayMs()* i odgovarajuću prekidnu rutinu u okviru koje se uvećava interni brojač proteklih milisekundi – *ISR(TIMER0_COMPA_vect)*. Kako se za implementaciju i izvršavanje ovih funkcija ne upotrebljava niti jedna makro konstanta, ova biblioteka neće posedovati niti jednu definiciju konstante pomoću makroa.

Celokupan koncept ove biblioteke se zasniva na promenljivoj *volatile uint32_t ms*, čija se uloga ogleda u skladištenju broja milisekundi proteklih od trenutka pokretanja programa. Prilikom definisanja ove promenljive, moguće je razmatrati dva različita pristupa.

Prvi pristup se zasniva na ideji da promenljiva *ms* treba da bude vidljiva u svim delovima programa gde je biblioteka uključena. Drugi pristup se zasniva na ideji enkapsulacije, odnosno sakrivanja, ove promenljive, gde će ona biti dostupna samo funkcijama ove biblioteke, čime se sprečava mogućnost ostatka programa da pristupi ili promeni vrednost sistemskog vremena.

Prilikom razvoja ove biblioteke, biće korišćen drugi pristup, gde će se promenljiva *ms* definisati tako, da bude vidljiva samo od strane biblioteke **timer0**. Ovo se postiže definisanjem promenljive u izvornoj datoteci *timer0.c*. U slučaju da je realizacija zasnovana na prvom pristupu, ona bi bila definisana unutar zaglavlja.

Nakon realizacije ovih biblioteka, hijerarhija projekta se zasniva na uključivanju biblioteka **pin** i **timer0** u glavnu datoteku, kao što je to ilustrovano na slici 5.1.



Slika 5.1: Hijerarhija projekta 1

Nakon ovih izmena, u *main* datoteci preostaju funkcije i makro konstante nastavku

u nastavku.

Naime, datoteka se sastoji od sledećih funkcija:

- pinPulsing
- pinPulse
- calculateHalfPeriod

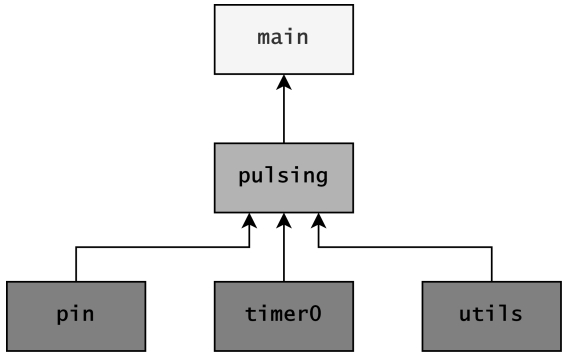
i makro konstanti:

- `DIODE_PIN`
- `FAST`
- `SLOW`
- `FAST_REPS`
- `SLOW_REPS`

Analizom preostalih funkcija, može se primetiti da funkcija *calculateHalfPeriod()* ne upravlja niti jednom hardverskom komponentom, već predstavlja pomoćnu funkciju. Prilikom razvoja složenih projekata, često se pojavljuje veliki broj ovakvih, pomoćnih, funkcija koje je potrebno na neki način organizovati. Ukoliko je njihov broj veliki, obično se organizuju u neke logički smislene, pomoćne (eng. *utility*) biblioteke. U okviru ovog projekta realizovaće se jedna pomoćna biblioteka, pod nazivom **utils**. Njen sadržaj će činiti upravo pomenuta funkcija.

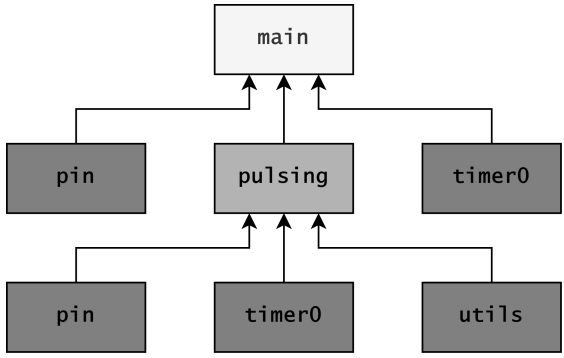
Dve funkcije koje kreiraju naizmeničnu promenu stanja na pinu, odnosno, drugim rečima, treperenje diode, su naizgled slične onima koje su smeštene u biblioteci **pin**. Međutim, za njihovu implementaciju potrebne su funkcije koje se nalaze ne samo u biblioteci **pin**, već i funkcije iz biblioteke **timer0** i **utils**. Prema tome, kako su ove funkcije složene, u prethodno navedenom smislu, biće odvojene u posebnu biblioteku. Ukoliko bi ove funkcije bile implementirane unutar biblioteke **pin**, tada bi u toj biblioteci morale biti uključene i biblioteke **timer0** i **utils**. Ovo bi kao posledicu imalo to, da ukoliko je potrebno uključiti biblioteku za upravljanje pinovima, automatski je potrebno uključiti i biblioteku za upravljanje tajmer/brojač modulom 0 i skup pomoćnih funkcija. Iz ovog razloga, praktikuje se kreiranje nove biblioteke koja će biti uključena, isključivo ukoliko postoji potreba za upotrebu navedenih funkcija. Biblioteka će biti nazvana **pulsing**, i u njoj implementirane funkcije *pinPulse()* i *pinPulsing()*. Biblioteka neće sadržati makro konstante, niti dodatne promenljive.

Nakon kreiranja sve četiri biblioteke, organizacija projekta izgleda kao na slici 5.2.



Slika 5.2: Hijerarhija projekta 2

Ukoliko se na ovaj način organizuje projekat, sve funkcije iz biblioteke **pulsing** će biti uključene u *main* datoteku, čime je moguće realizovati treperenje diode. Međutim, mora se imati na umu sam sadržaj *main* funkcije. Ova funkcija pored funkcije *pinPulsing()* sadrži i dve inicijalizacione funkcije, za pin na kojem je povezana dioda i za tajmer/brojač modul 0. Kako su ove dve funkcije implementirane u bibiliotekama **pin** i **utils** respektivno, potrebno je ove dve biblioteke takođe uključiti u *main* datoteku. Nakon pomenutih izmena, organizacija projekta izgleda kao na slici 5.3.



Slika 5.3: Hijerarhija projekta 3

U nastavku, na listingu 5.6, je dat primer koda koji je korišćen za potrebe ovog poglavlja.

```
/*
* main.c - Aplikacija koja implementira ogranicen broj
    treptanja diode
*/
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
#include <stdint.h>

// Makro za podesavanje visoke vrednosti signala na pinu
#define HIGH 1
// Makro za podesavanje niske vrednosti signala na pinu
#define LOW 0

// Makro za podesavanje izlaznog smera pina
#define OUTPUT 1
// Makro za podesavanje ulaznog smera pina
#define INPUT 0

// Makro za selektovanje porta B
#define PORT_B 0
// Makro za selektovanje porta C
#define PORT_C 1
// Makro za selektovanje porta D
#define PORT_D 2

// Makro za selektovanje pina na koji je povezana dioda
#define DIODE_PIN 5

// Makro za podesavanje periode u brzom rezimu treptanja
#define FAST 200
// Makro za podesavanje periode u sporom rezimu treptanja
#define SLOW 1000

// Makro za podesavanje broja brzih treptaja
#define FAST_REPS 15
// Makro za podesavanje broja sporih treptaja
#define SLOW_REPS 3

// Promenljiva koja skladisti broj milisekundi proteklih od
    pokretanja aplikacije
volatile uint32_t ms = 0;

/*
* pinPulsing - Funkcija koja implementira num_of_repetitions
    perioda podizanja i spustanja vrednosti na pinu
    odgovarajucom brzinom
*/
void pinPulsing(uint8_t port, uint8_t pin, uint32_t period,
    uint8_t num_of_repetitions);

/*
* pinPulse - Funkcija koja implementiran podizanje i spustanje
     vrednosti na pinu odgovarajucom brzinom
*/
void pinPulse(uint8_t port, uint8_t pin, uint32_t period);
```

```
/*
* pinSetValue - Funkcija koja postavlja vrednost na pinu
*/
void pinSetValue(uint8_t port, uint8_t pin, uint8_t value);

/*
* pinInit - Funkcija koja implementira inicijalizaciju pina
*/
void pinInit(uint8_t port, uint8_t pin, uint8_t direction);

/*
* timer0DelayMs - Funkcija koja implementira pauzu u broju
    milisekundi koji je prosledjen kao parametar
*/
void timer0DelayMs(uint32_t delay_length);

/*
* timer0Millis - Funkcija koja, na bezbedan nacin, vraca kao
    povratnu vrednost broj milisekundi proteklih od pocetka
    merenja vremena
*/
uint32_t timer0Millis();

/*
* timer0Init - Funkcija koja inicijalizuje timer 0 tako da
    pravi prekide svake milisekunde
*/
void timer0Init();

/*
* calculateHalfPeriod - Funkcija koja izracunava polovinu
    prosledjene periode
*/
uint32_t calculateHalfPeriod(uint32_t period) ;

/*
* main - funkcija koja implementira glavni deo aplikacije
*/
int16_t main()
{
    uint32_t period = 1000;     // Period jednog treptaja
    uint8_t repetitions = 5;  // Broj treptaja

    // Inicijalizacija
    pinInit(PORT_B, DIODE_PIN, OUTPUT);
    timer0Init();

    // Glavna petlja
    while (1)
```

```
    {
        // Brzo treptanje
        pinPulsing(PORT_B, DIODE_PIN, FAST, FAST_REPS);

        // Sporo treptanje
        pinPulsing(PORT_B, DIODE_PIN, SLOW, SLOW_REPS);

        // Kraj
        while (1)
        ;

    }

    return 0;
}

/*************************************************************/

void pinPulsing(uint8_t port, uint8_t pin, uint32_t period,
    uint8_t num_of_repetitions)
{
    uint8_t i;

    // Implementacija num_of_repetitions perioda
    for (i = 0; i < num_of_repetitions; i++)
        pinPulse(port, pin, period);
}

/*************************************************************/

void pinPulse(uint8_t port, uint8_t pin, uint32_t period)
{
    // Poluperioda u kojoj pin ima visoku vrednost
    pinSetValue(port, pin, HIGH);
    timer0DelayMs(calculateHalfPeriod(period));

    // Poluperioda u kojoj pin ima nisku vrednost
    pinSetValue(port, pin, LOW);
    timer0DelayMs(calculateHalfPeriod(period));
}

/*************************************************************/

void pinSetValue(uint8_t port, uint8_t pin, uint8_t value)
{
    // Postavljanje vrednosti pina
    switch(port)
    {
        case PORT_B:
            if (value == HIGH)
```

```
                PORTB |= 1 << pin;
            else
                PORTB &= ~(1 << pin);
        break;

        case PORT_C:
            if (value == HIGH)
                PORTC |= 1 << pin;
            else
                PORTC &= ~(1 << pin);
        break;

        case PORT_D:
            if (value == HIGH)
                PORTD |= 1 << pin;
            else
                PORTD &= ~(1 << pin);
        break;

        default:
        break;
    }
}

/*************************************************************/

void pinInit(uint8_t port, uint8_t pin, uint8_t direction)
{
    // Inicijalizacija smera pina
    switch (port)
    {
        case PORT_B:
            if (direction == OUTPUT)
                DDRB |= 1 << pin;
            else
                DDRB &= ~(1 << pin);
        break;

        case PORT_C:
            if (direction == OUTPUT)
                DDRC |= 1 << pin;
            else
                DDRC &= ~(1 << pin);
        break;

        case PORT_D:
            if (direction == OUTPUT)
                DDRD |= 1 << pin;
            else
                DDRD &= ~(1 << pin);
```

```
        break;

        default:
        break;
    }

}

/*************************************************************/

void timer0DelayMs(uint32_t delay_length)
{
    // trenutak pocevsi od kog se racuna pauza
    uint32_t t0 = timer0Millis();
    // implementacija pauze
    while(timer0Millis() - t0 < delay_length)
    ;
}

/*************************************************************/

uint32_t timer0Millis()
{
    uint32_t tmp;
    cli();          // Zabrana prekida
    tmp = ms;       // Ocitavanje vremena
    sei();          // Dozvola prekida
    return tmp;
}

/*************************************************************/

void timer0Init()
{
    // tajmer/brojac modul 0: CTC mod
    TCCR0A = 0x02;

    // Provera frekvencije takta prilikom kompilacije
    #if F_CPU > 20000000
    #error "Frekvencija takta mora biti manja od 20MHz!"
    #endif

    // Inicijalizacija promenljivih za preskaler i periodu
        tajmer/brojac modula 0
    uint32_t n = F_CPU / 1000;
    uint8_t clksel = 1;

    // Odredjivanje vrednosti preskalera i periode tajmer/
        brojac modula 0
    while (n > 255)
```

```
    {
        n /= 8;
        clksel++;
    }

    // tajmer/brojac modul 0: Podesavanje preskalera
    TCCR0B = clksel;
    // tajmer/brojac modul 0: Podesavanje periode
    OCR0A = (uint8_t)(n & 0xff) - 1;
    // tajmer/brojac modul 0: Dozvola prekida
    TIMSK0 = 0x02;
    // Globalna dozvola prekida
    sei();
}

/*************************************************************/

/**
* ISR - prekidna rutina tajmer/brojac modula 0 u modu CTC
*/
ISR(TIMER0_COMPA_vect)
{
    // Inkrementovanje broja milisekundi koje su prosle od
        pokretanja aplikacije
    ms++;
}

/*************************************************************/

uint32_t calculateHalfPeriod(uint32_t period)
{
    return (period/2);
}
```

Listing 5.6: Primer koda koji implementira naizmenično treptanje diode različitim brzinama

5.3 Zadaci za vežbu

Zadatak 5.3.1. U skladu sa procedurom objašnjenom u okviru ovog poglavlja, na osnovu koda sa listinga 5.6, potrebno je napisati sledeće biblioteke:

- **pin** – biblioteka za manipulaciju pinovima;
- **timer0** – biblioteka za precizno merenje vremena;
- **pulsing** – biblioteka koja implementira funkcije za rad sa ugrađenom diodom na Arduino platformi.

Nakon što je to urađeno, potrebno je, izvršavanjem datog koda na Arduino platformi, proveriti ispravnost rešenja.

Zadatak 5.3.2. Modifikovati biblioteku koja implementira funkcije za precizno merenje vremena – **timer0** na način da podržava merenje vremena na nivou *mikrosekunde*. Prototipi funkcija koje je potrebno realizovati su dati u nastavku.

- `uint32_t timer0Micros();`
  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća broj mikrosekundi proteklih od početka merenja vremena.
  - **Povratna vrednost**: Broj mikrosekundi proteklih od početka merenja vremena.
- `void timer0DelayUs(uint32_t delay_length_us);`
  - **Opis**: Funkcija koja implementira kašnjenje (pauzu) u broju mikrosekundi koji je prosleđen putem parametra `delay_length_us`.
  - **Povratna vrednost**: Nema povratnu vrednost.

Zadatak 5.3.3. Modifikovati biblioteku koja implementira funkcije za precizno merenje vremena – **timer0** na način da podržava merenje vremena na nivou sekunde, minute, sata i dana. Prototipi funkcija koje je potrebno realizovati su dati u nastavku.

- `void timer0SetTime(uint8_t day, uint8_t hour, uint8_t min, uint8_t sec, uint16_t millisec);`
  - **Opis**: Funkcija koja vrši postavljanje internog sata koji vodi računa o vremenu na nivou sekunda, minuta, sata i dana. Uzeti da je rezolucija merenja vremena jedna milisekunda.
  - **Povratna vrednost**: Nema povratnu vrednost.
- `uint32_t timer0Seconds();`
  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća broj sekundi proteklih u okviru trenutnog minuta.
  - **Povratna vrednost**: Broj sekundi proteklih u okviru trenutnog minuta.
- `uint32_t timer0Mins();`
  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća broj minuta proteklih u okviru trenutnog sata.
  - **Povratna vrednost**: Broj minuta proteklih u okviru trenutnog sata.
- `uint32_t timer0Hours();`
  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća broj sati proteklih u okviru trenutnog dana.
  - **Povratna vrednost**: Broj sati proteklih u okviru trenutnog dana.
- `uint32_t timer0Days();`
  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća broj dana proteklih od početka merenja vremena.
  - **Povratna vrednost**: Broj dana proteklih od početka merenja vremena.

Zadatak 5.3.4. Modifikovati funkciju *timer0SetTime()* iz biblioteke **timer0**, na način da interni sat podržava i rad sa mesecima i godinama. Prototip modifikovane funkcije je dat u nastavku.

- `void timer0SetTime(uint16_t years, uint8_t months, uint8_t day, uint8_t hour, uint8_t min, uint8_t sec, uint16_t millisec);`

  - **Opis**: Funkcija koja vrši postavljanje internog sata koji vodi računa o vremenu na nivou sekunda, minuta, sata, dana, meseca i godine. Uzeti da je rezolucija merenja vremena jedna milisekunda.
  - **Povratna vrednost**: Nema povratnu vrednost.

Nakon što je to odrađeno, potrebno je implementirati naredne funkcije:

- `uint8_t timer0GetDayOfWeek(uint16_t years, uint8_t months, uint8_t day);`

  - **Opis**: Funkcija koja, kao povratnu vrednost, vraća trenutni dan u nedelji. Dane u nedelji predstaviti pomoću enumerisanog tipa `days_t`, na sledeći način:

  ```
  enum week_days_t {MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
      FRIDAY, SATURDAY, SUNDAY};
  ```

  - **Povratna vrednost**: Dan u nedelji.

- `uint8_t timer0MonthLength(uint16_t years, uint8_t months);`
  - **Opis**: Funkciju koja izračunava i, kao povratnu vrednost, vraća broj dana u zadatom mesecu za zadatu godinu.
  - **Povratna vrednost**: Broj dana u datom mesecu za datu godinu.

- `uint8_t timer0IsLeapYear(uint16_t year);`
  - **Opis**: Funkciju koja proverava da li je godina, čija je vrednost prosleđena putem parametra `year`, *prestupna*.
  - **Povratna vrednost**: Vrednost 1, ukoliko je godina prestupna, odnosno 0, ukoliko nije.

Zadatak 5.3.5. Napraviti projekat unutar kog su realizovane biblioteke za rad analogno-digitalnim konvertorom i generatorom pseudoslučajnih brojeva.

Za početak, u okviru datog 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:

- `uint16_t SetBit(uint16_t reg, uint8_t bit_num);`
  - **Opis**: Funkcija vrši postavljanje bita na vrednost 1, na poziciji `bit_num`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Modifikovana vrednost promenljive `reg`.

- `uint16_t ClearBit(uint16_t reg, uint8_t bit_num);`
  - **Opis**: Funkcija vrši postavljanje bita na vrednost 0, na poziciji `bit_num`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Modifikovana vrednost promenljive `reg`.

- `uint16_t ToggleBit(uint16_t reg, uint8_t bit_num);`
  - **Opis**: Funkcija vrši postavljanje bita na suprotnu vrednost od postojeće, na poziciji `bit_num`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Modifikovana vrednost promenljive `reg`.

- `uint16_t CheckBit(uint16_t reg, uint8_t bit_num);`
  - **Opis**: Funkcija vrši proveru vrednosti bita, na poziciji `bit_num`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Vrednost bita na datoj poziciji (0 ili 1).

- `uint8_t BitmaskSet(uint8_t reg, uint8_t mask);`
  - **Opis**: Funkcija vrši postavljanje grupe bita na vrednost 1, specificiranih u promenljivoj `mask`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Modifikovana vrednost promenljive `reg`.

- `uint8_t BitmaskClear(uint8_t reg, uint8_t mask);`

  - **Opis**: Funkcija vrši postavljanje grupe bita na vrednost 0, specificiranih u promenljivoj `mask`, u okviru 16-bitne promenljive `reg`.
  - **Povratna vrednost**: Modifikovana vrednost promenljive `reg`.

- `uint16_t Not(uint16_t input);`
  - **Opis**: Funkcija vrši bitsku negaciju 16-bitne promenljive `input`.
  - **Povratna vrednost**: Rezultat operacije.

- `uint16_t And(uint16_t input1, uint16_t input2);`
  - **Opis**: Funkcija vrši bitsku i operaciju nad 16-bitnim promenljivima `input1` i `input2`.
  - **Povratna vrednost**: Rezultat operacije.

- `uint16_t Or(uint16_t input1, uint16_t input2);`
  - **Opis**: Funkcija vrši bitsku ili operaciju nad 16-bitnim promenljivima `input1` i `input2`.
  - **Povratna vrednost**: Rezultat operacije.

- `uint16_t Xor(uint16_t input1, uint16_t input2);`
  - **Opis**: Funkcija vrši bitsku eks-ili operaciju nad 16-bitnim promenljivima `input1` i `input2`.
  - **Povratna vrednost**: Rezultat operacije.

- `uint16_t ShiftLeft(uint16_t input, uint16_t num_of_shifts);`

  - **Opis**: Funkcija vrši pomeranje (eng. *shift*) ulazne promenljive `input` u levo za `num_of_shifts` mesta.
  - **Povratna vrednost**: Rezultat operacije.

- `uint16_t ShiftRight(uint16_t input, uint16_t num_of_shifts);`

  - **Opis**: Funkcija 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 komentarisati. 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 kod AVR mikrokontrolera. Kratak teorijski uvod o AD konverziji i registrima koji su neophodni za rad sa ADC modulom je dat u nastavku.

5.3.1 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 je 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 u *single conversion* režimu.

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 je data u nastavku.

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

Uloga pojedinačnih bita u okviru ovog registra je sledeća:

- **Biti 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.

Tabela 5.1: Izbor referentnog napona

REFS1	REFS0	Referentni napon
0	0	AREF, Interni Vref isključen
0	1	AVCC
1	0	Reserved
1	1	Interni 2.56V referentni napon

  U okviru ovog zadatka potrebno je koristiti napon napajanja kao referentni napon, odnosno vrednost AVCC.

- **Bit 5 – ADLAR: ADC Left Adjust Result**
  Ukoliko je postavljen na 1, vrši se levo poravnanje 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.

- **Biti 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.

Tabela 5.2: Izbor analognog ulaza

MUX[4:0]	Analogni ulaz
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5

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

ADCSRA registar

Struktura ADCSRA registra data je u nastavku.

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

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.
- **Biti 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.

Tabela 5.3: Izbor faktora skaliranja

ADPS2	ADPS1	ADPS0	Faktor skaliranja
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 odabiranje (semplovanje, eng. *sampling*) 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 brzinu, 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(uint8_t reference, uint8_t division_factor);`
  - **Opis**: Funkcija vrši inicijalizaciju AD konvertora na osnovu prosleđenih parametara (referentni napon i faktor skaliranja).
  - **Povratna vrednost**: Nema povratnu vrednost.
- `uint16_t ReadADC(uint8_t channel);`
  - **Opis**: Funkcija vrši očitavanje izlaza AD konvertora na osnovu parametra `channel` koji predstavlja broj analognog ulaznog kanala.
  - **Povratna vrednost**: Očitana vrednost sa izlaza AD konvertora.
- `void SetVref(uint8_t reference);`
  - **Opis**: Funkcija vrši postavljanje referentnog napona (u okviru ADMUX registra) na osnovu parametra reference.
  - **Povratna vrednost**: Nema povratnu vrednost.
- `void SetPrescaler(uint8_t division_factor);`
  - **Opis**: Funkcija vrši postavljanje faktora skaliranja (u okviru ADCSRA registra) na osnovu parametra `division_factor`.
  - **Povratna vrednost**: Nema povratnu vrednost.
- `void SetEnable(uint8_t enable);`
  - **Opis**: Funkcija vrši postavljanje dozvole rada AD konvertora (u okviru ADCSRA registra) na osnovu parametra `enable`.
  - **Povratna vrednost**: Nema povratnu vrednost.
- `void SetChannel(uint8_t channel);`
  - **Opis**: Funkcija 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 detaljno komentarisati.

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 prve varijante biblioteke **rand_mto**, koja se sastoji iz dve datoteke, zaglavlja *rand_mto.h* i izvorne datoteke *rand_mto.c*. Ova biblioteka implementira generator pseudoslučajnih brojeva upotrebom 16-bitnog *many-to-one* LFSR-a. Nakon toga, potrebno je preći na implementaciju druge varijante biblioteke, **rand_otm**, koja se, takođe, sastoji od dve datoteke, zaglavlja *rand_otm.h* i izvorne datoteka *rand_otm.c*. Ova varijanta implementira generator pseudoslučajnih brojeva upotrebom 16-bitnog *one-to-many* LFSR-a.

Teorijsko objašnjenje obe varijante registara je dato u nastavku.

5.3.2 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.

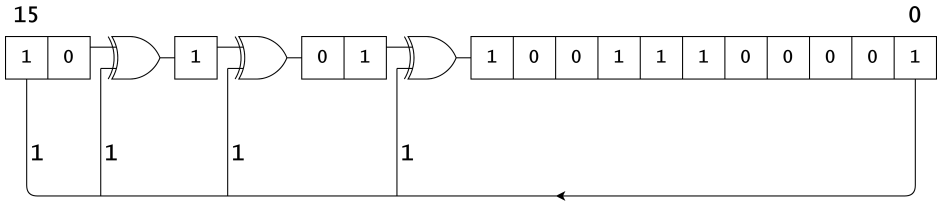
Međutim, uz dobar odabir funkcije za generisanje narednog stanja, moguće je ostvariti prilično velike periode i time generisati stanja u naizgled slučajnom redosledu. Iz tog razloga se LFSR često koriste kao generatori pseudoslučajnih brojeva u situacijama u kojima nam generisanje slučajnih brojeva visoke 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:

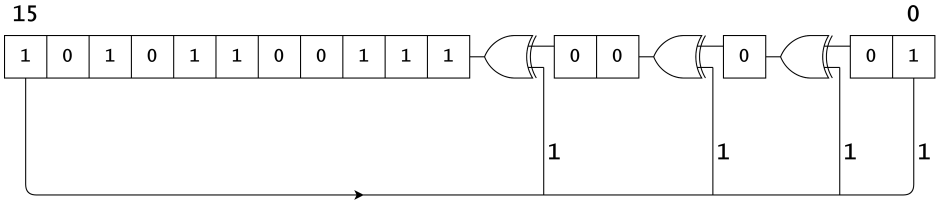
1. *One-to-many* – kod kojih se na osnovu jednog bita trenutnog stanja LFSR-a generiše više različitih bita u okviru narednog stanja LFSR-a. Primer poznatog one-to-many registra jeste LFSR u konfiguraciji Galoa.
2. *Many-to-one* – kod kojih se na osnovu više različitih bita u okviru trenutnog stanja LFSR-a generiše jedan bit narednog stanja. Primer many-to-one 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 5.4: LFSR u konfiguraciji Galoa u desnoj orijentaciji



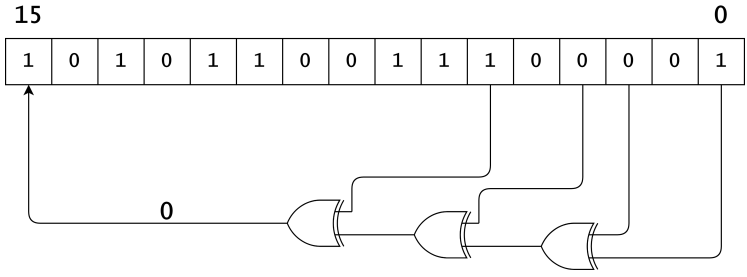
Slika 5.5: LFSR u konfiguraciji Galoa u levoj orijentaciji

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 5.6: 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(uint16_t seed);`
  - **Opis**: Funkcija vrši inicijalizaciju početnog stanja generatora na osnovu parametra `seed`.
  - **Povratna vrednost**: Nema povratnu vrednost.
- `uint16_t Rand();`
  - **Opis**: Funkcija vrši generisanje 16-bitnog pseduoslučajnog broja.
  - **Povratna vrednost**: Generisana 16-bitna vrednost.
- `uint16_t RandRange(uint16_t min, uint16_t max);`
  - **Opis**: Funkcija vrši generisanje 16-bitnog pseduoslučajnog broja iz intervala `min-max`.
  - **Povratna vrednost**: Generisana 16-bitna vrednost.
- `uint8_t GenerateNextBit(uint16_t current_state);`
  - **Opis**: Funkcija vrši generisanje naredne vrednost MSB, na osnovu trenutnog stanja registra.
  - **Povratna vrednost**: Generisana vrednost MSB.
- `uint16_t UpdateState(uint16_t state, uint8_t bit);`
  - **Opis**: Funkcija 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 komentarisati.

Zaglavlje *rand_otm.h*, na grani *branch_one_to_many*, sadrži deklaracije sledećih funkcija:

- `void InitRand(uint16_t seed);`
  - **Opis**: Funkcija vrši inicijalizaciju početnog stanja generatora na osnovu parametra `seed`.
  - **Povratna vrednost**: Nema povratnu vrednost.

- `void SetDirection(uint8_t dir);`
  - **Opis**: Funkcija vrši postavljanje orijentacije – leve ili desne (na ulazu se očekuje 0 ili 1 za desnu, odnosno levu orijentaciju, respektivno).
  - **Povratna vrednost**: Nema povratnu vrednost.

- `uint16_t Rand();`
  - **Opis**: Funkcija vrši generisanje 16-bitnog pseduoslučajnog broja.
  - **Povratna vrednost**: Generisana 16-bitna vrednost.

- `uint16_t RandRange(uint16_t min, uint16_t max);`
  - **Opis**: Funkcija vrši generisanje 16-bitnog pseduoslučajnog broja iz intervala `min-max`.
  - **Povratna vrednost**: Generisana 16-bitna vrednost.

- `uint8_t GetLSB(uint16_t state);`
  - **Opis**: Funkcija vrši izdvajanje vrednosti LSB trenutnog stanja registra.
  - **Povratna vrednost**: Izdvojena vrednost LSB.

- `uint8_t GetMSB(uint16_t state);`
  - **Opis**: Funkcija vrši izdvajanje vrednosti MSB trenutnog stanja registra.
  - **Povratna vrednost**: Izdvojena vrednost MSB.

- `uint16_t ShiftAndToggle(uint16_t state, uint8_t bit);`

  - **Opis**: Funkcija 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 detaljno komentarisati.

Konačno, u okviru *main.c* datoteke, 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, treba da iznosi 600ms (tj. poluperioda treptanja iznosi 300ms). Za realizaciju treptanja diode, potrebno je koristiti gotove funkcije iz biblioteke **pin**. Po potrebi, koristiti i gotove funkcije iz biblioteke **pulsing**. Za realizaciju kašnjenja, koristiti gotove funkcije iz biblioteke **timer0**. Nakon odgovarajućeg broja ponavljanja, potrebno je napraviti pauzu od 2s, nakon čega se postupak ponavlja.